

ARTES – A network for Real-Time research and
graduate Education in Sweden
1997 – 2006

Editor: Hans Hansson

March 5, 2006

Printed at Uppsala University,
The Department of Information Technology,
Uppsala, Sweden 2006.
Final typesetting by Samuel Åslund.

ISSN 1404-3041
ISSN 1404-3203
ISRN MDH-MRTC-197/2006-1-SE
ISBN 91-506-1859-8

To the memory of Bengt Asker

Chairman of the ARTES board 1997 – 2003

Bengt Asker was not only chairman of the ARTES board. With more than 50 years in the computer and software industry - he was one of the true Swedish pioneers in the area.

In the early 1950's, he was employed as an engineer at Saab in Linköping. As one of the users of the first computers at Saab in 1957, he became increasingly interested in computers and software. Bengt became directly involved in the development of software when Saab decided in the early 1960's to develop and market computers. He was the leader of the very successful development of Algol-Genius at Saab, and was responsible for the development of system software at Saab until 1975, when he was appointed manager of the Saab-Univac development centre in Linköping. In 1978 Bengt became manager for system and software development at Swedish Sperry. In the early 80's he spent some time in Silicon Valley, after which he returned to Sweden as development manager for the very successful computer terminal system Alfaskop at Ericsson Information Systems. He was one of the fathers of the Ericsson PC, and from 1989 until retirement he worked at Ericsson corporate headquarters. After retirement from Ericsson, Bengt Asker continued to be very active, his work including investigations for the Association of Swedish Engineering Industries and the initiation and teaching of a very popular IT-architecture course. He was also chairman of the steering committee for NUTEK's research programme on Embedded Systems, and chairman of the ARTES board. As a recognition of his contributions in this field, Bengt Asker was awarded an honorary doctorate at Mälardalen University in 2002.

Contributing authors:

Parosh Aziz Abdulla, UU	Kevin E. Moore,
Björn Andersson, CTH	University of Wisconsin
Johan Andersson, MDH	Mikael Nolin, MDH
Johan Bengtsson, UU	Thomas Nolte, MDH
Carl Bergenhem,	Christer Norström, MDH
Bergenhem Konsult	Aletta Nylén, UU
Anton Cervin, LU	Dag Nyström, MDH
Cecilia Ekelin, CTH	Zebo Peng, LiU
Johan Eker, LU	Paul Pop, LiU
Jad El-Khoury, KTH	Lars K. Rasmussen,
Petru Eles, LiU	University of South Australia
Roland Grönroos, UU	Ola Redell, KTH
Flavius Gruian, LU	Per Stenström, CTH
Erik Hagersten, UU	Håkan Sundell, CTH
Daniel Häggander, BTH	Aleksandra Tešanović, LiU
Hans Hansson, MDH	Philippas Tsigas, CTH
Jörgen Hansson, HS and LiU	Martin Törngren, KTH
Dan Henriksson, LU	Elisabeth Uhlemann, HH
Hoai Hoang, HH	Anders Wall, MDH
Jan Jonsson, CTH	Fredrik Warg, CTH
Magnus Jonsson, HH	Per-Arne Wiberg, HH
Martin Karlsson, UU	David A. Wood,
Pavel Krčál, UU	University of Wisconsin
Krzysztof Kuchcinski, LU	Wang Yi, UU
Lars Lundberg, BTH	Yi Zhang,
Pritha Mahata, UU	University of Birmingham
Sorin Manolache, LiU	Karl-Erik Årzén, LU

Abbreviations for academic sites are explained in Appendix A.

Preface

This book aims at summarizing the results of ARTES, a Swedish national research initiative which has been funded by the Swedish Foundation for Strategic Research (SSF), with a total of 95 MSEK (approx. 10 million Euros) between 1998 and 2006.

ARTES is a major research initiative that has unified and given strength to the Swedish Real-Time and Embedded Systems research community, and contributed substantially to advancing Sweden's international position in this area.

ARTES is however much more than a single research initiative. It has had a catalytic and coordinating effect for a total research effort extending far beyond the funding provided by SSF. It has created important synergies between disciplines, ensured industrial relevance in research, and facilitated important academic and industrial networking for approximately 100 senior researchers and some 200 post-graduate students.

Obviously, it is not possible, in a single volume, to give a complete presentation of the multitude of activities within ARTES and the results obtained. Instead we have decided to describe in more detail a few representative examples of the scientific results that have emerged from ARTES and complement these with an overview of ARTES activities and results in general.

ARTES is first and foremost a network of individuals with a common interest in real-time and embedded systems. Many persons have been involved in ARTES and deserve to be acknowledged for their contributions. Some are named in the following and to those whose names do not appear, I apologise and assure them of my gratitude for their efforts.

ARTES has an intimate relationship with the Swedish National Association for Real-Time Systems (SNART). It was in my capacity as the chairman of the SNART board in 1995 that I was asked by Bernt Ericson, chairman of the IT-committee at SSF, to co-ordinate the planning of a national initiative in the field of Real-Time Systems. This official request was preceded by informal contacts between Bernt Ericson and SNART board members Anders Nyman and Tony Larsson; all three being at that time employed by the Ericsson concern. SNART, as an as-

sociation in which representatives of industry and university researchers meet, has played an indispensable role for ARTES from its very beginning. Previous and current members of the SNART board are hereby acknowledged - in particular my fellow chairmen Lars Asplund, Martin Törngren, and Anton Cervin.

The work on the ARTES proposal during 1995 was a truly cooperative effort, with many meetings, discussions, the writing of various drafts, and eventually the finalizing of the proposal. Many persons actively contributed to this intense process. I would in particular like to acknowledge the contributions made by the ARTES coordination committee members (with names and affiliations in 1995 indicated): Jonas Barklund (Uppsala University; UU), Per Gunningberg (UU), Peter Lidén (Volvo), Lars Liljegren (ABB), Lennart Lindh (Mälardalen University; MDH), Jan Torin (Chalmers University of Technology; CTH), Anders Törne (Linköping University; LiU), Jan Wikander (Royal Institute of Technology in Stockholm; KTH), Wang Yi (UU), and Karl-Erik Årzén (Lund University; LU).

In parallel with the ARTES effort, a group of researchers led by Per Stenström (CTH) prepared a proposal for research into “Symmetric Multiprocessors in High Performance Real-Time Applications” (PAMP). SSF was positive to this proposal, but considered its scope to be too limited in view of the type of initiatives they were fostering at that time. Since PAMP was, in its nature, sufficiently closely related to ARTES, SSF asked ARTES to include PAMP in its programme. PAMP has been surprisingly well integrated with ARTES! I believe that the integration of PAMP and ARTES has been mutually beneficial. Many thanks to Per Stenström and the other PAMPers for a very fruitful cooperation!

Yet another extension of ARTES is due to Per Gunningberg (UU), who prompted SSF and Uppsala University to fund the targeted recruitment of Erik Hagersten; who at that time, was chief architect for Sun Microsystem’s high-end servers. As professor at Uppsala, Erik has made important contributions to the PAMP sub-programme of ARTES.

The initial funding period for ARTES was nominally from 1998 to 2002, though some activities had been initiated in 1997 and others remained in progress after 2002. With a total grant of 88 MSEK during this period, ARTES funded the work of more than 40 graduate students, in addition to numerous other activities. When an opportunity to extend graduate school activities was offered by SSF in 2003, the ARTES board gave Karl-Erik Årzén and me the task of preparing an application for financial support. We were able to submit an application accepted by SSF which awarded an additional 7 MSEK to be spent on the support of newly recruited graduate students during 2004-2006.

Over the years, the development of ARTES has been competently guided by a board with a majority of members representing Swedish

industry. The board members have generously shared their competence in formulating and reviewing plans, evaluating project and course proposals etc. I am very grateful for the support provided by all board members, and specifically the chairmen, the late Bengt Asker (1997-2003) and Jakob Axelsson (2004-). Bengt Asker deserves extra recognition, not only for being chairman of the ARTES board for many years, but also for being one of the most frequent and active participant in ARTES events, thereby sharing his wisdom and experiences with the entire ARTES network.

Without naming any particular individual I would additionally like to thank the many Swedish and international scientists and other experts who have contributed to ARTES by evaluating proposals, presenting their views and sharing their expertise at the annual ARTES summer school and other meetings, hosting visitors etc.

Of equal - or even greater - importance has been the dedication of the industrial partners in ARTES. For each research project supported, there has been industrial involvement in some form, ranging from major involvement in the project and monetary support to a reference group providing feedback on project plans and progress. This industrial involvement has been instrumental in ensuring relevance in the research and adoption of research results, as well as in providing excellent employment opportunities for graduated students.

Obviously, the importance to ARTES of its funding agency cannot be overestimated. The entire ARTES network is grateful for the confidence shown us by the Swedish Foundation for Strategic Research (SSF). I would in particular like to thank Olof Lindgren at SSF for support and enlightening interaction at all times.

It is not possible to coordinate, singlehanded, such a major research programme as ARTES. Support is required in the management of all the practical matters and ensuring that the job is actually done. ARTES has been extremely fortunate in having Roland Grönroos as research coordinator. Roland has been truly dedicated to this task and taken full responsibility for the daily operation of ARTES, and has become an important person in driving ARTES forward, taking appropriate action when needed. Without Roland, ARTES would not have been the same. Thank you for a very enjoyable cooperation!

In recent years, Paul Pettersson has been given increasing responsibility for the management of ARTES, first as Director of Studies, and since 2005 as Programme Director. I am really grateful for the way Paul is running the show; not to mention that he made it very easy for me to step down when I finally realized that it was not practical for ARTES (or for me) that I remained in charge.

Finally, I would like to thank the co-authors of this volume for their efforts and patience. We have on and off been working on this project

since 2003. It is with great pleasure that we now complete the book. I would in particular like to thank Roland Grönroos for support and for compiling the presentation of ARTES and the section coordinators Zebo Peng, Wang Yi, Magnus Jonsson, Per Stenström, Jan Jonsson, and Karl-Erik Årzén. In the final editing, competent and much appreciated support was provided by Samuel Åslund.

Many subjects and problems have been addressed within ARTES, as reported in this book. Despite much progress in the field, (resulting from the work of ARTES and the international research community in general), many important challenges remain. These motivate a continued strong research effort. Possibly even more important is that the results of much research, with direct or indirect commercial potential, have not yet made it all the way to industrial exploitation. Continued basic and applied research, as well as the industrial validation of results, their promotion and deployment are required to ensure a full return from the investment in ARTES!

Västerås in March 2006

Hans Hansson
ARTES Programme Director 1997-2004

Contents

1	Introduction	19
1.1	Application areas	20
1.2	Real-time systems research	21
1.3	Trends	24
1.4	Outline	25
	Bibliography	26
2	ARTES – Facts and figures	27
2.1	The creation of ARTES	27
2.2	The programme plan	30
2.3	Participating research groups	31
2.4	Graduate students	33
2.5	Results	37
I	Design Techniques for Embedded RT Systems	43
3	Introduction	45
4	Design Optimization of Multi-Cluster Embedded Systems for Real-Time Applications	49
4.1	Introduction	50
4.2	Heterogeneous real-time embedded systems	52
4.3	Schedulability analysis	57
4.4	Design optimisation	60
4.5	Multi-cluster systems	64
4.6	Multi-cluster optimisation	71
4.7	Multi-cluster analysis and scheduling	76
4.8	Frame-packing optimisation strategy	86
4.9	Experimental results	91
4.10	Conclusions	94
	Bibliography	95

5	Using DVS Processors to Achieve Energy Efficiency in Hard Real-Time Systems	103
5.1	Introduction	103
5.2	Hardware support	104
5.3	Stochastic scheduling	105
5.4	Finding the maximum required speeds	109
5.5	Runtime slack management	111
5.6	Summary and conclusions	118
	Bibliography	120
6	Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times	123
6.1	Introduction	123
6.2	Related work	125
6.3	Notation and problem formulation	127
6.4	Analysis algorithm	130
6.5	Experimental results	145
6.6	Limitations and extensions	153
6.7	Conclusions	155
	Bibliography	156
7	Wait/Lock-Free Applications	161
7.1	Introduction	163
7.2	Non-blocking synchronization	164
7.3	Shared memory	165
7.4	Shared concurrent data structures	170
7.5	Adapting lock-free to hard real-time systems	173
7.6	NOBLE – a software library	174
7.7	Conclusions	175
	Bibliography	176
8	Reconfigurable Embedded Real-Time Systems: A Story of COMET	181
8.1	Introduction	182
8.2	Components and aspects	183
8.3	ACCORD design method	185
8.4	Real-time component model (RTCOM)	188
8.5	COMET: a component-based embedded RT database	194
8.6	Related work	200
8.7	Summary	201
	Bibliography	203

<i>CONTENTS</i>	13
II Modeling and Verification	209
9 Introduction	211
9.1 Paper 1: Modeling and verification with timed automata .	211
9.2 Paper 2: Schedulability analysis using timed automata . .	212
9.3 Paper 3: Undecidability of linear time temporal logic for timed petri nets	212
9.4 Paper 4: A framework for analysis of timing and resource utilization targeting complex embedded systems	213
10 Modeling and Verification of Real-Time Systems Using Timed Automata	215
10.1 Introduction	215
10.2 Timed automata	216
10.3 Symbolic semantics and verification	221
10.4 DBM: Algorithms and data structures	230
10.5 UPPAAL	245
Bibliography	253
11 Decidable and Undecidable Problems in Schedulability Analysis Using Timed Automata	259
11.1 Introduction	260
11.2 Preliminaries	262
11.3 Decidable problems: A summary	265
11.4 Undecidability	272
11.5 Variants of the problem	277
11.6 Conclusions	279
Bibliography	280
12 Undecidability of LTL for Timed Petri Nets	283
12.1 Introduction	283
12.2 Timed petri nets	285
12.3 Lossy counter machines	287
12.4 Undecidability of LTL	288
12.5 Discrete-timed petri nets	294
Bibliography	294
13 A Framework for Analysis of Timing and Resource Utilization targeting Complex Embedded Systems	297
13.1 Introduction	298
13.2 Related work	299
13.3 Concepts of the ART framework	302
13.4 ART-ML and PPL languages	308
13.5 Validation of models	316
13.6 The tools	320

13.7 An industrial case study	323
13.8 Conclusions and future work	327
Bibliography	327
III Real-Time Communication	331
14 Introduction	333
14.1 Introduction	333
14.2 Application and traffic characteristics	334
14.3 Real-time communication services	336
14.4 Local area networks	337
14.5 Fieldbus networks	339
14.6 Packet-switched networks	340
14.7 Internet	343
14.8 Networks for high-performance computing systems	343
14.9 Fiber-optic networks	344
14.10 Wireless networks	345
14.11 Introduction to included papers	345
Bibliography	346
15 Real-Time Server-Based Communication for CAN	353
15.1 Introduction	353
15.2 The Controller Area Network (CAN)	355
15.3 Server-based scheduling	356
15.4 Server-based scheduling of CAN	359
15.5 Summary and conclusions	366
Bibliography	367
16 A pipelined fiber-ribbon ring network with heterogeneous real-time support	373
16.1 Introduction	374
16.2 The CCR-EDF network architecture	376
16.3 The CCR-EDF medium access protocol	377
16.4 A radar signal processing case	380
16.5 A large IP router case	385
16.6 Case definition and simulator setup	386
16.7 Simulations	388
16.8 Discussion on throughput ceiling	391
16.9 Conclusions	394
Acknowledgement	394
Bibliography	394

17 Wireless Real-Time Communication Using Deadline Dependent Coding	397
17.1 Introduction	398
17.2 Wireless communication	401
17.3 Probabilistic view	403
17.4 Deadline dependent coding	404
17.5 Conclusions	412
Bibliography	413
18 Real-Time Communication for Industrial Embedded Systems Using Switched Ethernet	417
18.1 Network architecture	418
18.2 Real-time traffic handling	418
18.3 Deadline scheduling and feasibility test	420
18.4 Deadline partitioning schemes	423
18.5 Conclusions	425
Bibliography	427
IV Multiprocessor Real-Time Systems	429
19 Introduction	431
19.1 Background	431
19.2 Multiprocessors: Technology overview	433
19.3 Articles in this chapter	438
Bibliography	438
20 Limits on Speculative Module-level Parallelism in Imperative and Object-oriented Programs on CMP Platforms	441
20.1 Introduction	442
20.2 Execution and architectural models	444
20.3 Methodology and benchmarks	447
20.4 Experimental results	451
20.5 Related work	458
20.6 Conclusions	459
Acknowledgments	460
Bibliography	461
21 Trade-offs and conflicts between performance and maintainability in large multiprocessor based RT Systems	465
21.1 Introduction	465
21.2 Industrial cases	467
21.3 Studies and experiences	469
21.4 Results	472

21.5 Discussion	473
21.6 Conclusions	474
Bibliography	475
22 Memory System Behavior of Java-Based Middleware	477
22.1 Introduction	478
22.2 Background	479
22.3 Methodology	484
22.4 Scaling results	486
22.5 Cache performance	497
22.6 Related work	500
22.7 Conclusions	501
Acknowledgements	502
Bibliography	502
V Real-Time Scheduling	505
23 Introduction	507
23.1 Scheduling preliminaries	508
23.2 Time-table-based scheduling	508
23.3 Priority-based scheduling	510
23.4 Included papers	511
24 Average-case performance of static-priority scheduling on multiprocessors	513
24.1 Introduction	513
24.2 Background	515
24.3 Average-Case Behavior	519
24.4 Architectural Impact	524
24.5 Discussion	531
24.6 Conclusions	532
Bibliography	532
25 Schedulability-Driven Communication Synthesis for Time Triggered Embedded Systems	537
25.1 Introduction	537
25.2 System Architecture	540
25.3 Problem Formulation	544
25.4 Schedulability Analysis	544
25.5 Optimization Strategy	554
25.6 Experimental Results	561
25.7 Conclusions	566
Bibliography	566

26	Generating Real-Time Schedules using Constraint Programming	571
26.1	Introduction	571
26.2	Related work	572
26.3	Constraint programming	573
26.4	Problem description	574
26.5	Real-time scheduling using constraint programming	575
26.6	Experimental evaluation	579
26.7	Summary and future work	585
	Bibliography	586
27	Static-priority scheduling on multiprocessors	589
27.1	Introduction	589
27.2	Introduction to periodic scheduling	594
27.3	Global scheduling	604
27.4	Bound on utilization bounds	615
27.5	Partitioned scheduling	616
27.6	Anomalies	620
27.7	Introduction to aperiodic scheduling	634
27.8	Global scheduling	636
27.9	Partitioned scheduling	645
27.10	Conclusions	654
	Bibliography	656
VI	Real-Time and Control	661
28	Introduction	663
28.1	Control implementation	663
28.2	Co-design tools	665
28.3	Control of real-time computing systems	665
28.4	Control in ARTES	666
28.5	Article overview	667
29	Tools for Real-Time Control System Co-Design	669
29.1	Introduction	669
29.2	AIDA	672
29.3	Jitterbug	677
29.4	TrueTime	682
29.5	XILO	691
	Bibliography	700

30 The Control Server Model for Codesign of Real-Time Control Systems	705
30.1 Introduction	706
30.2 The model	709
30.3 Control and scheduling codesign	712
30.4 Control server tasks as real-time components	717
30.5 Implementation	719
30.6 Control experiments	722
30.7 Conclusion and discussion of future work	726
Appendix: Cost calculation in Example 1	728
Bibliography	730
A Projects and Courses	733
B Theses by ARTES Real-Time Graduate Students	745
B.1 Students funded by ARTES	745
B.2 Abstracts of theses of students funded by ARTES	749
B.3 Theses of students without project support from ARTES	782
C Research groups	789
C.1 Luleå University of Technology Department of Computer Science and Electrical Engineering.	789
C.2 Mid Sweden University, Department of Information Technology and Media, in Sundsvall.	790
C.3 Uppsala University, Department of Information Technology.	790
C.4 Mälardalen University, in Västerås, Dept. of Computer Science and Electronics	792
C.5 Swedish Institute of Computer Science in Kista.	793
C.6 Royal Institute of Technology, Stockholm	794
C.7 Linköping University, Department of Computer and Information Science,	794
C.8 University of Skövde, School of Humanities and Informatics.	795
C.9 Chalmers University of Technology, The Department of Computer Science and Engineering.	796
C.10 Halmstad University, School of Information Science, Computer and Electrical engineering(IDE)	798
C.11 Blekinge Institute of Technology, School of Engineering .	798
C.12 Lund University	799
D Programme plan	801

Chapter 1

Introduction

By **Hans Hansson**

Department of Computer Science and Electronics

Mälardalen University

Email: `hans.hansson@mdh.se`

In many applications, computer systems sense their environment and directly influence it through actions. Such systems are subject to the real-time constraints of the environments in which they operate. For example, an autonomous vehicle needs a control system that responds quickly enough to avoid collisions. This requirement for timely behaviour is the distinguishing characteristic for real-time systems. Real-time systems must not only choose appropriate actions but also choose actions at appropriate times.

Research in real-time systems addresses precisely this issue by developing methods for guaranteeing timely reaction. Real-time computing is not about building “fast” systems; it is about building systems predictably “fast enough” to act on their environments in well specified ways.

On a slightly more detailed level real-time requirements typically express that an interaction should occur within a specified timing bound. It should be noted that this is quite different from requiring the interaction to be as fast as possible.

Essentially all real-time systems are embedded in products, and the vast majority of embedded computer systems are real-time systems. Real-Time Systems is the dominating application of computer technology, as more than 99% of the manufactured processors (more than 8 billion in 2000 [Hal00]) are used in embedded systems.

Real-Time and Embedded Systems Technology is a central concern for an increasing fraction of the industry. In fact, more than 50% of the value of Swedish export in 2005 was due to products for which embedded

computer systems and controlling software are essential, and for a large part of the remaining export the use of embedded systems products are essential for competitiveness.

1.1 Application areas

Real-time systems are playing a key role in many sectors of industry. For instance, distributed real-time control systems is now replacing and enhancing many of the conventional control systems in automobiles, making them more efficient and improving public safety. Real-time and embedded systems technology are essential, and in many cases of increasing importance, also in many other branches of industry, as illustrated by the following:

Process Industry is critically relying on real-time systems. The control systems used in process industry are good examples of large, distributed, hierarchical real-time systems where issues such as dependability, timeliness, hardware and software support are very important. Distributed control systems is an area where Sweden has strong tradition, for example through ABB Automation.

Manufacturing automation systems share many of the characteristics of process industry. The programmable Logic Control (PLC) systems used are also distributed real-time systems with the same requirements on dependability and timeliness. Manufacturing systems are also characterised by the use of embedded systems for, e.g., mechatronic applications. An example is industrial robots. Sweden has a world leading position in the development of industrial robots through ABB Robotics, and several small and medium-sized companies developing PLC systems.

Transportation systems include control systems embedded in automobiles, aircrafts, trains, ships, spacecrafts etc., as well as various transportation management and control systems, e.g. systems for fleet management, air-traffic and train control. Sweden has a long tradition in this area, with companies such as Bombardier Transportation, Kockums, Saab, Scania, and Volvo. Transportation system typically have high demands on dependability and predictability. Distributed computerised control is used or being introduced in virtually all transportation systems.

Telecom systems provide human-to-human communication. With the digitalisation and introduction of wired and wireless broadband transmission techniques, new applications with high real-time demands are introduced, e.g. multi-media, distributed interactive

design environments, and real-time data communication. Also, the telecommunication system itself is a complex distributed real-time system with high demands on availability and robustness. Ericsson is one of the largest Telecom system provider in the world, and has a market leading position in mobile telecommunication systems.

Defense systems, many of them very advanced, have for many years played an important role in Swedish industry, with companies such as Bofors, Kockums, CelciusTech, and Saab. Many of these systems include embedded control with ultra-high requirements on robustness, dependability, and predictability.

Medical real-time systems include patient monitoring systems, pacemakers, infusion pumps, and other systems for diagnosis and treatment. The majority of these systems are highly safety-critical since they under computerised control operate in close contact with (or even inside) human bodies.

Power generation & distribution share all the characteristics of the process industry. The control and supervisory systems used are essentially similar to the distributed control systems used in process industry. Power distribution networks and their control systems are examples of large real-time systems that are geographically distributed and have very demanding time constraints. Power Generation and Distribution is another of Sweden's strong industrial branches through ABB and the utilities Vattenfall and E.ON.

Common for essentially all branches of industry is the need for competence in developing and integrating predictable and dependable products, based on a mix of newly developed parts and existing legacy components. Meeting this and other challenges are required for Sweden to remain competitive. It has been the mission of ARTES to provide the required competence, both directly (by research and graduate education) and indirectly (by having a catalytic effect on the entire education system, as well as on participating industries).

1.2 Real-time systems research

Real-time systems is a multidisciplinary research area, in that it (at least) includes aspects of Automatic Control, Computer Science, Computer Engineering and Electrical Engineering.

Guided by the twofold vision

to transfer knowledge and competence to Swedish industry that will allow it to first utilise the latest achievements in real-time systems design

and

to reduce lead times for designing and modifying real-time systems by an order of magnitude by year 2005

research related to the following research areas and topics has been performed within ARTES.

Computer Implementations of Control Systems includes computer system aspects of the design and use of sensors, control algorithms and actuators.

Dependability includes fault detection, software and hardware fault tolerance, as well as dependability validation.

Design of Embedded Systems deals with design and methods for design of systems with embedded control. These are typically machines and other basically mechanical systems and devices.

Distributed Systems includes design and methods for design of cooperating systems that are both geographically and logically distributed.

Formal Methods refers to the use of mathematical techniques in the design and analysis of computer hardware and software. Typically, a formal method allows properties to be predicted from a mathematical model of the system.

Hardware Support deals with design and methods for design of hardware components (often dedicated) that support the execution of real-time applications.

Programming Languages refers to design and methods for design of application specific and general languages for the programming of real-time systems.

Real-Time Communication deals with technology for and analysis of data-communication solutions that provide some level of timing guarantee for the delivery of data.

Real-Time Databases deals with design and methods for design of databases required to provide their responses in a timely manner.

Resource Handling mainly deals with methods for predictable sharing of scarce resources, including allocation and scheduling issues.

Safety Critical Systems refers to design and methods for design of systems where the failure to meet time constraints can lead to accidents.

Software Engineering covers not only the technical aspects of building software systems, but also issues related to the processes used and people involved.

Systems Engineering is the application of engineering to solutions of a complete problem in its full environment by systematic assembly and matching of parts over the entire lifecycle of the system.

In addition, within the sub-programme PAMP, research on methods for using symmetric multi-processors in high-performance real-time applications has been conducted.

In most cases, the research projects have been covering several research areas. In addition, each project has also typically been related to one or (in some cases) several application areas. Though it may be a bit bold to claim that we have reached the above vision, substantial progress, important contributions, and industrial deployment of research results have been achieved.

This book presents some of the research results of ARTES (and PAMP). The material presented are representative examples, but do not cover all type of research within ARTES. To get a more complete overview, it may be a good idea to have a look at the theses that have been presented by ARTES graduate students, which are listed in Appendix B. Alternatively, the ARTES web www.artes.uu.se contains a wealth of information.

The research being presented in this book is not chosen at random. It represents the main directions of research within ARTES. The presentation is structured into six parts, covering the following important topics:

- **Design Techniques for Embedded Real-Time Systems**, which deals with development of real-time systems, taking the complexity and trade-off between hardware, software, communication, database components, etc. into account.
- **Modelling and Verification**, which deals with formal modelling and verification of real-time systems, including theoretical studies, techniques, software tools and industrial applications.
- **Real-Time Communication** covers real-time applications and their communication requirements, communication patterns and

traffic characteristics, as well as examples of specific real-time communication solutions.

- **Multiprocessors in Real-Time System**, deals with the use of multiprocessors in real-time systems, including how to develop software for them, and how to efficiently and predictably execute them.
- **Real-Time Scheduling** focuses on the basic algorithms used for the planning and scheduling of real-time applications, including underlying principles and techniques for schedule generation and analysis.
- **Real-Time and Control** deals with implementation of control systems, including simultaneously taking optimization of control and real-time into account, and the use of feedback-control in execution of real-time embedded systems.

1.3 Trends

Taking a look forward, it is clear that there is a continued rapid development in information and communication technology, and that we at a steady pace are moving into to the era of pervasive and ubiquitous computing. Products and production systems are becoming more and more dependent on real-time and embedded computer systems. There are several important trends behind this development:

- Rapid technology development; microelectronics is becoming smaller, cheaper, faster, more power efficient, and the use of both wired and wireless connectivity is increasing.
- The integration of products into larger systems and the integration of different types of systems, such as business and technical systems are both increasing, as is the number and complexity of functions in embedded computer systems.
- The balance-point between what is economically feasible and technically possible is constantly shifting, leading to replacement of specialised solutions by standardized programmable embedded computer systems, as well as
- the introduction of embedded computer systems technology in new areas; as a result, developers tend to be application domain experts rather than experts in computer technology.

The trends and rapid development put very strong pressure on improvements of processes and technologies, which in turn put demands

on research results. Hence, a continued strong research will be in high demand also in the foreseeable future.

1.4 Outline

In addition to this introductory chapter, the book consists of a chapter providing an overview of ARTES, followed by six parts. These parts are each focusing on a specific important real-time systems topic - structured into an initial chapter, providing overview and introduction to the topic, followed by a number of chapters presenting more specific technical contributions. The specific topics covered in this book are:

1. **Design Techniques for Embedded Real-Time Systems.** This part is co-ordinated by Prof. Zebo Peng, Linköping University, and contains five technical contributions on analysis and optimization techniques for heterogeneous real-time systems, energy-aware real-time scheduling, analysis of systems with stochastic execution times, wait/lock-free synchronization for real-time systems, and component-based design using aspect oriented programming.
2. **Modelling and Verification.** This part is co-ordinated by Prof. Wang Yi, Uppsala University, and contains four technical contributions on modeling and verification with timed automata, schedulability analysis using timed automata, undecidability of linear-time temporal logic for timed Petri-nets, and a framework for modeling and analysis of timing aspects of large legacy software systems.
3. **Real-Time Communication.** This part is co-ordinated by Prof. Magnus Jonsson, Halmstad University, and contains four technical contributions on using server-based techniques for scheduling of messages on the CAN-bus, a fiber-optic ring network for heterogeneous real-time communication, deadline dependent coding for wireless real-time communication, and switched Ethernet for real-time communication in industrial embedded systems.
4. **Multiprocessors in Real-Time System.** This part is co-ordinated by Prof. Per Stenström, Chalmers University of Technology, and contains three technical contributions on techniques for transparently extracting parallelism from sequential programs, how to deal with the conflict between software maintainability and performance, and the impact of the memory system on the performance of Java-based multiprocessor workloads.
5. **Real-Time Scheduling.** This part is co-ordinated by Dr. Jan Jonsson, Chalmers University of Technology, and contains four

technical contributions on average-case performance of static-priority multiprocessor scheduling, efficient generation of schedules for interprocess communication in time-triggered systems, using constraint techniques to generate real-time schedules, and finding good priority-assignment schemes for multiprocessor real-time systems.

6. **Real-Time and Control.** This part is co-ordinated by Prof. Karl-Erik Årzén, Lund University, and contains two technical contributions, the first gives an overview of four co-design tools that have been developed within ARTES, and the second present a real-time scheduling mechanism tailored to control and signal processing applications.

Bibliography

[Hal00] Tom R. Halfhill. Embedded markets breaks new ground, 2000.

Chapter 2

ARTES – Facts and figures

By **Roland Grönroos**[†] and **Hans Hansson**[‡]

[†]Department of Information Technology
Uppsala University
Email: `Roland.Gronroos@it.uu.se`

[‡]Department of Computer Science and Electronics
Mälardalen University
Email: `hans.hansson@mdh.se`

2.1 The creation of ARTES

The Swedish Foundation for Strategic Research (SSF) asked in February 1995 the Swedish National Real-Time Association (SNART) to coordinate the planning of a Swedish Network for Real-Time Systems Research. A planning grant of 150 000 SEK was provided by SSF. As chairman of SNART, Hans Hansson invited 28 universities to participate in the planning. A planning conference was held in Gothenburg in conjunction with the 5:th SNART meeting. The application was submitted in October 1995 with Hans Hansson as editor.

In the positive evaluation of the proposal by SSF in September 1996 it was pointed out that *”ARTES will boost leading groups in real time and strengthen their links to industry”*. The evaluator team also stated *”Additionally it is important to us to single out the value of the leadership as provided by Dr Hans Hansson.”*

The ARTES national network officially started in January 1998.

2.1.1 Funding

The agreement to start ARTES activities was signed June 10, 1997. The funding was initially 5 MSEK for 1998. During this year a programme plan was to be produced as a basis for further funding decisions. In the programme agreement signed in December 1998 the funding total funding for ARTES was 68.9 MSEK. In addition, the programme PAMP (Symmetric Multiprocessors in High-Performance Real-Time Applications) and a professorship for Erik Hagersten at Uppsala University were included with separate funding. In June 1999, SSF decided to increase the amount to 88 MSEK in total for the period until 2002-12-31. Finally, as the result of a separate application (ARTES++) for extended funding, SSF decided in August 2003 to support ARTES with an additional 7 MSEK, to prolong the research school activities until December 2006.

Table 2.1 shows the total funding and Figure 2.1 the funding over the years.

	Planned	Result
Funding	81	95
Other income		1.4
SUM	81	96.4
Costs		
Research projects	54.2	68.4
Graduate school	7.1	6.2
Mobility	3.5	3
Infrastructure	3.3	2.5
Administration	5.9	3.2
VAT (8%)	6.5	7.6
Allocated funding		2.7
Remaining 05-12-31		2.8
SUM	80.5	96.4

Table 2.1: ARTES budget for the SSF funding and the result 1997-2005.

The research activity peak in ARTES occurred in 2001-2002, when ARTES supported projects at a level of 1.5 MSEK per month; corresponding to one PhD carried out every 6 weeks. This corresponded to about 40% of the total post-graduate Real-Time studies in Sweden. Adding the importance of the networking and mobility activities, it is definitely fair to say that ARTES was the main driving force for the real-time research in Sweden.

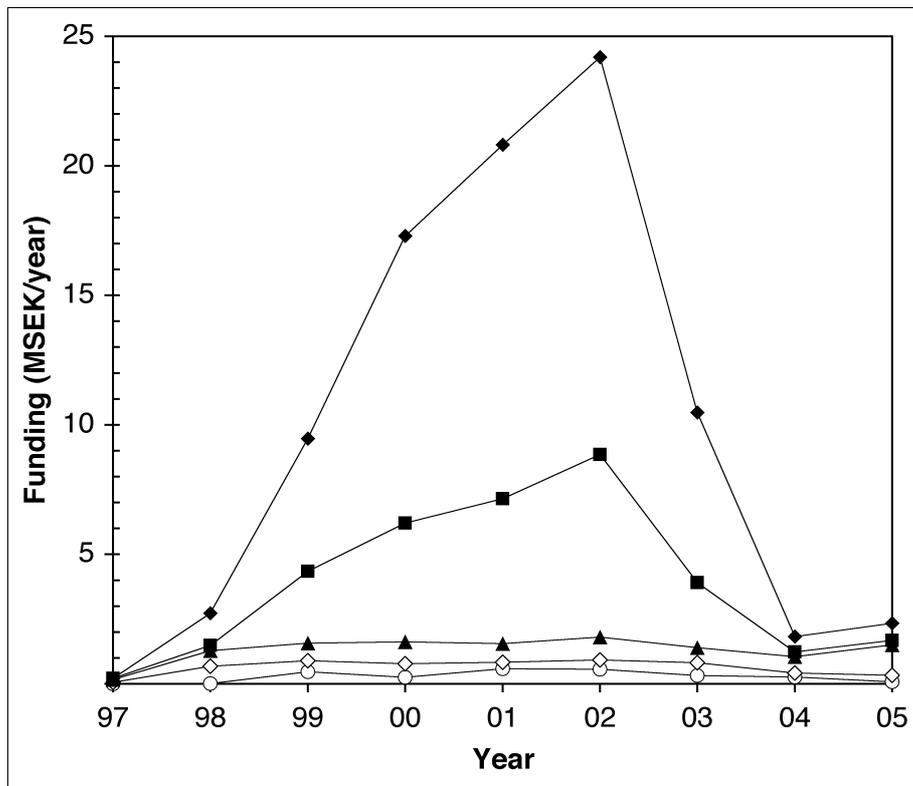


Figure 2.1: ARTES use of funding per year 1997-2005 as accumulated costs within each year. Research (◆). University fee (■). Courses (▲). Administration (◇). Information (○). Note: Research and University fee approximately equals the research project funding given in Table 2.1. VAT is not included.

2.1.2 Industrial participation

To ensure industrial relevance, ARTES required that at least one company expressed their support for each project. There was however no formal requirement for industrial participation, we have nevertheless asked the projects to report industrial participation and contributions in the project reports. The general picture is that for most projects only small amounts were reported, but in a few cases there were substantial industrial contributions; in particular for projects 10, 26, 29, 30 and 34 in Appendix A.1.

2.2 The programme plan

In the programme plan for ARTES (provided in Appendix D) submitted to SSF in 1998 the following twofold vision was formulated to guide ARTES:

For the network:

To transfer knowledge and competence to Swedish industry that will allow it to first utilise the latest achievements in real time systems design.

Specifically for the research projects:

To reduce lead times for designing and modifying real time systems by an order of magnitude by year 2005.

It was additionally stated that "Component based design together with a network for intense interactions between academic and industrial groups will be instrumental in fulfilling the above vision".

ARTES goals were

- to increase the number of PhDs and Licentiates, which play important roles in development of industrial real-time applications and products
- to increase the efficiency of graduate education,
- active industrial involvement in research and graduate education, as well as academic involvement in industry,
- to maximise synergy between the real-time components in strategic centres supported by SSF, as well as with other efforts in the area,
- to increase national and international cooperation in real-time systems research and education, and
- to provide a broad base for Swedish real-time systems research, and to make Swedish real-time systems research world leading in selected areas.

To reach these goals ARTES had four activities on its programme:

- **Research Projects** involving cooperation between industrial and academic network nodes. ARTES (including PAMP) planned to finance the graduate studies for 43 students, of which 36 were expected to complete a PhD by 2005.

- **The ARTES graduate school**, which provided graduate courses given in a format facilitating participation of students from universities nationwide. A special effort was the annual ARTES summer school, which was both a traditional academic summer school with tutorials, and a meeting place for academic and industrial participants, where projects, other cooperations and general issues were discussed.
- **A Mobility Programme** to increase interaction between industrial and academic network nodes, as well as with internationally leading research groups.
- **The Infrastructure Support** provided information and established various types of cooperations involving ARTES nodes, such as workshops and international cooperation.

2.3 Participating research groups

The geographical distribution of the ARTES academic nodes is shown in Figure 2.2 and the groups are listed below. A more detailed presentation of the participating research groups can be found in Appendix C. A list of ARTES industry nodes is shown in Table 2.2.

The geographical distribution of the ARTES academic nodes is shown in Figure 2.2. The following is a list of the participating research organisations and research groups actively involved in ARTES. Additional details are provided in Appendix C. A list of ARTES industry nodes is shown in Table 2.2.

Luleå University of Technology, Department of Computer Science and Electrical Engineering.
Embedded Internet System Laboratory (EISLAB).

Mid Sweden University, Department of Information Technology and Media, in Sundsvall.
Multimedia Communication Systems Lab.
The Electronics Design Division.

Uppsala University, Department of Information Technology.
Modeling and Analysis of Real Time Systems group.
Communications Research group.
Algorithmic Program Verification group.
Uppsala Architecture Research Team.

Mälardalen University, in Västerås, Dept. of Computer Science and Electronics.

- Programming language group
- Embedded Systems Software Engineering group
- Industrial Software Engineering group
- Monitoring and testing group
- Real-Time Systems Design Group
- Scalable Architecture for Real-time Applications group
- Safety-Critical Systems group
- Sensors and Biomedical Engineering group

Swedish Institute of Computer Science in Kista. Computer and Network Architectures Laboratory.

Royal Institute of Technology, Stockholm,
Department of Machine Design, Division of Mechatronics, Embedded Control Systems research group.
Department of Microelectronics and Information Technology, The Department of Electronic. System Architecture and Methodology group

Linköping University, Department of Computer and Information Science,
Real-Time Systems Laborator (RTSLAB).
Embedded Systems Laboratory (ESLAB)

University of Skövde, School of Humanities and Informatics.
The Distributed Real-Time Systems Research Group.

Chalmers University of Technology, The Department of Computer Science and Engineering.
The Division of Computing Science
Distributed Computing and Systems Research Group
The Division of Computer Engineering
High-Performance Computer Architecture Group
Computer Graphics Research Group
The Fault-tolerant Computing for Embedded Applications, The FORCE group.

Halmstad University, School of Information Science, Computer and Electrical engineering
Computing and Communication lab.

Blekinge Institute of Technology, School of Engineering
Parallel Architectures and Applications for Real-Time Systems (PAARTS) group.

Lund University, The Department of Computer Science, Embedded Systems Design Laboratory (ESDlab).

The Department of Automatic Control, Control and Real-Time Computing group.

The Department of Computer Science, Software Development Environments group

2.4 Graduate students

ARTES graduate students include three categories.

1. ARTES Real-Time Graduate Students, which are students that (essentially) are fully funded by ARTES, i.e., with support for research projects. This form was active 1998-2003.
2. Real-Time Graduate Students, which are students without funding for their research from ARTES, but with support to participate in the mobility programme 1998-2003 and in ARTES events free of charge (1997-present).
3. ARTES++ Real-Time Graduate Students, which are ARTES Real-Time Graduate Student that have applied for and obtained a grant from ARTES for course and mobility activities. This form was introduced in 2004.

A Real-time graduate student is a PhD or licentiate¹ student at a Swedish university which has applied to and been accepted by ARTES. The thesis subject of a Real-time graduate student is typically computer science, computer engineering, computer systems, industrial control systems, mechatronics, or automatic control and the thesis topic has been assessed by ARTES to have a strong connection to the real-time area. It is also expected that a Real-time graduate students shall be well motivated to work in cooperation with industry during the education and have an ambition to work in industry after the education.

A Real-time graduate student have the possibility to be financially supported by ARTES for travel and other purposes. He or she also has priority admittance to ARTES-courses and other common activities within ARTES. Thirdly, by being a Real-time graduate student, special supervision support can be arranged in an interdisciplinary way using the ARTES network. The ARTES network also gives excellent opportunities for personal contacts with representatives from industry, possibly leading to concrete cooperation's and employment after graduation.

¹Licentiate is a Swedish graduate degree approximately half way between a MSc. and a PhD.



Figure 2.2: Distribution of ARTES academic nodes in Sweden.

1. ABB Automation Products AB
2. ABB Digital Plant Technologies AB
3. Arcticus
4. Axis Communications AB
5. Carlstedt Research & Technology AB
6. Combitech Systems AB
7. Datex-Ohmeda AB
8. DDA Consulting
9. Enator Teknik Mälardalen AB
10. Enea Data AB
11. Enea OSE Systems AB
12. Ericsson Microwave Systems AB
13. Ericsson Mobile Communications AB
14. Ericsson Radio Systems AB
15. Ericsson Software Technology AB
16. Ericsson Utvecklings AB
17. HMS Fieldbus Systems AB
18. Innovation Team AB
19. KTHNOC/SUNET
20. Luftfartsverket
21. Mecel AB
22. Northern Real Time Applications
23. Prover Technology AB
24. Rolls-Royce
25. SAAB AB
26. SAAB Automobile AB
27. Saab Ericsson Space AB
28. Scania
29. Siemens-Elema AB
30. Sigma Exallon Systems AB
31. Sysdeco Mimer AB
32. Systemite AB
33. Telelogic AB
34. TTTech Computertechnik GmbH
35. Virtutech
36. Volvo Construction Equipment Components AB
37. Volvo Technological Development Corporation

Table 2.2: ARTES industry nodes

A Real-time graduate student is expected to participate in common activities for dissemination and exploitation of research results. He or she should contribute generally to ARTES activities, e.g. by making reports available in the ARTES web and by actively participating in ARTES events. It is also important that presentations of the research results give the appropriate visibility for and to the real-time community and ARTES.

Graduate students joined ARTES network from all over Sweden. Table 2.3 show the distribution of students active at different universities.

University	ARTES++ Real-Time Graduate Students	ARTES Real-Time Graduate Students	Real-Time Graduate Students	SUM
Chalmers University of Technology	4	9	23	36
Halmstad University	3	3	3	9
University of Skövde	2	2	7	11
Blekinge Institute of Technology	0	2	1	3
Royal Institute of Technology	1	6	5	12
Linköping University	4	3	14	21
Lunds University	2	4	12	18
Luleå University of Technology	3	0	0	3
Mälardalen University	13	7	24	44
Mid Sweden University, in Sundsvall	4	0	0	4
Swedish Institute of Computer Science	0	1	0	1
Umeå University	0	0	1	1
Uppsala University	7	4	16	27
SUM	43	41	106	190

Table 2.3: Distribution of graduate students at different universities. Note. The actual place for the students studies is given, the students were in some cases formally registered at another university.

2.5 Results

The results of a network and research programme like ARTES can be presented from a number of views, such as concrete results, network of contacts established, long term scientific and economic impact, etc. A deeper analysis is outside the scope of the preparation of this book. The presentation here will be focused on more concrete results, including research achievements, industrial involvements, patents, courses and other graduate school activities, mobility, infrastructure support, gender equality.

2.5.1 Research

ARTES gave support for the education of 31 graduate students within 26 ARTES projects and 10 graduate students within 8 PAMP projects (Appendix A.1). The efforts have resulted in 30 PhD and 28 licentiate exams until now (Appendix B.1). Seven students continue towards a PhD, to be completed in the coming years. The research was carried out at the academic research nodes (Table 2.4). The theses completed (including abstracts) are presented in Appendix B.2.

Academic research node	Percent of total project funding
Chalmers University of Technology	21%
Mälardalen University	16%
Royal Institute of Technology	12%
Uppsala University	10%
Linköping University	9%
Lund University	9%
Halmstad University	9%
Blekinge Institute of Technology	5%
University of Skövde	5%
Swedish Institute of Computer Science	3%
SUM	100%

Table 2.4: The academic research nodes and the distribution of project funding to them, the total amount of funding was 62 MSEK.

2.5.2 Industrial involvement

All projects had to show industrial support. In many projects, 2 or 3 industries were involved. In total there were 36 industries participating in ARTES/PAMP projects, and in some cases the same industry was involved in several projects, for example Mecel AB was active in 6 projects

and Ericsson's subsidiaries in 8 projects. A list of the industries that have participated in ARTES is shown in Appendix A.2.

2.5.3 Patents

ARTES supported commercialization of research results by information activities at the summer school, by encouraging the use of free legal advice from SSF and by giving a prize to ARTES RT-graduate students that applied for patents. Concrete results of these efforts include the following:

- Daniel Häggander applied to patent "A method and system for dynamic memory management in an object-oriented program." in July 2000.
- Magnus Ekman applied in November 2001 to patent a mechanism that reduces the energy consumption for the cache-coherence-protocol in a chip multiprocessor. The application of this is to increase the battery lifetime in mobile terminals.
- Henrik Thane applied in December 2002 to patent a real-time debugger for debugging software in embedded real-time systems.

2.5.4 The ARTES graduate school

ARTES had three main graduate school activities

- the annual summer school,
- the graduate student conference, and
- the support for course development as well as for giving courses.

The summer school has been arranged 7 times with 30-100 participants, usually organised in connection to a SNART meeting or conference. This has been an excellent opportunity to invite world-leading scientist to give tutorials and discuss with our students.

The graduate student conference has been a meeting place where 20-30 students meet and present their own research to each other and visit the different universities and some industrial partner.

ARTES has funded 35 graduate courses, presented in Appendix A.2. The course support has initiated cooperation in graduate education all over Sweden. ARTES demand that a course must have participants from other universities than the one giving the course. The funding is then proportional to the external participation. Industrial participants are included in the calculation to support transfer of knowledge to industry. The courses should be possible to follow with rather few meetings. We

have on several occasions found that graduate courses have been given totally without students from the home university.

In 2003 ARTES co-organised a special coordinated effort in graduate education: the European Summer School on Embedded Systems (ESSES), which was held in Västerås and Strängnäs. ESSES was a major summer school that lasted for almost three months (July-September) with lectures by numerous international leading experts on low-power, embedded systems and real-time systems, and with more 100 international and Swedish students. Main organiser of ESSES was Mälardalen Real-Time Research Centre (MRTC), together with the Korean Brain Korea 21 programme and ARTES, and with additional support from the European ARTIST network and CIS in Denmark.

2.5.5 Mobility

ARTES had an extensive mobility programme directed towards all ARTES RT-graduate students. The support was given in units of 10.000 SEK for participation in conferences, and in units of 30.000 SEK for longer visits abroad - usually to do some studies at another lab. Within ARTES++ additional support for industrial visits was introduced, even though some such visits were funded by special board decisions also earlier. Furthermore, the board was always open to suggestions from the students and researchers in case something came up. In a number of cases special support was granted for participation in international conferences held in Sweden. Mobility support generally required the recipient to write a report from the event or visit to be shared with others through the ARTES web (www.artes.uu.se). Until February 2006, 67 students have filed 108 conferences reports. There are six reports from industrial visits and 9 from international visits.

2.5.6 Infrastructure support

The following is a presentation of some of the ARTES activities made to support the development of the real-time and embedded systems area.

A pamphlet – "Take advantage of ARTES" – was made in 2000 to motivate industry to get in contact with researchers in academia. The pamphlet has also been very useful as a brief general presentation of ARTES.

A report – "Embedded Systems and the Future of Swedish IT-research" – was produced by the ARTES board and leading researchers. This document was sent in 2000 to SSF as a contribution to its Strategic Advisory Committees. It argues for the importance of research into embedded systems.

ARTES industry ambassador Anita Andler was engaged in 2001 as industry ambassador, with main task to support the dissemination of research results from the network to industry and society in general. She has arranged industry days at ABB, Ericsson, and SaabTech, as well as a West Sweden vehicle day. These events were organized with emphasis on how the particular industry could benefit from real-time research. The industrial contacts were served a "smorgasbord" of presentations of research activities to choose from. In addition, twenty press releases have been made resulting in at least 17 articles in 11 newspapers and magazines. Several of these were presentations of thesis work by ARTES graduate students. A highly appreciated popular science-writing contest was also arranged, and ARTES and the start up company Zealcore (with roots in project 7; see Appendix A.1) was presented at the Embedded Computing & Real-Time Computer Show in Stockholm.

The ARTIST EU Network of Excellence ARTES participated in the planning of the EU NoE ARTIST, which is a pan-european research network in which essentially all leading european research groups in real-time and embedded systems participate. Partly due to ARTES, the Swedish participation in this network is very high; in fact only France and Germany have a larger participation in ARTIST than Sweden. Also in the follow up NoE ARTIST2, Sweden is one of the largest contributors.

2.5.7 Gender Equality

ARTES has been conscious about the gender imbalance in the real-time and embedded systems field, and taken some actions to encourage and facilitate an increase in the number of female graduate students and researchers.

- The ARTES board decided to allocate dedicated funding for the establishment of a network for female graduate students and researchers.
- To show the graduate students that there are several internationally leading female researchers in the field, at the 2001 summer school ARTES arranged a full day of lectures given by female researchers.

Table 2.5 shows the gender of all real-time graduate students accepted from 1996 until 2005. The proportion of women accepted varies between different years with a mean of 13%.

Year	Female	Male	Sum	Female (%)
1996	1	11	12	8
1997		12	12	0
1998	3	20	23	13
1999	1	21	22	5
2000	6	25	31	19
2001	1	8	9	11
2002	3	9	12	25
2003	3	14	17	18
2004	1	9	10	9
2005	2	10	12	17
SUM	21	140	161	13

Table 2.5: Gender of ARTES Real-Time graduate student registered from 1996 until 2005.

Part I

Design Techniques for Embedded Real-Time Systems

Chapter 3

Introduction

By **Zebo Peng**

Department of Computer and Information Science

Linköping University

Email: zpe@ida.liu.se

Embedded real-time systems are becoming the dominating use of computers and will increase even further as we are entering the era of pervasive computing with massive amounts of cooperating computers controlling virtually all devices in our environment. Performance, predictability and low power consumption are key requirements for many of such systems. These requirements can be met by using specialized hardware components for the time-critical tasks and highly optimized software, together with efficient communication/synchronization mechanisms and embedded database whenever appropriate. Due to the many possible design alternatives and trade-offs between the hardware, software, communication, and database components, the design tasks have become very complex.

One major challenge for the research community is therefore to develop efficient design techniques and tools to support the analysis and synthesis of complex embedded real-time systems, under the pressure of constantly raising time-to-market demands. In recent years, many research activities have been going on in developing such techniques and tools, and several research projects in the ARTES program have produced very interesting results in this area. A few examples of such ARTES research activities are reported in this chapter. A short summary of each of these activities is given in the following paragraphs.

Analysis and synthesis of distributed heterogeneous systems:

Research in this area deal with analysis and synthesis of real-time applications running on heterogeneous distributed architectures. Due to the distributed nature, the communication mechanism and

protocol have to be carefully considered during the analysis and design process in order to guarantee that the timing constraints are satisfied under the competing objective of reducing the cost of the implementation. Several techniques for the analysis and optimization of such systems, taking into account communication protocol and parameters of the underlying architecture platform, have been developed in recent years. The paper “Design Optimization of Multi-Cluster Embedded Systems for Real-Time Applications,” by Paul Pop, Petru Eles, and Zebo Peng, reports on several results by an ARTES project focusing on the development of analysis and optimization techniques for heterogeneous real-time embedded systems. The paper addressed a particular class of such systems called multi-clusters, which consist of several networks interconnected via gateways. It presents a schedulability analysis technique for safety-critical applications distributed on multi-cluster systems, and several design optimization methods related to the partitioning and mapping of functionality, and the packing of application messages to frames. Optimization heuristics for frame packing aiming at producing a schedulable system have also been addressed by this paper.

Low-power systems analysis and synthesis: Reducing energy consumption has become an essential issue for embedded systems design, and many techniques have been developed to perform low-power synthesis of digital systems. The use of dynamic voltage supply processors has become very popular since they offer the best combination of flexibility and energy efficiency. When deploying DVS processors in real-time applications, special strategies are required in order to make sure that the real-time constraints are fulfilled and large amount of energy can be saved. This can for example be done by selecting the appropriate processing speeds and schedule the real-time tasks in a special way. The paper “Using DVS Processors to Achieve Energy Efficiency in Hard Real-Time Systems,” by Flavius Gruian and Krzysztof Kuchcinski, presents several techniques for speed scheduling, ranging from task to task set level, and involving both offline and runtime decisions. The methods described reduce the energy consumption while meeting all deadlines. Some of them are built on top of classic real-time scheduling techniques while others are totally new.

Analysis of systems with stochastic task execution times:

For soft real-time applications, a system is considered to function correctly even if some timeliness requirements are occasionally broken, since this leads only to a tolerable reduction of the service quality. Analysis of such a system should be focused on

the degree to which the system meets its timeliness requirements rather than on a binary answer indicating whether the whole system is schedulable or not. In many soft real-time applications, the task execution times vary also widely since they are dependent on many parameters. In such a context, analysis techniques based on worst case execution time assumption will lead to very pessimistic results, and many techniques have been developed to consider a more realistic model that assumes tasks to have varying execution times with given probability distributions. The paper “Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times,” by Sorin Manolache, Petru Eles and Zebo Peng, presents one of such techniques. It describes an analytic method to produce the expected deadline miss ratio of the tasks and the task graphs that represent a software real-time application. The reported method improves the currently existing ones by providing exact solutions for larger and less restricted task sets. In particular, it allows continuous task execution time probability distributions, and supports different scheduling policy. Furthermore, task dependencies and arbitrary deadlines are supported by the proposed technique.

Synchronization in real-time systems: A real-time system consists of a set of tasks that are related to each other in some way via resource sharing. These tasks have to synchronize in order to avoid inconsistency caused by overlapping and concurrent accesses. The traditional way of synchronizing the access of resources is to use mutual exclusion, with which the tasks can make sure that only one task can have access to a shared resource or data structure at one time. Mutual exclusion is often provided by the operating system as semaphores or mutex locks. However, mutual exclusion (also called blocking synchronization) has several drawbacks, priority-inversion being the most important one for real-time systems. To overcome these drawbacks, many synchronization techniques have been developed, including non-blocking synchronization techniques for shared memory systems. Non-blocking synchronization has the nice property of decoupling scheduling and communication. The paper “Applications of Wait/Lock-Free Protocols to Real-Time Systems,” by Håkan Sundel, Philippas Tsigas and Yi Zhang, presents several fundamental non-blocking Synchronization data structures, implemented with the lock free or the wait free strategies. A lock-free implementation of shared data structures guarantees that at any point in time in any possible execution some operation will complete in a finite number of steps. While in the case of wait-free implementations, each task is guar-

anteed to correctly complete any operation in a bounded number of its own steps. The paper described several interesting results of applying the non-blocking techniques to real-time applications.

Embedded systems with reconfigurable components: The introduction of component-based software development (CBSD) technique into the field of real-time and embedded systems offers many significant benefits. However, there are aspects of real-time and embedded systems that cannot be encapsulated in a component with well-defined interfaces as they crosscut the structure of the overall system, e.g., synchronization, memory optimization, power consumption, and temporal attributes. Aspect-oriented software development (AOSD) has emerged as a new principle for software development that provides an efficient way of modularizing crosscutting concerns in software systems. Several research activities are going on with the objective to allow integration of CBSD and AISD techniques into real-time system development process. The paper “Reconfigurable Embedded Real-Time Systems: a Story of COMET,” by Aleksandra Teaanovi, Dag Nyström, Jörgen Hansson and Christer Norström, documents the results of one of such activities. It deals with the concept of aspectual component-based real-time system development. The concept is based on a design method that assumes decomposition of real-time systems into components and aspects, and provides a real-time component model that supports the notion of time and temporal constraints, space and resource management constraints, and composability. The reported results in this paper show that the successful application of the proposed concept has a positive impact on real-time system development in enabling efficient configuration of real-time systems, improved reusability and flexibility of real-time software, and modularization of crosscutting concerns.

Chapter 4

Design Optimization of Multi-Cluster Embedded Systems for Real-Time Applications

By **Paul Pop, Petru Eles, and Zebo Peng**

Department of Computer and Information Science

Linköping University

Email: {paupo,petel,zpe}@ida.liu.se

An increasing number of real-time applications are today implemented using distributed heterogeneous architectures composed of interconnected networks of processors. The systems are heterogeneous not only in terms of hardware components, but also in terms of communication protocols and scheduling policies. Each network has its own communication protocol, each processor in the architecture can have its own scheduling policy, and several scheduling policies can share a processor. In this context, the task of designing such systems is becoming increasingly important and difficult at the same time. The success of such new design methods depends on the availability of analysis and optimization techniques. In this paper, we present analysis and optimization techniques for heterogeneous real-time embedded systems. We address in more detail a particular class of such systems called multi-clusters, composed of several networks interconnected via gateways. We present a schedulability analysis for safety-critical applications distributed on multi-cluster systems and briefly highlight characteristic design optimization problems: the partitioning and mapping of functionality, and the packing of application messages to frames. Optimization heuristics for frame packing aiming at producing a schedulable system are presented. Extensive experiments and a real-life example show the efficiency of the frame-packing approach.

4.1 Introduction

Embedded real-time systems have to be designed such that they implement correctly the required functionality. In addition, they have to fulfil a wide range of competing constraints: development cost, unit cost, reliability, security, size, performance, power consumption, flexibility, time-to-market, maintainability, correctness, safety, etc. Very important for the correct functioning of such systems are their timing constraints: ‘the correctness of the system behaviour depends not only on the logical results of the computations, but also on the physical instant at which these results are produced’ [1].

Real-time systems have been classified as ‘hard’ real-time and ‘soft’ real-time systems [1]. Basically, hard real-time systems are systems where failing to meet a timing constraint can potentially have catastrophic consequences. For example, a brake-by-wire system in a car failing to react within a given time interval can result in a fatal accident. On the other hand, a multi-media system, which is a soft real-time system, can, under certain circumstances, tolerate a certain amount of delays resulting maybe in a patchier picture, without serious consequences besides some possible inconvenience to the user.

Many real-time applications, following physical, modularity or safety constraints, are implemented using ‘distributed architectures’. Such systems are composed of several different types of hardware components, interconnected in a network. For such systems, the communication between the functions implemented on different nodes has an important impact on the overall system properties such as performance, cost, maintainability, etc.

The analysis and optimisation approaches presented are aimed towards heterogeneous distributed hard real-time systems that implement safety-critical applications where timing constraints are of utmost importance to the correct behaviour of the application.

The chapter is organised in ten sections. Section 4.2 presents the heterogeneous real-time embedded systems addressed, and the type of applications considered. Section 4.3 and 4.4 introduce the current state-of-the-art on the analysis and optimisation of such systems. The rest of the chapter focuses in more detail on some techniques for multi-cluster systems. The hardware and software architecture of multi-clusters, together with the application model, are outlined in Section 4.5. Section 4.6 identifies partitioning and mapping and frame packing as design optimisation problems characteristic to multi-clusters. We present an analysis for multi-cluster systems in Section 4.7, and show, in Section 4.8, how this analysis can be used to drive the optimisation of the packing of application messages to frames. The last two sections present

the experimental results of the frame packing optimisation and conclusions.

4.1.1 Automotive electronics

Although the discussion in this chapter is valid for several application areas, it is useful, for understanding the distributed embedded real-time systems evolution and design challenges, to exemplify the developments in a particular area.

If we take the example of automotive manufacturers, they were reluctant, until recently, to use computer controlled functions onboard vehicles. Today, this attitude has changed for several reasons. First, there is a constant market demand for increased vehicle performance, more functionality, less fuel consumption and less exhausts, all of these at lower costs. Then, from the manufacturers' side, there is a need for shorter time-to-market and reduced development and manufacturing costs. These, combined with the advancements of semiconductor technology, which is delivering ever-increasing performance at lower and lower costs, has led to the rapid increase in the number of electronically controlled functions onboard a vehicle [2].

The amount of electronic content in an average car in 1977 had a cost of \$110. Currently, the cost is \$1341, and it is expected that this figure will reach \$1476 by the year 2005, continuing to increase because of the introduction of sophisticated electronics found until now only in high-end cars [3,4]. It is estimated that in 2006 the electronics inside a car will amount to 25 per cent of the total cost of the vehicle (35 per cent for the high-end models), a quarter of which will be due to semiconductors [3,5]. High-end vehicles currently have up to 100 microprocessors implementing and controlling various parts of their functionality. The total market for semiconductors in vehicles is predicted to grow from \$8.9 billions in 1998 to \$21 billion in 2005, amounting to 10 per cent of the total worldwide semiconductors market [2,3].

At the same time with the increased complexity, the type of functions implemented by embedded automotive electronics systems has also evolved. Thanks to the semiconductors revolution, in the late 1950s, electronic devices became small enough to be installed on board vehicles. In the 1960s the first analogue fuel injection system appeared, and in the 1970s analogue devices for controlling transmission, carburetor, and spark advance timing were developed. The oil crisis of the 1970s led to the demand of engine control devices that improved the efficiency of the engine, thus reducing fuel consumption. In this context, the first microprocessor-based injection control system appeared in 1976 in the United States. During the 1980s, more sophisticated systems began to appear, such as electronically controlled braking systems, dashboards,

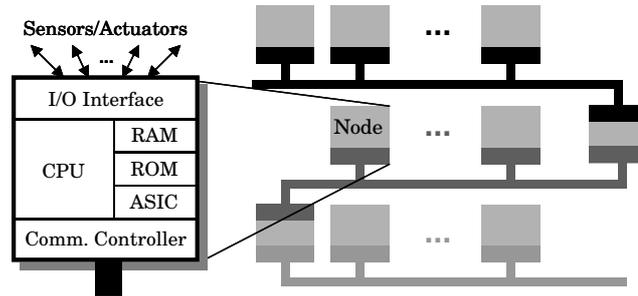


Figure 4.1: Distributed hard real-time systems

information and navigation systems, air conditioning systems, etc. In the 1990s, development and improvement have concentrated in the areas such as safety and convenience. Today, it is not uncommon to have highly critical functions like steering or braking implemented through electronic functionality only, without any mechanical backup, as is the case in drive-by-wire and brake-by-wire systems [6,7].

The complexity of electronics in modern vehicles is growing at a very high pace, and the constraints—in terms of functionality, performance, reliability, cost and time-to-market—are getting tighter. Therefore, the task of designing such systems is becoming increasingly important and difficult at the same time. New design techniques are needed, which are able to

- successfully manage the complexity of embedded systems
- meet the constraints imposed by the application domain
- shorten the time-to-market
- reduce development and manufacturing costs

The success of such new design methods depends on the availability of analysis and optimisation techniques, beyond those corresponding to the state-of-the-art, which are presented in the next section.

4.2 Heterogeneous real-time embedded systems

4.2.1 Heterogeneous hardware architecture

Currently, distributed real-time systems are implemented using architectures where each node is dedicated to the implementation of a single function or class of functions. The complete system can be, in general, composed of several networks, interconnected with each other (see Figure 4.1). Each network has its own communication protocol, and

inter-network communication is via a *gateway* which is a node connected to both networks. The architecture can contain several such networks, having different types of topologies.

A network is composed of several different types of hardware components, called 'nodes'. Typically, every node, also called 'electronic control unit' (ECU), has a communication controller, CPU, RAM, ROM and an I/O interface to sensors and actuators. Nodes can also have ASICs in order to accelerate parts of their functionality.

The microcontrollers used in a node and the type of network protocol employed are influenced by the nature of the functionality and the imposed real-time, fault-tolerance and power constraints. In the automotive electronics area, the functionality is typically divided in two classes, depending on the level of criticalness:

- *Body electronics* refers to the functionality that controls simple devices such as the lights, the mirrors, the windows, the dashboard. The constraints of the body electronic functions are determined by the reaction time of the human operator that is in the range of 100–200 ms. A typical body electronics system within a vehicle consists of a network of 10–20 nodes that are interconnected by a low bandwidth communication network such as LIN. A node is usually implemented using a single-chip 8-bit microcontroller (e.g. Motorola-a 68HC05 or Motorola 68HC11) with some hundred bytes of RAM and kilobytes of ROM, I/O points to connect sensors and to control actuators, and a simple network interface. Moreover, the memory size is growing by more than 25 per cent each year [6,8].
- *System electronics* are concerned with the control of vehicle functions that are related to the movement of the vehicle. Examples of system electronics applications are engine control, braking, suspension, vehicle dynamics control. The timing constraints of system electronic functions are in the range of a couple of milliseconds to 20 ms, requiring 16- or 32-bit microcontrollers (e.g. Motorola 68332) with about 16 kB of RAM and 256 kB of ROM. These microcontrollers have built-in communication controllers (e.g. the 68HC11 and 68HC12 automotive family of microcontrollers have an on-chip CAN controller), I/O to sensors and actuators, and are interconnected by high bandwidth networks [6,8].

Section 4.5 presents more details concerning the hardware and software architecture considered by our analysis and optimisation techniques.

4.2.2 Heterogeneous communication protocols

As the communications become a critical component, new protocols are needed that can cope with the high bandwidth and predictability required.

There are several communication protocols for real-time networks. Among the protocols that have been proposed for vehicle multiplexing, only the Controller Area Network (CAN) [9], the Local Interconnection Network (LIN) [10] and SAE's J1850 [11] are currently in use on a large scale. Moreover, only a few of them are suitable for safety-critical applications where predictability is mandatory [12]. Rushby [12] provides a survey and comparison of communication protocols for safety-critical embedded systems. Communication activities can be triggered either dynamically, in response to an event, or statically, at predetermined moments in time.

- Therefore, on one hand, there are protocols that schedule the messages statically based on the progression of time, for example, the SAFEbus [13] and SPIDER [14] protocols for the avionics industry, and the TTCAN [15] and Time-Triggered Protocol (TTP) [1] intended for the automotive industry.
- On the other hand, there are several communication protocols where message scheduling is performed dynamically, such as CAN used in a large number of application areas including automotive electronics, LonWorks [16] and Profibus [17] for real-time systems in general, etc. Out of these, CAN is the most well known and widespread event-driven communication protocol in the area of distributed embedded real-time systems.
- However, there is also a hybrid type of communication protocols, such as Byteflight [18] introduced by BMW for automotive applications and the FlexRay protocol [19], that allows the sharing of the bus by event-driven and time-driven messages.

The time-triggered protocols have the advantage of simplicity and predictability, while event-triggered protocols are flexible and have low cost. Moreover, protocols such as TTP offer fault-tolerant services necessary in implementing safety-critical applications. However, it has been shown [20] that event-driven protocols such as CAN are also predictable, and fault-tolerant services can also be offered on top of protocols such as the TTCAN. A hybrid communication protocol such as FlexRay offers some of the advantages of both worlds.

4.2.3 Heterogeneous scheduling policies

The automotive suppliers will select, based on their own requirements, the scheduling policy to be used in their ECU. The main approaches to scheduling are

- *Static cyclic scheduling* algorithms are used to build, off-line, a schedule table with activation times for each process, such that the timing constraints of processes are satisfied.
- *Fixed priority scheduling* (FPS). In this scheduling approach each process has a fixed (static) priority which is computed off-line. The decision on which ready process to activate is taken on-line according to their priority.
- *Earliest deadline first* (EDF). In this case, that process will be activated which has the nearest deadline.

Typically, processes scheduled off-line using static cyclic scheduling are non-pre-emptable, while processes scheduled using techniques such as FPS and EDF are pre-emptable. Another important distinction is between the event- and time-triggered approaches.

- *Time-triggered*. In the time-triggered approach activities are initiated at predetermined points in time. In a distributed time-triggered system it is assumed that the clocks of all nodes are synchronised to provide a global notion of time. Time-triggered systems are typically implemented using ‘non-pre-emptive static cyclic scheduling’, where the process activation or message communication is done based on a schedule table built off-line.
- *Event-triggered*. In the event-triggered approach activities happen when a significant change of state occurs. Event-triggered systems are typically implemented using ‘pre-emptive fixed-priority-based scheduling’, or ‘earliest deadline first’, where, as response to an event, the appropriate process is invoked to service it.

There has been a long debate in the real-time and embedded systems communities concerning the advantages of each approach [1,21,22]. Several aspects have been considered in favour of one or the other approach, such as flexibility, predictability, jitter control, processor utilisation and testability.

Lönn and Axelsson [23] have performed an interesting comparison of ET and TT approaches from a more industrial, in particular automotive perspective. The conclusion is that one has to choose the right approach, depending on the particularities of the application.

For certain applications, several scheduling approaches can be used together. Efficient implementation of new, highly sophisticated automotive applications, entails the use of time-triggered process sets together with event-triggered ones implemented on top of complex distributed architectures.

4.2.4 Distributed safety-critical applications

Considering the automotive industry, the way functionality has been distributed on an architecture has evolved over time. Initially, distributed real-time systems were implemented using architectures where each node is dedicated to the implementation of a single function or class of functions, allowing the system integrators to purchase nodes implementing required functions from different vendors, and to integrate them into their system [24]. There are several problems related to this restricted mapping of functionality:

- The number of such nodes in the architecture has exploded, reaching, for example, more than 100 in a high-end car, incurring heavy cost and performance penalties.
- The resulting solutions are sub-optimal in many aspects, and do not use the available resources efficiently in order to reduce costs. For example, it is not possible to move a function from one node to another node where there are enough available resources (e.g. memory, computation power).
- Emerging functionality, such as brake-by-wire in the automotive industry, is inherently distributed, and achieving an efficient fault-tolerant implementation is very difficult in the current setting.

This has created a huge pressure to reduce the number of nodes by integrating several functions in one node and, at the same time, to distribute certain functionality over several nodes (see Figure 4.2). Although an application is typically distributed over one single network, we begin to see applications that are distributed across several networks. For example, in Figure 4.2, the third application, represented as black dots, is distributed over two networks.

This trend is driven by the need to further reduce costs, improve resource usage, but also by application constraints such as having to be physically close to particular sensors and actuators. Moreover, not only are these applications distributed across networks, but their functions can exchange critical information through the gateway nodes.

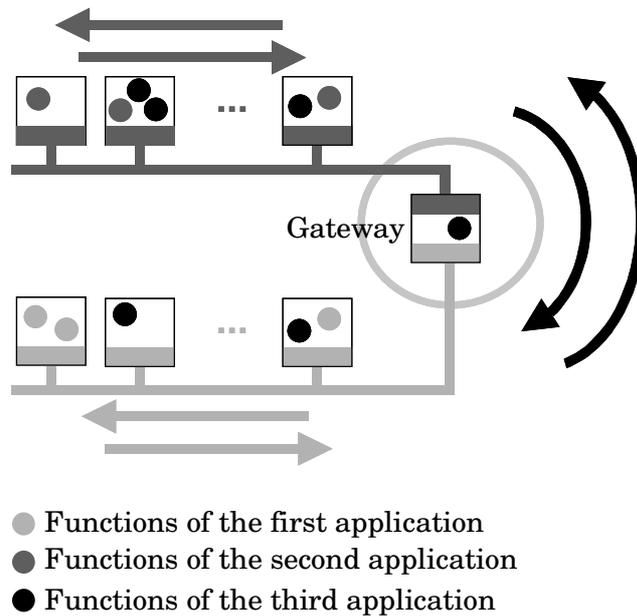


Figure 4.2: Distributed safety-critical applications

4.3 Schedulability analysis

There is a large quantity of research [1,25,26] related to scheduling and schedulability analysis, with results having been incorporated in analysis tools such as TimeWiz [27], RapidRMA [28], RTA-OSEK Planner [29] and Aires [30]. The tools determine if the timing constraints of the functionality are met, and support the designer in exploring several design scenarios, and help to design optimised implementations.

Typically, the timing analysis considers independent processes running on single processors. However, very often functionality consists of distributed processes that have data and control dependencies, exclusion constraints, etc. New schedulability analysis techniques are needed which can handle distributed applications, data and control dependencies, and accurately take into account the details of the communication protocols that have an important influence on the timing properties. Moreover, highly complex and safety-critical applications can in the future be distributed across several networks, and can use different, heterogeneous, scheduling policies.

Pre-emptive scheduling of independent processes with static priorities running on single-processor architectures has its roots in the work of Liu and Layland [31]. The approach has been later extended to accommodate more general computational models and has also been applied to distributed systems [32]. Several surveys on this topic have been

published [25,26,33]. Static cyclic scheduling of a set of data dependent software processes on a multiprocessor architecture has also been intensively researched [1,34].

Lee *et al* [35] has proposed an earlier deadline first strategy for non-pre-emptive scheduling of process with possible data dependencies. Pre-emptive and non-pre-emptive static scheduling are combined in the cosynthesis environment proposed by Dave *et al* [36,37]. In many of the previous scheduling approaches researchers have assumed that processes are scheduled independently. However, processes can be sporadic or aperiodic, are seldom independent and normally they exhibit precedence and exclusion constraints. Knowledge regarding these dependencies can be used in order to improve the accuracy of schedulability analyses and the quality of the produced schedules [38].

It has been claimed [22] that static cyclic scheduling is the only approach that can provide efficient solutions to applications that exhibit data dependencies. However, advances in the area of fixed priority pre-emptive scheduling show that such applications can also be handled with other scheduling strategies [39].

One way of dealing with data dependencies between processes in the context of static priority-based scheduling has been indirectly addressed by the extensions proposed for the schedulability analysis of distributed systems through the use of the ‘release jitter’ [32]. Release jitter is the worst-case delay between the arrival of a process and its release (when it is placed in the ready-queue for the processor) and can include the communication delay due to the transmission of a message on the communication channel.

In [32,40] time ‘offset’ relationships and ‘phases’, respectively, are used in order to model data dependencies. Offset and phase are similar concepts that express the existence of a fixed interval in time between the arrivals of sets of processes. The authors show that by introducing such concepts into the computational model, the pessimism of the analysis is significantly reduced when bounding the time behaviour of the system. The concept of ‘dynamic offsets’ has been later introduced and used to model data dependencies [41].

Currently, more and more real-time systems are used in physically distributed environments and have to be implemented on distributed architectures in order to meet reliability, functional, and performance constraints.

Researchers have often ignored or very much simplified the communication infrastructure. One typical approach is to consider communications as processes with a given execution time (depending on the amount of information exchanged) and to schedule them as any other process, without considering issues such as communication protocol, bus arbitration, packaging of messages, clock synchronisation, etc. [40].

Tindell and Clark [32] integrate processor and communication scheduling and provide a ‘holistic’ schedulability analysis in the context of distributed real-time systems. The validity of the analysis has been later confirmed in [42].

In the case of a distributed system the response time of a process also depends on the communication delay due to messages. In [32] the analysis for messages is done in a similar way as for processes: a message is seen as a non-pre-emptable process that is ‘running’ on a bus. The response time analyses for processes and messages are combined by realising that the ‘jitter’ (the delay between the ‘arrival’ of a process – the time when it becomes ready for execution – and the start of its execution) of a destination process depends on the ‘communication delay’ (the time it takes for a message to reach the destination process, from the moment it has been produced by the sender process) between sending and receiving a message. Several researchers have provided analyses that bound the communication delay for a given communication protocol:

- CAN protocol [20];
- time-division multiple access protocol [32];
- asynchronous transfer mode protocol [43];
- token ring protocol [44],
- fiber distributed data interface protocol [45].
- time-triggered protocol [46];
- FlexRay protocol [47].

Based on their own requirements, the suppliers choose one particular scheduling policy to be used. However, for certain applications, several scheduling approaches can be used together.

One approach to the design of such systems, is to allow ET and TT processes to share the same processor as well as static (TT) and dynamic (ET) communications to share the same bus. Bus sharing of TT and ET messages is supported by protocols which support both static and dynamic communication [19]. We have addressed the problem of timing analysis for such systems [47].

A fundamentally different architectural approach to heterogeneous TT/ET systems is that of heterogeneous multi-clusters, where each cluster can be either TT or ET. In a ‘time-triggered cluster’ processes and messages are scheduled according to a static cyclic policy, with the bus implementing a TDMA protocol, for example, the time-triggered protocol. On ‘event-triggered clusters’ the processes are scheduled according

to a priority-based pre-emptive approach, while messages are transmitted using the priority-based CAN bus. In this context, we have proposed an approach to schedulability analysis for multi-cluster distributed embedded systems [48]. This analysis will be outlined in Section 4.7.

When several event-driven scheduling policies are used in a heterogeneous system, another approach [49] to the verification of timing properties is to couple the analysis of local scheduling strategies via an event interface model.

4.4 Design optimisation

4.4.1 Traditional design methodology

There are several methodologies for real-time embedded systems design. The aim of a design methodology is to coordinate the design tasks such that the time-to-market is minimised, the design constraints are satisfied and various parameters are optimised.

The main design tasks that have to be performed are described in the following sections.

Functional analysis and design

The functionality of the host system, into which the electronic system is embedded, is normally described using a formalism from that particular domain of application. For example, if the host system is a vehicle, then its functionality is described in terms of control algorithms using differential equations, which are modelling the behaviour of the vehicle and its environment. At the level of the embedded real-time system which controls the host system, the functionality is typically described as a set of functions, accepting certain inputs and producing some output values.

The typical automotive application is a control application. The controller reads inputs from sensors, and uses the actuators to control the physical environment (the vehicle). A controller can have several modes of operation, and can interact with other electronic functions, or with the driver through switches and instruments.

During the functional analysis and design stage, the desired functionality is specified, analysed and decomposed into sub-functions based on the experience of the designer. Several suppliers and manufacturers have started to use tools such as Statemate [50], Matlab/Simulink [51], ASCET/SD [52] and SystemBuild/ MatrixX [53] for describing the functionality, in order to eliminate the ambiguities and to avoid producing incomplete or incoherent specifications.

At the level of functional analysis the exploration is currently limited to evaluating several alternative control algorithms for solving the control problem. Once the functionality has been captured using tools such as Matlab/Simulink, useful explorations can involve simulations of executable specifications in order to determine the correctness of the behaviour, and to assess certain properties of chosen solutions.

Architecture selection and mapping

The architecture selection task decides what components to include in the hardware architecture and how these components are connected.

According to current practice, architecture selection is an *ad hoc* process, based on the experience of the designer and previous product versions.

The mapping task has to decide what part of the functionality should be implemented on which of the selected components.

The manufacturers integrate components from suppliers, and thus the design space is severely restricted in current practice, by the fact that the mapping of functionality to an ECU is fixed.

Software design and implementation

This is the phase in which the software is designed and the code is written.

The code for the functions is developed manually for efficiency reasons, and thus the exploration that would be allowed by automatic code generation is limited.

At this stage the correctness of the software is analysed through simulations, but there is no analysis of timing constraints, which is left for the scheduling and schedulability analysis stage.

Scheduling and schedulability analysis

Once the functions have been defined and the code has been written, the scheduling task is responsible for determining the execution strategy for the functions ‘inside an ECU’, such that the timing constraints are satisfied.

Simulation is extensively used to determine if the timing constraints are satisfied. However, simulations are very time consuming and provide no guarantees that the timing constraints are met.

In the context of static cyclic scheduling, deriving a schedule table is a complex design exploration problem. Static cyclic scheduling of a set of data-dependent software processes on a multiprocessor architecture has received a lot of attention [1,34]. Such research has been used in

commercial tools such as TTP-Plan [54] which derives the static schedules for processes and messages in a time-triggered system using the time-triggered protocol for communication.

If fixed priority pre-emptive scheduling is used, exploration is used to determine how to allocate priorities to a set of distributed processes [55]. Their priority assignment heuristic is based on the schedulability analysis from [32]. For earliest deadline first the issue of distributing the global deadlines to local deadlines has to be addressed [56].

Integration

In this phase the manufacturer has to integrate the ECUs from different suppliers.

There is a lack of tools that can analyse the performance of the interacting functionality, and thus the manufacturer has to rely on simulation runs using the realistic environment of a prototype car. Detecting potential problems at such a late stage requires time-consuming extensive simulations. Moreover, once a problem is identified it takes a very long time to go through all the previous stages in order to fix it. This leads to large delays on the time-to-market.

In order to reduce the large simulation times, and to guarantee that potential violations of timing constraints are detected, manufacturers have started to use in-house analysis tools and commercially available tools such as Volcano Network Architect (for the CAN and LIN buses) [57].

Volcano makes inter-ECU communication transparent for the programmer. The programmer only deals with ‘signals’ that have to be sent and received, and the details of the network are hidden. Volcano provides basic API calls for manipulating signals. To achieve interoperability between ECUs from different suppliers, Volcano uses a ‘publish/subscribe’ model for defining the signal requirements. Published signals are made available to the system integrator by the suppliers, while subscribed signals are required as inputs to the ECU. The system integrator makes the publish/subscribe connections by creating a set of CAN frames, and creating a mapping between the data in frames and signals [58]. Volcano uses the analysis by Tindell *et al* [20] for bounding the communication delay of messages transmitted using the CAN bus.

Calibration, testing, verification

These are the final stages of the design process. Because not enough analysis, testing and verification has been done in earlier stages of the design, these stages tend to be very time consuming, and problems identified here lead to large delays in product delivery.

4.4.2 Function architecture co-design and platform-based design

New design methodologies are needed, which can handle the increasing complexity of heterogeneous systems, and their competing requirements in terms of performance, reliability, low power consumption, cost, time-to-market, etc. As the complexity of the systems continues to increase, the development time lengthens dramatically, and the manufacturing costs become prohibitively high. To cope with this complexity, it is necessary to reuse as much as possible at all levels of the design process, and to work at higher and higher abstraction levels.

‘Function/architecture co-design’ is a design methodology proposed in [59,60], which addresses the design process at higher abstraction levels. Function/architecture co-design uses a top-down synthesis approach, where trade-offs are evaluated at a high level of abstraction. The main characteristic of this methodology is the use, at the same time with the top-down synthesis, of a bottom-up evaluation of design alternatives, without the need to perform a full synthesis of the design. The approach to obtaining accurate evaluations is to use an accurate modelling of the behaviour and architecture, and to develop analysis techniques that are able to derive estimates and to formally verify properties relative to a certain design alternative. The determined estimates and properties, together with user-specified constraints, are then used to drive the synthesis process.

Thus, several architectures are evaluated to determine if they are suited for the specified system functionality. There are two extremes in the degrees of freedom available for choosing an architecture. At one end, the architecture is already given, and no modifications are possible. At the other end of the spectrum, no constraints are imposed on the architecture selection, and the synthesis task has to determine, from scratch, the best architecture for the required functionality. These two situations are, however, not common in practice. Often, a ‘hardware platform’ is available, which can be ‘parameterised’ (e.g. size of memory, speed of the buses, etc.). In this case, the synthesis task is to derive the parameters of the platform architecture such that the functionality of the system is successfully implemented. Once an architecture is determined and/or parameterised, the function/architecture co-design continues with the mapping of functionality onto the instantiated architecture.

This methodology has been used in research tools such as Polis [61] and Metropolis [62], and has also led to commercial tools such as the Virtual Component Co-design (VCC) [63].

In order to reduce costs, especially in the case of a mass market product, the system architecture is usually reused, with some modifications,

for several product lines. Such a common architecture is denoted by the term ‘platform’, and consequently the design tasks related to such an approach are grouped under the term ‘platform-based design’ [64]. The platform consists of a hardware infrastructure together with software components that will be used for several product versions, and will be shared with other product lines, in the hope to reduce costs and the time-to-market.

Keutzer *et al* [64] have proposed techniques for deriving such a platform for a given family of applications. Their approach can be used within any design methodology for determining a system platform that later on can be parameterised and instantiated to a desired system architecture.

Considering a given application or family of applications, the system platform has to be instantiated, deciding on certain parameters, and lower level details, in order to suit that particular application(s). The search for an architecture instance starts from a certain platform, and a given application. The application is mapped and compiled on an architecture instance, and the performance numbers are derived, typically using simulation. If the designer is not satisfied with the performance of the instantiated architecture, the process is repeated.

In the remainder of the chapter we will consider a platform consisting of event- and time-triggered clusters, using the CAN and TTP protocols for communication, respectively. We will discuss analysis and optimisation techniques for the configuration of the platform such that the given application is schedulable.

4.5 Multi-cluster systems

One class of heterogeneous real-time embedded systems is that of ‘multi-cluster’ systems. We consider architectures consisting of two clusters, one time-triggered and the other event-triggered, interconnected by gateways (see Figure 4.2):

- In a ‘time-triggered cluster’ (TTC) processes and messages are scheduled according to a static cyclic policy, with the bus implementing a TDMA protocol such as, e.g. the time-triggered protocol (TTP) [65].
- On ‘event-triggered clusters’ (ETC) the processes are scheduled according to a priority-based pre-emptive approach, while messages are transmitted using the priority-based CAN bus [9].

The next two sections present the hardware and software architecture of a two-cluster system, while Section 4.5.3 presents the application model

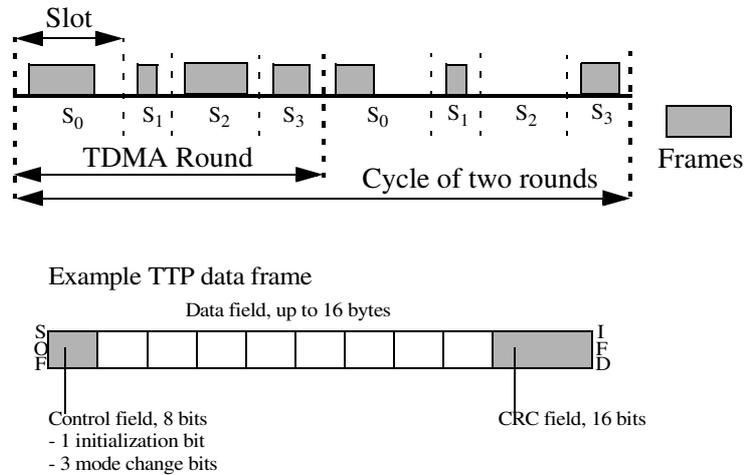


Figure 4.3: Time-triggered protocol

used. Section 4.6 will introduce design problems characteristic for multi-cluster systems composed of time-triggered clusters interconnected with event-triggered clusters: the partitioning of functionality between the TT and ET clusters, the mapping of functionality to the nodes inside a cluster and the packing of application message to frames on the TTP and CAN buses. Then, Section 4.8 will present two optimisation strategies for the frame packing problem.

4.5.1 Hardware architecture

A ‘cluster’ is composed of nodes which share a broadcast communication channel. Let \mathcal{N}_T (\mathcal{N}_E) be the set of nodes on the TTC (ETC). Every ‘node’ $\mathcal{N}_i \in \mathcal{N}_T \cup \mathcal{N}_E$ includes a communication controller and a CPU, along with other components. The gateways, connected to both types of clusters, have two communication controllers, for TTP and CAN. The communication controllers implement the protocol services, and run independently of the node’s CPU. Communication with the CPU is performed through a ‘Message Base Interface’ (MBI); see Figure 4.5.

Communication between the nodes on a TTC is based on the TTP [65]. The TTP integrates all the services necessary for fault-tolerant real-time systems. The bus access scheme is time-division multiple access (TDMA), meaning that each node N_i on the TTC, including the gateway node, can transmit only during a predetermined time interval, the TDMA slot S_i . In such a slot, a node can send several messages packed in a frame. A sequence of slots corresponding to all the nodes in the architecture is called a TDMA round. A node can have only one slot in a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically. The sequence and length of the

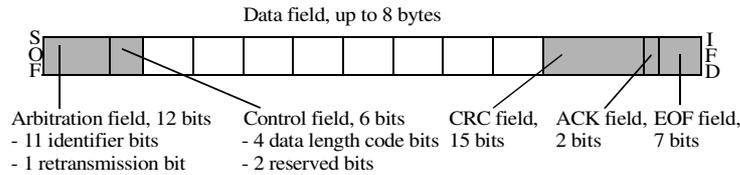


Figure 4.4: Controller area network data frame (CAN 2.0A)

slots are the same for all the TDMA rounds. However, the length and contents of the frames may differ.

The TDMA access scheme is imposed by a message descriptor list (MEDL) that is located in every TTP controller. The MEDL serves as a schedule table for the TTP controller which has to know when to send/receive a frame to/from the communication channel.

There are two types of frames in the TTP. The initialisation frames, or I-frames, which are needed for the initialisation of a node, and the normal frames, or N-frames, which are the data frames containing, in their data field, the application messages. A TTP data frame (Figure 4.3) consists of the following fields: start of frame bit (SOF), control field, a data field of up to 16 bytes containing one or more messages, and a cyclic redundancy check (CRC) field. Frames are delimited by the inter-frame delimiter (IDF, 3 bits).

For example, the data efficiency of a frame that carries 8 bytes of application data, i.e. the percentage of transmitted bits which are the actual data bits needed by the application, is 69.5 per cent (64 data bits transmitted in a 92-bit frame, without considering the details of a particular physical layer). Note that no identifier bits are necessary, as the TTP controllers know from their MEDL what frame to expect at a given point in time. In general, the protocol efficiency is in the range of 60–80 per cent [66].

On an ETC, the CAN [9] protocol is used for communication. The CAN bus is a priority bus that employs a collision avoidance mechanism, whereby the node that transmits the frame with the highest priority wins the contention. Frame priorities are unique and are encoded in the frame identifiers, which are the first bits to be transmitted on the bus.

In the case of CAN 2.0A, there are four frame types: data frame, remote frame, error frame and overload frame. We are interested in the composition of the data frame, depicted in Figure 4.4. A data frame contains seven fields: SOF, arbitration field that encodes the 11 bits frame identifier, a control field, a data field up to 8 bytes, a CRC field, an acknowledgement (ACK) field and an end of frame field (EOF).

In this case, for a frame that carries 8 bytes of application data, we will have an efficiency of 47.4 per cent [67]. The typical CAN protocol efficiency is in the range of 25–35 per cent [66].

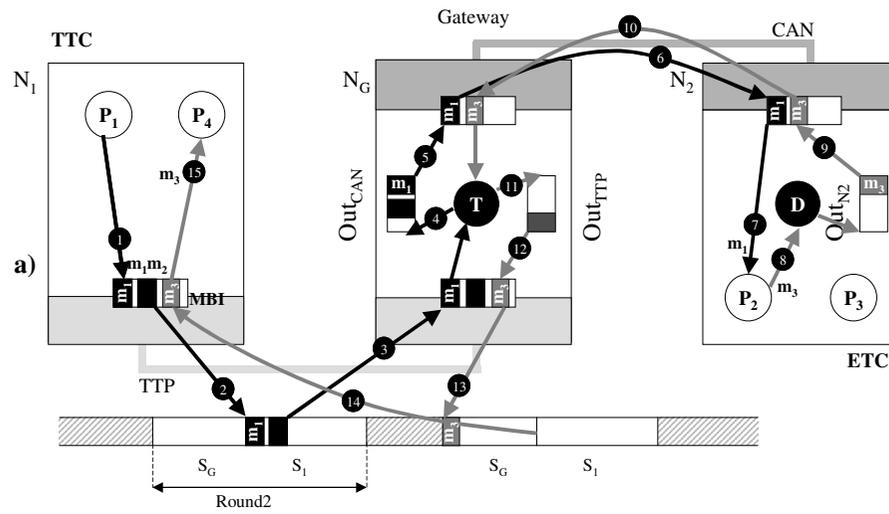


Figure 4.5: A message passing example

4.5.2 Software architecture

A real-time kernel is responsible for activation of processes and transmission of messages on each node. On a TTC, the processes are activated based on the local schedule tables, and messages are transmitted according to the MEDL. On an ETC, we have a scheduler that decides on activation of ready processes and transmission of messages, based on their priorities.

In Figure 4.5 we illustrate our message passing mechanism. Here we concentrate on the communication between processes located on different clusters. We have previously presented the message passing within a TTC [68], and the infrastructure needed for communications in an ETC [20].

Let us consider the example in Figure 4.5, where we have an application consisting of four processes and four messages (depicted in Figure 4.5(b)) mapped on the two clusters in Figure 4.5(c). Processes P_1 and P_4 are mapped on node N_1 of the TTC, while P_2 and P_3 are mapped on node N_2 of the ETC. Process P_1 sends messages m_1 and m_2 to processes P_2 and P_3 , respectively, while P_2 and P_3 send messages m_3 and m_4 to P_4 . All messages have a size of one byte.

The transmission of messages from the TTC to the ETC takes place in the following way (see Figure 4.5). P_1 , which is statically scheduled, is activated according to the schedule table, and when it finishes it calls the send kernel function in order to send m_1 and m_2 , indicated in the figure by the number (1). Messages m_1 and m_2 have to be sent from node N_1 to node N_2 . At a certain time, known from the schedule table, the kernel transfers m_1 and m_2 to the TTP controller by packing them into a frame in the MBI. Later on, the TTP controller knows from its MEDL when it has to take the frame from the MBI, in order to broadcast it on the bus. In our example, the timing information in the schedule table of the kernel and the MEDL is determined in such a way that the broadcasting of the frame is done in the slot S_1 of round 2 (2). The TTP controller of the gateway node N_G knows from its MEDL that it has to read a frame from slot S_1 of round 2 and to transfer it into its MBI (3). Invoked periodically, having the highest priority on node N_G , and with a period which guarantees that no messages are lost, the gateway process T copies messages m_1 and m_2 from the MBI to the TTP-to-CAN priority-ordered message queue Out_{CAN} (4). Let us assume that on the ETC messages m_1 and m_2 are sent independently, one per frame. The highest priority frame in the queue, in our case the frame f_1 containing m_1 , will tentatively be broadcast on the CAN bus (5). Whenever f_1 will be the highest priority frame on the CAN bus, it will successfully be broadcast and will be received by the interested nodes, in our case node N_2 (6). The CAN communication controller of node N_2 receiving f_1 will

copy it in the transfer buffer between the controller and the CPU, and raise an interrupt which will activate a delivery process, responsible to activate the corresponding receiving process, in our case P_2 , and hand over message m_1 that finally arrives at the destination (7).

Message m_3 (depicted in Figure 4.5 as a grey rectangle labelled ‘ m_3 ’) sent by process P_2 from the ETC will be transmitted to process P_4 on the TTC. The transmission starts when P_2 calls its send function and enqueues m_3 in the priority-ordered Out_{N_2} queue (8). When the frame f_3 containing m_3 has the highest priority on the bus, it will be removed from the queue (9) and broadcast on the CAN bus (10). Several messages can be packed into a frame in order to increase the efficiency of data transmission. For example, m_3 can wait in the queue until m_4 is produced by P_3 , in order to be packed together with m_4 in a frame. When f_3 arrives at the gateway’s CAN controller it raises an interrupt. Based on this interrupt, the gateway transfer process T is activated, and m_3 is unpacked from f_3 and placed in the Out_{TTP} FIFO queue (11). The gateway node N_G is only able to broadcast on the TTC in the slot S_G of the TDMA rounds circulating on the TTP bus. According to the MEDL of the gateway, a set of messages not exceeding size_{S_G} of the data field of the frame travelling in slot S_G will be removed from the front of the Out_{TTP} queue in every round, and packed in the S_G slot (12). Once the frame is broadcast (13) it will arrive at node N_1 (14), where all the messages in the frame will be copied in the input buffers of the destination processes (15). Process P_4 is activated according to the schedule table, which has to be constructed such that it accounts for the worst-case communication delay of message m_3 , bounded by the analysis in Section 4.7.1, and, thus, when P_4 starts executing it will find m_3 in its input buffer.

As part of our frame packing approach, we generate all the MEDLs on the TTC (i.e. the TT frames and the sequence of the TDMA slots), as well as the ET frames and their priorities on the ETC such that the global system is schedulable.

4.5.3 Application model

The functionality of the host system, into which the electronic system is embedded, is normally described using a formalism from that particular domain of application. For example, if the host system is a vehicle, then its functionality is described in terms of control algorithms using differential equations, which are modelling the behaviour of the vehicle and its environment. At the level of the embedded system which controls the host system, viewed as the system level for us, the functionality is typically described as a set of functions, accepting certain inputs and producing some output values.

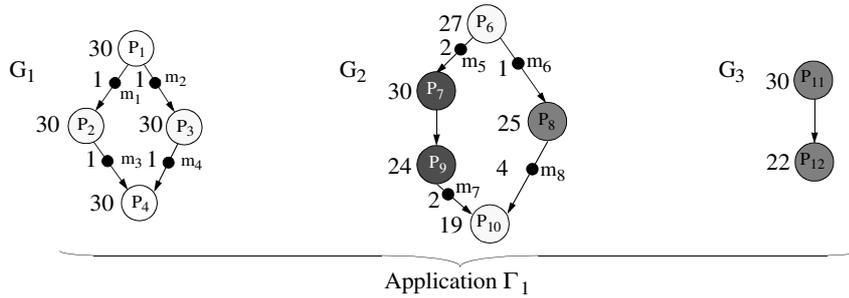


Figure 4.6: Application model

There is a lot of research in the area of system modelling and specification, and an impressive number of representations have been proposed. Edward, [69] presents an overview, classification and comparison of different design representations and modelling approaches.

The scheduling and mapping design tasks deal with sets of interacting processes. A ‘process’ is a sequence of computations (corresponding to several building blocks in a programming language) which starts when all its inputs are available. When it finishes executing, the process produces its output values. Researchers have used, for example, ‘dataflow process networks’ (also called ‘task graphs’, or ‘process graphs’) [70] to describe interacting processes, and have represented them using directed acyclic graphs, where a node is a process and the directed arcs are dependencies between processes.

Thus, we model an application Γ as a set of process graphs $G_i \in \Gamma$ (see Figure 4.6). Nodes in the graph represent processes and arcs represent dependency between the connected processes. A ‘process’ is a sequence of computations (corresponding to several building blocks in a programming language) which starts when all its inputs are available. When it finishes executing, the process produces its output values. Processes can be pre-emptable or non-pre-emptable. ‘Non-pre-emptable’ processes are processes that cannot be interrupted during their execution, and are mapped on the TTC. ‘Pre-emptable’ processes can be interrupted during their execution, and are mapped on the ETC. For example, a higher priority process has to be activated to service an event, in this case, the lower priority process will be temporarily pre-empted until the higher priority process finishes its execution.

A process graph is polar, which means that there are two nodes, called source and sink, that conventionally represent the first and last process. If needed, these nodes are introduced as dummy processes so that all other nodes in the graph are successors of the source and predecessors of the sink, respectively.

The communication time between processes mapped on the same processor is considered to be part of the process worst-case execution time and is not modelled explicitly. Communication between processes mapped to different processors is performed by message passing over the buses and, if needed, through the gateway. Such message passing is modelled as a communication process inserted on the arc connecting the sender and the receiver process (the black dots in Figure 4.6).

Potential communication between processes in different applications is not part of the model. Technically, such a communication is implemented by the kernels based on asynchronous non-blocking send and receive primitives. Such messages are considered non-critical and are not affected by real-time constraints. Therefore, communications of this nature will not be addressed.

Each process P_i is mapped on a processor $\mathcal{M}(P_i)$ (mapping represented by hashing in Figure 4.6), and has a worst-case execution time C_i on that processor (depicted to the left of each node). The designer can provide manually such worst-case times, or tools can be used in order to determine the worst-case execution time of a piece of code on a given processor [71].

For each message we know its size (in bytes, indicated to its left), and its period, which is identical with that of the sender process. Processes and messages activated based on events also have a uniquely assigned priority, $priority_{P_i}$ for processes and $priority_{m_i}$ for messages.

All processes and messages belonging to a process graph G_i have the same period $T_i = T_{G_i}$ which is the period of the process graph. A deadline D_{G_i} is imposed on each process graph G_i . Deadlines can also be placed locally on processes. Release times of some processes as well as multiple deadlines can be easily modelled by inserting dummy nodes between certain processes and the source or the sink node, respectively. These dummy nodes represent processes with a certain execution time but which are not allocated to any processing element.

4.6 Multi-cluster optimisation

Considering the types of applications and systems described in the previous section, and using the analysis outlined in Section 4.7, several design optimisation problems can be addressed.

In this section, we present problems which are characteristic to applications distributed across multi-cluster systems consisting of heterogeneous TT and ET networks:

- Section 4.6.1 briefly outlines the problem of partitioning the processes of an application into time- and event-triggered domains, and their mapping to the nodes of the clusters.

- Section 4.6.2 presents the problem of packing of messages to frames, which is of utmost importance in cost-sensitive embedded systems where resources, such as communication bandwidth, have to be fully utilised [58,72,73]. This problem will be discussed in more detail in Section 4.8.

The goal of these optimisation problems is to produce an implementation which meets all the timing constraints (i.e. the application is schedulable).

In order to drive our optimisation algorithms towards schedulable solutions, we characterise a given frame packing configuration using the degree of schedulability of the application. The ‘degree of schedulability’ [74] is calculated as:

$$\delta_{\Gamma} = \begin{cases} c_1 = \sum_{i=1}^n \max(0, r_i - D_i), & \text{if } c_1 > 0 \\ c_2 = \sum_{i=1}^n (r_i - D_i), & \text{if } c_1 = 0 \end{cases} \quad (4.1)$$

where n is the number of processes in the application, r_i is the worst-case response time of a process P_i and D_i its deadline. The worst-case response times are calculated by the `MultiClusterScheduling` algorithm using the response time analysis presented in Section 4.7.

If the application is not schedulable, the term c_1 will be positive, and, in this case, the cost function is equal to c_1 . However, if the process set is schedulable, $c_1 = 0$ and we use c_2 as a cost function, as it is able to differentiate between two alternatives, both leading to a schedulable process set. For a given set of optimisation parameters leading to a schedulable process set, a smaller c_2 means that we have improved the worst-case response times of the processes, so the application can potentially be implemented on a cheaper hardware architecture (with slower processors and/or buses). Improving the degree of schedulability can also lead to an improvement in the quality of control for control applications.

4.6.1 Partitioning and mapping

By partitioning, we denote the decision whether a certain process should be assigned to the TT or the ET domain (and, implicitly, to a TTC or an ETC, respectively). Mapping a process means assigning it to a particular node inside a cluster.

Very often, the partitioning decision is taken based on the experience and preferences of the designer, considering aspects such as the functionality implemented by the process, the hardness of the constraints,

sensitivity to jitter, legacy constraints, etc. Let \mathcal{P} be the set of processes in the application Γ . We denote with $\mathcal{P}_T \subseteq \mathcal{P}$ the subset of processes which the designer has assigned to the TT cluster, while $\mathcal{P}_E \subseteq \mathcal{P}$ contains processes which are assigned to the ET cluster.

Many processes, however, do not exhibit certain particular features or requirements which obviously lead to their implementation as TT or ET activities. The subset $\mathcal{P}^+ = \mathcal{P} \setminus (\mathcal{P}_T \cup \mathcal{P}_E)$ of processes could be assigned to any of the TT or ET domains. Decisions concerning the partitioning of this set of activities can lead to various trade-offs concerning, for example, the schedulability properties of the system, the amount of communication exchanged through the gateway, the size of the schedule tables, etc.

For part of the partitioned processes, the designer might have already decided their mapping. For example, certain processes, due to constraints such as having to be close to sensors/actuators, have to be physically located in a particular hardware unit. They represent the sets $\mathcal{P}_T^M \subseteq \mathcal{P}_T$ and $\mathcal{P}_E^M \subseteq \mathcal{P}_E$ of already mapped TT and ET processes, respectively. Consequently, we denote with $\mathcal{P}_T^* = \mathcal{P}_T \setminus \mathcal{P}_T^M$ the TT processes for which the mapping has not yet been decided, and similarly, with $\mathcal{P}_E^* = \mathcal{P}_E \setminus \mathcal{P}_E^M$ the unmapped ET processes. The set $\mathcal{P}^* = \mathcal{P}_T^* \cup \mathcal{P}_E^* \cup \mathcal{P}^+$ then represents all the unmapped processes in the application.

The mapping of messages is decided implicitly by the mapping of processes. Thus, a message exchanged between two processes on the TTC (ETC) will be mapped on the TTP bus (CAN bus) if these processes are allocated to different nodes. If the communication takes place between two clusters, two message instances will be created, one mapped on the TTP bus and one on the CAN bus. The first message is sent from the sender node to the gateway, while the second message is sent from the gateway to the receiving node.

Using the notation introduced, the partitioning and mapping problem can be described more exactly as follows. As an input we have an application Γ given as a set of process graphs and a two-cluster system consisting of a TT and an ET cluster. As introduced previously, \mathcal{P}_T and \mathcal{P}_E are the sets of processes already partitioned into TT and ET, respectively. Also, $\mathcal{P}_T^M \subseteq \mathcal{P}_T$ and $\mathcal{P}_E^M \subseteq \mathcal{P}_E$ are the sets of already mapped TT and ET processes. We are interested to find a partitioning for processes in $\mathcal{P}^+ = \mathcal{P} \setminus (\mathcal{P}_T \cup \mathcal{P}_E)$ and decide a mapping for processes in $\mathcal{P}^* = \mathcal{P}_T^* \cup \mathcal{P}_E^* \cup \mathcal{P}^+$, where $\mathcal{P}_T^* = \mathcal{P}_T \setminus \mathcal{P}_T^M$, and $\mathcal{P}_E^* = \mathcal{P}_E \setminus \mathcal{P}_E^M$ such that imposed deadlines are guaranteed to be satisfied. We have highlighted a possible solution to this problem [75].

4.6.2 Frame packing

In both the TTP and CAN protocols messages are not sent independently, but several messages having similar timing properties are usually packed into frames. In many application areas, such as automotive electronics, messages range from one single bit (e.g. the state of a device) to a couple of bytes (e.g. vehicle speed, etc.). Transmitting such small messages one per frame would create a high communication overhead, which can cause long delays leading to an unschedulable system. For example, 65 bits have to be transmitted on CAN for delivering one single bit of application data. Moreover, a given frame configuration defines the exact behaviour of a node on the network, which is very important when integrating nodes from different suppliers.

Let us consider the motivational example in Figure 4.7, where we have the process graph from Figure 4.7(d) mapped on the two-cluster system from Figure 4.7(e): P_1 and P_4 are mapped on node N_1 from the TTC, while P_2 and P_3 are mapped on N_2 from ETC. The data field of the frames is represented with a black rectangle, while the other frame fields are depicted with a grey colour. We consider a physical implementation of the buses such that the frames will take the time indicated in the figure by the length of their rectangles. We are interested to find a frame configuration such that the application is schedulable.

In the system configuration of Figure 4.7(a) we consider that, on the TTP bus, the node N_1 transmits in the first slot (S_1) of the TDMA round, while the gateway transmits in the second slot (S_G). Process P_3 has a higher priority than process P_2 , hence P_2 will be interrupted by P_3 when it receives message m_2 . In such a setting, P_4 will miss its deadline, which is depicted as a thick vertical line in Figure 4.7. Changing the frame configuration as in Figure 4.7(b), so that messages m_1 and m_2 are packed into frame f_1 and slot S_G of the gateway comes first, processes P_2 and P_3 will receive m_1 and m_2 sooner and thus reduce the worst-case response time of the process graph, which is still larger than the deadline. In Figure 4.7(c), we also pack m_3 and m_4 into f_2 . In such a situation, the sending of m_3 will have to be delayed until m_4 is queued by P_2 . Nevertheless, the worst-case response time of the application is further reduced, which means that the deadline is met, thus the system is schedulable.

However, packing more messages will not necessarily reduce the worst-case response times further, as it might increase too much the worst-case response times of messages that have to wait for the frame to be assembled, this is the case with message m_3 in Figure 4.7(c).

This design optimisation problem can be formulated more exactly as follows. As input to the frame-packing problem we have an application Γ given as a set of process graphs mapped on an architecture consisting

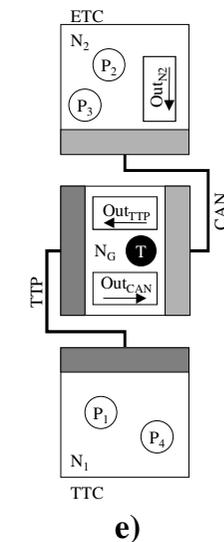
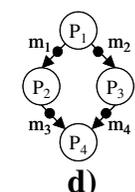
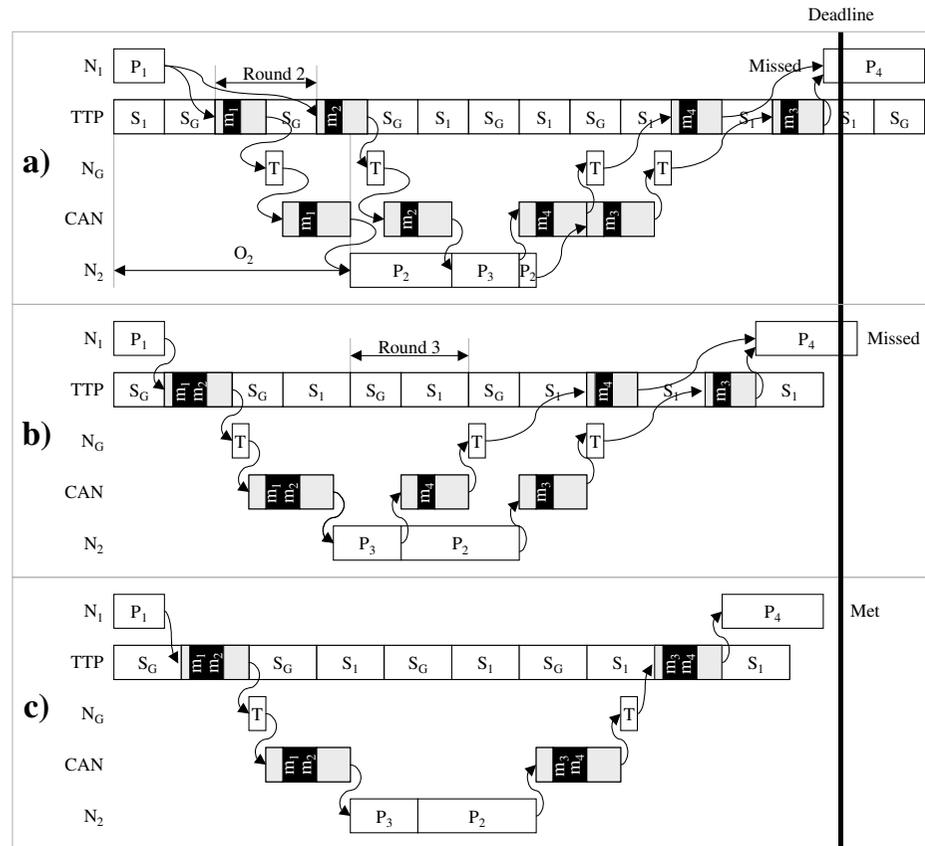


Figure 4.7: Frame packing optimisation example

of a TTC and an ETC interconnected through a gateway. We consider that the partitioning and mapping of processes has been already decided.

We are interested to find a mapping of messages to frames (a frame packing configuration) denoted by a 4-tuple $\psi = \langle \alpha, \pi, \beta, \sigma \rangle$ such that the application Γ is schedulable. Once a schedulable system is found, we are interested to further improve the ‘degree of schedulability’ so the application can potentially be implemented on a cheaper hardware architecture (with slower buses and processors).

Determining a frame configuration ψ means deciding on:

- The mapping of application messages transmitted on the ETC to frames (the set of ETC frames α), and their relative priorities, π . Note that the ETC frames α have to include messages transmitted from an ETC node to a TTC node, messages transmitted inside the ETC cluster, and those messages transmitted from the TTC to the ETC.
- The mapping of messages transmitted on the TTC to frames, denoted by the set of TTC frames β , and the sequence σ of slots in a TDMA round. The slot sizes are determined based on the set β , and are calculated such that they can accommodate the largest frame sent in that particular slot. We consider that messages transmitted from the ETC to the TTC are not statically allocated to frames. Rather, we will dynamically pack messages originating from the ETC into the ‘gateway frame’, for which we have to decide the data field length (see Section 4.5.2).

Several details related to the schedulability analysis were omitted from the discussion of the example. These details will be discussed in the next section.

4.7 Multi-cluster analysis and scheduling

Once a partitioning and a mapping is decided, and a frame packing configuration is fixed, the processes and messages have to be scheduled. For the TTC this means building the schedule tables, while for the ETC the priorities of the ET processes have to be determined and their schedulability has to be analysed.

The analysis presented in this section works under the following assumptions:

- All the processes belonging to a process graph G have the same period T_G . However, process graphs can have different periods.
- The offsets are *static* (as opposed to *dynamic* [42]), and are smaller than the period.

- The deadlines are arbitrary, i.e. can be larger than the period.

The basic idea is that on the TTC an application is schedulable if it is possible to build a schedule table such that the timing requirements are satisfied.

On the ETC, the answer whether or not a system is schedulable is given by a ‘schedulability analysis’. Thus, for the ETC we use a ‘response time analysis’, where the schedulability test consists of the comparison between the worst-case response time r_i of a process P_i and its deadline D_i . Response time analysis of data dependent processes with static priority pre-emptive scheduling has been proposed in [39,40,42], and has been also extended to consider the CAN protocol [20]. The authors use the concept of ‘offset’ in order to handle data dependencies. Thus, each process P_i is characterised by an offset O_i , measured from the start of the process graph, that indicates the earliest possible start time of P_i . Such an offset is, for example, O_2 in Figure 4.7(a), as process P_2 cannot start before receiving m_1 . The same is true for messages, their offset indicating the earliest possible transmission time. The response time analysis employed is presented in Section 4.7.1.

However, determining the schedulability of an application mapped on a multi-cluster system cannot be addressed separately for each type of cluster, since the inter-cluster communication creates a circular dependency: the static schedules determined for the TTC influence through their offsets the worst-case response times of the processes on the ETC, which in turn influence the schedule table construction on the TTC. In Figure 4.7(b) packing m_1 and m_2 in the same frame leads to equal offsets for P_2 and P_3 . Because of this, P_3 will delay P_2 (which would not be the case if m_2 sent to P_3 would be scheduled in round 3, e.g.) and thus the placement of P_4 in the schedule table has to be accordingly delayed to guarantee the arrivals of m_3 and m_4 .

In our analysis we consider the influence between the two clusters by making the following observations:

- The start time of process P_i in a schedule table on the TTC is its offset O_i .
- The worst-case response time r_i of a TT process is its worst-case execution time, i.e. $r_i = C_i$ (TT processes are not pre-emptable).
- The worst-case response times of the messages exchanged between two clusters have to be calculated according to the schedulability analysis described in Section 4.7.1.
- The offsets have to be set by a scheduling algorithm such that the precedence relationships are preserved. This means that, if process P_B depends on process P_A , the following condition must

```

MultiClusterScheduling( $\Gamma, \mathcal{M}, \Psi$ )
-- determines the set of offsets  $\phi$  and worst-case response times  $\rho$ 
1  for each  $O_i \in \phi$  do  $O_i = 0$  end for -- initially all offsets are zero
2  -- determine initial values for the worst-case response times
3  -- according to the analysis in Section 4.7.1
4   $\rho = \text{ResponseTimeAnalysis}(\Gamma, \mathcal{M}, \Psi, \phi)$ 
5  -- determine new values for the offsets, based on the response times  $\rho$ 
6   $\phi^{new} = \text{ListScheduling}(\Gamma, \mathcal{M}, \Psi, \rho)$ 
7   $\theta_\Gamma = \infty$  -- consider the system unschedulable at first
8  repeat -- iteratively improve the degree of schedulability  $\delta_\Gamma$ 
9    for each  $O_i^{new} \in \phi^{new}$  do -- for each newly calculated offset
10      $O_i^{old} = \phi.O_i; \phi.O_i = \phi^{new} O_i^{new}$  -- set the new offset, remember old
11      $\rho^{new} = \text{ResponseTimeAnalysis}(\Gamma, \mathcal{M}, \Psi, \phi^{new})$ 
12      $\delta_\Gamma^{new} = \text{SchedulabilityDegree}(\Gamma, \rho^{new})$ 
13     if  $\delta_\Gamma^{new} < \delta_\Gamma$  then -- the schedulability has improved
14       -- offsets are recalculated using  $\rho^{new}$ 
15        $\phi^{new} = \text{ListScheduling}(\Gamma, \mathcal{M}, \Psi, \rho^{new})$ 
16       break -- exit the for-each loop
17     else -- the schedulability has not improved
18        $\phi.O_i = O_i^{old}$  -- restore the old offset
19   end for
20 until  $\theta_\Gamma$  has not changed
21 return  $\rho, \phi, \delta_\Gamma$ 
end MultiClusterScheduling

```

Figure 4.8: The MulticlusterScheduling algorithm

hold: $O_B \geq O_A + r_A$. Note that for the processes on a TTC which receive messages from the ETC this translates to setting the start times of the processes such that a process is not activated before the worst-case arrival time of the message from the ETC. In general, offsets on the TTC are set such that all the necessary messages are present at the process invocation.

The `MultiClusterScheduling` algorithm in Figure 4.8 receives as input the application Γ , the frame configuration ψ , and produces the offsets ϕ and worst-case response times ρ .

The algorithm sets initially all the offsets to 0 (line 1). Then, the worst-case response times are calculated using the `ResponseTimeAnalysis` function (line 4) using the analysis presented in Section 4.7.1. The fixed-point iterations that calculate the response times at line 3 will converge if processor and bus loads are smaller than 100 per cent [39]. Based on these worst-case response times, we determine new values ϕ^{new} for the offsets using a list scheduling algorithm (line 6). We now have a schedule table for the TTC and worst-case response times for the ETC,

which are pessimistic. The following loop will reduce the pessimism of the worst-case response times.

The multi-cluster scheduling algorithm loops until the degree of schedulability δ_Γ of the application Γ cannot be further reduced (lines 8–20). In each loop iteration, we select a new offset from the set of ϕ^{new} offsets (line 10), and run the response time analysis (line 11) to see if the degree of schedulability has improved (line 12). If δ_Γ has not improved, we continue with the next offset in ϕ^{new} .

When a new offset O_i^{new} leads to an improved δ_Γ we exit the for-each loop 9–19 that examines offsets from ϕ^{new} . The loop iteration 8–20 continues with a new set of offsets, determined by `ListScheduling` at line 15, based on the worst-case response times ρ^{new} corresponding to the previously accepted offset.

In the multi-cluster scheduling algorithm, the calculation of offsets is performed by the list scheduling algorithm presented in Figure 4.9. In each iteration, the algorithm visits the processes and messages in the `ReadyList`. A process or a message in the application is placed in the `ReadyList` if all its predecessors have been already scheduled. The list is ordered based on the priorities [76]. The algorithm terminates when all processes and messages have been visited.

In each loop iteration, the algorithm calculates the earliest time moment (*offset*) when the process or message $node_i$ can start (lines 5–7). There are four situations:

1. The visited node is an ET message. The message m_i is packed into its frame f (line 9), and the offset O_f of the frame is updated. The frame can only be transmitted after all the sender processes that pack messages in this frame have finished executing. The offset of message m_i packed to frame f is equal to the frame offset O_f .
2. The node is a TT message. In this case, when the frame is ready for transmission, it is scheduled using the `ScheduleTTFrame` function (presented in Figure 4.10), which returns the *round* and the *slot* where the frame has been placed (line 16 in Figure 4.9). In Figure 4.10, the round immediately following *offset* is the initial candidate to be considered (line 2). However, it can be too late to catch the allocated slot, in which case the next round is considered (line 4). For this candidate round, we have to check if the slot is not occupied by another frame. If so, the communication has to be delayed for another round (line 7). Once a frame has been scheduled, we can determine the offsets and worst-case response times (Figure 4.9, line 18). For all the messages in the frame the offset is equal to the start of the slot in the TDMA round, and the worst-case response time is the slot length.

```

ListScheduling( $\Gamma, \mathcal{M}, \Psi, \rho$ ) -- determines the set of offsets  $\phi$ 
1  ReadyList = source nodes of all process graphs in the application
2  while ReadyList  $\neq \emptyset$  do
3      nodei = Head(ReadyList)
4      offset = 0 -- determine the earliest time when an activity can start
5      for each direct predecessor nodej of nodei do
6          offset = max(offset,  $O_j + r_j$ )
7      end for
8      if nodei is a message mi then
9          PackFrame(mi, f) -- pack each ready message m into its frame f
10          $O_f = \max(O_f, \textit{offset})$  -- update the frame offset
11         if f is complete then -- the frame is complete for transmission
12             if  $f \in \alpha$  then -- f is an ET frame
13                 -- the offset of messages is equal to the frame offset
14                 for each  $m_j \in f$  do  $O_j = O_f$  end for
15             else -- f is a TT frame
16                  $\langle \textit{round}, \textit{slot} \rangle = \text{ScheduleTTFrame}(f, \textit{offset}, \Psi)$ 
17                 -- set the TT message offsets based on the round and slot
18                 for each  $m_j \in f$  do  $O_j = \textit{round} * T_{TDMA} + O_{\textit{slot}}$  end for
19             endif; endif
20 else -- nodei is a process Pi
21     if  $\mathcal{M}(P_i) \in \mathcal{N}_E$  then -- if process Pi is mapped on the ETC
22          $O_i = \textit{offset}$  -- the ETC process can start immediately
23     else -- process Pi is mapped on the TTC
24         -- Pi has to wait also for the processor  $\mathcal{M}(P_i)$  to become available
25          $O_i = \max(\textit{offset}, \text{ProcessorAvailable}(\mathcal{M}(P_i)))$ 
26     end if; end if;
27     Update(ReadyList)
28 end while
29 return offsets
end ListScheduling

```

Figure 4.9: ListScheduling algorithm

```

ScheduleTTFrame ( $f, offset, \psi$ )
-- returns the slot and the round assigned to frame  $f$ 
1   $slot =$  the slot assigned to the node sending  $f$  -- the frame slot
2   $round = offset / T_{TDMA}$  -- the first round which could be a candidate
3  if  $offset \cdot round * T_{TDMA} > O_{slot}$  then -- the  $slot$  is missed
4       $round = round + 1$  -- if yes, take the next round
5  end if
6  while  $slot$  is occupied do
7       $round = round + 1$ 
8  end while
9  return  $round, slot$ 
end ScheduleTTFrame

```

Figure 4.10: Frame scheduling on the TTC

3. The algorithm visits a process P_i mapped on an ETC node. A process on the ETC can start as soon as its predecessors have finished and its inputs have arrived, hence $O_i = offset$ (line 22). However, P_i might experience, later on, interference from higher priority processes.
4. Process P_i is mapped on a TTC node. In this case, besides waiting for the predecessors to finish executing, P_i will also have to wait for its processor $\mathcal{M}(P_i)$ to become available (line 25). The earliest time when the processor is available is returned by the `ProcessorAvailable` function.

Let us now turn the attention back to the multi-cluster scheduling algorithm in Figure 4.8. The algorithm stops when the δ_Γ of the application Γ is no longer improved, or when a limit imposed on the number of iterations has been reached. Since in a loop iteration we do not accept a solution with a larger δ_Γ , the algorithm will terminate when in a loop iteration we are no longer able to improve δ_Γ by modifying the offsets.

4.7.1 Schedulability analysis for the ETC

For the ETC we use a response time analysis. A ‘response time analysis’ has two steps. In the first step, the analysis derives the worst-case response time of each process (the time it takes from the moment is ready for execution, until it has finished executing). The second step compares the worst-case response time of each process to its deadline and, if the response times are smaller or equal to the deadlines, the system is schedulable. The analysis presented in this section is used in the `ResponseTimeAnalysis` function (line 4 of the algorithm in Figure 4.8).

Thus, the response time analysis [77] uses the following equation for determining the worst-case response time r_i of a process P_i :

$$r_i = C_i + \sum_{\forall P_j \in hp(P_i)} \left\lceil \frac{r_i}{T_j} \right\rceil C_j \quad (4.2)$$

where C_i is the worst-case execution time of process P_i , T_j is the period of process P_j and $hp(P_i)$ denotes the set of processes that have a priority higher than the priority of P_i .

The summation term, representing the interference I_i of higher priority processes on P_i , increases monotonically in r_i , thus solutions can be found using a recurrence relation. Moreover, the recurrence relations that calculate the worst-case response time are guaranteed to converge if the processor utilisation is under 100 per cent.

The previously presented analysis assumes that the deadline of a process is smaller or equal to its period. This assumption has later been relaxed [32] to consider ‘arbitrary deadlines’ (i.e. deadlines can be larger than the period). Thus, the worst-case response time r_i of a process P_i becomes:

$$r_i = \max_{q=0, 1, 2, \dots} (J_i + w_i(q) - qT_i) \quad (4.3)$$

where J_i is the jitter of process P_i (the worst-case delay between the arrival of a process and the start of its execution), q is the number of busy periods being examined and $w_i(q)$ is the width of the level- i busy period starting at time qT_i . The level- i busy period is defined as the maximum time a processor executes processes of priority greater than or equal to the priority of process P_i , and is calculated as [32]:

$$w_i(q) = (q + 1)C_i + B_i + \sum_{\forall P_j \in hp(P_i)} \left\lceil \frac{w_i(q) + J_j}{T_j} \right\rceil C_j \quad (4.4)$$

The pessimism of the previous analysis can be reduced by using the information related to the precedence relations between processes. The basic idea is to exclude certain worst-case scenarios, from the critical instant analysis, which are impossible due to precedence constraints.

Methods for schedulability analysis of data dependent processes with static priority pre-emptive scheduling have been proposed [39,40,41,42]. They use the concept of ‘offset’ (or ‘phase’), in order to handle data dependencies. Tindell [39] shows that the pessimism of the analysis is reduced through the introduction of offsets. The offsets have to be determined by the designer.

In their analysis [39], the response time of a process P_i is:

$$r_i = \max_{q=0, 1, 2, \dots} \left(\max_{\forall P_j \in G} \left(w_i(q) + O_j + J_j - T_G \right) \left(q + \left\lceil \frac{O_j + J_j - O_i - J_i}{T_G} \right\rceil \right) - O_i \right) \quad (4.5)$$

where T_G the period of the process graph G , O_i and O_j are offsets of processes P_i and P_j , respectively, and J_i and J_j are the release jitters of P_i and P_j . In Equation (3.5), the level- i busy period starting at time qT_G is

$$w_i(q) = (q + 1)C - i + B_i + I_i \quad (4.6)$$

In the previous equation, the blocking term B_i represents interference from lower priority processes that are in their critical section and cannot be interrupted, and C_i represents the worst-case execution time of process P_i . The last term captures the interference I_i from higher priority processes in the application, including higher priority processes from other process graphs. Tindell [39] presents the details of the interference calculation.

Although this analysis is exact (both necessary and sufficient), it is computationally infeasible to evaluate. Hence, Tindell [39] proposes a feasible but not exact analysis (sufficient but not necessary) for solving Equation (3.5). Our implementations use the feasible analysis provided in Tindell [39] for deriving the worst-case response time of a process P_i .

We are now interested to determine the worst-case response time of frames and the worst-case queuing delays experienced by a frame in a communication controller.

Regarding the worst-case response time of messages, we have extended the CAN analysis from messages [20] and applied it in the context of frames on the CAN bus:

$$r_f = \max_{q=0, 1, 2, \dots} (J_f + W_f(q) + (1 + q)C_f) \quad (4.7)$$

In the previous equation J_f is the jitter of frame f which in the worst case is equal to the largest worst-case response time $r_{S(m)}$ of a sender process $S(m)$ which sends message m packed into frame f :

$$J_f = \max_{\forall m \in f} (r_{S(m)}) \quad (4.8)$$

In Equation (3.7), W_f is the ‘worst-case queuing delay’ experienced by f at the communication controller, and is calculated as:

$$W_f(q) = w_f(q) - qT_f \quad (4.9)$$

where q is the number of busy periods being examined, and $w_f(q)$ is the width of the level- f busy period starting at time qT_f .

Moreover, in Equation (3.7), C_f is the worst-case time it takes for a frame f to reach the destination controller. On CAN, C_f depends on the frame configuration and the size of the data field, s_f , while on TTP it is equal to the slot size in which f is transmitted.

The worst-case response time of message m packed into a frame f can be determined by observing that $r_m = r_f$.

The worst-case queuing delay for a frame (W_f in Equation (3.7)) is calculated differently for each type of queue:

1. The output queue of an ETC node, in which case $W_f^{N_i}$ represents the worst-case time a frame f has to spend in the Out_{N_i} queue on ETC node N_i . An example of such a frame is the one containing message m_3 in Figure 4.7(a), which is sent by process P_2 from the ETC node N_2 to the gateway node N_G , and has to wait in the Out_{N_2} queue.
2. The TTP-to-CAN queue of the gateway node, in which case W_f^{CAN} is the worst-case time a frame f has to spend in the Out_{CAN} queue of node N_G . In Figure 4.7(a), the frame containing m_1 is sent from the TTC node N_1 to the ETC node N_2 , and has to wait in the Out_{CAN} queue of gateway node N_G before it is transmitted on the CAN bus.
3. The CAN-to-TTP queue of the gateway node, where W_f^{TTP} captures the time f has to spend in the Out_{TTP} queue node N_G . Such a situation is present in Figure 4.7(a), where the frame with m_3 is sent from the ETC node N_2 to the TTC node N_1 through the gateway node N_G where it has to wait in the Out_{TTP} queue before it is transmitted on the TTP bus, in the S_G slot of node N_G .

On the TTC, the synchronisation between processes and the TDMA bus configuration is solved through the proper synthesis of schedule tables, hence no output queues are needed. The frames sent from a TTC node to another TTC node are taken into account when determining the offsets, and are not involved directly in the ETC analysis.

The next sections show how the worst queuing delays are calculated for each of the previous three cases.

Worst-case queuing delays in the Out_{N_i} and Out_{CAN} queues

The analyses for $W_f^{N_i}$ and W_f^{CAN} are similar. Once f is the highest priority frame in the Out_{CAN} queue, it will be sent by the gateway's CAN controller as a regular CAN frame, therefore the same equation

for w_f can be used:

$$w_f(q) = B_f + \sum_{\forall f_j \in hp(f)} \left\lceil \frac{w_f(q) + J_j}{T_j} \right\rceil C_j \quad (4.10)$$

The intuition is that f has to wait, in the worst case, first for the largest lower priority frame that is just being transmitted (B_f) as well as for the higher priority $f_j \in hp(f)$ frames that have to be transmitted ahead of f (the second term). In the worst case, the time it takes for the largest lower priority frame $f_k \in p(f)$ to be transmitted to its destination is:

$$B_f = \max_{\forall f_k \in lp(f)} (C_k) \quad (4.11)$$

Note that in our case, $lp(f)$ and $hp(f)$ also include messages produced by the gateway node, transferred from the TTC to the ETC.

Worst-case queuing delay in the Out_{TTP} queue

The time a frame f has to spend in the Out_{TTP} queue in the worst case depends on the total size of messages queued ahead of f (Out_{TTP} is a FIFO queue), the size S_G of the data field of the frame fitting into the gateway slot responsible for carrying the CAN messages on the TTP bus, and the period T_{TDMA} with which this slot S_G is circulating on the bus [46]:

$$w_f^{\text{TTP}}(q) = B_f + \left\lceil \frac{(q+1)s_f + I_f(w_f(q))}{S_G} \right\rceil T_{\text{TDMA}} \quad (4.12)$$

where I_f is the total size of the frames queued ahead of f . Those frames $f_j \in hp(f)$ are ahead of f , which have been sent from the ETC to the TTC, and have higher priority than f :

$$I_f(w) = \sum_{\forall f_j \in hp(f)} \left\lceil \frac{w_f + J_j}{T_j} \right\rceil s_j \quad (4.13)$$

where the frame jitter J_j is given by Equation (3.8).

The blocking term B_f is the time interval in which f cannot be transmitted because the slot S_G of the TDMA round has not arrived yet. In the worst case (i.e. the frame f has just missed the slot S_G), the frame has to wait an entire round T_{TDMA} for the slot S_G in the next TDMA round.

```

MultiClusterConfiguration( $\Gamma$ )
1  -- determine an initial partitioning and mapping  $\mathcal{M}$ ,
2  -- and an initial frame configuration  $\psi^0$ 
3   $\langle \mathcal{M}, \psi^0 \rangle = \text{PartitioningAndMapping}(\Gamma)$ 
4  -- the frame packing optimization algorithm
5   $\psi = \text{FramePackingOptimization}(\Gamma, \mathcal{M}, \psi^0)$ 
6  -- test if the resulted configuration leads to a schedulable application
7  if  $\text{MultiClusterScheduling}(\Gamma, \mathcal{M}, \psi)$  returns schedulable then
8      return  $\mathcal{M}, \psi$ 
9  else
10     return unschedulable
11 endif
end MultiClusterConfiguration

```

Figure 4.11: The general frame packing strategy

4.8 Frame-packing optimisation strategy

The general multi-cluster optimisation strategy is outlined in Figure 4.11. The `MultiClusterConfiguration` strategy has two steps:

1. In the first step, line 3, the application is partitioned on the TTC and ETC clusters, and processes are mapped to the nodes of the architecture using the `PartitioningAndMapping` function. The partitioning and mapping can be done with an optimisation heuristic [75]. As part of the partitioning and mapping process, an initial frame configuration $\psi^0 = \langle \alpha^0, \pi^0, \beta^0 \sigma^0 \rangle$ is derived. Messages exchanged by processes partitioned to the TTC will be mapped to TTC frames, while messages exchanged on the ETC will be mapped to ETC frames. For each message sent from a TTC process to an ETC process, we create an additional message on the ETC, and we map this message to an ETC frame. The sequence σ^0 of slots for the TTC is decided by assigning in order nodes to the slots ($S_i = N_i$). One message is assigned per frame in the initial set β^0 of TTC frames. For the ETC, the frames in the set α^0 initially hold each one single message, and we calculate the message priorities Π^0 based on the deadlines of the receiver processes.
2. The frame packing optimisation, is performed as the second step (line 5 in Figure 4.11). The `FramePackingOptimization` function receives as input the application Γ , the mapping \mathcal{M} of processes to resources and the initial frame configuration ψ^0 , and it produces as output the optimised frame packing configuration ψ . Such an optimisation problem is NP complete [78], thus obtaining the optimal

solution is not feasible. We present two frame packing optimisation strategies, one based on a simulated annealing approach, presented in Section 4.8.1, while the other, outlined in Section 4.8.2, is based on a greedy heuristic that uses intelligently the problem-specific knowledge in order to explore the design space.

If after these steps the application is unschedulable, we conclude that no satisfactory implementation could be found with the available amount of resources.

Testing if the application Γ is schedulable is done using the MultiClusterScheduling (MCS) algorithm (line 7 in Figure 4.11). The multi-cluster scheduling algorithm, presented in Figure 4.8, takes as input an application Γ , a mapping \mathcal{M} and an initial frame configuration ψ^0 , builds the TT schedule tables, sets the ET priorities for processes, and provides the global analysis.

4.8.1 Frame packing with simulated annealing

The first algorithm we have developed is based on a simulated annealing (SA) strategy [78], and is presented in Figure 4.12. The algorithm takes as input the application Γ , a mapping \mathcal{M} and an initial frame configuration ψ^0 , and determines the frame configuration ψ which leads to the best degree of schedulability δ_Γ (the smaller the value, the more schedulable the system, see Section 4.6).

Determining a frame configuration ψ means finding the set of ETC frames α and their relative priorities π , and the set of TTC frames B , including the sequence σ of slots in a TDMA round.

The main feature of a SA strategy is that it tries to escape from a local optimum by randomly selecting a new solution from the neighbours of the current solution. The new solution is accepted if it is an improved solution (lines 9–10 of the algorithm in Figure 4.12). However, a worse solution can also be accepted with a certain probability that depends on the deterioration of the cost function and on a control parameter called temperature (lines 12–13).

In Figure 4.12 we give a short description of this algorithm. An essential component of the algorithm is the generation of a new solution ψ_{new} starting from the current one ψ_{current} . The neighbours of the current solution ψ_{current} are obtained by performing transformations (called moves) on the current frame configuration ψ_{current} (line 8). We consider the following moves:

- moving a message m from a frame f_1 to another frame f_2 (or moving m into a separate single-message frame);
- swapping the priorities of two frames in α ;

```

SimulatedAnnealing( $\Gamma, \mathcal{M}, \psi^0$ )
1  -- given an application  $\Gamma$  finds out if it is schedulable and produces
2  -- the configuration  $\langle \Psi, \pi, \beta, \sigma \rangle$  leading to the smallest  $\delta_\Gamma$ 
3  -- initial frame configuration
4   $\psi_{current} = \psi^0$ 
5  temperature = initial temperature TI
6  repeat
7    for  $i = 1$  to temperature length TL do
8      generate randomly a neighboring solution  $\psi_{new}$  of  $\psi_{current}$ 
9       $\delta = \text{MultiClusterScheduling}(\Gamma, \mathcal{M}, \psi_{new}) -$ 
         $\text{MultiClusterScheduling}(\Gamma, \mathcal{M}, \psi_{current})$ 
10     if  $\delta < 0$  then  $\psi_{current} = \psi_{new}$ 
11     else
12       generate  $q = \text{Random}(0, 1)$ 
13       if  $q < e^{-\delta/\text{temperature}}$  then  $\psi_{current} = \psi_{new}$  end if
14     end if
15   end for
16   temperature =  $\varepsilon * \text{temperature}$ 
17 until stopping criterion is met
18 return  $\text{SchedulabilityTest}(\Gamma, \mathcal{M}, \psi_{best})$ , solution  $\psi$  best
    corresponding to the best degree of schedulability  $\delta_\Gamma$ 
end SimulatedAnnealing

```

Figure 4.12: The SimulatedAnnealing algorithm

- swapping two slots in the sequence σ of slots in a TDMA round.

For the implementation of this algorithm, the parameters *TI* (initial temperature), *TL* (temperature length), ε (cooling ratio) and the stopping criterion have to be determined. They define the ‘cooling schedule’ and have a decisive impact on the quality of the solutions and the CPU time consumed. We are interested to obtain values for *TI*, *TL* and ε that will guarantee the finding of good quality solutions in a short time.

We performed long runs of up to 48 h with the SA algorithm, for ten synthetic process graphs (two for each graph dimension of 80, 160, 240, 320, 400, see Section 4.9) and the best ever solution produced has been considered as the optimum. Based on further experiments we have determined the parameters of the SA algorithm so that the optimisation time is reduced as much as possible but the near-optimal result is still produced. For example, for the graphs with 320 nodes, *TI* is 700, *TL* is 500 and ε is 0.98. The algorithm stops if for three consecutive temperatures no new solution has been accepted.

4.8.2 Frame packing greedy heuristic

The `OptimizeFramePacking` greedy heuristic (Figure 4.13) constructs the solution by progressively selecting the best candidate in terms of the degree of schedulability.

We start by observing that all activities taking place in a multi-cluster system are ordered in time using the offset information, determined in the `StaticScheduling` function based on the worst-case response times known so far and the application structure (i.e. the dependencies in the process graph). Thus, our greedy heuristic outlined in Figure 4.13, starts with building two lists of messages ordered according to the ascending value of their offsets, one for the TTC, $messages_\beta$, and one for ETC, $messages_\alpha$. Our heuristic is to consider for packing in the same frame messages which are adjacent in the ordered lists. For example, let us consider that we have three messages, m_1 of 1 byte, m_2 of 2 bytes and m_3 of 3 bytes, and that messages are ordered as m_3, m_1, m_2 based on the offset information. Also, assume that our heuristic has suggested two frames, frame f_1 with a data field of 4 bytes, and f_2 with a data field of 2 bytes. The `PackMessages` function will start with m_3 and pack it in frame f_1 . It continues with m_2 , which is also packed into f_1 , since there is space left for it. Finally, m_1 is packed in f_2 , since there is no space left for it in f_1 .

The algorithm tries to determine, using the for-each loops in Figure 4.13, the best frame configuration. The algorithm starts from the initial frame configuration ψ^0 , and progressively determines the best change to the current configuration. The quality of a frame configuration is measured using the `MultiClusterScheduling` algorithm, which calculates the degree of schedulability δ_F (line 13). Once a configuration parameter has been fixed in the outer loops it is used by the inner loops:

- Lines 10–15: The innermost loops determine the best size S_α for the currently investigated frame f_α in the ETC frame configuration α_{current} . Thus, several frame sizes are tried (line 11), each with a size returned by `RecommendedSizes` to see if it improves the current configuration. The `RecommendedSizes(messages $_\alpha$)` list is built recognising that only messages adjacent in the $messages_\alpha$ list will be packed into the same frame. Sizes of frames are determined as a sum resulted from adding the sizes of combinations of adjacent messages, not exceeding 8 bytes. For the previous example, with m_1, m_2 and m_3 , of 1, 2 and 3 bytes, respectively, the frame sizes recommended will be of 1, 2, 3, 4, and 6 bytes. A size of 5 bytes will not be recommended since there are no adjacent messages that can be summed together to obtain 5 bytes of data.

- Lines 9–16: This loop determines the best frame configuration α . This means deciding on how many frames to include in α (line 9), and which are the best sizes for them. In α there can be any number of frames, from one single frame to n_α frames (in which case each frame carries one single message). Once a configuration α_{best} or the ETC, minimising δ_Γ , has been determined (saved in line 16), the algorithm looks for the frame configuration β which will further improve δ_Γ .
- Lines 7–17: The best size for a frame f_β is determined similarly to the size for a frame f_α .
- Lines 6–18: The best frame configuration β_{best} is determined. For each frame configuration β tried, the algorithm loops again through the innermost loops to see if there are better frame configurations α in the context of the current frame configuration β_{current} .
- Lines 4–19: After a β_{best} has been decided, the algorithm looks for a slot sequence σ starting with the first slot and tries to find the node which, when transmitting in this slot, will reduce δ_Γ . Different slot sequences are tried by swapping two slots within the TDMA round (line 5).

For the initial message priorities π^0 (initially, there is one message per frame) we use the ‘heuristic optimised priority assignment’ (HOPA) approach [55], where priorities in a distributed real-time system are determined, using knowledge of the factors that influence the timing behaviour, such that the degree of schedulability of the system is improved (line 1). The ETC message priorities set at the beginning of the algorithm are not changed by our greedy optimisation loops. The priority of a frame $f_\alpha \in \alpha$ is given by the message $m \in f_\alpha$ with the highest priority.

The algorithm continues in this fashion, recording the best ever ψ_{best} configurations obtained, in terms of δ_Γ , and thus the best solution ever is reported when the algorithm finishes.

4.9 Experimental results

For the evaluation of our frame-packing optimisation algorithms we first used process graphs generated for experimental purpose. We considered two-cluster architectures consisting of 2, 4, 6, 8 and 10 nodes, half on the TTC and the other half on the ETC, interconnected by a gateway. Forty processes were assigned to each node, resulting in applications of 80, 160, 240, 320 and 400 processes.

Table 4.1: Evaluation of the frame-packing optimisation algorithms

No. of processes	Straightforward solution (SP)			OptimizeFramePacking (OFP)			Simulated-Annealing (SA)
	average (%)	max (%)	time (s)	average (%)	max (%)	time (s)	time (s)
80	2.42	17.89	0.09	0.40	1.59	4.35	235.95
160	16.10	42.28	0.22	2.28	8.32	12.09	732.40
240	40.49	126.4	0.54	6.59	21.80	49.62	2928.53
320	70.79	153.08	0.74	13.70	30.51	172.82	7585.34
400	97.37	244.31	0.95	31.62	95.42	248.30	22099.68

We generated both graphs with random structure and graphs based on more regular structures such as trees and groups of chains. We generated a random structure graph deciding for each pair of two processes if they should be connected or not. Two processes in the graph were connected with a certain probability (between 0.05 and 0.15, depending on the graph dimension) on the condition that the dependency would not introduce a loop in the graph. The width of the tree-like structures was controlled by the maximum number of direct successors a process can have in the tree (from 2 to 6), while the graphs consisting of groups of chains had 2 to 12 parallel chains of processes. Furthermore, the regular structures were modified by adding a number of 3 to 30 random cross-connections.

The mapping of the applications to the architecture has been done using a simple heuristic that tries to balance the utilisation of processors while minimising communication. Execution times and message lengths were assigned randomly using both uniform and exponential distribution within the 10–100 ms and 1–2 bytes ranges, respectively. For the communication channels we considered a transmission speed of 256 kbps and a length below 20 meters. All experiments were run on a SUN Ultra 10.

The first result concerns the ability of our heuristics to produce schedulable solutions. We have compared the degree of schedulability δ_{Γ} obtained from our OptimizeFramePacking (OFP) heuristic (Figure 4.13) with the near-optimal values obtained by SA (Figure 4.12). Obtaining solutions that have a better degree of schedulability means obtaining tighter worst-case response times, increasing the chances of meeting the deadlines.

Table 4.1 presents the average percentage deviation of the degree of schedulability produced by OFP from the near-optimal values obtained with SA. Together with OFP, a straightforward approach (SF) is presented. The SF approach does not consider frame packing, and thus each message is transmitted independently in a frame. Moreover, for SF we considered a TTC bus configuration consisting of a straightforward ascending order of allocation of the nodes to the TDMA slots; the slot

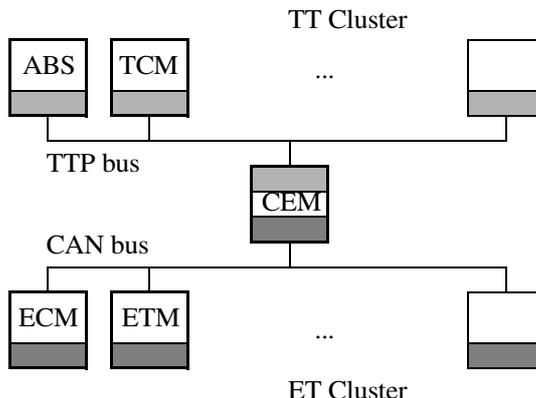


Figure 4.14: Hardware architecture for the cruise controller

lengths were selected to accommodate the largest message frame sent by the respective node, and the scheduling has been performed by the `MultiClusterScheduling` algorithm in Figure 4.8.

In Table 4.1 we have one row for each application dimension of 80–400 processes, and a header for each optimisation algorithm considered. For each of the SF and OFP algorithms we have three columns in the table. In the first column, we present the average percentage deviation of the algorithm from the results obtained by SA. The percentage deviation is calculated according to the formula:

$$\text{deviation} = \frac{\delta_{\Gamma}^{\text{approach}} - \delta_{\Gamma}^{\text{SA}}}{\delta_{\Gamma}^{\text{SA}}} 100 \quad (4.14)$$

The second column presents the maximum percentage deviation from the SA result, and the third column presents the average execution time of the algorithm, in seconds. For the SA algorithm we present only its average execution times.

Table 4.1 shows that when packing messages to frames, the degree of schedulability improves dramatically compared to the straightforward approach. The greedy heuristic `OptimizeFramePacking` performs well for all the graph dimensions, having, e.g., run-times which are on average under 50 for applications with 240 processes.

When deciding on which heuristic to use for design space exploration or system synthesis, an important issue is the execution time. On average, our optimisation heuristics needed a couple of minutes to produce results, while the simulated annealing approach had an execution time of up to 6 h.

4.9.1 The vehicle cruise controller

A typical safety-critical application with hard real-time constraints, is a vehicle cruise controller (CC). We have considered a CC system derived from a requirement specification provided by the industry. The CC delivers the following functionality: it maintains a constant speed for speeds over 35 km/h and under 200 km/h, offers an interface (buttons) to increase or decrease the reference speed and is able to resume its operation at the previous reference speed. The CC operation is suspended when the driver presses the brake pedal.

The specification assumes that the CC will operate in an environment consisting of two clusters. There are four nodes which functionally interact with the CC system: the Anti-lock Braking System (ABS), the Transmission Control Module (TCM), the Engine Control Module (ECM) and the Electronic Throttle Module (ETM) (see Figure 4.14).

It has been decided to map the functionality (processes) of the CC over these four nodes. The ECM and ETM nodes have an 8-bit Motorola M68HC11 family CPU with 128 kbytes of memory, while the ABS and TCM are equipped with a 16-bit Motorola M68HC12 CPU and 256 kbytes of memory. The 16-bit CPUs are twice as fast than the 8-bit ones. The transmission speed of the communication channel is 256 kbps and the frequency of the TTP controller was chosen to be 20 MHz.

We have modelled the specification of the CC system using a set of 32 processes and 17 [72] where the mapping of processes to the nodes is also given. The period was chosen 250 ms, equal to the deadline.

In this setting, the straightforward approach SF produced an end-to-end worst-case response time of 320 ms, greater than the deadline, while both the OFP and SA heuristics produced a schedulable system with a worst-case response time of 172 ms.

This shows that the optimisation heuristic proposed, driven by our schedulability analysis, is able to identify that frame packing configuration which increases the schedulability degree of an application, allowing the developers to reduce the implementation cost of a system.

4.10 Conclusions

Heterogeneous distributed real-time systems are used in several application areas to implement increasingly complex applications that have tight timing constraints. The heterogeneity is manifested not only at the hardware and communication protocol levels, but also at the level of the scheduling policies used. In order to reduce costs and use the available resources more efficiently, the applications are distributed across several networks.

We have introduced the current state-of-the-art analysis and optimisation techniques available for such systems, and addressed in more detail a special class of heterogeneous distributed real-time embedded systems called multi-cluster systems.

We have presented an analysis for multi-cluster systems and outlined several characteristic design problems, related to the partitioning and mapping of functionality and the optimisation of the access to the communication infrastructure. An approach to schedulability-driven frame packing for the synthesis of multi-cluster systems was presented as an example of solving such a design optimisation problem. We have developed two optimisation heuristics for frame configuration synthesis which are able to determine frame configurations that lead to a schedulable system. We have shown that by considering the frame packing problem, we are able to synthesise schedulable hard real-time systems and to potentially reduce the overall cost of the architecture.

The main message of the presented research is that efficient analysis and optimisation methods are needed and can be developed for the efficient implementation of applications distributed over interconnected heterogeneous networks.

Bibliography

- [1] KOPETZ, H.: ‘Real-time systems – design principles for distributed embedded applications’ (Kluwer Academic Publishers, 1997)
- [2] KOPETZ, H.: ‘Automotive electronics’. Proceedings of the 11th Euromicro conference on *Real-time systems*, 1999, pp. 132–140
- [3] HANSEN, P.: ‘The Hansen report on automotive electronics’
<http://www.hansenreport.com/>, July–August, 2002
- [4] LEEN, G., and HEFFERNAN, D.: ‘Expanding automotive electronic systems’, *Computer*, 2002, **35**(1), pp. 88–93
- [5] JOST, K.: ‘From fly-by-wire to drive-by-wire’.
Automotive Engineering International, 2001,
<http://www.sae.org/automeg/electronics09-2001>
- [6] CHIODO, M.: ‘Automotive electronics: a major application field for hardware-software co-design’ in ‘Hardware/software co-design’ (Kluwer Academic Publishers, 1996), pp. 295–310
- [7] X-by-Wire Consortium, *X-By-Wire: Safety related fault tolerant systems in vehicles*,
<http://www.vmars.tuwien.ac.at/projects/xbywire/>, 1998

- [8] KOPETZ, H.: ‘Automotive electronics – present state and future prospects’. Proceedings of the 25th international symposium on *Fault-tolerant computing*, 1995
- [9] Robert Bosch GmbH: CAN Specification, Version 2.0, <http://www.can.bosch.com/>, 1991
- [10] Local interconnect network protocol specification, <http://www.lin-subbus.org>, 2003
- [11] SAE Vehicle Network for Multiplexing and Data Communications Standards Committee, SAE J1850 Standard, 1994
- [12] RUSHBY, J.: ‘Bus architectures for safety-critical embedded systems’. *Lecture notes in computer science*, vol. 2211 (Springer Verlag, Heidelberg, 2001), pp. 306–323
- [13] HOYME, K., and DRISCOLL, K.: ‘SAFEbus’, *IEEE Aerospace and Electronic Systems Magazine*, 1992, **8**(3), pp. 34–39
- [14] MINER, P.S.: ‘Analysis of the SPIDER fault-tolerance protocols’. Proceedings of the 5th NASA *Langley formal methods workshop*, 2000
- [15] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: ‘Road vehicles – Controller area network (CAN) – Part 4: Time-triggered communication’. ISO/DIS 11898-4, 2002
- [16] Echelon: LonWorks: The LonTalk Protocol Specification, <http://www.echelon.com>, 2003
- [17] PROFIBUS INTERNATIONAL: *PROFIBUS DP Specification*, <http://www.profibus.com/>, 2003
- [18] BERWANGER, J., PELLER, M., and GRIESSBACH, R.: ‘A new high performance data bus system for safety-related applications’, <http://www.byteflight.de>, 2000
- [19] THE FLEXRAY GROUP: FlexRay Requirements Specification, Version 2.0.2, <http://www.flexray-group.com/>, 2002
- [20] TINDELL, K., BURNS, A., and WELLINGS, A.: ‘Calculating CAN message response times’, *Control Engineering Practice*, 1995, **3**(8), pp. 1163–1169
- [21] AUDSLEY, N., TINDELL, K., and BURNS, A.: ‘The end of line for static cyclic scheduling?’. Proceedings of the Euromicro Workshop on *Real-time systems*, 1993, pp. 36–41

- [22] XU, J., and PARNAS, D.L.: ‘On satisfying timing constraints in hard-real-time systems’, *IEEE Transactions on Software Engineering*, 1993, **19**(1), pp. 132–146
- [23] LÖNN, H., and AXELSSON, J.: ‘A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications’. Proceedings of the Euromicro conference on *Real-time systems*, 1999, pp. 142–149
- [24] EAST-EEA project, *ITEA Full Project Proposal*, <http://www.itea-office.org>, 2002
- [25] AUDSLEY, N., BURNS, A., DAVIS, R., TINDELL, K., and WELLINGS, A.: ‘Fixed priority preemptive scheduling: an historical perspective’, *Real-Time Systems*, 1995, **8**(2/3), pp. 173–198
- [26] BALARIN, F., LAVAGNO, L., MURTHY, P., and SANGIOVANNI-VINCENTELLI, A.: ‘Scheduling for embedded real-time systems’, *IEEE Design and Test of Computers*, January–March, 1998, **15**(1), 71–82
- [27] TimeWiz: <http://www.timesys.com>
- [28] RapidRMA: <http://www.tripac.com>
- [29] RTA-OSEK Planner, <http://www.livedevices.com>
- [30] Aires, <http://kabru.eecs.umich.edu/aires/>
- [31] LIU, C.L., and LAYLAND, J.W.: ‘Scheduling algorithms for multiprogramming in a hard-real-time environment’, *Journal of the ACM*, 1973, **20**(1), pp. 46–61
- [32] TINDELL, K., and CLARK, J.: ‘Holistic schedulability analysis for distributed hard real-time systems’, *Microprocessing & Microprogramming*, 1994, **50** (2–3), pp. 117–134
- [33] STANKOVIC, J.A., and RAMAMRITHAM, K.: ‘Advances in real-time systems’ (IEEE Computer Society Press, Washington, 1993)
- [34] XU, J., and PARNAS, D.L.: ‘Priority scheduling versus pre-runtime scheduling’, *Journal of Real Time Systems*, 2000, **18**(1), pp. 7–24
- [35] LEE, C., POTKONJAK, M., and WOLF, W.: ‘Synthesis of hard real-time application specific systems’, *Design Automation for Embedded Systems*, 1999, **4**(4), pp. 215–241

- [36] DAVE, B.P., and JHA, N.K.: ‘COHRA: hardware-software cosynthesis of hierarchical heterogeneous distributed systems’, *IEEE Transactions on CAD*, 1998 **17**(10), pp. 900–919
- [37] DAVE, B.P., LAKSHMINARAYANA, G., and JHA, N.J.: ‘COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems’, *IEEE Transactions on VLSI Systems*, 1999, **7**(1), pp. 92–104
- [38] BURNS, A., and WELLINGS, A.: ‘Real-time systems and programming languages’ (Addison Wesley, Reading, MA, 2001)
- [39] TINDELL, K.: ‘Adding time-offsets to schedulability analysis’. Department of Computer Science, University of York, Report No. YCS-94-221, 1994
- [40] YEN, T.Y., and WOLF, W.: ‘Hardware-software co-synthesis of distributed embedded systems’ (Kluwer Academic Publishers, 1997)
- [41] PALENCIA, J.C., and GONZÁLEZ HARBOUR, M.: ‘Exploiting precedence relations in the schedulability analysis of distributed real-time systems’, Proceedings of the 20th IEEE *Real-time systems* symposium, 1999, pp. 328–339
- [42] PALENCIA, J.C., and GONZÁLEZ HARBOUR, M.: ‘Schedulability analysis for tasks with static and dynamic offsets’. Proceedings of the 19th IEEE *Real-time systems* symposium, 1998, pp. 26–37
- [43] ERMEDAHL, H., HANSSON, H., and SJÖDIN, M.: ‘Response-time guarantees in ATM Networks’. Proceedings of the *IEEE Real-time systems* symposium, 1997, pp. 274–284
- [44] STROSNIDER, J.K., and MARCHOK, T.E.: ‘Responsive, deterministic IEEE 802.5 Token ring scheduling’, *Journal of Real-Time Systems*, 1989, **1**(2), pp. 133–158
- [45] AGRAWAL, G., CHEN, B., ZHAO, W., and DAVARI, S.: ‘Guaranteeing synchronous message deadlines with the token medium access control protocol’. *IEEE Transactions on Computers*, 1994, **43**(3), pp. 327–339
- [46] POP, P., ELES, P., and PENG, Z.: ‘Schedulability-driven communication synthesis for time-triggered embedded systems’, *Real-Time Systems Journal*, 2004, **26**(3), pp. 297–325
- [47] POP, T., ELES, P., and PENG, Z.: ‘Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems’.

- Proceedings of the international symposium on *Hardware/software codesign*, 2002, pp. 187–192
- [48] POP, T., ELES, P., and PENG, Z.: ‘Schedulability analysis and optimization for the synthesis of multi-cluster distributed embedded systems’. Proceedings of *Design automation and test in Europe* conference, 2003, pp. 184–189
- [49] RICHTER, K., JERSAK, M., and ERNST, R.: ‘A formal approach to MpSoC performance verification’, *Computer*, 2003, **36**(4), pp. 60–67
- [50] STATEMATE: <http://www.ilogix.com>
- [51] MATLAB/SIMULINK: <http://www.mathworks.com>
- [52] ASCET/SD: http://en.etasgroup.com/products/ascet_sd/
- [53] SYSTEMBUILD/MATRIX: <http://www.ni.com/matrixx>
- [54] TTP-PLAN: <http://www.tttech.com/>
- [55] GUTIÉRREZ GARCÍA, J.J. and GONZÁLEZ HARBOUR, M.: ‘Optimized priority assignment for tasks and messages in distributed Hard real-time systems’. Proceedings of the workshop on *Parallel and distributed real-Time systems*, 1995, pp. 124–132
- [56] JONSSON, J., and SHIN, K.G.: ‘Robust adaptive metrics for deadline assignment in distributed hard real-time systems’, *Real-Time Systems: The International Journal of Time-Critical Computing Systems*, 2002, **23**(3), pp. 239–271
- [57] VOLCANO NETWORK ANALYZER:
<http://www.volcanoautomotive.com/>
- [58] RAJNAK, A., TINDELL, K., and CASPARSSON, L.: ‘Volcano Communications Concept’ (Volcano Communication Technologies AB, 1998)
- [59] KIENHUIS, B., DEPRETTERE, E., VISSERS, K., and VAN DER WOLF, P.: ‘An approach for quantitative analysis of application-specific dataflow architectures’. Proceedings of the IEEE international conference on *Application-specific systems, architectures and processors*, 1997, pp. 338–349
- [60] TABBARA, B., TABBARA, A., and SANGIOVANNI-VINCENTELLI, A.: ‘Function/architecture optimization and co-design of embedded systems’ (Kluwer Academic Publishers, Boston, MA, 2000)

- [61] BALARIN, F. *et al.*: ‘Hardware-software co-design of embedded systems: The POLIS approach’ (Kluwer Academic Publishers, Boston, MA, 1997)
- [62] BALARIN, F., WATANABE, Y., HSIEH, H., LAVAGNO, L., PASERONE, C., and SANGIOVANNI-VINCENTELLI, A.: ‘Metropolis: An integrated electronic system design environment’, *Computer*, 2003, **36**(4), pp. 45–52
- [63] VIRTUAL COMPONENT CO-DESIGN:
<http://www.cadence.com/>
- [64] KEUTZER, K., MALIK, S., and NEWTON, A.R.: ‘System-level design: orthogonalization of concerns and platform-based design’, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2000, **19**(12), pp. 1523–1543
- [65] TTTECH: TTP/C Specification Version 0.5, 1999, available at <http://www.tttech.com>
- [66] TTTECH: ‘Comparison CAN–Byteflight–FlexRay–TTP/C’. Technical report, available at <http://www.tttech.com>
- [67] NOLTE, T., HANSSON, H., NORSTRÖM, C., and PUNNEKKAT, S.: ‘Using bit-stuffing distributions in CAN analysis’. Proceedings of the IEEE/IEE *Real-time embedded systems* workshop, 2001
- [68] POP, P., ELES, P., and PENG, Z.: ‘Scheduling with optimized communication for time triggered embedded systems’. Proceedings of the international workshop on *Hardware-software codesign*, 1999, pp. 178–182
- [69] EDWARDS, S.: ‘Languages for digital embedded systems’ (Kluwer Academic Publishers, Boston, MA, 2000)
- [70] LEE, E.A., and PARKS, T.M.: ‘Dataflow process networks’, *Proceedings of the IEEE*, 1995, **83**, pp. 773–801
- [71] PUSCHNER, P., and BURNS, A.: ‘A review of worst-case execution-time analyses’, *Real-Time Systems Journal*, 2000, **18**(2/3), pp. 115–128
- [72] POP, P.: ‘Analysis and synthesis of communication-intensive heterogeneous real-time systems’. Linköping Studies in Science and Technology, PhD dissertation No. 833, 2003
- [73] SANDSTRÖM, K., and NORSTRÖM, C.: ‘Frame packing in real-time communication’. Proceedings of the international conference

- on *Real-time computing systems and applications*, 2000, pp. 399–403
- [74] POP, P., ELES, P., and PENG, Z.: ‘Bus access optimization for distributed embedded systems based on schedulability analysis’. Proceedings of the *Design automation and test in Europe* conference, 2000, pp. 567–574
- [75] POP, P., ELES, P., PENG, Z., IZOSIMOV, V., HELLRING, M., and BRIDAL, O.: ‘Design optimization of multi-cluster embedded systems for real-time applications’. Proceedings of *Design, automation and test in Europe* conference, 2004, pp. 1028–1033
- [76] ELES, P., DOBOLI, A., POP, P., and PENG, Z.: ‘Scheduling with bus access optimization for distributed embedded systems’, *IEEE Transactions on VLSI Systems*. 2000, **8**(5), pp. 472–491
- [77] AUDSLEY, N.C., BURNS, A., RICHARDSON, M.F., and WELLINGS, A.J.: ‘Hard real-time scheduling: the deadline monotonic approach’. Proceedings of the 8th IEEE workshop on *Real-time operating systems and software*, 1991, pp. 127–132
- [78] REEVES, C.R.: ‘Modern heuristic techniques for combinatorial problems’ (Blackwell Scientific Publications, Oxford, 1993)

Chapter 5

Using DVS Processors to Achieve Energy Efficiency in Hard Real-Time Systems

By **Flavius Gruian** and **Krzysztof Kuchcinski**

Department of Computer Science

Lund University

Email: {Falvius.Gruian,Krzysztof.Kuchcinski}@cs.lth.se

Timeliness and energy efficiency are often perceived as opposing requirements in digital systems design. However, with the advent of dynamic voltage scaling (DVS) processors even hard real-time systems can become energy efficient if proper scheduling methods are employed. Using DVS processors introduces another dimension into the problem, namely speed, leading to a whole new class of scheduling techniques. Several such techniques for speed scheduling are presented in here, ranging from task to task set level, involving both offline and runtime decisions. The methods described in here reduce the energy consumption while meeting all deadlines. Some of them are built on top of classic real-time scheduling techniques while others are totally new. The simulations and experiments carried out on an XScale i80200 evaluation platform confirm the validity of our methods.

5.1 Introduction

As the consumers demand more and more functionality from their laptops, PDAs, cellular phones, other mobile devices, and household appliances, reducing the energy consumption becomes an essential issue for embedded systems design. In this context, Dynamic Voltage Supply (DVS) processors seem to offer the best combination of flexibility and energy efficiency. Furthermore, real-time constraints can be ful-

filled and energy consumption reduced at the same time by selecting the processing speeds appropriately. However, with the new dimension of speed (clock frequency and supply voltage) introduced by these, special scheduling strategies are required in order to take full advantage of the available features. In this section we describe a number of such speed scheduling techniques, covering a rather wide spectrum of approaches from task to group level, from static to dynamic methods, including more complex, probabilistic techniques. Furthermore, these methods being rather orthogonal, can be combined to yield even more important energy reductions.

The current section is organized as follows. Sub-section 5.2 describes the hardware support for our techniques, namely the DVS processors, including our modeling assumptions and real figures. Sub-section 5.3 presents a task level speed scheduling technique based on execution pattern probability distribution. The following sub-sections address task set level scheduling. An offline technique for determining the maximum required speed for each task in a task set is presented in sub-section 5.4. Sub-section 5.5 describes a runtime technique for slack/speed management built on top of rate-monotonic scheduling strategy. Finally our conclusions are gathered in sub-section 5.6.

5.2 Hardware Support

A wide variety of DVS processor systems are available today on the market or as prototypes [BPSB00, PBB00, AMD00, Fle01, Int01]. Although their main characteristic is the ability to adjust their speed (core clock frequency and voltage) at run-time, different solutions achieve this in various ways. The number of speed settings is limited, varying between two (*Intel SpeedStep*) and tens of speeds (*Transmeta Crusoe*). Often these speed settings are not ideal due to the discrete increments in both voltage and clock frequency. Furthermore, only some parts of the processor are able to operate at different speeds, while others, such as the I/O pads, need to satisfy certain standards. Additionally, a speed switch has different characteristics for different processors or even for different initial and final speeds. The most important in our case is arguably the switch latency, which spans from tens of microseconds to milliseconds. Although the speed scheduling methods we describe make several simplifying assumptions (negligible overhead speed switches and continuous range of speeds), these have only a marginal effect on our techniques. Moreover, these assumptions may become closer to reality with newer generations of processors.

5.3 Stochastic Scheduling

Stochastic scheduling is a task level speed selection strategy designed for tasks with variable execution pattern. In a stochastic schedule a task instance starts executing at a low speed and then gradually accelerates, to meet the deadlines. Since the instance might not be a worst case, it can happen that high speed (and power eager) regions are avoided. In principle this approach minimizes the common case energy consumption at the expense of the less probable instances, as opposed to other approaches (i.e. compiler assisted) that attempt minimize the energy of each and every instance. For a detailed comparison between stochastic scheduling and compiler assisted techniques please refer to [Gru01]. The scheduling technique we present here is based on the same principle as the method described in [LS01], the difference residing in the selection of the speed switching points and the mapping of virtual speeds to real processor speeds.

The stochastic schedule for a task τ is obtained using its execution pattern $\eta(y)$, that can be found off-line, via simulation, or built and improved at run-time. Let us denote by X the random variable associated with the number of clock cycles used by a task instance. We also use the cumulative density of probability function, $cdf(x) = \sum_{y=1}^x \eta(y)$, associated with the random variable X . This function reflects the probability that a task instance finishes before a certain number of clock cycles. If WCE is the worst case number of clock cycles, clearly $cdf(z) = 1$ for $\forall z \geq \text{WCE}$.

Note that building a schedule for a task in our case means assigning a specific processor speed for every clock cycle up to WCE. Each cycle x , depending on the adopted speed, will consume a specific energy, e_x . But each of these cycles are executed with a certain probability, so the average energy consumed by cycle x can be computed as $(1 - cdf(x))e_x$. To obtain the expected energy for the whole task, we have to consider all the cycles up to WCE:

$$\bar{E} = \sum_{x=1}^{\text{WCE}} (1 - cdf(x))e_x \quad (5.1)$$

This is the value we want to minimize by choosing appropriate voltage levels and clock frequencies for each cycle.

A clock length of k_x corresponds to a clock frequency $f_x = 1/k_x$. The clock cycle energy for a certain frequency depending on the reference clock energy is:

$$e_x = e_{f_x} = e_{ref} \left(\frac{f_x}{f_{ref}} \right)^\beta = \frac{e_{ref}}{f_{ref}^\beta} \frac{1}{k_x^\beta} = \mathcal{K} \frac{1}{k_x^\beta} \quad (5.2)$$

where e_{ref}/f_{ref}^β is constant (\mathcal{K}) for a given processor. For clarity we bind now $\beta = 2$ (the common case), but the rest of the calculus can be carried out for any reasonable value of β .

Task τ has to complete its execution during an allowed execution time, A . If we denote the clock length associated to clock cycle x by k_x , this can be written as:

$$\sum_{x=1}^{\text{WCE}} k_x \leq A \quad (5.3)$$

If we substitute 5.2 in 5.1 we obtain:

$$\bar{E} = \mathcal{K} \sum_{x=1}^{\text{WCE}} \frac{(1 - \text{cdf}(x))}{k_x^2} \quad (5.4)$$

which is the value to be minimized. For the sake of simplicity and without loss of generality we will assume $\mathcal{K} = 1$ from now on. By mathematical induction, the right hand side of 5.4 has a lower bound (using also 5.3):

$$E_{LB} = \frac{\left(\sum_{x=1}^{\text{WCE}} \sqrt[3]{1 - \text{cdf}(x)}\right)^3}{\left(\sum_{y=1}^{\text{WCE}} k_y\right)^2} \geq \frac{1}{A^2} \left(\sum_{x=1}^{\text{WCE}} \sqrt[3]{1 - \text{cdf}(x)}\right)^3 \quad (5.5)$$

This energy lower bound can be reached if and only if:

$$k_y = A \frac{\sqrt[3]{1 - \text{cdf}(y)}}{\sum_{x=1}^{\text{WCE}} \sqrt[3]{1 - \text{cdf}(x)}} \quad (5.6)$$

These are the optimal values for the clock cycle length in each clock cycle up to WCE.

For processors with a discrete range of speeds, these values will most likely not overlap with the available clock lengths. To transform the ideal clock frequencies into real ones, we have to distribute the work done in each ideal clock cycle to real clock cycles. To obtain the work performed during real clock cycles, we start from the two consecutive available clock cycles bounding the ideal clock cycle k_y , $CK_i < k_y \leq CK_{i+1}$. The work of the ideal cycle can be performed during the same time interval using real clock cycles, if:

$$k_y = w_{iy}CK_i + w_{(i+1)y}CK_{i+1} \wedge w_{iy} + w_{(i+1)y} = 1 \wedge k_y \in (CK_i, CK_{i+1}]$$

where w_{iy} is the part of the work of k_y given to CK_i and the rest is the work given to CK_{i+1} . Thus, each ideal cycle in the task will distribute its work between two of the several available clock lengths. It is possible that several ideal clock cycles k_y will distribute some of their work to the same real clock cycle j , since they end up to be right above or right below

the real clock cycle value: $k_y \in (CK_{j-1}, CK_j]$ or $k_y \in (CK_j, CK_{j+1}]$. For simplicity, we denote the set of ideal clock cycles k_y distributing their work load to the same real clock cycle j by \mathcal{Y}_j . The accumulated workloads for each available clock cycle is obtained by summing up the workloads resulting from individual ideal clock cycles:

$$w_j = \sum_{k_y \in \mathcal{Y}_j} w_{jy}, \quad j \in 1 \dots VL \quad (5.7)$$

where VL is the number of available processor speeds (corresponding to pairs of supply voltage levels and clock frequencies). Note that the cumulated workloads w_j are real numbers. Since we can only execute an integer number of clock cycles, these workloads have to be transformed to integers.

5.3.1 Computational complexity

Computing the stochastic schedule for a task has to be done before the task starts executing. If the allowed execution time A varies, the actual distribution of the workload to speeds also varies. In that case, the exact stochastic schedule has to be determined at run-time. Yet, there are computations that can be performed off-line. If the probability distribution of the execution pattern is available off-line, the coefficients of A in equation 5.6 can also be computed off-line. If the probability distribution is built at run-time, these coefficients need to be recomputed every time the distribution changes. This step has an algorithmic complexity of order $O(\text{wCE})$. The exact values for the ideal clock cycles have to be computed at run-time, when the allowed execution time becomes known (also exhibiting $O(\text{wCE})$ complexity). Computing the cumulative workloads requires finding the bounding clock frequencies, available on a real processor, for each ideal clock cycle $O(\text{wCE} \log VL)$. Transforming the real workloads into integer numbers will also exhibit $O(VL)$. Finally, assuming $VL \leq \text{wCE}$, which is reasonable since VL , the number of processor speeds, is rather small, the overall worst case complexity for computing the stochastic schedule is $O(\text{wCE} \log VL)$.

Example 1 (Task Level Voltage Schedules on Intel 80200)

To practically examine the energy saved by a stochastic approach versus a wCE-stretch method we used an i80200 XScale system. The task in this experiment is composed of a simple loop that terminates after a variable number of iterations in each instance. The number of iterations is computed at the beginning of each job according to a normal distribution with $\mu = 950$ and $\sigma = 300$. At the highest speed (733MHz), the best case and the worst case execution times are 30ms and 78ms, respectively. We considered that the task has to finish in at most 117ms (which is 150%

of the worst case at the highest speed). In a classic non-probabilistic approach, we have to choose the ideal speed for which the worst case will finish exactly at the deadline. This will be realized in practice by using two real speed settings: 533MHz@1.3V and 466MHz@1.2V. Using only the 533MHz@1.3V speed setting would give 107ms execution time in the worst case while using only the 466MHz@1.2V speed setting would be too slow since the worst case would take about 123ms. Finally, the best non-stochastic schedule will be composed by executing first 1150 iterations of the main loop at 466MHz@1.2V and the rest at 533MHz@1.3V. The power distribution for this schedule is depicted in the oscilloscope trace from Figure 5.1 **above-left**.

To obtain a stochastic schedule in this situation we have to know the best and worst case execution patterns. These two cases happen for the extremes of the value controlling the iterations in the task main loop. More precisely, these are 50 and 1850 respectively (the $[-3\sigma, 3\sigma]$ interval). Using these numbers as the input to our method for computing a stochastic schedule we get the number of iterations to be executed at each speed between the lowest and the highest: [186, 640, 181, 109, 79, 62, 593]. The power distribution for this schedule is depicted in the oscilloscope trace from Figure 5.1 **below-left**. When using the stochastic schedule, the worst case takes almost 117ms including the speed switches.

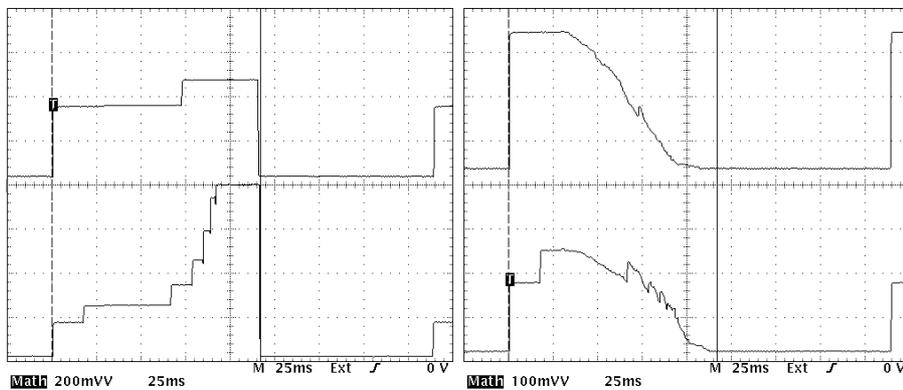


Figure 5.1: **On the left:** The power distribution for the non-stochastic (**above**) and stochastic (**below**) schedules used in example 1. Different power levels reflect the different speed settings used. The worst case covers the whole schedule, while the other cases only the initial sections of the schedule. **On the right:** The averaged power curve of the corresponding schedules, the area under the graph giving the energy consumption.

Running the two schedules for a large number of instances and averaging the power consumption, we obtain the average energy consumptions for both. The non-stochastic average energy, as resulting from the

trace in Figure 5.1, is around 24mJ while the stochastic schedule average energy is around 20mJ. Finally, this means that in the long run, using a stochastic scheduling approach, the energy consumption decreases by approximately 18% in this case.

5.4 Finding the Maximum Required Speeds

The Maximum Required Speed (MRS) method presented in this section is designed as an off-line step for task sets scheduling. The main idea of this approach rests on reaching a close to 100% processor utilization by identifying tasks that can run slower. More precisely, MRS is not a scheduling algorithm in itself, but a procedure to compute the smallest possible processing speeds for which a task set is still schedulable with a specific on-line algorithm. At run-time, the task set is still scheduled according to the on-line algorithm, but the processor speeds are set using the off-line computed values. Note that the speed choice must not affect the feasibility of the schedule.

For RM scheduling, a simple approach would be to compute a utilization-based speed starting from the condition proposed by Liu and Layland in [LL73]. With this approach the maximum required speed is unique and equal to:

$$s_{\text{RM-MRS}} = \frac{U}{N(2^{1/N} - 1)} \quad (5.8)$$

where U is the processor utilization and N the number of tasks. At a closer look, the schedule feasibility condition proposed in [LL73] is a sufficient one and covers the worst possible case for the task group characteristics. An exact analysis as proposed in [LSD89] may further reveal possibilities for stretching tasks while still meeting the deadlines. Based on this, [SC99] describes a method to compute the maximum required frequency (speed) for a task set. We go even further, and instead of computing a single common maximum required speed for the whole task set $\{\tau_i\}_{i=1\dots N}$, as in [SC99], we compute individual speeds for each task τ_i . Note that the speed required by a task is inverse proportional to the task stretching factor. Finding the maximum required speeds is in fact equivalent to finding the minimal stretching factors $\{\alpha_i\}_{i=1\dots N}$, computed as follows. Next, each task will be modeled by three relevant parameters, namely its worst case execution time \mathcal{C}_i (at the reference clock speed), its deadline D_i and period T_i . We also consider that the tasks in the group are indexed according to their priority, computed as in RM-scheduling ($1/T_i$).

The stretching factors are computed in an iterative manner, starting from the highest priority tasks and continuing with lower priority tasks.

Consider that index q points to the latest task which has been assigned a stretching factor. Initially, $q = 0$. Each of the tasks $\{\tau_i\}_{q < i \leq N}$ has to be executed before one of its scheduling points S_i ([LSD89]):

$$S_i = \left\{ kT_j \mid 1 \leq j \leq i \wedge 1 \leq k \leq \left\lfloor \frac{T_i}{T_j} \right\rfloor \right\} \quad (5.9)$$

The above equation defines all the scheduling points when the deadlines are equal to task periods, $T_i = D_i$. For task sets where $T_i \neq D_i$, we need to change the set of scheduling points according to:

$$S'_i = \{t \mid t \in S_i \wedge t < D_i\} \cup \{D_i\} \quad (5.10)$$

Task τ_i exactly meets its deadline if there exists a scheduling point $S_{ij} \in S'_i$ for which the following relation holds:

$$\sum_{1 \leq r \leq q} \alpha_r C_r \left\lceil \frac{S_{ij}}{T_r} \right\rceil + \alpha_{ij} \sum_{q < p \leq i} C_p \left\lceil \frac{S_{ij}}{T_p} \right\rceil = S_{ij} \quad (5.11)$$

Note that for the tasks which already have assigned a stretching factor we used that one, α_r , while for the rest of the tasks we assumed they will all use the same and yet to be computed stretching factor, α_{ij} , which is dependent on the scheduling point. For task τ_i , the best scheduling choice, from the energy point of view, is the largest of its α_{ij} . At the same time, from equation 5.11, this is equal for all tasks $\{\tau_i\}_{q < i \leq N}$. In fact, there is a task with index m for which its best stretching factor is the smallest among all other tasks:

$$\exists m, q < m \leq N, \text{ such that } \max_{q < j \leq m} (\alpha_{mj}) = \min_{q < i \leq n} (\max_{q < j \leq i} (\alpha_{ij})) \quad (5.12)$$

Note that this does not necessarily correspond to the last task, τ_N . If $q = 0$, this task sets the minimal clock frequency as computed in [SC99]. Having found the index m , all tasks between q and m can be at most stretched (equally) by the stretching factor of m . Thus, we assign them stretching factors as:

$$\alpha_r = \max_{q < j \leq m} (\alpha_{mj}), r = q + 1 \dots m \quad (5.13)$$

With this, an iteration of the algorithm for finding the stretching factors is complete. The next iteration then proceeds for $q = m$. The process ends when q reaches N , meaning all tasks have been given their own off-line stretching factors. Finally, the maximum required processor speed for each task is given by the inverse of its off-line stretching factor:

$$s_i = 1/\alpha_i, i = 1 \dots N \quad (5.14)$$

A similar approach can be derived also for EDF scheduling strategy, as described in [Gru02]. Furthermore, the method can be adapted for tasks with synchronization, as shown in [JG02].

Example 2 (RM-MRS for a Set of Five Tasks) *This example contains the results of MRS for the set of five tasks described in Table 5.1. For this set, the task deadlines are equal to the task periods. For the RM scheduling method, the stretching factors are computed individually. Note that tasks 3 and 4 can be stretched off-line more than 1 and 2, while 5 has the largest stretching factor. The processor utilization changes from 0.687 to 0.994. Observe also that the stretching factors for the lower priority tasks require more iterations to compute. For the EDF scheduling, there is a single stretching factor, common to all tasks, equal to $1/0.687$ (refer to [Gru02]). The maximum required processor speeds relative to the reference speed are obtained by inverting the α factors.*

Table 5.1: A Numerical Example of MRS

Task			MRS α factors (and speeds)		
#	WCET (C)	Period (T)	RM		EDF
			$\alpha_{\#}$ (speeds)	iterations	α (speed)
1	1	5	1.428 (0.700)	1	.
2	5	11	1.428 (0.700)	1	.
3	1	45	1.785 (0.560)	2	1.4556 (0.687)
4	1	130	1.785 (0.560)	2	.
5	1	370	2.357 (0.424)	3	.

If we consider that the tasks have fixed execution pattern, we can easily compute the energy consumptions for the RM-MRS and EDF-MRS. For this we found out the number of executed instances of each task over the task set hyper-period, computed as the least common multiplier (lcm) of the task periods. For our example, $\text{lcm}(5, 11, 45, 130, 370)$ is 476190. Next, we know that the energy consumed during a clock cycle is dependent on the square of the processor speed. The energy of a task instance is therefore proportional to $C_i s_i^2$. Finally, we can sum up the energy consumption after the number of instances for each task. The numerical results are detailed in Table 5.2. Note that, for this example, we assumed that no power is consumed during idle and speed switching. Also, the processor is ideal in the sense that it can run at any speed under the reference speed. For this particular example, both RM-MRS and EDF-MRS manage to save about 52% of the energy consumed by using only the classic RM and EDF employing the same, reference speed for all tasks.

5.5 Runtime Slack Management

For tasks with fixed execution pattern, static/off-line scheduling techniques are sufficient. However if tasks with variable execution pattern are present in the system, there are certain idle periods that are very hard or impossible to predict off-line. These idle times can be efficiently used by other tasks if run-time/on-line speed scheduling methods are

Table 5.2: RM-MRS and EDF-MRS energy consumption for the task set in Table 5.1

Task #	Instance Energy		Instances per Hyper-period	All Instances Energy	
	RM-MRS	EDF-MRS		RM-MRS	EDF-MRS
1	0.4904	0.4720	95238	46704.0	44952.3
2	2.4520	2.3599	43290	106147.0	102160.1
3	0.3139	0.4720	10582	3321.6	4994.7
4	0.3139	0.4720	3663	1149.8	1728.9
5	0.1800	0.4720	1287	231.7	607.5
Total Energy Consumption:				157554.1	154443.5
% from Max Speed Energy (327220.0)				48.15%	47.20%

employed. On-line speed scheduling of task sets was already addressed previously in several publications. In [SC99] a task instance is run at a lower speed only if it is the only one running and has enough time until a new task arrives. In all other situations tasks are executed at the speed dictated by the off-line analysis. The algorithm presented in [LK99] uses only two voltage levels. The slack produced by finishing a task early is entirely used to run the processor at the low voltage. As soon as this slack is consumed, the task switches to the high voltage. The method we describe next is perhaps most resemblant to the optimal scheduling method OPASTS presented in [HPS98]. However OPASTS performs analysis over task hyper-periods, which may lead to working on a huge number of task instances for certain task sets.

In the following, we describe a task set speed scheduling strategy built on top of rate-monotonic (RM) scheduling. Our method is designed to work on processors with an arbitrary number of speeds. It has a low computational complexity, independent of the characteristics of the task sets. Briefly, in our strategy, an early finishing task may pass on its unused processor time to any of the tasks executing next. But this slack cannot be used by any task at any time since deadlines have to be met. We solve this by considering several levels of slacks, with different priorities, as in the slack stealing algorithm from [LRT92].

Next, we present the on-line slack management strategy in detail and prove that our strategy keeps the worst case response time guaranteed by RM-scheduling. We conclude by describing a few experiments showing the effectivity of our method.

5.5.1 The Slack Distribution Strategy

Our on-line slack distribution strategy makes use of several levels of slack. For a task set $\{\tau_i\}_{1 \leq i \leq N}$ that exhibits M different priorities, we use M levels of real-time slack $\{S_j\}_{1 \leq j \leq M}$. Without great loss of

generality consider that the tasks have different priorities, or $M = N$. We also consider that the task set and slack levels are already ordered by priority, where level 1 corresponds to the highest priority. The slack in each level is a cumulative value, the sum of the unused processor times remaining from the tasks with higher priority. Initially, all level slacks S_j are set to 0. At run-time, the slack levels are managed as follows.

Whenever an instance k of a task τ_i with priority i **starts executing**, it can use an arbitrary part $\Delta\mathcal{C}_i^k$ of the slack available at level i , S_i . So the allowed execution time for instance k of task τ_i will be:

$$A_i^k = \mathcal{C}_i + \Delta\mathcal{C}_i^k \quad (5.15)$$

where \mathcal{C}_i is task τ_i worst case execution time for the maximum (reference) processor speed. The remaining slack from level i cannot be used again on the same level. Therefore the slack level i is reset to 0. We can also see this as a degradation of the slack from level i into level $i + 1$ slack. To summarize, each level of slack will be updated according to:

$$S'_j = \begin{cases} 0 & , j \leq i \\ S_j - \Delta\mathcal{C}_i^k & , j > i \end{cases} \quad (5.16)$$

Whenever a task instance **finishes its execution**, it will generate some slack if it finishes before its allowed time. If X_i^k is the actual execution time of instance k of task τ_i , the generated slack is:

$$\Delta A_i^k = A_i^k - X_i^k \quad (5.17)$$

This slack can be used by the lower priority tasks. In this case, the slack levels are updated according to:

$$S''_j = \begin{cases} S_j & , j \leq i \\ S_j + \Delta A_i^k & , j > i \end{cases} \quad (5.18)$$

Finally, **idle processor times** are subtracted for all slacks.

The computational complexity required by the on-line method is, thus, linearly dependent to the number of slack levels: $O(M)$. Note that task instances can only use slack generated from higher priority tasks and produce lower priority slack. We call this slack degradation. Whenever the lowest priority task starts executing, all slack levels are reset (see equation 5.16). Note also that not necessarily all slack at one level is used by a single task. Various strategies can be employed, but we mention here only the two we used in our experiments:

- **Greedy**: the task gets all the slack available for its level:

$$\Delta\mathcal{C}_i^k = S_i \quad (5.19)$$

- **Mean proportional:** The slack is proportionally distributed according to the mean execution time \overline{X}_i for each task instances waiting to execute:

$$\Delta C_i^k = S_i \frac{\overline{X}_i}{\sum_{\tau_j \in \text{ReadyQ}} \overline{X}_j} \quad (5.20)$$

5.5.2 Worst Case Response Time Analysis

The strategy of managing the slack just described, allows us to keep the critical instance response time for all tasks, as we prove next. The response time $R_i(t)$ for task τ_i is computed as:

$$R_i = A_i + I_i(t) \quad (5.21)$$

where A_i is its allowed execution time, as before, and $I_i(t)$ is the interference from the other tasks. From the managing strategy given before, the cumulated slack on each level i , at a certain time t is of the form:

$$S_i(t) = S_{i-1} - \sum_k \Delta C_{i-1}^k + \sum_k \Delta A_{i-1}^k, \quad k = \left\lceil \frac{t}{T_{i-1}} \right\rceil \quad (5.22)$$

More informally, the slack of level i is composed of all slack from level $i-1$, less the slack used by the instances of tasks with priority $i-1$ but plus all the slack generated by these. The number of instances executed in the current hyper-period, k , is determined by the task period. Note that S_1 is always zero. Eliminating the iteration:

$$S_i(t) = \sum_{1 \leq j < i} \left(\sum_k \Delta A_j^k - \sum_k \Delta C_j^k \right), \quad k = \left\lceil \frac{t}{T_j} \right\rceil \quad (5.23)$$

The task with the highest priority will never receive slack, therefore $\Delta C_1^k = 0$. The interference from the high priority tasks, $I_i(t)$ is the time used to execute all arrived instances of these high priority tasks:

$$I_i(t) = \sum_{1 \leq j < i} \sum_k X_j^k, \quad k = \left\lceil \frac{t}{T_j} \right\rceil \quad (5.24)$$

With the notations from 5.5.1, we write the relation between the instance k execution time, X_j^k , its allowed time A_j^k , its used slack ΔC_j^k and its produced slack, ΔA_j^k :

$$X_j^k = A_j^k - \Delta A_j^k = C_j + \Delta C_j^k - \Delta A_j^k, \quad k = \left\lceil \frac{t}{T_j} \right\rceil \quad (5.25)$$

Introducing this in 5.24:

$$I_i(t) = \sum_{1 \leq j < i} \sum_k (\mathcal{C}_j + \Delta \mathcal{C}_j^k - \Delta A_j^k), \quad k = \left\lceil \frac{t}{T_j} \right\rceil \quad (5.26)$$

The last two terms in the sum are actually giving the slack of level i , as in 5.23, so we can re-write 5.26 as:

$$I_i(t) = \sum_{1 \leq j < i} k \mathcal{C}_j - S_i(t), \quad k = \left\lceil \frac{t}{T_j} \right\rceil \quad (5.27)$$

Note that the maximal response time for a task is obtained when it uses all the slack available at its level:

$$R_i(t) = \mathcal{C}_i + I_i(t) + S_i(t) \quad (5.28)$$

From the last two equations:

$$R_i(t) = \mathcal{C}_i + \sum_{1 \leq j < i} \left\lceil \frac{t}{T_j} \right\rceil \mathcal{C}_j \quad (5.29)$$

which is exactly the response time as defined by the RM analysis, obtained when all tasks execute their worst case [LL73, BW01]. Thus, if the RM analysis decides that a task set can be scheduled, the conclusion remains valid when using our on-line slack distribution policy.

5.5.3 Experimental Results

To evaluate our slack distribution strategy from the energy consumption point of view, we compared it to several other methods or possible bounds:

- **100%:** No speed scheduling is performed, all tasks run as fast as possible.
- **Upper Bound:** Is the theoretical upper bound in energy reduction. This is obtained in a post-execution analysis, by considering that the tasks are uniformly stretched up to maximum processor utilization. This limit is hardly achievable in practice, since the actual execution patterns for all task instances are never available beforehand. Moreover, this optimum usually violates some deadlines, being therefore impractical.
- **Off-line+1stretch:** This method is composed of the off-line RM-MRS step from 5.4 and a very simple run-time speed scheduling, originally described in [SC99]. Namely, whenever a job is running alone on the processor, it is allowed to use all the time until the next arrival of any job.

- **All:** Our run-time speed scheduling strategy using the *mean proportional* slack distribution method, and the *Stochastic Scheduling* task level strategy (see 5.3), augmented with the previous *Off-line+1stretch* method. This is the most complete low-energy scheduling approach, combining off-line, run-time, task level and task set level scheduling strategies.
- **Ideal:** Same as *All*, except assuming exact knowledge about each job execution pattern. This means that every time a task arrives, its exact execution pattern becomes known. Thus, we can directly set an ideal speed for that job.

The virtual processor used for these experiments has 14 voltage levels, with clock frequencies equally distributed between $f=100\text{MHz}$ and 11MHz . A power-down mode is also available, in which the processor consumes 5% of the highest frequency average energy. We assumed that the NOP instruction consumes only 20% of the average power at the maximum speed, as in [SC99]. For the first scheme, 100%, we assume that whenever the processor is idle, it executes NOPs, while for the rest of the schemes, we assume that the processor goes into the available low-power mode.

First, we took two real-life hard-RT applications and simulated them on the virtual processor using the scheduling modes given above. The first application, slightly adapted from [KRH⁺96], is a computerized numerical control machine (CNC controller) and has the task set characteristics given in Table 5.3. The second application, adapted from [LVM91], is a generic avionics platform (GAP) and has the task set characteristics given in Table 5.4. For both task sets we assumed tasks with probabilistic execution patterns as follows. We considered that the number of clock cycles varies between a best case (BCE) and a worst case (WCE) according to a normal distribution. All distributions have the mean $\mu = (\text{BCE} + \text{WCE})/2$ and the standard deviation $\sigma = (\text{WCE} - \text{BCE})/6$. Keeping the WCE given in the initial specification, we varied the BCE such that the BCE/WCE ratio changed from 0.1 to 0.9. A small ratio means that the task execution pattern can vary a lot, while a close to 1 ratio means that the task has almost fixed execution pattern. For each ratio we applied the scheduling methods given above and estimated the energy consumption via simulation. The results are depicted in Figure 5.2.

We also tested our *All* scheduling policy on randomly generated task sets of 50 and 100 tasks. The task sets were generated as follows. For each set, the task periods (and deadlines) were selected using a uniform distribution in $100 \dots 5000$ and $100 \dots 10000$ respectively. The worst case execution times were then randomly generated such that the task set would yield approximately 0.67 processor utilization, for the fastest

Table 5.3: Task set characteristics for the CNC controller

# (priority)	in [KRH ⁺ 96] appear as	Task Characteristics		
		WCE@ f_{max} (μs)	T (μs)	D (μs)
1	τ_{simpl}	35	2400	2400
2	τ_{calv}	40	2400	2400
3	τ_{xref}	165	2400	2400
4	τ_{yref}	165	2400	2400
5	τ_{xctrl}	570	9600	4000
6	τ_{yctrl}	570	7800	4000
7	τ_{dist}	180	4800	4800
8	τ_{stts}	720	4800	4800

Table 5.4: Task set characteristics for GAP

# (priority)	in [LVM91] appear as	Task Characteristics	
		WCE@ f_{max} (μs)	T = D (μs)
1	Radar_Tracking_Filter	200	2500
2	RWR_Contact_Mgmt	500	2500
3	Data_Bus_Poll_Device	100	4000
4	Weapon_Aiming	300	5000
5	Radar_Target_Update	500	5000
6	Nav_Update	800	5900
7	Display_Graphic	900	8000
8	Display_Hook_Update	200	8000
9	Tracking_Target_Upd	500	10000
10	Weapon_Release	300	20000
11	Nav_Steering_Cmds	300	20000
12	Display_Stores_Update	100	20000
13	Display_Keyset	100	20000
14	Display_Stat_Update	300	20000
15	BET_E_Status_Update	100	100000
16	Nav_Status	100	100000
	Timer_Interrupt	not included (implicit)	
	Weapon_Protocol	not included (aperiodic)	
Total Utilization = 84.05			

clock. The average utilization after off-line RM-MRS turned out to be 0.92 for the sets of 50 tasks, and 0.85 for the sets of 100 tasks. Using the same processor type as in the previous experiment, we simulated the run-time behavior of the several scheduling methods. We again used post-simulation data to obtain the upper bound, as in the previous experiment. Figure 5.3 depicts averages over one hundred sets of tasks.

As results from these experiments, for tasks with probabilistic execution pattern, an on-line slack distribution strategy yields a dramatical

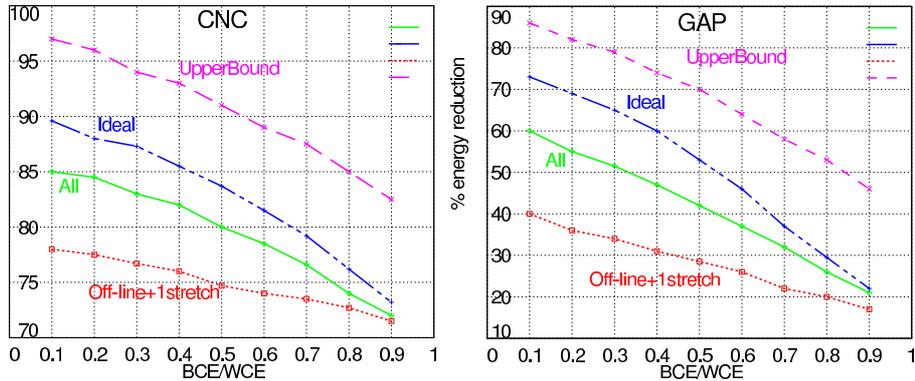


Figure 5.2: The energy reduction off the 100% execution mode for the CNC controller (left) and the GAP application (right).

decrease in energy consumption. Simpler on-line scheduling techniques, such as *Off-line+1stretch* [SC99], are improved significantly by adding our slack management policy. Task level scheduling strategies may additionally contribute to this result, yet with a marginal 3-4%. In fact, any further decrease in energy consumption requires great efforts, since we are approaching the practical bound. An interesting property of the *All* policy seems to be the fidelity it follows the *UpperBound*. *All* is consistently 12%, 25%, 20% and 25% under the *UpperBound* for CNC, GAP, 50 task sets, and 100 task sets, respectively. This means that our scheduling strategy uses the variation of BCE/WCE ideally, performing with the same efficiency independent of the task execution pattern characteristics.

5.6 Summary and Conclusions

We have shown that hard real-time and energy efficiency are not necessarily opposing characteristics for systems employing dynamic voltage scaling (DVS) processors. In principle, the technique attempts to reduce the processing speed whenever this does not affect the real-time deadlines. A number of such techniques for uni-processor systems have been presented, both offline and runtime, for isolated tasks and groups of tasks. All these techniques are orthogonal and can be employed together to reduce the energy consumption at multiple levels.

At task level, we have focused on a method especially designed for tasks with variable execution time, namely *stochastic scheduling*. Starting from the probability distribution of the task execution time, stochastic scheduling derives a static schedule which minimizes the expected energy consumption over a large number of execution instances.

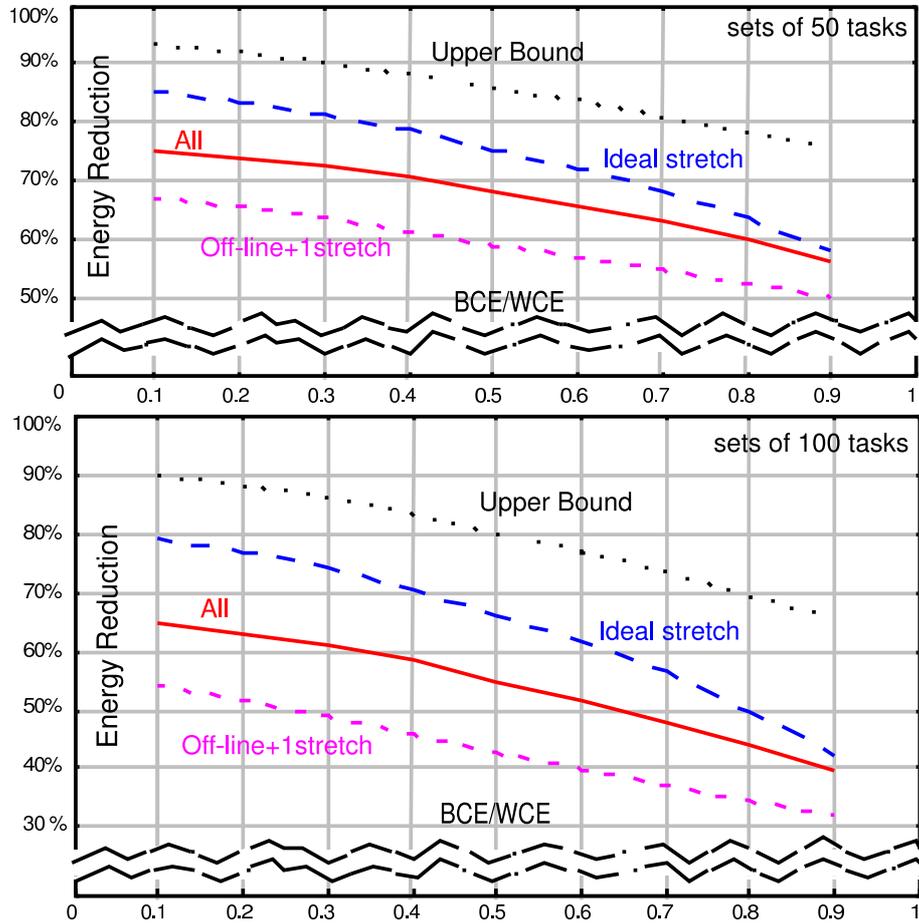


Figure 5.3: The average energy reduction off the 100% execution mode for a hundred sets of (*above*) 50 tasks and (*below*) 100 tasks.

For task groups we described first an offline algorithm for selecting the maximal required speeds for rate-monotonic scheduling. Then we presented a run-time slack management strategy built on top of rate-monotonic scheduling. Our run-time technique attempts to reduce the processing speeds for all tasks by intelligently distributing the available free processor time among all ready instances. Finally, the experimental results we presented show the efficiency of our techniques in particular, and the possibility of energy reduction in hard real-time systems in general.

Bibliography

- [AMD00] AMD. *AMD PowerNow™ Technology Dynamically Manages Power and Performance, Rev. A*, November 2000. Informational White Paper No. 24404.
- [BPSB00] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid State Circuits*, 35(11), November 2000.
- [BW01] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages*. Addison-Wesley, 3rd edition, 2001.
- [Fle01] M. Fleischmann. LongRun power management - dynamic power management for crusoe processors. Technical report, Transmeta Corporation, January 17, 2001.
- [Gru01] F. Gruian. On energy reduction in hard real-time systems containing tasks with stochastic execution times. In *IEEE Workshop on Power Management for Real-Time and Embedded Systems*, pages 11–16, May 29 2001.
- [Gru02] F. Gruian. *Energy-Centric Scheduling for Real-Time Systems*. Doctoral dissertation, Dept. of Computer Science, Lund Institute of Technology, December 2002.
- [HPS98] I. Hong, M. Potkonjak, and M. B. Srivastava. On-line scheduling of hard real-time tasks on variable voltage processor. In *Digest of Technical Papers of ICCAD'98*, pages 653–656, 1998.
- [Int01] Intel. *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Datasheet*, September 2001. Order Number: 273414-003.
- [JG02] R. Jejurikar and R. Gupta. Energy aware task scheduling with task synchronization for embedded real time systems. In *Proceedings of the 2002 International Conference on Compilers, Architectures, and Synthesis for Embedded Systems*, pages 164–169, 2002.
- [KRH⁺96] N. Kim, M. Ryu, S. Hong, M. Saksena, C.-H. C.-H. Choi, and H. Shin. Visual assessment of a real-time system design: A case study on a CNC controller. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pages 300–310, 1996.

- [LK99] Y.-H. Lee and C. M. Krishna. Voltage-clock scaling for low energy consumption in real-time embedded systems. In *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications*, pages 272–279, 1999.
- [LL73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real time environment. *JACM*, 20(1):46–61, 1973.
- [LRT92] J. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Proceedings of the 1992 Real Time Systems Symposium*, pages 110–123, 1992.
- [LS01] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with PACE. In *Proceedings of ACM SIGMETRICS 2001 Conference*, pages 50–61, 2001.
- [LSD89] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proceedings of the 1989 Real Time Systems Symposium*, pages 166–171, 1989.
- [LVM91] C. D. Locke, D. R. Vogel, and T. J. Mesler. Building a predictable avionics platform in Ada: a case study. In *Proceedings of the 12th IEEE Real-Time Systems Symposium*, pages 181–189, 1991.
- [PBB00] T. A. Pering, T. D. Burd, and R. W. Brodersen. Voltage scheduling in the lpARM microprocessor system. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pages 96–101, 2000.
- [SC99] Y. Shin and W. Choi. Power conscious fixed priority scheduling for real-time systems. In *Proceedings of the 36th Design Automation Conference*, pages 134–139, 1999.

Chapter 6

Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times

By **Sorin Manolache**, **Petru Eles** and **Zebo Peng**

Department of Computer and Information Science

Linköping University

Email: {sorma,petel,zpe}@ida.liu.se

6.1 Introduction

The design process of embedded real-time systems typically starts from an informal specification together with a set of constraints. This initial informal specification is then captured as a more rigorous model, formulated in one or several modelling languages [PDHT96, SWC95]. During the system level design space exploration phase, different architecture, mapping and scheduling alternatives are assessed in order to meet the design requirements and possibly optimise the values of certain quality or cost indicators [DG97, BS98, LPW99, PR99, Wol94, Ern98]. The existence of accurate, fast and flexible automatic tools for performance estimation in every design phase is of capital importance for cutting down design process iterations, time, and cost.

Performance estimation tools used in the early design stages do not benefit from detailed information regarding the design and, hence, can provide only rough estimates of the final performance of the system to be implemented. In the later design stages, before proceeding to the synthesis and/or integration of the software, hardware and communi-

cation components of the system, it is important that the system performance, as predicted by the estimation tools based on the now more detailed system model, is accurate with respect to the real performance of the manufactured and deployed product. An accurate performance estimation at this stage would leverage the design process by allowing the designer to efficiently explore several design alternatives. Such a performance estimation algorithm is the topic of this article.

Historically, real-time system research emerged from the need to understand, design, predict, and analyse safety critical applications such as plant control and aircraft control, to name a few [LL73, Kop97, But97]. Therefore, the community focused on hard real-time systems, where breaking a timeliness requirement is intolerable as it may lead to catastrophic consequences. In such systems, if not all deadlines are guaranteed to be met, the system is said to be unschedulable. The schedulability analysis of hard real-time systems answers the question whether the system is schedulable or not [ABD⁺95, Fid98, BLMSV98, SR93]. The only way to ensure that no real-time requirement is broken is to make conservative assumptions about the systems, such as, for example, that every task instantiation is assumed to run for a worst case time interval, called the worst case execution time (WCET) of the task.

Applications belonging to a different class of real-time systems, the soft real-time systems, are considered to still correctly function even if some timeliness requirements are occasionally broken, as this leads to a tolerable reduction of the service quality [BLA98]. For example, multimedia applications, like JPEG and MPEG encoding, audio encoding, etc., exhibit this property [AB98a]. In this context, we would be interested in the *degree* to which the system meets its timeliness requirements rather than in the binary answer provided by the hard real-time system schedulability analysis. In our work, this degree is expressed as the *expected ratio of missed deadlines* for each task graph or task.

The execution time of a task is a function of application-dependent, platform-dependent, and environment-dependent factors. The amount of input data to be processed in each task instantiation as well as its value and type (pattern, configuration) are application-dependent factors. The type of processing unit that executes a task is a platform-dependent factor influencing the task execution time. If the time needed for communication with the environment (database lookups, for example) is to be considered as a part of the task execution time, then network load is an example of an environmental factor influencing the task execution time.

Input data amount and type may vary, as for example is the case for differently coded MPEG frames. Platform-dependent characteristics, like cache memory behaviour, pipeline stalls, write buffer queues, may also introduce a variation in the task execution time. Thus, ob-

viously, all of the enumerated factors influencing the task execution time may vary. Therefore, a model considering variable execution time would be more realistic as the one considering fixed, worst case execution times. In a more general model of task execution times, arbitrary task execution time probability distribution functions (ETPDFs) are considered. These distributions can be extracted from performance models [van96] by means of analytic methods or simulation and profiling [van03, Gv00, Gau98]. Obviously, the fixed task execution time model is a particular case of such a stochastic one.

An approach based on a worst case execution time model would implement the task on an expensive system which guarantees the imposed deadline for the worst case situation. This situation, however, will usually occur with a very small probability. If the nature of the application is such that a certain percentage of deadline misses is affordable, a cheaper system, which still fulfils the imposed quality of service, can be designed.

This article proposes an algorithm for obtaining the expected ratio of missed deadlines per task graph or task, given a set of task graphs with the following assumptions: the tasks are periodic, the task execution times have given generalised probability distribution functions, the task execution deadlines are given and arbitrary, the scheduling policy belongs to practically any class of non-preemptive scheduling policies, and a designer supplied maximum number of concurrent instantiations of the same task graph is tolerated in the system.

The sequel of the article is structured as follows. The next section surveys related work and comparatively comments on our approach. Section 6.3 captures the problem formulation. Section 6.4 discusses our algorithm in detail. Section 6.5 presents experiments and the last section draws the conclusions.

6.2 Related work

Atlas and Bestavros [AB98b] extend the classical rate monotonic scheduling policy [LL73] with an admittance controller in order to handle tasks with stochastic execution times. They analyse the quality of service of the resulting schedule and its dependence on the admittance controller parameters. The approach is limited to rate monotonic analysis and assumes the presence of an admission controller at run-time.

Abeni and Buttazzo's work [AB99] addresses both scheduling and performance analysis of tasks with stochastic parameters. Their focus is on how to schedule both hard and soft real-time tasks on the same processor, in such a way that the hard ones are not disturbed by ill-behaved soft tasks. The performance analysis method is used to assess

their proposed scheduling policy (constant bandwidth server), and is restricted to the scope of their assumptions.

Tia *et al.* [TDS⁺95] assume a task model composed of independent tasks. Two methods for performance analysis are given. One of them is just an estimate and is demonstrated to be overly optimistic. In the second method, a soft task is transformed into a deterministic task and a sporadic one. The latter is executed only when the former exceeds the promised execution time. The sporadic tasks are handled by a server policy. The analysis is carried out on this particular model.

Zhou *et al.* [ZHS99] and Hu *et al.* [HZZ01] root their work in Tia's. However, they do not intend to give per-task guarantees, but characterise the fitness of the entire task set. Because they consider all possible combinations of execution times of all requests up to a time moment, the analysis can be applied only to very small task sets due to complexity reasons.

De Veciana *et al.* [dJG00] address a different type of problem. Having a task graph and an imposed deadline, their goal is to determine the path that has the highest probability to violate the deadline. In this case, the problem is reduced to a non-linear optimisation problem by using an approximation of the convolution of the probability densities.

Lehoczky [Leh96] models the task set as a Markovian process. The advantage of such an approach is that it is applicable to arbitrary scheduling policies. The process state space is the vector of lead-times (time left until the deadline). As this space is potentially infinite, Lehoczky analyses it in heavy traffic conditions, when the underlying stochastic process weakly converges to a reflected Brownian motion with drift. As far as we are aware, the heavy traffic theory fails yet to smoothly apply to real-time systems. Not only that there are cases when such a reflected Brownian motion with drift limit does not exist, as shown by Dai and Wang [DW93], but also the heavy traffic phenomenon is observed only for processor loads close to 1, leading to very long (infinite) queues of ready tasks and implicitly to systems with very large latency. This aspect makes the heavy traffic phenomenon undesirable in real-time systems.

Other researchers, such as Kleinberg *et al.* [KRT00] and Goel and Indyk [GI99], apply approximate solutions to problems exhibiting stochastic behaviour but in the context of load balancing, bin packing and knapsack problems. Moreover, the probability distributions they consider are limited to a few very particular cases.

Kim and Shin [KS96] considered applications implemented on multiprocessors and modelled them as queueing networks. They restricted the task execution times to exponentially distributed ones, which reduces the complexity of the analysis. The tasks were considered to be scheduled according to a particular policy, namely first-come-first-served

(FCFS). The underlying mathematical model is then the appealing continuous time Markov chain.

Díaz *et al.* [DGK⁺02] derive the expected deadline miss ratio from the probability distribution function of the response time of a task. The response time is computed based on the system-level backlog at the beginning of each hyperperiod, i.e. the residual execution times of the jobs at those time moments. The stochastic process of the system-level backlog is Markovian and its stationary solution can be computed. Díaz *et al.* consider only sets of independent tasks and the task execution times may assume values only over discrete sets. In their approach, complexity is mastered by trimming the transition probability matrix of the underlying Markov chain or by deploying iterative methods, both at the expense of result accuracy. According to the published results, the method is exercised only on extremely small task sets.

Kalavade and Moghé [KM98] consider task graphs where the task execution times are arbitrarily distributed over discrete sets. Their analysis is based on Markovian stochastic processes too. Each state in the process is characterised by the executed time and lead-time. The analysis is performed by solving a system of linear equations. Because the execution time is allowed to take only a finite (most likely small) number of values, such a set of equations is small.

Our work is mostly related to the ones of Zhou *et al.* [ZHS99], Hu *et al.* [HZS01], Kalavade and Moghé [KM98] and Díaz *et al.* [DGK⁺02]. It differs mostly by considering less restricted application classes. As opposed to Kalavade and Moghé's work and to Díaz *et al.*'s work, we consider *continuous* ETPDFs. In addition to Díaz *et al.*'s approach, we consider task sets with dependencies among tasks. Also, we accept a much larger class of scheduling policies than the fixed priority ones considered by Zhou and Hu. Moreover, our original way of concurrently constructing and analysing the underlying stochastic process, while keeping only the needed stochastic process states in memory, allows us to consider larger applications [MEP01].

6.3 Notation and problem formulation

This section introduces the notation used throughout the article and gives the problem formulation.

6.3.1 Notation

Let $T = \{\tau_1, \tau_2, \dots, \tau_N\}$ be a set of N tasks and $G = \{G_1, G_2, \dots, G_h\}$ denote h task graphs. A task graph $G_i = (V_i, E_i \subset V_i \times V_i)$ is a directed acyclic graph (DAG) whose set of vertices V_i is a non-empty subset of the set of tasks T . The sets V_i , $1 \leq i \leq h$, form a partition of T . There

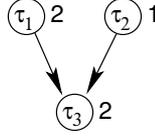


Figure 6.1: Application example

exists a directed edge $(\tau_j, \tau_k) \in E_i$ if and only if the task τ_k is dependent on the task τ_j . This dependency imposes that the task τ_k is executed only after the task τ_j has completed execution.

Let $G_i = (V_i, E_i)$ and $\tau_k \in V_i$. Then let ${}^\circ\tau_k = \{\tau_j : (\tau_j, \tau_k) \in E_i\}$ denote the set of predecessor tasks of the task τ_k . Similarly, let $\tau_k^\circ = \{\tau_j : (\tau_k, \tau_j) \in E_i\}$ denote the set of successor tasks of the task τ_k . If ${}^\circ\tau_k = \emptyset$ then task τ_k is a *root*. If $\tau_k^\circ = \emptyset$ then task τ_k is a *leaf*.

Let $\Pi = \{\pi_i \in \mathbb{N} : \tau_i \in T\}$ denote the set of *task periods*, or task inter-arrival times, where π_i is the period of task τ_i . Instantiation $u \in \mathbb{N}$ of task τ_i demands execution (is released) at time moment $u \cdot \pi_i$. The period π_i of any task τ_i is assumed to be a common multiple of all periods of its predecessor tasks (π_j divides π_i , where $\tau_j \in {}^\circ\tau_i$). Let k_{ij} denote $\frac{\pi_i}{\pi_j}$, $\tau_j \in {}^\circ\tau_i$. Instantiation $u \in \mathbb{N}$ of task τ_i may start executing only if instantiations $u \cdot k_{ij}, u \cdot k_{ij} + 1, \dots, (u + 1) \cdot k_{ij} - 1$ of tasks $\tau_j, \forall \tau_j \in {}^\circ\tau_i$, have completed.

Let us consider the example in Figure 6.1. The circles indicate the tasks, the numbers outside the circles indicate the task periods. In this example, instantiation 0 of task τ_1 and instantiations 0 and 1 of task τ_2 have to complete in order instantiation 0 of task τ_3 to be ready to run. Instantiation 1 of task τ_1 and instantiations 2 and 3 of task τ_2 have to complete in order instantiation 1 of task τ_3 to be ready to run. However, there is no execution precedence constraint between instantiation 0 of task τ_1 and instantiations 0 and 1 of task τ_2 on one hand and instantiation 1 of task τ_3 on the other hand.

π_{G_j} will denote the period of the task graph G_j and π_{G_j} is equal to the least common multiple of all π_i , where π_i is the period of τ_i and $\tau_i \in V_j$.

The model, where task periods are integer multiples of the periods of predecessor tasks, is more general than the model assuming equal task periods for tasks in the same task graph. This is appropriate, for instance, when modelling protocol stacks. For example, let us consider a part of baseband processing on the GSM radio interface [MP92]. A data frame is assembled out of 4 radio bursts. One task implements the decoding of radio bursts. Each time a burst is decoded, the result is sent to the frame assembling task. Once the frame assembling task gets all the needed data, that is every 4 invocations of the burst decoding task,

the frame assembling task is invoked. This way of modelling is more modular and natural than a model assuming equal task periods which would have crammed the four invocations of the radio burst decoding task in one task. We think that more relaxed models, with regard to relations between task periods, are not necessary, as such applications would be more costly to implement and are unlikely to appear in common engineering practice.

The real-time requirements are expressed in terms of relative deadlines. Let $\Delta_T = \{\delta_i \in \mathbb{N} : \tau_i \in T\}$ denote the set of *task deadlines*. δ_i is the deadline of task τ_i . Let $\Delta_G = \{\delta_{G_j} \in \mathbb{N} : 1 \leq j \leq h\}$ denote the set of task graph deadlines, where δ_{G_j} is the deadline of task graph G_j . If there is at least one task $\tau_i \in V_j$ that has missed its deadline δ_i , then the entire graph G_j missed its deadline.

If $D_i(t)$ denotes the number of missed deadlines of the task τ_i (or of the task graph G_i) over a time span t and $A_i(t)$ denotes the number of instantiations of task τ_i (task graph G_i) over the same time span, then $\lim_{t \rightarrow \infty} \frac{D_i(t)}{A_i(t)}$ denotes the *expected deadline miss ratio* of task τ_i (task graph G_i).

Let Ex_i denote an execution time of an instantiation of the task τ_i . Let $ET = \{\epsilon_1, \epsilon_2, \dots, \epsilon_N\}$ denote a set of N *execution time probability density functions*. ϵ_i is the probability density of the execution time of task τ_i . The execution times are assumed to be statistically independent. All the tasks are assumed to execute on the same processor. In this case, the inter-task communication time is comprised in the task execution time.

If a task misses its deadline, the real-time operating system takes a decision based on a designer-supplied *late task policy*. Let $Bounds = \{b_1, b_2, \dots, b_h\}$ be a set of h integers greater than 0. The late task policy specifies that at most b_i instantiations of the task graph G_i are allowed in the system at any time. If an instantiation of graph G_i demands execution when b_i instantiations already exist in the system, the instantiation with the earliest arrival time is discarded (eliminated) from the system. An alternative to this late task policy will be discussed in Section 6.5.5

In the common case of more than one task mapped on the same processor, the designer has to decide on a *scheduling policy*. Such a scheduling policy has to be able to unambiguously determine the running task at any time on that processor.

Let an *event* denote a task arrival or discarding. In order to be acceptable in the context described in this article, a scheduling policy is assumed to preserve the sorting of tasks according to their execution priority between consecutive events (the priorities are allowed to change in time, but the sorting of tasks according to their priorities is allowed to

change only at event times). All practically used priority based scheduling policies [LL73, But97, Fid98, ABD⁺95], both with static priority assignment (rate monotonic, deadline monotonic) and with dynamic assignment (earlier deadline first (EDF), least laxity first (LLF)), fulfill this requirement.

The scheduling policy is nevertheless restricted to non-preemptive scheduling. This limitation is briefly discussed in Section 6.4.1.

6.3.2 Problem formulation

Input

The following data is given as an input to the analysis procedure:

- The set of task graphs G ,
- The set of task periods Π ,
- The set of task deadlines Δ_T and the set of task graph deadlines Δ_G ,
- The set of execution time probability density functions ET ,
- The late task policy $Bounds$, and
- The scheduling policy.

Output

The result of the analysis is the set $Missed_T = \{m_{\tau_1}, m_{\tau_2}, \dots, m_{\tau_N}\}$ of expected deadline miss ratios for each task and the set $Missed_G = \{m_{G_1}, m_{G_2}, \dots, m_{G_h}\}$ of expected deadline miss ratios for each task graph.

6.4 Analysis algorithm

The goal of the analysis is to obtain the expected deadline miss ratios of the tasks and task graphs. These can be derived from the behaviour of the system. The behaviour is defined as the evolution of the system through a *state space* in time. A *state* of the system is given by the values of a set of variables that characterise the system. Such variables may be the currently running task, the set of ready tasks, the current time and the start time of the current task, etc.

Due to the considered periodic task model, the task arrival times are deterministically known. However, because of the stochastic task execution times, the completion times and implicitly the running task at an arbitrary time instant or the state of the system at that instant

cannot be deterministically predicted. The mathematical abstraction best suited to describe and analyse such a system with random character is the stochastic process.

In this section, we first sketch the stochastic process construction and analysis procedure based on a simplified example. Then the memory efficient construction of the stochastic process underlying the application is detailed. Third, the algorithm is refined in order to handle multiple concurrently active instantiations of the same task graph. Finally, the complete algorithm is presented.

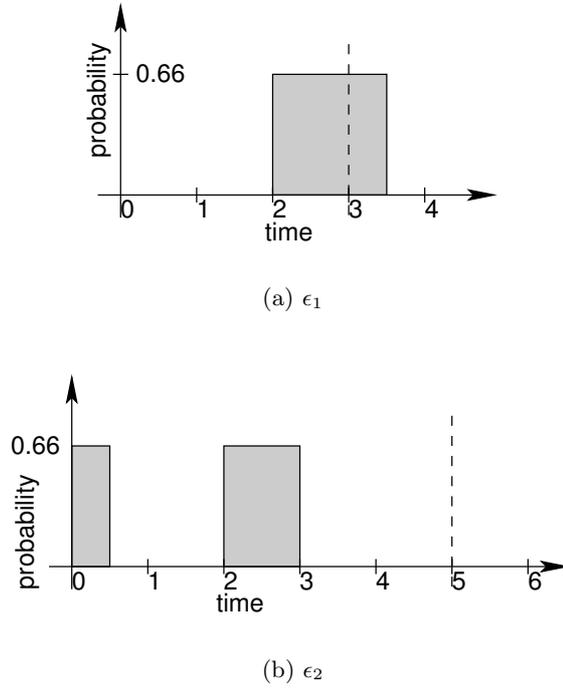
6.4.1 The underlying stochastic process

Let us define LCM as the least common multiple of the task periods. For simplicity of the exposition, we first assume that at most one instantiation of each task graph is tolerated in the system at the same time ($b_i = 1, \forall 1 \leq i \leq h$). In this case, all the late tasks are discarded at the time moments $LCM, 2 \cdot LCM, \dots, k \cdot LCM, \dots$ because at these moments new instantiations of all tasks arrive. The system behaves at these time moments as if it has just been started. The time moments $k \cdot LCM, k \in \mathbb{N}$ are called renewal points. Regardless of the chosen definition of the state space of the system, the system states at the renewal points are equivalent to the initial state which is unique and deterministically known. Thus, the behaviour of the system over the intervals $[k \cdot LCM, (k + 1) \cdot LCM), k \in \mathbb{N}$, is statistically equivalent to the behaviour over the time interval $[0, LCM)$. Therefore, in the case when $b_i = 1, 1 \leq i \leq h$, it is sufficient to analyse the system solely over the time interval $[0, LCM)$.

One could choose the following state space definition: $S = \{(\tau, W, t) : \tau \in T, W \in \text{set of all multisets of } T, t \in \mathbb{R}\}$, where τ represents the currently running task, W stands for the multiset of ready tasks at the start time of the running task, and t represents the start time of the currently running task. A state change occurs at the time moments when the scheduler has to decide on the next task to run. This happens

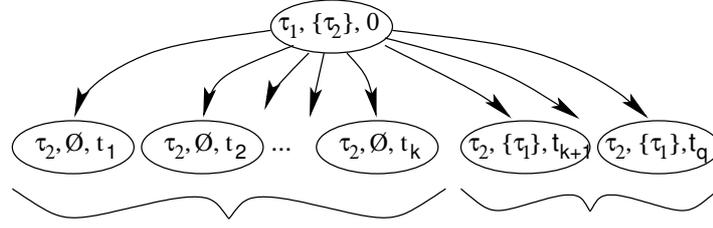
- when a task completes its execution, or
- when a task arrives and the processor is idling, or
- when the running task graph has to be discarded.

The point we would like to make is that, by choosing this state space, the information provided by a state $s_i = (\tau_i, W_i, t_i)$, together with the current time, is sufficient to determine the next system state $s_j = (\tau_j, W_j, t_j)$. The time moment when the system entered state s_i , namely t_i , is included in s_i . Because of the deterministic arrival times of tasks, based on the time moments t_j and on t_i , we can derive the

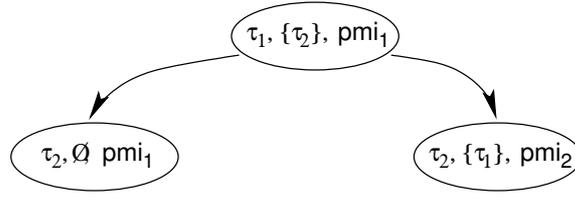
Figure 6.2: ETPDFs of tasks τ_1 (ϵ_1) and τ_2 (ϵ_2)

multiset of tasks that arrived in the interval $(t_i, t_j]$. The multiset of ready tasks at time moment t_i , namely W_i , is also known. We also know that τ_i is not preempted between t_i and t_j . Therefore, the multiset of ready tasks at time moment t_j , prior to choosing the new task to run, is the union of W_i and the tasks arrived during the interval $(t_i, t_j]$. Based on this multiset and on the time t_j , the scheduler is able to predictably choose the new task to run. Hence, in general, knowing a current state s and the time moment t when a transition out of state s occurs, the next state s' is unambiguously determined.

The following example is used throughout this subsection in order to discuss the construction of the stochastic process. The system consists of one processor and the following application: $G = \{(\{\tau_1\}, \emptyset), (\{\tau_2\}, \emptyset)\}$, $\Pi = \{3, 5\}$, i.e. a set of two independent tasks with corresponding periods 3 and 5. The tasks are scheduled according to a non-preemptive EDF scheduling policy [LL73]. In this case, *LCM*, the least common multiple of the task periods is 15. For simplicity, in this example it is assumed that the relative deadlines equal the corresponding periods ($\delta_i = \pi_i$). The ETPDFs of the two tasks are depicted in Figure 6.2. Note that ϵ_1 contains execution times larger than the deadline δ_1 .



(a) Individual task completion times



(b) Intervals containing task completion times

Figure 6.3: State encoding

Let us assume a state representation like the one introduced above: each process state contains the identity of the currently running task, its start time and the multiset of ready task at the start time of the currently running one. For our example application, the initial state is $(\tau_1, \{\tau_2\}, 0)$, i.e. task τ_1 is running, it has started to run at time moment 0 and task τ_2 is ready to run, as shown in Figure 6.3(a). t_1, t_2, \dots, t_q in the figure are possible finishing times for the task τ_1 and, implicitly, possible starting times of the waiting instantiation of task τ_2 . The number of next states equals the number of possible execution times of the running task in the current state. In general, because the ETPDFs are continuous, the set of state transition moments form a dense set in \mathbb{R} leading to an underlying stochastic process theoretically of uncountable state space. In practice, the stochastic process is extremely large, depending on the discretisation resolution of the ETPDFs. Even in the case when the task execution time probabilities are distributed over a discrete set, the resulting underlying process becomes prohibitively large and practically impossible to solve.

In order to avoid the explosion of the underlying stochastic process, in our approach, we have grouped time moments into equivalence classes and, by doing so, we limited the process size explosion. Thus, practically, a set of equivalent states is represented as a single state in the stochastic process.

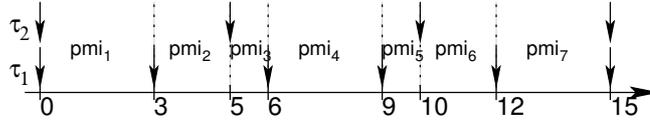


Figure 6.4: Priority monotonicity intervals

As a first step to the analysis, the interval $[0, LCM)$ is partitioned in disjunct intervals, the so-called *priority monotonicity intervals (PMI)*. A PMI is delimited by task arrival times and task execution deadlines. Figure 6.4 depicts the PMIs for the example above. The only restriction imposed on the scheduling policies accepted by our approach is that inside a PMI the ordering of tasks according to their priorities is not allowed to change. This allows the scheduler to predictably choose the next task to run regardless of the completion time within a PMI of the previously running task. All the widely used scheduling policies we are aware of (rate monotonic (RM), EDF, first come first served (FCFS), LLF, etc.) exhibit this property.

Consider a state s characterised by (τ_i, W, t) : τ_i is the currently running task, it has been started at time t , and W is the multiset of ready tasks. Let us consider two next states derived from s : s_1 characterised by (τ_j, W_1, t_1) and s_2 by (τ_k, W_2, t_2) . Let t_1 and t_2 belong to the same PMI. This means that no task instantiation has arrived or been discarded in the time interval between t_1 and t_2 , and the relative priorities of the tasks inside the set W have not changed between t_1 and t_2 . Thus, $\tau_j = \tau_k =$ the highest priority task in the multiset W , and $W_1 = W_2 = W \setminus \{\tau_j\}$. It follows that all states derived from state s that have their time t belonging to the same PMI have an identical currently running task and identical sets of ready tasks. Therefore, instead of considering individual times we consider time intervals, and we group together those states that have their associated start time inside the same PMI. With such a representation, the number of next states of a state s equals the number of PMIs the possible execution time of the task that runs in state s is spanning over.

We propose a representation in which a stochastic process state is a triplet (τ, W, pmi) , where τ is the running task, W the multiset of ready tasks, and pmi is the PMI containing the start time of the running task. In our example, the execution time of task τ_1 (which is in the interval $[2, 3.5]$, as shown in Figure 6.2(a)) is spanning over the PMIs pmi_1 — $[0, 3]$ —and pmi_2 — $[3, 5]$. Thus, there are only two possible states emerging from the initial state, as shown in Figure 6.3(b).

Figure 6.5 depicts a part of the stochastic process constructed for our example. The initial state is $s_1 : (\tau_1, \{\tau_2\}, pmi_1)$. The first field indicates that an instantiation of task τ_1 is running. The second field indicates

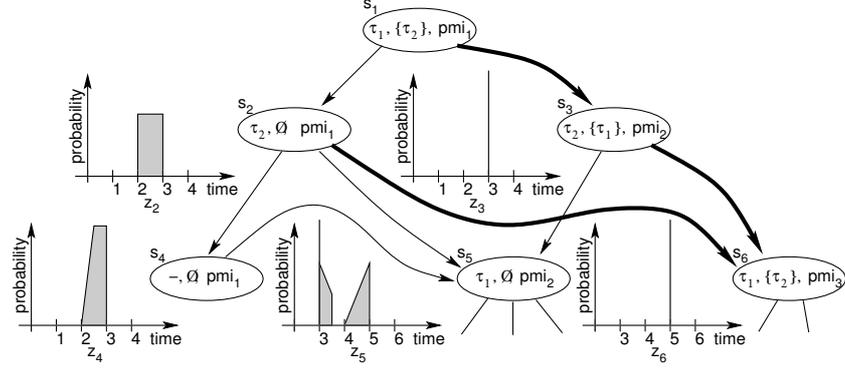


Figure 6.5: Stochastic process example

that an instantiation of task τ_2 is ready to execute. The third field shows the current PMI ($pmi_1 = [0, 3)$). If the instantiation of task τ_1 does not complete until time moment 3, then it will be discarded. The state s_1 has two possible next states. The first one is state $s_2 : (\tau_2, \emptyset, pmi_1)$ and corresponds to the case when the τ_1 completes before time moment 3. The second one is state $s_3 : (\tau_2, \{\tau_1\}, pmi_2)$ and corresponds to the case when τ_1 was discarded at time moment 3. State s_2 indicates that an instantiation of task τ_2 is running (it is the instance that was waiting in state s_1), that the PMI is $pmi_1 = [0, 3)$ —and that no task is waiting. Consider state s_2 to be the new current state. Then the next states could be state $s_4 : (-, \emptyset, pmi_1)$ (task τ_2 completed before time moment 3 and the processor is idle), state $s_5 : (\tau_1, \emptyset, pmi_2)$ (task τ_2 completed at a time moment sometime between 3 and 5), or state $s_6 : (\tau_1, \{\tau_2\}, pmi_3)$ (the execution of task τ_2 reached over time moment 5 and, hence, it was discarded at time moment 5). The construction procedure continues until all possible states corresponding to the time interval $[0, LCM)$, i.e. $[0, 15)$ have been visited.

Let \mathcal{P}_i denote the set of predecessor states of a state s_i , i.e. the set of all states that have s_i as a next state. The set of successor states of a state s_i consists of those states that can directly be reached from state s_i . Let Z_i denote the time when state s_i is entered. State s_i can be reached from any of its predecessor states $s_j \in \mathcal{P}_i$. Therefore, the probability $P(Z_i \leq t)$ that state s_i is entered before time t is a weighted sum over j of probabilities that the transitions $s_j \rightarrow s_i$, $s_j \in \mathcal{P}_i$, occur before time t . The weights are equal to the probability $P(s_j)$ that the system was in state s_j prior to the transition. Formally, $P(Z_i \leq t) = \sum_{j \in \mathcal{P}_i} P(Z_{ji} \leq t | s_j) \cdot P(s_j)$, where Z_{ji} is the time of transition $s_j \rightarrow s_i$. Let us focus on Z_{ji} , the time of transition $s_j \rightarrow s_i$. If the state transition occurs because the processor is idle and a new task arrives or because the running task graph has to be discarded, the time of the transition is

deterministically known as task arrivals and deadlines have fixed times. If, however, the cause of the state transition is a task completion, the time Z_{ji} is equal to $Z_j + Ex_\tau$, where task τ is the task which runs in state s_j and whose completion triggers the state transition. Because Z_{ji} is a sum involving the random variable Ex_τ , Z_{ji} too is a random variable. Its probability density function, is computed as the convolution $z_j * \epsilon_\tau = \int_0^\infty z_j(t-x) \cdot \epsilon_\tau(x) dx$ of the probability density functions of the terms.

Let us illustrate the above based on the example depicted in Figure 6.5. z_2, z_3, z_4, z_5 , and z_6 are the probability density functions of Z_2, Z_3, Z_4, Z_5 , and Z_6 respectively. They are shown in Figure 6.5 to the left of their corresponding states s_2, s_3, \dots, s_6 . The transition from state s_4 to state s_5 occurs at a precisely known time instant, time 3, at which a new instantiation of task τ_1 arrives. Therefore, z_5 will contain a scaled Dirac impulse at the beginning of the corresponding PMI. The scaling coefficient equals the probability of being in state s_4 (the integral of z_4 , i.e. the shaded surface below the z_4 curve). The probability density function z_5 results from the superposition of $z_2 * \epsilon_2$ (because task τ_2 runs in state s_2) with $z_3 * \epsilon_2$ (because task τ_2 runs in state s_3 too) and with the aforementioned scaled Dirac impulse over pmi_2 , i.e. over the time interval $[3, 5)$.

The probability of a task missing its deadline is easily computed from the transition probabilities of those transitions that correspond to a deadline miss of a task instantiation (the thick arrows in Figure 6.5, in our case). The probabilities of the transitions out of a state s_i are computed exclusively from the information stored in that state s_i . For example, let us consider the transition $s_2 \rightarrow s_6$. The system enters state s_2 at a time whose probability density is given by z_2 . The system takes the transition $s_2 \rightarrow s_6$ when the attempted completion time of τ_2 (running in s_2) exceeds 5. The completion time is the sum of the starting time of τ_2 (whose probability density is given by z_2) and the execution time of τ_2 (whose probability density is given by ϵ_2). Hence, the probability density of the completion time of τ_2 is given by the convolution $z_2 * \epsilon_2$ of the above mentioned densities. Once this density is computed, the probability of the completion time being larger than 5 is easily computed by integrating the result of the convolution over the interval $[5, \infty)$. If τ_2 in s_2 completes its execution at some time $t \in [3, 5)$, then the state transition $s_2 \rightarrow s_5$ occurs (see Figure 6.5). The probability of this transition is computed by integrating $z_2 * \epsilon_2$ over the interval $[3, 5)$.

As can be seen, by using the PMI approach, some process states have more than one incident arc, thus keeping the graph “narrow”. This is because, as mentioned, one process state in our representation captures

several possible states of a representation considering individual times (see Figure 6.3(a)).

The non-preemption limitation could, in principle, be overcome if we extended the information stored in the state of the underlying stochastic process. Namely, the *residual* run time probability distribution function of a task instantiation, i.e. the PDF of the time a preempted instantiation still has to run, has to be stored in the stochastic process state. This would several times multiply the memory requirements of the analysis. Additionally, preemption would increase the possible behaviours of the system and, consequently, the number of states of its underlying stochastic process.

Because the number of states grows rapidly even with our state reduction approach and each state has to store its probability density function, the memory space required to store the whole process can become prohibitively large. Our solution to master memory complexity is to perform the stochastic process construction and analysis simultaneously. As each arrow updates the time probability density z of the state it leads to, the process has to be constructed in topological order. The result of this procedure is that the process is never stored entirely in memory but rather that a *sliding window of states* is used for analysis. For the example in Figure 6.5, the construction starts with state s_1 . After its next states (s_2 and s_3) are created, their corresponding transition probabilities determined and the possible deadline miss probabilities accounted for, state s_1 can be removed from memory. Next, one of the states s_2 and s_3 is taken as current state, let us consider state s_2 . The procedure is repeated, states s_4 , s_5 and s_6 are created and state s_2 removed. At this moment, one would think that any of the states s_3 , s_4 , s_5 , and s_6 can be selected for continuation of the analysis. However, this is not the case, as not all the information needed in order to handle states s_5 and s_6 are computed. More exactly, the arcs emerging from states s_3 and s_4 have not yet been created. Thus, only states s_3 and s_4 are possible alternatives for the continuation of the analysis in topological order. The next section discusses the criteria for selection of the correct state to continue with.

6.4.2 Memory efficient analysis method

As shown in the example in Section 6.4.1, only a sliding window of states is simultaneously kept in memory. All states belonging to the sliding window are stored in a priority queue. Once a state is extracted from this queue and its information processed, it is eliminated from the memory. The key to the process construction in topological order lies in the order in which the states are extracted from this queue. First, observe that it is impossible for an arc to lead from a state with a PMI

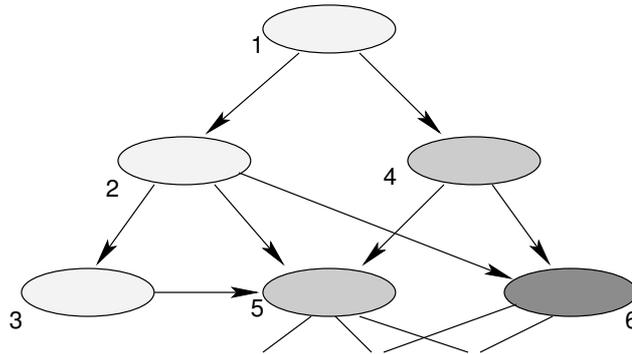


Figure 6.6: State selection order

number u to a state with a PMI number v such that $v < u$ (there are no arcs back in time). Hence, a first criterion for selecting a state from the queue is to select the one with the smallest PMI number. A second criterion determines which state has to be selected out of those with the same PMI number. Note that inside a PMI no new task instantiation can arrive, and that the task ordering according to their priorities is unchanged. Thus, it is impossible that the next state s_k of a current state s_j would be one that contains waiting tasks of higher priority than those waiting in s_j . Hence, the second criterion reads: among states with the same PMI, one should choose the one with the waiting task of highest priority. Figure 6.6 illustrates the algorithm on the example given in Section 6.4.1 (Figure 6.5). The shades of the states denote their PMI number. The lighter the shade, the smaller the PMI number. The numbers near the states denote the sequence in which the states are extracted from the queue and processed.

6.4.3 Flexible discarding

The examples considered so far dealt with applications where at most one active instance of each task graph is allowed at any moment of time ($b_i = 1$, $1 \leq i \leq h$).

In order to illustrate the construction of the stochastic process in the case $b_i > 1$, when several instantiations of a task graph G_i may exist at the same time in the system, let us consider an application consisting of two independent tasks, τ_1 and τ_2 , with periods 2 and 4 respectively. $LCM = 4$ in this case. The tasks are scheduled according to a rate monotonic (RM) policy [LL73]. At most one active instantiation of τ_1 is tolerated in the system at a certain time ($b_1 = 1$) and at most two concurrently active instantiations of τ_2 are tolerated in the system ($b_2 = 2$).

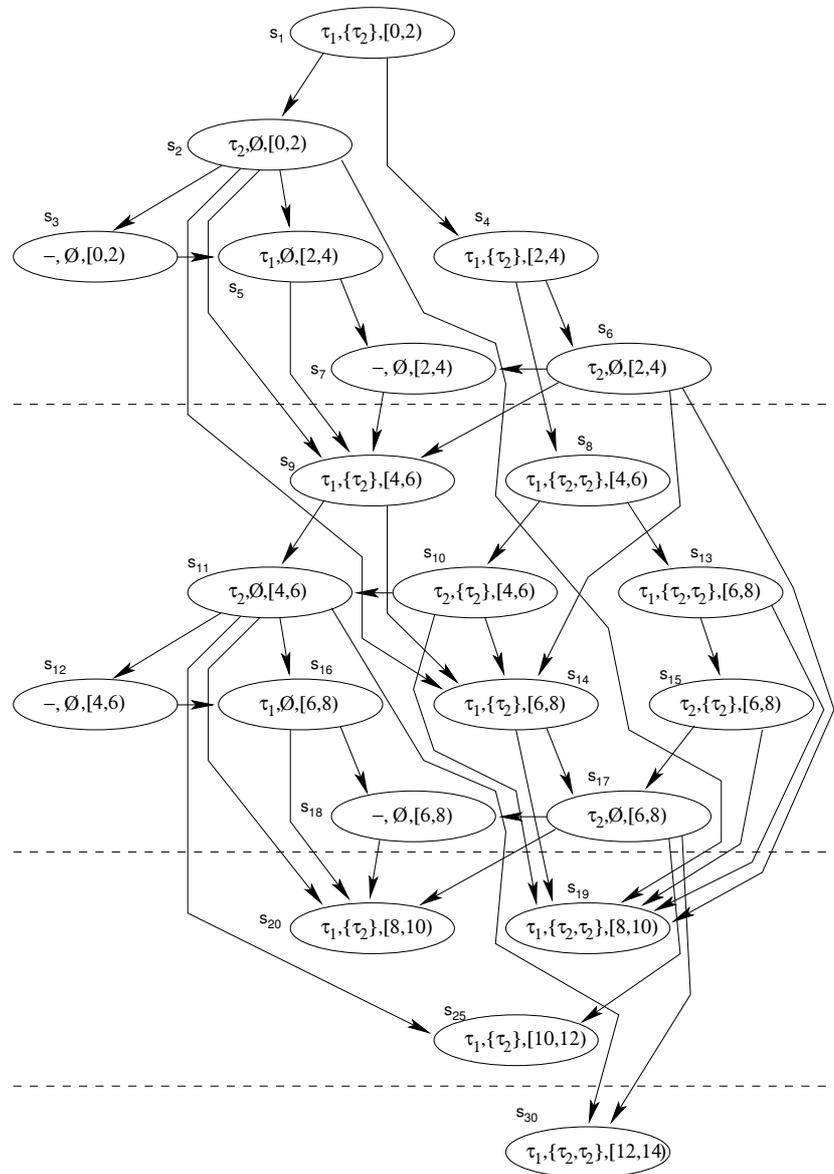


Figure 6.7: Part of the stochastic process underlying the example application

Figure 6.7 depicts a part of the stochastic process underlying this example. It is constructed using the procedure sketched in Sections 6.4.1 and 6.4.2. The state indexes show the order in which the states were analysed (extracted from the priority queue mentioned in Section 6.4.2).

Let us consider state $s_6 = (\tau_2, \emptyset, [2, 4))$, i.e. the instantiation of τ_2 that arrives at time moment 0 has been started at a moment inside the PMI $[2, 4)$ and there have not been any ready tasks at the start time of τ_2 . Let us assume that the finishing time of τ_2 lies past the $LCM = 4$. At time moment 4, a new instantiation of τ_2 arrives and the running instantiation is *not* discarded, as $b_2 = 2$. On one hand, if the finishing time of the running instantiation belongs to the interval $[6, 8)$, the system performs the transition $s_6 \rightarrow s_{14}$ (Figure 6.7). If, on the other hand, the running instantiation attempts to run past the time moment 8, then at this time moment a *third* instantiation of τ_2 would require service from the system and, therefore, the running task (the oldest instantiation of τ_2) is eliminated from the system. The transition $s_6 \rightarrow s_{19}$ in the stochastic process in Figure 6.7 corresponds to this latter case. We observe that when a task execution spans beyond the time moment LCM , the resulting state is not unique. The system does not behave as if it has just been restarted at time moment LCM , and, therefore, the intervals $[k \cdot LCM, (k+1) \cdot LCM)$, $k \in \mathbb{N}$, are not statistically equivalent to the interval $[0, LCM)$. Hence, it is not sufficient to analyse the system over the interval $[0, LCM)$ but rather over several consecutive intervals of length LCM .

Let an interval of the form $[k \cdot LCM, (k+1) \cdot LCM)$ be called the *hyperperiod* k and denoted H_k . $H_{k'}$ is a *lower* hyperperiod than H_k ($H_{k'} < H_k$) if $k' < k$. Consequently, H_k is a *higher* hyperperiod than $H_{k'}$ ($H_k > H_{k'}$) if $k > k'$.

For brevity, we say that a state s belongs to a hyperperiod k (denoted $s \in H_k$) if its PMI field is a subinterval of the hyperperiod k . In our example, three hyperperiods are considered, $H_0 = [0, 4)$, $H_1 = [4, 8)$, and $H_2 = [8, 12)$. In the stochastic process in Figure 6.7, $s_1, s_2, \dots, s_7 \in H_0$, $s_8, s_9, \dots, s_{18} \in H_1$, and $s_{19}, s_{20}, s_{25} \in H_2$ (note that not all states have been depicted in Figure 6.7).

In general, let us consider a state s and let \mathcal{P}_s be the set of its predecessor states. Let k denote the *order* of the state s defined as the lowest hyperperiod of the states in \mathcal{P}_s ($k = \min\{j : s' \in H_j, s' \in \mathcal{P}_s\}$). If $s \in H_k$ and s is of order k' and $k' < k$, then s is a *back state*. In our example, s_8, s_9, s_{14} , and s_{19} are back states of order 0, while s_{20}, s_{25} and s_{30} are back states of order 1.

Obviously, there cannot be any transition from a state belonging to a hyperperiod H to a state belonging to a lower hyperperiod than H ($s \rightarrow s', s \in H_k, s' \in H_{k'} \Rightarrow H_k \leq H_{k'}$). Consequently, the set \mathcal{S} of all states belonging to hyperperiods greater or equal to H_k can be

constructed from the back states of an order smaller than k . We say that \mathcal{S} is *generated* by the aforementioned back states. For example, the set of all states $s_8, s_9, \dots, s_{18} \in H_1$ can be derived from the back states s_8, s_9 , and s_{14} of order 0. The intuition behind this is that back states are inheriting all the needed information across the border between hyperperiods.

Before continuing our discussion, we have to introduce the notion of *similarity* between states. We say that two states s_i and s_j are similar ($s_i \sim s_j$) if all the following conditions are satisfied:

1. The task which is running in s_i and the one in s_j are the same,
2. The multiset of ready tasks in s_i and the one in s_j are the same,
3. The PMIs in the two states differ only by a multiple of LCM , and
4. $z_i = z_j$ (z_i is the probability density function of the times when the system takes a transition to s_i).

Let us consider the construction and analysis of the stochastic process, as described in Sections 6.4.1 and 6.4.2. Let us consider the moment x , when the last state belonging to a certain hyperperiod H_k has been eliminated from the sliding window. R_k is the set of back states stored in the sliding window at the moment x . Let the analysis proceed with the states of the hyperperiod H_{k+1} and let us consider the moment y when the last state belonging to H_{k+1} has been eliminated from the sliding window. Let R_{k+1} be the set of back states stored in the sliding window at moment y .

If the sets R_k and R_{k+1} contain pairwise similar states, then it is guaranteed that R_k and R_{k+1} generate identical stochastic processes during the rest of the analysis procedure (as stated, at a certain moment the set of back states unambiguously determines the rest of the stochastic process). In our example, $R_0 = \{s_8, s_9, s_{14}, s_{19}\}$ and $R_1 = \{s_{19}, s_{20}, s_{25}, s_{30}\}$. If $s_8 \sim s_{19}$, $s_9 \sim s_{20}$, $s_{14} \sim s_{25}$, and $s_{19} \sim s_{30}$ then the analysis process may stop as it reached convergence.

Consequently, the analysis proceeds by considering states of consecutive hyperperiods until the information captured by the back states in the sliding window does not change any more. Whenever the underlying stochastic process has a steady state, this steady state is guaranteed to be found.

6.4.4 Construction and analysis algorithm

The analysis is performed in two phases:

1. Divide the interval $[0, LCM)$ in PMIs,

2. Construct the stochastic process in topological order and analyse it.

The concept of PMI (called in their paper “state”) was introduced by Zhou *et al.* [ZHS99] in a different context, unrelated to the construction of a stochastic process. Let A denote the set of task arrivals in the interval $[0, LCM]$, i.e. $A = \{x | 0 \leq x \leq LCM, \exists 1 \leq i \leq N, \exists k \in \mathbb{N} : x = k\pi_i\}$. Let D denote the set of deadlines in the interval $[0, LCM]$, i.e. $D = \{x | 0 \leq x \leq LCM, \exists 1 \leq i \leq N, \exists k \in \mathbb{N} : x = k\pi_i + \delta_i\}$. The set of PMIs of $[0, LCM)$ is $\{[a, b) | a, b \in A \cup D \wedge \nexists x \in (A \cup D) \cap (a, b)\}$. If PMIs of a higher hyperperiod $H_k, k > 0$, are needed during the analysis, they are of the form $[a + k \cdot LCM, b + k \cdot LCM)$, where $[a, b)$ is a PMI of $[0, LCM)$.

The algorithm proceeds as discussed in Sections 6.4.1, 6.4.2 and 6.4.3. An essential point is the construction of the process in topological order, which allows only parts of the states to be stored in memory at any moment. The algorithm for the stochastic process construction is depicted in Figure 6.8.

A global priority queue stores the states in the sliding window. The state priorities are assigned as shown in Section 6.4.2. The initial state of the stochastic process is put in the queue. The explanation of the algorithm is focused on the `construct_and_analyse` procedure. Each invocation of this procedure constructs and analyses the part of the underlying stochastic process which corresponds to one hyperperiod H_k . It starts with hyperperiod H_0 ($k = 0$). The procedure extracts one state at a time from the queue. Let $s_j = (\tau_i, W_i, pm_i)$ be such a state. The probability density of the time when a transition occurs to s_j is given by the function z_j . The priority scheme of the priority queue ensures that s_j is extracted from the queue only after *all* the possible transitions to s_j have been considered, and thus z_j contains accurate information. In order to obtain the probability density of the time when task τ_i completes its execution, the probability density of its starting time (z_j) and the ETPDF of τ_i (ϵ_i) have to be convoluted. Let ξ be the probability density resulting from the convolution.

Figure 6.9 presents an algorithmic description of the `next_state` procedure. Based on ξ , the finishing time PDF of task τ_i if task τ_i is never discarded, we compute `max_exec_time`, the maximum execution time of task τ_i . `max_time` is the maximum between `max_exec_time` and the time at which task τ_i would be discarded. `PMI` will then denote the set of all PMIs included in the interval between the start of the PMI in which task τ_i started to run and `max_time`. Task τ_i could, in principle, complete its execution during any of these PMIs. We consider each PMI as being the one in which task τ_i finishes its execution. A new underlying stochastic process state corresponds to each of these possi-

```

divide  $[0, LCM)$  in PMIs;
 $pmi\_no$  = number of PMIs between 0 and  $LCM$ ;
put first state in the priority queue  $pqueue$ ;
 $R_{old} = \emptyset$ ;           //  $R_{old}$  is the set of densities  $z$ 
                        // of the back states after iteration  $k$ 
 $(R_{new}, Missed) = construct\_and\_analyse()$ ; //  $Missed$  is the set
                        // of expected deadline miss ratios
do
     $R_{old} = R_{new}$ ;
     $(R_{new}, Missed) = construct\_and\_analyse()$ ;
while  $R_{new} \neq R_{old}$ ;

construct_and_analyse:
while  $\exists s \in pqueue$  such that  $s.pmi \leq pmi\_no$  do
     $s_j = \text{extract state from } pqueue$ ;
     $\tau_i = s_j.running$ ;           // first field of the state
     $\xi = convolute(\epsilon_i, z_j)$ ;
     $nextstatelist = next\_states(s_j)$ ; // consider task dependencies!
    for each  $s_u \in nextstatelist$  do
        compute the probability of the transition
                                from  $s_j$  to  $s_u$  using  $\xi$ ;
        update deadline miss probabilities  $Missed$ ;
        update  $z_u$ ;
        if  $s_u \notin pqueue$  then
            put  $s_u$  in the  $pqueue$ ;
        end if;
        if  $s_u$  is a back state and  $s_u \notin R_{new}$  then
             $R_{new} = R_{new} \cup \{s_u\}$ ;
        end if;
    end for;
    delete state  $s_j$ ;
end while;
return  $(R_{new}, Missed)$ ;

```

Figure 6.8: Construction and analysis algorithm

```

next_states( $s_j = (\tau_i, W_i, t_i)$ ):
  nextstates =  $\emptyset$ ;
  max_exec_time =  $\sup\{t : \xi(t) > 0\}$ ;    // the largest finishing time
  of  $\tau_i$ 
  max_time =  $\max\{\text{max\_exec\_time}, \text{discarding\_time}_i\}$ ;    // the maximum
  between
                                     // finishing time and discarding time of  $\tau_i$ 
  PMI =  $\{[lo_p, hi_p) \in \text{PMIs} : lo_p \geq t_i \wedge hi_p \leq \text{max\_time}\}$     // the set
  of PMIs
                                     // included in the interval  $[t_i, \text{max\_time}]$ 
  for each  $[lo_p, hi_p) \in \text{PMI}$  do
    Arriv =  $\{\tau \in T : \tau \text{ arrived in the interval } [t_i, hi_p)\}$ ;
    Discarded =  $\{\tau \in W_i : \tau \text{ was discarded in the interval}$ 
   $[t_i, hi_p)\}$ ;
    Enabled =  $\{\tau \in T : \tau \text{ becomes ready to execute as an}$ 
  effect of  $\tau_i$ 's
    completion}
     $W = (W_i \setminus \text{Discarded}) \cup \text{Enabled} \cup \{\tau \in \text{Arriv} : \mathcal{P}_\tau = \emptyset\}$ ;
  // add the newly
                                     // arrived tasks with no predecessors, as
  they are
                                     // ready to execute, and the newly enabled
  ones
    select the new running task  $\tau_u$  from  $W$ 
    based on the scheduling policy
     $W_u = W \setminus \{\tau_u\}$ ;
    add  $(\tau_u, W_u, [lo_p, hi_p))$  to nextstatelist;
  done;
  return nextstates;

```

Figure 6.9: *next_states* procedure

ble finishing PMIs. For each PMI, we determine the multiset *Arriv* of newly arrived tasks while task τ_i was executing. Also, we determine the multiset *Discarded* of those tasks which were ready to execute when task τ_i started, but were discarded in the mean time, as the execution of task τ_i spanned beyond their deadlines. Once task τ_i completes its execution, some of its successor tasks may become ready to execute. These successor tasks which become ready to execute as a result of task τ_i 's completion form the set *Enabled*. The new multiset of ready tasks, W , is the union of the old multiset of ready tasks except the ones that were discarded during the execution of task τ_i , $W_i \setminus \text{Discarded}$, and the set *Enabled* and those newly arrived tasks which have no predecessor and therefore are immediately ready to run. Once the new set of ready tasks is determined, the new running task τ_u is selected from multiset W based on the scheduling policy of the application. A new stochastic process state $(\tau_u, W \setminus \{\tau_u\}, [lo_p, hi_p])$ is constructed and added to the list of next states.

The probability densities z_u of the times a transition to $s_u \in \text{next-statalist}$ is taken are updated based on ξ . The state s_u is then added to the priority queue and s_j removed from memory. This procedure is repeated until there is no task instantiation that started its execution in hyperperiod H_k (until no more states in the queue have their PMI field in the range $k \cdot \text{pmi_no}, \dots, (k + 1) \cdot \text{pmi_no}$, where pmi_no is the number of PMIs between 0 and LCM). Once such a situation is reached, partial results, corresponding to the hyperperiod H_k are available and the `construct_and_analyse` procedure returns. The `construct_and_analyse` procedure is repeated until the set of back states R does not change any more.

6.5 Experimental results

The most computation intensive part of the analysis is the computation of the convolutions $z_i * \epsilon_j$. In our implementation we used the FFTW library [FJ98] for performing convolutions based on the Fast Fourier Transform. The number of convolutions to be performed equals the number of states of the stochastic process. The memory required for analysis is determined by the maximum number of states in the sliding window. The main factors on which the size of the stochastic process depends are LCM (the least common multiple of the task periods), the number of PMIs, the number of tasks N , the task dependencies, and the maximum allowed number of concurrently active instantiations of the same task graph.

As the selection of the next running task is unique, given the pending tasks and the time moment, the particular scheduling policy has a

reduced impact on the process size. Hence, we use the non-preemptive EDF scheduling policy in the experiments below. On the other hand, the task dependencies play a significant role, as they strongly influence the set of ready tasks and, by this, the process size.

The ETPDFs were randomly generated. An interval $[E_{min}, E_{max}]$ has been divided into smaller intervals. For each of the smaller intervals, the ETPDF has a constant value, different from the value over other intervals. The curve shape has of course an influence on the final result of the analysis, but it has little or no influence on the analysis time and memory consumed by the analysis itself. The interval length $E_{max} - E_{min}$ influences the analysis time and memory, but only marginally.

The periods are randomly picked from a pool of periods with the restriction that the period of task τ has to be an integer multiple of the periods of the predecessors of task τ . The pool comprises periods in the range $2, 3, \dots, 24$. Large prime numbers have a lower probability to be picked, but it occurs nevertheless.

In the following, we report on six sets of experiments. The first four investigate the impact of the enumerated factors (LCM , the number N of tasks, the task dependencies, the maximum allowed number of concurrently active instantiations of the same task graph) on the analysis complexity. The fifth set of experiments considers a different problem formulation. The sixth experiment is based on a real-life example in the area of telecommunication systems.

The aspects of interest were the stochastic process size, as it determines the analysis execution time, and the maximum size of the sliding window, as it determines the memory space required for the analysis. Both the stochastic process size and the maximum size of the sliding window are expressed in number of states. All experiments were performed on an UltraSPARC 10 at 450 MHz.

6.5.1 Stochastic process size vs. number of tasks

In the first set of experiments we analysed the impact of the number of tasks on the process size. We considered task sets of 10 up to 19 independent tasks. LCM , the least common multiple of the task periods, was 360 for all task sets. We repeated the experiment four times for average values of the task periods $a = 15.0, 10.9, 8.8$, and 4.8 (keeping $LCM = 360$). The results are shown in Figure 6.10. Figure 6.11 depicts the maximum size of the sliding window for the same task sets. As it can be seen from the diagram, the increase, both of the process size and of the sliding window, is linear. The steepness of the curves depends on the task periods (which influence the number of PMIs). It is important to notice the big difference between the process size and the maximum number of states in the sliding window. In the case of 19 tasks, for example,

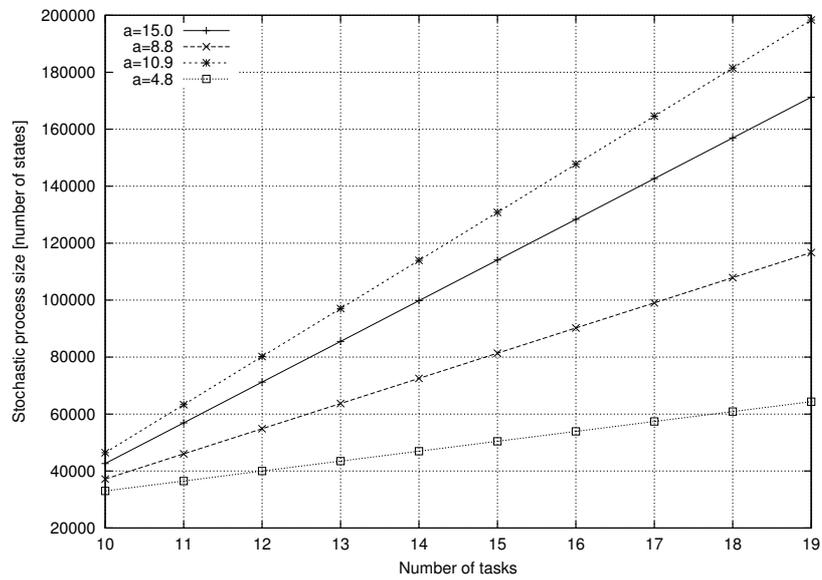


Figure 6.10: Stochastic process size vs. number of tasks

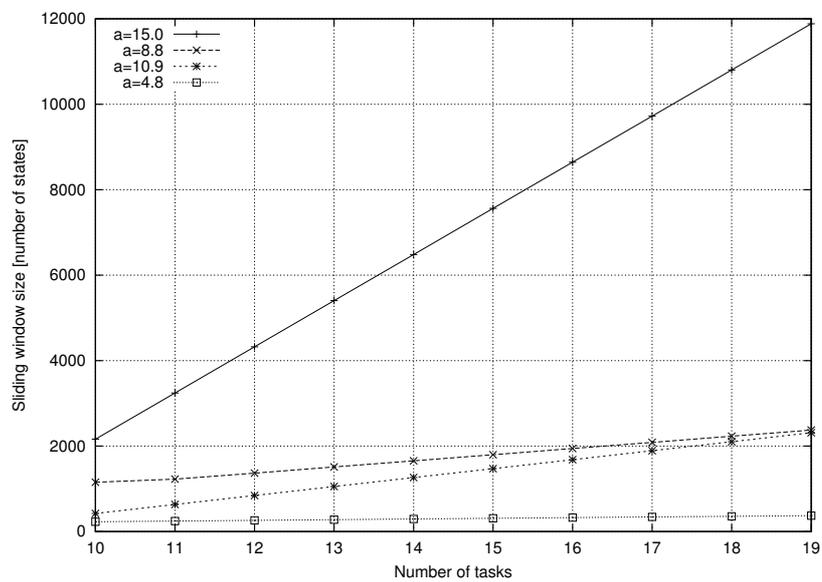


Figure 6.11: Size of the sliding window of states vs. number of tasks

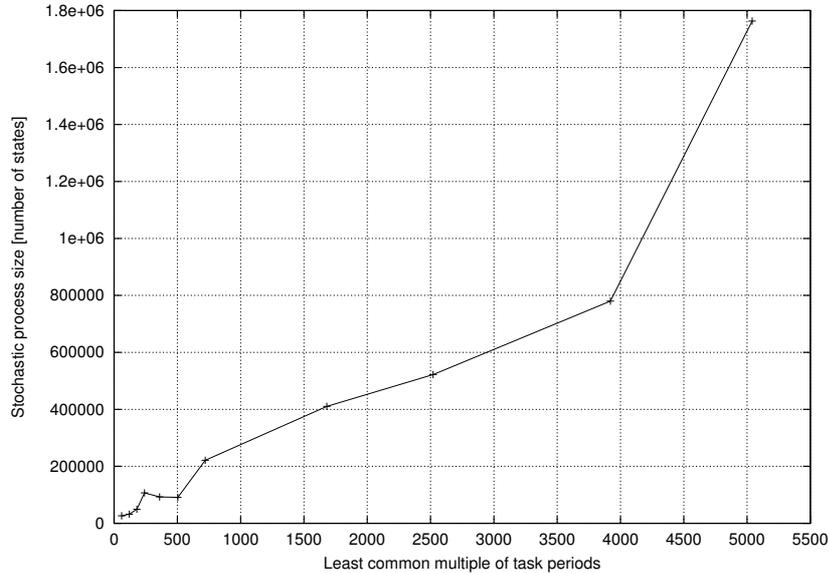


Figure 6.12: Stochastic process size vs. application period LCM

the process size is between 64356 and 198356 while the dimension of the sliding window varies between 373 and 11883 (16 to 172 times smaller). The reduction factor of the sliding window compared to the process size was between 15 and 1914, considering all our experiments.

6.5.2 Stochastic process size vs. application period LCM

In the second set of experiments we analysed the impact of the application period LCM (the least common multiple of the task periods) on the process size. We considered 784 sets, each of 20 independent tasks. The task periods were chosen such that LCM takes values in the interval $[1, 5040]$. Figure 6.12 shows the variation of the average process size with the application period.

6.5.3 Stochastic process size vs. task dependency degree

With the third set of experiments we analysed the impact of task dependencies on the process size. A task set of 200 tasks with strong dependencies (28000 arcs) among the tasks was initially created. The application period LCM was 360. Then 9 new task graphs were successively derived from the first one by uniformly removing dependencies between the tasks until we finally got a set of 200 independent tasks. The results are depicted in Figure 6.13 with a logarithmic scale for the y axis. The x axis represents the degree of dependencies among the

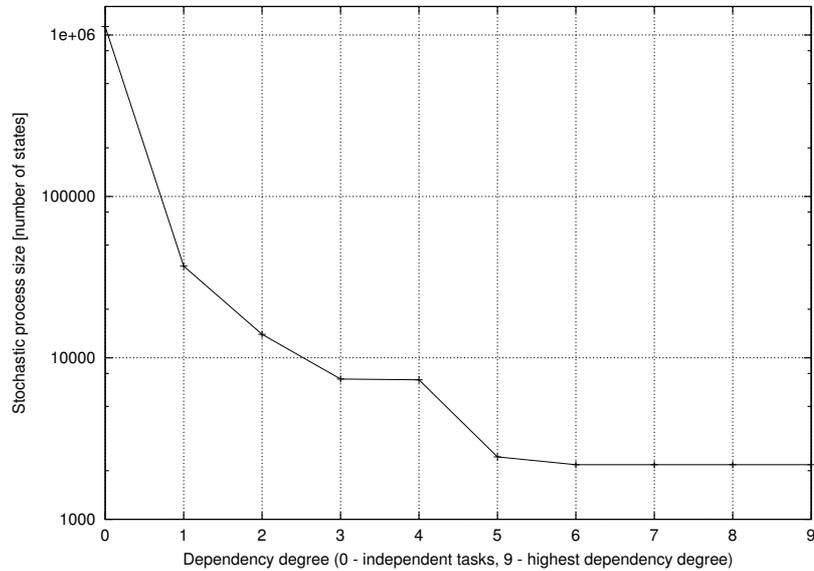


Figure 6.13: Stochastic process size vs. task dependency degree

tasks (0 for independent tasks, 9 for the initial task set with the highest amount of dependencies).

As mentioned, the execution time for the analysis algorithm strictly depends on the process size. Therefore, we showed all the results in terms of this parameter. For the set of 200 independent tasks used in this experiment (process size 1126517) the analysis time was 745 seconds. In the case of the same 200 tasks with strong dependencies (process size 2178) the analysis took 1.4 seconds.

6.5.4 Stochastic process size vs. average number of concurrently active instantiations of the same task graph

In the fourth set of experiments, the impact of the average number of concurrently active instantiations of the same task graph on the stochastic process size was analysed. 18 sets of task graphs containing between 12 and 27 tasks grouped in 2 to 9 task graphs were randomly generated. Each task set was analysed between 9 and 16 times considering different upper bounds for the maximum allowed number of concurrently active task graph instantiations. These upper bounds ranged from 1 to 3. The results were averaged for the same number of tasks. The dependency of the underlying stochastic process size as a function of the average of the maximum allowed number of instantiations of the same task graph that are concurrently active is plotted in Figure 6.14. Note that the y-axis is logarithmic. Different curves correspond to different sizes of the con-

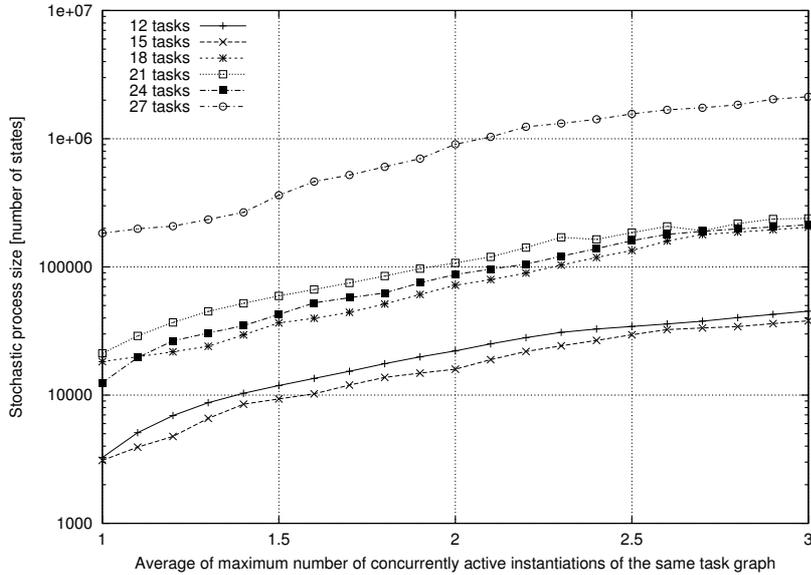


Figure 6.14: Stochastic process size vs. average number of concurrently active instantiations of the same task graph

sidered task sets. It can be observed that the stochastic process size is approximately linear in the average of the maximum allowed number of concurrently active instantiations of the same task graph.

6.5.5 Rejection versus discarding

As formulated in Section 6.3.1, when there are b_i concurrently active instantiations of task graph G_i in the system, and a new instantiation of G_i demands service, the oldest instantiation of G_i is eliminated from the system. Sometimes, such a strategy is not desired, as the oldest instantiation might have been very close to finishing, and by discarding it, the invested resources (time, memory, bandwidth, etc.) are wasted.

Therefore, our problem formulation has been extended to support a late task policy in which, instead of discarding the oldest instantiation of G_i , the newly arrived instantiation is denied service (rejected) by the system.

In principle, the rejection policy is easily supported by only changing the `next_states` procedure in the algorithm presented in Section 6.4.4. However, this has a strong impact on the analysis complexity as shown in Table 6.1. The significant increase in the stochastic process size (up to two orders of magnitude) can be explained considering the following example. Let s be the stochastic process state under analysis, let τ_j belonging to task graph G_i be the task running in s and let us consider that there are b_i concurrently active instantiations of G_i in the system.

The execution time of τ_j may be very large, spanning over many PMIs. In the case of discarding, it was guaranteed that τ_j will stop running after at most $b_i \cdot \pi_{G_i}$ time units, because at that time moment it would be eliminated from the system. Therefore, when considering the discarding policy, the number of next states of a state s is upper bounded. When considering the rejection policy, this is not the case any more.

Moreover, let us assume that b_i instantiations of the task graph G_i are active in the system at a certain time. In the case of discarding, capturing this information in the system state is sufficient to unambiguously identify those b_i instantiations: they are the last b_i that arrived, because always the oldest one is discarded. For example, the two ready instantiations of τ_2 in the state $s_{13} = (\tau_1, \{\tau_2, \tau_2\}, [6, 8])$ in Figure 6.7 are the ones that arrived at the time moments 0 and 4. However, when the rejection policy is deployed, just specifying that b_i instantiations are in the system is not sufficient for identifying them. We will illustrate this by means of the following example. Let $b_i = 2$, and let the current time be $k\pi_{G_i}$. In a first scenario, the oldest instantiation of G_i , which is still active, arrived at time moment $(k-5)\pi_{G_i}$ and it still runs. Therefore, the second oldest instantiation of G_i is the one that arrived at time moment $(k-4)\pi_{G_i}$ and all the subsequent instantiations were rejected. In a second scenario, the instantiation that arrived at time moment $(k-5)\pi_{G_i}$ completes its execution shortly before time moment $(k-1)\pi_{G_i}$. In this case, the instantiations arriving at $(k-3)\pi_{G_i}$ and $(k-2)\pi_{G_i}$ were rejected but the one arriving at $(k-1)\pi_{G_i}$ was not. In both scenarios, the instantiation arriving at $k\pi_{G_i}$ is rejected, as there are two concurrently active instantiations of G_i in the system, but these two instantiations cannot be determined without extending the definition of the stochastic process state space. Extending this space with the task graph arrival times is partly responsible for the increase in number of states of the underlying stochastic process.

The fifth set of experiments reports on the analysis complexity when the rejection policy is deployed. 101 task sets of 12 to 27 tasks grouped in 2 to 9 task graphs were randomly generated. For each task set two analysis were performed, one considering the discarding policy and the other considering the rejection policy. The results were averaged for task sets with the same cardinality and shown in Table 6.1.

6.5.6 Decoding of a GSM dedicated signalling channel

Finally, we present an example from industry, in particular the mobile communication area. Figure 6.15 depicts a set of 8 tasks that co-operate in order to decode the digital bursts corresponding to a GSM 900 signalling channel [MP92]. The incoming bursts are demodulated by the demodulation task, based on the frequency indexes generated by the

Tasks	Average stochastic process size [number of states]		Relative increase
	Discarding	Rejection	
12	2223.52	95780.23	42.07
15	7541.00	924548.19	121.60
18	4864.60	364146.60	73.85
21	18425.43	1855073.00	99.68
24	14876.16	1207253.83	80.15
27	55609.54	5340827.45	95.04

Table 6.1: Discarding compared to rejection

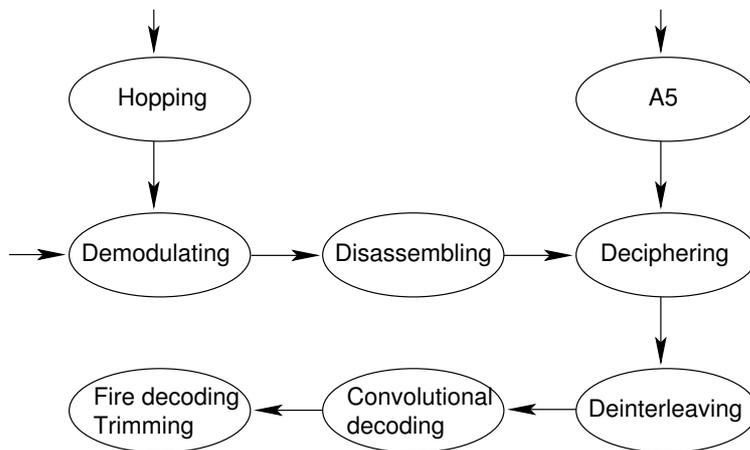


Figure 6.15: Decoding of a GSM dedicated signalling channel

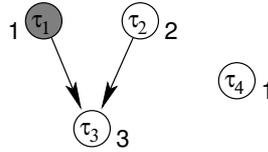


Figure 6.16: Example of multiprocessor application

frequency hopping task. The demodulated bursts are disassembled by the disassembling task. The resulting digital blocks are deciphered by the deciphering task based on a key generated by the A5 task. The deciphered block proceeds through bit deinterleaving, convolutional decoding (Viterbi decoding) and the so called fire decoding. The whole application runs on a single DSP processor and the tasks are scheduled according to fixed priority scheduling. All tasks have the same period, imposed by the TDMA scheme of the radio interface.

In this example, there are two sources of variation in execution times. The demodulating task has both data and control intensive behaviour, which can cause pipeline hazards on the deeply pipelined DSP it runs on. Its execution time probability density is derived from the input data streams and measurements. Another task will finally implement a deciphering unit. Due to the lack of knowledge about the deciphering algorithm A5 (its specification is not publicly available), the deciphering task execution time is considered to be uniformly distributed between an upper and a lower bound.

When two channels are scheduled on the DSP, the ratio of missed deadlines is 0 (all deadlines are met). Considering three channels assigned to the same processor, the analysis produced a ratio of missed deadlines, which was below the one enforced by the required QoS. It is important to note that using a hard real-time model with WCET, the system with three channels would result as unschedulable on the selected DSP. The underlying stochastic process for the three channels had 130 nodes and its analysis took 0.01 seconds. The small number of nodes is caused by the strong harmony among the task periods, imposed by the GSM standard.

6.6 Limitations and extensions

Although our proposed method is, as shown, efficiently applicable to the analysis of applications implemented on monoprocessor systems, it can handle only small scale multiprocessor applications. This section identifies the causes of this limitation and sketches an alternative approach to handle multiprocessor applications.

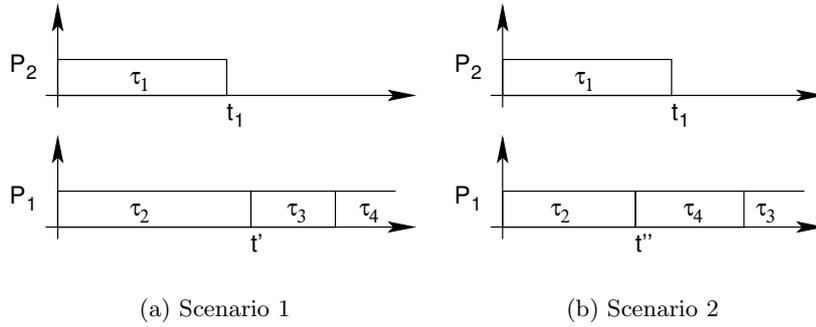


Figure 6.17: Two execution scenarios

When analysing multiprocessor applications, one approach could be to decompose the analysis problem into several subproblems, each of them analysing the tasks mapped on one of the processors. We could attempt to apply the presented approach in order to solve each of the subproblems. Unfortunately, in the case of multiprocessors and with the assumption of data dependencies among tasks, this approach cannot be applied. The reason is that the set of ready tasks cannot be determined based solely on the information regarding the tasks mapped on the processor under consideration. To illustrate this, let us consider the example in Figure 6.16. Tasks τ_2 , τ_3 , and τ_4 are mapped on processor P_1 and task τ_1 is mapped on processor P_2 . The numbers near the tasks indicate the task priorities. For simplicity, let us assume that all tasks have the same period π , and hence there is only one priority monotonicity interval $[0, \pi)$. Let us examine two possible scenarios. The corresponding Gantt diagrams are depicted in Figure 6.17. At time moment 0 task τ_1 starts running on processor P_2 and task τ_2 starts running on processor P_1 . Task τ_1 completes its execution at time moment $t_1 \in [0, \pi)$. In the first scenario, task τ_2 completes its execution at time moment $t' > t_1$ and task τ_3 starts executing on the processor P_1 at time moment t' because it has the highest priority among the two ready tasks τ_3 and τ_4 at that time. In the second scenario, task τ_2 completes its execution at time moment $t'' < t_1$. Therefore, at time moment t'' , only task τ_4 is ready to run and it will start its execution on the processor P_1 at that time. Thus, the choice of the next task to run is not independent of the time when the running task completes its execution inside a PMI. This makes the concept of PMIs unusable when looking at the processors in isolation.

An alternative approach would be to consider all the tasks and to construct the global state space of the underlying stochastic process accordingly. In principle, the approach presented in the previous sections

could be applied in this case. However, the number of possible execution traces, and implicitly the stochastic process, explodes due to the parallelism provided by the application platform. As shown, the analysis has to store the probability distributions z_i for each process state in the sliding window of states, leading to large amounts of needed memory and limiting the appropriateness of this approach to very small multiprocessor applications. Moreover, the number of convolutions $z_i * \epsilon_j$, being equal to the number of states, would also explode, leading to prohibitive analysis times.

We have addressed these problems [MEP02] by using an approximation approach for the task execution time probability distribution functions. Approximating the generalised ETPDFs with weighted sums of convoluted exponential functions leads to approximating the underlying generalised semi-Markov process with a continuous time Markov chain. By doing so, we avoid both the computation of convolutions and the storage of the z_i functions. However, as opposed to the method presented in this paper, which produces exact values for the expected deadline miss ratios, the alternative approach [MEP02] generates just approximations of the real ratios.

6.7 Conclusions

This work proposes a method for the schedulability analysis of task sets with probabilistically distributed task execution times. Our method improves the currently existing ones by providing *exact* solutions for *larger* and *less restricted* task sets. Specifically, we allow continuous task execution time probability distributions, and we do not restrict our approach to one particular scheduling policy. Additionally, task dependencies are supported, as well as arbitrary deadlines.

The analysis of task sets under such generous assumptions is made possible by three complexity management methods:

1. the introduction and exploitation of the PMI concept,
2. the concurrent construction and analysis of the stochastic process, and
3. the usage of a sliding window of states made possible by the construction in topological order.

As the presented experiments demonstrate, the proposed method can efficiently be applied to applications implemented on monoprocessor systems.

Bibliography

- [AB98a] L. Abeni and G. C. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th Real Time Systems Symposium*, pages 4–13, 1998.
- [AB98b] A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 123–132, 1998.
- [AB99] L. Abeni and G. Butazzo. QoS guarantee using probabilistic deadlines. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pages 242–249, 1999.
- [ABD⁺95] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Journal of Real-Time Systems*, 8(2-3):173–198, March-May 1995.
- [BLA98] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *Proceedings of the 19th Real Time Systems Symposium*, pages 286–295, 1998.
- [BLMSV98] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-Vincentelli. Scheduling for embedded real-time systems. *IEEE Design and Test of Computers*, pages 71–82, January–March 1998.
- [BS98] J. E. Beck and D. P. Siewiorek. Automatic configuration of embedded multicomputer systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(2):84–95, 1998.
- [But97] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer Academic, 1997.
- [DG97] G. De Micheli and R. K. Gupta. Hardware/software co-design. *Proceedings of the IEEE*, 85(3):349–365, March 1997.
- [DGK⁺02] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. Lo Bello, J. M. López, S. L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *Proceedings of the 23rd Real-Time Systems Symposium*, 2002.
- [dJG00] G. de Veciana, M. Jacome, and J.-H. Guo. Assessing probabilistic timing constraints on system performance. *Design*

- Automation for Embedded Systems*, 5(1):61–81, February 2000.
- [DW93] J. G. Dai and Y. Wang. Nonexistence of Brownian models for certain multiclass queueing networks. *Queueing Systems*, 13:41–46, 1993.
- [Ern98] R. Ernst. Codesign of embedded systems: Status and trends. *IEEE Design and Test of Computers*, pages 45–54, April-June 1998.
- [Fid98] C. J. Fidge. Real-time schedulability tests for preemptive multitasking. *Journal of Real-Time Systems*, 14(1):61–93, 1998.
- [FJ98] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 3, pages 1381–1384, 1998.
- [Gau98] H. Gautama. A probabilistic approach to the analysis of program execution time. Technical Report 1-68340-44(1998)06, Faculty of Information Technology and Systems, Delft University of Technology, 1998.
- [GI99] A. Goel and P. Indyk. Stochastic load balancing and related problems. In *IEEE Symposium on Foundations of Computer Science*, pages 579–586, 1999.
- [Gv00] H. Gautama and A. J. C. van Gemund. Static performance prediction of data-dependent programs. In *Proceedings of the 2nd International Workshop on Software and Performance*, pages 216–226, September 2000.
- [HZS01] X. S. Hu, T. Zhou, and E. H.-M. Sha. Estimating probabilistic timing performance for real-time embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(6):833–844, December 2001.
- [KM98] A. Kalavade and P. Moghé. A tool for performance estimation of networked embedded end-systems. In *Proceedings of the 35th Design Automation Conference*, pages 257–262, 1998.
- [Kop97] H. Kopetz. *Real-Time Systems*. Kluwer Academic, 1997.
- [KRT00] J. Kleinberg, Y. Rabani, and E. Tardos. Allocating bandwidth for bursty connections. *SIAM Journal on Computing*, 30(1):191–217, 2000.

- [KS96] J. Kim and K. G. Shin. Execution time analysis of communicating tasks in distributed systems. *IEEE Transactions on Computers*, 45(5):572–579, May 1996.
- [Leh96] J. P. Lehoczky. Real-time queueing theory. In *Proceedings of the 18th Real-Time Systems Symposium*, pages 186–195, December 1996.
- [LL73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):47–61, January 1973.
- [LPW99] C. Lee, M. Potkonjak, and W. Wolf. Synthesis of hard real-time application specific systems. *Design Automation of Embedded Systems*, 4:215–242, 1999.
- [MEP01] S. Manolache, P. Eles, and Z. Peng. Memory and time-efficient schedulability analysis of task sets with stochastic execution time. In *Proceedings of the 13th Euromicro Conference on Real Time Systems*, pages 19–26, June 2001.
- [MEP02] S. Manolache, P. Eles, and Z. Peng. Schedulability analysis of multiprocessor real-time applications with stochastic task execution times. In *Proceedings of the 20th International Conference on Computer Aided Design*, pages 699–706, November 2002.
- [MP92] M. Mouly and M.-B. Pautet. *The GSM System for Mobile Communication*. Palaiseau, 1992.
- [PDHT96] B. Powell Douglass, D. Harel, and M. Trakhtenbrot. *Statecharts in Use: Structured Analysis and Object-Orientation*, pages 368–394. Springer, 1996.
- [PR99] M. Potkonjak and J. M. Rabaey. Algorithm selection: A quantitative optimization-intensive approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(5):524–532, May 1999.
- [SR93] J. Stankovic and K. Ramamritham, editors. *Advances in Real-Time Systems*. IEEE Computer Society Press, 1993.
- [SWC95] A. Sarkar, R. Waxman, and J. P. Cohoon. *Specification-Modeling Methodologies for Reactive-System Design*, pages 1–34. Kluwer Academic Publishers, 1995.

- [TDS⁺95] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W. S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pages 164–173, May 1995.
- [van96] A. J. van Gemund. *Performance Modelling of Parallel Systems*. PhD thesis, Delft University of Technology, 1996.
- [van03] A. J. C. van Gemund. Symbolic performance modeling of parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 2003. to be published.
- [Wol94] W. Wolf. Hardware-software co-design of embedded systems. *Proceedings of the IEEE*, 82(7):967–989, 1994.
- [ZHS99] T. Zhou, X. (S.) Hu, and E. H.-M. Sha. A probabilistic performance metric for real-time system design. In *Proceedings of the 7th International Workshop on Hardware-Software Co-Design*, pages 90–94, 1999.

Chapter 7

Applications of wait/lock-free protocols to real-time systems

By **Håkan Sundell**[†], **Philippas Tsigas**[‡] and **Yi Zhang**^{*}

[†]Parallel Scalable Solutions AB
Berghemsgatan 6
431 37 Mölndal
Email: phs@pss-ab.se

[‡]Department of Computer Science and Engineering
Chalmers University of Technology
Email: tsigas@cs.chalmers.se

^{*}School of Computer Science
University of Birmingham
Email: y.zhang@cs.bham.ac.uk

Design of real-time systems is getting increasingly complex as systems are involving more and more parallelism. The basic reason behind this increasing difficulty is that the available solutions for the sub problems are not compositional, i.e. new problems are created.

A real-time system is consisting of a number of tasks that need to execute and also communicate and share common resources, as for example memory. This communication is though not straight forward as access to shared resources has to be controlled in order to avoid inconsistency. Moreover, due to limited processing power not all tasks can run at the same time, which states the need for efficient scheduling. In order for a real-time system to be reliable, it has to be analysed for schedulability and fault-tolerance. However, solutions using mutual exclusion for solving the shared resource problem, does not fit very well with schedul-

ing solutions, and moreover scale poorly with increasing parallelism. In more detail, the combination of mutual exclusion and scheduling gives the following problems:

- Complex and pessimistic schedulability analysis. The complexity increases rapidly with increasing level of parallelism, i.e. multiprocessor system.
- Poor system utilization. A large part of the available computing power consists of overhead or idle periods.
- Reduced fault-tolerance. The whole system can stop or miss deadlines due to local faults in a single task.

Our approach is to avoid mutual exclusion for handling shared resources and thus achieving a compositional solution together with uni- and multiprocessor scheduling. By using non-blocking synchronization for handling shared resources, real-time systems can benefit from simpler and more optimistic schedulability analysis, increased utilization and parallelism, and increased fault-tolerance as problems like priority inversion or deadlocks are not possible.

Our main contributions to non-blocking synchronization and real-time systems are as follows:

- New construction of a shared register for networks.
- New constructions for achieving a consistent snapshot view of global states.
- New construction of a shared queue for reliable inter-task communication for real-time threads in Java.
- New construction of a shared data buffer for efficient inter-task communication.
- New construction of a shared priority queue for efficient inter-task communication.
- Mechanisms and analysis methods for applying lock-free constructions to hard real-time systems.
- Design and construction of a shared software library, containing versatile components for efficient inter-task communication.

Our research results have gained considerable attention from academia as well as industry. For our priority queue implementation we have received a best paper award at the IEEE IPDPS 2003 symposium. More than 30 researchers from all over the world have registered

to and studied our shared software library of non-blocking components. We have received a lot of external requests and queries from industry internationally and have, for example, directly contributed with implementations to a company focusing on business services in USA. This company deals with a large number of time-sensitive transactions that are dependent on high reliability and high through-put in order to be successful. Our research results have been further developed into commercially oriented software components, and have recently been made globally available to the industry through the company Parallel Scalable Solutions AB.

7.1 Introduction

In a real-time system we usually have several tasks, executed in an order controlled by a scheduler, i.e. we have a multi-tasking system. As these tasks are related to each other in some way with shared resources, they have to synchronize in order to avoid the otherwise very likely risk of inconsistency caused by overlapping and concurrent accesses.

The general way of synchronizing the access of resources is to use *mutual exclusion*. Using mutual exclusion the tasks can make sure that only one task can have access to a shared resource or data structure at one time. Mutual exclusion is often provided by the operating system as semaphores or mutex locks. However, mutual exclusion or also called blocking synchronization has some drawbacks:

- They cause *blocking*. This means that other tasks that are eligible to run have to wait for some other task to finish the access of the shared resource. Blocking also makes the computation of worst-case response times more complicated, and the currently used computation methods are quite pessimistic. Blocking also can cause unwanted delays of other tasks as the effect propagates through the task schedule, also called the convoy effect.
- If used improperly they can cause *deadlocks*, i.e. no task involved in the deadlock can proceed, maybe because they are waiting for a lock to be released that is owned by a task that has died or is otherwise incapable of proceeding.
- They can cause *priority inversion*. The exclusion of other tasks while one low priority task is holding the lock can cause a high priority task to actually have to wait for middle priority tasks to finish.

These problems have been recognised since long and are thoroughly researched. Several solutions exist, the most common solutions are called

the *priority ceiling protocol* (PCP) and *immediate priority ceiling protocol* (IPCP) [Bak91, LSS87, Raj91, SRL90]. They solve the priority inversion problem for uniprocessor systems and limits the amount of blocking a task can be exposed to. However, the situation is much worse in a multiprocessor real-time system, where a task may be blocked by another task running on a different processor [Raj90].

7.2 Non-Blocking Synchronization

Researchers have also looked for other solutions to the problems of synchronising the access of shared resources. The alternative to using mutual exclusion is called *non-blocking*. The shared resources can still be accessed in a predictive manner, called *atomic*. Atomic means that the operation can be viewed by the processes as it occurred at a unique instant in time, i.e. the effect of two operations can not be viewed as happening at the same time. As non-blocking algorithms do not involve mutual exclusion, they can be executed concurrently, and thus proving atomicity is quite more complex than for mutual exclusion. The correctness condition used for concurrent operations is called *linearizability* [HW90].

There are two basic levels of non-blocking algorithms, called *lock-free* and *wait-free*. Common with most non-blocking algorithms are that they take advantage of atomic primitives.

7.2.1 Lock-Free

Lock-Free implementations of shared data structures guarantee that at any point in time in any possible execution some operation will complete in a finite number of steps. In cases with overlapping accesses, some of them might have to repeat the operation in order to correctly complete it. This implies that there might be cases in which the timing may cause some process(es) to have to retry a potentially unbounded number of times, leading to an unacceptable worst-case behaviour for hard real-time systems. However, they usually perform well in practice, and the number of possible retries can be reliably predicted using scheduling information.

7.2.2 Wait-Free

In wait-free implementations each task is guaranteed to *correctly* complete any operation in a *bounded* number of its own steps, regardless of overlaps and the execution speed of other processes; i.e. while the lock-free approach might allow (under very bad timing) individual processes

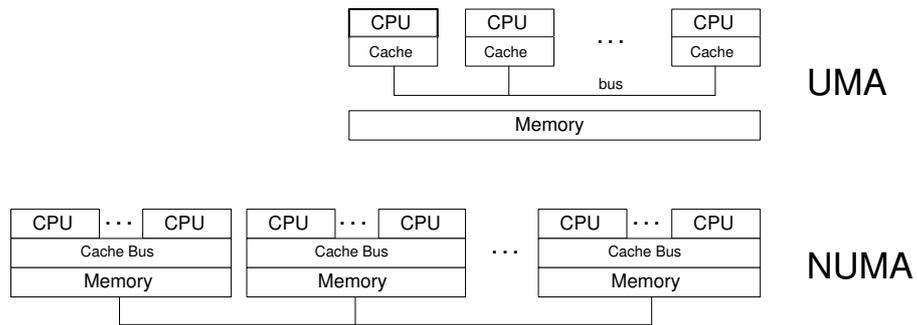


Figure 7.1: Uniform versus non-uniform memory access (UMA vs NUMA) architecture for multiprocessor systems

to starve, wait-freedom strengthens the lock-free condition to ensure individual progress for every task in the system.

7.3 Shared Memory

Most modern systems have support for shared memory, where tasks can concurrently access data in a consistent way. For multiprocessor systems, the shared memory system can be either *uniformly distributed* as in the uniform memory access (UMA) architecture [ACFS94] or *non-uniformly distributed* as in the non-uniform memory access (NUMA) architecture [LaR91]. In a NUMA system, the response time of a memory access depends on the actual distance between the processor and the real memory, although it is the same for each processor on the same node. For the UMA system the response time for memory accesses are the same all over the system, see Figure 7.1.

In order to enable synchronization between tasks, the system has to provide some kind of atomic primitives for this purpose. There are different kinds of atomic primitives available on different platforms, some less powerful than the others. All platforms do not directly support all known atomic primitives, although most atomic primitives can usually be implemented using other means. Basic read and write operations of normal-sized integers are usually designed to be atomic with respect to each other and other operations. The most common atomic primitives for synchronisation are Test-And-Set (TAS), Fetch-And-Add (FAA) and Compare-And-Swap (CAS), see Figure 7.2.

In theory, the CAS operation is sufficient to implement any lock-free or wait-free algorithm. However, because of the high complexity involved in designing non-blocking algorithms, many published results rely on even more powerful atomic primitives that are available on very

```

function TAS(value:pointer to word):boolean
  atomic do
    if *value=0 then
      *value:=1;
      return true;
    else return false;

procedure FAA(address:pointer to word, number:integer)
  atomic do
    *address := *address + number;

function CAS(address:pointer to word, oldvalue:word, newvalue:word):boolean
  atomic do
    if *address = oldvalue then
      *address := newvalue;
      return true;
    else return false;

```

Figure 7.2: The Test-And-Set (TAS), Fetch-And-Add (FAA) and Compare-And-Swap (CAS) atomic primitives.

few or even none known computer architectures. In our research we have therefore focused on designing algorithms for practical use within real-time systems, i.e. using only commonly available atomic primitives.

7.3.1 Shared Registers over Networks [ST00]

Some systems might have a limited support for shared memory or even none. The shared memory can then be implemented using message passing, for example the *controller area network* (CAN) bus [Arn87]. In [ST00] we present a space efficient wait-free algorithm for implementing a shared buffer for real-time multi-processor systems.

We first start with a simple, elegant and easy to implement unbounded protocol that appeared in [VA86]. The algorithm uses a matrix of 1-reader 1-writer registers, see figure 7.3. We denote the reader task and the writer task running on processor i with Rd_i and Wr_i respectively. The matrix is formed in a way such that each register Rg_{ij} can be read by processor j and written by processor i . The algorithm originally uses unbounded time-stamps and consequently the data read from or written into each of the registers contains a data pair of a value and a tag.

The value for the tags during the execution increase rapidly and thus many bits have to be allocated for the tag value on the register to ensure there is no overflow. As the register contains a limited number of bits, this means that we have fairly few bits to allocate for the actual value. In real-time systems random access memory is also a limited resource and registers usually contain few bits. Further the system is usually required

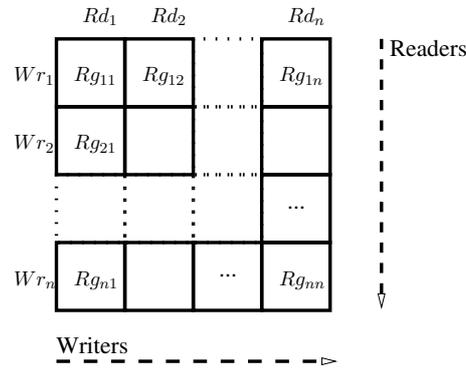


Figure 7.3: Architecture of the Algorithm

be able to run continuously for a very long period, which means that we have to allocate a lot of bits for the tag field, to ensure there is no overflow. Therefore the time-stamps have to be recycled and we achieved the following resulting lemma.

Lemma 1 *In any possible execution the time-stamps that two consecutive tasks can observe are going to be $MaxTag = \sum_{i=1}^n \left\lceil \frac{T_{max}}{T_{Wr_i}} \right\rceil + \sum_{i=1}^n \left\lceil \frac{R_{max}}{T_{Wr_i}} \right\rceil$ far apart.*

We can also conclude that in order to correctly compare possibly wrap-around time-stamps, we can use a tag field with double the size of the maximum possible value of the tag.

7.3.2 Globally Consistent Shared States - Snapshots [AEH⁺99] [STZ00]

The snapshot problem involves taking an “instantaneous” picture of a set of variables, all in one atomic operation. The snapshot is taken by one task, the *scanner*, while each of the snapshot variables may concurrently and independently be *updated* by other processes (called *updaters*). A snapshot object is also called a *composite register*, consisting of a number of *components* (indexed 1 through c), which constitute the entities which can be updated and scanned.

Wait-Free Snapshots using Primitives

In [AEH⁺99] we evaluate lock and wait-free snapshot algorithms analytically and we present the results of our experimental study and evaluation in a simulated real-time system appropriate for automotive/avionics monitoring/control. The system consists of CAN-bus-connected nodes,

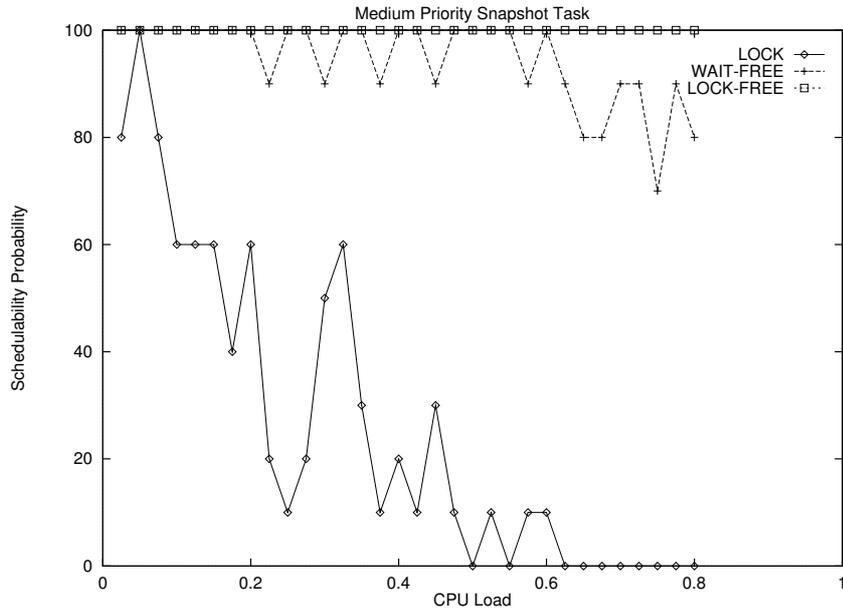


Figure 7.4: Schedulability experiments for a single-node system with a medium priority snapshot task

each of which is connected to a set of devices, whose measurements activate the updates in the system via I/O controllers. The scan operations are executed by a specific controller task in the system. We perform a comparative study and evaluation among lock-based, lock-free and our wait-free snapshot. The wait-free snapshot algorithm is based on the TAS atomic primitive. Our results suggest that our algorithms are promising, more efficient and safe alternatives to lock-based algorithms for the above and similar real-time applications, and that they do not really imply additional cost to the system, compared to the lock-based approach. Compared to the lock-free approach, our wait-free solution not only provides stronger guarantees, but it is also more predictable; i.e. retains its performance under more frequent updates, while the lock-free scan may take longer time, or even not terminate at all. In multi-node systems it seems that the lock-free method is not useful at all, while our algorithms perform good in both single and multi-node systems.

Wait-Free Snapshots using Timing Information

In [STZ00] we show how to exploit information that is part of the special nature of the real-time systems in order to design a simple snapshot algorithm with one scanner that is efficient in time and space as needed. The algorithm that we propose here outperforms significantly — due to

its simplicity — the respective one not using this information [AEH⁺99]. Experiments on a Sun Enterprise 10000 Starfire [Cha98] has shown that the new construction gives 400 % better response time for the update operations for all scenarios and with 20 % better response time for all practical settings. Please notice that we have one scan task at a time and multiple concurrent update tasks per component in multiple components.

We started with a simple unbounded snapshot protocol that first appeared in Kirousis et al [KST94]. The protocol uses buffers of infinite length. In our bounded snapshot protocol, the buffer length can be

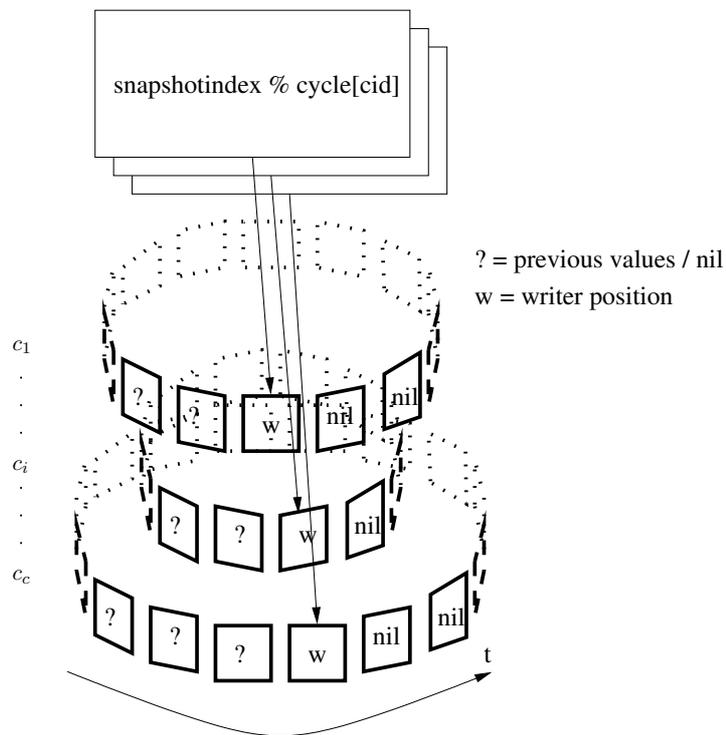


Figure 7.5: Bounded Snapshot Protocol

of different length for each individual component, and that the buffer length is dependent on the timing characteristics of the updaters that write to this component, and also dependent on the timing characteristics of the snapshot task which advances the buffer index. See figure 7.5 for an explanation how the algorithm interacts with the cyclic buffers. The length of the buffer for each component can be computed as follows:

$$l'_k = \left\lceil \frac{2 * \max_{i \in wr(k)} T_{W_i}}{T_S} \right\rceil + 2$$

7.4 Shared Concurrent Data Structures

7.4.1 Shared Buffers [TZ99]

In this paper we are interested in the general read/write buffer problem where several reader-tasks, (*the readers*), access a buffer maintained by several writer-tasks, (*the writers*). There is no limit on the length of the buffer that can be increased by the writers on-line depending on the length of the data that they want to write in one atomic operation. The shared data object can be used to increase the word length of the system.

In [TZ99], we present an efficient non-blocking solution to the general readers/writers inter-process communication problem; our solution allows any arbitrary number of readers and writers to perform their respective operations. With a simple and efficient memory management scheme in it, our protocol needs $n + m + 1$ memory slots (buffers) for n readers and m writers and is optimal with respect to space requirements. Sorensen and Hemacher in [SH75] have proven that $n + m + 1$ memory slots (buffers) are necessary. Together with the protocol, its schedulability analysis and a set schedulability tests for a set of random task sets are presented. Both the schedulability analysis and the schedulability experiments show that the algorithm presented in this paper exhibits less overhead than the lock based protocol. Our protocol extends previous results by allowing any arbitrary number of tasks to perform read or write operations concurrently without trading efficiency.

We assume that writer-tasks have the highest priority on the host processor and no two writer-tasks execute on one host processor. Two writer-tasks may overlap but no write processes can be preempted by another process. Reader processes may have different priorities and may be scheduled with the writer process on the same processor. This is because writer-tasks usually interact with the environment (e.g. sampler for temperature or pressure), and they are of high priority. To the best of our knowledge the above assumption holds in most real-time systems.

The response time of a reader task i can be calculated with the following formula:

$$R_i = C_i + \sum_{j \in HP(i)} \left\lceil \frac{D_j}{R_i} \right\rceil C_j + \min \left(\sum_{j \in HP(i)} \left\lceil \frac{D_j}{R_i} \right\rceil, \left\lceil \frac{\sum_{j \in HP(i)} \left\lceil \frac{D_j}{R_i} \right\rceil C_j}{2 * P_w} \right\rceil \right) T_r$$

where $HP(i)$ represents all tasks who run on the same processor as task i and whose priorities is higher than the priority of task i .

7.4.2 Queues [TZ02]

In this work, we present algorithmic implementations of the wait-free queue classes of the Real-time Specification for Java. These implementations are designed to have the unidirectional nature of these queues in mind and they are more efficient, with respect to space, compared to previous wait-free implementations, without losing in time complexity. The wait-free queue classes proposed in the Real-time Specification for Java are of general interest to any real-time synchronization system where hard real-time tasks have to synchronize with soft or even non real-time tasks.

In this paper implementations of RTSJ queue classes with $O(M+N)$ space complexity and $O(N)$ time complexity are presented. Experiments we've performed suggest that our algorithms are typically 9% – 36% faster than the previous best one. The wait-free queue classes that are provided by RTSJ have been designed to enable communication between the real-time `NoHeapRealtimeThreads` and the regular Java threads; they have a unidirectional nature with one side of the queue (read or write) for the real-time threads and the other one (write or read, respectively) for the non-real-time ones. The implementations presented in this paper are designed having the unidirectional nature of these queues in mind in order to gain efficiency; to the best of our knowledge our implementations are the first unidirectional wait-free queue implementations in the literature.

To evaluate the performance of our algorithm, we compare it with the best previously known solution that was proposed in [ARJ97]. We performed our experiments on a real-time environment simulator based on the Real-Time Threads (RTT) Package [LHC93]. The RTT provides priority-based preemptive scheduling for real-time threads that is similar to the real-time java virtual machine specification. To the best of our knowledge there is no implementation of a real-time java virtual machine available yet.

We performed experiments with 1, 2, 4, 8 and 16 concurrent enqueue tasks. The parameters of the task sets were selected so that the tasks are schedulable and are based on the following formulas:

$$T_i = (n - i) * T, \quad i = 0, \dots, (n - 1)$$

$$C_i = \begin{cases} (n - i) * C, & i = 1, \dots, (n - 1) \\ (18 - (n - 1)), & i = 0 \end{cases}$$

T and C are the period and computation time of the highest priority task respectively and $\frac{C}{T} = 20$. T_i and C_i are the period and computation time of the task i respectively. N is the number of concurrent enqueue tasks. The task sets are scheduled with the rate-monotonic scheduling algorithm. With the task parameters described above, the processor

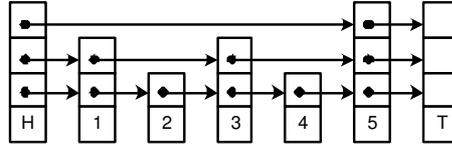


Figure 7.6: The Skiplist data structure with 5 nodes inserted.

utilization is $U = \sum \frac{C_i}{T_i} = 90\%$ for all task sets and all task sets are schedulable under rate-monotonic scheduling [KRP⁺93].

The results of our experiments show that our algorithm does not only decrease dramatically the space requirements but is also from 9% to 36% faster than best previously known.

7.4.3 Priority Queues [ST03]

Priority queues are fundamental data structures. From the operating system level to the user application level, they are frequently used as basic components. For example, the ready-queue that is used in the scheduling of tasks in many real-time systems, can usually be implemented using a concurrent priority queue. Consequently, the design of efficient implementations of priority queues is a research area that has been extensively researched. A priority queue supports two operations, the *Insert* and the *DeleteMin* operation. The abstract definition of a priority queue is a set of key-value pairs, where the key represents a priority. The *Insert* operation inserts a new key-value pair into the set, and the *DeleteMin* operation removes and returns the value of the key-value pair with the lowest key (i.e. highest priority) that was in the set.

In [ST03] we present a lock-free algorithm of a concurrent priority queue that is designed for efficient use in both pre-emptive as well as in fully concurrent environments. Inspired by Lotan and Shavit [LS00], the algorithm is based on the randomized Skiplist [Pug90] data structure, but in contrast to [LS00] it is lock-free. It is also implemented using common synchronization primitives that are available in modern systems. Experiments were performed on three different platforms, consisting of a multiprocessor system using different operating systems and equipped with either 2, 6 or 29 processors. Our results show that our algorithm outperforms the other lock-based implementations in practical scenarios for 3 threads and more, in both highly pre-emptive as well as in fully concurrent environments. We also present an extended version of our algorithm that also addresses certain real-time aspects of the priority queue as introduced by Lotan and Shavit [LS00].

When we have concurrent *Insert* and *DeleteMin* operations we might want to have certain real-time properties of the semantics of the

DeleteMin operation, as expressed in [LS00]. The *DeleteMin* operation should only return items that have been inserted by an *Insert* operation that finished before the *DeleteMin* operation started. To ensure this we are adding timestamps to each node.

As we are only using the timestamps for relative comparisons, we do not need real absolute time, only that the timestamps are monotonically increasing. Therefore we can implement the time functionality with a shared counter, the synchronization of the counter is handled using CAS. However, the shared counter usually has a limited size (i.e. 32 bits) and will eventually overflow. Therefore the values of the timestamps have to be recycled. We will do this by exploiting information that are available in real-time systems, with a similar approach as in [ST00].

$$MaxTag < \sum_{i=0}^n \left(\left[\frac{N + 2n + \sum_{k=0}^n \left\lceil \frac{\max_j R_j}{T_k} \right\rceil}{T_i \sum_{l=0}^n \frac{1}{T_l}} \right] + 1 \right) \quad (7.1)$$

7.5 Adapting Lock-Free to Hard Real-Time Systems [TZ03]

In this paper, we address the problem of applying lock-free synchronization in real-time shared memory multiprocessor systems. First, we look into the causes of unbounded worst case execution behavior when accessing lock-free shared objects in such systems. Then, in order to make lock-free synchronization applicable to real-time shared memory multiprocessors, we propose inter-process an protocol that bounds the worst case execution time of an operation accessing a lock-free shared object in such systems. The proposed protocol works for lock-free synchronization the same way that MPCP or DPCP work for mutual exclusion. With the help of such protocols, the worst case execution time of accessing shared objects is bounded and tasks which communicate through shared objects can be scheduled as independent tasks. Our results complement the results in [AR95, JAJ97] that were proposed for uniprocessor systems. In [AR95, JAJ97], the interference between tasks, running on the *same processor*, accessing lock-free shared data object was analyzed. Based on this analysis, a way was shown to bound the number of retries when accessing lock-free shared objects under certain conditions with RM and EDF scheduling for uniprocessors.

Lemma 2 *In real-time shared memory multiprocessors, where non-preemptive scheduling is applied on each processor and all tasks use the retry-level based protocol to access lock-free shared objects, the number*

of overlap interferences (in the worst case) on the i th task running on processor j with retry-level $R(T_i, a)$ while accessing shared object a , is denoted as B_i , and is the smallest natural number that satisfies the following formula:

$$B_i = \sum_{k \in OP(j)} \sum_{l \in TS'(k)} \lceil \frac{(B_i + 1) * s}{P(T_l)} \rceil$$

where $TS'(k) = \{t | t \in TS(k, a) \vee R_t^a < R_i^a\}$

Theorem 1 *In real-time shared memory multiprocessors, where non-preemptive scheduling is applied on each processor and all tasks use the retry-level based protocol to access lock-free shared objects, the worst case execution time of the i th task is*

$$W_{cet} = W_o(i) + (B_i + 1) * s$$

where B_i is given by Lemma 2.

Theorem 2 *In real-time shared memory multiprocessors, where preemptive scheduling is applied on each processor, and all tasks use the combination of the ICPP protocol on uniprocessor level, and the retry-level based protocol to access a lock-free object a , then the number of interferences on an operation while accessing a lock-free shared object a for task T_i , is denoted as B_i , and is the smallest natural number that satisfies the following formula:*

$$B_i = \sum_{k \in OP(j)} \sum_{l \in TS'(k)} \lceil \frac{(B_i + 1) * s}{P(T_l)} \rceil$$

where $TS'(k) = \{t | t \in TS(k, a) \vee R_t^a < R_i^a\}$

7.6 NOBLE – A Library of Efficient Inter-Process Communication Protocols [ST02]

When designing NOBLE we identified a number of characteristics that had to be covered by our design in order to make NOBLE easy to use for a wide range of practitioners. We designed NOBLE to have the following features:

- Usability-Scope - NOBLE provides a collection of fundamental shared data objects that are widely used in parallel and real-time applications.

- Easy to use - NOBLE provides a simple interface that allows the user to use the non-blocking implementations of the shared data objects in the same way the user would have used lock-based ones.
- Easy to Adapt - No need for changes at the application level are required by NOBLE and different implementations of the same shared data objects are supported via a uniform interface.
- Efficient - Users will have to experience major improvements in order to decide to replace the existing trusted synchronization code with new methods. NOBLE has been designed to be as efficient as possible.
- Portable - NOBLE has been designed to support general shared memory multi-processor architectures and offers the same user interface on different platforms. The library has been designed in layers, so that only changes in a limited part of the code are needed in order to port the library to a different platform.
- Adaptable for different programming languages.

From all the experimental results that we collected we could definitely conclude that NOBLE, largely because of its non-blocking characteristics and partly because of its efficient implementation, outperforms the respective lock-based implementations significantly. For the singly linked list, NOBLE was up to 64 times faster than the lock-based implementation. In general, the performance benefits from using NOBLE and lock-free synchronization methods increase with the number of processors. For the experiments with low contention, NOBLE still performs better than the respective lock-based implementations, for a high number of processors NOBLE performs up to 3 times faster than the respective lock-based implementations.

7.7 Conclusions

We have studied non-blocking synchronization in the context of soft and hard real-time systems. Apparently, it is a very suitable alternative to mutual exclusion as it avoids the severe and complex problems that are connected to real-time systems and handling shared resources. Avoiding mutual exclusion also avoids problems as deadlocks, convoy effects and priority inversion, and consequently leads to a simpler and more optimistic schedulability analysis.

The simplest form of non-blocking algorithms, called lock-free, is suitable for soft real-time systems thanks to its efficiency and low overhead. Combined with scheduling information and/or special additional

scheduling mechanisms it can also be applied to hard real-time systems. The strongest form of non-blocking algorithms, called wait-free, is directly suitable for hard real-time systems thanks to its strong termination guarantees. It can greatly help efforts with estimating worst-case execution time guarantees for code regions accessing shared resources as the number of execution steps is bounded.

We have designed several new non-blocking algorithms for inter-task communication purposes. These have also been implemented and tested in real executions on real multiprocessor systems. Our algorithms are based on atomic primitives that are commonly available and are thus practically applicable on a majority of available system platforms. We have gained substantial interest from academia as well as industry with our results. Our results have been further developed into independent software components and are now available commercially through a new company (Parallel Scalable Solutions AB). One of the main visions with the ARTES project was to bring new knowledge to the industry, and in that respect we think we have been highly successful.

We think that non-blocking techniques is highly applicable within real-time systems, and will continue to play a dominant part in the future evolution towards more parallelism and increased reliability.

Bibliography

- [ACFS94] B. Alpern, L. Carter, E. Feig, and T. Selker. The uniform memory hierarchy model of computation. *Algorithmica*, (12):72–109, 1994.
- [AEH⁺99] B. Allvin, A. Ermedahl, H. Hansson, M. Papatriantafidou, H. Sundell, and Ph. Tsigas. Evaluating the performance of wait-free snapshots in real-time systems. In *SNART'99 Real Time Systems Conference*, pages 196–207, August 1999.
- [AR95] J. Anderson and S. Ramamurthy. Using lock-free objects in hard real-time applications. In *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing*, pages 272–272. ACM, August 1995.
- [ARJ97] J. Anderson, S. Ramamurthy, and R. Jain. Implementing wait-free objects on priority-based systems. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing*, pages 229–238. ACM, August 1997.
- [Arn87] D. Arnett. A high performance solution for in-vehicle networking - controller area network (CAN). Technical Report 870823, SAE Technical Paper Series, April 1987.

- [Bak91] T. Baker. Stack-based scheduling on real-time processes. *Real-Time Systems*, 3(1):97–69, March 1991.
- [Cha98] A. Charlesworth. Starfire: Extending the SMP envelope. *IEEE Micro*, January 1998.
- [HW90] Maurice Herlihy and J. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, 1990.
- [JAJ97] S. Ramamurthy J. Anderson and K. Jeffay. Real-time computing with lock-free shared objects. *Transactions on Computer Systems*, 15(1):134–165, February 1997.
- [KRP⁺93] M.H. Klein, T. Ralya, B. Pollak, R. Obenza, M. Gonza, and L. Harbour. *A Practitioners Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real Time Systems*. Kluwer Academic Publishers, 1993.
- [KST94] Lefteris M. Kirousis, Paul Spirakis, and Philippas Tsigas. Reading many variables in one atomic operation: Solutions with linear or sublinear complexity. *IEEE Transactions on Parallel and Distributed Systems*, 5(7):688–696, July 1994.
- [LaR91] R. P. LaRowe Jr. *Page Placement For Non-Uniform Memory Access Time (NUMA) Shared Memory Multiprocessors*. PhD thesis, Duke University, Durham, North Carolina, 1991.
- [LHC93] S. L. A. Lo, N. C. Hutchinson, and S. T. Chanson. Architectural considerations in the design of real-time kernels. In *Proceedings of the 14th Real-Time Systems Symposium*, pages 138–147. IEEE, December 1993.
- [LS00] I. Lotan and Nir Shavit. Skiplist-based concurrent priority queues. In *Proceedings of the International Parallel and Distributed Processing Symposium 2000*, pages 263–268. IEEE press, 2000.
- [LSS87] J.P. Lehoczky, L. Sha, and J.K. Strosnider. Aperiodic responsiveness in hard real-time environments. In *IEEE Real-Time Systems Symposium*, pages 262–270, 1987.
- [Pug90] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

- [Raj90] R. Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 116–123, 1990.
- [Raj91] R. Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, 1991.
- [SH75] P. Sorensen and V. Hemacher. A real-time system design methodology. *INFOR*, 13(1):1–18, February 1975.
- [SRL90] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.
- [ST00] Håkan Sundell and Philippas Tsigas. Space efficient wait-free buffer sharing in multiprocessor real-time systems based on timing information. In *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications (RTCSA 2000)*, pages 433–440. IEEE press, 2000.
- [ST02] Håkan Sundell and Philippas Tsigas. NOBLE: A non-blocking inter-process communication library. In *Proceedings of the 6th Workshop on Languages, Compilers and Run-time Systems for Scalable Computers*, 2002.
- [ST03] Håkan Sundell and Philippas Tsigas. Fast and lock-free concurrent priority queues for multi-thread systems. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium*, page 11. IEEE press, April 2003.
- [STZ00] Håkan Sundell, Philippas Tsigas, and Yi Zhang. Simple and fast wait-free snapshots for real-time systems. In *Proceedings of the 4th International Conference On Principles Of Distributed Systems (OPODIS 2000)*, Studia Informatica Universalis, pages 91–106, 2000.
- [TZ99] Philippas Tsigas and Yi Zhang. Non-blocking data sharing in multiprocessor real-time systems. In *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*, pages 247–254. IEEE press, 1999.
- [TZ02] Philippas Tsigas and Yi Zhang. Efficient wait-free queue algorithms for real-time synchronization. Technical Report

- 2002-05, Computing Science, Chalmers University of Technology, 2002.
- [TZ03] Philippas Tsigas and Yi Zhang. Lock-free object sharing for shared memory real-time multiprocessors. Technical Report 2003-03, Computing Science, Chalmers University of Technology, 2003.
- [VA86] P.M.B. Vitanyi and B. Awerbuch. Atomic shared register access by asynchronous hardware. In *27th IEEE Annual Symposium on Foundations of Computer Science*, pages 233–243, October 1986.

Chapter 8

Reconfigurable Embedded Real-Time Systems: A Story of COMET

By Aleksandra Tešanović[†], Dag Nyström[‡], Jörgen Hansson[†]
and Christer Norström[‡]

[‡]Department of Computer Engineering
Mälardalen University

Email: {dag.nystrom,christer.norstrom}@mdh.se

[†]Department of Computer Science
Linköping University

Email: {alete,jorha}@ida.liu.se

Increasing complexity of real-time systems and demands for enabling their configurability and reusability are strong motivations for applying new software engineering principles, such as aspect-oriented and component-based development. In this chapter we present the concept of aspectual component-based real-time system development. The concept is based on a design method that assumes decomposition of real-time systems into components and aspects, and provides a real-time component model that supports the notion of time and temporal constraints, space and resource management constraints, and composability. Initial results show that the successful application of the proposed concept has a positive impact on real-time system development in enabling efficient configuration of real-time systems, improved reusability and flexibility of real-time software, and modularization of crosscutting concerns. We provide arguments for this by presenting an application of the proposed concept on the design and development of a configurable embedded real-time database, called COMET.

8.1 Introduction

A large majority of computational activities in the modern society are performed within embedded and real-time systems. Successful deployment of these systems depends on low development costs, a short time to market, and high degree of tailorability [1]. Thus, the introduction of the component-based software development (CBSD) [2] into real-time and embedded systems development offers significant benefits, namely: (i) configuration of embedded and real-time software for a specific application using components from the component library, thus reducing the system complexity as components can be chosen to provide the functionality needed by the system; (ii) rapid development and deployment of real-time software as many software components, if properly designed and verified, can be reused in different embedded and real-time applications; and (iii) evolutionary design as components can be replaced or added to the system, which is appropriate for complex embedded real-time systems that require continuous hardware and software upgrades.

However, there are aspects of real-time and embedded systems that cannot be encapsulated in a component with well-defined interfaces as they crosscut the structure of the overall system, e.g., synchronization, memory optimization, power consumption, and temporal attributes. Aspect-oriented software development (AOSD) has emerged as a new principle for software development that provides an efficient way of modularizing crosscutting concerns in software systems [3]. AOSD allows encapsulating crosscutting concerns of a system in “modules”, called aspects.

Applying AOSD in real-time and embedded system development would reduce the complexity of the system design and development, and provide means for a structured and efficient way of handling crosscutting concerns in a real-time software system. Hence, the integration of the two disciplines, CBSD and AOSD, into real-time systems development would enable: (i) efficient system configuration using components and aspects from the library based on the system requirements, (ii) easy tailoring of components and/or a system for a specific application, i.e., reuse context, by changing the behavior (code) of a component by applying aspects. This results in enhanced flexibility of the real-time and embedded software through the notion of system configurability and component tailorability.

To successfully apply software engineering techniques such as AOSD and CBSD when developing real-time systems, a number of research challenges need to be addressed. In this chapter we focus on the following two: (i) issues that should be addressed and the criteria that should be enforced by a design method to allow integration of the two software engineering techniques into real-time systems; and (ii) characteristics of

a component model that can capture and adopt principles of CBSD and AOSD in a real-time and embedded environment.

We address the above issues by presenting our main contributions that can be summarized as follows: (i) **the concept of aspectual component-based real-time system development (ACCORD)** that is founded on a design method that decomposes real-time systems into components and aspects; and (ii) **a reconfigurable real-time component model (RTCOM)** that describes what a real-time component, supporting different types of aspects and enforcing information hiding, looks like. We developed ACCORD with hard real-time systems in mind; the approach is also general enough to be used for building firm and soft real-time systems. In this chapter we focus only on hard real-time systems, and present a case study that shows how ACCORD is applied to the design and development of a configurable embedded (hard) real-time database, called COMET. Requirements for the COMET database have been extracted from an automotive industry case study [4]. The material presented in the chapter is based on [5, 6, 7].

The chapter is organized as follows. In section 8.2 a background to component-based and aspects-oriented software development is presented. In section 8.3 we present an outline of ACCORD and its design method. We present RTCOM in section 8.4. In section 8.5 we show an application of ACCORD to the development of COMET. Related work is presented in section 8.6. The chapter finishes with a summary containing the main conclusions in section 8.7.

8.2 Components and Aspects

In this section, background to component-based and aspects-oriented software development is presented and main differences between components in component-based and aspect-oriented software development are discussed.

Software components are the core of CBSD. Developing systems using existing components offers many advantages to developers and users, such as decreased development costs, increased quality of software, shortened time-to-market, and reduced maintenance costs [8, 9, 10, 11]. While frameworks and standards for components today primarily focus on CORBA, COM, or JavaBeans, the increasing need for component-based development has also been identified in the area of real-time and embedded systems [12, 13, 1]. However, different definitions and interpretations of a component exist. We consider a component to be a software artifact that models and implements a well-defined set of func-

tions, and has well-defined (but not necessarily standardized) component interfaces [14].

While CBSD uses black box as an abstraction metaphor for the components, AOSD uses the white box component metaphor to emphasize that all details of the component implementation should be revealed. Both black box and white box component abstractions have their advantages and disadvantages. For example, hiding all details of a component implementation in a black box manner has the advantage that a component user does not have to deal with the component internals. In contrast, having all details revealed in a white box manner allows a component user to freely optimize and tailor the component for a particular software system.

Typically, aspect-oriented implementation of a software system has the following constituents: (i) components, written in a component language, e.g., C, C++, and Java; (ii) aspects, written in a corresponding aspect language, e.g., AspectC [15], AspectC++ [16], and AspectJ [17] developed for Java;¹ and (iii) an aspect weaver, which is a special compiler that combines components and aspects in a process called aspect weaving.

The AOSD community uses the term component to denote a traditional software module, e.g., program, function, or method, completely accessible by the users. Hence, components in AOSD do not enforce information hiding and are fully open to changes and modifications of their internal structure via aspect weaving. An aspect is commonly considered to be a property of a system that affects its performance or semantics, and that crosscuts the functionality of the system [3].

In existing aspect languages, each aspect declaration consists of advices and pointcuts. A *pointcut* in an aspect language consists of one or more join points, and it is described by a pointcut expression. A *join point* refers to a point in the component code where aspects should be woven, e.g., a method, a type (struct or union). Figure 8.1(a) shows the definition of a named pointcut `getLockCall`, which refers to all calls to the function, i.e., join point, `getLock()` within the program with which the aspect is to be woven, and exposes a single integer argument to that call.²

An *advice* is a declaration used to specify the code that should run when the join points, specified by a pointcut expression, are reached. Different kinds of advices can be declared, such as: (i) *before advice* code is executed before the join point, (ii) *after advice* code is executed immediately after the join point, and (iii) *around advice* code is executed in place of the join point. Figure 8.1(b) shows an example of an after

¹These aspect languages share many similarities with AspectJ.

²The example presented is written in AspectC++.

```
pointcut getLockCall(int lockId)=  
    call("void getLock(int)")&&args(lockId);
```

(a) A pointcut definition

```
advice getLockCall(lockId):  
    void after (int lockId)  
    {  
        cout<<"Lock requested is"<<lockId<<endl;  
    }
```

(b) An example of the advice definition

Figure 8.1: An example of an aspect declaration

advice. With this advice each call to `getLock()` is followed by the execution of the advice code, i.e., printing of the lock id.

The main motivation and the main benefits of CBSD overlap and complement the ones of AOSD. Furthermore, making aspects and aspect weaving usable in CBSD would allow improved flexibility in tailoring of components and, thus, enhanced reuse of components in different systems. To allow aspects to invasively change the component code and still preserve information hiding to the largest extent possible requires using the grey box abstraction metaphor for the component. The grey box component preserves some of the main features of a black box component, such as well-defined interfaces as access points to the component, and it also allows aspect weaving to change the behavior and the internal state of the component.

8.3 ACCORD Design Method

We have argued that the growing need for enabling development of configurable real-time and embedded systems that can be tailored for a specific application, and managing the complexity of the requirements in the real-time system design, calls for an introduction of new concepts and new software engineering paradigms into real-time system development. In this section we present ACCORD as a proposal to address these

new needs. Through the notion of aspects and components, ACCORD enables efficient application of the divide-and-conquer approach to complex system development. To effectively apply ACCORD, we provide a design method with the following constituents.

- A decomposition process with two sequential phases: (i) decomposition of the real-time system into a set of components, and (ii) decomposition of the real-time system into a set of aspects.
- Components as software artifacts that implement a number of well-defined functions, and where they have well-defined interfaces.
- Aspects as properties of a system affecting its performance or semantics, and crosscutting the functionality of the system [3].
- A real-time component model (RTCOM) that describes a real-time component, which supports aspects but also enforces information hiding. RTCOM is specifically developed to: (i) enable an efficient decomposition process, (ii) support the notion of time and temporal constraints, and (iii) enable efficient analysis of components and the composed system.

The design of a real-time system using ACCORD method is performed in two phases. In the first phase, a real-time system is decomposed into a set of components. Decomposition is guided by the need to have functionally exchangeable units that are loosely coupled, but with strong cohesion. In the second phase, a real-time system is decomposed into a set of aspects. Aspects crosscut components and the overall system. This phase typically deals with non-functional requirements³ and crosscutting concerns of a real-time system, e.g., resource management and temporal attributes. After the design, components and aspects are implemented based on RTCOM.

Analogous to the classical object-oriented design method that initially identifies objects as building blocks of a system, ACCORD initially identifies components and aspects as building blocks of a real-time software system. Hence, ACCORD can be viewed as an extension to the classical object-oriented design method, which in turn implies that ACCORD is founded on a well-established design method.

8.3.1 Aspects in Real-Time Systems

We classify aspects in a real-time system as follows (see figure 8.2): (i) application aspects, (ii) run-time aspects, and (iii) composition aspects.

³Non-functional requirements are sometimes referred to as extra-functional requirements [18].

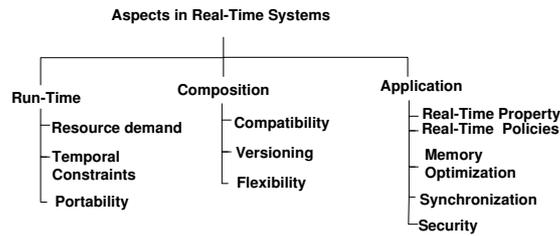


Figure 8.2: Classification of aspects in real-time systems

Application aspects can change the internal behavior of components as they crosscut the code of components in the system. The application in this context refers to the application toward which a real-time and embedded system should be configured, e.g., memory optimization aspect, security aspect, real-time property aspect, and real-time policy aspect. Application aspects enable reconfiguring components for a specific application, as they change code of the components. Hence, we informally define application aspects as programming (aspect) language-level constructs encapsulating crosscutting concerns that invasively change the code of the component. These aspects correspond to the traditional aspects in AOSD.

Run-time aspects give information needed by the run-time system to ensure that integrating a real-time system would not compromise timeliness or available memory consumption. These aspects enable a component to be mapped to a task (or a group of tasks) with specific temporal requirements. We informally define run-time aspects as design-level and/or aspect language-level constructs encapsulating crosscutting concerns that contain the information the component behavior with respect to the target run-time environment.

Composition aspects describe with which components a component can be combined, the version of the component, and possibilities of extending the component with additional aspects. Composition aspects can be viewed as language-independent design-level constructs encapsulating crosscutting concerns that describe the component behavior with respect to the composition needs of each component. This implies that composition aspects do not invasively change the code of the component.

Having separation of aspects in a number of categories eases reasoning about various application-related requirements, as well as the composition of a system and its integration into a run-time environment. For example, one could define what (run-time) aspects a real-time system configuration should fulfill so that appropriate components and application aspects could be chosen from the library. Aspect separation and classification offers a flexibility since additional aspect types can be

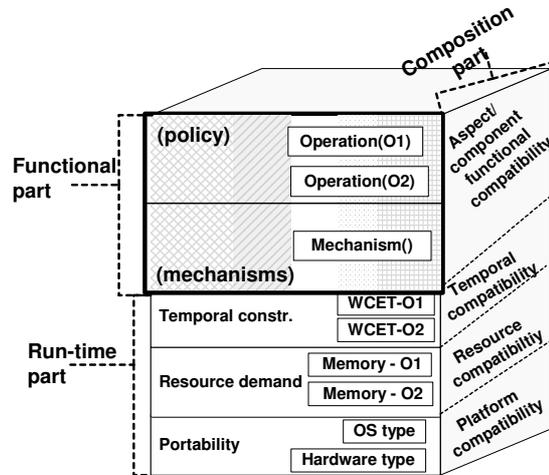


Figure 8.3: A real-time component model (RTCOM)

added to components, and therefore, to the overall real-time system, further improving reconfigurability of the system and its integration with the run-time environment.

8.4 Real-Time Component Model (RTCOM)

In this section we present RTCOM, which can be viewed as a component colored with aspects, both inside (application aspects) and outside (run-time and composition aspects). RTCOM is a language-independent component model, consisting of the following parts (see figure 8.3): (i) the functional part, (ii) the run-time system dependent part, and (iii) the composition part.

RTCOM represents a coarse-granule component model as it provides a broad infrastructure within its functional part. This broad infrastructure enables reconfiguring of a component through aspect weaving, thereby changing the functionality and the behavior of the component to suit the needs of a specific application. RTCOM components are grey box components as they are encapsulated in interfaces, but changes to their behavior can be performed in well-defined places of the component structure via aspect weaving. In contrast, traditional component models are typically black box, fine-grained, and allow only limited configuration of a component (see [18] for an overview of component models). Although a fine-grained component is often more optimal for one particular application in terms of code size, it does not allow component tailoring for various applications, but merely fine-tuning of the restricted set of parameters in the component [18]. For each component designed and implemented based on RTCOM, the functional part of a compo-

ment is first implemented together with application aspects. Then, the run-time system dependent part is defined, followed by the composition part and rules for composing different components and application aspects. In this section we focus on reconfigurability via application aspect weaving, and hence, discuss primarily the functional part of RTCOM. Detailed descriptions of remaining parts of RTCOM, focusing on the run-time part and the run-time aspects, can be found in [5, 6].

8.4.1 Functional Part of RTCOM

Within RTCOM we define a component as follows.

Definition 1 (Component) *A component c is a tuple $\langle M, O, I \rangle$, where M is a set of mechanisms encapsulated by component c , O is a set of operations of component c , and I is a set of component interfaces.*

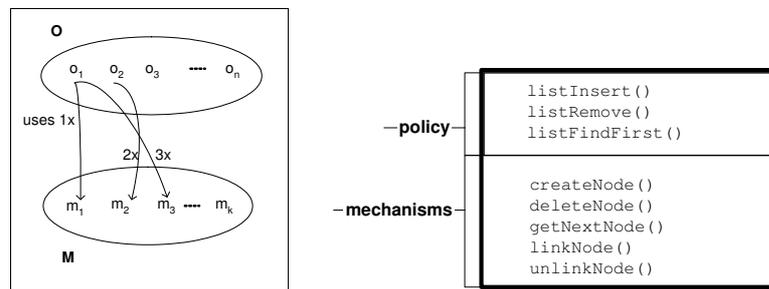
The following are the follow-up definitions and extensive elaboration on each of the constituents of the definition 1 using the introduced notation. To define the functional part of RTCOM, we first need to define the notion of mechanisms and operations of a component, as follows.

Definition 2 (Mechanisms) *A set of mechanisms M of component c is a non-empty set of functions encapsulated by component c .*

Definition 3 (Operations) *A set of operations O of component c is a set of functions implemented in c where for each operation $o \in O$ there exists a non-empty subset of mechanisms $K \subseteq M$, a subset of operations L from other components ($C \setminus \{c\}$), and a mapping such that $o = f(K, L)$.*

The implication of definition 2 is the establishment of mechanisms as fine-granule methods or functions of each component. Definition 3 implies that each component provides a set of operations to other components and/or to the system. Operations can be viewed as coarse-granule methods or function calls as they are implemented using the underlying component mechanisms. Additionally, each operation within the component can call any number of operations provided by other components in the system. An example of how operations and mechanisms could be related in a component is given in figure 8.4(a). For example, operation $o_1 \in O$ is implemented using the subset of component mechanisms $\{m_1, m_3\}$, while operation o_2 is implemented using the subset $\{m_2\}$ of component mechanisms. Furthermore, each operation in the component can use a mechanism in its implementation one or several times, e.g., o_1 uses m_1 once and m_3 three times.

The functional part of RTCOM represents the actual code implemented in the component, and is characterized by definition 4.



(a) Operations and mechanisms

(b) The functional part of the linked list component

Figure 8.4: The functional part of a component (a) relationship between operations and mechanisms and (b) an example of linked list component

Definition 4 (Functional part) *Let c belong to a well-formed component set C . Then the functional part of component c is represented by the tuple $\langle M, O \rangle$, where M is the set of mechanisms of the component and O is the set of operations implemented by the component.*

The functional part of RTCOM, its constituents, their relationship and properties introduced so far, we illustrate through a small example of an ordinary linked list implemented based on RTCOM. The functional part of the linked list component, i.e., the code, consists of component mechanisms and operations (as prescribed by definition 4). The mechanisms needed are the ones for the manipulation of nodes in the list, i.e., `createNode`, `deleteNode`, `getNextNode`, `linkNode`, and `unlinkNode` (see figure 8.4(b)). Operations in the linked list component, namely `listInsert`, `list-`

`Remove` and `listFindFirst`, are implemented using the underlying component mechanisms. In this example, `listInsert` uses the mechanisms `createNode` and `linkNode` to create and link a new node into the list in first-in-first-out (FIFO) order. Thus, the implementation of the operations in a component defines the behavior of the component. Here, the component provides a FIFO ordering of nodes in the list, and, hence, exhibits the FIFO component policy.

The policy of a component can be changed or modified by weaving of application aspects into the component code, i.e., functional part of the component. Therefore, application aspects are directly dependent on the functional part of RTCOM. The definition of application aspects is influenced by two requirements: (i) preserving information hiding of the component to the largest extent possible, and (ii) enabling temporal analysis of the resulting woven components. To satisfy the two

requirements we utilize the notion of mechanisms as building blocks of application aspects, and provide application aspects as follows.

Definition 5 (Application aspects) *An application aspect $a \in A$ is a set of tuples $\langle a^t, P \rangle$ where:*

- $t \in \{before, after, around\}$;
- a^t is an advice of type t defined by mapping $a^t = f(K)$, $K \subseteq M$; and
- P is a set of pointcuts that describes the subset of operations in components that can be preceded, succeeded, or replaced by advice a^t depending on the type of the advice.

Definition 5 extends the traditional definition of programming-language level aspects by specifying pointcuts and advices in terms of mechanisms and operations. This enables performing temporal analysis of the woven system, and thereby use of aspects in real-time environments. This also enables existing aspect languages to be used for implementing application aspects in real-time systems, and enables existing weavers to be used to integrate aspects into components while maintaining predictability of the real-time system. In RTCOM, pointcuts refer to operations, implying that a pointcut in an application aspect points to one or several operations of a component where modifications of the component code are allowed. Having mechanisms of the components as basic building blocks of the advices is a decisive factor in enabling temporal analysis of the resulting woven code (see [19] for details on performing worst-case execution time analysis). Furthermore, the implementation of a whole application aspect is not limited only to mechanisms of one component since an aspect can contain any finite number of advices that can precede, succeed, or replace operations through out the system configuration. Advices and, hence, application aspects can be implemented using the mechanisms from a number of components.

Assume that we want to change the policy of the linked list component given in figure 8.4(b) from FIFO to priority-based ordering of the nodes. This can be achieved by weaving an appropriate application aspect. Figure 8.5 shows the `listPriority` application aspect, which consists of one named pointcut `listInsertCall`, identifying `listInsert` operation of the component as a join point in the component code (lines 2-3). The `listInsertCall` before advice is implemented using the component mechanism `getNextNode` to determine the position of the node based on its priority (lines 5-14). Weaving of the `listPriority` application aspect into the code of the linked list component would result in a component where each execution of the operation `listInsert` is

```

aspect listPriority{
1:
2: pointcut listInsertCall(List_Operands * op)=
3:   call("void listInsert(List_Operands*)")&&args(op);
4:
5: advice listInsertCall(op):
6:   void before(List_Operands * op){
7:     while
8:       the node position is not determined
9:     do
10:      node = getNextNode(node);
11:      /* determine position of op->node based
12:       on its priority and the priority of the
13:       node in the list*/
14:    }
15: }

```

Figure 8.5: The listPriority application aspect

preceded by the execution of the advice `listInsertCall`, i.e., before placing the node into the list its position is determined based on its priority.

Expressing semantics of aspect weaving formally (definition 7) requires an introduction of the mathematical interpretation of the sequential execution of two code fragments in the following manner. Given that x and y represent two code fragments, and xy denotes their sequential compositions (resulting from a textual concatenation of the two pieces of code), then we can define the mathematical interpretation of the code xy as follows.

Definition 6 *Let x and y be two pieces of code (two sequences of statements in some programming language). Let o and o' be the mathematical representation of x and y , respectively. Then we denote the mathematical representation of the code xy by $glue(o, o')$.*

Using the formal notation introduced so far, we can formally express the semantics of application aspect weaving as follows.

Definition 7 (Weaving of application aspects) *Let $a = \langle a^t, P \rangle$ be an application aspect where $a^t = f(K)$, $K \subseteq M$, and P is a set of pointcuts, $P \subseteq O$. Weaving of application aspect $a \in A$ in the component $c = \langle M, O, I \rangle$, results in a component $c' = \langle M, O', I \rangle$ where for all $o'_i \in O'$ the following holds:*

- if $o_i \in O \setminus P$ then $o'_i = o_i$
- if $o_i \in P$ then

$$o'_i = \begin{cases} glue(a^t, o_i) & \text{if } t = \text{before} \\ glue(o_i, a^t) & \text{if } t = \text{after} \\ a^t & \text{if } t = \text{around} \end{cases}$$

For example, assume that we have a component $c = \langle M, O, I \rangle$ such that M is the set of mechanisms, $O = \{o_1, \dots, o_6\}$ is the set of operations, and I the set of component interfaces. Then, weaving of an application aspect a , consisting of advices a_1^{before} , a_2^{after} , and their respective pointcut sets, $P_1 = \{o_1, o_3\}$ and $P_2 = \{o_6\}$, would result in the component $c' = \langle M, O', I \rangle$ where

$$O' = \{glue(a_1^{before}, o_1), o_2, glue(a_1^{before}, o_3), o_4, o_5, glue(o_6, a_2^{after})\}.$$

Hence, in component c' , the execution of operations o_1 and o_3 are preceded by the execution of the code of advice a_1^{before} , and the execution of operation o_6 is succeeded by the execution of advice a_2^{after} . Operations o_2 , o_4 , and o_5 remain unchanged.

Operations are flexible parts of the component as their implementation can change by weaving application aspects, while mechanisms are fixed parts of the component infrastructure. Since advices are implemented using the mechanisms of the components, each advice can change the behavior of the component by changing one or more operations in the component.

To enable easy implementation of application aspects into a component, the design of the functional part of the component is performed in the following manner. First, mechanisms, as basic blocks of the component, are implemented. Here, particular attention should be given to the identified application aspects, and the table that reflects the cross-cutting effects of application aspects to different components should be made to help the designer in the remaining steps of the RTCOM design and implementation. Next, the operations of the component are implemented using component mechanisms (see definition 3). Note that the implemented operations provide an initial component policy, i.e., basic and somewhat generic component functionality. This initial policy we denote a *policy framework* of the component. The policy framework could be modified by weaving different application aspects to change the component policy.

The development process of the functional part of a component results in a component colored with application aspects. Therefore, in the graphical view of RTCOM in figure 8.3, application aspects are represented as vertical layers in the functional part of the component as they influence component behavior, i.e., change component policy.

8.5 COMET: a COMponent-based Embedded real-Time database

This section shows how to apply the introduced concept of aspectual component-based real-time system development on a design and development of a concrete real-time system by presenting the application of the design method to development of a configurable real-time embedded database system, called COMET.

8.5.1 Background

The goal of the COMET project is to enable development of a configurable real-time database for embedded systems, i.e., enable development of different database configurations for different embedded and real-time applications. The types of requirements we are dealing with can best be illustrated on the example of one of the COMET target applications: control systems in the automotive industry. These systems are typically hard real-time safety-critical systems consisting of several distributed nodes implementing a specific functionality. Although nodes depend on each other and collaborate to provide required behavior for the overall vehicle control system, each node can be viewed as a stand-alone real-time system. The size of the nodes can vary significantly, from very small nodes to large nodes. For instance, a vehicle control system could consist of a small number of resource adequate nodes responsible for the overall performance of the vehicle, e.g., 32-bit CPUs with a few Mb of RAM, and a large number of nodes responsible for controlling specific subsystems in the vehicle, which are significantly resource-constrained, e.g., an 8-bit micro-controller and a few Kb of RAM [4].

Depending on the functionality of a node and the available memory, different database configurations are preferred. In safety-critical nodes tasks are often non-preemptive and scheduled off-line, avoiding concurrency by allowing only one task to be active at any given time. This, in turn, influences functionality of a database in a given node with respect to concurrency control. Less critical nodes having preemptable tasks would require concurrency control mechanisms. Furthermore, some nodes require critical data to be logged, e.g., warning and errors, and require backups on startup and shutdown, while other nodes only have RAM (main-memory), and do not require non-volatile backup facilities from the database. Hence, to provide support for data management in these types of systems we need to enable development of different database configurations to suit the needs of each node with respect to memory consumption, concurrency control, recovery, scheduling techniques, transaction models, and storage models. In the following

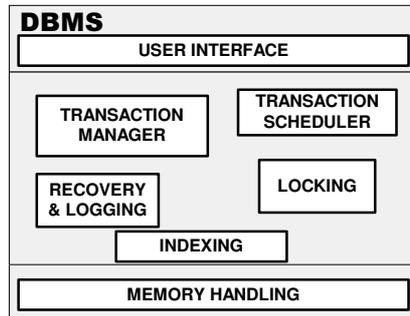


Figure 8.6: COMET decomposition into a set of components

sections we show how we have reached this goal by applying ACCORD to the design and development of the COMET system.

8.5.2 COMET Components

Following the ACCORD design method presented in section 8.3 we have first performed the decomposition of COMET into a set of components with well-defined functions and interfaces. COMET has seven basic components (see figure 8.6): user interface component, transaction scheduler component, locking component, indexing component, recovery and logging component, memory handling component, and transaction manager component.

The *user interface component* (UIC) enables users to access data in the database, and different applications often require different ways of accessing data in the system. All the operations on data in the database are received via the UIC. The main activities of the UIC consist of receiving and parsing the incoming requests from the application and the user. UIC takes the incoming requests and devises the execution plans.

The *transaction scheduler component* (TSC) provides mechanisms for performing scheduling of transactions coming into the system, based on the scheduling policy chosen. COMET is designed to support a variety of scheduling policies, e.g., EDF and RM [20]. The TSC is also in charge of maintaining the list of all transactions in the system, including scheduled transactions as well as unscheduled but active transactions, i.e., transactions submitted for execution. Hard real-time applications, such as real-time embedded systems controlling a vehicle, typically do not require sophisticated transaction scheduling and concurrency control, i.e., the system allows only one transaction to access the database at a time [4]. Therefore, the TSC should be a flexible and exchangeable part of the database architecture.

The *locking component* (LC) deals with locking of data, and it provides mechanisms for lock manipulation and maintains lock records in the database. The LC provides the policy framework for the lock administration in which all locks are granted. This policy framework can be changed into a specific policy according to which the LC deals with lock conflicts by weaving concurrency control aspect (see section 8.5.4).

The *indexing component* (IC) deals with indexing of data. Indexing strategies could vary depending on the real-time application with which the database should be integrated, e.g., multi-versioning suitable for applications with a large number of read-only transactions [21]. Additionally, it is possible to customize an indexing strategy depending on the number of transactions active in the system and the indexing algorithm needed.

The *recovery and logging component* (RLC) is in charge of recovery and logging of data in the database. As COMET stores data in main-memory, there is a need for different recovery and logging techniques, depending on the type of the storage, e.g., non-volatile EEPROM or Flash.

The *memory handling component* (MHC) manages access to data in the physical storage. For example, each time a tuple is added or deleted, the MHC is invoked to allocate and release memory. Generally, all reads or writes to/from the memory in COMET involve the MHC.

The *transaction manager component* (TMC) coordinates the activities of all components in the system with respect to transaction execution. For example, the TMC manages the execution of a transaction by requesting lock and unlock operations provided by the LC, followed by requests to the operations, which are provided by the IC, for inserting or updating data items.

8.5.3 COMET Aspects

Following ACCORD, after decomposing the system into a set of components with well-defined interfaces, we decompose the system into a set of aspects. The decomposition of COMET into aspects fully corresponds to the ACCORD decomposition (given in section 8.3.1) into three types of aspects: run-time, composition, and application aspects. However, as COMET is a real-time database system, refinement to the application aspects is made to reflect both real-time and database issues. Hence, in the COMET decomposition of application aspects, the real-time policy aspects include real-time scheduling and concurrency control policy, while the real-time property aspect (in ACCORD) is the transaction model aspect, which is database-specific. The crosscutting effects of the application aspects to COMET components are shown in the table 8.1. Note that application aspects can also crosscut or depend on

Table 8.1: Crosscutting effects of different application aspects on the COMET components

Components Application aspects	UIC	TSC	LC	IC	RLC	MHC	TMC
Transaction	X	X	X	X	X	X	X
Real-time scheduling		X					X
Concurrency control	X	X	X				X
Memory optimization	X	X	X	X	X		X
Synchronization		X	X	X	X		X
Security	X		X	X		X	X

other application aspects. In this chapter however, we primarily focus on crosscutting effects of application aspects to different components. For more details on dependencies and inter-relationships of aspects we refer interested readers to [22, 23].

As can be seen from table 8.1, all identified application aspects cross-cut more than one component. For example, the concurrency control (CC) aspect crosscuts several components, namely TSC, LC, and TMC in the following manner. The TMC is responsible for invoking the LC to obtain and release locks. The way the LC is invoked by the TMC depends on the CC policy enforced in the database and, hence, needs to be adjusted separately for each type of CC policy, i.e., each type of the CC aspect. Furthermore, the way to deal with lock conflicts is enforced by the LC. Hence, the LC should be modified with CC aspect to facilitate lock resolution policy prescribed by the CC policy of the CC aspect. Since scheduling and CC are tightly coupled in the sense that CC polices typically require information about the transactions in the system maintained by the TSC, this means that the TSC should be modified by CC aspect to provide adequate support for the chosen CC policy.

The application aspects could vary depending on the particular application of the real-time system, thus, particular attention should be made to identify the application aspects for each real-time system.

8.5.4 COMET RTCOM

Components and aspects in COMET are implemented based on RTCOM (discussed in section 8.4). Hence, the functional part of components is implemented first, together with application aspects. We illustrate this

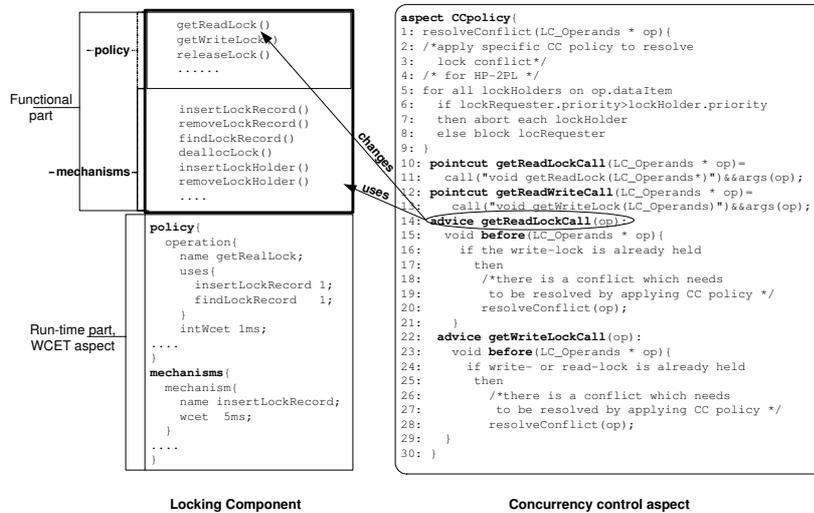


Figure 8.7: The locking component and the concurrency control aspect

process, its benefits and drawbacks, by the example of one component (namely the LC) and one application aspect (namely the CC aspect).

The LC performs the following functionality: assigning locks to requesting transactions and maintaining a lock table, thus, it records all locks obtained by transactions in the system. As can be seen from the table 8.1, the LC is crosscut with several application aspects. The application aspect that influences the policy, i.e., changes the behavior of the LC, is a CC aspect, which defines the way lock conflicts should be handled in the system. To enable tailorability of the LC, and reuse of code in the largest possible extent, the LC is implemented with the policy framework in which lock conflicts are ignored and locks are granted to all transactions. The policy framework can be modified by weaving CC aspects that define other ways of handling lock conflicts. As different CC policies in real-time database systems exist, the mechanisms in the LC should be compatible with most of the existing CC algorithms.

The LC contains mechanisms such as (see left part of the figure 8.7): `insertLockRecord()`, `removeLockRecord()`, etc., for maintaining the table of all locks held by transactions in the system. The policy part consists of the operations performed on lock records and transactions holding and/or requesting locks, e.g., `getReadLock()`, `getWriteLock()`, `releaseLock()`. The operations in the LC are implemented using underlying LC mechanisms. The mechanisms provided by the LC are used by the CC aspects implementing the class of pessimistic (locking) protocols, e.g., HP-2PL [24] and RWPCP [25]. However, as a large class of optimistic protocols are implemented using locking mechanisms, the mechanisms provided

by the LC can also be used by CC aspects implementing optimistic protocols, e.g., OCC-TI [26] and OCC-APR [27].

The right part of the figure 8.7 represents the specification for the real-time CC aspect (lines 1-30) that can be applied to a class of pessimistic locking CC protocols. We chose to give more specific details for the HP-2PL protocol, as it is both commonly used in main-memory database systems and a well-known pessimistic CC protocol.

The CC aspect has several pointcuts and advices that execute when the pointcut is reached. As defined by the RTCOM pointcut model, the pointcuts refer to the operations: `getReadLockCall()` and `getWriteLockCall()` (lines 10 and 12). The first pointcut intercepts the call to the function `getReadLock()`, which grants a read lock to the transaction and records it in the lock table. Similarly, the second pointcut intercepts the call to the function that gives a write lock to the transaction and records it in the lock table. Before granting a read or write lock, the advices in lines 14-21 and 22-29 check if there is a lock conflict. If a conflict exists, the advices deal with it by calling the local aspect function `resolveConflict()` (lines 1-9), where the resolution of the conflict should be done by implementing a specific CC policy. As this function is called from the advices it can be considered a part the body of each advice (equivalent would be to place the code of the function in each advice separately). Furthermore, `resolveConflict()` traverses the list of transactions holding a lock using underlying mechanisms of the LC. Hence, the overall advices are implemented using mechanisms of the LC to traverse the lock table (lines 16-19 and 24-27) and the list of transactions holding a lock (in the function `resolveConflict()`).

So far we have shown that the CC aspect affects the policy of the LC, but the CC aspect also crosscuts other components (see table 8.1). In the example of the CC aspect implementing pessimistic HP-2PL protocol (see figure 8.7), the aspect uses the information about transaction priority (lines 5-8), which is maintained by the TSC, thus crosscutting the TSC. Optimistic protocols, e.g., OCC-TI, would require additional pointcuts to be defined in the TMC, as the protocol (as compared to pessimistic protocols) assumes execution of transactions in three phases: read, validate and write.

Additionally, depending on the CC policy implemented, the number of pointcuts and advices varies. For example, some CC policies (like RWPCP, or optimistic policies) require additional data structures to be initialized. In such cases, an additional pointcut named `initPolicy()` could be added to the aspect that would intercept the call to initialize the LC. A before advice `initPolicy` would then initialize all necessary data structures in the CC aspect after data structures in the LC have been initialized.

8.6 Related Work

In this section we address the research in the area of component-based real-time and database systems, and the real-time and database research projects that are using aspects to separate concerns.

The focus in existing component-based real-time systems is enforcement of real-time behavior. In these systems a component is usually mapped to a task, e.g., passive component [1], binary component [28], and port-based object component [29]. Therefore, analysis of real-time components in these solutions addresses the problem of temporal scopes at a component level as task attributes [28, 1, 29]: worst-case execution time, release time, deadline. ACCORD with its RTCOM model supports mapping of a component to a task, and takes a broader view of the composition process by allowing real-time systems to be composed out of tasks and components that are not necessarily mapped to a task. ACCORD, in contrast to other approaches building real-time component-based systems [28, 1, 29], enables support for multidimensional separation of concerns and allows integration of aspects into the component code. VEST [1, 30] indeed uses an aspect-oriented paradigm but does not provide a component model that enables weaving of application aspects into the component code, rather it focuses on composition aspects.

In the area of database systems, the aspect-oriented databases (AOD) initiative aims at bringing the notion of separation of concerns to databases. The focus of this initiative is on providing a non-real-time database with limited configurability using only aspects (i.e., no components) [31]. To the best of our knowledge, KIDS [32] is the only research project focusing on construction of a configurable database composed out of components (database subsystems), e.g., object management and transaction management. Commercial component-based databases introduce limited customization of the database servers [33, 34], by allowing components for managing non-standard data types, data cartridges and DataBlade modules, to be plugged into a fully functional database system. A somewhat different approach to componentization is Microsoft's Universal Data Access Architecture [35], where the components are data providers and they wrap data sources enabling the translation of all local data formats from different data stores to a common format. However, from a real-time point of view none of the component-based database approaches discussed enforce real-time behavior and use aspects to separate concerns in the system.

Existing real-time design methods [36, 37, 38, 39, 1, 30] focus on task structuring and two different views on the system, temporal and structural, with moderate emphasis on the information hiding. The analysis of the real-time system under design, although missing from

early design approaches [36, 37], has been highlighted as important for real-time system development [38, 39, 1, 30]. Furthermore, configuration guidelines and tools for system decomposition and configuration have been an essential part of all design methods for real-time systems so far and have, more or less, been enforced by all existing real-time design methods. RT-UML [40] is an example of an infrastructure that provides configuration tools in the form of a visual language. Note, however, that RT-UML cannot be considered a design method as it essentially provides only syntax, not semantics, for the real-time system design, e.g., its powerful expressiveness could be used by a design method as means of specifying real-time software components [41].

In contrast to real-time design methods, modern software engineering design methods [42, 43, 44, 2] primarily focus on the component model, strong information hiding, and interfaces as means of component communication. Also, the notion of separation of concerns is considered to be fundamental in software engineering as it captures aspects of the software system early in the system design [17, 44, 16, 15, 45].

It can be observed that there is a gap between the design approaches from different communities as the real-time community has focused primarily on real-time issues not exploiting modularity of software to the extent that the software engineering community has done. ACCORD helps in bridging this gap as it provides support for aspects and aspect weaving into the code of the components, efficient component and system tailoring, and better reusability and flexibility of real-time software - the issues that have not been fully addressed by existing real-time design approaches.

8.7 Summary

In recent years, one of the key research challenges in the software engineering research community has been enabling configuration of systems and reuse of software by composing systems using components from a component library. Our research focuses on applying aspect-oriented and component-based software development to real-time system development by introducing a novel concept of aspectual component-based real-time system development (ACCORD). In this chapter we presented ACCORD and its elements, which we have applied in the development of a real-time database system, called COMET. ACCORD introduces the following into real-time system development: (i) a design method, which enables improved reuse and configurability of real-time and database systems by combining basic ideas from component-based and aspect-oriented communities with real-time concerns, thus bridging the gap between real-time systems, embedded systems, database systems, and

software engineering, (ii) a real-time component model, called RTCOM, which enables efficient development of configurable real-time systems, and (iii) a new approach to modeling real-time policies as aspects improving the flexibility of real-time systems. In the COMET example we have shown that applying ACCORD could have an impact on the real-time system development in providing efficient configuration of real-time systems, improved reusability and flexibility of real-time software, and modularization of crosscutting concerns.

The work presented in this chapter is based on the work done under ARTES financing [5, 6, 7], encompassing a period of two years, December 2000 until January 2003. In recent years, however, we have extended and refined the work on the ACCORD framework and the COMET platform in a number of directions as follows.

- We have developed a formalization method for reconfigurable real-time components as a part of the continued work on RTCOM [46]. The method enables proving that the reconfiguration of components via aspect weaving provides expected functional and temporal behavior in the reconfigured component.
- To support a designer in configuration and analysis of real-time and embedded systems composed using components and aspects, we have developed a tool set denoted ACCORD development environment [47]. The tool set consists of the following tools, all developed as stand-alone tools and integrated with the environment: (i) ACCORD modeling environment that enables the configuration and checking of the functional correctness of the system, and (ii) aspect-level worst case execution time and memory footprint analyzer that enables worst-case execution time and memory footprint analysis of the system configured out of aspects and components.
- For guaranteeing the performance of a real-time system composed using aspects and components in open and unpredictable soft real-time environments, we have developed a method for providing reconfigurable quality of service (QoS) management [48]. Using this method, QoS of the system, e.g., deadline miss ratio and utilization, is maintained using a feedback control loop. Furthermore, the feedback control-based QoS policies can be reconfigured to suit the target application by using aspects and components.
- We developed a QoS-aware framework for supporting dynamic (on-line) reconfiguration of a system, i.e., adding, removing, or exchanging components, while preserving real-time performance [49]. The framework and its mechanism for dynamic reconfiguration is especially useful for embedded and real-time systems requiring constant upgrades in response to changes in the environment

or changes to the system goals. The framework also ensures that the real-time performance in terms of desired QoS is maintained after dynamic reconfiguration.

At the same time, the COMET platform has evolved and currently includes a great number of components and aspects enabling configuration of over 30 different distinct database systems. The configurations range from configurations with different concurrency control algorithms, to configurations providing various QoS management policies, and configurations providing active behavior in terms of event-condition-action rules [50].

Bibliography

- [1] Stankovic, J.: VEST: a toolset for constructing and analyzing component based operating systems for embedded and real-time systems. In: Proceedings of the 1st International Conference on Embedded Software, (EMSOFT'01). Volume 2211 of Lecture Notes in Computer Science., Springer-Verlag (2001) 390–402
- [2] Szyperski, C.: Component Software - Beyond Object-Oriented Programming. Addison-Wesley (1999)
- [3] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In: Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97). Volume 1241 of Lecture Notes in Computer Science., Springer-Verlag (1997) 220–242
- [4] Nyström, D., Tešanović, A., Norström, C., Hansson, J., Bånkestad, N.E.: Data management issues in vehicle control systems: a case study. In: Proceedings of the 14th IEEE Euromicro International Conference on Real-Time Systems (ECRTS'02), IEEE Computer Society Press (2002) 249–256
- [5] Tešanović, A., Nyström, D., Hansson, J., Norström, C.: Towards aspectual component-based real-time systems development. In: Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA'03). Volume 2968 of Lecture Notes in Computer Science., Springer-Verlag (2003)
- [6] Tešanović, A., Nyström, D., Hansson, J., Norström, C.: Aspects and components in real-time system development: Towards reconfigurable and reusable software. *Journal of Embedded Computing* **1** (2004)

- [7] Tešanović, A.: Towards aspectual component-based real-time system development. Licentiate Thesis, Department of Computer Science, Linköping University (2003), ISBN 91-7373-681-3.
- [8] Bosch, J.: Design and Use of Software Architectures. ACM Press in collaboration with Addison-Wesley (2000)
- [9] Crnkovic, I., Larsson, M.: A case study: Demands on component-based development. In: Proceedings of 22th International Conference of Software Engineering, Limerick, Ireland, ACM (2000) 23–31
- [10] Crnkovic, I., Larsson, M., Lüders, F.: State of the practice: Component-based software engineering course. In: Proceedings of 3rd International Workshop of Component-Based Software Engineering, IEEE Computer Society (2000)
- [11] Fleisch, W.: Applying use cases for the requirements validation of component-based real-time software. In: Proceedings of 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'99), IEEE Computer Society Press (1999) 75–84
- [12] Freidrich, L., Stankovic, J., Humphrey, M., Marley, M., Haskins, J.: A survey of configurable, component-based operating systems for embedded applications. *IEEE Micro* **21** (2001) 54–68
- [13] Meyer, B., Mingins, C.: Component-based development: From buzz to spark. *IEEE Computer* **32** (1999) 35–37 Guest Editors' Introduction.
- [14] Dittrich, K.R., Geppert, A.: Component Database Systems: Introduction, Foundations, and Overview. In: Component Database Systems. Morgan Kaufmann Publishers (2000)
- [15] Coady, Y., Kiczales, G., Feeley, M., Smolyn, G.: Using AspectC to improve the modularity of path-specific customization in operating system code. In: Proceedings of the Joint European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9). (2002)
- [16] Spinczyk, O., Gal, A., Schröder-Preikschat, W.: AspectC++: an aspect-oriented extension to C++. In: Proceedings of the 40th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS'02), Sydney, Australia, Australian Computer Society (2002)

- [17] Xerox Corporation: The AspectJ Programming Guide. (2002)
Available at: <http://aspectj.org/doc/dist/progguide/index.html>.
- [18] Crnkovic, I., Larsson, M., eds.: Building Reliable Component-Based Real-Time Systems. Artech House Publishers (2002)
- [19] Tešanović, A., Nyström, D., Hansson, J., Norström, C.: Integrating symbolic worst-case execution time analysis into aspect-oriented software development. OOPSLA 2002 Workshop on Tools for Aspect-Oriented Software Development (2002)
- [20] Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM* **20** (1973) 46–61
- [21] Rastogi, R., Seshadri, S., Bohannon, P., Leinbaugh, D.W., Silberschatz, A., Sudarshan, S.: Improving predictability of transaction execution times in real-time databases. *Journal of Real-Time Systems* **19** (2000) 283–302
- [22] Kienzle, J., Yu, Y., Xiong, J.: On composition and reuse of aspects. In: In Proceedings of the Workshop on Foundations of Aspect-Oriented Languages (FOAL 2003), Boston, USA (2003)
- [23] Sipma, H.: A formal model for cross-cutting modular transition systems. In: In Proceedings of the Workshop on Foundations of Aspect-Oriented Languages (FOAL'03), Boston, USA (2003)
- [24] Abbott, R.K., Garcia-Molina, H.: Scheduling real-time transactions: a performance evaluation. *ACM Transactions on Database Systems* **17** (1992) 513–560
- [25] Sha, L., Rajkumar, R., Son, S.H., Chang, C.H.: A real-time locking protocol. *IEEE Transactions on Computers* **40** (1991) 793–800
- [26] Lee, J., Son, S.H.: Using dynamic adjustment of serialization order for real-time database systems. In: Proceedings of the 14th IEEE Real-Time Systems Symposium. (1993)
- [27] Datta, A., Son, S.H.: Is a bird in the hand worth more than two birds in the bush? Limitations of priority cognizance in conflict resolution for firm real-time database systems. *IEEE Transactions on Computers* **49** (2000) 482–502
- [28] Isovich, D., Lindgren, M., Crnkovic, I.: System development with real-time components. In: Proceedings of the Workshop on Pervasive Component-Based Systems at the 14th European Conference on Object-Oriented Programming (ECOOP'00), France (2000)

- [29] Stewart, D.S.: Designing software components for real-time applications. In: Proceedings of Embedded System Conference, San Jose, CA (2000) Class 408, 428.
- [30] Stankovic, J., Zhu, R., Poornalingam, R., Lu, C., Yu, Z., Humphrey, M., Ellis, B.: VEST: an aspect-based composition tool for real-time systems. In: Proceedings of the 9th IEEE Real-Time Applications Symposium (RTAS'03), IEEE Computer Society Press (2003) 58–69
- [31] Rashid, A., Pulvermuller, E.: From object-oriented to aspect-oriented databases. In: Proceedings of the 11th International Conference on Database and Expert Systems Applications (DEXA'00). Volume 1873 of Lecture Notes in Computer Science., Springer-Verlag (2000) 125–134
- [32] Geppert, A., Scherrer, S., Dittrich, K.R.: KIDS: Construction of database management systems based on reuse. Technical Report ifi-97.01, Department of Computer Science, University of Zurich (1997)
- [33] Oracle Corporation: All your data: The Oracle extensibility architecture. Oracle Technical White Paper. Redwood Shores, CA (1999)
- [34] Informix Corporation: Developing DataBlade modules for Informix-Universal Server. Informix DataBlade Technology (2001) Available at <http://www.informix.com/datablades/>.
- [35] OLE DB Technical Materials: Universal data access through OLE DB. OLE DB White Papers (2001) Available at <http://www.microsoft.com/data/techmat.htm>.
- [36] Gomaa, H.: A software design method for real-time systems. Communications of the ACM **27** (1984) 938–949
- [37] Gomaa, H.: A software design method for Ada based real time systems. In: Proceedings of the 6th Washington Ada Symposium, ACM Press (1989) 273–284
- [38] Kopetz, H., Zainlinger, R., Fohler, G., Kantz, H., Puschner, P., Schütz, W.: The design of real-time systems: from specification to implementation and verification. Software Engineering Journal **6** (1991) 72–82
- [39] Burns, A., Wellings, A.: HRT-HOOD: a Structured Design Method for Hard Real-Time Ada Systems. Volume 3 of Real-Time Safety Critical Systems. Elsevier (1995)

- [40] Douglass, B.P.: *Real-Time UML: Developing Efficient Objects for Embedded Systems*. Addison-Wesley (2000)
- [41] Crnkovic, I., Hnich, B., Jonsson, T., Kiziltan, Z.: Specification, implementation, and deployment of components. *Communications of the ACM* **45** (2002) 35–40
- [42] Dogac, A., Dengi, C., Öszu, M.T.: Distributed object computing platform. *Communications of the ACM* **41** (1998) 95–103
- [43] Öszu, M.T., Yao, B.: *Building Component Database Systems Using CORBA. Data Management Systems*. In: *Component Database Systems*. Morgan Kaufmann Publishers (2000)
- [44] Aßmann, U.: *Invasive Software Composition*. Springer-Verlag (2002)
- [45] Aksit, M., Bosch, J., van der Sterren, W., Bergmans, L.: Real-time specification inheritance anomalies and real-time filters. In: *Proceedings of the 8th European Conference on Object-Oriented Programming (ECOOP'94)*. Volume 821 of *Lecture Notes in Computer Science*., Springer-Verlag (1994) 386–407
- [46] Tešanović, A., Nadjm-Tehrani, S., Hansson, J.: Modular Verification of Reconfigurable Components. In: *Component-Based Software Development for Embedded Systems - An Overview on Current Research Trends*. Volume 3778 of *Lecture Notes in Computer Science*. Springer-Verlag (2005) 59–81
- [47] Tešanović, A., Mu, P., Hansson, J.: Development environment for configuration and analysis of embedded real-time systems. In: *Proceedings of the 4th International Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS'05)*. (2005)
- [48] Tešanović, A., Amirijoo, M., Björk, M., Hansson, J.: Empowering configurable QoS management in real-time systems. In: *Proceedings of the Fourth ACM SIG International Conference on Aspect-Oriented Software Development (AOSD'05)*, ACM Press (2005) 39–50
- [49] Tešanović, A., Amirijoo, M., Nilsson, D., Norin, H., Hansson, J.: Ensuring real-time performance guarantees in dynamically reconfigurable embedded systems. In: *Proceedings of the IFIP International Conference on Embedded and Ubiquitous Computing*. Volume 3824 of *Lecture Notes in Computer Science*., Springer-Verlag (2005) 131–141

- [50] Tešanović, A.: Developing Reusable and Reconfigurable Real-Time Software using Aspects and Components. PhD thesis, Department of Computer and Information Science, Linköping University, Sweden (2006)

Part II

Modeling and Verification

Chapter 9

Introduction

By **Wang Yi**

Division of Computer Systems
Department of Information Technology
Uppsala University
Email: yi@docs.uu.se

In the past decade, there has been an increasing interest in formal modeling and verification in academia as well as industry. Several major advances have been made in the area. The technology is emerging with traditional techniques for system development and validation e.g. testing in hardware construction, protocol design and real time embedded systems. In this chapter, we collect contributions from the ARTES community to the topic. The chapter contains four articles covering theoretical studies, techniques, software tools and industrial applications. The first article presents the theory, algorithms and data structures behind UPPAAL, a model checker for real time systems developed jointly by Uppsala University and Aalborg University. The second article studies an alternative model for timed systems, namely Timed Petri Nets (TPNs). It shows essentially the undecidability of liveness properties for the TPN model. The third article applies the theory of timed automata to solve scheduling problems. The last article is an application of modelling techniques for the development of industrial robotics.

9.1 Paper 1: Modeling and Verification with Timed Automata

Johan Bengtsson and Wang Yi
Department of Information Technology
Uppsala University

This paper provides a tutorial and pointers to results and related work on timed automata with a focus on semantical and algorithmic aspects of modelling and verification tools. We present the concrete and abstract semantics of timed automata (based on transition rules, regions and zones), decision problems, and algorithms for verification. A detailed description on DBM (Difference Bound Matrices) is included, which is the central data structure behind several verification tools for timed systems. As an example, we give a brief introduction to the tool UPPAAL.

9.2 Paper 2: Schedulability Analysis Using Timed Automata

Pavel Krčál and Wang Yi
Department of Information Technology
Uppsala University

This paper studies schedulability problems of timed systems with non-uniformly recurring computation tasks. Assume a set of real time tasks whose best and worst execution times, and deadlines are known. Timed automata is to describe the arrival patterns (and release times) of tasks. From the literature, it is known that the schedulability problem for a large class of such systems is decidable and can be checked efficiently. The paper provides a summary on what is decidable and what is undecidable in schedulability analysis using timed automata. The main technical contribution is that the schedulability problem will be undecidable if these three conditions hold: (1) the execution times of tasks are intervals, (2) a task can announce its completion time, and (3) a task can preempt another task. It is also shown that if one of the above three conditions is dropped, the problem will be decidable. Thus the result can be used as an indication in identifying classes of timed systems that can be analysed efficiently.

9.3 Paper 3: Undecidability of Linear Time Temporal Logic for Timed Petri Nets

Parosh Abdulla, Pritha Mahata and Aletta Nylén
Department of Information Technology
Uppsala University

This paper shows the undecidability of (action based) linear-time temporal logic (LTL) for *Timed Petri Nets (TPNs)*. The model of TPNs is a generalization of standard Petri nets in the sense that each token is equipped with a real-valued clock representing the age of the token. Furthermore each arc in the net is provided with an interval restricting the ages of tokens allowed to travel through the arc. The paper shows that it is undecidable to check a certain fixed property expressible in LTL for TPNs. More precisely, it is an undecidable problem to check whether there exists a computation of the TPN along which a given transition is fired infinitely often. The undecidability result is established through a reduction from a similar problem for the model of *lossy counter machines*.

9.4 Paper 4: A Framework for Analysis of Timing and Resource Utilization targeting Complex Embedded Systems

Johan Andersson, Anders Wall and Christer Norström
Department of Computer Science and Engineering
Mälardalen University

This paper presents the ART Framework, a set of methods and tools for introducing analyzability with respect to timing and resource utilization in development/maintenance of complex embedded software systems. The framework enables a *behavior impact analysis*, in which problems due to side-effects of a new feature can be identified early, before vast resources have been invested in implementation and testing. This can reduce costs and improve system reliability as the risk of introducing timing or resource problems is reduced. The core of the ART Framework is a process for how to construct an analyzable model of an existing system, which enable behavior impact analysis. The paper also presents the modeling language and analysis tools included in the framework, as well as a case study where the framework has been applied to a representative industrial system, an industrial robot controller from ABB.

Chapter 10

Modeling and Verification of Real-Time Systems Using Timed Automata

By **Johan Bengtsson** and **Wang Yi**

Division of Computer Systems

Department of Information Technology

Uppsala University

Email: yi@it.uu.se

This chapter is to provide a tutorial and pointers to results and related work on timed automata with a focus on semantical and algorithmic aspects of verification tools. We present the concrete and abstract semantics of timed automata (based on transition rules, regions and zones), decision problems, and algorithms for verification. A detailed description on DBM (Difference Bound Matrices) is included, which is the central data structure behind several verification tools for timed systems. As an example, we give a brief introduction to the tool UPPAAL.

10.1 Introduction

Timed automata is a theory for modeling and verification of real time systems. Examples of other formalisms with the same purpose, are timed Petri Nets, timed process algebras, and real time logics [BD91, RR88, Yi91, NS94, AH94, Cha99]. Following the work of Alur and Dill [AD90, AD94], several model checkers have been developed with timed automata being the core of their input languages *e.g.* [Yov97, LPY97]. It is fair to say that they have been the driving force for the application and development of the theory. The goal of this chapter is

to provide a tutorial on timed automata with a focus on the semantics and algorithms based on which these tools are developed.

In the original theory of timed automata [AD90, AD94], a timed automaton is a finite-state Büchi automaton extended with a set of real-valued variables modeling clocks. Constraints on the clock variables are used to restrict the behavior of an automaton, and Büchi accepting conditions are used to enforce progress properties. A simplified version, namely *Timed Safety Automata* is introduced in [HNSY94] to specify progress properties using local invariant conditions. Due to its simplicity, Timed Safety Automata has been adopted in several verification tools for timed automata *e.g.* UPPAAL [LPY97] and Kronos [Yov97]. In this presentation, we shall focus on Timed Safety Automata, and following the literature, refer them as *Timed Automata* or simply automata when it is understood from the context.

The rest of the chapter is organized as follows: In the next section, we describe the syntax and operational semantics of timed automata. The section also addresses decision problems relevant to automatic verification. In the literature, the decidability and undecidability of such problems are often considered to be the fundamental properties of a computation model. Section 10.3 presents the abstract version of the operational semantics based on regions and zones. Section 10.4 describes the data structure DBM (Difference Bound Matrices) for the efficient representation and manipulation of zones, and operations on zones, needed for symbolic verification. Section 10.5 gives a brief introduction to the verification tool UPPAAL. Finally, as an appendix, we list the pseudo-code for the presented DBM algorithms.

10.2 Timed Automata

A timed automaton is essentially a finite automaton (that is a graph containing a finite set of nodes or locations and a finite set of labeled edges) extended with real-valued variables. Such an automaton may be considered as an abstract model of a timed system. The variables model the logical clocks in the system, that are initialized with zero when the system is started, and then increase synchronously with the same rate. Clock constraints *i.e.* guards on edges are used to restrict the behavior of the automaton. A transition represented by an edge can be taken when the clocks values satisfy the guard labeled on the edge. Clocks may be reset to zero when a transition is taken.

The first example Figure 10.1(a) is an example timed automaton. The timing behavior of the automaton is controlled by two clocks x and y . The clock x is used to control the self-loop in the location **loop**. The

single transition of the loop may occur when $x = 1$. Clock y controls the execution of the entire automaton. The automaton may leave **start** at any time point when y is in the interval between 10 and 20; it can go from **loop** to **end** when y is between 40 and 50, *etc.*

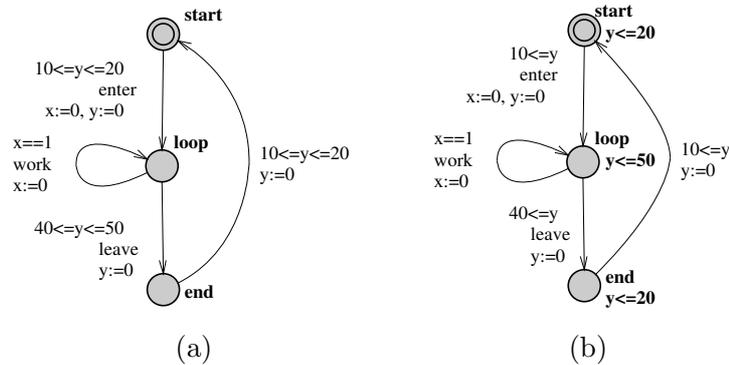


Figure 10.1: Timed Automata and Location Invariants

Timed Büchi Automata A guard on an edge of an automaton is only an enabling condition of the transition represented by the edge; but it can not force the transition to be taken. For instance, the example automaton may stay forever in any location, just idling. In the initial work by Alur and Dill [AD90], the problem is solved by introducing Büchi-acceptance conditions; a subset of the locations in the automaton are marked as accepting, and only those executions passing through an accepting location infinitely often are considered valid behaviors of the automaton. As an example, consider again the automaton in Figure 10.1(a) and assume that **end** is marked as accepting. This implies that all executions of the automaton must visit **end** infinitely many times. This imposes implicit conditions on **start** and **loop**. The location **start** must be left when the value of y is at most 20, otherwise the automaton will get stuck in **start** and never be able to enter **end**. Likewise, the automaton must leave **loop** when y is at most 50 to be able to enter **end**.

Timed Safety Automata A more intuitive notion of progress is introduced in *timed safety automata* [HNSY94]. Instead of accepting conditions, in timed safety automata, locations may be put local timing constraints called *location invariants*. An automaton may remain in a location as long as the clocks values satisfy the invariant condition of the location. For example, consider the timed safety automaton in Figure 10.1(b), which corresponds to the Büchi automaton in Figure 10.1(a)

with **end** marked as an accepting location. The invariant specifies a local condition that **start** and **end** must be left when y is at most 20 and **loop** must be left when y is at most 50. This gives a local view of the timing behavior of the automaton in each location.

In the rest of this chapter, we shall focus on timed safety automata and refer such automata as *Timed Automata* or simply automata without confusion.

10.2.1 Formal Syntax

Assume a finite set of real-valued variables \mathcal{C} ranged over by x, y etc. standing for clocks and a finite alphabet Σ ranged over by a, b etc. standing for actions. A guard is a conjunctive formula of atomic constraints of the form $x \sim n$ for $x \in \mathcal{C}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. We use $\mathcal{B}(\mathcal{C})$ to denote the set of guards, ranged over by g .

Definition 1 (Timed Automaton) A timed automaton \mathcal{A} is a tuple $\langle N, l_0, E, I \rangle$ where

- N is a finite set of locations (or nodes),
- $l_0 \in N$ is the initial location,
- $E \subseteq N \times \mathcal{B}(\mathcal{C}) \times \Sigma \times 2^{\mathcal{C}} \times N$ is the set of edges and
- $I : N \rightarrow \mathcal{B}(\mathcal{C})$ assigns invariants to locations

We shall write $l \xrightarrow{g,a,r} l'$ when $\langle l, g, a, r, l' \rangle \in E$.

As in verification tools e.g. UPPAAL [LPY97], we restrict location invariants to constraints that are downwards closed, in the form: $x \leq n$ or $x < n$ where n is a natural number.

Concurrency and Communication

To model concurrent systems, timed automata can be extended with parallel composition. In process algebras, various parallel composition operators have been proposed to model different aspects of concurrency (see e.g. CCS and CSP [Mil89, Hoa78]). These algebraic operators can be adopted in timed automata. In the UPPAAL modeling language [LPY97], the CCS parallel composition operator [Mil89] is used, which allows interleaving of actions as well as hand-shake synchronization. The precise definition of this operator is given in Section 10.5.

Essentially the parallel composition of a set of automata is the product of the automata. Building the product automaton is an entirely syntactical but computationally expensive operation. In UPPAAL, the product automaton is computed on-the-fly during verification.

10.2.2 Operational Semantics

The semantics of a timed automaton is defined as a transition system where a state or configuration consists of the current location and the current values of clocks. There are two types of transitions between states. The automaton may either delay for some time (a delay transition), or follow an enabled edge (an action transition).

To keep track of the changes of clock values, we use functions known as *clock assignments* mapping \mathcal{C} to the non-negative reals \mathbb{R}_+ . Let u, v denote such functions, and use $u \in g$ to mean that the clock values denoted by u satisfy the guard g . For $d \in \mathbb{R}_+$, let $u + d$ denote the clock assignment that maps all $x \in \mathcal{C}$ to $u(x) + d$, and for $r \subseteq \mathcal{C}$, let $[r \mapsto 0]u$ denote the clock assignment that maps all clocks in r to 0 and agree with u for the other clocks in $\mathcal{C} \setminus r$.

Definition 2 (Operational Semantics) *The semantics of a timed automaton is a transition system (also known as a timed transition system) where states are pairs $\langle l, u \rangle$, and transitions are defined by the rules:*

- $\langle l, u \rangle \xrightarrow{d} \langle l, u + d \rangle$ if $u \in I(l)$ and $(u + d) \in I(l)$ for a non-negative real $d \in \mathbb{R}_+$
- $\langle l, u \rangle \xrightarrow{a} \langle l', u' \rangle$ if $l \xrightarrow{g, a, r} l', u \in g, u' = [r \mapsto 0]u$ and $u' \in I(l')$

10.2.3 Verification Problems

The operational semantics is the basis for verification of timed automata. In the following, we formalize decision problems in timed automata based on transition systems.

Language Inclusion

A *timed action* is a pair (t, a) , where $a \in \Sigma$ is an action taken by an automaton \mathcal{A} after $t \in \mathbb{R}_+$ time units since \mathcal{A} has been started. The absolute time t is called a *time-stamp* of the action a . A *timed trace* is a (possibly infinite) sequence of timed actions $\xi = (t_1, a_1)(t_2, a_2)\dots(t_i, a_i)\dots$ where $t_i \leq t_{i+1}$ for all $i \geq 1$.

Definition 3 *A run of a timed automaton $\mathcal{A} = \langle N, l_0, E, I \rangle$ with initial state $\langle l_0, u_0 \rangle$ over a timed trace $\xi = (t_1, a_1)(t_2, a_2)(t_3, a_3)\dots$ is a sequence of transitions:*

$$\langle l_0, u_0 \rangle \xrightarrow{d_1, a_1} \langle l_1, u_1 \rangle \xrightarrow{d_2, a_2} \langle l_2, u_2 \rangle \xrightarrow{d_3, a_3} \langle l_3, u_3 \rangle \dots$$

satisfying the condition $t_i = t_{i-1} + d_i$ for all $i \geq 1$.

The timed language $L(\mathcal{A})$ is the set of all timed traces ξ for which there exists a run of \mathcal{A} over ξ .

Undecidability The negative result on timed automata as a computation model is that the language inclusion checking problem *i.e.* to check $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is undecidable [AD94, ACH94]. Unlike finite state automata, timed automata is not determinizable in general. Timed automata can not be complemented either, that is, the complement of the timed language of a timed automaton may not be described as a timed automaton.

The inclusion checking problem will be decidable if \mathcal{B} in checking $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is restricted to the deterministic class of timed automata. Research effort has been made to characterize interesting classes of determinizable timed systems *e.g.* event-clock automata [AFH99] and timed communicating sequential processes [YJ94]. Essentially, the undecidability of language inclusion problem is due to the arbitrary clock reset. If all the edges labeled with the same action symbol in a timed automaton, are also labeled with the same set of clocks to reset, the automaton will be determinizable. This covers the class of event-clock automata [AFH99].

We may abstract away from the time-stamps appearing in timed traces and define the *untimed language* $L_{untimed}(\mathcal{A})$ as the set of all traces in the form $a_1a_2a_3\dots$ for which there exists a timed trace $\xi = (t_1, a_1)(t_2, a_2)(t_3, a_3)\dots$ in the timed language of \mathcal{A} .

The inclusion checking problem for untimed languages is decidable. This is one of the classic results for timed automata [AD94].

Bisimulation

Another classic result on timed systems is the decidability of timed bisimulation [Cer92]. Timed bisimulation is introduced for timed process algebras [Yi91]. However, it can be easily extended to timed automata.

Definition 4 *A bisimulation R over the states of timed transition systems and the alphabet $\Sigma \cup \mathbb{R}_+$, is a symmetrical binary relation satisfying the following condition:*

for all $(s_1, s_2) \in R$, if $s_1 \xrightarrow{\sigma} s'_1$ for some $\sigma \in \Sigma \cup \mathbb{R}_+$ and s'_1 , then $s_2 \xrightarrow{\sigma} s'_2$ and $(s'_1, s'_2) \in R$ for some s'_2 .

Two automata are timed bisimilar iff there is a bisimulation containing the initial states of the automata.

Intuitively, two automata are timed bisimilar iff they perform the same action transition at the same time and reach bisimilar states. In [Cer92], it is shown that timed bisimulation is decidable.

We may abstract away from timing information to establish bisimulation between automata based actions performed only. This is captured by the notion of untimed bisimulation. We define $s \xrightarrow{\epsilon} s'$ if $s \xrightarrow{d} s'$ for some real number d . Untimed bisimulation is defined by replacing the

alphabet with $\Sigma \cup \{\epsilon\}$ in Definition 4. As timed bisimulation, untimed bisimulation is decidable [LW97].

Reachability Analysis

Perhaps, the most useful question to ask about a timed automaton is the reachability of a given final state or a set of final states. Such final states may be used to characterize safety properties of a system.

Definition 5 *We shall write $\langle l, u \rangle \rightarrow \langle l', u' \rangle$ if $\langle l, u \rangle \xrightarrow{\sigma} \langle l', u' \rangle$ for some $\sigma \in \Sigma \cup \mathbb{R}_+$. For an automaton with initial state $\langle l_0, u_0 \rangle$, $\langle l, u \rangle$, is reachable iff $\langle l_0, u_0 \rangle \rightarrow^* \langle l, u \rangle$. More generally, given a constraint $\phi \in \mathcal{B}(\mathcal{C})$ we say that the configuration $\langle l, \phi \rangle$ is reachable if $\langle l, u \rangle$ is reachable for some u satisfying ϕ .*

The notion of reachability is more expressive than it appears to be. We may specify invariant properties using the negation of reachability properties, and bounded liveness properties using clock constraints in combination with local properties on locations [LPY01] (see Section 10.5 for an example).

The reachability problem is decidable. In fact, one of the major advances in verification of timed systems is the symbolic technique [Dil89, YL93, HNSY94, YPD94, LPY95], developed in connection with verification tools. It adopts the idea from symbolic model checking for untimed systems, which uses boolean formulas to represent sets of states and operations on formulas to represent sets of state transitions. It is proven that the infinite state-space of timed automata can be finitely partitioned into symbolic states using clock constraints known as *zones* [Bel57, Dil89]. A detailed description on this is given in Section 10.3 and 10.4.

10.3 Symbolic Semantics and Verification

As clocks are real-valued, the transition system of a timed automaton is infinite, which is not an adequate model for automated verification.

10.3.1 Regions

The foundation for the decidability results in timed automata is based on the notion of *region equivalence* over clock assignments [AD94, ACD93].

Definition 6 (Region Equivalence) *Let k be a function, called a clock ceiling, mapping each clock $x \in \mathcal{C}$ to a natural number $k(x)$ (i.e. the ceiling of x). For a real number d , let $\{d\}$ denote the fractional part of d , and $[d]$ denote its integer part. Two clock assignments u, v are region-equivalent, denoted $u \sim_k v$, iff*

1. for all x , either $\lfloor u(x) \rfloor = \lfloor v(x) \rfloor$ or both $u(x) > k(x)$ and $v(x) > k(x)$,
2. for all x , if $u(x) \leq k(x)$ then $\{u(x)\} = 0$ iff $\{v(x)\} = 0$ and
3. for all x, y if $u(x) \leq k(x)$ and $u(y) \leq k(y)$ then $\{u(x)\} \leq \{u(y)\}$ iff $\{v(x)\} \leq \{v(y)\}$

Note that the region equivalence is indexed with a clock ceiling k . When the clock ceiling is given by the maximal clock constants of a timed automaton under consideration, we shall omit the index and write \sim instead. An equivalence class $[u]$ induced by \sim is called a *region*, where $[u]$ denotes the set of clock assignments region-equivalent to u . The basis for a finite partitioning of the state-space of a timed automaton is the following facts: First, for a fixed number of clocks each of which has a maximal constant, the number of regions is finite. Second, $u \sim v$ implies (l, u) and (l, v) are bisimilar w.r.t. the untimed bisimulation for any location l of a timed automaton. We use the equivalence classes induced by the untimed bisimulation as symbolic (or abstract) states to construct a finite-state model called the *region graph* or *region automaton* of the original timed automaton. The transition relation between symbolic states is defined as follows:

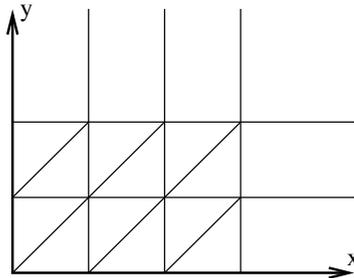


Figure 10.2: Regions for a System with Two Clocks

- $\langle l, [u] \rangle \Rightarrow \langle l, [v] \rangle$ if $\langle l, u \rangle \xrightarrow{d} \langle l, v \rangle$ for a positive real number d and
- $\langle l, [u] \rangle \Rightarrow \langle l', [v] \rangle$ if $\langle l, u \rangle \xrightarrow{a} \langle l', v \rangle$ for an action a .

Note that the transition relation \Rightarrow is finite. Thus the region graph for a timed automaton is finite. Several verification problems such as reachability analysis, untimed language inclusion, language emptiness [AD94] as well as timed bisimulation [Cer92] can be solved by techniques based on the region construction.

However, the problem with region graphs is the potential explosion in the number of regions. In fact, it is exponential in the number of

clocks as well as the maximal constants appearing in the guards of an automaton. As an example, consider Figure 10.2. The figure shows the possible regions in each location of an automaton with two clocks x and y . The largest number compared to x is 3, and the largest number compared to y is 2. In the figure, all corner points (intersections), line segments, and open areas are regions. Thus, the number of possible regions in each location of this example is 60.

A more efficient representation of the state-space for timed automata is based on the notion of *zone* and *zone-graphs* [Dil89, HNSY92, YL93, YPD94, HNSY94]. In a zone graph, instead of regions, zones are used to denote symbolic states. This in practice gives a coarser and thus more compact representation of the state-space. The basic operations and algorithms for zones to construct zone-graphs are described in Section 10.4. As an example, a timed automaton and the corresponding zone graph (or reachability graph) is shown in Figure 10.3. We note that for this automaton the zone graph has only 8 states. The region-graph for the same example has over 50 states.

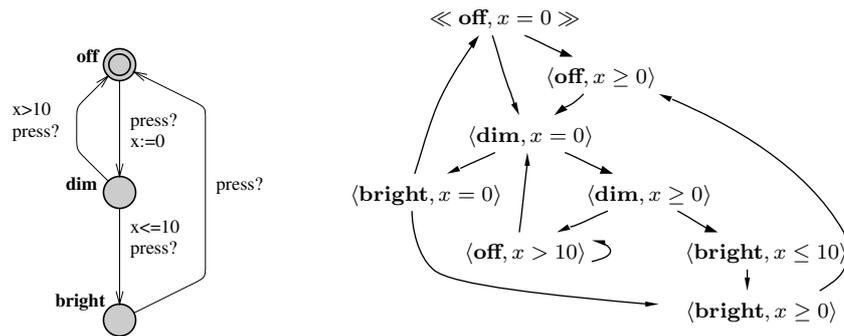


Figure 10.3: A Timed Automaton and its Zone Graph

10.3.2 Zones

Recall that a guard is a conjunctive formula of atomic constraints of the form $x \sim n$ where $x \in \mathcal{C}$, $\sim \in \{\leq, <, =, >, \geq\}$ and n is a natural number. As an extended notion, a clock constraint is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \sim n$ for $x, y \in \mathcal{C}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$.

A zone is a clock constraint. Strictly speaking, a *zone* is the solution set of a clock constraint, that is the maximal set of clock assignments satisfying the constraint. It is well-known that such sets can be efficiently represented and stored in memory as DBMs (*Difference Bound Matrices*)

[Bel57]. For a clock constraint D , let $[D]$ denote the maximal set of clock assignments satisfying D . In the following, to save notation, we shall use D to stand for $[D]$ without confusion. Then $\mathcal{B}(\mathcal{C})$ denotes the set of zones.

A symbolic state of a timed automaton is a pair $\langle l, D \rangle$ representing a set of states of the automaton, where l is a location and D is a zone. A symbolic transition describes all the possible concrete transitions from the set of states.

Definition 7 *Let D be a zone and r a set of clocks. We define $D^\dagger = \{u+d \mid u \in D, d \in \mathbb{R}_+\}$ and $r(D) = \{[r \mapsto 0]u \mid u \in D\}$. Let \rightsquigarrow denote the symbolic transition relation over symbolic states defined by the following rules:*

- $\langle l, D \rangle \rightsquigarrow \langle l, D^\dagger \wedge I(l) \rangle$
- $\langle l, D \rangle \rightsquigarrow \langle l', r(D \wedge g) \wedge I(l') \rangle$ if $l \xrightarrow{g, a, r} l'$

We shall study these operations in details in Section 10.4 where D^\dagger is written as $\text{up}(D)$ and $r(D)$ as $\text{reset}(D, r := 0)$. It will be shown that the set of zones $\mathcal{B}(\mathcal{C})$ is closed under these operations, in the sense that the result of the operations is also a zone. Another important property of zones is that a zone has a canonical form. A zone D is *closed under entailment* or just closed for short, if no constraint in D can be strengthened without reducing the solution set. The canonicity of zones means that for each zone $D \in \mathcal{B}(\mathcal{C})$, there is a unique zone $D' \in \mathcal{B}(\mathcal{C})$ such that D and D' have exactly the same solution set and D' is closed under entailment. Section 10.4 describes how to compute and represent the canonical form of a zone. It is the key structure for the efficient implementation of state-space exploration using the symbolic semantics.

The symbolic semantics corresponds closely to the operational semantics in the sense that $\langle l, D \rangle \rightsquigarrow \langle l', D' \rangle$ implies for all $u' \in D'$, $\langle l, u \rangle \rightarrow \langle l', u' \rangle$ for some $u \in D$. More generally, the symbolic semantics is a correct and full characterization of the operational semantics given in Definition 2.

Theorem 3 *Assume a timed automaton with initial state $\langle l_0, u_0 \rangle$.*

1. (soundness) $\langle l_0, \{u_0\} \rangle \rightsquigarrow^* \langle l_f, D_f \rangle$ implies $\langle l_0, u_0 \rangle \rightarrow^* \langle l_f, u_f \rangle$ for all $u_f \in D_f$.
2. (Completeness) $\langle l_0, u_0 \rangle \rightarrow^* \langle l_f, u_f \rangle$ implies $\langle l_0, \{u_0\} \rangle \rightsquigarrow^* \langle l_f, D_f \rangle$ for some D_f such that $u_f \in D_f$

The soundness means that if the initial symbolic state $\langle l_0, \{u_0\} \rangle$ may lead to a set of final states $\langle l_f, D_f \rangle$ according to \rightsquigarrow , all the final states should be reachable according to the concrete operational semantics. The completeness means that if a state is reachable according to the concrete operational semantics, it should be possible to conclude this using the symbolic transition relation.

Unfortunately, the relation \rightsquigarrow is infinite, and thus the zone-graph of a timed automaton may be infinite, which can be a problem to guarantee termination in a verification procedure. As an example, consider the automaton in Figure 10.4. The value of clock y drifts away unboundedly, inducing an infinite zone-graph.

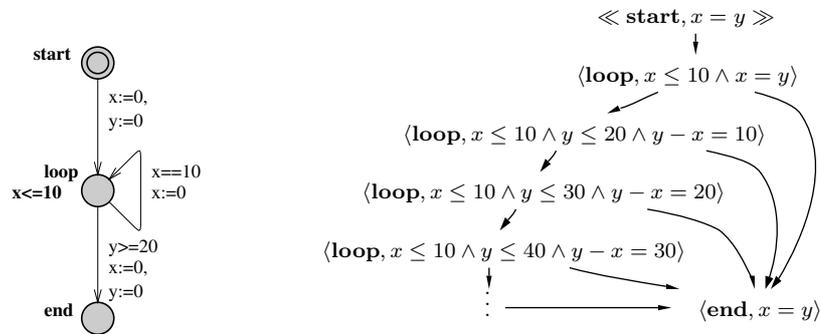


Figure 10.4: A Timed Automaton with an Infinite Zone-Graph

The solution is to transform (*i.e.* normalize) zones that may contain arbitrarily large constants to their representatives in a class of zones whose constants are bounded by fixed constants *e.g.* the maximal clock constants appearing in the automaton, using an abstraction technique similar to the widening operation [Hal93]. The intuition is that once the value of a clock is larger than the maximal constant in the automaton, it is no longer important to know the precise value of the clock, but only the fact that it is above the constant.

10.3.3 Zone-Normalization

A well-studied zone-normalization procedure is the so-called *k-normalization* operation on zones [Rok93, Pet99], which has been implemented in several verification tools for timed automata *e.g.* UPPAAL to guarantee termination.

The *k-normalization* operation is indexed by a clock ceiling k . According to [Rok93, Pet99], $\text{norm}_k(D)$ can be computed from the canonical representation of D by

1. removing all constraints of the form $x < m$, $x \leq m$, $x - y < m$ and $x - y \leq m$ where $m > k(x)$,
2. replacing all constraints of the form $x > m$, $x \geq m$, $x - y > m$ and $x - y \geq m$ where $m > k(x)$ with $x > k(x)$ and $x - y > k(x)$ respectively.

Let $[D]_k$ denote the resulted zone by the above transformation. As an example, the normalized zone-graph of the automaton in Figure 10.4 is shown in Figure 10.5 where the clock ceiling is given by the maximal clock constants appearing in the automaton.

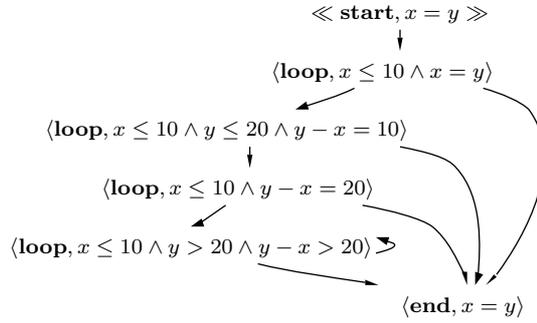


Figure 10.5: Normalized Zone Graph for the Automaton in Fig. 10.4

Note that for a fixed number of clocks with a clock ceiling k , there can be only finitely many normalized zones. The intuition is that if the constants allowed to use are bounded, one can write down only finitely many clock constraints. This gives rise to a finite characterization for \rightarrow .

Definition 8 $\langle l, D \rangle \rightsquigarrow_k \langle l', \text{norm}_k(D') \rangle$ if $\langle l, D \rangle \rightsquigarrow \langle l', D' \rangle$.

The normalized symbolic transition relation \rightsquigarrow_k is sound, complete and finite in the following sense.

Theorem 4 Assume a timed automaton with initial state $\langle l_0, u_0 \rangle$, whose maximal clock constants are bounded by a clock ceiling k . Assume that the automaton has no guards containing difference constraints in the form of $x - y \sim n$.

1. (soundness) $\langle l_0, \{u_0\} \rangle \rightsquigarrow_k^* \langle l_f, D_f \rangle$ implies $\langle l_0, u_0 \rangle \rightarrow^* \langle l_f, u_f \rangle$ for all $u_f \in D_f$ such that $u_f(x) \leq k(x)$ for all x .
2. (Completeness) $\langle l_0, u_0 \rangle \rightarrow^* \langle l_f, u_f \rangle$ with $u_f(x) \leq k(x)$ for all x , implies $\langle l_0, \{u_0\} \rangle \rightsquigarrow_k^* \langle l_f, D_f \rangle$ for some D_f such that $u_f \in D_f$

3. (*Finiteness*) The transition relation \rightsquigarrow_k is finite.

Note that in our definition for timed automata, guards are allowed to contain only constraints in the form of $x \sim n$ where \sim can be any comparison operator, and n is a non-zero natural number. The soundness will not hold for automata whose guards contain diagonal constraints specifying bounds on clock differences. We demonstrate this by an example. Consider the automaton shown in Figure 10.6. The final location of the automaton is not reachable according to the operational semantics. This is because in location S_2 , the clock zone is $(x - y > 2$ and $x > 2)$ and the guard on the outgoing edge is $x < z + 1 \wedge z < y + 1$ which is equivalent to $x - z < 1 \wedge z - y < 1 \wedge x - y < 2$. Obviously the zone at S_2 can never enable the guard, and thus the last transition will never be possible. However, because the maximal constants for clock x is 1 (and 2 for y), the zone in location S_2 : $x - y > 2 \wedge x > 2$ will be normalized to $x - y > 1 \wedge x > 1$ by the maximal constant 1 for x , which enables the guard $x - z < 1 \wedge z - y < 1$ and thus the symbolic reachability analysis based on the above normalization algorithm would incorrectly conclude that the last location is reachable.

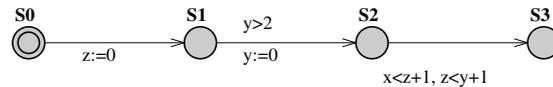


Figure 10.6: A counter example

The zones in canonical forms, generated in exploring the state-space of the counter example are given in Figure 10.7. The implicit constraints that all clocks are non-negative are not shown.

Note that at S_0 and S_1 , the normalized and un-normalized zones are identical. The problem is at S_2 where the intersection of the guard (on the only outgoing edge) with the un-normalized zone is empty and non-empty with the normalized zone.

10.3.4 Symbolic Reachability Analysis

Model-checking concerns two types of properties *liveness* and *safety*. The essential algorithm of checking liveness properties is loop detection, which is computationally expensive. The main effort on verification of timed systems has been put on safety properties that can be checked using reachability analysis by traversing the state-space of timed automata.

$$\begin{array}{ll}
S_0 : \begin{cases} x - y = 0 \\ y - z = 0 \\ z - x = 0 \end{cases} & S_0 : \begin{cases} x - y = 0 \\ y - z = 0 \\ z - x = 0 \end{cases} \\
S_1 : \begin{cases} x - y = 0 \\ z - x \leq 0 \\ z - y \leq 0 \end{cases} & S_1 : \begin{cases} x - y = 0 \\ z - x \leq 0 \\ z - y \leq 0 \end{cases} \\
S_2 : \begin{cases} y - x < -2 \\ y - z \leq 0 \\ z - x \leq 0 \\ \mathbf{0} - x < -2 \end{cases} & S_2 : \begin{cases} y - x < -1 \\ y - z \leq 0 \\ z - x \leq 0 \\ \mathbf{0} - x < -1 \end{cases} \\
& S_3 : \begin{cases} y - x < -1 \\ y - z < 0 \\ z - x < 0 \\ \mathbf{0} - x < -1 \\ \mathbf{0} - z < 0 \\ x - z < 1 \end{cases}
\end{array}$$

(a) Zones without normalization (b) Zones normalized with k -normalization

Figure 10.7: Zones for the counter example in Figure 10.6

Reachability analysis can be used to check properties on states. It consists of two basic steps, computing the state-space of an automaton under consideration, and searching for states that satisfy or contradict given properties. The first step can either be performed prior to the search, or done *on-the-fly* during the search process. Computing the state-space on-the-fly has an obvious advantage over pre-computing, in that only the part of the state-space needed to prove the property is generated. For example, consider a system with a million states. Generating the full state-space of this system will be time-consuming and may even be infeasible due to memory requirements. However, if the properties of interests can be proven by only examining the first hundred states of the system then an on-the-fly method will only need to generate those states and the result will be almost instantaneous. It should be noted though, that even on-the-fly methods will generate the entire state-space to prove certain properties, *e.g.* invariant properties.

Several model-checkers for timed systems are designed and optimized for reachability analysis based on the symbolic semantics and the zone-representation (see Section 10.4). As an example, we present the core of the verification engine of UPPAAL (see Algorithm 1).

Assume a timed automaton \mathcal{A} with a set of initial states and a set of final states (*e.g.* the bad states) characterized as $\langle l_0, D_0 \rangle$ and $\langle l_f, \phi_f \rangle$ respectively. Assume that k is the clock ceiling defined by the maximal constants appearing in \mathcal{A} and ϕ_f , and \mathcal{G} denotes the set of difference

Algorithm 1 Reachability analysis

```

PASSED =  $\emptyset$ , WAIT =  $\{\langle l_0, D_0 \rangle\}$ 
while WAIT  $\neq \emptyset$  do
  take  $\langle l, D \rangle$  from WAIT
  if  $l = l_f \wedge D \cap \phi_f \neq \emptyset$  then return “YES”
  if  $D \not\subseteq D'$  for all  $\langle l, D' \rangle \in$  PASSED then
    add  $\langle l, D \rangle$  to PASSED
    for all  $\langle l', D' \rangle$  such that  $\langle l, D \rangle \rightsquigarrow_{k, \mathcal{G}} \langle l', D' \rangle$  do
      add  $\langle l', D' \rangle$  to WAIT
    end for
  end if
end while
return “NO”

```

constraints appearing in \mathcal{A} and ϕ_f . Algorithm 1 can be used to check if the initial states may evolve to any state whose location is l_f and whose clock assignment satisfies ϕ_f . It computes the normalized zone-graph of the automaton on-the-fly, in search for symbolic states containing location l_f and zones intersecting with ϕ_f .

The algorithm computes the transitive closure of $\rightsquigarrow_{k, \mathcal{G}}$ step by step, and at each step, checks if the reached zones intersect with ϕ_f . From Theorem 4, it follows that the algorithm will return with a correct answer. It is also guaranteed to terminate because $\rightsquigarrow_{k, \mathcal{G}}$ is finite. As mentioned earlier, for a given timed automaton with a fixed set of clocks whose maximal constants are bounded by a clock ceiling k , and a fixed set of diagonal constraints contained in the guards, the number of all possible normalized zones is bounded because a normalized zone can not contain arbitrarily large or arbitrarily small constants. In fact the smallest possible zones are the refined regions. Thus the whole state-space of a timed automaton can only be partitioned into finitely many symbolic states and the worst case is the size of the region graph of the automaton, induced by the refined region equivalence. Therefore, the algorithm is working on a finite structure and it will terminate.

Algorithm 1 also highlights some of the issues in developing a model-checker for timed automata. Firstly, the representation and manipulation of states, primarily zones, is crucial to the performance of a model-checker. Note that in addition to the operations to compute the successors of a zone according to $\rightsquigarrow_{k, \mathcal{G}}$, the algorithm uses two more operations to check the emptiness of a zone as well as the inclusion between two zones. Thus, designing efficient algorithms and data-structures for zones is a major issue in developing a verification tool for timed automata, which is addressed in Section 10.4. Secondly, PASSED holds all encountered states and its size puts a limit on the size of systems we can

verify. This raises the research challenges *e.g.* state compression [Ben02], state-space reduction [BJLY98] and approximate techniques [Bal96].

10.4 DBM: Algorithms and Data Structures

The preceding section presents the key elements needed in symbolic reachability analysis. In this section, we describe how to represent zones, compute the operations and check properties on zones.

10.4.1 DBM basics

Recall that a clock constraint over \mathcal{C} is a conjunction of atomic constraints of the form $x \sim m$ and $x - y \sim n$ where $x, y \in \mathcal{C}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $m, n \in \mathbb{N}$. A zone denoted by a clock constraint D is the maximal set of clock assignments satisfying D . The most important property of zones is that they can be represented as matrices *i.e.* DBMs (Difference Bound Matrices) [Bel57, Dil89], which have a canonical representation. In the following, we describe the basic structure and properties of DBMs.

To have a unified form for clock constraints we introduce a reference clock $\mathbf{0}$ with the constant value 0. Let $\mathcal{C}_0 = \mathcal{C} \cup \{\mathbf{0}\}$. Then any zone $D \in \mathcal{B}(\mathcal{C})$ can be rewritten as a conjunction of constraints of the form $x - y \preceq n$ for $x, y \in \mathcal{C}_0$, $\preceq \in \{<, \leq\}$ and $n \in \mathbb{Z}$.

Naturally, if the rewritten zone has two constraints on the same pair of variables only the intersection of the two is significant. Thus, a zone can be represented using at most $|\mathcal{C}_0|^2$ atomic constraints of the form $x - y \preceq n$ such that each pair of clocks $x - y$ is mentioned only once. We can then store zones using $|\mathcal{C}_0| \times |\mathcal{C}_0|$ matrices where each element in the matrix corresponds to an atomic constraint. Since each element in such a matrix represents a bound on the difference between two clocks, they are named *Difference Bound Matrices* (DBMs). In the following presentation, we use D_{ij} to denote element (i, j) in the DBM representing the zone D .

To construct the DBM representation for a zone D , we start by numbering all clocks in \mathcal{C}_0 as $0, \dots, n$ and the index for $\mathbf{0}$ is 0. Let each clock be denoted by one row in the matrix. The row is used for storing lower bounds on the difference between the clock and all other clocks, and thus the corresponding column is used for upper bounds. The elements in the matrix are then computed in three steps.

- For each constraint $x_i - x_j \preceq n$ of D , let $D_{ij} = (n, \preceq)$.
- For each clock difference $x_i - x_j$ that is unbounded in D , let $D_{ij} = \infty$. Where ∞ is a special value denoting that no bound is present.

- Finally add the implicit constraints that all clocks are positive, *i.e.* $\mathbf{0} - x_i \leq 0$, and that the difference between a clock and itself is always 0, *i.e.* $x_i - x_i \leq 0$.

As an example, consider the zone $D = x - \mathbf{0} < 20 \wedge y - \mathbf{0} \leq 20 \wedge y - x \leq 10 \wedge x - y \leq -10 \wedge \mathbf{0} - z < 5$. To construct the matrix representation of D , we number the clocks in the order $\mathbf{0}, x, y, z$. The resulting matrix representation is shown below:

$$M(D) = \begin{pmatrix} (0, \leq) & (0, \leq) & (0, \leq) & (5, <) \\ (20, <) & (0, \leq) & (-10, \leq) & \infty \\ (20, \leq) & (10, \leq) & (0, \leq) & \infty \\ \infty & \infty & \infty & (0, \leq) \end{pmatrix}$$

To manipulate DBMs efficiently we need two operations on bounds: comparison and addition. We define that $(n, \preceq) < \infty$, $(n_1, \preceq_1) < (n_2, \preceq_2)$ if $n_1 < n_2$ and $(n, <) < (n, \leq)$. Further we define addition as $b_1 + \infty = \infty$, $(m, \leq) + (n, \leq) = (m + n, \leq)$ and $(m, <) + (n, \preceq) = (m + n, <)$.

Canonical DBMs

Usually there are an infinite number of zones sharing the same solution set. However, for each family of zones with the same solution set there is a unique constraint where no atomic constraint can be strengthened without losing solutions.

To compute the canonical form of a given zone we need to derive the tightest constraint on each clock difference. To solve this problem, we use a graph-interpretation of zones. A zone may be transformed to a weighted graph where the clocks in \mathcal{C}_0 are nodes and the atomic constraints are edges labeled with bounds. A constraint in the form of $x - y \preceq n$ will be converted to an edge from node y to node x labeled with (n, \preceq) , namely the distance from y to x is bounded by n .

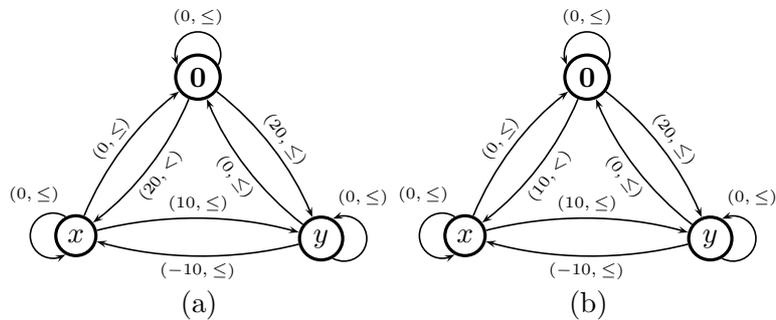


Figure 10.8: Graph interpretation of the example zone and its closed form

As an example, consider the zone $x - \mathbf{0} < 20 \wedge y - \mathbf{0} \leq 20 \wedge y - x \leq 10 \wedge x - y \leq -10$. By combining the atomic constraints $y - \mathbf{0} \leq 20$ and $x - y \leq -10$ we derive that $x - \mathbf{0} \leq 10$, *i.e.* the bound on $x - \mathbf{0}$ can be strengthened. Consider the graph interpretation of this zone, presented in Figure 10.8(a). The tighter bound on $x - \mathbf{0}$ can be derived from the graph, using the path $\mathbf{0} \rightarrow y \rightarrow x$, giving the graph in Figure 10.8(b). Thus, deriving the tightest constraint on a pair of clocks in a zone is equivalent to finding the shortest path between their nodes in the graph interpretation of the zone. The conclusion is that a canonical, *i.e.* closed, version of a zone can be computed using a shortest path algorithm. Floyd-Warshall algorithm [Flo62] (Algorithm 2) is often used to transform zones to canonical form. However, since this algorithm is quite expensive (cubic in the number of clocks), it is desirable to make all frequently used operations preserve the canonical form, *i.e.* the result of performing an operation on a canonical zone should also be canonical.

Algorithm 2 `close(D)`: Floyd's algorithm for computing shortest path

```

for  $k := 0$  to  $n$  do
  for  $i := 0$  to  $n$  do
    for  $j := 0$  to  $n$  do
       $D_{ij} := \min(D_{ij}, D_{ik} + D_{kj})$ 
    end for
  end for
end for

```

Minimal Constraint Systems

A zone may contain redundant constraints. For example, a zone contains constraints $x - y < 2, y - z < 5$ and $x - z < 7$. The constraint $x - z < 7$ is obviously redundant because it may be derived from the first two. It is desirable to remove such constraints to store only the minimal number of constraints. Consider, for instance, the zone $x - y \leq 0 \wedge y - z \leq 0 \wedge z - x \leq 0 \wedge 2 \leq x - \mathbf{0} \leq 3$. This is a zone in a minimal form containing only five constraints. The closed form of this zone contains more than 12 constraints. It is known, *e.g.* from [LLPY97], that for each zone there is a minimal constraint system with the same solution set. By computing this minimal form for all zones and storing them in memory using a sparse representation we might reduce the memory consumption for state-space exploration. This problem has been thoroughly investigated in [LLPY97, Pet99, Lar00].

The following is a summary of the published work on the minimal representation of zones. We present an algorithm that computes the minimal form of a closed DBM. Closing a DBM corresponds to computing the shortest path between all clocks. Our goal is to compute the minimal set of bounds for a given shortest path closure. For clarity, the algorithm is presented in terms of directed weighted graphs. However, the results are directly applicable to the graph interpretation of DBMs.

First we introduce some notation: we say that a cycle in a graph is a *zero cycle* if the sum of weights along the cycle is 0, and an edge $x_i \xrightarrow{n_{ij}} x_j$ is *redundant* if there is another path between x_i and x_j where the sum of weights is no larger than n_{ij} .

In graphs without zero cycles we can remove all redundant edges without affecting the shortest path closure [LLPY97]. Further, if the input graph is in shortest path form (as for closed DBMs) all redundant edges can be located by considering alternative paths of length two.

As an example, consider Figure 10.9. The figure shows the shortest path closure for a zero-cycle free graph (a) and its minimal form (b). In the graph we find that $x_1 \xrightarrow{9} x_2$ is made redundant by the path $x_1 \xrightarrow{7} x_4 \xrightarrow{2} x_2$ and can thus be removed. Further, the edge $x_3 \xrightarrow{15} x_2$ is redundant due to $x_3 \xrightarrow{5} x_1 \xrightarrow{9} x_2$. Note that we consider edges marked as redundant when searching for new redundant edges. The reason is that we let the redundant edges represent the path making them redundant, thus allowing all redundant edges to be located using only alternative paths of length two. This procedure is repeated until no more redundant edges can be found.

This gives the $O(n^3)$ procedure for removing redundant edges presented in Algorithm 3. The algorithm can be directly applied to zero-cycle free DBMs to compute the minimal number of constraints needed to represent a given zone.

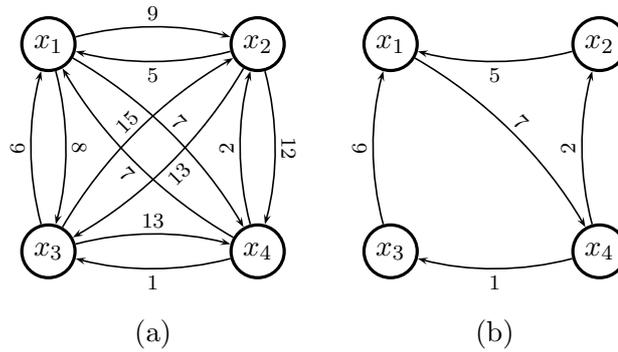


Figure 10.9: A zero cycle free graph and its minimal form

Algorithm 3 Reduction of Zero-Cycle Free Graph G with n nodes

```

for  $i := 1$  to  $n$  do
  for  $j := 1$  to  $n$  do
    for  $k := 1$  to  $n$  do
      if  $G_{ik} + G_{kj} \leq G_{ij}$  then
        Mark edge  $i \rightarrow j$  as redundant
      end if
    end for
  end for
end for
Remove all edges marked as redundant.

```

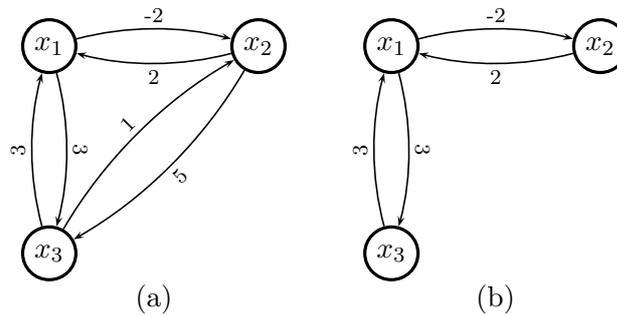


Figure 10.10: A graph with a zero-cycle and its minimal form

However, this algorithm will not work if there are zero-cycles in the graph. The reason is that the set of redundant edges in a graph with zero-cycles is not unique. As an example, consider the graph in Figure 10.10(a). Applying the above reasoning on this graph would remove $x_1 \xrightarrow{3} x_3$ based on the path $x_1 \xrightarrow{-2} x_2 \xrightarrow{5} x_3$. It would also remove the edge $x_2 \xrightarrow{5} x_3$ based on the path $x_2 \xrightarrow{2} x_1 \xrightarrow{3} x_3$. But if both these edges are removed it is no longer possible to construct paths leading into x_3 . In this example there is a dependence between the edges $x_1 \xrightarrow{3} x_3$ and $x_2 \xrightarrow{5} x_3$ such that only one of them can be considered redundant.

The solution to this problem is to partition the nodes according to zero-cycles and build a super-graph where each node is a partition. The graph from Figure 10.10(a) has two partitions, one containing x_1 and x_2 and the other containing x_3 . To compute the edges in the super-graph we pick one representative for each partition and let the edges between the partitions inherit the weights from edges between the representatives. In our example, we choose x_1 and x_3 as representatives for their equivalence classes. The edges in the graph are then $\{x_1, x_2\} \xrightarrow{3} \{x_3\}$ and $\{x_3\} \xrightarrow{3} \{x_1, x_2\}$. The super-graph is clearly zero-cycle free and can be reduced using Algorithm 3. This small graph can not be reduced further. The relation between the nodes within a partition is uniquely defined by the zero-cycle and all other edges may be removed. In our example all edges within each equivalence class are part of the zero-cycle and thus none of them can be removed. Finally the reduced super-graph is connected to the reduced partitions. In our example we end up with the graph in Figure 10.10(b). Pseudo-code for the reduction-procedure is shown in Algorithm 4.

Algorithm 4 Reduction of negative-cycle free graph G with n nodes

```

for  $i := 1$  to  $n$  do
  if  $\text{Node}_i$  is not in a partition then
     $E_{q_i} = \emptyset$ 

```

```

for  $j := i$  to  $n$  do
  if  $G_{ij} + G_{ji} = 0$  then
     $Eq_i := Eq_i \cup \{\text{Node}_i\}$ 
  end if
end for
end if
end for
Let  $G'$  be a graph without nodes.
for each  $Eq_i$  do
  Pick one representative  $\text{Node}_i \in Eq_i$ 
  Add  $\text{Node}_i$  to  $G'$ 
  Connect  $\text{Node}_i$  to all nodes in  $G'$  using weights from  $G$ .
end for
Reduce  $G'$ 
for each  $Eq_i$  do
  Add one zero cycle containing all nodes in  $Eq_i$  to  $G'$ 
end for

```

Now we have an algorithm for computing the minimal number of edges to represent a given shortest path closure that can be used to compute the minimal number of constraints needed to represent a given zone.

10.4.2 Basic Operations on DBMs

This subsection presents all the basic operations on DBMs except the ones for zone-normalization, needed in symbolic state space exploration of timed automata, both for forwards and backwards analysis. The two operations for zone-normalization are presented in the next subsection.

First note that even if a verification tool only explores the state space in one direction all operations are still useful for other purposes, *e.g.* for generating diagnostic traces. The operations are illustrated graphically in Figure 10.11.

To simplify the presentation we assume that the input zones are consistent and in canonical form. The basic operations on DBMs can be divided into two classes:

1. **Property-Checking:** This class includes operations to check the consistency of a DBM, the inclusion between zones, and whether a zone satisfies a given atomic constraint.
2. **Transformation:** This class includes operations to compute the strongest postcondition and weakest precondition of a zone according to conjunction with guards, time delay and clock reset.

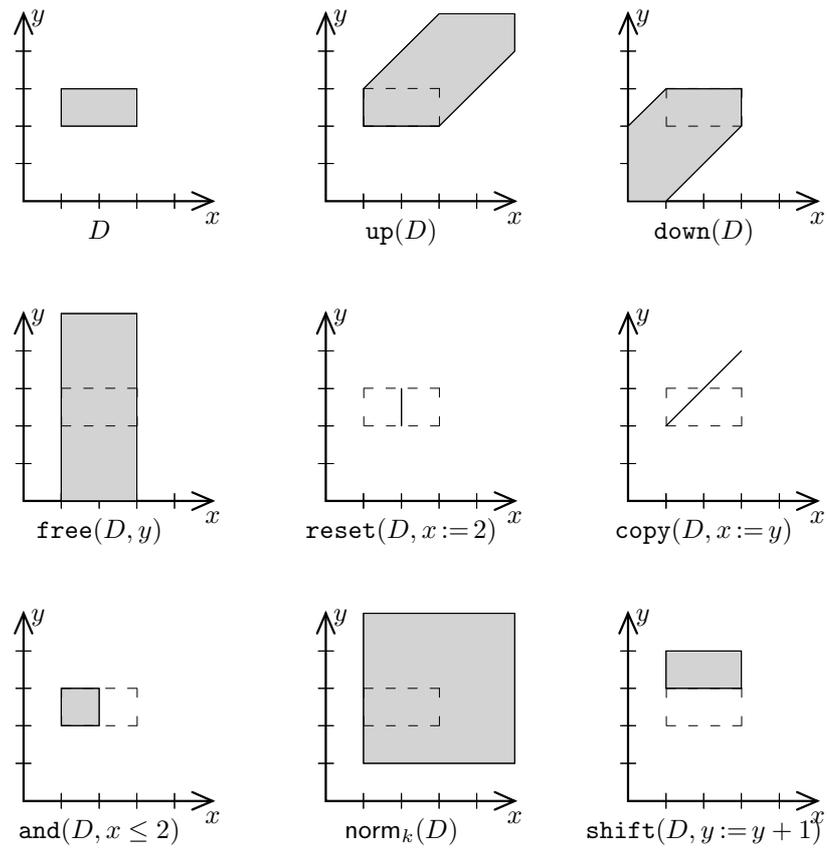


Figure 10.11: DBM operations applied to the same zone where for $norm_k(D)$, k is defined by $k(x) = 2$ and $k(y) = 1$

Property-Checking**consistent**(D)

The most basic operation on a DBM is to check if it is consistent, *i.e.* if the solution set is non-empty. In state-space exploration this operation is used to remove inconsistent states from the exploration.

For a zone to be inconsistent there must be at least one pair of clocks where the upper bound on their difference is smaller than the lower bound. For DBMs this can be checked by searching for negative cycles in the graph interpretation. However, the most efficient way to implement a consistency check is to detect when an upper bound is set to lower value than the corresponding lower bound and mark the zone as inconsistent by setting D_{00} to a negative value. For a zone in canonical form this test can be performed locally. To check if a zone is inconsistent it will then be enough to check whether D_{00} is negative.

relation(D, D')

Another key operation in state space exploration is inclusion checking for the solution sets of two zones. For DBMs in canonical form, the condition that $D_{ij} \leq D'_{ij}$ for all clocks $i, j \in \mathcal{C}_0$ is necessary and sufficient to conclude that $D \subseteq D'$. Naturally the opposite condition applies to checking if $D' \subseteq D$. This allows for the combined inclusion check described in Algorithm 5.

Algorithm 5 **relation**(D, D')

```

 $f_{D \subseteq D'} := \mathbf{tt}$ 
 $f_{D \supseteq D'} := \mathbf{tt}$ 
for  $i := 0$  to  $n$  do
  for  $j := 0$  to  $n$  do
     $f_{D \subseteq D'} := f_{D \subseteq D'} \wedge (D_{ij} \leq D'_{ij})$ 
     $f_{D \supseteq D'} := f_{D \supseteq D'} \wedge (D_{ij} \geq D'_{ij})$ 
  end for
end for
return  $\langle f_{D \subseteq D'}, f_{D \supseteq D'} \rangle$ 

```

$\text{satisfied}(D, x_i - x_j \preceq m)$

Sometimes it is desirable to non-destructively check if a zone satisfies a constraint, *i.e.* to check if the zone $D \wedge x_i - x_j \preceq m$ is consistent without altering D . From the definition of the **consistent**-operation we know that a zone is consistent if it has no negative cycles. Thus, checking if $D \wedge x_i - x_j \preceq m$ is non-empty can be done by checking if adding the guard to the zone would introduce a negative cycle. For a DBM on canonical form this test can be performed locally by checking if $(m, \preceq) + D_{ji}$ is negative.

Transformations

$\text{up}(D)$

The **up** operation computes the strongest postcondition of a zone with respect to delay, *i.e.* $\text{up}(D)$ contains the clock assignments that can be reached from D by delay. Formally, this operation is defined as $\text{up}(D) = \{u + d \mid u \in D, d \in \mathbb{R}_+\}$.

Algorithmically, **up** is computed by removing the upper bounds on all individual clocks (In a DBM all elements D_{i0} are set to ∞). This is the same as saying that any clock assignment in a given zone may delay an arbitrary amount of time. The property that all clocks proceed at the same speed is ensured by the fact that constraints on the differences between clocks are not altered by the operation.

This operation preserves the canonical form, *i.e.* applying **up** to a canonical DBM will result in a new canonical DBM. The **up** operation is also presented in Algorithm 6.

Algorithm 6 $\text{up}(D)$

```

for  $i := 1$  to  $n$  do
   $D_{i0} := \infty$ 
end for

```

$\text{down}(D)$

This operation computes the weakest precondition of D with respect to delay. Formally $\text{down}(D) = \{u \mid u + d \in D, d \in \mathbb{R}_+\}$, *i.e.* the set of clock assignments that can reach D by some delay d . Algorithmically, down is computed by setting the lower bound on all individual clocks to $(0, \leq)$. However due to constraints on clock differences this algorithm may produce non-canonical DBMs. As an example, consider the zone in Figure 10.12(a). When down is applied to this zone (Figure 10.12(b)), the lower bound on x is 1 and not 0, due to constraints on clock differences. Thus, to obtain an algorithm that produce canonical DBMs the difference constraints have to be taken into account when computing the new lower bounds.

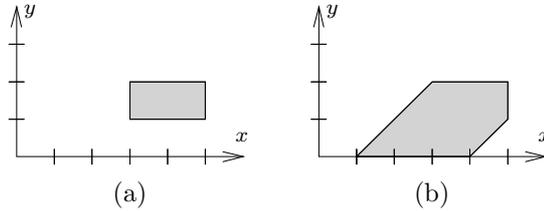


Figure 10.12: Applying down to a zone.

To compute the lower bound for a clock x , start by assuming that all other clocks y_i have the value 0. Then examine all difference constraints $y_i - x$ and compute a new lower bound for x under this assumption. The new bound on $0 - x$ will be the minimum bound on $y_i - x$ found in the DBM. Pseudo-code for down is presented in Algorithm 7.

Algorithm 7 $\text{down}(D)$

```

for  $i := 1$  to  $n$  do
   $D_{0i} = (0, \leq)$ 
  for  $j := 1$  to  $n$  do
    if  $D_{ji} < D_{0i}$  then
       $D_{0i} = D_{ji}$ 
    end if
  end for
end for

```

$\text{and}(D, x_i - y_j \preceq b)$

The most used operation in state-space exploration is conjunction, *i.e.* adding a constraint to a zone. The basic step of the **and** operation is to check if $(b, \preceq) < D_{ij}$ and in this case set the bound D_{ij} to (b, \preceq) . If the bound has been altered, *i.e.* if adding the guard affected the solution set, the DBM has to be put back on canonical form. One way to do this would be to use the generic shortest path algorithm, however for this particular case it is possible to derive a specialization of the algorithm allowing re-canonicalization in $O(n^2)$ instead of $O(n^3)$.

The specialized algorithm takes the advantage that D_{ij} is the only bound that has been changed. Since the Floyd-Warshall algorithm is insensitive to how the nodes in the graph are ordered, we may decide to treat x_i and x_j last. The outer loop of Algorithm 2 will then only affect the DBM twice, for $k = i$ and $k = j$. This allows the canonicalisation algorithm to be reduced to checking, for all pairs of clocks in the DBM, if the path via either x_i or x_j is shorter than the direct connection. The pseudo code for this is presented in Algorithm 8.

Algorithm 8 $\text{and}(D, g)$

```

if  $D_{yx} + (m, \preceq) < 0$  then
   $D_{00} = (-1, \preceq)$ 
else if  $(m, \preceq) < D_{xy}$  then
   $D_{xy} = (m, \preceq)$ 
  for  $i := 0$  to  $n$  do
    for  $j := 0$  to  $n$  do
      if  $D_{ix} + D_{xj} < D_{ij}$  then
         $D_{ij} = D_{ix} + D_{xj}$ 
      end if
      if  $D_{iy} + D_{yj} < D_{ij}$  then
         $D_{ij} = D_{iy} + D_{yj}$ 
      end if
    end for
  end for
end if

```

free(D, x)

The **free** operation removes all constraints on a given clock, *i.e.* the clock may take any positive value. Formally this is expressed as $\text{free}(D, x) = \{u[x = d] \mid u \in D, d \in \mathbb{R}_+\}$. In state-space exploration this operation is used in combination with conjunction, to implement reset operations on clocks. It can be used in both forwards and backwards exploration, but since forwards exploration allows other more efficient implementations of reset, **free** is only used when exploring the state-space backwards.

A simple algorithm for this operation is to remove all bounds on x in the DBM and set $D_{0x} = (0, \leq)$. However, the result may not be on canonical form. To obtain an algorithm preserving the canonical form, we change how new difference constraints regarding x are derived. We note that the constraint $\mathbf{0} - x \leq 0$ can be combined with constraints of the form $y - \mathbf{0} \preceq m$ to compute new bounds for $y - x$. For instance, if $y - \mathbf{0} \leq 5$ we can derive that $y - x \leq 5$. To obtain a DBM on canonical form we derive bounds for D_{yx} based on D_{y0} instead of setting $D_{yx} = \infty$. In Algorithm 9 this is presented as pseudo code.

Algorithm 9 **free**(D, x)

```

for  $i := 0$  to  $n$  do
  if  $i \neq x$  then
     $D_{xi} = \infty$ 
     $D_{ix} = D_{i0}$ 
  end if
end for

```

reset($D, x := m$)

In forwards exploration this operation is used to set clocks to specific values, *i.e.* $\text{reset}(D, x := m) = \{u[x = m] \mid u \in D\}$. Without the requirement that output should be on canonical form, **reset** can be implemented by setting $D_{x0} = (m, \leq)$, $D_{0x} = (-m, \leq)$ and remove all other bounds on x . However, if we instead of removing the difference constraints compute new values using constraints on the other clocks, like in the implementation of **free**, we obtain an implementation that preserve the canonical form. Such an implementation is presented in Algorithm 10.

Algorithm 10 $\text{reset}(D, x := m)$

```

for  $i := 0$  to  $n$  do
   $D_{xi} := (m, \leq) + D_{0i}$ 
   $D_{ix} := D_{i0} + (-m, \leq)$ 
end for

```

copy($D, x := y$)

This is another operation used in forwards state-space exploration. It copies the value of one clock to another. Formally, we define $\text{copy}(D, x := y)$ as $\{u[x = u(y)] \mid u \in D\}$. As **reset**, **copy** can be implemented by assigning $D_{xy} = (0, \leq)$, $D_{yx} = (0, \leq)$, removing all other bounds on x and re-canonicalize the zone. However, a more efficient implementation is obtained by assigning $D_{xy} = (0, \leq)$, $D_{yx} = (0, \leq)$ and then copy the rest of the bounds on x from y . For pseudo code, see Algorithm 11

Algorithm 11 $\text{copy}(D, x := y)$

```

for  $i := 0$  to  $n$  do
  if  $i \neq x$  then
     $D_{xi} := D_{yi}$ 
     $D_{ix} := D_{iy}$ 
  end if
end for
 $D_{xy} := (0, \leq)$ 
 $D_{yx} := (0, \leq)$ 

```

shift($D, x := x + m$)

The last reset operation is shifting a clock, *i.e.* adding or subtracting a clock with an integer value, *i.e.* $\mathbf{shift}(D, x := x + m) = \{u[x = u(x) + m] \mid u \in D\}$. The definition gives a hint on how to implement the operation. We can view the shift operation as a substitution of $x - m$ for x in the zone. With this reasoning m is added to the upper and lower bounds of x . However, since lower bounds on x are represented by constraints on $y - x$, m is subtracted from all those bounds. This operation is presented in pseudo-code in Algorithm 12

Algorithm 12 $\mathbf{shift}(D, x := x + m)$

```

for  $i := 0$  to  $n$  do
  if  $i \neq x$  then
     $D_{xi} := D_{xi} + (m, \leq)$ 
     $D_{ix} := D_{ix} + (-m, \leq)$ 
  end if
end for
 $D_{x0} := \max(D_{x0}, (0, \leq))$ 
 $D_{0x} := \min(D_{0x}, (0, \leq))$ 

```

10.4.3 Zone-Normalization

The operations for zone-normalization are to transform zones which may contain arbitrarily large constants to zones containing only bounded constants in order to obtain a finite zone-graph.

For a timed automaton and a safety property to be checked, that contain no diagonal constraints, the k -normalization $\text{norm}_k(D)$ is needed, and it can be computed based on the canonical form of D (see Section 10.3). It is to remove all upper bounds higher than the maximal constants and lower all lower bounds higher than the maximal constants down to the maximal constants. The result of $\text{norm}_k(D)$ is illustrated graphically in Figure 10.11.

In the canonical DBM representation of a zone, the operation consists of two steps: first, remove all bounds $x - y \preceq m$ such that $(m, \preceq) > (k(x), \preceq)$ and second, set all bounds $x - y \preceq m$ such that $(m, \preceq) < (-k(y), <)$ to $(-k(y), <)$. Pseudo-code for k -normalization is given in Algorithm 13 where k_i denotes $k(x_i)$. The k -normalization will not preserve the canonical form of a DBM, and the best way to put the result back on canonical form is to use Algorithm 2.

Algorithm 13 $\text{norm}_k(D)$

```

for  $i := 0$  to  $n$  do
  for  $j := 0$  to  $n$  do
    if  $D_{ij} \neq \infty$  and  $D_{ij} > (k_i, \preceq)$  then
       $D_{ij} = \infty$ 
    else if  $D_{ij} \neq \infty$  and  $D_{ij} < (-k_j, <)$  then
       $D_{ij} = (-k_j, <)$ 
    end if
  end for
end for
close( $D$ )

```

10.5 UPPAAL

In the last decade, there have been a number of tools developed based on timed automata to model and verify real time systems, notably Kronos [Yov97] and UPPAAL [LPY97]. As an example, we give a brief introduction to the UPPAAL tool (www.uppaal.com).

UPPAAL is a tool box for modeling, simulation and verification of timed automata, based on the algorithms and data-structures presented in previous sections. The tool was released for the first time in 1995, and since then it has been developed and maintained in collaboration between Uppsala University and Aalborg University.

10.5.1 Modeling with UPPAAL

The core of the UPPAAL modeling language is networks of timed automata. In addition, the language has been further extended with features to ease the modeling task and to guide the verifier in state space exploration. The most important of these are shared integer variables, urgent channels and committed locations.

Networks of Timed Automata

A *network of timed automata* is the parallel composition $A_1 | \dots | A_n$ of a set of timed automata A_1, \dots, A_n , called processes, combined into a single system by the CCS parallel composition operator with all external actions hidden. Synchronous communication between the processes is by hand-shake synchronization using input and output actions; asynchronous communication is by shared variables as described later. To model hand-shake synchronization, the action alphabet Σ is assumed to consist of symbols for input actions denoted $a?$, output actions denoted $a!$, and internal actions represented by the distinct symbol τ .

An example system composed of two timed automata is shown in Figure 10.13. The network models a time-dependent light-switch (to the left) and its user (to the right). The user and the switch communicate using the label *press*. The user can press the switch (*press!*) and the switch waits to be pressed (*press?*). The product automaton, *i.e.* the automaton describing the combined system is shown in Figure 10.14.

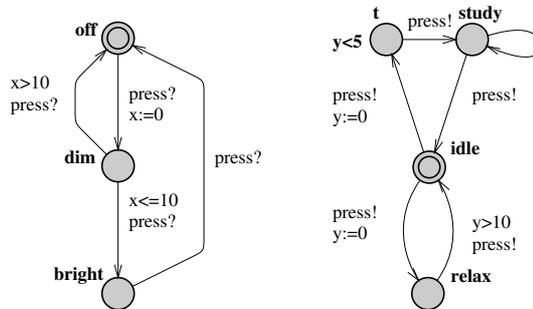


Figure 10.13: Network of Timed Automata

The semantics of networks is given as for single timed automata in terms of transition systems. A state of a network is a pair $\langle l, u \rangle$ where l denotes a vector of current locations of the network, one for each process, and u is as usual a clock assignment remembering the current values of the clocks in the system. A network may perform two types of transitions, delay transitions and discrete transitions. The rule for delay transitions is similar to the case of single timed automata

Shared Integer Variables

Clocks may be considered as typed variables with type *clock*. In UPPAAL, one may also use integer variables and arrays of integers, each with a bounded domain and an initial value. Predicates over the integer variables can be used as guards on the edges of an automaton process and the integer variables may be updated using resets on the edges. In the current version of UPPAAL, the syntax related to integer variables resembles the standard C syntax. Both integer guards and integer resets are standard C expressions with the restriction that guards can not have side-effects.

The semantics of networks can be defined accordingly. The clock assignment u in the state configuration $\langle l, u \rangle$ can be extended to store the values of integer variables in addition to clocks. Since delay does not affect the integer variables, the delay transitions are the same as for networks without integer variables. The action transitions are extended in the natural way, *i.e.* for an action transition to be enabled the extended clock assignment must also satisfy all integer guards on the corresponding edges and when a transition is taken the assignment is updated according to the integer and clock resets.

There is a problem with variable updating in a synchronizing transition where one of the processes participating in the transition updates a variable used by the other. In UPPAAL, for a synchronization transition, the resets on the edge with an output-label is performed before the resets on the edge with an input-label. This destroys the symmetry of input and output actions. But it gives a natural and clear semantics for variable updating. The transition rule for synchronization is modified accordingly:

- $\langle l, u \rangle \xrightarrow{\tau} \langle l[l'_i/l_i][l'_j/l_j], u' \rangle$ if there exist $i \neq j$ such that
 1. $l_i \xrightarrow{g_i, a^?, r_i} l'_i, l_j \xrightarrow{g_j, a^!, r_j} l'_j$ and $u \in g_i \wedge g_j$, and
 2. $u' = [r_i \mapsto 0]([r_j \mapsto 0]u)$ and $u' \in I(l[l'_i/l_i][l'_j/l_j])$

Urgent Channels

To model urgent synchronizing transitions, which should be taken as soon as they are enabled, UPPAAL supports a notion of urgent channels. An urgent channel works much like an ordinary channel, but with the exception that if a synchronization on an urgent channel is possible the system may not delay. Interleaving with other enabled action transitions, however, is still allowed. In order to keep clock constraints representable using convex zones, clock guards are not allowed on edges synchronizing on urgent channels.

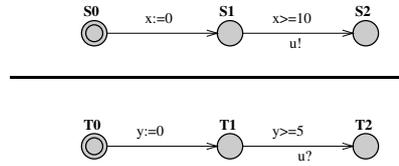


Figure 10.15: An example of a network with non convex timing regions.

To illustrate why this restriction is necessary, consider the network shown in Figure 10.15. Both processes may independently go from their first state to their second state. In the second state, the first process must delay for at least 10 time units before it is allowed to synchronize on the urgent channel u . In the second state, the other process must delay for at least 5 time units before it is allowed to synchronize on the urgent channel u . As soon as both processes have spent the minimal time periods required in their second state, they should synchronize and move to their third state. The problem is in $[\mathbf{S2}, \mathbf{T2}]$ where the zone may be for example $(x \geq 10 \wedge y = 5) \vee (y \geq 5 \wedge x = 10)$ which is a non-convex zone.

For this example, the problem can be solved by splitting the non-convex zone into two convex ones. But in general, the splitting is a computationally expensive operation. In UPPAAL, we decided to avoid such operations for the sake of efficiency. So only integer guards are allowed on edges involving synchronizations on urgent channels.

Committed Locations

To model atomic sequences of actions, *e.g.* atomic broadcast or multicast, UPPAAL supports a notion of *committed locations*. A committed location is a location where no delay is allowed. In a network, if any process is in a committed location then only action transitions starting from such a committed location are allowed. Thus, processes in committed locations may be interleaved only with processes in a committed location.

Syntactically, each process A_i in a network may have a subset $N_i^C \subseteq N_i$ of locations marked as committed locations. Let $C(l)$ denote the set of locations in l , that are committed. For the same reason as in the case of urgent channels, as a syntactical restriction, no clock constraints but predicates over integer variables are allowed to appear in a guard on an outgoing edge from a committed location.

The transition rules are given in the following, where \rightarrow_c denotes the transition relation for a network with committed locations and \rightarrow

denotes the transition relation for the same network without considering the committed locations.

- $\langle l, u \rangle \xrightarrow{d}_c \langle l, u + d \rangle$ if $\langle l, u \rangle \xrightarrow{d} \langle l, u + d \rangle$ and $C(l) = \emptyset$
- $\langle l, u \rangle \xrightarrow{\tau}_c \langle l[l'_i/l_i], u' \rangle$ if
 1. $\langle l, u \rangle \xrightarrow{\tau} \langle l[l'_i/l_i], u' \rangle$, and
 2. Either $l_i \in C(l)$ or $C(l) = \emptyset$
- $\langle l, u \rangle \xrightarrow{\tau}_c \langle l[l'_i/l_i][l'_j/l_j], u' \rangle$ if
 1. $\langle l, u \rangle \xrightarrow{\tau} \langle l[l'_i/l_i][l'_j/l_j], u' \rangle$, and
 2. Either $l_i \in C(l)$, $l_j \in C(l)$ or $C(l) = \emptyset$

10.5.2 Verifying with UPPAAL

The model checking engine of UPPAAL is designed to check a subset of TCTL formula [ACD90] for networks of timed automata. The formulas should be one of the following forms:

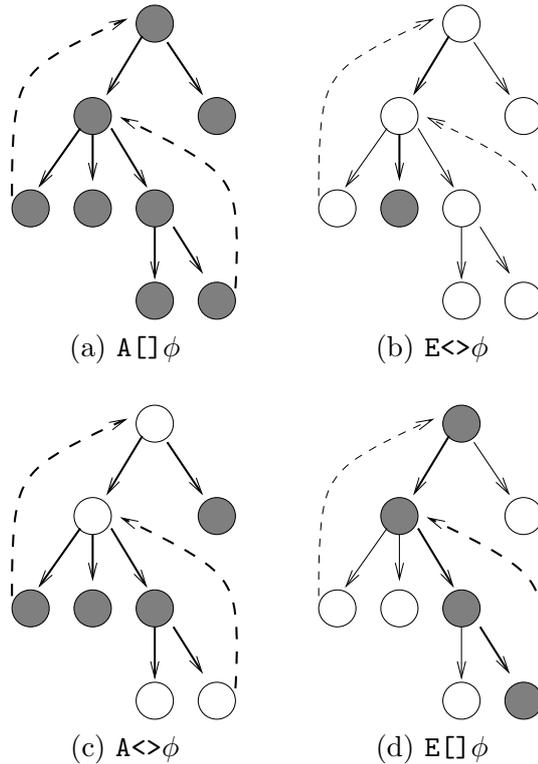


Figure 10.16: (T)CTL-formulae

- $A[] \phi$ — Invariantly ϕ .
- $E\langle\rangle \phi$ — Possibly ϕ .
- $A\langle\rangle \phi$ — Always Eventually ϕ .
- $E[] \phi$ — Potentially Always ϕ .
- $\phi \dashrightarrow \psi$ — ϕ always leads to ψ . This is the shorthand for $A[] (\phi \Rightarrow A\langle\rangle\psi)$.

where ϕ, ψ are local properties that can be checked locally on a state, *i.e.* boolean expressions over predicates on locations and integer variables, and clock constraints in $\mathcal{B}(\mathcal{C})$.

The transition system of a network may be unfolded into an infinite tree containing states and transitions. The semantics of the formulas are defined over such a tree. The letters A and E are used to quantify over paths. A is used to denote that the given property should hold for all paths of the tree while E denotes that there should be at least one path of the tree where the property holds. The symbols $[]$ and $\langle\rangle$ are used to quantify over states within a path. $[]$ denotes that all states on the path should satisfy the property, while $\langle\rangle$ denotes that at least one state in the execution satisfies the property. In Figure 10.16 the four basic property types are illustrated using execution trees, where the dashed arrows are used to denote repetitions in the trees. The states satisfying ϕ are denoted by filled nodes and edges corresponding to the paths are highlighted using bold arrows.

The two types of properties most commonly used in verification of timed systems are $E\langle\rangle\phi$ and $A[]\psi$. They are dual in the sense that $E\langle\rangle\phi$ is satisfied if and only if $A[]\neg\phi$ is not satisfied. This type of properties are often classified as safety properties, *i.e.* meaning that the system is safe in the sense that a specified hazard can not occur. It is also possible to transform so called bounded liveness properties, *i.e.* properties stating that some desired state will be reached within a given time, into safety properties using observer automata [ABL98] or by annotating the model [LPY98]. For example, to check if an automaton will surely reach a location l within 10 time units, we use one clock x (set to 0 initially) and introduce a boolean variable l_b (set to false initially). For each incoming edge to l in the automaton, set l_b to true. Then if the automaton satisfies the invariant property " $x \leq 10 \vee l_b$ ", it will surely reach l within 10 time units provided that the automaton contains no zeno loops which stop time to progress.

The other three types of properties are commonly classified as unbounded liveness properties, *i.e.* they are used to express and check for global progress. These properties are not commonly used in UPPAAL

case-studies. It seems to be the case that bounded liveness properties are more important for timed systems.

10.5.3 The UPPAAL Architecture

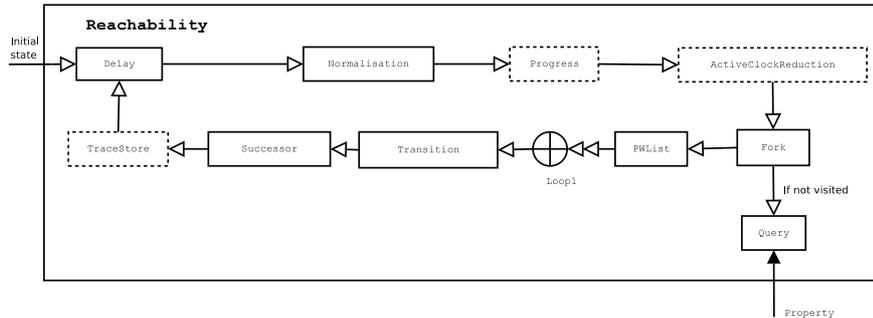


Figure 10.17: Schematic view of the reachability pipeline in UPPAAL.

To provide a system that is both efficient, easy to use and portable, UPPAAL is split into two components, a graphical user interface written in Java and a verification engine written in C++. The engine and the GUI communicate using a protocol, allowing the verification to be performed either on the local workstation or on a powerful server in a network.

To implement the reachability analysis algorithm 1, the UPPAAL verification engine is organized as a pipeline that incarnates the natural data flow in the algorithm. A sketch of this pipeline is shown in Figure 10.17. This architecture simplifies both activating and deactivating optimizations at runtime by inserting and removing stages dynamically, and introducing new optimizations and features in the tool by implementing new or changing existing stages.

In addition to the zone-manipulation algorithms described in Section 10.4 and the pipeline architecture, in UPPAAL a number of optimizations have been implemented:

- Minimal constraint systems [LLPY97] and CDDs [LPWY99, BLP⁺99], to reduce memory consumption by removing redundant information in zones before storing them.
- Selective storing of states in PASSED [LLPY97], where static analysis is used to detect states that can be omitted safely from PASSED without losing termination.
- Compression [Ben01] and sharing [BDLY03, DBLY03] of state data, to reduce the memory consumption of PASSED and WAIT.

- Active clock reduction [DY96], that use live-range analysis to determine when the value of a clock is irrelevant. This does not only reduce the size of individual states but also the perceived state-space.
- Supertrace [Hol91] and Hash Compaction [WL93, SD95] where already visited states are stored only as hash signatures, and Convex-hull approximation [Bal96] where convex hulls are used to approximate unions of zones, for reducing memory consumption at a risk of inconclusive results.

Bibliography

- [ABL98] Luca Aceto, Augusto Bergueno, and Kim G. Larsen. Model checking via reachability testing for timed automata. In *Proceedings, Fourth Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1384 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.
- [ACD90] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, 1990.
- [ACD93] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Journal of Information and Computation*, 104(1):2–34, 1993.
- [ACH94] Rajeev Alur, Costas Courcoubetis, and Thomas A. Henzinger. The observational power of clocks. In *International Conference on Concurrency Theory*, pages 162–177, 1994.
- [AD90] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *Proceedings, Seventeenth International Colloquium on Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer-Verlag, 1990.
- [AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Journal of Theoretical Computer Science*, 126(2):183–235, 1994.
- [AFH99] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211(1–2):253–273, 1999.

- [AH94] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.
- [Bal96] Felice Balarin. Approximate reachability analysis of timed automata. In *Proceedings, 17th IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, 1996.
- [BD91] Bernard Berthomieu and Michel Diaz. Modeling and verification of timed dependent systems using timed petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.
- [BDLY03] Gerd Behrmann, Alexandre David, Kim G. Larsen, and Wang Yi. Unification & sharing in timed automata verification. In *Proceedings, Tenth International SPIN Workshop*, volume 2648 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.
- [Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [Ben01] Johan Bengtsson. Reducing memory usage in symbolic state-space exploration for timed systems. Technical Report 2001-009, Department of Information Technology, Uppsala University, 2001.
- [Ben02] Johan Bengtsson. Clocks, dbms and states in timed systems. Ph.D. Thesis, ACTA Universitatis Upsaliensis 39, Uppsala University, 2002.
- [BJLY98] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In *Proceedings, Ninth International Conference on Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500. Springer-Verlag, 1998.
- [BLP+99] Gerd Behrmann, Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Efficient timed reachability analysis using clock difference diagrams. In *Proceedings, Eleventh International Conference on Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 341–353. Springer-Verlag, 1999.
- [Cer92] Karlis Cerans. Decidability of bisimulation equivalences for parallel timer processes. In *Computer Aided Verification*, volume 663 of *LNCS*. Springer, 1992.

- [Cha99] Zhou Chaochen. Duration calculus, a logical approach to real-time systems. *Lecture Notes in Computer Science*, 1548:1–7, 1999.
- [DBLY03] Alexandre David, Gerd Behrmann, Kim G. Larsen, and Wang Yi. A tool architecture for the next generation of UPPAAL. Technical Report 2003-011, Department of Information Technology, Uppsala University, 2003.
- [Dil89] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings, Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer-Verlag, 1989.
- [DY96] Conrado Daws and Sergio Yovine. Reducing the number of clock variables of timed automata. In *Proceedings, 17th IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, 1996.
- [Flo62] Robert W. Floyd. AcM algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, June 1962.
- [Hal93] N. Halbwachs. Delay analysis in synchronous programs. In *Fifth Conference on Computer-Aided Verification*, Elounda (Greece), July 1993. LNCS 697, Springer Verlag.
- [HNSY92] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 394–406, 1992.
- [HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Journal of Information and Computation*, 111(2):193–244, 1994.
- [Hoa78] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–676, August 1978.
- [Hol91] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.
- [Lar00] Fredrik Larsson. Efficient implementation of model-checkers for networks of timed automata. Licentiate Thesis 2000-003, Department of Information Technology, Uppsala University, 2000.

- [LLPY97] Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: Compact data structure and state space reduction. In *Proceedings, 18th IEEE Real-Time Systems Symposium*, pages 14–24. IEEE Computer Society Press, 1997.
- [LPWY99] Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nordic Journal of Computing*, 1999.
- [LPY95] Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of the 16th IEEE Real-Time Systems Symposium*, pages 76–87. IEEE Computer Society Press, December 1995.
- [LPY97] Kim G. Larsen, Paul Petterson, and Wang Yi. UPPAAL in a nutshell. *Journal on Software Tools for Technology Transfer*, 1997.
- [LPY98] Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal Design and Analysis of a Gear-Box Controller. In *Proceedings, Fourth Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 281–297. Springer-Verlag, 1998.
- [LPY01] Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal Design and Analysis of a Gearbox Controller. *Springer International Journal of Software Tools for Technology Transfer (STTT)*, 3(3):353–368, 2001.
- [LW97] Kim Guldstrand Larsen and Yi Wang. Time-abstracted bisimulation: Implicit specifications and decidability. *Information and Computation*, 134(2):75–101, 1997.
- [Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.
- [NS94] Xavier Nicollin and Joseph Sifakis. The algebra of timed processes, ATP: Theory and application. *Journal of Information and Computation*, 114(1):131–178, 1994.
- [Pet99] Paul Pettersson. *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice*. PhD thesis, Uppsala University, 1999.
- [Rok93] Tomas Gerhard Rokicki. *Representing and Modeling Digital Circuits*. PhD thesis, Stanford University, 1993.

- [RR88] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58(1-3):249–261, 1988.
- [SD95] Ulrich Stern and David L. Dill. Improved probabilistic verification by hash compaction. In *Correct Hardware Design and Verification Methods: IFIP WG10.5 Advanced Research Working Conference Proceedings*, 1995.
- [WL93] Pierre Wolper and Dennis Leroy. Reliable hashing without collision detection. In *Proceedings, Fifth International Conference on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 59–70. Springer-Verlag, 1993.
- [Yi91] Wang Yi. CCS + time = an interleaving model for real time systems. In *Proceedings, Eighteenth International Colloquium on Automata, Languages and Programming*, volume 510 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.
- [YJ94] Wang Yi and B. Jonsson. Decidability of timed language-inclusion for networks of real-time communicating sequential processes. In *Proc. 14th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 880 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.
- [YL93] Mihalis Yannakakis and David Lee. An efficient algorithm for minimizing real-time transition systems. In *Proceedings, Fifth International Conference on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 210–224. Springer-Verlag, 1993.
- [Yov97] Sergio Yovine. Kronos: a verification tool for real-time systems. *Journal on Software Tools for Technology Transfer*, 1, October 1997.
- [YPD94] Wang Yi, Paul Petterson, and Mats Daniels. Automatic verification of real-time communicating systems by constraint-solving. In *Proceedings, Seventh International Conference on Formal Description Techniques*, pages 223–238, 1994.

Chapter 11

Decidable and Undecidable Problems in Schedulability Analysis Using Timed Automata

By **Pavel Krčál** and **Wang Yi**

Division of Computer Systems

Department of Information Technology

Uppsala University

Email: {pavelk,yi}@it.uu.se

We study schedulability problems of timed systems with non-uniformly recurring computation tasks. Assume a set of real time tasks whose best and worst execution times, and deadlines are known. We use timed automata to describe the arrival patterns (and release times) of tasks. From the literature, it is known that the schedulability problem for a large class of such systems is decidable and can be checked efficiently.

In this paper, we provide a summary on what is decidable and what is undecidable in schedulability analysis using timed automata. Our main technical contribution is that the schedulability problem will be undecidable if these three conditions hold: (1) the execution times of tasks are intervals, (2) a task can announce its completion time, and (3) a task can preempt another task. We show that if one of the above three conditions is dropped, the problem will be decidable. Thus our result can be used as an indication in identifying classes of timed systems that can be analysed efficiently.

11.1 Introduction

Timed automata [AD94] have been developed as a basic semantic model for real time systems. Recently they have been applied to solve scheduling problems, such as job-shop scheduling [AM01, AM02, Abd02, Feh02, HLP01] and real time scheduling [MV94, FPY02, AFM⁺02, FMPY03, WH04]. The basic idea behind these works is to model real time tasks (or jobs) and scheduling strategies of a system as variants of timed automata and then check the reachability of pre-specified states. As the reachability problem of such automata is decidable, the scheduling problems can be solved automatically and in many cases efficiently (e.g. for fixed priority scheduling [FMPY03]) using a model-checker such as KRONOS [Yov97], UPPAAL [LPY97] or HYTECH [HHWT97]. For preemptive scheduling, stop-watch automata have been used to model preemption [CL00, Cor94, AM02]. But since the reachability problem for this class of automata is undecidable [ACH⁺95] there is no guarantee for termination in the general case.

We adopt the model presented in [EWY98]. The essential idea behind the model is to use a timed automaton (*control automaton*) to describe the release (or arrival) patterns of the tasks. Tasks are released when the control automaton makes a discrete transition. Each task has specified its computation (execution) time and its deadline. Released tasks are stored in a task queue and executed on a processor according to a scheduling strategy. There is a straightforward translation of such a system into a timed automaton for non-preemptive scheduling strategies. For the preemptive case, the schedulability problem was suspected to be undecidable due to the nature of preemption that may need the power of stop-watch to model.

In the original work, tasks in the queue cannot send any information back to the control automaton. The only communication between the control automaton and the task queue is the release of tasks. Once a task has been released, the control automaton has no possibility to find out any information about its execution. In particular, the control automaton does not know whether a task has finished or not. The behaviour of the control automaton is independent from the state of the queue. We say that this is a system with *one-way communication*. In this paper, we study systems where tasks can tell the control automaton that their execution has been just finished (systems with *two-way communication*). As an illustration, consider a company, where the boss assigns jobs to employees from time to time. In the model with one-way communication, the employees do not tell the boss when they finish their jobs whereas in the model with two-way communication they do.

The execution time of a task can vary within an interval – only the best and the worst execution time of each task is known. This is a natu-

ral assumption for modeling, because in many cases we cannot establish the exact computation time of a task (it depends on many circumstances from which the model abstracts), but we can establish some bounds for it. In the schedulability analysis of the systems with two-way communication we have to consider each possible finishing time, because it can influence the future behaviour of the control automaton. For instance, the boss can become too optimistic when several employees finish their jobs quickly and he can assign too many jobs to other employees. Or on the other hand, the boss can assign fewer jobs (or stop assigning new jobs at all) when the processing of some old one takes too much time (e.g. exception handling).

Recent results [FPY02, FMPY03] show that some of the schedulability problems related to preemptive scheduling are decidable. The decidability of the schedulability analysis has been proven for the following models. In [FPY02] only systems with one-way communication are considered. However, the execution times of tasks can be intervals though it is not stated clearly in this work. The best execution time is not important for the schedulability analysis of the systems with one-way communication. In [FMPY03] tasks can update data variables shared between them and the control automaton upon their completion (system with two-way communication), but the computation time should be a known constant for each task. The natural question is if schedulability analysis remains decidable for systems with two-way communication and interval execution times.

Unfortunately, the answer is negative. As the main technical contribution of this paper, we show that (1) the interval execution time of tasks, (2) the ability of the control automaton to test the exact completion time of tasks, and (3) preemption are sufficient and necessary to code the halting problem for two-counter machines. We also summarise previous decidability results and discuss other variants of the problem. Our goal is to identify as closely as possible the borderline between decidable and undecidable problems in schedulability analysis using timed automata. Hopefully, our result can be used as an indication in determining which classes of real-time models can be analysed efficiently.

The rest of the paper is organised as follows. In Section 11.2 we formally introduce our model and define the schedulability problem. A summary of previous decidability results is given in Section 11.3. Section 11.4 contains the undecidability proof of the schedulability problem for our model. We discuss several variants of this model in Section 11.5. Section 11.6 concludes the paper.

11.2 Preliminaries

To model two-way communication, we assume that each automaton has a distinguished clock, that is reset whenever a task finishes. This allows each task to announce its completion to the automaton. We have chosen resetting of the clock because even this simple model of two-way communication between tasks and the control automaton is sufficient for encoding of any two-counter machine. Other models of two-way communication, such as updating shared data variables upon completion are discussed later.

Syntax. Let \mathcal{P} ranged over by P, Q, R denote a finite set of task types. A task type may have different instances that are copies of the same program released at different time points. Each task type P is characterised as a pair $([B, W], D)$, where $[B, W]$ is the execution time interval and D is the deadline for P , $B \leq W \leq D \in \mathcal{N}_0$ and $W \neq 0$ (\mathcal{N}_0 denotes the set of non-negative integers). The deadline D is relative, meaning that when an instance of a task type P is released, it should finish within D time units. We use $B(P)$, $W(P)$, and $D(P)$ to denote the best execution time, the worst execution time, and the relative deadline of the task type P .

As in timed automata, assume a finite set of real-valued variables \mathcal{C} for clocks. We use $\mathcal{B}(\mathcal{C})$ ranged over by g to denote the set of conjunctive formulas of atomic constraints in the form: $a \sim K$ or $a - b \sim L$ where $a, b \in \mathcal{C}$ are clocks, $\sim \in \{\leq, <, \geq, >\}$, and K, L are natural numbers. We use $\mathcal{B}_I(\mathcal{C})$ for the subset of $\mathcal{B}(\mathcal{C})$ where all atomic constraints are of the form $a \sim K$ and $\sim \in \{<, \leq\}$. The elements of $\mathcal{B}(\mathcal{C})$ are called *clock constraints* or *guards*.

Definition 8 A *timed automaton extended with tasks*, over clocks \mathcal{C} and tasks \mathcal{P} is a tuple $\langle N, l_0, E, I, M, check \rangle$ where

- $\langle N, l_0, E, I \rangle$ is a *timed automaton* where
 - N is a finite set of locations,
 - $l_0 \in N$ is the initial location,
 - $E \subseteq N \times \mathcal{B}(\mathcal{C}) \times 2^{\mathcal{C}} \times N$ is the set of edges, and
 - $I : N \mapsto \mathcal{B}_I(\mathcal{C})$ is a function assigning each location with a clock constraint (a location invariant),
- $M : N \hookrightarrow \mathcal{P}$ is a partial function assigning locations with a task type,¹ and

¹Note that M is a partial function meaning that some of the locations may have no task.

- *check* $\in \mathcal{C}$ is the clock which is reset whenever a task finishes.

As a simplification we will use $l \xrightarrow{g,r} l'$ to denote $(l, g, r, l') \in E$.

For convenience, in the rest of the paper we use extended timed automata (ETA) or simply automata when it is understood from the context instead of timed automata extended with tasks.

Operational Semantics. Extended timed automata may perform two types of transitions just as standard timed automata. Intuitively, a discrete transition in an automaton denotes an event triggering a task. Whenever a task is released, it will be put in a scheduling (or task) queue for execution. A delay transition corresponds to the execution of the running task with the highest priority and idling for the other tasks waiting to run.

We represent the values of clocks as functions (called clock assignments) from \mathcal{C} to the non-negative reals. A state of an automaton is a triple (l, σ, q) where l is the current control location, σ the clock assignment, and q is the current task queue. We assume that the task queue takes the form: $[P(b_1, w_1, d_1), \dots, Q(b_n, w_n, d_n)]$ where $P(b_i, w_i, d_i)$ denotes a released instance of task type P with remaining best case computation time b_i , remaining worst case computation time w_i , and remaining relative deadline d_i .

By the *discrete part* of a queue

$$[P(b_1, w_1, d_1), Q(b_1, w_1, d_1), \dots, R(b_n, w_n, d_n)]$$

we denote the sequence $[P, Q, \dots, R]$ together with the information about the status of each task. A task status is either *waiting*, *running*, or *preempted*. The head of the queue has always status *running*, other tasks are *preempted* if they were already *running*, *waiting* otherwise.

A *queue reordering function* is a sorting function which changes the ordering of the task queue elements. It takes a task queue as an input and returns a task queue with the unmodified tasks that may be sorted in a different order. Let *task parameter test* be a comparison of the form $x \sim K$ or $x - y \sim L$ where $x, y \in \{b_i, w_i, d_i \mid i \in \mathcal{N}\}$, $\sim \in \{\leq, <, \geq, >\}$, and K, L are natural numbers.

Definition 9 A scheduling strategy (Sch) is a queue reordering function which

- when applied to two queues q_1, q_2 with the same discrete parts returns both of them in the same order if q_1 and q_2 cannot be distinguished by any task parameter test,
- changes the ordering of task instances of different types only, but never changes the ordering of task instances of the same type.

A *non-preemptive* strategy will never change the position of the first element in a queue, whereas a *preemptive* scheduling strategy may change the position of the first element in the queue (the one which is currently executed). E.g. FPS (fixed priority scheduling) or EDF (earliest deadline first) can be preemptive scheduling strategies. For example, $\text{EDF}([P(3.1, 4.9, 10), Q(4, 4.5, 5.3)]) = [Q(4, 4.5, 5.3), P(3.1, 4.9, 10)]$.

Run is a function which given a real number t and a task queue q returns the task queue after t time units of execution on a processor. The result of $\text{Run}(q, t)$ for $q = [P(b_1, w_1, d_1), Q(b_2, w_2, d_2), \dots, R(b_n, w_n, d_n)]$ is defined as $q' = [P(b_1 - t, w_1 - t, d_1 - t), Q(b_2, w_2, d_2 - t), \dots, R(b_n, w_n, d_n - t)]$. For example, let $q = [Q(2, 3, 5), P(4, 7, 10)]$. Then $\text{Run}(q, 3) = [Q(-1, 0, 2), P(4, 7, 7)]$ in which the first task has been executed for 3 time units (and it will be removed from the queue).

A task instance $P(b, w, d)$ in the queue may finish when $b = 0$ and $w \geq 0$, and it must finish when $w = 0$. Finished tasks are removed from the queue.

Further, for a non-negative real number t , we use $\sigma + t$ to denote the clock assignment which maps each clock a to the value $\sigma(a) + t$, $\sigma \models g$ to denote that the clock assignment σ satisfies the constraint g and $\sigma[r]$ for $r \subseteq \mathcal{C}$ to denote the clock assignment which maps each clock in r to 0 and agrees with σ for the other clocks (i.e. $\mathcal{C} \setminus r$). We omit braces when r is a singleton.

Definition 10 *Given a scheduling strategy Sch , the semantics of an extended timed automaton $\langle N, l_0, E, I, M, \text{check} \rangle$ with initial state (l_0, σ_0, q_0) is a transition system defined by the following rules:*

- $(l, \sigma, q) \xrightarrow{\text{Sch}} (l', \sigma[r], \text{Sch}(M(l') :: q))$ if $l \xrightarrow{g, r} l'$, $\sigma \models g$, and $\sigma[r] \models I(l')$
- $(l, \sigma, []) \xrightarrow{t} (l, \sigma + t, [])$ if $(\sigma + t) \models I(l)$
- $(l, \sigma, P :: q) \xrightarrow{t} (l, \sigma + t, \text{Run}(P :: q, t))$ if $t \leq w(P)$ and $(\sigma + t) \models I(l)$
- $(l, \sigma, P :: q) \xrightarrow{0} (l, \sigma[\text{check}], q)$ if $b(P) \leq 0 \leq w(P)$ and $\sigma[\text{check}] \models I(l)$

where $P :: q$ denotes the queue with the task instance P inserted in q (at the first position), and $[]$ denotes the empty queue.

Schedulability. We first mention that we have the same notion of reachability as for ordinary timed automata.

Definition 11 *We will write $(l, \sigma, q) \xrightarrow{\text{Sch}} (l', \sigma', q')$ if $(l, \sigma, q) \xrightarrow{\text{Sch}} (l', \sigma', q')$ or $(l, \sigma, q) \xrightarrow{t} (l', \sigma', q')$ for a delay t . For an automaton*

with initial state (l_0, σ_0, q_0) and for a scheduling strategy Sch , we say that (l, σ, q) is reachable iff $(l_0, \sigma_0, q_0) \xrightarrow{\text{Sch}}^* (l, \sigma, q)$.

Now we can formalise the notion of schedulability for extended timed automata.

Definition 12 (*Schedulability*) A state (l, σ, q) where $q = [P(b_1, w_1, d_1), \dots, Q(b_n, w_n, d_n)]$ is a failure denoted $(l, \sigma, \text{Error})$ if there exists i such that $d_i < 0$, that is, a task failed in meeting its deadline. Naturally an automaton \mathcal{A}_{ETA} with initial state (l_0, σ_0, q_0) is non-schedulable with Sch iff $(l, \sigma, \text{Error})$ is reachable for some l and σ . Otherwise, we say that \mathcal{A}_{ETA} is schedulable with Sch .² More generally, we say that \mathcal{A}_{ETA} is schedulable iff there exists a scheduling strategy Sch with which \mathcal{A}_{ETA} is schedulable.

11.3 Decidable Problems: a Summary

In this section we give a summary of the previous decidability results. The decidability results apply to some simpler variants of the schedulability problem for extended timed automata. We show, that whenever preemption, clock resets, or interval execution times are not allowed, the problem becomes decidable.

We will use the following notion of stopwatch automaton in our proofs. A *stopwatch automaton* [HKPV98] is a timed automaton where each location has assigned a set of clocks which are stopped when the automaton delays in this location, i.e. whose values do not increase during a delay transition.

To show all the results in the same framework we model the extended timed automata by stopwatches. The key observation for our results is the fact that the schedulability of an extended timed automaton with a given scheduling strategy can be reduced to the reachability problem for a stopwatch automaton. Adding stopwatches to timed automata turns the reachability problem undecidable in general [HKPV98], but we show that each decidable variant of the schedulability problem corresponds to some subclass of stopwatch automata with decidable reachability problem.

Lemma 3 *For every extended timed automaton \mathcal{A}_{ETA} and every scheduling strategy Sch , there exists a stopwatch automaton \mathcal{A}_{SA} with a location*

²Note that the state might not be failure denoted when ETA is deadlocked and time cannot progress, even if time flow would lead to a failure denoted state. We consider a system where such states are reachable but no failure denoted state is reachable as schedulable. This is not fundamental for our proofs and these states could be prohibited. But it would induce some technical difficulties making the undecidability proof less clear.

l_{Error} such that \mathcal{A}_{ETA} is schedulable with Sch iff the location l_{Error} in \mathcal{A}_{SA} is unreachable.

We construct \mathcal{A}_{SA} as a synchronous product (synchronizing on the common set of events called *synchronization channels*) of two automata $\mathcal{A}_{\text{control}}$ and $\mathcal{A}_{\text{queue}}$. The automaton $\mathcal{A}_{\text{control}}$ is the control automaton (the timed automaton part of \mathcal{A}_{ETA}) with synchronization channels added to the transitions leading to the locations where a task is released. The automaton $\mathcal{A}_{\text{queue}}$ is a stopwatch automaton encoding the task queue together with the scheduling strategy. It also moves to the location l_{Error} when a deadline is violated.

The main idea of the construction of $\mathcal{A}_{\text{queue}}$ is to encode the discrete part of the queue into its locations and to use special clocks c and d for each task instance in the queue to remember how long the task has been already computed and how long it has been in the queue. This information is sufficient to express the best and the worst remaining computation time and the remaining relative deadline of tasks in the transition guards.

The size of the task queue is unbounded in general, but the system becomes unschedulable when the queue size exceeds a certain bound. We can take the sum of all task deadlines as such a bound. If the queue size exceeds this bound then there are more task instances of the same task type in the queue than the time units within which all of them should finish. Because the worst case computation time of each task is at least one and due to the second condition in Definition 9 only one instance of a task type can be partially computed at a time, the last one will certainly miss the deadline if the execution of all these tasks takes the worst case computation time.

Therefore, there are only finitely many interesting discrete parts of the queues and we can encode the information about the current discrete part of the queue into locations of $\mathcal{A}_{\text{control}}$. If a new task arrives when the (bounded) queue is full the automaton moves to the location l_{Error} .

To store the current values of task parameters, clocks c_{Pi} (*computation clock*) and d_{Pi} (*deadline clock*) are introduced for all possible instances of all task types in the queue, where the subscript P denotes the task type and the subscript i denotes the index of this instance. These clocks are reused when an instance finishes. The instance index is not used to indicate the order in which tasks have arrived, but to distinguish different instances in the queue.

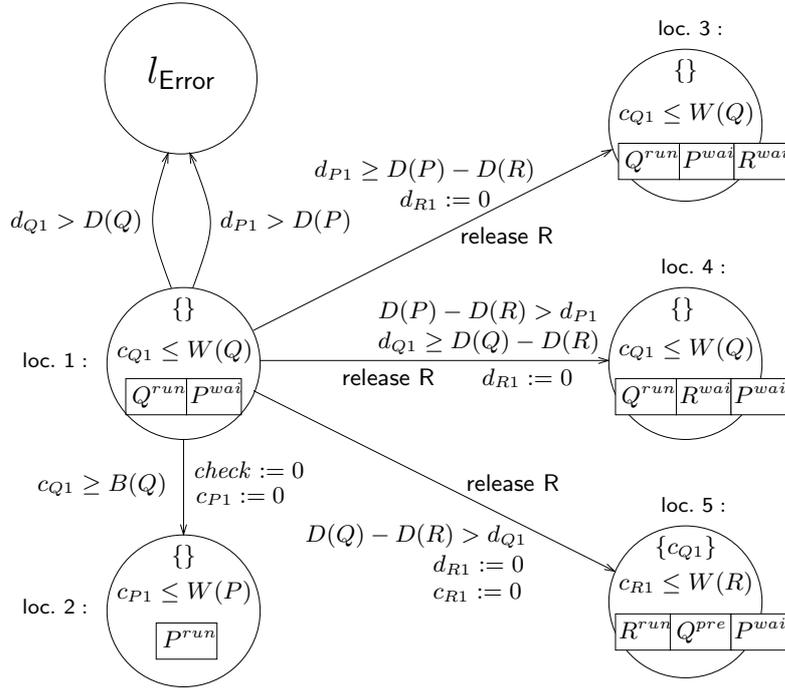
Deadline clock d_{Pi} measures how long has the task instance been in the queue. It is reset when the task instance is released ($\mathcal{A}_{\text{control}}$ and $\mathcal{A}_{\text{queue}}$ synchronize on the corresponding channel). To check that no task instance misses its deadline, we add a transition to l_{Error} to all locations where this task instance is in the queue. This transition is enabled if

and only if the value of the clock d_{P_i} exceeds the deadline of the task type P . The value of d_{P_i} does not play any role (it is never checked) in locations where this task instance is not in the queue.

Computation clock c_{P_i} is a stopwatch and it is used to measure how long has the task been computed on the processor. It is reset when the task changes its status from *waiting* to *running*, it runs in all locations where the corresponding task is running and it is stopped when the corresponding task is preempted. The task can finish when the value of its computation clock is greater than its best computation time and it must finish before the value of its computation clock exceeds the worst case computation time. The value of c_{P_i} does not play any role in locations where this task instance is not in the queue. When a task finishes, the clock *check* is reset.

The scheduling strategy *Sch* is applied when $\mathcal{A}_{control}$ and \mathcal{A}_{queue} synchronize on a task release channel. The automaton \mathcal{A}_{queue} takes transition from the location representing the current discrete part of the queue to a one representing the discrete part of the queue with the new task added and reordered by the scheduling strategy. There can be many transitions outgoing from one location that correspond to the release of the same task, each guarded by task parameter tests according to the scheduling strategy. Note, that all task parameter tests can be expressed as timed automata guards. Also, according to the Definition 9, no scheduling strategy can distinguish two queues with the same discrete part if they cannot be distinguished by a task parameter test. Therefore, any scheduling strategy can be encoded in a stopwatch automaton.

Example. The Figure 11.1 shows a fragment of \mathcal{A}_{queue} for \mathcal{A}_{ETA} with three tasks P , Q , and R . Location 1 represents the queue with two task instances, where the instance of Q is running and the instance of P is waiting. If the deadline of either P or Q is missed then the automaton can proceed to the location l_{Error} . If the computation clock of the running task instance gets greater than or equal to the best computation time of the task type Q (and it can never exceed the worst case computation time due to the invariant) then this instance can finish – the automaton proceeds to the location 2, resets the clock *check* and the computation clock of the next task in the queue (instance of P). The status of this task is changed to *running* in this location. Moreover, there are transitions with synchronization channels representing release of an instance of R outgoing from location 1 (we omit transitions for release of P and Q). Locations 3, 4, and 5 encode different results of reordering of the queue by the scheduling strategy (EDF, in this case). Only one transition is enabled for different orderings of remaining relative deadlines. For example, the guard on the transition to the location 3 is equivalent to

Figure 11.1: A part of an example stopwatch automaton \mathcal{A}_{queue} .

$d(R) \geq d(P)$,³ the fact that the remaining deadline of the newly arrived task is greater than or equal to the deadline of the last task in the queue. Because $d(R) = D(R)$ (this task instance did not spend any time in the queue) and $d(P) = D(P) - d_{P1}$ (this task instance has spent exactly d_{P1} time units in the queue), the constraint can be written as $d_{P1} \geq D(P) - D(R)$. The constraints on the other transitions are derived similarly. The location 5 corresponds to the situation when R preempts Q . Therefore, the computation clock c_{Q1} of Q is stopped in this location (all clocks are running in other locations). The deadline clock for R is reset when entering the locations 3, 4, and 5, and the computation clock for R is reset when entering the location 5.

Now we show that the schedulability problem is decidable for the first simplification of the general setting – we consider only non-preemptive scheduling strategies. At first, we prove that given a scheduling strategy the schedulability problem is decidable. Then we will argue that there are only finitely many different scheduling strategies.

³Here we denote by $d(R)$ and $d(P)$ the remaining deadline of the (only) instance of the task type R and P in the queue respectively.

Theorem 5 ([EWY98]) *The problem of checking schedulability with Sch for extended timed automata is decidable if Sch is a non-preemptive scheduling strategy.*

Proof: If we use the construction from Lemma 3 then for non-preemptive scheduling strategies we obtain (simple) timed automaton. Therefore, schedulability analysis with Sch is reduced to the non-reachability of a given location in a timed automaton, which is decidable.

The proof in [EWY98] handles only tasks with the worst case execution time equal to the best execution time and with one-way communication. We would get the same result by adding variable execution time and two-way communication. \square

Theorem 6 *The problem of checking schedulability for extended timed automata is decidable if we consider only non-preemptive scheduling strategies.*

Proof: We show that there are only finitely many different scheduling policies with which a given extended timed automaton can become schedulable. This implies that the decision procedure can check the schedulability of an extended timed automaton with all of these scheduling policies and answer positively if at least one check succeeds.

To show that there are only finitely many interesting scheduling policies recall that the task queue is bounded. Therefore, there are only finitely many different discrete parts of the queue to which a scheduling strategy can be applied. There are also only finitely many interesting task parameter tests, because all parameters are bounded by the deadline of their task type. In other words, we can consider only integer constants smaller than the greatest deadline in the task parameter tests. From this it follows that the number of task parameters is also finite. Therefore, there are only finitely many task parameter tests that do not have a constant value (either *true* or *false*) for all queues.

This argument is similar to region construction for timed automata. There are only finitely many different regions for task parameters and all values of task parameters within one region are equivalent with respect to a scheduling strategy. \square

Secondly, the schedulability problem is decidable for preemptive scheduling strategies when tasks are not allowed to communicate with an automaton (to reset clocks). We can view this system as an \mathcal{A}_{ETA} where the value of the clock *check* is never used. Execution times of tasks can be intervals.

There exists an optimal preemptive scheduling strategy for aperiodic scheduling – Earliest Deadline First (EDF) scheduling strategy [But97]. Therefore, checking of the schedulability is equivalent to checking of the

schedulability with EDF. If the system is not schedulable with EDF then no other scheduling strategy can make it schedulable.

EDF has an important property formalized in the following lemma.

Lemma 4 *Let the queue in \mathcal{A}_{ETA} be scheduled by EDF. If there is a run r_1 of \mathcal{A}_{ETA} leading to a failure denoted state along which some tasks finish before their worst case computation time, then there exists a run r_2 of \mathcal{A}_{ETA} leading to a failure denoted state along which computation times of all tasks are equal to their worst case computation times.*

Proof: We construct r_2 so that we let the control automaton to perform the same transitions as in r_1 and all tasks finish after the worst case computation time. The run r_2 is exactly the same as r_1 , except for the position of some zero-labeled transitions (transitions finishing a task execution). The computation time of all tasks in r_2 is equal to their worst case computation time, which together with the scheduling strategy (EDF) determines their finishing time, i.e. when the zero-labeled transitions are taken. We have to show that r_2 is a legal run of \mathcal{A}_{ETA} and that it leads to a failure denoted state.

Because there is just one-way communication in the system, different behaviour of the queue cannot influence the timed automaton part of \mathcal{A}_{ETA} . In particular, it does not influence validity of the invariants in the locations and it cannot disable transitions that were enabled along r_1 . Also, tasks cannot be forced to finish before their worst case computation time. Therefore, r_2 is a legal run of \mathcal{A}_{ETA} .

To show that r_2 leads to a failure denoted state we show that there is no time point where the remaining computation time of any task in r_1 is strictly smaller than its corresponding remaining computation time in r_2 .

We prove the following stronger proposition. Let (l, σ, q_1) and (l, σ, q_2) denote states of \mathcal{A}_{ETA} at the same time point along r_1 and r_2 respectively. Then q_1 can be obtained from q_2 by dropping some tasks, all remaining deadlines are the same for the tasks present in both queues, and all remaining computation times of tasks in q_1 are greater than or equal to the corresponding remaining computation times in q_2 . The proof is done by induction on the number of tasks releases.

1. The proposition holds trivially until the first task is released.
2. Assume that the proposition holds before n -th task P is released. When P is released, the EDF scheduling strategy is applied to the queue. It leaves the order of the current tasks in the queue unchanged, it only inserts P to the same position with respect to the tasks present in both queues. This is because EDF decides just according to the ordering of the remaining deadlines and they

depend only on the release times of tasks which are the same for q_1 and q_2 . In other words, if a task Q has the earliest deadline in q_2 and it is still in q_1 then it has also the earliest deadline there. Thus, the proposition holds after release of P .

Before a next task is released the following holds: when a task Q along r_2 is running then either the same task along r_1 is running or it has already finished. This follows from the induction hypothesis and from the fact that tasks along r_2 take the worst case computation time. Therefore, if a task in r_2 finishes then the same task also finishes in r_1 or it has already finished. Also the remaining computation times of tasks in the queue along r_2 are greater than or equal to their corresponding remaining computation times along r_1 .

□

As a consequence, we can consider only the worst case computation time of all tasks (all best computation times are equal to the worst computation times). If such modified system is unschedulable then it will not become schedulable for any values of the best computation times.

Another important property of EDF is that when a task P preempts a task Q then P will finish before Q is scheduled to run again. This can be easily seen from the fact that EDF decides only upon the comparisons of remaining relative deadlines whose order is constant in time. Therefore, each task can be preempted only for the time period which is the sum of computation times of all preempting tasks (tasks being computed when this task is preempted).

Theorem 7 ([FPY02]) *The problem of checking schedulability for extended timed automata with one-way communication (tasks do not reset clocks) is decidable.*

Proof: We show that checking schedulability with EDF is decidable for such systems. Due to Lemma 4 we consider all tasks having their best computation time equal to their worst case computation time. Following the construction from Lemma 3 we obtain a stopwatch automaton where the time for which clocks are stopped is an integer (sum of worst case computation times). Since the value of a stopped clock is never checked, we can replace stopwatches by clocks with subtraction. The amount of time for which a stopwatch was stopped is subtracted from this clock when the preempting task is finished (this amount is subtracted from computation clock of all preempted tasks). Because the value of subtracted clocks is bounded and tasks can be preempted only for integer amounts of time, we obtain bounded timed automaton with subtraction for which the reachability problem was proven decidable in [FPY02].

Similar ideas have also been used for suspension automata in [MV94].

□

The schedulability is decidable even for extended timed automata with two-way communication (tasks may reset clocks by the end of their execution) when the computation time is a known constant for each task.

Theorem 8 ([FMPY03]) *The problem of checking schedulability for extended timed automata is decidable if $B(P) = W(P)$ for all tasks P .*

Proof: The proof is given in [FMPY03]. We can also add resetting of the clock *check* to a bounded timed automaton with subtraction and use the argument from the proof of Theorem 7. □

When tasks have interval execution time in the system with two-way communication the computation of the control automaton can be influenced by the exact completion time of a task. The control automaton can take into consideration if a task has already finished or not and can proceed to different locations in different cases. For example, if a task does not finish in a given time the machine stops (it deadlocks, or proceeds to some idle state, i.e. no new tasks are released, just the already released tasks are computed). By this we can cut off (e.g. deadlock) branches where a task had “wrong” computation time. In fact, we can construct an automaton which proceeds to a certain location if and only if a task has been executed for some given exact (real) time.

Preemption enables us to sum up (accumulate) running times of tasks. Response time of the preempted task is increased by running time of each preempting task. This is sufficient to encode the two-counter machine.

11.4 Undecidability

Our main result in this paper is that the schedulability problem with fixed priority scheduling strategy for the automata defined in the previous section is undecidable. However, the proof does not depend on fixed priority scheduling strategy and it can be easily modified for almost all preemptive scheduling strategies (e.g. the proof holds for EDF without any modification).

Theorem 9 *The problem of checking whether extended timed automaton (defined in Definition 8) is schedulable with fixed priority scheduling strategy is undecidable.*

The proof is done by reduction of the halting problem for two-counter machine to the schedulability problem for ETA. A *two-counter machine* consists of a finite state control unit and two unbounded non-negative

integer counters. Initially, both counters contain the value 0. Such a machine can execute three types of instructions: incrementation of a counter, decrementation of a counter, and branching based upon whether a specific counter contains the value 0. Note that decrementation of a counter with the value 0 leaves this counter unchanged. After execution of an instruction, a machine changes deterministically its state. One state of a two-counter machine is distinguished as *halt state*. A machine halts if and only if it reaches this state.

We present an encoding of a two-counter machine M using extended timed automaton \mathcal{A}_M such that M halts if and only if \mathcal{A}_M is non-schedulable, based on the undecidability proofs of [HKPV98]. In the construction, the states of M correspond to specific locations of \mathcal{A}_M and each counter is encoded by a clock. We show how to simulate the two-counter machine operations. First, we adopt the notion of W-wrapping of [HKPV98].

Definition 13 *An extended timed automaton over set of clocks \mathcal{C} is W-wrapping if for all states (l, σ, q) reachable from its initial state and for all clocks $c \in \mathcal{C}$: $\sigma \models c \leq W$. A W-wrapping edge for a clock c and a location l is an edge from l to itself that is labeled with the guard $c = W$ and which resets the clock c . A clock that is reset only by wrapping edges is called system clock.⁴ Each time period between two consecutive time points at which any system clock contains value 0 is called W-wrapping period.*

We use wrapping to simulate discrete steps of two-counter machine. Each step is modeled by several W-wrapping periods. We define the wrapping-value of a clock to be the value of the clock when the system clock is 0. Note that a clock is carrying the same wrapping value if it is not reset by another edge than the wrapping edges. This principle is shown in Figure 11.2, where a is a system clock and clock t contains the same wrapping-value when the automaton takes transitions e_1 and e_3 .

We encode a two-counter machine M with counters C and D using a 4-wrapping automaton \mathcal{A}_M with one system clock denoted a and five other clocks c, d, h, t and *check*. In particular, we encode counters C and D of M by clocks c and d like this: counter value v corresponds to the clock wrapping-value 2^{1-v} . We use the density of the continuous domain to encode arbitrarily large values of the counters. Decrementation (incrementation) of a counter corresponds to doubling (halving) the wrapping-value of the corresponding clock. Test for zero corresponds to the check whether the clock wrapping-value equals to 2.

Now we show how to simulate the decrementation operation by doubling the wrapping-value of the clock d . To do this, we use two tasks:

⁴Note that all system clocks contain the same value.

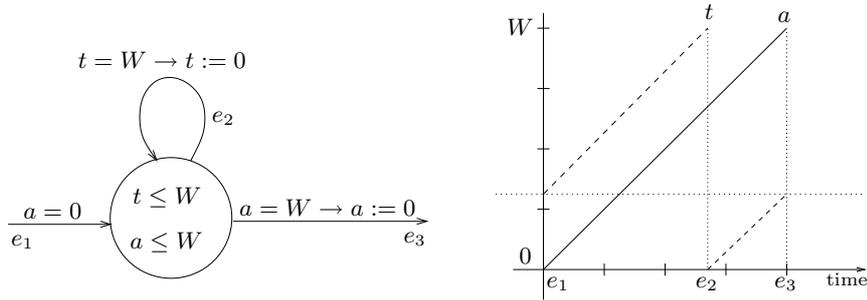


Figure 11.2: The wrapping edge e_2 makes clock t carry the same wrapping-value when the transitions e_1 and e_3 are taken.

short and *long*. The task *short* has execution time within interval $[0, 1]$ and deadline 50; the task *long* has execution time within interval $[8, 8]$ and deadline 100. The tasks reset clock *check* by the end of their execution. Moreover, the priority of *short* is higher than the priority of *long*, i.e. *short* always preempts *long*. Notice that the execution time of task *short* can vary and the execution time of the task *long* is fixed.

The basic idea of doubling a wrapping-value $v \in (0, 1]$ of clock d is as follows: we assume that the current wrapping-value of d is v . We copy it to clock t (that is, to make the wrapping-value of clock t to be v). We release the task *long* non-deterministically and reset d . The idea is to use d to record the response time for *long*. We release two instances of *short* before *long* finishes, that is preempt *long* twice by *short*. We make sure that the execution time of each of these two instances of *short* is exactly v time units. Note that v can be any real number within the interval $(0, 1]$. Then the response time for *long* is exactly $8 + 2v$. Note that if *long* finishes at a time point when the system clock a is reset to 0, the wrapping-value of d is $2v$. As *long* is released non-deterministically, there will be surely one such computation.

In Figure 11.3, we show the part of \mathcal{A}_M that doubles the wrapping-value of clock d . Figure 11.4 illustrates the time chart of the doubling process. Assume that a two-counter machine M is currently in state s_i and that it wants to decrease the counter D and then move to state s_j . The locations l_i and l_j of \mathcal{A}_M correspond to the states s_i and s_j respectively. Note that the dashed edge shows the transition of the two-counter machine (it is not a transition of \mathcal{A}_M). Note also that the decrementation operation leaves a counter with value 0 unchanged; the automaton can move from l_i directly to l_j through the transition e_0 when d contains the wrapping-value 2 (which corresponds to the counter value 0). Otherwise, the following steps are taken to double the wrapping-value of d .

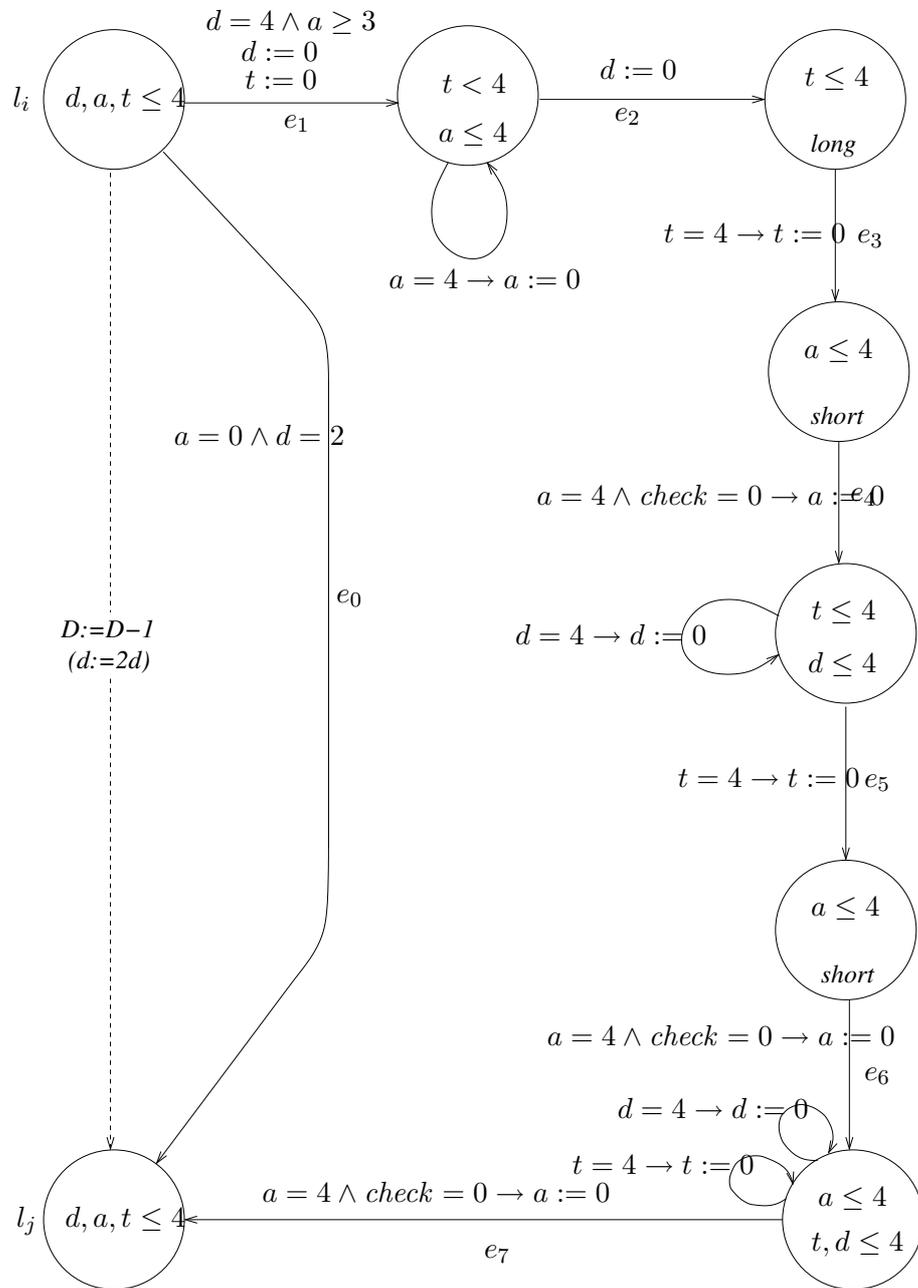


Figure 11.3: A part of reduction automaton corresponding to a decrementation of D . The wrapping edges for clocks $c, h, check$, and for all clocks in locations l_i, l_j are omitted. The location invariants $c \leq 4, h \leq 4$, and $check \leq 4$ are also omitted.

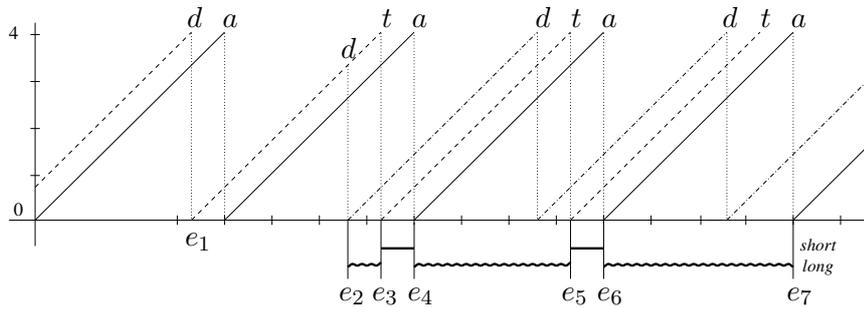


Figure 11.4: Time chart of the doubling procedure.

Firstly, the wrapping-value of d is copied to clock t (by transition e_1), that is, t carries the same wrapping-value as d . Then the automaton non-deterministically guesses the doubled wrapping-value of d (note that when d is reset, it will carry a new wrapping-value). It resets d at nondeterministically chosen time instant and at the same time it releases the task *long* (transition e_2).

The automaton waits until clock t reaches time 4, then resets t and releases *short* (transition e_3), which preempts *long*. Note that the wrapping-value of t will remain to be v and at this time point the value of the system clock a is $4 - v$. Therefore a will reach 4 in v time units.

The next transition e_4 is guarded by two constraints: $a = 4$, $check = 0$. To satisfy these constraints, the automaton has to wait in this location for v time units, and task *short* must finish at this time point, which resets the clock *check*.⁵

By this we make *short* run (and prevent *long* from running) exactly for v time units. Now we repeat this procedure again. That is, the automaton waits until $t = 4$. Then it releases the task *short* and forces it to run exactly for v time units (transitions e_5 and e_6).

Now, if the non-deterministic guess of the doubled wrapping-value of d was correct, task *long* must finish when $a = 4$, which makes the guard on e_7 become true and the automaton moves to location l_j .

So if the location l_j is reachable, the wrapping-value of d is $2v$. This is stated in the following lemma.

Lemma 5 *Let (l_i, σ, q) be an arbitrary state of the automaton shown in Figure 11.3 where $\sigma(d) = v$ and $v \in (0, 1]$, and q is empty. Then (l_j, σ', q') is reachable for some σ' and q' and if (l_j, σ', q') is reachable, it must be the case that $q' = []$, and $\sigma'(d) = 2v$.*

⁵We have to make sure that *check* is not reset by a wrapping edge when it is tested by a guard of the automaton. This causes no technical difficulties and it is omitted from Figure 11.3.

Proof:The proof is obvious from the construction in Figure 11.3. \square

To increment a counter we need to halve a wrapping-value of a clock, say c . For this, we use the clock h to copy the wrapping-value of c . The new wrapping-value v of c is nondeterministically guessed and it is checked by the above doubling procedure. If the wrapping-value of h (the original wrapping-value of c) is $2v$, then the automaton can proceed to the location corresponding to the destination state in an increment instruction.

To simulate branching, we construct two transitions outgoing from a location with guards $a = 0 \wedge c = 2$ and $a = 0 \wedge c \neq 2$. The initial state of M corresponds to a location where both c and d contain the wrapping-value 2. This can be achieved by integer guards and resets.

The halt state corresponds to the location *halt* with unguarded self-loop releasing the task *long* whenever it is visited. It follows that the automaton \mathcal{A}_M is schedulable if and only if the location *halt* is unreachable, i.e. the two-counter machine M does not halt.

11.5 Variants of the Problem

The proof can be easily modified even for some variants of the original setting. By this we want to show that the only sufficient conditions for undecidability are the following: the execution times of tasks are within intervals, an automaton can test the exact completion time of tasks, and one task can be preempted by another one.

The schedulability problem is undecidable if we use data variables to model two-way communication as described in [FMPY03] instead of a distinguished clock. Tasks can assign values to the variables shared between them and the control automaton. The automaton can use these variables in assignments and guards.

In our construction, tasks assign the value 1 to the data variable A upon completion. The edges e_4 and e_6 in the original construction are substituted by the edges e' , e'' and by the location l_{int} from Figure 11.5. The location where the task *short* is released is left exactly after t time units, but *short* is not finished yet ($A = 0$). However, the time does not flow in the location l_{int} . Just the task *short* can finish here. When this happens the edge e'' becomes enabled and the automaton can proceed. In fact, an automaton \mathcal{A}_M uses interleaving at the time instant when a task finishes to enforce and to measure correct execution time.

The schedulability problem is also undecidable if the tasks can only be released at integer time points. In this case, we encode a counter value v as a clock wrapping-value $4 - 2^{1-v}$. Decrementing (incrementing) of a counter does not correspond to doubling (halving) a clock wrapping-value anymore. Both instructions correspond to more com-

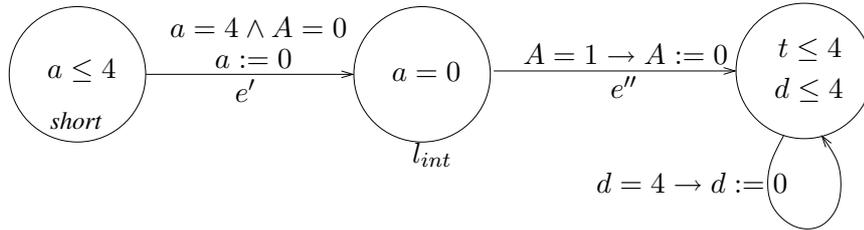


Figure 11.5: Substitution of the clock resets by interleaving and data updates.

plicated operations. Otherwise, the construction becomes even simpler. We do not need auxiliary clock t . Both *long* and *short* are released when $a = 0$. The task *short* should finish when $d = 0$ and we reset the clock d to obtain the doubled wrapping-value when the task *long* finishes. Synchronisation can be forced either by clock resets or by data variable updates. Figure 11.6 shows the time chart of the doubling procedure.

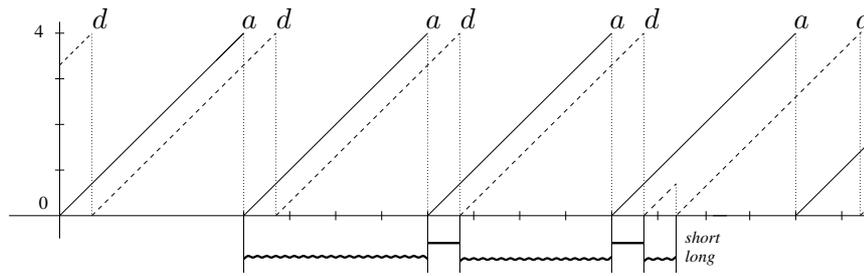


Figure 11.6: The time chart of the doubling procedure for integer release points.

Schedulability will also remain undecidable if we prohibit $B(P) = W(P)$, i.e. no task is allowed to have a constant computation time. Then we use the task $long_1$ with the execution interval $[7, 8]$ instead of *long*. The guessed wrapping-value of d can be less or equal to the correctly doubled value, because the task *long* can finish sooner. However, we repeat the whole doubling procedure with the task $long_2$ which has the execution interval $[8, 9]$. Now the automaton does not guess new wrapping-value of d , but uses the wrapping-value from the previous step. Therefore, this verifying procedure can succeed only if the wrapping-value of d was guessed correctly in the first doubling procedure.

Figure 11.6 also shows that we can use even weaker model of task releases. Each task is released periodically and it has assigned a boolean flag deciding whether it is enabled or not. It means that the value of the

flag corresponding to a task is checked at the beginning of each period. The task is released if and only if the value of the flag is true. Flags are updated by the timed automaton together with clock resets.

The task *long* has period 12 and it is enabled all the time (each instruction takes 12 time units). The task *short* has period 4 and it is enabled 4 and 8 time units after *long* was released. If *short* does not finish when the clock d is reset then the automaton proceeds to a location where all tasks are disabled forever. A new task *unschedulable* with period 1, computation time 2 and deadline 3 is introduced to ensure unschedulability when the location corresponding to the halt instruction of the two-counter machine is reached. This task is initially disabled and becomes enabled only when the automaton enters this location.

It is sufficient to use just one preemption for doubling a clock value. Therefore, even for systems where each task can be preempted only once during its execution the schedulability problem turns out to be undecidable. Figure 11.7 shows the time chart of the doubling procedure. The task *short* has the execution time within interval $[3, 4]$ and the task *long* has the execution time within $[4, 4]$. New doubled value is nondeterministically guessed and the guess is verified by the procedure.

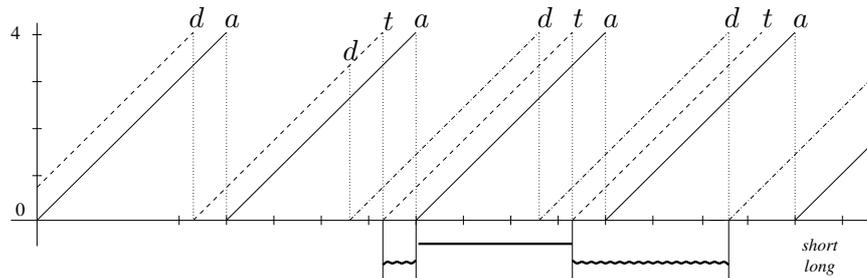


Figure 11.7: Time chart of the doubling procedure using just one preemption.

11.6 Conclusions

We have studied timed systems where preemption can occur at any real time point. For these systems, the schedulability checking problem is decidable if either the computation time of each task is a known constant or the control automaton cannot test the exact completion time of the tasks. We have showed that the scheduling problem becomes undecidable if both of these restrictions are dropped. By comparing this result with known decidability results, we try to identify the borderline between decidable and undecidable problems in schedulability analysis for these systems.

Bibliography

- [Abd02] Y. Abdeddaïm. *Scheduling with Timed Automata*. PhD thesis, Verimag, 2002.
- [ACH⁺95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.
- [AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- [AFM⁺02] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times - a tool for modelling and implementation of embedded systems. In *Proceedings of TACAS'02*, volume 2280 of *LNCS*, pages 460–464. Springer–Verlag, 2002.
- [AM01] Y. Abdeddaïm and O. Maler. Job-shop scheduling using timed automata. In *Proceedings of CAV01*, volume 2102 of *LNCS*, pages 478–492. Springer–Verlag, 2001.
- [AM02] Y. Abdeddaïm and O. Maler. Preemptive job-shop scheduling using stopwatch automata. In *Proceedings of TACAS02*, volume 2280 of *LNCS*, pages 113–126. Springer–Verlag, 2002.
- [But97] G. C. Buttazzo. *Hard Real-Time Computing Systems. Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.
- [CL00] F. Cassez and F. Laroussinie. Model-checking for hybrid systems by quotienting and constraints solving. In *Proceedings of CAV'00*, volume 1855 of *LNCS*, pages 373–388. Springer–Verlag, 2000.
- [Cor94] J. Corbett. Modeling and analysis of real-time Ada tasking programs. In *Proceedings of IEEE RTSS'94*, pages 132–141, 1994.
- [EWY98] C. Ericsson, A. Wall, and W. Yi. Timed automata as task models for event-driven systems. In *Proceedings of Nordic Workshop on Programming Theory*, 1998.
- [Feh02] A. Fehnker. *Citius, Vilius, Melius - Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. PhD thesis, KU Nijmegen, 2002.

- [FMPY03] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability analysis using two clocks. In *Proceedings of TACAS 2003*, volume 2619 of *LNCS*, pages 224–239. Springer–Verlag, 2003.
- [FPY02] E. Fersman, P. Pettersson, and W. Yi. Timed automata with asynchronous processes: Schedulability and decidability. In *Proceedings of TACAS’02*, volume 2280 of *LNCS*, pages 67–82. Springer–Verlag, 2002.
- [HHWT97] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):123–133, 1997.
- [HKPV98] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What’s decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.
- [HLP01] T. Hune, K.G. Larsen, and P. Pettersson. Guided Synthesis of Control Programs using UPPAAL. *Nordic Journal of Computing*, 8(1):43–64, 2001.
- [LPY97] K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.
- [MV94] J. McManis and P. Varaiya. Suspension automata: A decidable class of hybrid automata. In *Proceedings of CAV’94*, volume 818 of *LNCS*, pages 105–117. Springer–Verlag, 1994.
- [WH04] L. Waszniowski and Z. Hanzálek. Analysis of real time operating system based applications. In *Proceedings of FORMATS’03*, volume 2791 of *LNCS*, pages 219–233. Springer–Verlag, 2004.
- [Yov97] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.

Chapter 12

Undecidability of LTL for Timed Petri Nets

By **Parosh Aziz Abdulla, Pritha Mahata and Aletta Nylén**

Department of Information Technology

Uppsala University

Email: {parosh,pritha,aletta}@it.uu.se

We show undecidability of (action based) linear-time temporal logic (LTL) for timed Petri nets. This is to be contrasted with decidability of the problem of checking safety properties, boundedness and zenoness for timed Petri nets, and the problem of checking LTL formulae for (untimed) Petri nets. The undecidability result is shown through a reduction from a similar problem for lossy counter machines [May00]. We also show that the undecidability result holds even for timed Petri nets interpreted over a discrete-time domain.

12.1 Introduction

Petri nets are one of the most widely used graphical models for analysis and verification of concurrent systems. In order to capture the timing aspects of the concurrent systems, several timed extensions of Petri nets [RP85, MF76, BD91, RH80, HV87, CR83, SP85, GMMP91, AN01] are proposed (see [Bow96] for a survey).

In this paper we show undecidability of (action based) linear-time temporal logic (LTL) for *Timed Petri Nets (TPNs)* [AN01], in which each token is equipped with a real-valued clock. The firing conditions of a transition include the usual ones for Petri nets. Additionally, each arc between a place and a transition is labelled with a sub-interval of natural

numbers. When firing a transition, tokens which are removed (added) from (to) places should have ages in the intervals of corresponding arcs. The transitions are not forced to fire in this model. This means that we may choose to "let time pass" instead of firing enabled transitions, even if that disables a transition by making some of the needed tokens "too old".

In fact, we use a technique similar to [RGdFE99] and show that it is already undecidable to check a certain fixed property expressible in LTL for TPNs. More precisely, we show undecidability of whether there exists a computation of the TPN along which a given transition is fired infinitely often. The undecidability result is shown through a reduction from a similar problem for *lossy counter machines* [May00]. Finally we show that the problem is also undecidable even for timed Petri nets interpreted over a discrete-time domain.

Our result should be contrasted with decidability of the following related problems:

- Checking safety properties (whether something bad will ever happen!) for TPNs [AN00, AN01]. More precisely, in [AN01] it is shown how to characterize the set of markings in a TPN from which a given upward closed set of markings is reachable.
- Zenoness for TPNs: whether there is an infinite computation from a marking in a finite amount of time [AMM04]. In fact, this work characterizes the set of all markings from which there is such an infinite computation in finite time.
- Syntactic boundedness for TPNs: whether the size of each reachable marking is bounded by some natural number [AMM04].
- Checking LTL formulae for standard (untimed) Petri nets [Esp94]. In fact [BM99] shows that there is an effectively constructible representation of the set of markings satisfying a formula.

We also recall the following undecidability results.

- Even small fragments of branching time logics are known to be undecidable for Petri nets [Esp97, May01].
- All state based temporal logics are undecidable for Petri nets [Esp97].
- Semantic boundedness is undecidable both for TPNs and for discrete-timed Petri nets. In semantic boundedness, we ask the question whether the size of all the reachable markings with the dead

tokens (tokens which will never be used in any transition further) removed is bounded. The problem was shown to be undecidable for discrete timed Petri nets (TPNs interpreted over a discrete-timed domain) in [dFERA00], while in [AMM04] we show that this problem is also undecidable for TPNs.

Outline In the next section TPNs and the recurrent place problem for TPNs are introduced. In Section 12.3 we present LCMs and the recurrent state problem for LCMs. In Section 12.4 we prove that LTL is undecidable for TPNs. Finally in Section 12.5, we show the undecidability of recurrent place problem for TPNs interpreted over a discrete-time domain.

12.2 Timed Petri Nets

Timed Petri Nets (TPNs) are Petri nets where each token is equipped with a real-valued clock representing the age of the token. The firing conditions for transitions include the usual ones for Petri nets. Furthermore, each arc between a place and a transition is labeled with an interval. When a transition is fired, the tokens removed from the input places of the transition and the tokens added to the output places should have ages lying in the intervals of the corresponding arcs.

We let \mathbb{N} and $\mathbb{R}^{\geq 0}$ denote the sets of natural numbers and nonnegative reals respectively. A *multiset* B over a set A is a mapping from A to \mathbb{N} such that for $a \in A$, $B(a)$ is the number of occurrences of a in B . We define A^M to be the set of multisets over A . Sometimes we write multisets as lists, so e.g. $\langle 2.4, 5.1, 5.1, 2.4, 2.4 \rangle$ represents a multiset B over $\mathbb{R}^{\geq 0}$ where $B(2.4) = 3$, $B(5.1) = 2$ and $B(x) = 0$ for $x \neq 2.4, 5.1$. We may also write B as $\langle 2.4^3, 5.1^2 \rangle$. For multisets B_1 and B_2 over a set A , we say that $B_1 \leq B_2$ if $B_1(a) \leq B_2(a)$ for each $a \in A$. We define $B_1 + B_2$ to be the multiset B where $B(a) = B_1(a) + B_2(a)$, and (assuming $B_1 \leq B_2$) we define $B_2 - B_1$ to be the multiset B where $B(a) = B_2(a) - B_1(a)$, for each $a \in A$. We use \emptyset to denote the empty multiset, i.e., $\emptyset(a) = 0$ for each $a \in A$.

We use a set *Intrv* of *intervals* of the form $[a, b]$ or $[a, c)$, where $a, b \in \mathbb{N}$ and $c \in \mathbb{N} \cup \{\infty\}$. For $x \in \mathbb{R}^{\geq 0}$, we write $x \in [a, b]$ ($x \in [a, b)$) to denote that $a \leq x \leq b$ ($a \leq x < b$).

A *Timed Petri Net (TPN)* is a tuple $N = \langle P, T, In, Out \rangle$ where P is a finite set of *places*, T is a finite set of *transitions* and $In, Out : T \times P \rightarrow Intrv^M$ describes the flow relation. If $In(t, p) \neq \emptyset$ ($Out(t, p) \neq \emptyset$) we say that p is an *input (output) place* of t .

A *marking* M of N is a finite multiset over $P \times \mathbb{R}^{\geq 0}$. The marking M defines numbers and ages of the tokens in each place in the net. That is, $M(p, x)$ defines the number of tokens with age x in place p . For example if marking $M = \langle \langle p_1, 2.5 \rangle, \langle p_1, 1.3 \rangle, \langle p_2, 4.7 \rangle, \langle p_2, 4.7 \rangle \rangle$, then there are two tokens in p_1 with ages 2.5 and 1.3 and two tokens each with age 4.7 in the place p_2 in M . For each place p we define $M(p)$ to be the multiset over $\mathbb{R}^{\geq 0}$, where $M(p)(x) = M(p, x)$. Notice that untimed Petri nets are a special case in our model where all intervals are of the form $[0, \infty)$.

We define two types of transition relations on markings. A *timed transition* increases the age of all tokens by the same real number. Formally $M_1 \xrightarrow{\delta} M_2$ if M_1 is of the form $\langle \langle p_1, x_1 \rangle, \dots, \langle p_n, x_n \rangle \rangle$ and there is $\delta \in \mathbb{R}^{\geq 0}$ such that $M_2 = \langle \langle p_1, x_1 + \delta \rangle, \dots, \langle p_n, x_n + \delta \rangle \rangle$.

We define the set of *discrete transitions* \xrightarrow{D} as $\bigcup_{t \in T} \xrightarrow{t}$, where \xrightarrow{t} represents the effect of firing the transition t . More precisely, we define $M_1 \xrightarrow{t} M_2$ if, for each place p with $In(t, p) = \langle \mathcal{I}_1, \dots, \mathcal{I}_m \rangle$ and $Out(t, p) = \langle \mathcal{J}_1, \dots, \mathcal{J}_n \rangle$, there are multisets $B_1 = \langle x_1, \dots, x_m \rangle$ and $B_2 = \langle y_1, \dots, y_n \rangle$ over $\mathbb{R}^{\geq 0}$, such that the following holds.

- $B_1 \leq M_1(p)$.
- $x_i \in \mathcal{I}_i$, for $i : 1 \leq i \leq m$.
- $y_i \in \mathcal{J}_i$, for $i : 1 \leq i \leq n$.
- $M_2(p) = M_1(p) - B_1 + B_2$.

Intuitively, a transition t may be fired only if for each incoming arc to the transition, there is a token with the right age in the corresponding input place. This token will be removed from the input place when the transition is fired. Furthermore, for each outgoing arc a token with an age in the interval will be added to the output place.

We define the relation $\xrightarrow{\quad}$ to be $\xrightarrow{D} \cup \xrightarrow{\delta}$ and $\xrightarrow{*}$ as its reflexive, transitive closure.

For a marking M_0 of a TPN N , an M_0 -*computation* π of N is of the form M_0, M_1, M_2, \dots , where $M_i \xrightarrow{D} \xrightarrow{\delta} M_{i+1}$, for $i \geq 0$. We say that π *visits a place p infinitely often* if there are infinitely many i such that $M_i(p) \neq \emptyset$.

Example 1 *Figure 12.1 shows an example of a TPN where $P = \{Q, R, S\}$ and $T = \{t_1, t_2, t_3\}$. For instance, $In(t_2, Q) = [3, 5]$, $Out(t_2, R) = [0, 1]$ and $Out(t_2, S) = [1, 2]$. A marking of the given net is $M_0 = [(Q, 2.0), (R, 4.3), (R, 3.5)]$. A timed transition from M_0 is given*

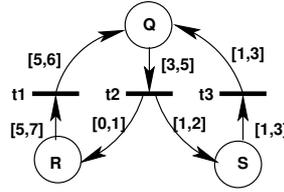


Figure 12.1: A small timed Petri net.

by $M_0 \xrightarrow{1.5} M_1$ where $M_1 = [(Q, 3.5), (R, 5.8), (R, 5.0)]$. An example of a discrete transition is given by $M_1 \xrightarrow{t_2} M_2$ where $M_2 = [(R, 0.2), (S, 1.6), (R, 5.8), (R, 5.0)]$.

The *recurrent place problem* for TPNs (RPP-TPN) is defined as follows.

Instance A TPN N , a marking M of N and a place p of N .

Question Is there an M -computation of N visiting p infinitely often?

In Theorem 11, we will show that RPP-TPN is undecidable.

12.3 Lossy Counter Machines

A *lossy counter machine (LCM)* is a tuple $L = \langle Q, C, \delta \rangle$, where Q is a finite set of *states*, C is a finite set of *counters* and δ is a finite set of *transitions*. A transition is a triple of the form $\langle q_1, instr, q_2 \rangle$, where $q_1, q_2 \in Q$ and *instr* is an *instruction*. An instruction is of one of the following three forms

- $c+$ which increases the value of counter c by 1.
- $c-$ which decreases the value of counter c by 1.
- $c := 0$ which resets the value of counter c to 0.

A *configuration* γ of L is of the form $\langle q, Val \rangle$, where $q \in Q$ and Val is a mapping from the set C of counters to the set \mathbb{N} of natural numbers. We define a transition relation \longrightarrow on the set of configurations such that $\langle q_1, Val_1 \rangle \longrightarrow \langle q_2, Val_2 \rangle$ iff one of the following conditions is satisfied:

1. $\langle q_1, c+, q_2 \rangle \in \delta$, $Val_2(c) = Val_1(c) + 1$ and $Val_2(c') = Val_1(c')$ if $c' \neq c$.

2. $\langle q_1, c-, q_2 \rangle \in \delta$, $Val_1(c) > 0$, $Val_2(c) = Val_1(c) - 1$ and $Val_2(c') = Val_1(c')$ if $c' \neq c$.
3. $\langle q_1, c := 0, q_2 \rangle \in \delta$, $Val_2(c) = 0$ and $Val_2(c') = Val_1(c')$ if $c' \neq c$.
4. $q_2 = q_1$, $Val_2(c) = Val_1(c) - 1$ for some $c \in C$, and $Val_2(c') = Val_1(c')$ if $c' \neq c$.

Abusing notations, we use $\xrightarrow{*}$ for denoting the reflexive, transitive closure of \longrightarrow for LCMs.

For a configuration γ_0 , a γ_0 -computation π of L is of the form $\gamma_0, \gamma_1, \gamma_2, \dots$, where $\gamma_i \longrightarrow \gamma_{i+1}$, for $i \geq 0$. For a state $q \in Q$, we say that π *visits q infinitely often* if there are infinitely many i such that γ_i is of the form $\langle q, Val_i \rangle$. The *recurrent state problem* for LCMs (RSP-LCM) is defined as follows.

Instance A LCM L , a configuration γ of L and a state q of L .

Question Is there a γ -computation of L visiting q infinitely often?

Theorem 10 [May00] *RSP-LCM is undecidable.*

12.4 Undecidability of LTL

In this section, we show undecidability of linear-time temporal logic (LTL) for TPNs. We do that by showing that RPP-TPN (Section 12.2) is undecidable through a reduction from RSP-LCM (Section 12.3). It is straightforward to show that RPP-TPN is reducible to the problem of checking whether there is a computation of the TPN along which a given transition is fired infinitely often. It follows that (action based) LTL is also undecidable.

Theorem 11 *RPP-TPN is undecidable.*

We show this through a reduction from RSP-LCM.

Theorem 12 *RSP-LCM can be reduced to RPP-TPN.*

Proof: We say that an instance of RSP-LCM or RPP-TPN is a positive instance if there is a computation such that the state or place is visited infinitely often, that is if the answer to the question in the definition of the problem is yes. Given an instance of RSP-LCM we construct an instance of RPP-TPN such that one of them is a positive instance if and

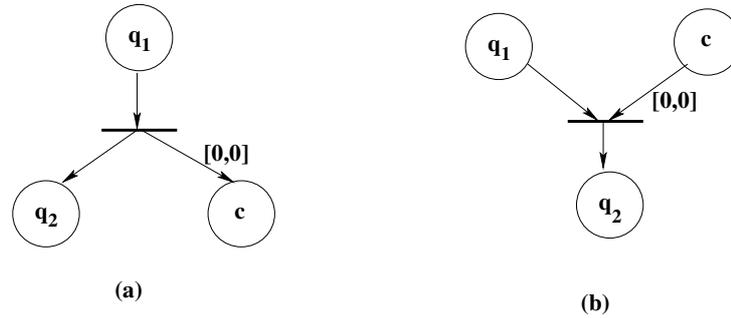


Figure 12.2: (a) Simulating the operation of increasing the counter. (b) Simulating the operation of decreasing the counter.

only if the other one is a positive instance. Suppose that we are given an instance of RSP-LCM, i.e., an LCM L , a configuration γ of L and a state q of L . We construct an equivalent instance of RPP-TPN, i.e., we derive a TPN N , a marking M of N , and a place p of N , such that RPP-TPN has a positive answer if and only if RSP-LCM has a positive answer.

Suppose that LCM $L = \langle Q, C, \delta \rangle$. We construct a corresponding TPN $N = \langle P, T, In, Out \rangle$ as follows. For each state $q \in Q$ there is a place in P which we call place q . We use P_Q to denote to the set of places of N corresponding to states. Also, for each counter $c \in C$ there is a place in P which we call place c . We use P_C to denote to the set of places corresponding to counters. Intuitively, the state of L is defined in N by the element of P_Q which contains a token. (The TPN N satisfies the invariant that there is at most one place in P_Q which contains a token). The value of counter c in L is defined in N by the number of tokens in place c which have ages equal to 0 (tokens which have ages more than 0 are considered to have been lost and do not affect the value of the counter). Losses in L are simulated either by making the age of the token strictly greater than 0, or by firing a special *loss* transition which can always remove tokens from the places in P_C . The flow relation corresponding to In and Out reflects these properties and is defined as follows.

- An *increment* $\iota = \langle q_1, c+, q_2 \rangle$ in δ is simulated by a transition ι in T which is of the form in Figure 12.2(a). The transition moves a token from place q_1 to place q_2 and adds a token of age 0 to place c . We do not let timed pass after this transition. Thus the increment operation is simulated by the sequence of transitions $\longrightarrow_{\iota} \longrightarrow_0$.

- A *decrement* $\iota = \langle q_1, c-, q_2 \rangle$ in δ is simulated by a transition ι in T which is of the form in Figure 12.2(b). The transition moves a token from place q_1 to place q_2 and removes a token of age 0 from place c . Again, we do not let any time pass after this transition. Thus the decrement operation is simulated by the sequence of transitions $\longrightarrow_{\iota} \longrightarrow_0$.
- For each place c in $P_C = \{c_1, \dots, c_n\}$ there is a transition which we call $loss_c$ (Figure 12.3).

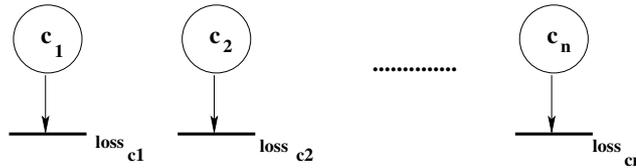


Figure 12.3: Simulating losses.

- A *reset* instruction ι of the form $\langle q_1, c := 0, q_2 \rangle$ has the most complicated simulation. The construction is shown in Figure 12.4.

We use two intermediate places r_i^1 and r_i^2 for each such instruction ι . The transition t_i^1 is fired by moving a token from place q_1 to place r_i^1 . The token in r_i^1 has to stay there for a time equal to 1 and then transition t_i^2 is fired. If more time passes, then this token in r_i^1 will forever stay in place r_i^1 after which no tokens will ever reside in any place in P_Q and thus the net will deadlock. The idea is that we reset the value of counter c to 0, by making the ages of all tokens in place c at least equal to 1. Observe that we simulate resetting the counter in L by resetting the counter in N . All tokens in each of the places in P_C which had age $[0, 0]$ have now age equal to 1. Thus, in order to avoid resetting the values of the counters other than c , we add, for each $c' \in C - \{c\}$ a new transition. In Figure 12.4, we assume that $C - \{c\} = \{c_1, c_2, \dots, c_n\}$, and thus we add the transitions $l_i^1, l_i^2, \dots, l_i^n$. These transitions are used to *refresh* the ages of the tokens in the places in $P_C - \{c\}$. Suppose $P_C \setminus \{c\} = \{c_1, \dots, c_n\}$. Now, if a token in place c_1 has its age equal to 1, and thus has become too old for firing other transitions (*increments* and *decrements*), it is replaced by a fresh token of age 0. The refreshing of the counters in $P_C \setminus \{c\}$ have to take place without letting any more time pass. Otherwise, the net will deadlock if the token in r_i^2 becomes older than 0. Finally, when the transition t_i^3 is fired, the new control state will be q_2 , and each token in place c will have an age which is at least one. The

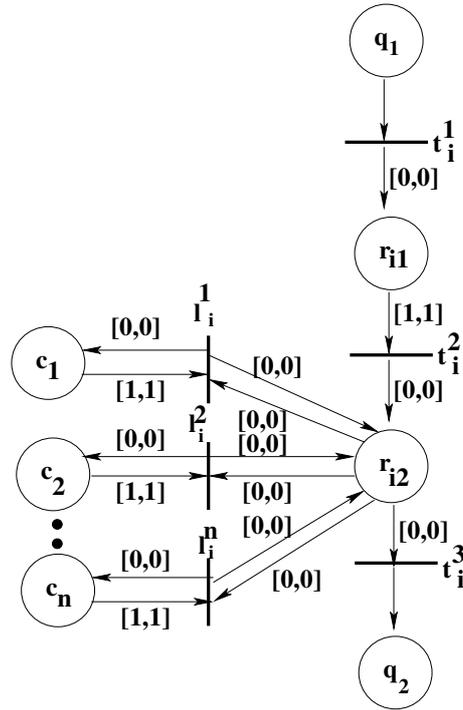


Figure 12.4: Simulating the operation of testing the value of the counter c .

resulting marking will therefore correspond to the counter c having the value 0. The refreshing process for the counters c_1, \dots, c_n will be stopped after firing t_i^3 , since the token in r_i^2 will now be removed. Notice that some tokens in c_1, c_2, \dots, c_n may be lost (i.e., may still have age equal to 1), since the TPN has a lazy semantics and these tokens may not have been refreshed.

Consider a marking M of N and a configuration $\gamma = \langle q, Val \rangle$ of L . We say that M is an *encoding* of γ if M contains a token in place q and the number of tokens with ages equal to 0 in place c is equal to $Val(c)$ for each $c \in C$. Furthermore, all other places in M are empty.

We also use the following intermediate encodings. Consider a reset instruction $\iota = \langle q_1, c := 0, q_2 \rangle$ in L .

- A marking is called an *intermediate-1 encoding* of a LCM-configuration $\gamma = (q, Val)$ if it has a token in place r_i^1 , no token in the places in P_C and the number of tokens in place c is equal to $Val(c)$.

- A marking is called an *intermediate-2 encoding* of a LCM-configuration $\gamma = (q, Val)$ if
 - it has a token in place r_i^2 with age 0 and no token in the places in P_C ,
 - the number of tokens in place c' with age less than or equal to 1 is less or equal to $Val(c')$ for each counter $c' \in P_C \setminus \{c\}$.
 - the number of tokens in place c with age 1 is equal to $Val(c)$.

We derive N from L as described above. We define M to be the encoding of γ and define p to be q .

Correctness of the construction:

\Rightarrow :

Given a γ_0 -computation π of L which visits a state q infinitely often, we show that there is a M_0 -computation π' such that M_0 is an encoding of γ_0 and π' visits the place q infinitely often.

To show this, it is enough to prove the following.

Given two configurations γ, γ' of L such that $\gamma \longrightarrow \gamma'$ and a marking M which is an encoding of γ , there is a sequence of transitions in N of the form $M = M_0 \longrightarrow_D M_1 \longrightarrow_\delta \cdots \longrightarrow_D \longrightarrow_\delta M_n = M'$ where $n \geq 1$ and the following holds.

- M' is an encoding of γ' .
- M_i is an intermediate encoding for $0 < i < n$.

Since $\gamma \longrightarrow \gamma'$, we know that γ' is derived from γ , using one of the four possible types of transitions described for LCMs. We show the claim only for the least obvious case, namely when γ' is derived from γ by executing a reset instruction ι which resets the value of a counter c for 0. The other cases can be explained in a similar manner. Let $\gamma = \langle q, Val \rangle$ and $\gamma' = \langle q', Val' \rangle$. We show that there is a marking M' which is an encoding of γ' . Since M is an encoding of γ , it means that place q_1 in M contains a token. From the construction described above (Figure 12.4) we know that from M , we can fire t_i^1 and produce a marking M_1 such that $M \xrightarrow{t_i^1} M_1$ and $M_1 = M - (q_1, x) + (r_i^1, 0)$. This means that $M_1((c', 0)) = M((c', 0))$ for each counter $c' \in P_C$. Next we let time pass by one time unit and obtain a marking M_2 such that $M_1 \xrightarrow{1} M_2$. This means that $M_2((c', 1)) = M_1((c', 0))$ for each counter $c' \in P_C$. Now, the transition t_i^2 is enabled from M_2 and firing t_i^2 from M_2 yields a marking M_3 such that $M_2 \xrightarrow{t_i^2} M_3$ and $M_3((c', 1)) = M_2((c', 1))$ for each counter $c' \in P_C$. Now suppose for a counter $c_1 \in P_C \setminus \{c\}$, $Val(c_1) = n$. Then

we fire the transition l_i^1 n times and refresh all n tokens of age 1 in c_1 to age 0. Similarly we refresh all tokens of age 1 in other counters in $P_C \setminus \{c\}$. Each of these transitions are executed without letting any time pass. Notice that the refreshing phase always terminates. Since we did not let time pass during 'refreshing' phase, there is still a token in r_i^2 of age 0 and we fire the transition t_i^3 by moving the token from r_i^2 to q_2 , yielding a marking M' . This means that for each counter $c' \in P_C \setminus \{c\}$, $M'((c', 0)) = M((c', 0))$. Furthermore, the ages of all tokens in place c is equal to 1, i.e., $M'((c, 1)) = M((c, 0))$. This means that the new marking M' will be an encoding of γ' . Notice that we do not pass time after firing t_i^3 .

⇐:

Suppose that there is an infinite M -computation π of N visiting place q infinitely often. Let π be of the form M_0, M_1, \dots . Consider the maximal subsequence $\pi' = M'_0, M'_1, \dots$ of π , where each M'_i is an encoding of some configuration of L . The sequence π' exists and is infinite, since q is visited infinitely often.

We prove that there is an infinite γ_0 -computation visiting state q infinitely often, where M'_0 is an encoding of γ_0 . To prove this, it is enough to show that given two markings M'_i and M'_{i+1} in π' such that $M'_i \xrightarrow{*} M'_{i+1}$ and a configuration γ_{j_i} which is encoded by M'_i , there is a configuration $\gamma_{j_{i+1}}$ such that $\gamma_{j_i} \xrightarrow{*} \gamma_{j_{i+1}}$. Let $\gamma_{j_i} = \langle q_i, Val_i \rangle$.

Since $M'_i \xrightarrow{*} M'_{i+1}$ we know that there are $M''_0, M''_1, \dots, M''_m$ such that $M''_0 = M'_i$, $M''_m = M'_{i+1}$ and there are intermediate encodings M''_1, \dots, M''_{m-1} such that $M''_0 \longrightarrow M''_1 \longrightarrow \dots \longrightarrow M''_m$. There are two cases. Either $m = 1$ or $m > 1$.

If $m = 1$, then there are again two cases.

- We have $M'_i \longrightarrow_D M'_{i+1}$. In this case we know that M'_{i+1} can be derived from M'_i by firing a discrete transition in D corresponding to one among Figures 12.2(a), 12.2(b) and 12.2(c). In this case, $\gamma_{j_{i+1}}$ is obtained from γ_{j_i} by executing an increment/decrement/loss instruction of L .
- We have $M'_i \longrightarrow_\delta M'_{i+1}$ where $\delta > 0$. In this case, we know that $\gamma_{j_{i+1}}$ can be derived from γ_{j_i} by losing the current values of all the counters from γ_{j_i} .

If $m > 1$, then M'_{i+1} is obtained from M'_i by firing transitions corresponding to those in Figure 12.4 (these are the only transitions in N which can make all places in P_Q empty and thus prevent the markings M''_1, \dots, M''_{m-1} from being encodings of configurations of L). Notice that

$M_0'' \xrightarrow{t_i^1} M_1'' \xrightarrow{1} M_2'' \xrightarrow{t_i^2} M_3'' \dots \xrightarrow{1} M_{m-1}'' \xrightarrow{t_i^3} M_m''$ and M_1'', M_2'' are intermediate-1 encodings, while M_3'', \dots, M_{m-1}'' are intermediate-2 encodings. This means that $\langle q_i, c := 0, q_{i+1} \rangle$ is an instruction in L , for some counter c . From the construction of Figure 12.4, we know that all tokens in place c will eventually have age at least 1. Furthermore, the ages of some of the tokens in $P_C - \{c\}$ may also exceed 1, since not all tokens need to be refreshed. We can derive γ_{i+1} from γ_i by first performing loss transitions corresponding to tokens which become too old followed by executing the instruction $\langle q_i, c := 0, q_{i+1} \rangle$.

□

12.5 Discrete-Timed Petri Nets

In the previous section, we showed that repeated reachability of a place in TPN is undecidable. This result also holds if TPNs are interpreted over discrete-timed domain. The semantics of a discrete-timed Petri net is different from that of a dense-timed Petri nets in the following way.

- Firstly, the ages of the token are natural numbers rather than real numbers.
- Secondly, timed transition takes only discrete steps.

Our construction in Section 12.4 can also be used to show undecidability of repeated place reachability of a discrete-timed Petri net, since it does not have any assumption on dense-time (all intervals are closed). Thus, action-based LTL is also undecidable for discrete-timed Petri nets.

Acknowledgement

We would like to thank Richard Mayr for helpful discussions.

Bibliography

- [AMM04] P. A. Abdulla, P. Mahata, and R. Mayr. Decidability of zenoness, syntactic boundedness and token-liveness for dense-timed petri nets. In *Proc. FST&TCS04*, volume 3328 of *Lecture Notes in Computer Science*, pages 59–71, 2004.
- [AN00] P. A. Abdulla and A. Nylén. Better is better than well: On efficient verification of infinite-state systems. In *Proc. LICS'*

- 00 16th *IEEE Int. Symp. on Logic in Computer Science*, pages 132–140, 2000.
- [AN01] P. A. Abdulla and A. Nylén. Timed Petri nets and BQOs. In *Proc. ICATPN'2001: 22nd Int. Conf. on application and theory of Petri nets*, volume 2075 of *Lecture Notes in Computer Science*, pages 53–70, 2001.
- [BD91] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. on Software Engineering*, 17(3):259–273, 1991.
- [BM99] A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *Symp. on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 323–333, 1999.
- [Bow96] F. D. J. Bowden. Modelling time in Petri nets. In *Proc. Second Australian-Japan Workshop on Stochastic Models*, 1996.
- [CR83] J. E. Coolahan and N. Roussopoulos. Timing requirements for time driven system using augmented petri nets. In *IEEE Transactions on Software Engineering*, volume SE9, 1983.
- [dFERA00] D. de Frutos Escrig, V. Valero Ruiz, and O. Marroquín Alonso. Decidability of properties of timed-arc Petri nets. In *ICATPN 2000*, number 1825, pages 187–206, 2000.
- [Esp94] J. Esparza. On the decidability of model checking for several mu-calculi and Petri nets. In *CAAP '94*, volume 787 of *Lecture Notes in Computer Science*, pages 115–129. Springer Verlag, 1994.
- [Esp97] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.
- [GMMP91] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezzè. A unified high-level Petri net formalism for time-critical systems. *IEEE Trans. on Software Engineering*, 17(2):160–172, 1991.
- [HV87] M. A. Holliday and M. K. Vernon. A generalized timed petri net model for performance analysis. In *IEEE Transactions on Software Engineering*, volume SE13, 1987.
- [May00] R. Mayr. Undecidable problems in unreliable computations. In *Theoretical Informatics (LATIN'2000)*, number 1776 in *Lecture Notes in Computer Science*, 2000.

- [May01] R. Mayr. Decidability of model checking with the temporal logic *cf.* *Theoretical Computer Science*, 256:31–62, 2001.
- [MF76] P. Merlin and D.J. Farber. Recoverability of communication protocols - implications of a theoretical study. *IEEE Trans. on Computers*, COM-24:1036–1043, Sept. 1976.
- [RGdFE99] V. Valero Ruiz, F. Cuartero Gomez, and D. de Frutos Escrig. On non-decidability of reachability for timed-arc Petri nets. In *Proc. 8th International Workshop on Petri Nets and Performance Models*, pages 188–196, 1999.
- [RH80] C. V. Ramamoorthy and G. S. Ho. Performance evaluation of asynchronous concurrent systems using petri nets. In *IEEE Transactions on Software Engineering*, volume SE6, 1980.
- [RP85] R. Razouk and C. Phelps. Performance analysis using timed Petri nets. In *Protocol Testing, Specification, and Verification*, pages 561–576, 1985.
- [SP85] P. D. Scotts and T. W. Pratt. Hierarchical modelling of software systems with timed petri nets. In *1st International Workshop on Timed Petri Nets, Torino, Italy*, July 1985.

Chapter 13

A Framework for Analysis of Timing and Resource Utilization targeting Complex Embedded Systems

By **Johan Andersson, Anders Wall** and **Christer Norström**

Department of Computer Science and Engineering
Mälardalen University

Email: {johan.x.andersson, anders.wall, christer.norstrom}@mdh.se,
aal98001@student.mdh.se

A problem in common of many complex software systems embedded in industrial products is the absence of analyzability as formal models of the system behavior does not exist. When performing maintenance of such systems it is hard to predict how changes will impact specific system properties related to timing and resource utilization and there is therefore a significant risk of running into problems with unexpected side-effects of the changes made, which increases development time required and costs.

In this paper we present the ART Framework, a set of methods and tools that enable behavior impact analysis for existing industrial real-time systems. The ART Framework enables developers of complex software systems to identify problematic side-effects of a proposed design before vast resources have been invested in implementation and testing. This reduces the risk of expensive and time-consuming problems discovered late in a development project and also reduces the risk releasing software containing latent critical errors.

13.1 Introduction

Large industrial software systems evolve as new features are being added. This is necessary for the companies in order to be competitive. However, this evolution typically causes the software architecture to degrade, leading to increased maintenance costs. Systems of this kind, e.g. industrial robot control systems, automotive systems and process control systems, have typically evolved considerably from their first release as a result from many years of maintenance and are today maintained by a staff where most of the people were not involved in the initial design of the system. Most such systems are not analyzable today as they lack formal models describing their behavior.

The architectural degradation is a result of maintenance operations (e.g. new features and bug fixes) performed in a less than optimal manner due to e.g. time pressure or insufficient documentation. As a result of these maintenance operations, not only the size but also the complexity of the system increases as new dependencies are introduced and architectural guidelines are broken. As a result it becomes harder and harder to predict the impact a certain maintenance operation will have on the system's behavior and the risk increases that an error occur due to an unexpected side-effect of a change. For industrial software systems with real-time requirements this is especially a problem, since side-effects on timing can cause critical errors. Moreover, such errors are often very hard to find when testing the system, as they might only occur very specific, rare situations that are hard to reproduce [Sch].

By introducing analyzability with respect to properties of interest, e.g. response times, the understandability of the system can be increased and side-effects of suggested designs can be predicted in early phases of development, before a lot of resources have been invested in development and testing. Thereby, the development cost and time required for new features can be reduced, and the risk of releasing software with latent critical errors is reduced.

Introducing analyzability, and consequently introducing the possibility of understanding the impact that changes will have on the system behavior with respect to timing, can be done in two distinct ways: intrusively or non-intrusively. In an intrusive approach the system is re-designed in order to make it analyzable. An example of an intrusive approach is to redesign the system to fulfilling the requirements of the fixed priority scheduling principle. The intrusive approach is, however, associated with a high cost as it might require a considerable effort to re-design the system. It is also associated with a high risk since errors might be introduced that, in worst case, is not captured during testing.

A non-intrusive approach is to construct an analyzable model of the system. Hence, the system is kept intact and unchanged which mini-

mizes the cost and the risks. There is however risks with this approach as well. The model might not be updated as the system evolves, e.g. due to time pressure.

The work presented in this paper focuses on the latter approach. Initially a system model is constructed based on both the structure and the dynamic behavior of the system. The model is thereafter validated with respect to interesting system properties and expected types of changes. Given that a model has been constructed and validated, it can be used for Impact Analysis, i.e. predicting the impact a change will have on the runtime behavior of the system.

The work originates in a case study at ABB Robotics in Sweden, where a statistical model describing the temporal behavior of a large industrial real-time system was constructed [A. 03]. The work resulted in the development of the ART Framework, a probabilistic modelling and analysis framework, including a modelling language and analysis tools. The system studied at ABB is based on VxWorks, a commonly used Real-Time OS [Win].

Apart from Impact Analysis, there is another use of the tools presented in this paper, Regression Analysis. Instead of analyzing a model, after each major change of the system, measurements are made of the current implementation and analyzed with respect to important system properties. The results from the analysis are compared with results from previous versions of the system. Unexpected differences can point out potential problems and undesired behavior.

The contribution of this paper is a presentation of the ART Framework, including methods for Impact Analysis, Regression Analysis, Model Validation as well as languages and tools support.

The paper is organized as follows: In the next section we present related work, Section 13.3 presents the general approach, the ART Framework. Section 13.4 is an overview of the two languages included in this framework, the modelling language ART-ML and the property language PPL. Section 13.5 presents a method for validation of simulation models. In Section 13.6 we present a set of tools developed to support this approach. Section 13.7 describes an industrial case study where the feasibility of this approach has been evaluated and finally, in Section 13.8, we conclude the paper and give hints on future work.

13.2 Related Work

Dynamic Analysis is the area of analyzing data generated by a program at execution time and includes e.g. performance analysis, error localization and runtime system monitoring. There are quite a lot of work within the area that relates to this, since we base the ART Framework

on the concept of analyzing observations of the system, i.e. recorded data. Parts of the work within the Dynamic Analysis community also deals with reconstructing architectural descriptions of systems, based on observations.

For instance, a system called DiscoTech is presented in [YGS⁺04]. Based on run time observations an architectural view of the system is constructed. If the general design pattern used in the system is known, mappings can be made that transforms low level system events into high level architectural operations and from that construct an architectural description of the system. The system presented is designed for Java based systems. The type of operations that are monitored are typically object creation, method invocation and instance variable assignments. Automated, or mechanical, generation of models based on observations of the system is very related to the construction of the structural model described in Section 13.3.1. We intend to investigate automated generation/validation of ART-ML models in future work.

Another work related to the construction of ART-ML models is [vDHK⁺04]. They present a process for reconstructing software architectures, Symphony. The process incorporates the state of the practice, is problem-driven and uses a rich set of architectural views. It provides guidance for performing reconstruction. Symphony consists of two stages. The first stage is to create a reconstruction strategy, selecting what views to reconstruct. The second stage is the execution of the strategy, i.e. to perform the reconstruction of the selected earlier views.

An approach for deterministic replay is presented in [TSHP03]. A common problem when debugging real-time systems is to be able to reproduce an error in order to find the source of this error, i.e. the bug. Due to behaviors such as task-switches and interrupts, finding the bug by studying the code alone is very hard. In their approach, they instrument a real-time system with software probes, collecting various data describing the state of the system. After an error has been observed, the data can be stored and used to replay the execution using a debugger. This is very related to the ART Framework, since they use a similar technique for recording, i.e. software probes, records similar things and has the same overall purpose, to make it easier to develop complex dependable systems reliability. There are clear differences as well. Compared to our work, they record much more details of the execution of the system, but for a much shorter time. They use this very detailed data to exactly reproduce an execution, in order to find bugs, while the data recorded in the ART Framework is used to build probabilistic models, enabling Impact Analysis.

There is a lot of work within the area of formal methods. Model Checking is a technique for verifying different properties of models. For real-time systems, a commonly used tool is Uppaal [BLL⁺95, BDL⁺01].

Uppaal is an integrated tool environment for modeling, simulation and verification of real-time systems. Uppaal is suitable for systems which can be modeled as a collection of nondeterministic processes with finite control structure and real-valued clocks that are communicating through channels or shared variables. Some major areas where this is applied include real-time controllers and communication protocols where especially those in which timing aspects are critical. A general problem with model checking is the state-space explosion. The general idea behind Model Checking is to search all states of the model for a certain condition. However, if the model contains a lot of parallel processes and clocks, the number of states easily becomes overwhelming and thus too large to search. Compared to the simulation approach in this work, Model Checking gives a lot higher confidence, since all states of the model is explored. However, the state space explosion problem limits the complexity of the models that can be analysed, so in many situations Model Checking is not an option and Model Checking has the same problems with model validity as the simulation based approach in this work.

A tool-suite called STRESS is presented in [ABRW94]. The STRESS environment is a collection of tools for analyzing and simulating the behavior of hard real-time safety-critical applications. STRESS contains a modeling language where the behavior of the tasks in the system can be modeled. It is also possible to define algorithms for resource sharing and task scheduling. STRESS is in some ways similar to the ART Framework, but there are a lot of differences too, as STRESS is primarily intended as a tool for testing scheduling and resource management algorithms. It does not allow probabilistic modeling like the ART Framework.

Another simulation framework called DRTSS is presented in [SL96]. DRTSS is a high level simulation framework that allows its users to construct discrete-event simulators of complex, multi-paradigm, distributed real-time systems. The DRTSS framework contains a set of algorithms and protocols from which one can pick the appropriate ones and build a simulator. New algorithms and protocols can be added to the original set. It has support for searching for extremes in the timing behavior of the simulated system. DRTSS is a part of the PERTS tool-suite, which was developed at the University of Illinois at Urbana-Champaign. The PERTS tool-suite has been commercialized by Tri-Pacific Software Inc. [Tri].

Analytical methods for dealing with probabilistic temporal attributes have been proposed in the literature. In [MEP01], an analytical method for temporal analysis of task models with stochastic execution times is presented. However, sporadic tasks cannot be handled. A solution for this could not easily be found. Without fixed inter-arrival times,

i.e. in presence of sporadic tasks, a least common divider of the tasks inter-arrival times can not be found.

Another analytical approach to probabilistic analysis is presented in [LN03]. Here they assume execution times and deadlines that both vary over time in an unpredictable manner, while their arrival times are fixed. Basically, the task model consists of a set of scenarios where every scenario is associated with a probability. For instance, a task may arrive with a certain execution time and deadline with a specified probability. Tasks execute probabilistically depending on several factors, e.g. the scheduling algorithm. The paper proposes solutions for Earliest Deadline First (EDF), and Least Laxity First (LLF). Even though the computational complexity of this solution has not yet been established, it seems, intuitively, that it is quite large.

13.3 Concepts of the ART Framework

The purpose of the ART Framework is to increase the maintainability, reliability, and understandability of complex real-time software systems by introducing analyzability. The core of the framework is the process describing how the model is constructed and used. Since this process is general it can be instantiated using any appropriate modeling and analysis method. The process is intended to be integrated in the life-cycle process at software development organizations. The general idea in the ART Framework is the use of a model for impact analysis of timing and utilization of logical resources caused by maintenance operations, e.g. changing an existing feature or adding a new feature. Models are constructed through reverse engineering of an existing system's implementation by identifying the architectural structure and by profiling of the runtime system, an example being execution time distributions for features or tasks. We will start with a brief overview of the process (Figure 13.1) and describe its individual steps in more details later on in this paper. The process consists of five steps:

1. Construct (or update) a structural model of the system, based on system documentation and the source code.
2. Populated the structural model with data measured on a running system. This data is typically probabilities of different behaviors and execution times.
3. Validate the constructed model by comparing predictions made using the model with observations of the real system. If the model does not capture the system's behavior sufficiently, the first two steps are repeated in order to construct a better model and the

new model is validated. This process should be repeated until a valid model is achieved.

4. Use the model for prototyping a change to the system, for instance if a new feature is to be added, the model is used for prototyping the change.
5. Analyse the updated model in order to identify any negative effects on the overall system behavior, such as deadline misses or starvation on critical message queues.

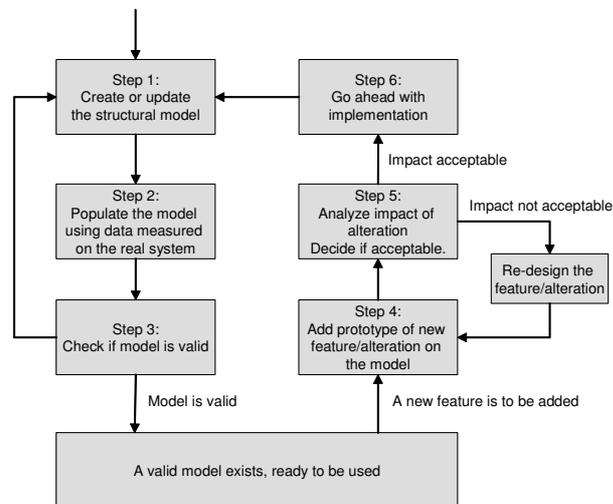


Figure 13.1: The process of introducing and using a model for impact analysis

If the impact of the change on the system is unacceptable, the change should be re-designed. On the other hand, if the results from the analysis are satisfactory the change can be implemented in the real system. The final step when changing the system is to update the model so that it reflects the implementation in the real system by profiling the system in order to update the estimated execution times, steps 1-3 in Figure 13.1.

13.3.1 Constructing the structural model

To construct a structural model of a system is to document the architecture and behavior of the system in a notation suitable for analysis, at an appropriate level of abstraction. The resulting model describes what tasks there are, their attributes such as scheduling priority and their behavior with respect to timing and interaction with other tasks using e.g. message queues and semaphores. To construct this model requires not

only studies of the system documentation and code, but also to involve system experts throughout the complete process. This is important in order to select what parts of the system to focus the modeling effort on, since it is likely that some parts of the system are more critical than others and thus more interesting to model. Other parts of the system can be described in a less detailed manner, in the extreme case with no behavioural description, only describing the execution time distribution.

Iterative reviewing of the model is necessary in order to avoid misunderstandings, due to e.g. different backgrounds and views of the system. If the system is large, this step can be tedious, several man months is realistic if the model engineer is unfamiliar to the system, according to our experiences. An experienced system architect can probably construct this structural model faster, but since such experts often are very busy, it is likely that the model developer is a less experienced engineer. In order to simplify the construction of the model, reverse-engineering tools such as Rigi [MK88, Rig] or Understand for C++ [Und] can be used. These tools parse the code and visualize the relations between classes and files.

13.3.2 Profiling and populating the model

In step 2 of the process described in Figure 13.1, the system is profiled in order to populate the model with execution times distributions and probabilities. The runtime behavior of the system is recorded with respect to task timing, i.e. when tasks start and finish, how the task preempt each other, their execution times and the usage of logical resources such as the number of messages in message queues. This requires the introduction of software probes, unless hardware probes are used [Sho02]. The problem with the latter approach is that it requires special-purpose hardware. The output from the profiling is a stream of time-stamped events, where each event represents the execution of a probe. Typically, the execution of a probe corresponds to a task-switch or an operation on a logical resource (e.g. message queue).

The measured data can be graphically visualized in order to increase the understandability of the system. In Figure 13.2, two such graphs are depicted, showing the temporal behavior of two tasks in a real system we have studied. These tasks are complex and time-critical, communicating with many parts of the system. Each graph shows the execution time and response time of task instances during a specific time interval. A task instance, a job, is one execution of a task. Each instance results in two dots in the graph corresponding to the task, the execution time (stars) and response time (squares), for the corresponding instance. In these graphs, we can observe a discrepancy around time 400. The execution- and response-times of these instances of the tasks

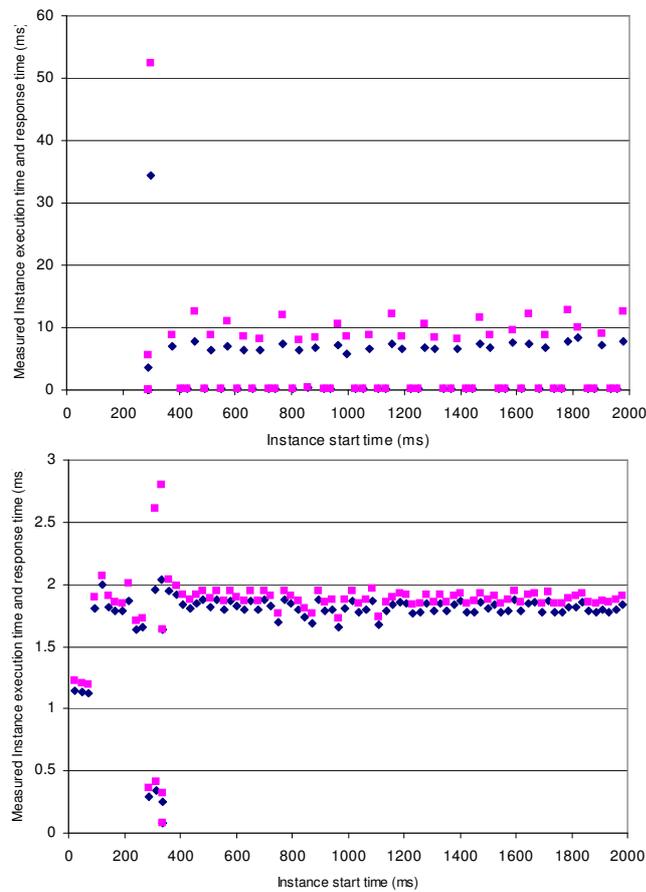


Figure 13.2: Visualization of observed execution and response times of two tasks

are very high, in comparison to the other instances in the graphs. This could be a coincidence, but most likely this is due to dependencies with other tasks, e.g. communication or a response to a global state change in the system. The temporal behaviors of these tasks are thus not independent. By introducing such dependencies in the model of the system, the model will become more accurate with respect to the implemented system. In this case, the cause could be that both tasks reacts to a global state change, caused by a command from a human operator or a message from another system. To update the model with this dependency, the states and the various reactions on state transitions must be modeled as well, either in detail or in a probabilistic manner.

13.3.3 Analysis of system properties

The analysis method decides what properties that can be analyzed and also affects the confidence assessment of the result. The main focus of the ART Framework is to support analysis of probabilistic system properties related to timing and usage of logical resources.

A deadline property is a requirement on a response time, either on a particular task or features involving several tasks, i.e. end-to-end response time. A deadline property can be formulated as hard, absolute requirements, or as a soft deadline. An example of a formulation of a soft deadline is that at least 90% of the response times of the instances of a task are less than a specified deadline and that no more than two consecutive instances exceeds that deadline. The PPL language, described in Section 13.4.2, can be used to formulate such properties. Other system properties related to timing can be e.g. if task X is preempting task Y, and in that case, how often?

Resource usage properties are those addressing limited logical resources of a system such as fixed size message buffers and dynamic memory allocation. When analyzing such properties, the typical concern is to avoid "running out" of the critical resource. An example is the invariant that a data buffer must never be empty. It could be a system requirement that the buffer always contains data, in order to avoid blocking the reading task.

The analysis of system properties is done by recording a trace of the dynamic behavior, either from a real system or from a simulation, and based on the recording the properties can be evaluated, e.g. probabilistic properties such as the soft deadline property described above can be evaluated by counting the number of instances of a task for which the property holds and comparing it to the total number of instances. Comparing to model checking, instead of an exhaustive search of all possible scenarios, the output from a running system or a simulator is analyzed. This gives a realistic picture of the system behavior, but the analysis result is not necessarily safe, as this does not explore all possible situations. However, we believe that this is a suitable solution for large and complex systems, as industrial software systems often are. Even though a simulation might miss some situations, it may still point out potential problems and thus guiding the developers making the right decisions. Formal methods such as model checking often has problems to manage the complexity of such systems, the state space becomes too large to search. Traditional methods for response time analysis are not suitable either [A. 03], as they are too pessimistic and make assumptions that often are violated by the real world system.

Due to the size of the recorded trace, tool support is necessary and therefore we have developed the PPL analysis tool, which evaluates prop-

erties formulated in the PPL language. The language is described in Section 13.4.2. The ART Framework contains graphical front-end tools, making PPL relatively user friendly. This is described in Section 13.6.

13.3.4 Validating the Model

The third step in the process is to validate the model, i.e. to determine whether or not the constructed model is a correct description of the system with respect to the system properties that the model is intended to describe. This is typically done by comparing observations of the system's behaviour with the predictions derived from the model.

This is however not trivial, since a direct comparison of the traces is not feasible. The comparison must be made on a higher level of abstraction, by comparing system properties. The properties are evaluated as described in Section 13.3.3, with respect to both the predictions based on the model and measurements of the real runtime system.

In order to facilitate future usage of the model, it should be easy to keep the model and the system consistent as the system evolves. The effort of adjusting the model to reflect the impact of a maintenance operation should not be similar to constructing the initial model, the necessary change in the model should be intuitive and similar to the change in the system. Therefore, it is necessary to verify that the model is robust with respect to typical, foreseen, changes of the system. Model validation and robustness is further discussed in Section 13.5, where we present a methodology for establishing the validity of a model.

13.3.5 Using the Model for Impact Analysis

Given that a model has been constructed and validated, it can be used for predicting the impact an maintenance operation will have on the runtime behavior of the system. The change is prototyped in the model and simulations of the updated model are made, generating execution traces. These are analyzed (as described in Section 13.3.3) in order to evaluate important system properties. This analysis can, in a very early phase, indicate if there are potential problems associated with the change that are related to timing and usage of logical resources. If this is the cases the designers should change their design in order to consume fewer resources. Since the change is not implemented yet, this means in practice to impose a resource budget on the implementer.

If the impact of the change is acceptable, and is implemented, the model should be updated in order to reflect the implementation. This corresponds to steps 1 to 3 in the process in Figure 13.1, i.e. updating the model structure, profiling and validation.

13.3.6 Regression Analysis and Trend Identification

Two other uses of the measured data are regression analysis and trend identification. The regression analysis is to analyze the current release of the system with respect to certain invariants. This is very close to regression testing, but instead of testing the functional behavior, timing and resource usage is analyzed.

The use of trend identification is to compare different releases of the system with respect to the system properties of interest to study how the evolution of the system affects them. There might be trends that will cause problems in future releases, e.g. execution times are increasing for each release as more features are added. If such a trend is allowed to continue, eventually overload situations will occur. If this is observed early, the appropriate measures can be taken before actual problem occurs. If a model has been developed, the Impact Analysis, described in Section 13.3.5, can be used in order to predict how an extrapolation of a trend will affect the system.

In order to use this in a development organization, measurements of new releases needs to be made. This would typically be made during the system testing. The only change would be that after each test case, an execution trace is stored. This trace is analyzed and compared with earlier releases, using a highly automated tool. Based on a set of rules, typically defined by system experts, the tool decides if there are alarming differences and in that case instructs the tester to notify a system expert. A tool supporting this is presented in Section 13.6.3.

13.4 ART-ML and PPL Languages

In this section we describe our approach of the modeling and analysis. First we will present the ART-ML modelling language, the notation which we use to construct probabilistic models. We will also present the Probabilistic Property Language, PPL, which is used to formulate the system properties that we wish to analyze.

13.4.1 The Modeling Language ART-ML

The ART-ML language describes a system as a set of tasks. Each ART-ML task consists of two parts, the attributes and the behavior. The attributes describe static properties of the tasks, such as name, scheduling priority and optional periodicity. The behavior part of a task is described in an imperative language, C extended with ART-ML primitives, and describes the temporal and to some extent the functional behavior.

```

TASK SENSOR
  TASK_TYPE: PERIODIC
  PERIOD: 2000 us
  PRIORITY: 1
BEHAVIOR:
  execute((0.40, 1000),(0.54, 1300), (0.06, 1400));
  sendMessage(CTRLDATAQ, MSG_A, NO_WAIT);
  chance(0.19){
    execute ((0.60, 200), (0.40, 230));
    sendMessage(CTRLCMDQ, MSG_B, FOREVER);
  }
END

```

Figure 13.3: A Typical ART-ML task

Models written in ART-ML are translated into C using a translator and then compiled and linked together with the ART-ML C-library. The resulting executable is a synthesis of the ART-ML model, i.e. a specialized simulator program for that model only. When this simulator is executed, it produces an output in the form of a trace.

Apart from tasks there are two other elements in the ART-ML language: Message Queues and Semaphores. They can be accessed from the behavior part of tasks through ART-ML functions, such as *sendMessage*. Next follows the basic elements of an ART-ML model explained in detail.

Task

A task consists of three parts, a name, a set of attributes and a behavioral description. The attributes are the scheduling priority and how the task is activated, one-shot, periodically (with or without offset) or sporadically. The behavior is described in C, extended with ART-ML primitives and routines.

Within the task behavior it is possible to call ART-ML routines that correspond to typical OS services, such as sending and receiving messages to message queues and semaphore operations. There is also a special statement for consuming execution time, *execute*. The *execute*-statement is used for modeling sections of code from the real system by their execution time only.

The execution time of the section is described using a discrete probability distribution, i.e. a list of possible execution times, where each execution time is associated with a certain probability of occurrence, see Figure 13.3. This allows probabilities to be used to describe variations in

the execution time. Depending on the selected level of abstraction when constructing the model, an execute-statement can represent a whole task or a smaller section of code.

When a task performs an execute it supplies a probability distribution as parameter. An execution time is chosen according to the distribution and the task is put into executing state for that amount of time. During this time, the task can be preempted by other tasks in the system. The task can not be preempted unless executing a kernel routine such as execute, send, sem_take, delay etc. The execution time and probabilities used in an execute statement is assumed to be from measurements (i.e. profiling as dicussed in Section 13.3.2) of the real system.

```

A()                // C = 1000 us
if (cond1)        // Ptrue = 0.60
{
    B()           // C = 300 us
    if (cond2)   // Ptrue = 0.10
    {
        C()      // C = 100 us
    }
}
}

```

Figure 13.4: The code corresponding to the first execute-statement in Figure 13.3

As mentioned, an execute-statement corresponds to a section of code in the real system. Figure 13.4 depicts the code that is modelled by the first execute-statement in Figure 13.3. There are three different functions/blocks of code, (A, B and C), for the sake of simplicity having constant execution times (C_A , C_B and C_C). They can be executed in three different ways, A, A-B or A-B-C depending on the conditions cond1 and cond2. These conditions are not known in the model, due to the necessary level of abstraction, but statistical data from observations of the system can be used to derive the different executions times and calculate the probabilities of the different cases. This execution time distribution is used to form the execute statement, as depicted in Figure 13.3 and Figure 13.4. This allows the model to accurately describe the execution time of the tasks in the system, without making it unnecessary complex.

Another ART-ML specific statement is the chance statement, non-deterministic selection with probability. It is a variant of the classic IF-statement, but instead of checking if the value of an expression non-zero, as in C, the argument expression is compared with a random number,

linearly distributed in the interval [1-100]. The probabilistic selection is evaluated as True if the value of the expression is less than the random number. In that case, just as in C, the next statement/block is executed. If the value is equal or larger than the random number, any else-statement/block is executed. The chance-statement is related to the execute-statement, but instead of probabilistic selection of execution times, chance allows probabilistic selection of different behaviours. A chance statement can be used for mimicking behaviors observed in measurements of the real system, where the exact cause is not included in the model, e.g. external stimuli. For instance, in the measurements of the system we can observe that when a certain task executes, it sometimes sends a message to a particular message queue. This can be modeled using a chance statement and a statistical probability derived from the measurements.

Message queue

An ART-ML message queue is a fixed size FIFO buffer, storing messages sent by tasks. A message contains only an integer. That should be sufficient since ART-ML message queues are only intended to model communication events and not to transfer large amounts of data. Other tasks can read the messages from the message queue, in a FIFO manner. A message is sent to a message queue using the ART-ML library routine *sendMessage* and reading a message is done by calling *recvMessage*.

To use a message queue in a model, it must be declared. The syntax for declaring a message queue is as follows:

```
MESSAGEQUEUE name size;
```

Semaphore

An ART-ML semaphore provides mutual exclusion between tasks and conforms to the concept of the well known binary semaphores proposed by Dijkstra in the 1960's. A semaphore is declared and identified with its name. A semaphore is locked using the *sem_wait* library routine (corresponding to djikstra's P, or wait) and released using *sem_post* routine (corresponding to djikstra's V, signal). The syntax for declaring a semaphore is:

```
SEMAPHORE name;
```

13.4.2 The Probabilistic Property Language

The purpose of the Probabilistic Property Language, PPL, is to allow formulation of queries on properties related to the temporal behavior

a system, such as response times and usage of logical resources. PPL allows formulation of probabilistic properties, e.g. soft deadlines such as “at least 99 % of task X should be completed within 1000 time units”.

Compared to temporal logics with probability and time, e.g. TPCTL [HJ94], there are obvious similarities. It is possible to use a temporal logic instead of PPL, but PPL is specially designed for expressing probabilistic properties of the temporal behavior of tasks which makes PPL queries more intuitive for software developers without previous experience of formal methods.

In this section the implemented version of PPL is presented using a set of examples.

The PPL query

A typical use of PPL is to check a deadline property of a task. Example 1 presents a PPL query that checks if all instances i of task A meets a deadline of 1000 time units with a probability of 1. A task instance is a particular execution of the task. A task instance is represented in PPL using its task identity, start time, finishing time and execution time.

Example 1:

$$P(A(i), A(i).response < 1000) = 1$$

The first parameter to the P operator is a quantifier specifying that the condition in the second argument should be checked for all instances i of the task A . This is different from the original version of PPL [], where the P-function did not accept any quantifier argument. It was discovered during the implementation of the PPL analysis tool that the original definition of PPL had ambiguous semantics when multiple tasks are referred in a query. The quantifier parameter is necessary in those cases to solve the ambiguity. The second parameter is the condition to check. In the example the condition specifies that the response time of the task A should be below 1000. The P operator returns the ratio of the instances in the execution trace for which the condition holds. If the P operator has a return value of 1, it means that the condition holds for all observed instances of the task, i.e. a probability of 1.

PPL allows checking probabilistic properties such as a soft deadline. For instance, a soft deadline requirement could be that at least 90% of the task instances should meet the deadline. An example of a soft deadline is presented in Example 2.

Example 2:

$$P(A(i), A(i).response < 1000) > 0.9$$

PPL has a data model allowing a query to refer to task instances in the execution trace using a combination of task name, instance index and property name, on the form “ $task(i).property$ ”. The data model

provides five properties for each task instances in the execution trace, which can be used in PPL queries. These are specified in Table 13.1.

Property	Description
start	The time when the instance started
end	The time when the instance finished
exec	The execution time of the instance
response	The response time of the instance
probeN	The value of probe N when the instance started

Table 13.1: The properties of a task instance in PPL

The “probeN” property of a task instance corresponds to the value of the generic probe “probeN” at the time the task instance is started. A generic probe may monitor any quantifiable property, but typically generic probes are used to monitor logical resources of different kinds, such as the current utilization of a buffer. Further, PPL contains a set of operators and function that allow conditions to be formulated on the data model. These operators are described in Table 13.2.

Relational operators	=, <, <=, >=, >	value op value -> bool
Logical connectives	and, or, not	bool op bool -> bool
Arithmetic operators	+, -, *, /, abs	value op value -> value
Statistical functions	max, min, avg, median	op(list) -> value
Index operator	X(i)	op(list, index) -> instance
Following operator	X(following(Y(i)))	op(list, list, index)-> instance

Table 13.2: The operators of PPL

PPL queries using the instance operator

The index operator is used to differentiate instances of the same task. One property that can be checked using the index operator is temporal separation, i.e. a property that specifies the minimum distance in time between two consecutive instances of a task. This is demonstrated by Example 3.

Example 3:

```
P(A(i), A(i+1).start - A(i).end >= 1000) = 1}
```

Another use of the instance operator is demonstrated in Example 4, which specifies that two consecutive instances must not violate the deadline of 1000 time units.

Example 4:

$$P(A(i), A(i).response > 1000 \text{ and } A(i+1).response > 1000) = 0$$

Expressing a requirement that e.g. 5 consecutive instances must not miss their deadline would result in a very large expression if the presented from the previous example is used. To simplify such queries it is possible to specify intervals rather than single integers in the index operator. Example 5 specifies that there must never be 5 consecutive task instances that violate the deadline of 1000 time units.

Example 5:

$$P(A(i), A(i+[1..4]).response > 1000) = 0$$

Queries using functions and unbounded variables

In order to relate adjacent instances of different tasks, the *following* function can be used. Example 6 shows a query checking if there are any situations where an instance of A and the following instance of B have execution times above 1100 time units and 1700 time units respectively.

Example 6:

$$P(A(i), A(i).exec > 1100 \text{ and } B(\text{following}(A(i))).exec > 1700) > 0$$

Moreover, PPL queries may contain an unbounded variable. For instance, by specifying the probability as an unbounded variable, the result of the query is the minimum/maximum value for which the condition holds. A query using an unbounded variable to evaluate the probability of meeting a deadline of 2000 time units is presented in Example 7.

Example 7:

$$P(A(i), A(i).response < 2000) = X$$

It is also possible to use unbounded variables inside the second parameter of the P-operator. A query evaluating the shortest deadline D that is met with a probability of at least 0.9 is presented in Example 8.

Example 8:

$$P(A(i), A(i).response < D) \geq 0.9$$

Statistical functions

As presented in Table 13.2, there is also a set of statistical functions that may be used to extract simple statistical measures of the different tasks. The statistical functions can be used as stand-alone queries as in Example 9.

Example 9:

```
avg(A.response)
```

```
median(A.exec)
```

```
max(A.exec)
```

The statistical functions can also be used instead of constant values inside the second parameter of the P-operator, as in Example 10.

Example 10:

```
P(A(i), A(i).resp > avg(A.resp)*2 ) = X
```

The above described PPL query returns the probability (X) of task A having a response time above a certain limit, which is specified as “two times the average response time”.

Queries on logical resources

PPL also allows queries on data from generic probes. A generic probe may monitor any quantifiable property of the system, but are typically used to monitor logical resources, such as the usage of a data buffer. In the current implementation, the generic probes are identified using a number. If the number of messages in a certain message queue is monitored using generic probe number 21, it is possible to formulate a PPL query checking that the message queue is never empty when taskX is activated as presented in Example 11.

Example 11:

```
P(taskX(i), taskX(i).probe21 > 0) = 1
```

It is also possible to specify conditions on a probe that are independent of what tasks that are to be executed, by replacing the name of the task with a wildcard character. This is demonstrated by Example 12. For such queries the probabilities are calculated differently, by summing the lengths of the time intervals where the condition holds and divide that with the length of the recording. The resulting value is thus the fraction of the total time in the recording where the condition holds. This is an approximation of the probability that the condition holds at an arbitrary point in time.

Example 12:

```
P(*, *.probe21 > 0) = 1
```

Tool Support

The PPL language is supported by two tools available within the framework. The Tracealyzer tool contains a PPL terminal, where it is possible to formulate and run queries with respect to an execution trace. This is the preferred tool for experimenting with PPL. The Property Evaluation Tool (PET) is a dedicated front-end application for PPL, allowing designed to run a batch of queries on two different execution traces and present the results side-by-side. The tools are presented in Section 13.6.

13.5 Validation of models

Validating a model is basically the activity of comparing the predictions from the model with observations of the real system. However, a direct comparison between traces from a simulation and traces from the real system is not feasible since the model is a probabilistic abstraction. Instead, we compare the model and the system based on a set of properties, comparison properties. The method presented in Section 13.3.3 is used in order to evaluate these comparison properties, with respect to both the predictions based on the model and measurements of the real runtime system. If the predicted values match the observed, we considered the model observable equivalent to the real system. A typical comparison property can be the average response time of a task. It is affected by many factors and characterizes the temporal behavior of the system. Selecting the correct comparison properties is important in order to get a valid comparison. Moreover, as many system properties as practically possible should be included in the set of comparison properties in order to get high confidence in the comparison. The selected system properties should not only be relevant, but also be of different types in order to compare a variety of aspects of a model. Other types of comparison properties could be related to e.g. the number of messages in message queues (min, max, average) or pattern in the task scheduling (interarrival times, precedence, preemption).

Even if the model gives accurate predictions, there is another issue to consider, the model robustness. If the model is not robust, the model might become invalid as the system evolves, even if the corresponding updates are made on the model. Typically, a too abstract model tends to be non-robust, since it might not model dependencies between tasks that allow the impact of a change to propagate. Hence, it may require adding more details to the model in order to keep it valid and consistent with the implementation. If a model is robust, it implies that the relevant behaviors and semantic relations are indeed captured by the model at an appropriate level of abstraction.

13.5.1 Observable Property Equivalence

We propose a measure of model validity named Observable Property Equivalence, where the model is compared to the real system with respect to a set of system properties of interest. However, since the relation of observable property equivalence is not transitive, this is not a true equivalence relation, only a measure of similarity.

A model and a corresponding system are observable property equivalent if they are equivalent with respect to a set of comparison properties, i.e. statistical measures of the observed temporal behavior. However, as discussed earlier, since the model is an abstraction of the system, it is necessary to allow a certain amount of tolerance in comparison.

In Definition 9 we formalize the observation of a system, x , that is either the real implemented system, executing on the real hardware, or a model of a system executed in a simulator. The resulting recording is a list of time-stamped events related to tasks-switches and operations on logical resources. The environment e specifies the configuration of the system and any external stimuli that effects the system behavior during the observation.

Definition 9 $R = Rec(x, e, d)$

The function Rec returns a recording, R , of the execution of x , in the environment e , with the duration d time units. R is a list of events, where each event contains a time-stamp, an event type and generic data, where the semantics are specific for each event type. \square

Definition 10 presents the function $Eval$, which evaluates a system property p with respect to the recording R .

Definition 10 $v = Eval(p, R)$

The function $Eval$ evaluates the property p with respect to the recording R . The result, v , is a decimal value. If the property p is a boolean expression, v is either 1 (true) or 0 (false). \square

Since a certain amount of tolerance is often necessary in the comparison, we introduce a function which expressing the tolerance allowed for a specific comparison property,

Definition 11 $t = Tol(p)$

The function Tol returns the maximum allowed difference between two evaluations of the same property p on two different recordings. The return value, t , is a decimal value. If the property p is of boolean type, the function returns a tolerance of 0. \square

Definition 12 presents the definition of observable property equivalence. If evaluations of all comparison properties with respect to the

model results in values sufficiently close to the values from the real system recording, the model and the system are observable property equivalent.

Definition 12 *Given that P is the set of comparison properties, M is a model of the system S , E_M is the environment model of M and E_S is the environment of the system S , iff*

$$\forall p \in P : Abs(Eval(p, Rec(M, E_M, d)) - Eval(p, Rec(S, E_S, d))) \leq Tol(p)$$

then $S \equiv M$, i.e. S and M are observable property equivalent with respect to P , in the specific environment. \square

Obviously, this relation of similarity relies heavily on the comparison properties and tolerances used. It is important to select a suitable set of comparison properties in order to compare as much as possible of the behavior of the model with the corresponding system.

13.5.2 Model Robustness

In this section we propose a method for ensuring the robustness of an ART-ML model. We refer to this activity as sensitivity analysis. To exemplify the importance of model robustness, imagine a system containing a binary semaphore protecting a shared resource. A timeout occurs if a task has been waiting on the semaphore for a certain predefined time. If the timeout occur, the execution time of the task is increased due to the error handling necessary. However, in all previous versions of the system, this timeout has never occurred. If the timeout is left out when constructing the model of the system the model still seems accurate since the timeout never occurs. However, as a result from changing the system, e.g. increasing the execution time of another task, the timeout will in some cases occur. Since the timeout was not included in the model, the system's behavior will diverge from the behavior predicted based on the model.

Our approach to sensitivity analysis is influenced by system identification. System identification is a technique used in the domain of control theory [Joh93]. By measuring and observing the input-output relationship between signals in the process a model can be determined in terms of a transfer function. Validating models based on the system identification approach is somewhat related to testing. Typically, output signals predicted using the model is compared with the output signals of the physical process. Hence, the model is regarded as correct if the analysis and the physical process generate approximately the same output, if fed with the same input.

Testing the model with different input signals and comparing the prediction with the signals produced by the actual system is fine if the

process is continuous in its nature. It is fair to assume that we can interpolate the behavior in between the tested signals. However, computer software is not continuous; they are discontinuous systems meaning that the behavior may change dramatically as a result of small changes in the system. A model of a software system can thus quickly become invalid when the system evolves, if the model is not robust with respect to typical changes. By analyzing the impact on the system caused by different changes, it is possible to determine if the model is sensitive to such changes, i.e. less robust.

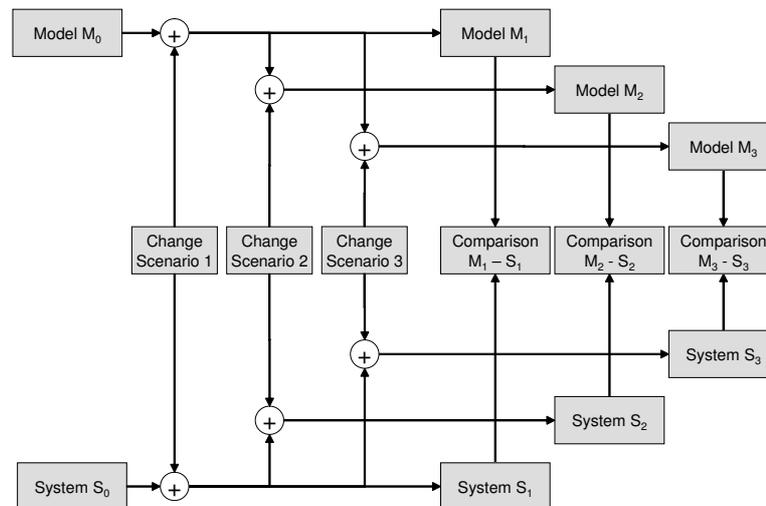


Figure 13.5: The Sensitivity Analysis

The robustness of a model can be analyzed using a sensitivity analysis. The basic idea is to test different probable alterations and verify that they affect the behavior predicted by the model in the same way as they affect the observed behavior of the system. Performing a sensitivity analysis is typically done after major changes of the model, in the validation step of the process. The process of performing sensitivity analysis is depicted in Figure 13.5. First a set of change scenarios has to be elicited. The change scenarios should be representative for the probable changes that the system may undergo. Typical examples of change scenarios are to change the execution times of a task, to introduce new types of messages on already existing communication channels or change the rate sending messages. The change scenario elicitation requires, just as developing scenarios for architectural analysis, experienced engineers that can perform educated guesses about relevant and probable changes.

The next step is to construct a set of system variants $S = (S_1, \dots, S_n)$ and a set of corresponding models $M = (M_1, \dots, M_n)$. The system vari-

ants in S are versions of the original system, S_0 , where n different changes have been made corresponding to the n different change scenarios. The model variants in M are constructed in a similar way, by introducing the corresponding changes in the initial model M_0 .

Note that these changes only need to reflect the impact on the temporal behavior and resource usage caused by the change scenarios, they do not have to be complete functional implementations. For instance, if the new feature is some sort of service offered by a server task to a client task, the implementation necessary would be to introduce two new messages on the communication channel, a request and a reply, and some minor changes in the tasks. When the request message is received by the server, it should consume a realistic amount of time (e.g. by executing an empty for-loop) and then send a reply message. The client should send requests at the appropriate times and wait for the reply. These changes are therefore easy to implement.

Each model variant is then compared with its corresponding system variant by investigating if they are equivalent as described in 13.5.1. If all variants are equivalent, including the original model and system, we say that the model is robust.

13.6 The Tools

This section presents three tools within the ART Framework, supporting the process described in Section 13.3.

- An ART-ML simulator, used to produce execution traces based on an ART-ML model.
- The Tracealyzer, a tool for visualizing the contents of an execution trace and also allow PPL queries to be executed on the data.
- The Property Evaluation Tool. A tool for analyzing and comparing execution traces, using a predefined set of PPL queries.

The Tracealyzer and Property Evaluation Tool are available for download at

<http://www.idt.mdh.se/~jxn01>

, they can be used freely for academic and other non-commercial purposes, but they are not open source. The only platform supported (so far) is Microsoft Windows.

13.6.1 The ART-ML Simulator

The ART-ML Simulator is basically a C library and an ART-ML to C translator application. Given an ART-ML model the translator outputs an ANSI C representation which can be compiled and linked together with the ART-ML library in order to produce an executable file containing a compiled version the ART-ML model together with configuration and logging functionality. It is possible to specify what seed (an integer number) to use for the random number generator, which makes it possible to reproduce simulations of non-deterministic models. This ART-ML simulator is approximately 5 times faster compared to first ART-ML simulator, which interpreted the model. The output of the simulator is a binary file readable by the Tracealyzer and the Property Evaluation Tool. If desired, the Tracealyzer translate the binary file to text format.

13.6.2 The Tracealyzer tool

The Tracealyser has two main features, visualization of an execution trace and a PPL terminal, a front-end for the PPL analysis tool. Figure 8 depicts the user interface of the Tracealyzer. The left part contains a window presenting a section of the execution trace. It is possible to navigate in the trace by using the mouse or the scrollbar. It is also possible to zoom in and out.

The leftmost part of the trace shows how the tasks execute, over time. The shaded boxes correspond to uninterrupted execution of a task. The point in time between two boxes corresponds to a task-switch. It is possible to select a task instance, by clicking on it. This will display information about the selected instance in the textbox named "Selected Task Instance" in the right part of the window. A selected task instance is marked with a red frame. This information presented includes the name of the task, the execution time and response time of the instance and the average execution and respose times for the task. If more statistics about the different tasks is desired, it is possible to generate a report, containing a lot of information about all tasks.

The upper right part of the window contains a lists, labeled *probes*, showing a list of the probes that have been found in the trace. Selecting one of the probes in the probe list will display the value of the probe over time, next to the tasks. Since a probe can monitor various things, the meaning of a certain probes is defined by the developer that puts the probes in the system. The probes shown in the screenshot of the Tracealyzer monitors the number of messages in different message queues. It is also possible to save a list of the task instances to a text file. This way, the data can be imported into e.g. Excel and visualized in other ways than the ones provided by the Tracealyser. For instance, Figure

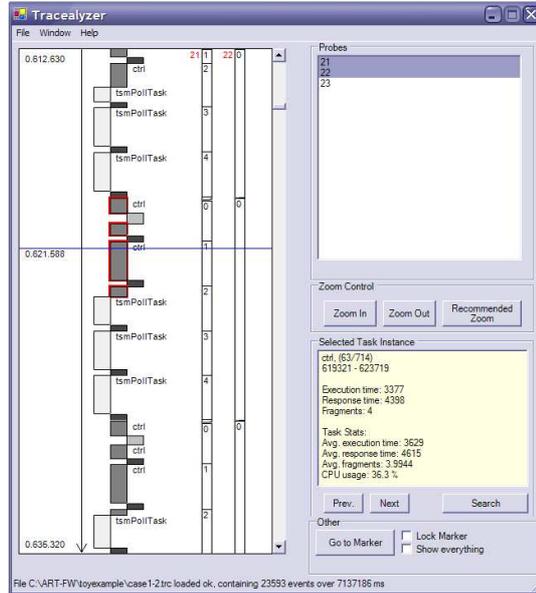


Figure 13.6: The Tracealyzer Tool

13.9 were created using this feature together with a common office application.

Apart from visualizing the data in an execution trace, the Tracealyzer also contains a PPL terminal. It is basically a front-end for the PPL analysis engine. The terminal contains two fields, one input where PPL queries can be typed and one output where the result is presented.

13.6.3 Property Evaluation Tool

The Property Evaluation Tool (PET) is a tool for comparing execution traces with respect to different system properties. These properties are formulated as PPL queries. The application has three uses: Impact Analysis, Regression Analysis and Model Validation. In the Impact Analysis case, execution traces from simulation of two slightly different models are compared. One of the models is considered "valid" and used as reference. The other model contains a prototype of a new feature or other changes. By comparing these traces, the impact of the new feature can be analyzed. Impact Analysis is discussed in Section 13.3.

In the Regression Analysis case, no data from simulation of models are used. Instead, two execution traces measured from two versions of the real systems are analyzed and compared, in order to identify trends and alarming differences, which might be a result of undesired behavior in the system. Regression Analysis is discussed in Section 13.3.6.

Properties	New System	Ref. System	Guard	Status
min(sensor() exec)	6.000000	6.000000		
avg(sensor() exec)	219.963934	220.322953		
max(sensor() exec)	431.000000	247.000000		
P(sensor(), sensor() exec < X) > 0.99	X = (240.00)	X = (240.00)		
min(sensor() resp)	6.000000	6.000000		
avg(sensor() resp)	219.963934	220.322953		
max(sensor() resp)	431.000000	247.000000		
P(sensor(), sensor() resp < X) > 0.99	X = (240.00)	X = (240.00)		
min(ctrl() exec)	7.000000	2919.000000		
avg(ctrl() exec)	3482.137279	3498.375333		
max(ctrl() exec)	4101.000000	4083.000000		
P(ctrl(), ctrl() exec < X) > 0.99	X = (4071.00)	X = (4083.00)		
min(ctrl() resp)	7.000000	3169.000000		
avg(ctrl() resp)	4582.667078	4614.934667		
max(ctrl() resp)	6990.000000	6999.000000		
P(ctrl(), ctrl() resp < X) > 0.99	X = (7637.00)	X = (7599.00)		

Figure 13.7: The Property Evaluation Tool

The application uses a set of rules, typically specified by a system expert, in order to judge what differences in system properties between system versions (execution traces) that are alarming and which one that can safely be ignored.

When used for model validation, a trace from simulation is compared with a trace measured from the real system. This way, it is possible to gain confidence in the model validity. This is discussed in Section 13.5.

The PET application is a front-end to the PPL analysis engine. The user of PET selects a configuration file, containing a predefined set of system properties, formulated as PPL queries. These properties are the point-of-view for the comparison. Then, the user selects two execution traces and starts the analysis and comparison process. The properties are evaluated with respect to the two traces and the results are presented. Finally, a property can be associated with a guard. A guard is a condition on the result from a property. Guards are used to formulate the rules on how much a property is allowed to before the user is notified.

13.7 An Industrial Case Study

We have applied the framework on an industrial control system. The system we have investigated is a robot control system, developed by ABB Robotics. It was initially designed in the beginning of the nineties. In essence, the controller is object-oriented and consists of approximately 2 500 000 LOC divided on 400-500 classes organized in 15 subsystems.

The system contains three nodes that are tightly connected, a main node that in essence generates the path to follow, the axis node, which controls each axis of the robot, and finally the I/O node, which interacts with external sensors and actuators. In this work we have studied a critical part with respect to control in the main node. The controller runs under the preemptive multitasking real-time operating system VxWorks from the company Wind River [Win].

Maintaining such a complex system requires careful analyses to be carried prior to adding new functions or redesigning parts of the system not to introduce unnecessary complexity and thereby increasing both the development and maintenance cost.

13.7.1 The model

We have modeled some critical tasks for the concrete robot system in the main computer (see Figure 13.8). The axis computer periodically sends requests to the main computer and expects a reply in the form of motor references within a certain time. There are three tasks in the main computer that are responsible for generating these motor references: A, B, and C. The tasks B and C have high priority, are periodic, and runs frequently. A executes mostly in the beginning of each robot movement and has lower priority. The final processing of the motor references is performed by task C. Task C sends the references to the axis node. Moreover, task C is dependent on data produced by task B. If the queue between them becomes empty, task C cannot deliver any references to the axis node. This state is considered as a critical system state and the robot halts. Task A sends data to task B when a movement of the robot is requested. If the queue between task A and task B gets empty, the robot movement stops. In this state, task B sends default references to task C. The complete case study is presented in [AN02]. All comments have been removed and variable names have been changed for business secrecy reasons. The model is not complete with respect to all components in the system.

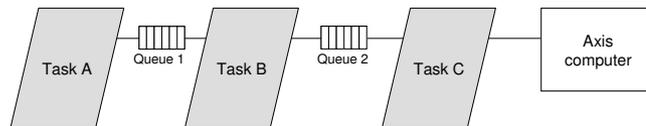


Figure 13.8: The task structure of the critical control part of the system

13.7.2 The results

The model we made is quite an abstraction of the existing system. There were approximately 60 tasks in the system which was reduced to six in the model. This level of abstraction was selected since there were three tasks of particular interest which was modeled in detail. The rest of the tasks were modeled with respect to CPU utilization only, no behavior was described. The axis computer was modeled as a task with zero execution time. The 2.500 KLOC in the existing implementation was reduced to 200 LOC in the model.

A more detailed model would not only represent a more accurate view of the system, it will also prune the state-space which the simulator has to consider. For instance, by introducing some dependency between tasks, the size of the statespace is reduced. This allows the simulator to explore a larger fraction of the possible behaviours of the system during a given the amount of simulation time and thus improving the confidence in the simulation result.

Despite the course-grained model, the result when comparing response times produced by the simulator and the response times measured on the system is quite good. In Figure 13.9, the response times from the simulation and the real system are plotted. The resemblance is obvious. However, at the time of this case study, no tools that would allow a more formal comparison were available, such as PPL and the supporting tools presented in Section 13.6.

13.7.3 Validation results

The results from the case study indicates that we have made one valid model out of many which may be valid for the system in its current state. However, we can not assume the model to be completely correct. In order to validate the model and establish confidence in the model we must develop a set of change scenarios as described in Section 13.5. Our initial list of scenarios was:

- add/remove tasks to the system,
- add/remove functional behavior in an existing task,
- change the behavior of existing functionality, i.e. changing execution times,
- change the priority of existing tasks,
- change message queue sizes,
- add shared resources,

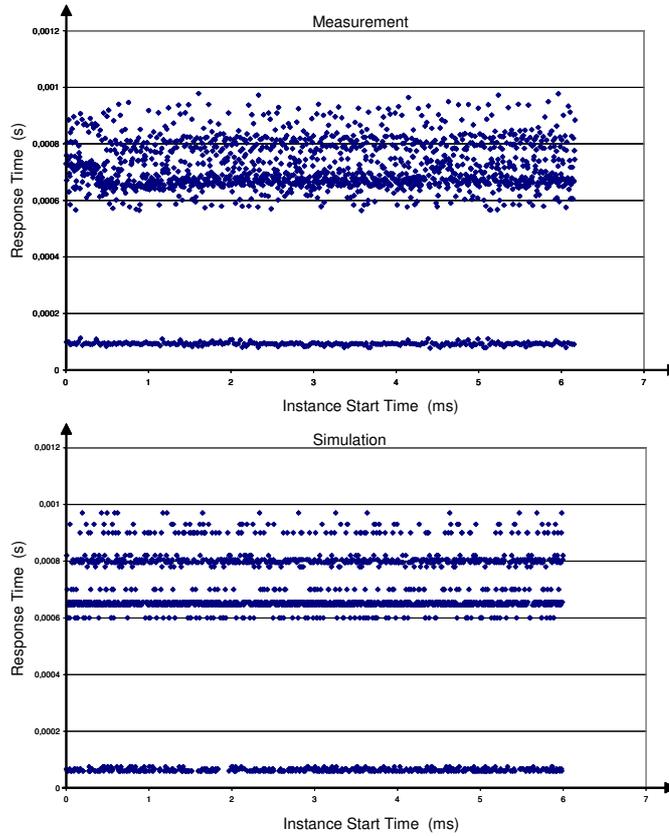


Figure 13.9: Response time distribution of a task, simulation and measurement on real system.

- change the period time of a task, and
- change the triggering condition of a task.

After interviewing engineers at ABB Robotics, the list was reduced to only include the most likely changes: add/remove tasks to the system, add/remove functional behavior in an existing task, change the behavior of existing functionality, and change the priority of existing tasks.

We developed four different cases out of the scenarios:

- Case 0: No change at all,
- Case 1: Add a new task called dummy with a short, varying execution time and low priority,
- Case 2: Raise the priority of the dummy task drastically, and

- Case 3: Increase the period time for the dummy task and extend its execution time

We model changes in a task's functional behavior by changing its execution time.

In general, by observing the results we see that the model indeed capture the temporal behavior of the system quite well. The simulations follow the measured system over the change scenarios. However, there are small differences in execution times between the model and the system. Consequently, we need to tune the execution time distributions in the model as it is too coarse grained. Moreover, we had to model yet another composed task since the priority of the dummy task is within the range of the low priority composed task.

The complete result from the validation is provided in appendix E in [Wal03].

13.8 Conclusions and Future work

In this paper we have presented the ART Framework; the general ideas, the languages ART-ML and PPL, the three tools within the framework and an approach for validating ART-ML models. We have presented a process for use the ART Framework for impact analysis and we have also presented how the framework can be used for regression analysis of timing properties. We believe that this approach is very useful for analysing properties of complex real-time systems, related to timing and resource utilization.

The next step in this work is to perform an industrial case study evaluating the benefit of performing regression analysis, as described in 13.3.6, and a continuation of the case study presented in 13.7, on modeling and analysis, using a more advanced model and the (new) tools presented in 13.6.

One problem with the approach described in this paper is the error-prone work of constructing the model. Instead of manually constructing the whole structural model, tools could be developed that mechanically generate at least parts of it, based on either a static analysis of the code, dynamic analysis of the runtime behavior or a hybrid approach. This is also part of our future work.

Bibliography

- [A. 03] A. Wall and J. Andersson and C. Norström. Probabilistic simulation-based analysis of complex real-time systems. In *Proceedings of the 6th IEEE International Symposium on Object-oriented Real-time distributed Computing*, 2003.

- [ABRW94] N.C. Audsley, A. Burns, M.F. Richardson, and A.J. Wellings. STRESS: A Simulator for Hard Real-Time Systems. *Software-Practive and Experience*, 24(6):534,564, 1994.
- [AN02] J. Andersson and J. Neander. Timing Analysis of a Robot Controller, 2002.
- [BDL⁺01] Gerd Behrmann, Alexandre David, Kim G. Larsen, Oliver Mller, Paul Pettersson, and Wang Yi. UPPAAL - present and future. In *Proc. of 40th IEEE Conference on Decision and Control*. IEEE Computer Society Press, 2001.
- [BLL⁺95] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real-Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer-Verlag, October 1995.
- [HJ94] H. Hansson and B. Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
- [Joh93] R. Johansson. *System Modeling Identification*. Prentice-Hall, 1993. ISBN 0-13-482308-7.
- [LN03] A. Leulseged and N. Nissanke. Probabilistic Analysis of Multi-processor Scheduling of Tasks with Uncertain Parameters. In *Proceedings of the 9th Conferance on Real-Time and Embedded Computing Systems and Applications*, pages 317–336, 2003.
- [MEP01] S. Manolache, P. Eles, and Z. Peng. Memory and Time-efficient Schedulability Analysis of Task Sets with Stochastic Execution Time. In *Proceedings of the 13nd Euromicro Conference on Real-Time Systems*. Department of Computer and Information Science, Linköping University, Sweden, 2001.
- [MK88] H.A. Muller and K. Klashinsky. Rigi: a system for programming-in-the-large. In *Proceedings of the 10th International Conference on Software Engineering*, 1988.
- [Rig] Rigi Group Home Page.
<http://www.rigi.csc.uvic.ca/index.html>.

- [Sch] W. Schutz. On the testability of distributed real-time systems. In *Proceedings of the 10th Tenth Symposium on Reliable Distributed Systems*, pages 52–61. IEEE.
- [Sho02] Mohammed El Shobaki. On-chip monitoring of single- and multiprocessor hardware real-time operating systems. In *8th International Conference on Real-Time Computing Systems and Applications*. IEEE, March 2002.
- [SL96] M.F. Storch and J.W.–S. Liu. DRTSS: a simulation framework for complex real-time systems. In *Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium (RTAS '96)*. Dept. of Comput. Sci., Illinois Univ., Urbana, IL, USA, 1996.
- [Tri] Tripac: RAPID Sim High-Performance Simulation of Real-Time Systems. <http://www.tripac.com>.
- [TSHP03] H. Thane, D. Sundmark, J. Huselius, and A. Pettersson. Replay Debugging of Real-Time systems using Time Machines. In *Proceedings of the International Parallel and Distributed Processing Symposium, 2003*.
- [Und] Scientific Toolworks, Inc: Maintenance, Understanding, Metrics and Documentation Tools for Ada, C, C++, Java, and FORTRAN. <http://www.scitools.com/>.
- [vDHK⁺04] A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva. Symphony: View-Driven Software Architecture Reconstruction. In *Proceedings of the IEEE/IFIP Working Conference on Software Architecture (WICSA)*. IEEE Computer Society, 2004.
- [Wal03] A. Wall. *Architectural Modeling and Analysis of Complex Real-Time Systems*. PhD thesis, Mälardalen University, 2003. ISBN 91-88834-05-0.
- [Win] Wind River Website. <http://www.windriver.com>.
- [YGS⁺04] H. Yan, D. Garlan, B. Schmerl, J. Aldrich, and R. Kazman. Discotech: A system for Discovering Architectures from Running Systems. In *Proceedings of the 26th International Conference on Software Engineering*, 2004.

Part III

Real-Time Communication

Chapter 14

Introduction

By **Magnus Jonsson**
School of Information Science
Computer and Electrical Engineering
Halmstad University
Email: Magnus.Jonsson@ide.hh.se

14.1 Introduction

More and more real-time systems are complex and distributed systems consisting of many sub-systems that must cooperate. In other words, those sub-systems must communicate with each other and to do this, still fulfilling the overall system specification, real-time communication must be supported. Moreover, real-time communication is also becoming widely needed in networks like the Internet, spanning over large distances. In for example Internet, the support for different traffic classes is often described as supporting different QoS (Quality of Service) levels and does normally include some kind of specification to meet real-time demands.

Real-time communication often relies on some kind of scheduling like EDF (Earliest Deadline First), but there are some important differences compared to standard single-processor scheduling. First, the transmission of a packet is, in most cases, non-preemptive. The ongoing transmission, and possibly also some of the already queued messages, can then not be interrupted when a more important message (e.g., with a shorter relative deadline) arrives or is generated. Moreover, a network, instead of a single link, makes the situation much more complex and one must consider things like medium access method, topology, multiple users on multiple nodes, non-deterministic access delay etc.

Regarding approaches to handle real-time communication, one is to schedule the traffic according to, e.g., relative deadline (also called laxity or delay bound) or priority. In other words, dynamic (on-line) communication scheduling is used. Without any further analysis this only gives a best-effort service, i.e., no real-time guarantees are given. Secondly, for networks with a deterministic scheduling/behavior, a schedability analysis can be made through static (off-line) scheduling to ensure a timely behavior. A third approach is to have a semi-static scheduling where admission control with a schedability analysis is used when a new logical channel for real-time traffic is requested. The new channel is accepted only if the network can guarantee the specified QoS, not violating the granted QoS for existing connections. As an example, any of these three approaches can be used in a network with priority scheduling:

- Approach 1: Nothing more than prioritizing important traffic (no guarantees).
- Approach 2: Analyze the network behavior at system design, given the behavior of the communicating processors/processes.
- Approach 3: Admission control (like Approach 2 but analysis during run-time, each time the traffic characteristics changes).

Below, we will give an overview of real-time communication characteristics and representative methods to give real-time communication support in different kinds of networks. After that, the appended papers are shortly introduced.

14.2 Application and traffic characteristics

Real-time systems and applications are often divided into classes depending on how strict timing requirements they have. One example is the division into critical, essential, and nonessential timing requirements, while another is the division into hard and soft real-time requirements. It is, however, not always the case that the real-time communication for an application can inherit the same classification as the application in whole. As an example, IP telephony can be regarded as a soft real-time application since it still might be possible to understand what is said even though some data is lost or not delivered in time. Despite this, a single message is useless if not delivered in time and might therefore be considered as having a hard deadline. A compromise is to state a probability that a message is delivered in time.

Since the classification into soft and hard real-time communication is somewhat unclear, real-time communication support is normally described more in terms of the service offered. For example, the communication support for a certain traffic class with real-time requirements

might have both a guaranteed (minimum) throughput and a bounded delay. Some common such parameters to describe traffic requirements and QoS levels together with more general traffic classifications are:

- The traffic can be of either aperiodic (asynchronous) or periodic (synchronous) nature.
- A relative deadline (delay bound) might be stated. In case of periodic traffic it can be shorter, equal, or longer than the period which, depending on the specific network, can have significant influence on the real-time methods and analysis.
- Rate characteristics such as minimum inter-arrival rate and maximum message-length are parameters describing the desired throughput in a more precise manner.
- The delay jitter is important to be bounded for some applications and is defined as the difference between the minimum and the maximum end-to-end delay.
- The reliability in terms of, e.g., loss rate is used when describing services without hundred percent guaranteed timely delivery.

Sometimes, the traffic characteristics or network requirements are in an interval like [required, desired] or by both minimum and average values. A good example of this is when having multimedia traffic with dynamically changing throughput requirements due to variable bit-rate compression.

The linear bounded arrival process model is sometimes used to characterize and bound periodic but bursty traffic, i.e., traffic that sometimes comes in large bunches of packets instead of always being regularly spaced in time. The intended amount of data from the source is specified as:

- S = maximum packet size
- R = maximum packet rate
- W = maximum work ahead

where W allows for short violations of R (bursts). In any interval of duration t , at most $W + t \cdot R$ packets may arrive (be generated).

Different applications can have fundamentally different traffic characteristics and real-time demands. For example, automotive applications like steer-by-wire have very strict demands where safety-oriented designs with high reliability must be considered. Methods like repetitive transmissions and/or fault-tolerant network architectures with hardware redundancy must be used to ensure correct and timely delivery. In automation industry, the traffic is often of periodic nature with a master-slave communication pattern where one or several master nodes periodically retrieve sensor values from slave nodes.

Data and telecommunication equipment like large distributed routers in the Internet are characterized by having requirements on short delays to not add too much to the end-to-end delay and not contribute too much to large queuing requirements. Regarding a specific Internet application, IP telephony should, although it is not always technically possible, have a maximum end-to-end delay of 150 ms to avoid that a half-duplex session is entered where one talks and the other listens [Hassan et al. 2000]. Interactive video applications have higher throughput requirements but the same delay requirements as IP telephony [Wolf and R. Steinmetz 1997]. For non-interactive multimedia applications, longer delays might be acceptable but the jitter can be a problem. Playback buffers can be used to cope with the jitter but rather large buffers might be needed if the network not includes mechanisms to reduce the jitter. Other things worth mentioning about multimedia communication are that: *(i)* related streams (e.g., sound and video) must be tightly synchronized (less than 80 ms skew), *(ii)* there is normally no time for retransmissions but 100 % reliability are not often required, and *(iii)* support for multicast is often desired. When evaluating networks regarding their capability of transporting multimedia traffic, traced traffic (e.g., from an MPEG compressed movie) is often used.

The area of sensor networks is currently getting a lot of attention in the research community and real-time communication is one sub-area [Stankovic et al. 2003]. New protocols and methods to handle real-time communication in dynamic wireless low-power constrained networks must be developed. Other application areas where real-time communication is important include avionics, signal processing applications, process control, and remote surgery.

14.3 Real-time communication services

Examples of real-time user services that can be offered, to the application or the programmer, by the network are: RTVC (Real Time Virtual Channel) [Ferrari and Verma 1990], guarantee-seeking messages [Arvind et al. 1991], and different kinds of best-effort and non-real-time services. If the service offered is deterministic, it is normally offering both a guaranteed minimum throughput and a bounded end-to-end delay. If the service offered is probabilistic, it can still be the case that a guarantee is offered but only as a guarantee to meet the specified QoS level at a certain probability. Some networks have inherent heterogeneous real-time support, i.e., having support for several traffic classes with rather different characteristics and real-time demands [Bergenheim and Jonsson 2003]. Using active networking, the QoS support can even be adapted during run-time by having service modules dynamically loaded

into the network equipment [Metzler et al. 1999]. A related topic is about approaches to support real-time traffic over heterogeneous networks, i.e., networks composed of rather different sub-networks. One example is to support connection-oriented real-time communication in an FDDI-ATM-FDDI heterogeneous network [Chen et al. 1997]. Another example is the proposal of a general flexible TDMA (Time Division Multiple Access) approach that can be used to support real-time communication over a heterogeneous network with both WLAN (Wireless Local Area Network) and fieldbus technology (CAN; Controller Area Network) [Mock and Nett 1999], while a third example investigates the use of wireless communication to connect several PROFIBUS segments by forwarding and insertion of extra idle-time between messages [Alves et al. 2002]. It is not only pure data delivery services for communication between two nodes that can be enhanced with real-time support. One example is the support of real-time services for special communication patterns like many-to-many [Fan et al. 2004].

14.4 Local area networks

Traditionally, a LAN (Local Area Network) implies a shared-medium network where a MAC (Medium Access Control) protocol is used to control which node that should have the possibility to send in each instance. With shared-medium networks, we mainly refer to bus networks and ring networks. A MAC protocol with a deterministic behavior is needed if guaranteed real-time services shall be supported. For example, CSMA/CD (Carrier Sense Multiple Access, Collision Detect) used in Ethernet is not deterministic. Several of the deterministic MAC protocols can be used together with a schedability analysis to calculate the on-line performance of predefined communication patterns off-line.

Several MAC protocols and extensions to basic MAC protocols to support real-time communication over bus networks have been proposed. TDMA (Time Division Multiple Access) is one example where the access to the medium is divided into time-slots. In static TDMA, each node normally has the access to the bus in one time-slot per cycle but many variants of TDMA exist where, e.g., time-slots are dynamically booked. As long as not only relying on self-synchronization on transmitted frames (packets), clock-synchronization is very important in bus networks. Mars, proposed by Kopetz et al., is one example of a distributed real-time system where a TDMA based network is included [Kopetz et al. 1989].

Window protocols can also be used to obtain real-time functionality over bus networks, which the following example shows. Assume that the access to the medium is divided into windows (time intervals) and

that a message has a priority/node/process specific value. Transmission is then only allowed if the value is in the window (and the medium is free). On collision, the window is split into several windows to reduce the risk of having two nodes in the same window. When having a deadline controlled window protocol, the basic idea is to have a value set according to the relative deadline and to let windows for earlier deadlines come earlier [Zhao et al. 1990]. Using VTCSMA (Virtual Time CSMA) [Molle and Kleinrock 1985] instead, all nodes are clock synchronized and a (virtual) time to start transmission is calculated depending on, e.g., deadline [Zhao and Ramamritham 1987]. Collisions can still appear and are treated with random delays etc. In other words, no deadline guarantees can be given and no schedability analysis can be made.

The IEEE 802.3 network (daily called Ethernet) has no inherent support for real-time communication but several kinds of extensions and modifications have been proposed. In RETHER [Venkatramani and Chiueh 1994], a software implemented token-based protocol is added on top of the normal MAC protocol. To reduce overhead, the token-based protocol is activated only when there is real-time traffic to be sent. Other researchers have proposed to control the amount of outgoing traffic from a node in an adaptive way to increase the overall possibilities in the network to meet the real-time demands [Kweon et al. 2000] [Carpenzano et al. 2002].

Token Ring (IEEE 802.5) is a ring network with a priority-based arbitration supporting eight priority levels. A single token is circulated in the ring and, as long as it is free, a node can grab the token to get permission to send. If a frame passes without the token (i.e., a normal data frame), a node can try to reserve the token for the next round by setting the special reservation field to the priority value of its message. This is only allowed if the priority field is not already having the same or a higher value. This reservation mechanism gives the node with the highest-priority message the possibility to send in the next round.

FDDI (Fiber Distributed Data Interface) is an optical fiber ring network, which relies on the same principles as in Token Bus (IEEE 802.4; see [Montuschi et al. 1992] for a discussion on timing), namely the Timed-Token protocol (see [Malcolm and Zhao 1994] for real-time aspects of the Timed-Token protocol). All nodes know the value of the constant $TTRT$ (Target Token Rotation Time), while the TRT (Token Rotation Time) is measured by each node for each turn the token travels. A node is allowed to transmit asynchronous traffic if $TTRT - TRT > 0$. In addition to asynchronous traffic, a predefined part of the bandwidth can be allocated for synchronous data for which a node is guaranteed to be allowed to send when it gets the token.

Switched Ethernet is a technology that is frequently used in LANs today. An additional Ethernet standard supported by many switches does even support priority queuing with between two and eight priority levels. An extra 3-bit header field is added to the frames, while the switches have priority queues. EtheReal was an early work on switched real-time Ethernet but was only throughput oriented (i.e., no explicit treatment of real-time demands) [Varadarajan and Chiueh 1998]. Later work on switched real-time Ethernet has been much concentrated on the fact that the collision probability can be totally eliminated by only including point-to-point links and network interface cards and switches that support full-duplex transmission [Alves et al. 2000] [Hoang et al. 2002].

14.5 Fieldbus networks

Fieldbus networks have their main origin in the demand to reduce cabling cost in, e.g., vehicles [Leen et al.] and automation industry. As many of the targeted applications can be classified as real-time systems, a lot of attention has been paid to real-time communication in fieldbus networks. Since typical data consists of sensor values or other short data, the frame length is normally rather short in fieldbus networks.

CAN (Controller Area Network) is one of the fieldbuses that have got a lot of attention. Each frame transports eight bytes of data at the most. A node can listen to arbitrary “addresses” and also to many “addresses” simultaneously. Several nodes can even listen to the same address (multicast). A special detail is that the “address” also acts as a priority level. The MAC protocol works in the following way. When sending over the bus, a dominant bit overwrites a recessive bit. The sender, starting with the most significant bit of the “address”, checks if the received bit is identical with the transmitted bit. It stops transmission if the recessive bit is overwritten, i.e., because a collision with a higher-priority message has occurred. An already begun transmission (more than just “address”) is never interrupted. This MAC protocol resolves collisions in a way so the highest priority message in the whole network is ensured to be transmitted. However, the bit-by-bit arbitration mechanism limits the maximum distance depending on the bit rate used. For a 1 Mbit/s network, the maximum farthest distance between two nodes is set to 100 m.

Analyses of the real-time performance of CAN have been published in several papers (see, e.g., [Tindell et al. 1994]). Moreover, several extensions have been proposed, e.g., to support deadline scheduling by dynamically updating the priority value according to the deadline [Zuberi and Shin 1995] or by using server-based scheduling [Nolte et al.

2003]. TTCAN (Time-Triggered CAN) is a network in which the original CAN protocol is extended with a session layer on top of the CAN link layer to support time-triggered communication [Leen and Heffernan 2002]. One node (although several can exist for fault-tolerance) is the time master in the network and triggers a new cycle regularly by sending a reference message. FTT-CAN (Flexible Time-Triggered CAN) is similar to TTCAN but allows for traffic changes during runtime [Ferreira et al. 2002].

The master-slave arrangement where the slaves only transmit upon request by the master is an approach to make it easier to analyze the network and achieve a time-deterministic behavior. This approach is used in the MIL-STD-1553, developed for avionic applications in the early 1970's [Schuh 1988]. The network has been used in, e.g., JAS, F-15, and space shuttles. One or several buses (for redundancy) can be used where only the bus controller is allowed to initiate communication. The bus controller can, e.g., periodically ask remote terminals for information.

TTP [Kopetz and Grünsteidl 1994] and FlexRay [FlexRay] are two fieldbuses developed for safety critical applications like steer-by-wire. The TTP protocol (latterly called TTP/C) relies on static TDMA schedules where different nodes can have different amounts of scheduled transmission per TDMA round. Even though TTP/C is based on static scheduling, rapid mode-changes between different predefined schedules are supported. A bigger picture of how to use a TTP/C network, combined with segments using the simpler TTP/A protocol [Kopetz et al. 2000], in the Time Triggered Architecture (TTA) is presented in [Kopetz and Bauer 2003]. Both FlexRay and TTP/C support network replication for redundancy and a choice between a bus topology and a star topology [Rushby 2001]. FlexRay, however, supports both time-triggered and event-triggered communication by, at design time, defining two separate portions of the time cycle.

A lot of other work considering real-time aspects of fieldbus networks has been published, of which a few examples follows. The Timed-Token protocol variant used in Profibus is analyzed in [Tovar and Vasques 1999], while offline scheduling of sporadic traffic in FIP networks is analyzed in [Pedro and Burns 1997].

14.6 Packet-switched networks

In a typical packet-switched network, each packet traverses a number of physical links towards the final destination. After each hop, the packet is stored (queued) in a switch (or router). The choice of queuing architecture, traffic handling etc is essential for the QoS characteristics. Service disciplines for real-time communication in packet-switched net-

works can be classified into the following three classes according to the service they offer [Zhang 1995]. Guaranteed deterministic services offer a hundred percent guarantee of the stated QoS level by having an admission control mechanism to verify that the specified requirements can be met. A schedulability analysis is run, when a new logical channel/RTVC is requested, for both existing and the new logical connections. Each switch on the path must guarantee the specified QoS, not violating QoS for existing connections. For this to be able, each source of a logical connection must obey to the predefined characteristics of its traffic generation. Guaranteed statistical services are similar to guaranteed deterministic services but have a non-zero loss probability. In other words, there is a small probability that delay and throughput bounds are violated or that buffer-overflow occurs. Having statistical bounds instead of counting on worst-case scenarios gives higher possible utilization for QoS traffic. For the third class, predicted services, guarantees are not given at all. Instead, a measurement-based admission control is used to predict the performance. For example, the current network load can be measured.

As stated, it is important that a source obeys to its specified maximum traffic generation. Because jitter can be accumulated for each hop, and therefore make a worst-case analysis very pessimistic, it is good if the switches also implement some policing mechanism to regulate the injection rate. The Leaky Bucket policing algorithm restricts traffic to follow the Linear Bounded Arrival Process Model (see above). Leaky bucket allows for an average rate of r and burst sizes of b by having a “bucket” filled with r tokens per second (see Figure 14.1). The bucket holds up to b tokens. Packets are only sent as long as there are tokens in the bucket. One token is removed for each packet

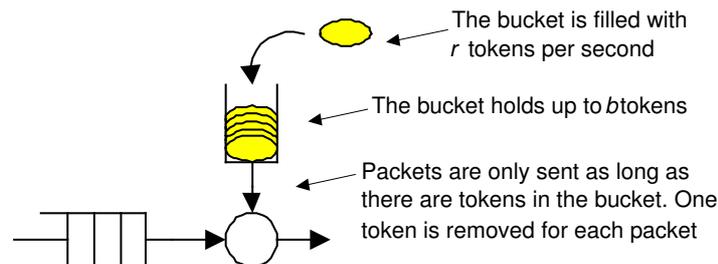


Figure 14.1: The leaky bucket algorithm.

in the bucket and one token is removed for each packet. If new tokens are generated when there are b tokens in the bucket, the tokens are discarded.

Below, we will give a few examples of guaranteed deterministic services, which can be divided into work-conserving service disciplines and

non-work-conserving service disciplines. Using a work-conserving service discipline, transmission will occur as long as there are packets eligible for transmission. This maintains a good utilization. Non-work-conserving service disciplines instead, can be good with regard to jitter and hence the size of jitter eliminating buffers. WFQ (Weighted Fair Queuing) is one of the most well-known work-conserving service disciplines [Demers et al. 1989]. For each physical channel, each traffic class (or logical connection) has a FIFO (First In First Out) queue i , $1 \leq i \leq N$, with weight w_i , assuming N traffic classes (see Figure 14.2). Assuming non-empty queues when served, WFQ emulates an ideal system where packets are taken from all queues simultaneously in a bit-by-bit fashion, each queue being served at a rate equal to the fraction

$$\frac{w_i}{\sum_{j=1}^N w_j}$$

of the line rate. The earliest calculated finish times for a packet to be served in the ideal system decides the next queue to serve. A delay bound analysis for WFQ together with the leaky bucket algorithm used in each switch in the network is presented in [Parekh and Gallager 1994].

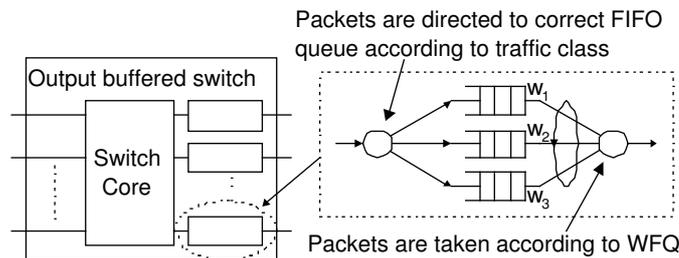


Figure 14.2: The WFQ service discipline exemplified in an output buffered switch.

Delay-EDD (Delay-Earliest-Due-Date) is another well-known work-conserving service discipline [Ferrari and Verma 1990]. EDF is used to sort packets in the switches and a deadline is set to each packet arriving at a switch. The deadline is set to the expected arrival time plus the delay bound for the current hop. By adding all delay bounds together for the path, an end-to-end delay bound is obtained. Important improvements of Delay-EDD to make it more generally applicable have been published [Kandlur et al. 1991] [Zheng and Shin 1994].

Jitter-EDD (Jitter-Earliest-Due-Date) [Verma et al. 1991] is an example of a non-work-conserving service discipline and is similar to Delay-EDD but with the following functional differences. Each switch on the

route stamps each outgoing packet with the difference between its deadline and the actual transmission time, T_{diff} . The next switch on the route holds the packet for a period of T_{diff} before it is made eligible to be scheduled for further transmission.

14.7 Internet

As the Internet consists of a very wide variety of different networks coupled together, and with different operators, it is quite obvious that it is hard to give any real-time guarantees. If not just considering well-defined parts of the Internet were, for example, all routers have sophisticated QoS support and no shared-medium Ethernet networks are used, it is even impossible to give any guarantees. The fact that the IP network protocol is connectionless makes the situation even harder since routing decisions might change, and hence the possibilities for QoS, during a session. Anyhow, there are mechanisms for QoS over the Internet [Xiao and Ni 1999] [El-Gendy et al. 2003].

Using the RSVP signaling protocol in combination with IntServ (Integrated Services) gives the opportunity of having admission control for separate flows. Both a guaranteed service, with delay bound for specified traffic agreement, and a controlled load service, that simulates a lightly loaded network (a kind of statistical non-bounded service), are supported. The problem is, however, scalability. Huge amounts of IP datagrams (packets) can pass the core Internet routers and each datagram should be inspected to discover which flow it belongs to and then handled according to specified QoS. Variable-sized headers make life even worse because simple offsets cannot be used. DiffServ (Differentiated Services) was developed with the scalability as the main driving force. With DiffServ, there is no treatment of separate flows. Instead, aggregation is introduced and the eight-bit ToS field (Type of Service) is used to distinguish among different traffic classes. All flows for a specific service class are treated in the same way (can vary with routing decision).

14.8 Networks for high-performance computing systems

High-performance computing systems are normally using a number of processors coupled together to form a parallel or distributed computer. The interconnection network is the heart of such a system. Since both high throughput and low delay are crucial for these systems, switched short-distance networks have been widely investigated and used (see, e.g., [Toda et al. 1994] [Rexford et al. 1998] for research considering

real-time aspects), while wormhole routing is an important method to reduce the delay.

Wormhole routing is a kind of cut-through (i.e., only the header of the packet with the destination address must arrive before the switch/router can start to forward the packet to an output port [Kermani and Kleinrock 1979]) where it is not needed to store the whole packet in a router if an output port is busy [Dally and Seitz 1987]. Instead, a flow control signal is propagated back to the transmitter to order it, and intermediate routers, to stop sending. The transmission of the packet is resumed when the busy port becomes free.

Several methods to obtain real-time services over wormhole-routed networks have been proposed. One way is to divide the physical channels into multiple virtual channels, each with dedicated buffers in the routers [Dally 1992]. Packet deadlines can then be used in combination with different priorities for the virtual channels to schedule real-time traffic [Li and Mutka 1994]. Using TDMA to obtain real-time communication over wormhole networks has also been investigated [Garcia et al. 2000], while other examples are found in [Song et al. 1999] [Sundaresan and Bettati 1997].

RACEway from Mercury Computer Systems [Kuszmaul 1995] is an example of a network specially developed for parallel, embedded real-time systems. Circuit switching with source routing is used. Support for real-time traffic is obtained by using priorities, where a higher priority transmission preempts a lower priority transmission. The link bandwidth is 160 MByte/s. Infiniband is a more general emerging standard but likely to become important for both cluster computing and embedded systems. Support for different real-time traffic classes in Infiniband has been investigated [Pelissier 2000] [Alfaro et al. 2002].

14.9 Fiber-optic networks

To mention a few words about fiber-optic networks, networks taking advantage of the Wavelength Division Multiplexing (WDM) technique are chosen. The advantage with those networks compared to, e.g., FDDI, is that a number of channels (wavelengths of the light) can coexist in a totally passive optical network. The WDM star network is the most well known passive optical network.

A real-time protocol for packet switched communication in WDM star networks is described in [Yan et al. 1996]. The QoS associated with a real-time packet in the network is related to the probability of missing its deadline. The protocol tries to globally minimize the number of packets not managing to keep their QoS by adaptively changing the priority of queued packets. Although dynamic real-time properties are

supported, the matter of the success or failure of a packet transmission depends on the global state of the network, and transmission success cannot be guaranteed in advance. A network that really supports hard real-time traffic is presented in [Jonsson et al. 1997]. The protocol is based on TDMA with semi-dynamic slot-allocation.

14.10 Wireless networks

Wireless networking is a field where, e.g., the high bit-error rate often must be specially treated. Moreover, the error characteristics can be rather bursty and location dependent. New scheduling algorithms must therefore be developed [Cao and Li 2001]. Other problems to cope with include the highly dynamic properties apparent in ad hoc (multi-hop) networks. Real-time protocols and scheduling algorithms have been proposed for both WLANs (Wireless LANs) [Gu and Zhang 2003] and for Wireless ATM [Frigon et al. 2001]. Routing issues for real-time communication in ad hoc networks have also been investigated [Chlamtac 1989] [Chakrabarti and Mishra 2001] [Bilstrup and Wiberg 1999], while another field is that of considering information theory to improve the chances to meet deadlines [Uhlemann et al. 2000].

14.11 Introduction to included papers

The four following appended papers gives a good coverage of the real-time communication research performed in ARTES. Also, they together provide good examples for several of the fields surveyed above.

The first paper presents novel work for the otherwise well explored CAN bus. Nolte, Nolin, and Hansson investigate the use of server-based scheduling with EDF for CAN, where a master node gives the other nodes credits to send in the next cycle. During the cycle, the native CAN protocol is used for medium access control. The master node keeps track of the other nodes' activities to, for example, be able to take care of unused resources when scheduling.

In the second paper, by Bergenhem and Jonsson, a ring network built up by fiber-ribbon links over which data and medium access control packets travel in separate fibers is described. The network has support for spatial reuse of the bandwidth to allow for several transmissions in different segments of the network simultaneously. Together with the slotted real-time protocol, this is shown to be a good solution for, e.g., high-performance signal processing applications.

In the third paper, Uhlemann, Rasmussen, and Wiberg present a framework for deadline dependent coding. This is a novel area of combining the fields of information theory and real-time communication, and

is of special value for wireless communication systems because of the usually high bit-error rate. Both error correcting codes and retransmission schemes are discussed in the paper from a real-time point-of-view.

The fourth paper, by Hoang and Jonsson, presents work on using switched Ethernet for industrial real-time applications. It is described how to incorporate deadline scheduling in the switch and the end-nodes, still supporting the widespread TCP/IP suite. Moreover, deadline-partitioning schemes to divide the deadline (delay bound) for the different physical links to be traversed by logical connections are discussed. This can be used to remove bottlenecks, for example, between master nodes and the switch in a master-slave system.

Bibliography

[Alfaro et al. 2002] F. J. Alfaro, J. L. Sanchez, and J. Duato, “A Strategy to Manage Time Sensitive Traffic in InfiniBand,” *Proc. Workshop on Communication Architecture for Clusters (CAC’02) in conjunction with International Parallel and Distributed Processing Symposium (IPDPS’02)*, Fort Lauderdale, FL, USA, Apr. 15, 2002.

[Alves et al. 2000] M. Alves, E. Tovar, G. Fohler, and G. Buttazzo, “CIDER - envisaging a COTS communication infrastructure for evolutionary dependable real-time systems,” *Proc. for the Work in Progress and Industrial Experience Sessions, 12th Euromicro Conference on Real-Time Systems (ECRTS’2000)*, Stockholm, Sweden, June 2000, pp. 19-22.

[Alves et al. 2002] M. Alves, E. Tovar, F. Vasques, G. Hammer, and K. Rother, “Real-time communications over hybrid wired/wireless PROFIBUS-based networks,” *Proc. 14th Euromicro Conference on Real-Time Systems*, pp. 142-151, 19-21 June, 2002.

[Arvind et al. 1991] K. Arvind, K. Ramamritham, and J. A. Stankovic, “A local area network architecture for communication in distributed real-time systems,” *Journal of Real-Time Systems*, vol. 3, no. 2, pp. 115-147, May 1991.

[Bergenheim and Jonsson 2003] Bergenheim, C. and M. Jonsson, “A pipelined ring network – heterogeneous real-time in radar signal processing,” *Proc. IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN’2003)*, Innsbruck, Austria, Feb. 10-13, 2003, pp. 825-832.

[Bilstrup and Wiberg 1999] U. Bilstrup and P.-A. Wiberg, “Routing protocol for wireless real-time multihop network,” *Proc. 11th Euromicro Conference on Real-Time Systems*, York, UK, June 9-11, 1999, pp. 5-7.

[Cao and Li 2001] Y. Cao and V. O. K. Li, "Scheduling algorithms in broad-band wireless networks," *Proceedings of the IEEE*, vol. 89, no. 1, pp. 76-87, Jan. 2001.

[Carpenzano et al. 2002] A. Carpenzano, R. Caponetto, L. Lo Bello, and O. Mirabella, "Fuzzy traffic smoothing: an approach for real-time communication over Ethernet networks," *Proc. 4th IEEE International Workshop on Factory Communication Systems*, pp. 241-248, Aug. 28-30, 2002.

[Chakrabarti and Mishra 2001] S. Chakrabarti and A. Mishra, "QoS issues in ad hoc wireless networks," *IEEE Communications Magazine*, vol. 39, no. 2, pp. 142-148, Feb. 2001.

[Chen et al. 1997] B. Chen, A. Sahoo, W. Zhao, and A. Raha, "Connection-oriented communications for real-time applications in FDDI-ATM-FDDI heterogeneous networks," *Proc. 17th International Conference on Distributed Computing Systems*, pp. 225-234, 27-30 May 1997.

[Chlamtac 1989] I. Chlamtac, "Bounded delay routing for real time communication in tactical radio networks," *Proc. IEEE MILCOM*, 1989.

[Dally 1992] W. J. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194-205, Mar. 1992.

[Dally and Seitz 1987] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. 36, no. 5, pp. 547-553, May 1987.

[Demers et al. 1989] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Proc. ACM SIGCOMM'89*, Austin, Texas, USA, Sept. 25-27, 1989, pp. 1-12.

[El-Gendy et al. 2003] M. A. El-Gendy, A. Bose, K. G. Shin, "Evolution of the Internet QoS and support for soft real-time applications," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1086-1104, July 2003.

[Fan et al. 2004] Fan, X., M. Jonsson, and H. Hoang, "Efficient many-to-many real-time communication using an intelligent Ethernet switch," *Proc. International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'2004)*, Hong Kong, China, May 10-12, 2004.

[Ferrari and Verma 1990] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal of Selected Areas in Communications*, vol. 8, no. 3, pp. 368-379, Apr. 1990.

[Ferreira et al. 2002] J. Ferreira, P. Pedreiras, L. Almeida, and J. A. Fonseca, "The FTT-CAN protocol for flexibility in safety-critical systems," *IEEE Micro*, vol. 22, no. 4, pp. 46-55, July-Aug. 2002.

[FlexRay] The FlexRay Consortium: <http://www.flexray.com/>.

[Frigon et al. 2001] J.-F. Frigon, V. C. M. Leung, and H. C. B. Chan, “Dynamic reservation TDMA protocol for wireless ATM networks,” *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 2, pp. 370-383, Feb. 2001.

[Garcia et al. 2000] A. Garcia, L. Johansson, M. Jonsson, and M. Weckstén, “Guaranteed periodic real-time communication over wormhole switched networks,” *Proc. ISCA 13th International Conference on Parallel and Distributed Computing Systems (PDCS-00)*, Las Vegas, NV, USA, Aug. 8-10, 2000.

[Gu and Zhang 2003] D. Gu and J. Zhang, “QoS enhancement in IEEE 802.11 wireless local area networks,” *IEEE Communications Magazine*, vol. 41, no. 6, pp. 120-124, June 2003.

[Hassan et al. 2000] M. Hassan, A. Nayandoro, and M. Atiquzzaman, “Internet telephony: services, technical challenges, and products”, *IEEE Communications Magazine*, vol. 38, no. 4, pp. 96-103, Apr. 2000.

[Hoang et al. 2002] Hoang, H., M. Jonsson, U. Hagström, and A. Kallerdahl, “Switched real-time Ethernet with earliest deadline first scheduling - protocols and traffic handling,” *Proc. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'2002) in conjunction with International Parallel and Distributed Processing Symposium (IPDPS'02)*, Fort Lauderdale, FL, USA, April 15-16, 2002.

[Jonsson et al. 1997] M. Jonsson, K. Börjesson, and M. Legardt, “Dynamic time-deterministic traffic in a fiber-optic WDM star network,” *Proc. 9th Euromicro Workshop on Real Time Systems (EWRTS'97)*, Toledo, Spain, June 11-13, 1997, pp. 25-33.

[Kandlur et al. 1991] D. D. Kandlur, K. G. Shin, and D. Ferrari, “Real-time communication in multi-hop networks,” *Proc. 11th International Conference on Distributed Computing Systems*, pp. 300-307, May 20-24, 1991.

[Kermani and Kleinrock 1979] P. Kermani and L. Kleinrock, “Virtual cut-through: a new computer communication switching technique,” *Computer Networks*, vol. 3, pp. 267-286, 1979.

[Kopetz and Grünsteidl 1994] H. Kopetz and G. Grünsteidl, “TTP - a protocol for fault-tolerant real-time systems,” *Computer*, vol. 27, no. 1, pp. 14-23, Jan 1994.

[Kopetz and Bauer 2003] H. Kopetz and G. Bauer, “The time-triggered architecture,” *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112-126, Jan. 2003.

[Kopetz et al. 1989] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger, “Distributed fault-tolerant real-time systems: the Mars approach,” *IEEE Micro*, vol. 9, no. 1, pp. 25-40, Feb. 1989.

[Kopetz et al. 2000] H. Kopetz, M. Holzmann, and W. Elmenreich, “A universal smart transducer interface: TTP/A,” *Proc. Third IEEE*

International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'2000), pp. 16-23, Mar. 15-17, 2000.

[Kuszmaul 1995] B. C. Kuszmaul, "The RACE network architecture," *Proc. 9th International Parallel Processing Symposium (IPPS'95)*, Santa Barbara, CA, USA, Apr. 25-28, 1995, pp. 508-513.

[Kweon et al. 2000] S.-K. Kweon, K. G. Shin, and G. Workman, "Achieving real-time communication over Ethernet with adaptive traffic smoothing," *Proc. 6th IEEE Real-Time Technology and Applications Symposium (RTAS'2000)*, Washington, D.C., USA, 31 May - 2 June 2000, pp. 90-100.

[Leen and Heffernan 2002] G. Leen and D. Heffernan, "TTCAN: a new time-triggered controller area network," *Microprocessors and Microsystems*, vol. 26, no. 2, pp. 77-94, Mar. 17, 2002.

[Leen et al.] G. Leen, D. Heffernan, and A. Dunne, "Digital networks in the automotive vehicle," *Computing & Control Engineering Journal*, vol. 10, no. 6, pp. 257-266, Dec. 1999.

[Li and Mutka 1994] J.-P. Li and M. W. Mutka, "Priority based real-time communication for large scale wormhole networks," *Proc. Eighth International Parallel Processing Symposium*, Cancun, Mexico, Apr. 26-29, 1994, pp. 433-438.

[Malcolm and Zhao 1994] N. Malcolm and W. Zhao, "The timed-token protocol for real-time communications," *Computer*, vol. 27, no. 1, pp. 35-41, Jan. 1994.

[Metzler et al. 1999] B. Metzler, T. Harbaum, R. Wittmann, and M. Zitterbart, "AMnet: heterogeneous multicast services based on active networking," *Proc. IEEE Second Conference on Open Architectures and Network Programming (OPENARCH'99)*, pp. 98-105, 26-27 Mar. 1999.

[Mock and Nett 1999] M. Mock and E. Nett, "Real-time communication in autonomous robot systems," *Proc. Fourth International Symposium on Autonomous Decentralized Systems (ISADS'1999)*, Tokyo, Japan, 21-23 March 1999, pp. 34-41.

[Molle and Kleinrock 1985] M. Molle L. Kleinrock, "Virtual time CSMA: why two clocks are better than one," *IEEE Transactions on Communications*, vol. 33, no. 9, pp. 919-933, Sept. 1985.

[Montuschi et al. 1992] P. Montuschi, L. Ciminiera, and A. Valenzano, "Time characteristics of IEEE 802.4 token bus protocol," *IEE Proceedings-E (Computers and Digital Techniques)*, vol. 139, no. 1, pp. 81-87, Jan. 1992

[Nolte et al. 2003] T. Nolte, M. Sjödin, H. Hansson, "Server-based scheduling of the CAN bus," *Proc. IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'03)*, vol. 1, pp. 169-176, Sept. 16-19, 2003.

[Parekh and Gallager 1994] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated ser-

vices networks: the multiple node case,” *IEEE/ACM Transactions on Networking*, vol. 2, no. 2, pp. 137-150, Apr. 1994.

[Pedro and Burns 1997] P. Pedro and A. Burns, “Worst case response time analysis of hard real-time sporadic traffic in FIP networks,” *Proc. Ninth Euromicro Workshop on Real-Time Systems*, Toledo, Spain, June 11-13, 1997, pp. 3-10.

[Pelissier 2000] J. Pelissier, “Providing quality of service over Infini-band architecture fabrics,” *Proc. Hot Interconnects*, 2000.

[Rexford et al. 1998] J. Rexford, J. Hall, and K. G. Shin, “A router architecture for real-time communication in multicomputer networks,” *IEEE Transactions on Computers*, vol. 47, no. 10, pp. 1088-1101, Oct. 1998.

[Rushby 2001] J. Rushby, “Bus architectures for safety-critical embedded systems,”

First Workshop on Embedded Software (EMSOFT’2001), Oct. 8-10, 2001, Lake Tahoe, CA, USA. Published in Springer-Verlag *Lecture Notes in Computer Science*, vol. 2211, pp. 306-323.

[Schuh 1988] R. A. Schuh, “An overview of the 1553 bus with testing and simulation considerations,” *Proc. 5th IEEE Instrumentation and Measurement Technology Conference (IMTC-88)*, pp. 20-25, Apr. 20-22, 1988.

[Song et al. 1999] H. Song, B. Kwon, and H. Yoon, “Throttle and preempt: a flow control policy for real-time traffic in wormhole networks,” *Journal of Systems Architecture*, vol. 45, no. 8, pp. 633-649, Feb. 1999.

[Sundaresan and Bettati 1997] S. Sundaresan and R. Bettati, “Distributed connection management for real-time communication over wormhole-routed networks,” *Proc. 17th International Conference on Distributed Computing Systems*, Baltimore, MD, USA, May 27-30, 1997, pp. 209-216.

[Stankovic et al. 2003] J. A. Stankovic, T. E. Abdelzaher, C. Lu, L. Sha, and J. C. Hou, “Real-time communication and coordination in embedded sensor networks,” *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1002-1022, July 2003.

[Tindell et al. 1994] Tindell, K. W., H. Hansson, and A. J. Wellings, “Analysing real-time communications: controller area network (CAN),” *Proc. 15th IEEE Real-Time Systems Symposium*, pp. 259-263, 1994.

[Toda et al. 1994] K. Toda, K. Nishida, E. Takahashi, and Y. Yamaguchi, “A priority forwarding router chip for real-time interconnection networks,” *Proc. 15th IEEE Real-Time Systems Symposium*, pp. 63-73, 1994.

[Tovar and Vasques 1999] E. Tovar and F. Vasques, “Real-time field-bus communications using Profibus networks,” *IEEE Transactions on Industrial Electronics*, vol. 46, no. 6, pp. 1241-1251, Dec. 1999.

[Uhlemann et al. 2000] E. Uhlemann, P.-A. Wiberg, T. M. Aulin, and L. K. Rasmussen, "Deadline dependent coding - a framework for wireless real-time communication," *Proc. 7th International Conference on Real-Time Computing Systems and Applications (RTCSA '00)*, Cheju Island, South Korea, Dec. 12-14, 2000, pp. 135-142.

[Varadarajan and Chiueh 1998] S. Varadarajan and T. S. Chiueh, "EtheReal: a host-transparent real-time Fast Ethernet switch," *Proc. 6th International Conference on Network Protocols*, 1998, pp. 12-21.

[Venkatramani and Chiueh 1994] C. Venkatramani and T. Chiueh, "Supporting real-time traffic on Ethernet," *Proc. Real-Time Systems Symposium (RTSS'1994)*, pp. 282-286, Dec. 7-9, 1994.

[Verma et al. 1991] D. Verma, H. Zhang and D. Ferrari, "Guaranteeing delay jitter bounds in packet switching networks," *Proc. Tri-Comm'91*, Chapel Hill, NC, USA, Apr. 1991, pp. 35-46.

[Wolf and R. Steinmetz 1997] L. C. Wolf and R. Steinmetz, "Multi-media communication," *Proceedings of the IEEE*, vol. 85, no. 12, Dec. 1997.

[Xiao and Ni 1999] X. Xiao and L. M. Ni, "Internet QoS: a big picture," *IEEE Network*, vol. 13, no. 2, pp. 8-18, Mar./Apr. 1999.

[Yan et al. 1996] A. Yan, A. Ganz, and C. M. Krishna, "A distributed adaptive protocol providing real-time services on WDM-based LAN's," *Journal of Lightwave Technology*, vol. 14, no. 6, pp. 1245-1254, June 1996.

[Zhang 1995] H. Zhang, "Service disciplines for guaranteed performance in packet-switching networks," *Proceedings of the IEEE*, vol. 83, no. 10, pp. 1374-1396, Oct. 1995.

[Zhao and Ramamritham 1987] W. Zhao and K. Ramamritham, "Virtual time CSMA protocols for hard real-time communications," *IEEE Transactions on Software Engineering*, vol. 13, no. 8, pp. 938-952, Aug. 1987.

[Zhao et al. 1990] W. Zhao, J. A. Stankovic, K. Ramamritham, "A window protocol for transmission of time-constrained messages," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1186-1203, Sept. 1990.

[Zheng and Shin 1994] Q. Zheng and K. G. Shin, "On the ability of establishing real-time channels in point-to-point packet-switched networks", *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 1096- 1105, Feb./Mar./Apr. 1994.

[Zuberi and Shin 1995] K. M. Zuberi and K. G. Shin, "Non-preemptive scheduling of messages on controller area network for real-time control applications," *Proc. Real-Time Technology and Applications Symposium (RTAS'1995)*, pp. 240-249, May 15-17, 1995.

Chapter 15

Real-Time Server-Based Communication for CAN

By **Thomas Nolte**, **Mikael Nolin** and **Hans Hansson**
Mälardalen Real-Time Research Centre
Department of Computer Science and Engineering
Mälardalen University
Email: {thomas.nolte,mikael.nolin,hans.hansson}@mdh.se

This chapter investigates the concept of share-driven scheduling of networks with real-time properties. Share-driven scheduling provides fairness and bandwidth isolation between predictable as well as unpredictable streams of messages on the network.

The need for this kind of scheduled real-time communication network is high in applications that have requirements on flexibility, both during development for assigning communication bandwidth to different applications, and during run-time to facilitate dynamic addition and removal of system components.

We illustrate the share-driven scheduling concept by applying it to the popular Controller Area Network (CAN) resulting in a new share-driven scheduler called Server-CAN. By separating streams of messages by using network access servers (N-Servers), scheduling is performed at three layers where native CAN message arbitration is the scheduling at the lowest level. On top of this is the server-based scheduling to separate different streams of messages within the system. Looking at a single N-Server, several users might share one server. Hence, scheduling is performed at the third level every time the N-Server is being scheduled for message transmission. Here different queuing policies play a role in the scheduling performed.

15.1 Introduction

In optimising the design of a real-time communications system it is important to both guarantee the timeliness of periodic messages and to

minimize the interference from periodic traffic on the transmission of aperiodic messages. Therefore, we propose the usage of *server-based scheduling techniques*, which can handle streams of both periodic and aperiodic traffic. The need for this kind of scheduled real-time communication network is high in applications that have requirements on flexibility, both during development for assigning communication bandwidth to different applications, and during run-time to facilitate dynamic addition and removal of system components.

In this chapter we present Server-CAN, a share-driven scheduler, that provides mechanisms for server-based scheduling of the Controller Area Network (CAN) [CAN02, ISO93], which is one of the more commonly used communications networks, used extensively in the automotive industry as well as in several other application domains that have real-time requirements. CAN is implementing Fixed-Priority Scheduling (FPS) using a bit-wise arbitration mechanism in the Medium Access Control (MAC) layer. This mechanism avoids collisions in a deterministic way and is amenable to timing analysis [TB94, TBW95, THW94].

Using server-based scheduling techniques for CAN, fairness among users of the network is guaranteed (e.g., “misbehaving” processes cannot aversively interfere with well-behaved processes). In contrast with other proposals to schedule CAN, aperiodic messages are not sent “in the background” of periodic messages or in separate time-slots (e.g., [PA00]). Instead, aperiodic and periodic messages are jointly scheduled using servers. This substantially facilitates meeting response-time requirements, both for aperiodic and periodic messages.

We propose the usage of server techniques based on the Earliest Deadline First (EDF) scheduling strategy, since EDF allows optimal resource utilisation. Examples of such server-based scheduling techniques are the Total Bandwidth Server (TBS) [SB94, SBS95] and the Constant Bandwidth Server (CBS) [Abe98]. In this chapter we are working with a distributed CAN network, where a dynamic priority version of the Polling Server (PS) [LSS97, SSL89] is used. By using servers, we investigate the concept of share-driven scheduling of networks with real-time properties. Share-driven scheduling provides fairness and bandwidth isolation between predictable as well as unpredictable streams of messages on the network.

As a side effect of using servers, the CAN frame identifiers (which are also used as priorities during the arbitration) will not play a significant role in the frame response-time. This greatly simplifies the assignment of frame identifiers (which is often done in an ad-hoc fashion at an early stage in a project). This also allows migration of legacy systems (where identifiers cannot easily be changed) into our new framework. Also, many domain specific standards specify which identifiers to be used,

further limiting the possibility to use the identifiers to tune network timing behaviour [CiA96, J1998].

This chapter is organised as follows: in Section 15.2 the Controller Area Network (CAN) is presented along with different methods to schedule CAN. Section 15.3 presents different server-based scheduling techniques. In Section 15.4 the two Server-CAN scheduling mechanisms for CAN are presented together with their real-time performance. Finally the chapter is summarised and concluded in Section 15.5.

15.2 The Controller Area Network (CAN)

The Controller Area Network (CAN) [CAN02, ISO93] is a broadcast bus designed to operate at speeds of up to 1Mbps. CAN is extensively used in automotive systems, as well as in other applications. CAN is a collision-avoidance broadcast bus, which uses deterministic collision resolution to control access to the bus (so called CSMA/CA). The frame identifier is required to be unique, in the sense that two simultaneously active frames originating from different sources must have distinct identifiers. Besides identifying the frame, the identifier serves two purposes: (1) assigning a priority to the frame, and (2) enabling receivers to filter frames. CAN guarantees that the highest priority active frame will be transmitted. Hence, CAN behaves like a priority-based queue.

15.2.1 Scheduling of CAN

The real-time research community uses three major types of scheduling: priority-driven (e.g., FPS or EDF) [LL73], time-driven (table-driven) [HL96, Kop98], and share-driven [PG93, SAWJ⁺96].

Priority-driven

For CAN, the most natural scheduling method is Fixed-Priority Scheduling (FPS) since it is the policy used by the CAN protocol. Analysis, like the FPS response-time tests to determine the schedulability of CAN message frames, have been presented by Tindell *et al.* [TB94, TBW95, THW94]. This analysis is based on the standard FPS response-time analysis for CPU scheduling presented by Audsley *et al.* [ABR⁺93].

Several methods for dynamic-priority type of scheduling have been proposed for CAN. By manipulating the message identifier, and therefore changing the message priority dynamically, several approaches to mimic EDF type of scheduling have been presented [LK98, LKJ98, Nat00, ZS95]. However, these solutions are all based on manipulating the identifier of the CAN frame to reflect the deadline of the frame and thus reducing the number of possible identifiers to be used by the system

designers. This is not an attractive alternative, since it interferes with other design activities, and is even sometimes in conflict with adopted standards and recommendations [CiA96, J1998].

Time-driven

Table-driven, or time-driven, scheduling of CAN is provided by, e.g., TT-CAN [ISO00] and FTT-CAN [AFF98, AFF99]. Flexible Time-Triggered CAN (FTT-CAN), presented by Almeida *et al.*, supports priority-driven scheduling in combination with time-driven scheduling. In FTT-CAN, time is partitioned into Elementary Cycles (ECs) that are initiated by a special message, the Trigger Message (TM). This message contains the schedule for the synchronous traffic (time-triggered traffic) that shall be sent within this EC. The schedule is calculated and sent by a specific node called the *master node*. FTT-CAN supports both periodic and aperiodic traffic by dividing the EC in two parts. In the first part, the asynchronous window, a (possibly empty) set of aperiodic messages are sent using CAN's native arbitration mechanism. In the second part, the synchronous window, traffic is sent according to the schedule delivered in the TM. It has also been shown how to schedule periodic messages according to EDF, using the synchronous part of the EC [PA01].

Share-driven

Share-driven scheduling of CAN (Server-CAN) has been presented in our earlier work on server-based scheduling of CAN, e.g., [Nol06, NNH05]. The server-based scheduling approach is presented in more detail in Section 15.4 below. By providing the option of share-driven scheduling of CAN, designers are given more freedom in designing a distributed real-time system where the nodes are interconnected with a CAN-bus.

15.3 Server-based scheduling

In the scheduling literature many types of *servers* are described, implementing server-based schedulers. In general, each server is characterised partly by its unique mechanism for assigning deadlines, and partly by a set of parameters used to configure the server. Examples of such parameters are *bandwidth*, *period*, and *capacity*.

15.3.1 Fixed priority schedulers (FPS)

Several server-based schedulers for FPS systems exist where the simplest one is the Polling Server (PS) [LSS97, SSL89]. A polling server allocates a share of the CPU to its users. This share is defined by the server's

period and capacity, i.e., the PS is guaranteed to allow its users to execute within the server's capacity each server period. The server is scheduled according to Rate Monotonic (RM) together with the normal tasks (if existing) in the system. However, a server never executes by itself. A server will only mediate the right to execute for its users, if some of its users have requested to use the server's capacity. Otherwise the server's capacity will be left unused for that server period. Hence, the PS serves its users by providing its capacity to the users each server period. However, if the PS is activated and no user is ready to use the server capacity, the capacity is lost for that server period and the server's users have to wait to the next server period to be served. Hence, the worst-case service a user can get is if it requests capacity right after the server is activated (with its capacity replenished). If no users have requested the server's capacity at the time when the server was activated, the server's capacity is lost. Then the user has to wait until the next time the server is activated making capacity available for the user. The behaviour of a PS server is in the worst-case equal to a task with the period of the server's period, and a worst-case execution time equal to the server's capacity. Hence, the analysis of a system running PS is straightforward.

Another server-based scheduler for FPS systems that is slightly better than the PS is the Deferrable Server (DS) [LSS97, SLS95]. Here, the server is also implemented as a periodic task scheduled according to RM together with the (if existing) other periodic tasks. The difference is that the server is not polling its users, i.e., checking if there are any pending users each server period and if not drop all its capacity. Instead, the DS preserves its capacity throughout the server period allowing its users to use the capacity at any time during the server period. As with the PS, the DS replenish its capacity at the beginning of each server period. In general, the DS is giving better response times than the PS. However, by allowing the servers' users to execute at any time during the servers' period violates the traditional RM scheduling (where the highest priority task has to execute as soon it is scheduled), lowering the schedulability bound for the periodic task set. A trade-off to the DS allowing a higher schedulability but a slight degradation in the response times is the Priority Exchange (PE) algorithm [LSS97]. Here the servers' capacities are preserved by exchanging it for the execution time of a lower priority periodic task. Hence, the servers' capacities are not lost but preserved at a lower priority, at the priority of the low priority task involved in the exchange. Note that the PE mechanisms are more complex than the DS mechanisms, which should be taken into consideration in the trade-off.

By changing the way capacity is replenished, the Sporadic Server (SS) [SSL89] is a server-based scheduler for FPS systems that allows

high schedulability without degradation. Instead of replenishing capacity at the beginning of the server period, SS replenish its capacity once the capacity has been consumed by its users. As DS, SS violates the traditional RM scheduling by not executing the highest priority task once it is scheduled for execution. However, this violation does not impact on the schedulability as the same schedulability bound is offered for a system running both with and without SS.

There are server-based schedulers for FPS systems having better performance in terms of response-time. However, this usually comes at a cost of high computational and implementation complexity together with high memory requirements. One of these schedulers is the Slack Stealer [LRT92]. It should be noted that there are no optimal algorithms in terms of minimising the response time. It has been proven [TLS96] the non existence of an algorithm that can both minimise the response time offered to users and at the same time guarantees the schedulability of the periodic tasks. Hence, there is a trade-off between response-time and schedulability when finding a suitable server-based scheduler for the intended target system.

15.3.2 Dynamic priority schedulers (DPS)

Looking at DPS systems, a number of server-based schedulers have been developed over the years. Many of the server-based schedulers for FPS systems have also been extended to EDF based DPS systems, e.g., the Dynamic Priority Exchange (DPE) [SB94, SB96] is an extension of PE and the Dynamic Sporadic Server (DSS) [SB94, SB96] is an extension of the SS. A very simple (implementation wise) server-based scheduler that provides faster response-times compared with SS yet not violating the overall load of the system (causing other tasks to miss their deadlines) is the Total Bandwidth Server (TBS) [SB94, SB96]. TBS makes sure that the server never uses more bandwidth than allocated to it (under the assumption that the users do not consume more capacity than specified by their worst-case execution times), yet providing a fast response time to its users (i.e., assigning its users with a close deadline as the system is scheduled according to EDF). A quite complex server-based scheduler is the Earliest Deadline Late server (EDL) [SB94, SB96] (which is a DPS version of the Slack Stealer). Also, there is an Improved Priority Exchange (IPE) [SB94, SB96] which has similar performance compared with the EDL [But05], yet being less complex (implementation wise). Also, TBS has been enhanced by improving its deadline assignment rule [But05]. When the worst-case execution times are unknown, the Constant Bandwidth Server (CBS) [AB98] can be used, guaranteeing that the servers' users will never use more than the servers' capacity.

15.4 Server-based scheduling of CAN

On a single node in a distributed system, task scheduling can be performed in several ways. If tasks are periodic they can simply be scheduled directly once every period time. However, if tasks are either sporadic or aperiodic, the time of task arrival is unknown making the scheduling a bit harder. This problem is for task scheduling traditionally solved by using server-based scheduling techniques.

15.4.1 Server

In real-time scheduling, a *server* is a conceptual entity that controls the access to some shared resource. Sometimes multiple servers are associated with a single resource. For instance, in this chapter we will have multiple servers mediating access to a single CAN bus. The servers have exclusive associated bandwidth in terms of capacity, C , and period time, T . Moreover, servers have an associated deadline in order to be scheduled. Also, the servers have a local queue, in which its pending users are stored.

15.4.2 User

A server might have one or several *users*. In this chapter a user is a stream of messages, e.g., the sending of messages by an application, and these users can be of either periodic, sporadic or aperiodic nature. Since each server has exclusive right to a share of the total system bandwidth, all users sharing a server will share this portion of bandwidth. Hence, depending on the type of queue used at the server, e.g., FIFO or priority-based, different guarantees of timeliness can be offered. Suppose a priority-based queue is used, then the users will experience a service similar to the one of an exclusive network, essentially with the only difference in the lower bandwidth. Hence, the timely behaviour will be, compared to an exclusive CAN network, divided by C/T , i.e., the server's share. A variant of the FPS response-time analysis could be used to calculate the timing properties.

The Server-CAN scheduler

In this chapter, Server-CAN is implementing server-based techniques by assigning a deadline to the user upon user-arrival. All users are then scheduled according to the EDF scheduling policy. By using servers, different users, or groups of users, can be separated from each other in order to guarantee a certain share of the resource. This is traditionally done by the usage of server-based techniques. By using servers, a badly behaving user in one server, can not starve other well behaving users in

other servers, since each server cannot consume more than its allocated bandwidth, this due to the central scheduling mechanism. As shown in Figure 15.1, several servers can be allocated to one node.

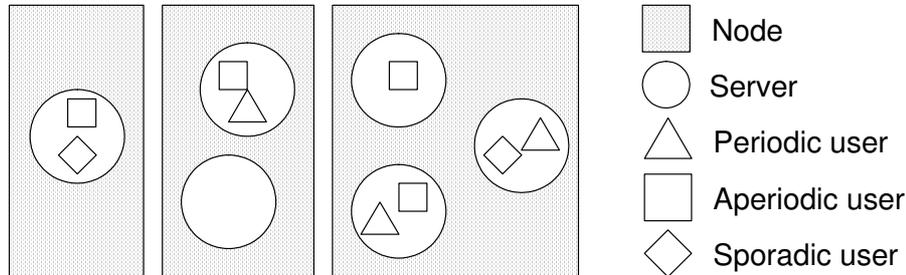


Figure 15.1: System with servers.

Each server has bandwidth associated to it. Traditionally this is expressed by some capacity, C , and a period time, T . Hence, the bandwidth available is expressed by C/T . Each of the servers in the system has one or several associated users. Whenever a user requests to use some of the capacity available in the server, it will be associated with a deadline based on the server parameters.

Using the server-based concept, servers and users can potentially join and leave the system arbitrarily as long as the total utilisation, or bandwidth demand, in the system by all the servers is less or equal to the theoretical maximum. Note that the joining and leaving of users does not affect other users than those sharing the same server.

15.4.3 Scheduling mechanism

In Server-CAN the scheduling is performed at the master node, called M-Server. The M-Server has all information regarding the servers in the system. The servers are called network access servers (N-Servers). Hence the M-Server has all the N-Server's parameters. These N-Servers have an associated deadline in order to be scheduled for message transmission. Moreover, each N-Server has a local message queue, in which all its user messages are stored. As soon as the N-Server is being scheduled for message transmission, the N-Server selects a message from its local message queue. The server-scheduling mechanism is depicted in Figure 15.2.

The scheduling is performed at the M-Server according to the EDF policy (Figure 15.2:1, i.e., (1) in Figure 15.2). Time is divided into Elementary Cycles (ECs), similar to the FTT-CAN approach [AFF98, AFF99]. In order for an N-Server not to cause more than one erroneous

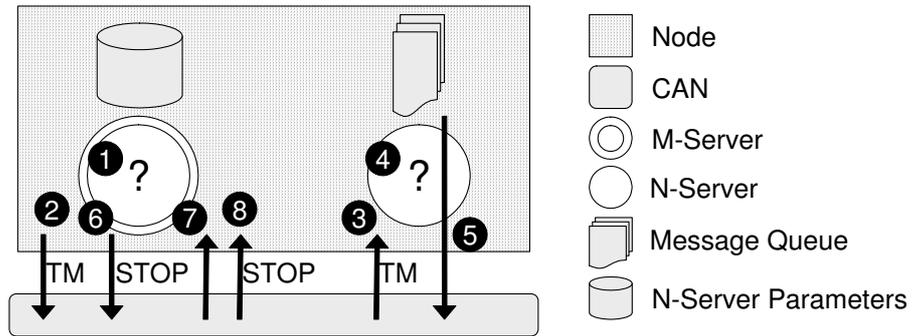


Figure 15.2: Server-scheduling mechanism for CAN.

guess during its N-Server period, the M-Server will apply the following rule when deciding the schedule for the EC starting at time t

1. Only N-Servers that are within one N-Server period T_s from their absolute deadline d_s will be eligible for scheduling. Formally the following condition must hold for a N-Server s to be eligible for scheduling:

$$d_s - t \leq T_s \tag{15.1}$$

2. Among the eligible N-Servers, select N-Servers in EDF order to be scheduled during the EC.

Hence, a schedule is created containing the eligible N-Servers with the earliest deadlines, potentially filling up one EC (if enough eligible N-Servers exist). When this is done, the schedule is put into a trigger message (TM) (an example TM is shown in Figure 15.3) and multicasted to all the N-Servers in the system (Figure 15.2:2).

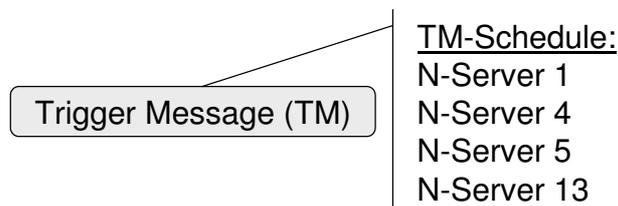


Figure 15.3: Example Trigger Message (TM) and its contents.

When the N-Servers receive the TM (Figure 15.2:3), they will read it to see whether or not they are allowed to send a message (Figure 15.2:4).

If they are, their message is immediately (if existing) queued in their local CAN controller, and sent using CAN's native arbitration mechanism (Figure 15.2:5). Otherwise, if not scheduled, the node has to wait for the next TM to see if it is scheduled that time.

In order to terminate the EC, the M-Server is also sending a STOP message to itself (Figure 15.2:6). This is done right after the TM is sent. The STOP message is of lowest priority possible, acting as an indicator for when all the nodes that were allowed to send a message within the active EC actually have sent their messages. If the M-Server receives the STOP message, the conclusion can be drawn that all nodes that were allowed to send messages within the EC have already sent their messages. This since the STOP message, with its low priority, is the last message to be sent within the EC. An example EC is depicted in Figure 15.4.

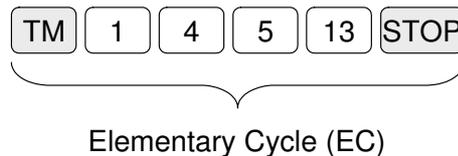


Figure 15.4: Elementary Cycle (EC) containing messages from 4 N-Servers.

The M-Server is always reading all the messages that are sent on the CAN bus (Figure 15.2:7), i.e., the M-Server is polling the bus. This in order to update its server variables based on the actual traffic sent on the bus. Since servers are scheduled for message transmission even though they might not always have any messages to send, this has to be taken care of by updating the server parameters accordingly. When a server is being scheduled for message transmission and it sends a message, its corresponding server parameters are updated accordingly. However, if it would not send a message, its server parameters must be updated according to this. There are different ways of updating the server-parameters in the case when the server did not send a message. Depending on how the server-parameters are updated the server will have different real-time characteristics, as explained later in Section 15.4.5.

When the M-Server reads the STOP message (Figure 15.2:8), the EC is terminated and the next EC is initiated based on the updated server-variables.

15.4.4 Limitations

Note that this scheduling mechanism limits the available bandwidth on the network. The theoretical maximum network utilisation is expressed by

$$\sum_{s=1}^{|N|} \left(\frac{|M|}{T_s} \right) \leq 1 - \left(\frac{|TM| + |STOP| + T_{sched}}{T_{EC}} \right) \quad (15.2)$$

where $|N|$ is the number of N-Servers in the system, $|M|$ is the length (in time) of a message (typically of worst-case length which is 135 bits), T_s is the period of N-Server s , $|TM|$ is the length (in time) of a TM message (typically 135 bits), $|STOP|$ is the length (in time) of a STOP message (typically 55 bits), T_{sched} represents the computational overhead of updating the N-Server absolute deadlines, producing the new schedule and encoding it into a TM after receiving the STOP message as well as the decoding of the TM in the slaves, and T_{EC} is the length (in time) of the EC.

15.4.5 Parameter updating

The server-parameters can be updated in a number of ways depending on the desired result. In [Nol06, NNH05] we present two Server-CAN schedulers with different approaches of updating the server-parameters called S³-CAN and PS²-CAN.

Simple Server-Scheduled CAN (S³-CAN)

The N-Server parameter-updating of S³-CAN is optimistic in the sense that as soon an erroneous guess regarding message readiness at an N-Server is made, no action is done to cover for the loss of bandwidth. This loss of bandwidth is due to the increase of the protocol overhead caused by more frequent TM and STOP messages when the EC is terminated earlier, i.e., shorter EC than originally intended. Using S³-CAN, all N-Servers in the system are sharing this loss of bandwidth, affecting the timeliness of the N-Servers.

If all N-Servers did send their messages when scheduled, their N-Server parameters are updated normally. However, if any N-Server did not send its message when it was scheduled for message transmission, the assumption for that N-Server will be that a message now has arrived to that specific N-Server, so it must be scheduled in T_s time units from now (i.e., within N-Server s 's period from now). Hence, its N-Server parameters are updated accordingly.

Looking at S³-CAN, as the M-Server receives the STOP message, the EC is terminated and the absolute deadlines of all (within the EC) sched-

uled N-Servers are updated according to whether they sent messages or not. The following rules apply for updating the N-Server absolute deadlines

1. If the scheduled N-Server sent a message, the M-Server's guess on message readiness was correct. In this case the M-Server updates the N-Server's absolute deadline d_s according to

$$d_s^m = d_s^{m-1} + D_s \quad (15.3)$$

where D_s is the N-Server relative deadline.

2. If the scheduled N-Server did **not** send a message, the M-Server's guess was incorrect and the N-Server's absolute deadline is updated according to

$$d_s^m = \max(t + D_s, d_s^{m-1}) \quad (15.4)$$

where t is the time of the scheduling.

Looking at the above rules for updating the N-Server deadlines, they mimic a TBS server with capacity set to one (1) message.

An upper bound on the response-time R_s for a single message arriving at an arbitrary N-Server s is captured by

$$R_s = T_{EC} - |TM| + T_s - \min(i : i \in N, T_i) + \left\lceil \frac{|N|}{|EC|} \right\rceil \times T_{EC} - |STOP| \quad (15.5)$$

where T_{EC} is the length of an EC in time (including the transmission of TM and STOP messages), $|TM|$ is the length of a TM message in time, T_s is the N-Server period of N-Server s , $\min(i : i \in N, T_i)$ represents the N-Server period of the shortest period blocking N-Server (where N is the set of all N-Servers in the system), $|N|$ denotes the number of N-Servers in the system, $|EC|$ denotes the number of messages fitting in an EC assuming worst-case message size, and $|STOP|$ is the length of a STOP message in time.

For simplicity Equation 15.5 can be rewritten and both $|TM|$ and $|STOP|$ can safely (pessimistically) be removed (for readability) forming the following equation representing the worst-case response-time for a single message transmitted through an arbitrary N-Server s

$$R_s = T_s + \left(1 + \left\lceil \frac{|N|}{|EC|} \right\rceil \right) \times T_{EC} - \min(i : i \in N, T_i) \quad (15.6)$$

For more details on S³-CAN including formal proofs of the worst-case response-time, interested readers are referred to [Nol06].

Periodic Server-Scheduled CAN (PS²-CAN)

The previously presented Server-CAN scheduler, S³-CAN, has a worst-case response-time for a message, sent through an arbitrary N-Server s , that has a large part which is independent of N-Server s 's relative deadline (D_s). This can for many network configurations (e.g., configurations with many N-Servers) give unreasonable worst-case response-times. Therefore, this section presents a modification to S³-CAN where the worst-case response-time of an N-Server s instead is mainly dependent on the N-Server relative deadline D_s (which is for PS²-CAN equal to the N-Server period T_s). The modification is called Periodic Server-Scheduled CAN (PS²-CAN). PS²-CAN is a dynamic priority version of the Polling Server (PS) [LSS97, SSL89].

In S³-CAN the increased protocol overhead caused by any erroneous guesses penalise mostly N-Servers with long periods (since N-Servers with short periods are less likely to suffer from shorter period blocking N-Servers). PS²-CAN will compared to S³-CAN penalise the N-Servers which causes erroneous guesses (i.e., the N-Servers that do not send messages when they are scheduled for message transmission) by delaying them at one N-Server period (hence the name PS²-CAN).

In PS²-CAN the M-Server will always treat an N-Server that has been scheduled for message transmission as if the N-Server actually did send a message (regardless whether it did send any message). Hence the rules for updating the N-Server states are changed to a single rule

- Treat the N-Server as if it sent a message, and update its N-Server parameters and state accordingly. The next guess is that the N-Server has more messages to send.

$$d_s^m = d_s^{m-1} + D_s \quad (15.7)$$

Using PS²-CAN, the worst-case response-time for N-Server s is not greater than

$$R_s = 2 \times T_s + T_{EC} \quad (15.8)$$

where $2 \times T_s$ is the worst-case response-time (in multiples of N-Server periods) for a single message using PS²-CAN, and T_{EC} covers for the time granularity due to EC-scheduling. For more details on PS²-CAN including formal proofs of the worst-case response-time, interested readers are referred to [Nol06].

Comparison Comparing PS²-CAN with S³-CAN, the main difference is that with PS²-CAN an N-Server with a short period will not have to wait for a time longer than twice its server period. Using S³-CAN the maximum time an N-Server could have to wait for is determined by the

number of N-Servers in the system together with the size of the EC. In that case an N-Server might have to wait for a period longer than twice its server period.

Both S^3 -CAN and PS^2 -CAN claim to enforce bandwidth isolation among N-Servers. This is true, since the only way for a N-Server to send a message is to be scheduled in a TM. This scheduling is performed by the M-Server based on the N-Server parameters. As long as the N-Servers are not failing, the bandwidth isolation property among N-Servers will be kept. However, no bandwidth isolation is provided among users sharing the same N-Server, as several users might share one N-Server.

If the system consists of a large set of N-Servers with short server periods, PS^2 -CAN is better, since the worst-case response-time for these N-Servers are limited based on their periods rather than the total number of N-Servers, whereas if the system consists of a few N-Servers with long server period, S^3 -CAN gives lower worst-case response times.

For more details on concerning the evaluation of S^3 -CAN and PS^2 -CAN, interested readers are referred to [Nol06].

15.5 Summary and conclusions

This chapter has presented two share-driven Server-CAN schedulers for scheduling of the Controller Area Network (CAN) [CAN02, ISO93]. The presented protocols are the first server-based scheduling mechanisms using dynamic priorities proposed for CAN. Earliest Deadline First (EDF) scheduling is used to achieve high utilisation. The presented approach allows for utilisation of the CAN bus in a more flexible way compared to other scheduling approaches, e.g., native CAN and Flexible Time-Triggered communication on CAN (FTT-CAN). Servers provide fairness among the streams of messages as well as timely message delivery.

Two scheduling mechanisms are presented (S^3 -CAN and PS^2 -CAN) for a centralised media access server (M-Server) to schedule a set of network servers (N-Servers). Both schedulers provide fairness between different N-Servers, and worst-case message response-time formulae are presented for both scheduling mechanisms. The scheduling mechanisms differ in how they handle N-Servers that do not fully use their allocated bandwidth. S^3 -CAN has a worst-case response-time that is proportional to the number of N-Servers in the system, whereas PS^2 -CAN has a worst-case response-time that is proportional to the N-Server's deadline and period.

The second Server-CAN scheduler, PS^2 -CAN, is a Polling Server [LSS97, SSL89] implementation that gives fair bandwidth isolation and

responsiveness among the N-Servers in the system. The strength of such a server implementation is its simplicity.

The first Server-CAN scheduler, S³-CAN, is a more complex server implementation trying to be more responsive in its polling of the N-Servers. S³-CAN is updating its N-Server parameters similarly to the deadline assignment of the Total Bandwidth Server [SB94, SBS95]. However, depending on how the N-Servers are configured with respect to their actual runtime behaviour (i.e., actual message load during runtime), a badly configured set of N-Servers can for S³-CAN introduce blocking which potentially could cause a waste of bandwidth (and therefore affect the worst-case response-time that can be guaranteed for the N-Servers). However, to produce the worst-case scenarios for S³-CAN require a highly unlikely combination of the phase between the message arrival patterns and N-Server period to occur.

The main strength of server-based scheduling for CAN, compared to other scheduling approaches, is that it can give good service to streams of aperiodic messages. Aperiodic messages on native CAN would make it (in the general case) impossible to give any real-time guarantees for the periodic messages sharing the bus. Using Server-CAN, it is possible to schedule for unknown aperiodic or sporadic messages by guessing that they are arriving, and if an erroneous guess is made, not much bandwidth is wasted. This since the STOP message, together with the arbitration mechanism of CAN, allows for detection of when no more messages are pending so that potential slack in the system can be reclaimed followed by the start of scheduling new messages (the next EC) without wasting bandwidth.

Bibliography

- [AB98] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE International Real-Time Systems Symposium (RTSS'98)*, pages 4–13, Madrid, Spain, December 1998. IEEE Computer Society.
- [Abe98] L. Abeni. Server Mechanisms for Multimedia Applications. Technical Report RETIS TR98-01, Scuola Superiore S. Anna, Pisa, Italy, 1998.
- [ABR⁺93] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal*, 8(5):284–292, September 1993.

- [AFF98] L. Almeida, J. A. Fonseca, and P. Fonseca. Flexible Time-Triggered Communication on a Controller Area Network. In *Proceedings of the Work-In-Progress Session of the 19th IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid, Spain, December 1998. IEEE Computer Society.
- [AFF99] L. Almeida, J. A. Fonseca, and P. Fonseca. A Flexible Time-Triggered Communication System Based on the Controller Area Network: Experimental Results. In *Proceedings of the International Conference on Fieldbus Technology (FeT'99)*, Magdeburg, Germany, September 1999.
- [But05] G. C. Buttazzo, editor. *Hard real-time computing systems*. Springer-Verlag, 2nd ed., 2005. ISBN 0-387-23137-4.
- [CAN02] CAN Specification 2.0, Part-A and Part-B. CAN in Automation (CiA), Am Weichselgarten 26, D-91058 Erlangen. <http://www.can-cia.de/>, 2002.
- [CiA96] CiA. CANopen Communication Profile for Industrial Systems, Based on CAL, October 1996. CiA Draft Standard 301, rev 3.0, <http://www.canopen.org>.
- [HL96] C. W. Hsueh and K. J. Lin. An Optimal Pinwheel Scheduler Using the Single-Number Reduction Technique. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS'96)*, pages 196–205, Los Alamitos, CA, USA, December 1996. IEEE Computer Society.
- [ISO93] ISO 11898. Road vehicles – interchange of digital information – controller area network (CAN) for high-speed communication. *International Standards Organisation (ISO)*, ISO Standard-11898, November 1993.
- [ISO00] ISO 11898-4. Road vehicles – controller area network (CAN) – part 4: Time-triggered communication. *International Standards Organisation (ISO)*, ISO Standard-11898-4, December 2000.
- [J1998] SAE J1938. Design/Process Checklist for Vehicle Electronic Systems. *SAE Standards*, May 1998.
- [Kop98] H. Kopetz. The Time-Triggered Model of Computation. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS'98)*, pages 168–177, Madrid, Spain, December 1998. IEEE Computer Society.

- [LK98] M. Livani and J. Kaiser. EDF Consensus on CAN Bus Access for Dynamic Real-Time Applications. In *Proceedings of the 6th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'98)*, Orlando, Florida, USA, March 1998.
- [LKJ98] M. A. Livani, J. Kaiser, and W. J. Jia. Scheduling Hard and Soft Real-Time Communication in the Controller Area Network (CAN). In *Proceedings of the 23rd IFAC/IFIP Workshop on Real-Time Programming*, Shantou, ROC, June 1998.
- [LL73] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):40–61, 1973.
- [LRT92] J. P. Lehoczky and S. Ramos-Thuel. An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks Fixed-Priority Preemptive systems. *Proceedings Real-Time Systems Symposium*, pages 110–123, December 1992.
- [LSS97] J.P. Lehoczky, L. Sha, and J. Strosnider. Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. In *Proceedings of 8th IEEE Real-Time Systems Symposium (RTSS'87)*, pages 261–270, San Jose, California, USA, December 1997. IEEE Computer Society.
- [Nat00] M. Di Natale. Scheduling the CAN Bus with Earliest Deadline Techniques. In *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS'00)*, pages 259–268, Orlando, Florida, USA, November 2000. IEEE Computer Society.
- [NNH05] T. Nolte, M. Nolin, and H. Hansson. Real-time server-based communication for CAN. *IEEE Transactions on Industrial Informatics*, 1(3):192–201, August 2005.
- [Nol06] T. Nolte. *Share-driven scheduling of embedded networks*. PhD thesis, Department of Computer and Science and Electronics, Mälardalen University, Sweden, May 2006.
- [PA00] P. Pedreiras and L. Almeida. Combining Event-triggered and Time-triggered Traffic in FTT-CAN: Analysis of the Asynchronous Messaging System. In *Proceedings of the 3rd IEEE International Workshop on Factory Communication Systems (WFCS'00)*, pages 67–75, Porto, Portugal, September 2000. IEEE Industrial Electronics Society.

- [PA01] P. Pedreiras and L. Almeida. A Practical Approach to EDF Scheduling on Controller Area Network. In *Proceedings of the IEEE/IEE Real-Time Embedded Systems Workshop (RTES'01) at the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*, London, England, December 2001.
- [PG93] A. K. Parekh and R. G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.
- [SAWJ⁺96] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and G. Plaxton. A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems. In *Proceedings of 17th IEEE Real-Time Systems Symposium (RTSS'96)*, pages 288–299, Los Alamitos, CA, USA, December 1996. IEEE Computer Society.
- [SB94] M. Spuri and G. C. Buttazzo. Efficient Aperiodic Service under Earliest Deadline Scheduling. In *Proceedings of the 15th IEEE Real-Time Systems Symposium (RTSS'94)*, pages 2–11, San Juan, Puerto Rico, December 1994. IEEE Computer Society.
- [SB96] M. Spuri and G. C. Buttazzo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. *Real-Time Systems*, 10(2):179–210, March 1996.
- [SBS95] M. Spuri, G. C. Buttazzo, and F. Sensini. Robust Aperiodic Scheduling under Dynamic Priority Systems. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, pages 210–219, Pisa, Italy, December 1995. IEEE Computer Society.
- [SLS95] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, January 1995.
- [SSL89] B. Sprunt, L. Sha, and J.P. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Systems. *Real-Time Systems*, 1(1):27–60, 1989.
- [TB94] K. W. Tindell and A. Burns. Guaranteed Message Latencies for Distributed Safety-Critical Hard Real-Time Control Networks. Technical Report YCS 229, Dept. of Computer Science, University of York, York, England, June 1994.

- [TBW95] K. W. Tindell, A. Burns, and A. J. Wellings. Calculating Controller Area Network (CAN) Message Response Times. *Control Engineering Practice*, 3(8):1163–1169, 1995.
- [THW94] K. W. Tindell, H. Hansson, and A. J. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). In *Proceedings of 15th IEEE Real-Time Systems Symposium (RTSS'94)*, pages 259–263, San Juan, Puerto Rico, December 1994. IEEE Computer Society.
- [TLS96] T.-S. Tia, J. W.-S. Liu, and M. Shankar. Algorithms and optimality of scheduling soft aperiodic requests in fixed-priority preemptive systems. *Real-Time Systems Journal*, 10(1):23–43, 1996.
- [ZS95] K. M. Zuberi and K. G. Shin. Non-Preemptive Scheduling of Messages on Controller Area Network for Real-Time Control Applications. In *Proceedings of the 1st IEEE Real-Time Technology and Applications Symposium (RTAS'95)*, pages 240–249, Chicago, IL, USA, May 1995. IEEE Computer Society.

Chapter 16

A pipelined fiber-ribbon ring network with heterogeneous real-time support

By **Carl Bergenhem**[‡] and **Magnus Jonsson**[†]

[‡]Bergenhem Konsult
Email: carlb@bergenhem.com

[†]Laboratory for Computing and Communication
Halmstad University
Email: magnus.jonsson@ide.hh.se

This paper presents a fiber-optic ring network with support for heterogeneous real-time communication. The CCR-EDF (Control Channel Ring network with Earliest Deadline First scheduling) network is an optical fibre-ribbon pipelined ring network with a separate channel for network arbitration. The medium access protocol, that uses the control channel, provides low-level support for hard real time traffic and group communication functions such as barrier synchronisation and global reduction. The topology of the network is a pipelined unidirectional fibre-ribbon ring that supports several simultaneous transmissions in non-overlapping segments. Access to the network is divided into time-slots. In each slot the node that has the highest priority message is permitted to transmit. The novel medium access protocol uses the deadline information of individual packets, queued for sending in each node, to make decisions, in a master node, about who gets to send. This feature of the medium access protocol gives the network the functionality for earliest deadline first scheduling. Different classes of traffic are supported for the user. These are guaranteed

logical real-time channels (LRTC), best effort (BE) channels and non real-time (NRT) traffic.

16.1 Introduction

The CCR-EDF (Control Channel Ring network with Earliest Deadline First scheduling) network is an optical fibre-ribbon pipelined ring network with a separate channel for network arbitration. The medium access protocol, that uses the control channel, provides low level support for hard real time traffic and group communication functions such as barrier synchronisation and global reduction. [1]. The basic network structure is depicted in Figure 16.1. The novel medium access protocol uses the deadline information of individual packets, queued for sending in each node, to make decisions, in a master node, about who gets to send. The new protocol may be used with a previously presented network topology; the control channel based fibre ribbon pipeline ring (CC-FPR) network [2].

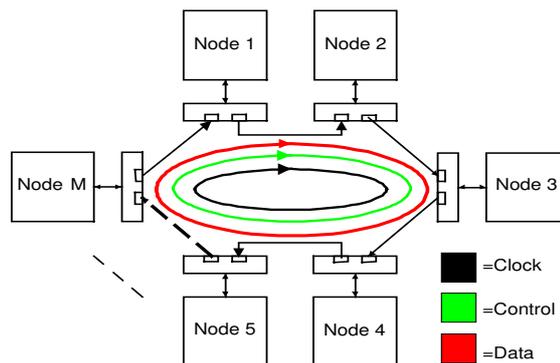


Figure 16.1: A Control Channel based Fiber Ribbon Pipeline Ring network

In addition to local area networks, the CCR-EDF network is also suitable for parallel and distributed real-time systems. Application examples are future radar signal processing systems, distributed multimedia systems, satellite imaging and other image processing applications. A typical example is the radar signal processing system described in [3], [4]. Often, these systems are classified as real-time computer systems. In a real-time computer system, correct function depends both on the time at which a result is produced and on its accuracy [5]. In distributed real-time systems, the interconnection network is a very important part of the computer system. Often, guaranteeing real-time services are much more important in these systems than performance, e.g., average latency.

Radar signal processing (RSP) in an airborne environment is an area of communication where the network may be applied. The computation in RSP is done in a distributed fashion. Therefore the communications network is a vital part of a RSP system. Studies have shown that the radar processing algorithms map suitably on a ring topology such as the one discussed in this report [6].

The paper also presents results of simulations where the network has been tested together with defined cases. These cases are first defined and presented. The first case is a radar signal processing (RSP) application. The RSP system has a number of different requirements depending on the application, but the algorithms comprise mainly linear operations such as matrix-by-vector multiplication, matrix inversion, FIR-filtering, DFT etc. In general, the functions will work on relatively short vectors and small matrices, but at a fast pace and with large sets of vectors and matrices. Even if the incoming data can be viewed as large 3-dimensional data cubes, it is quite easy to divide the data into small tasks that each can be executed on a separate processing element in a parallel computer. Actually, the computational demands are so high that it implies that parallel computer systems are needed. In turn, the performance of parallel computers is highly dependent on the performance of their interconnection networks, especially in data intensive applications like radar signal processing. In addition to the transfer of radar data, there are control functions and registration functions controlling and monitoring the system. Both data and control traffic have real-time demands. Results of the simulations indicate that the CCR-EDF network works well with the RSP application studied. To show the networks broad applicability, a case is also defined, not simulated, for the network used within large IP-routers. When building large IP routers multiple racks are used (hundreds or thousands of input /output –ports in total). The intelligence (to take routing decisions etc) can be implemented either in racks with line cards (interfacing to ingoing and outgoing physical channels) and/or in centralized routing racks. These racks must be interconnected to realise the router. For flexibility in design, operation, and maintenance of the system, a network such as CCR-EDF (or network with similar properties i.e. real-time support etc.) is desirable. The complete system takes the form of a distributed router.

The article is organised as follows. The network itself and the protocol are described in Sections 2 and 3, respectively. Two cases, the radar signal processing case and the large IP-router case are presented in Sections 4 and 5 respectively. The radar signal processing case, which is simulated in the later sections, is further defined together with the simulator setup, in Section 6. Three different simulations are presented and discussed in Section 7. A discussion about how throughput is defined is presented in Section 8. Finally, conclusions are drawn in Section 9.

16.2 The CCR-EDF network architecture

Motorola OPTOBUSTM bi-directional links (or similar) with ten fibres per direction are (in this report) assumed to be used but the links are arranged in a unidirectional ring architecture where only $\lceil N/2 \rceil$ bi-directional links are needed to close a ring of N nodes. All links are assumed to be of the same length. The increasingly good price/performance ratio for fibre-ribbon links indicates a great success potential for the proposed type of networks.

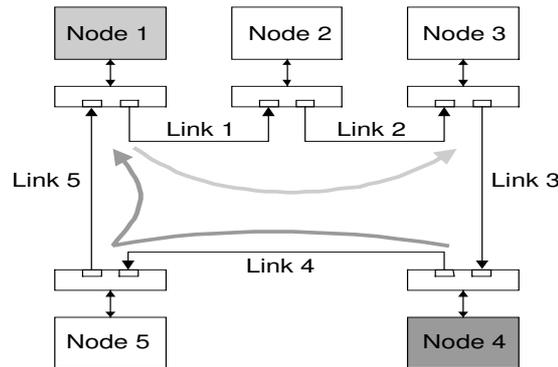


Figure 16.2: Example where Node 1 sends a single destination packet to Node 3, and Node 4 sends a multicast packet to Node 5 and Node 1.

The ring can dynamically (for each slot) be partitioned into segments to obtain a pipeline optical ring network [7]. Several transmissions can be performed simultaneously through spatial bandwidth reuse, thus achieving an aggregated throughput higher than the single-link bit rate (see Figure 16.2 for an example). Even simultaneous multicast transmissions are possible as long as multicast segments do not overlap. Although simultaneous transmissions are possible in the network because of spatial reuse, each node can only transmit one packet at a time and receive one packet at a time.

For the following discussion, the term master node is exchangeable with “the node with clocking responsibility etc.”. That is, the master node also clocks the network. The clocking strategy functions as follows. During a slot, the node that has the highest priority message, according to the arbitration process, has the responsibility to clock the network. In the following slot, the clocking responsibility is handed over to the node that has the highest priority message in that slot. This may be another node or the same one as in the previous slot. Thus clock hand-over is always done in accordance with the result of the medium access arbitration process, described further in the next section. The result of the arbitration process is knowledge of all messages at the heads of

the local queues in all nodes, and therefore also knowledge about which node has the highest priority message in the entire system. The current master distributes this information to all nodes. A distribution packet is sent so that the end of the packet corresponds with the end of the slot. This implies that, when the master stops the clock at the end of the slot, all nodes have the information that they need to perform clock hand-over that takes place in the gap between slots. The node that has highest priority in the coming slot detects when the clock signal is stopped and assumes the master role. The highest priority node knows that it will be master because of the information received in the distribution phase packet.

Since the node that is master, also the node that has the highest priority message, has responsibility for generating the clock, then there cannot occur a situation where the node cannot send its message. This is because the node will at most send $N - 1$ hops (where N is the number of nodes) and will never have to transmit past a master, i.e. cross the clock break at the master node. In the CCR-EDF network, access to the network is divided into time-slots. There is no concept of cycle since a cycle cannot be defined. In other cases a cycle would be e.g. when all nodes have been master once. However, in this network the master role is not shared equally among nodes but is given to the node with the highest priority message.

16.3 The CCR-EDF medium access protocol

The medium access protocol has two main tasks. The first is to decide and signal which packet(s) is to be sent during a slot. The second task is that the network must know exactly which node has the highest priority message in each slot. This is to be able to perform clock hand-over to the correct node. Therefore, this information is included as an index in the distribution phase packet.

The two phases of medium access are collection phase and distribution phase (see Figure 16.3). As can be seen, medium access arbitration occurs in the time slot prior to the actual transmission. The protocol is time division multiplexed into slots to share access between nodes. A disadvantage with the CC-FPR protocol presented in [2] is that a node only considers the time constraints of packets that are queued in it, and not in downstream nodes. As an example (see Figure 16.2), Node 1 decides that it will send and books

Links 1 and 2, regardless of what Node 2 may have to send. This means that packets with very tight deadlines may miss their deadlines. The novel network presented here does not suffer from this problem. In the collection phase, the current master initiates by generating an empty

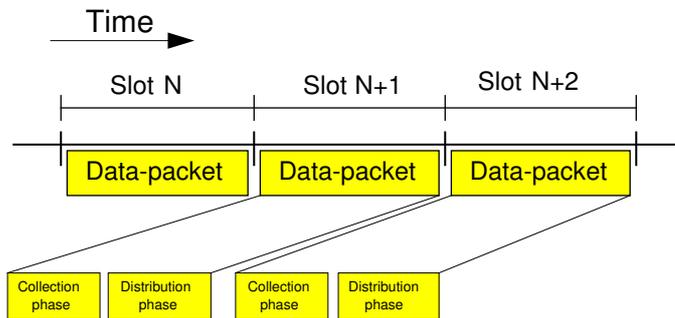


Figure 16.3: The two phases, collection and distribution, of the TCMA protocol. Notice that the network arbitration information, for data in slot $N + 1$, is sent in the previous slot, slot N .

packet, with a start bit only, and transmits it on the control channel. Each node appends its own request to this packet as it passes. The master receives the complete request packet (see Figure 16.4) and sorts the requests according to priority (urgency).

Type	Start	Prio	Link reservation field	Destination field	Prio	Link reservation field	Destination field	
Nbr of bits	1	5	node i	node i	5	node $i+1$	node $i+1$	etc.
			N	N		N	N	

Figure 16.4: Contents of the collection phase packet. The figure shows that one request per node make up the complete packet. Each request consists of three fields, the primary field, the link reservation field and the destination field.

The network can handle three classes of traffic: logical real-time connection, best effort, and non-real time. Which class of traffic that a certain message belongs to, is signalled to the master with the priority field in the request (see Figure 16.4). Table 16.1 shows the allocation of the priority field to each user service in the network. The time until deadline (referred to as laxity) of a message is mapped, with a certain function, to be expressed within the limitation of the priority field size, see Table 16.1. This applies to both logical real-time connection and best effort traffic. A shorter laxity of the packet implies a higher priority of the request. The result of the mapping is written to the priority field. One priority level is reserved (0 in the proposed implementation of the protocol) and used by a node to indicate that it does not have a request. If so, the node signals this to

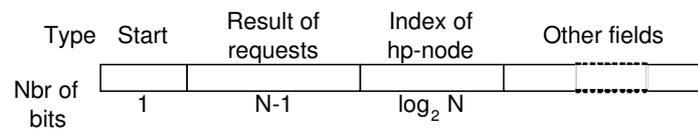


Figure 16.5: The modified TCMA distribution phase packet. The field labelled “index of hp-node” contains the index of the node that has the highest priority message.

0	Nothing to send
1	Non-Real Time
2–16	Best Effort
17–31	Logical real-time connection

Table 16.1: The allocation of priority levels to user services. A higher priority within the traffic class implies shorter laxity and a more urgent message.

the master by using the reserved priority level and also writes zeros in the other fields of the request packet. Observe that messages that are part of LRTCs (logical real-time connections) always have higher priority than any other service. However, a possible situation, considering spatial reuse, is that a best effort message uses the spatially reused capacity and may be transmitted simultaneously as a logical real-time connection message. The best effort message does not affect the logical real-time connection message. Observed locally in a node, best effort messages will only be requested to be sent if there is no logical real-time connection message queued. The same applies to non real-time message. They are only sent if there are no best effort and no logical real-time connection messages.

Request priority is a central mechanism of the CCR-EDF protocol. Deadlines are mapped with a function to priority. For the following discussion, a logarithmic mapping function is assumed. This mapping gives higher resolution of laxity, the closer to its deadline a packet gets. Further discussion of deadline to priority mapping function is out of the scope of this paper.

When the completed collection phase packet arrives back at the master, the requests are processed. There can only be N requests in the master, as each node gets to send one request per slot. The list of requests is sorted in the same way as the local queues. The master traverses the list, starting with the request with highest priority (closest to deadline)

and then tries to fulfil as many of the N requests as possible, avoiding overlapping transmission segments.

The second phase, the distribution phase, is described as follows. The master sends a packet, see Figure 16.5, on the control channel that contains the result of the arbitration. This is either acceptance or denial of a node's request and also which node contains the highest priority message in that slot. All nodes read the message. A request was granted if the corresponding bit in the "request result field" of the distribution phase packet contains a "1". The protocol also has a feature to permit several non-overlapping transmissions, that is grant several requests, during one slot. This function is called spatial reuse and is used during run-time. Observe that the distribution phase packet also may contain other information such as acknowledgement for transmission, group communication etc. These are further described in [8] and will not be part of the discussion here.

The addition of an index that points to the node that has highest priority in the current coming slot, see Figure 16.4, enables all nodes to know who will have the highest priority message in the coming slot and that the node will assume the role as master and clock the network. The index field needs to be $\lceil \log_2 N \rceil$ bits wide to represent numbers up to N . When all nodes have received the distribution phase packet, and hand-over has taken place, the new slot commences and data may begin to flow in the data channel.

16.4 A radar signal processing case

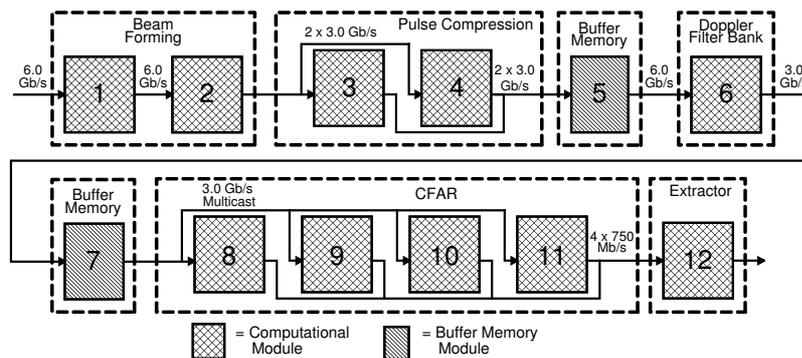


Figure 16.6: An irregular pipeline chain. Data flow between the modules in the radar signal processing chain.

In Table 16.2, there are three different communication patterns for data traffic. The difference between the cases is how the processing takes place and thus, how the bulk of the data is communicated. The

	Communication pattern	Delay Bound	Traffic amount	Arrival model
Control traffic	Out: broadcast In: many-to-one (Master / Slave)	100 ms per direction (guaranteed or highest priority)	1 kByte	Periodic @ 1 ms, for both directions
Data traffic	a)Irregular pipeline	1 ms for each packet in the data cube. *3)	96 Mbit data cubes @ 62.5 Hz *7)	A new data cube arrives every 16 ms *1)
	b)Straight pipeline	0.5 ms *4)	96 Mbit data cubes @ 62.5 Hz *7)	*1)
	c)SPMD	A corner turn must take place within 4ms(soft deadline)	96 Mbit data cubes *5) 60 kbit messages in CT *6)	*1) *2) Two corner turns will occur every 16 ms.
Other traffic	Assume broadcast for worst case	Non real-time. No bound but a certain capacity is needed.	100 Mbit/s is assumed to be representative	Periodic with an average period of 50 ms is assumed to be representative

Table 16.2: Assumptions for the traffic in the RSP case study.

communication pattern of the control traffic is independent of the data traffic; all nodes, despite processing model, are assumed to have central control. Generally, in the pipeline processing models there will be more than one data cube in processing at a time and only one at a time in the SPMD model. The three communication patterns are briefly discussed below.

Figure 16.6 depicts an example of an irregular pipeline. The difference between the two mentioned pipelines in [3] and [10] is the number of nodes in the processing chain and amount of the incoming data. Certain steps of the RSP algorithm are much more computationally intensive than others (the CFAR step in Figure 16.6 is an example). It is assumed for this processing mode that all nodes are equally powerful. Therefore some steps will be performed on more than one node and communication services to support this are required for efficiency. The irregular pipeline assumes that there will be multicast communication between certain nodes in the processing chain. Multicast communication is shown in Figure 16.6 where Node 7 multicasts to Nodes 9 through 11. Also, a many-to-one service is required when Node 12 collects the information from the previous four stages. One-to-many communication is also included in the chain.

For the straight pipeline, we assume that the bulk of the data communication is to the next neighbour (see Figure 16.7). Here it is assumed that processing is done in a pipelined fashion, such that the processing

steps are divided among all modules equally. This is purified version only for evaluation purposes. For the straight pipeline we assume a network with 20 identical modules and a required latency for the whole pipeline of 100 ms. This gives 0.5 ms allocated for communication per data cube per module and a total of 90 ms processing time, per data cube.

In the SPMD (same program multiple data) processing model, all nodes take part in the same processing step at a time, where each node works on a part of the data cube. When the processing step is complete, the data is redistributed, if needed, to the appropriate nodes for the next step of processing. An SPMD processing model is depicted in Figure 16.8. It is assumed that each node has I/O capabilities for communication with external devices, e.g., for data from the antenna and for sending results that are ready.

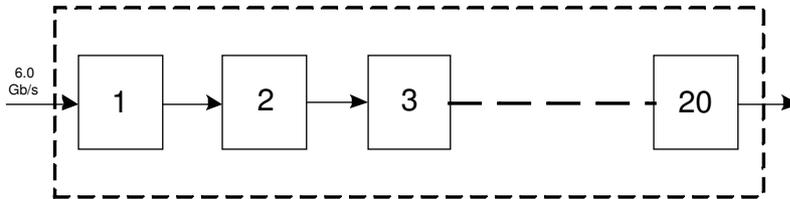


Figure 16.7: The straight pipeline

The main studied application is radar signal processing (RSP). In the main and supporting algorithms we can identify several different types of traffic. Examples of these and what they may require are:

- The radar signal data itself has soft real-time requirements, but expected performance must be determinable.
- The control traffic has, for obvious reasons, hard real-time requirements. Also real-time debugging requires a guaranteed service.
- Other general traffic such as logging and long term statistics do not have any real time constraints at all. System start-up, bootstrapping, firmware upgrading are tasks that don't require real time support. However, some kind of performance analysis is needed in order to design for desired performance.

These traffic types are further explained in Table 16.3, while some definitions and explanations are made later in this chapter. Some notes referred to in the table are:

*1) All three data traffic models will have periodic arrival of data cubes with a period of 16 ms.

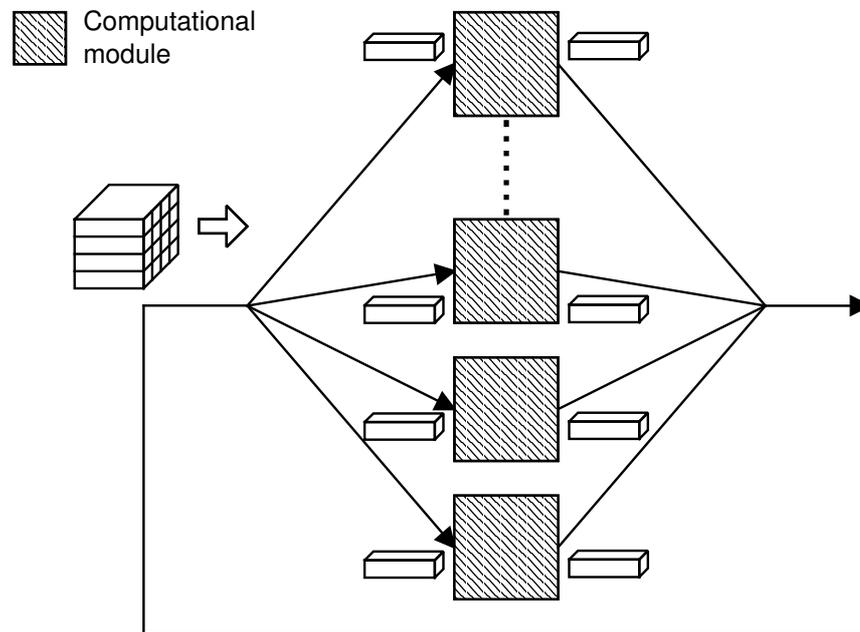


Figure 16.8: If the SPMD model is used, all modules work together on one stage in the signal processing chain at a time.

*2) New data cubes and results are received and sent on dedicated I/O.

*3) The communication delay bound (deadline) for the whole pipeline is 10 ms (10 % of 100 ms total latency including processing), with a new whole data cube arriving every 16 ms. The (soft) deadline for each packet in the data cube is 1 ms ($\approx 10 \text{ ms} / 12 \text{ steps}$). Priority based or soft deadline based service class is assumed (guaranteed service might be implemented on a higher level).

*4) The communication delay bound for the whole pipeline is 10 ms, with a new whole data cube arriving every 16 ms. The (soft) deadline for each packet in the data cube is 0.5 ms ($10 \text{ ms} / 20 \text{ steps}$). Priority based or soft deadline based service class (guaranteed service might be implemented on a higher level).

*5) 96 Mbit data cubes arriving with a rate of 62.5 Hz, i.e., a new data cube each 16 ms.

*6) $96 \text{ Mbit} / (40 \cdot 40) = 60 \text{ kbit}$ messages in a corner turn (CT) with the all-to-all communication pattern (assuming that all 40 nodes are both source and destinations in the corner turn).

*7) It may be possible to start to send parts of the data cube before all processing is completed.

	Communication pattern	Delay bound	Traffic amount	Arrival model
Control traffic	Evenly distributed	10 ms max. latency, (Guaranteed or at least highest priority)	5-10 % of the amount of data traffic	Internet communication *1) *3)
Data traffic	Evenly distributed	*2), 100 ms max and 20 ms average latency (priority based, i.e., no deadline sorting of cells)	In the order of Tbit/s	Bursty. *1) A number of consecutive data cells will be send after one control cell.
Other low priority (e.g. routing table updates)	Evenly distributed	None, but certain capacity may be needed. Upper level protocols may handle this with time-outs.	At most 1 % of total traffic	More or less periodic

Table 16.3: The various traffic types in the large IP-router network.

Table 16.1 specifies parameters of the traffic in the RSP network. Parts of the assumptions presented in Table 16.1 are also found in [3] [10]. For both pipelined RSP chains (straight and irregular), the maximum latency of a result is 100 ms, including communications and processing. This latency is assumed to be composed of ca. 10 % communications delay. These two assumptions can be found in [3]. Both directions of the control traffic are periodic with a period of 1 ms. It is assumed that processing is co-ordinated in a master / slave fashion and that there is a node in the network that acts as master over the other nodes that do the processing. These latter nodes are referred to as slaves.

It is assumed that the RSP algorithm is the same for all the three different communication patterns for data traffic. The RSP algorithm has two corner turns per batch. It is also assumed that the incoming data rate from the antenna is 6.0 Gb/s. The amount of traffic depends on the representation of numbers, precision, etc. [3] [10].

The arrival model for data traffic case a) and b) is related to how often a new data cube is accepted and to the pipelined fashion. However, data is sent with finer granularity (minimum ca 1 500 bits per message), than a whole data cube between the nodes. Despite this, we can assume only target at a throughput that can deliver the whole data cube in time, and at a maximum delay for each packet. A new data cube arrives equally often in the SPMD case, but utilises separate I/O.

16.5 A Large IP router case

A rule of thumb concerning delay in routing is that the maximum delay from input port to output port should be 1 ms. In a router-architecture such as in Figure 16.4, the latency over the SAN is assumed to be 100 μ s maximum. We assume that each router module consist of both input and output ports. A summary of the different traffic types in the interconnection network is presented in Table 16.3, where some notes referred to in the table are:

*1) We assume that each IP-datagram is split into fixed size cells at the source port and sent across the interconnection network to the destination. Therefore, there will be one control cell for each IP-datagram followed by a number of data cells that comprise the split IP-datagram. The arrival of control traffic is linked to the arrival of the IP-datagrams. Possibly, information about several IP-datagrams may be transferred in the same control cell.

*2) Data cells of IP-datagrams that exceed their deadlines are not rejected straightaway (soft deadlines). However some other action is required so that a stream is given higher priority to increase probability of keeping deadlines. The “action” may be to raise the priority of the IP-datagrams, belonging to a stream, that is missing its deadlines. Feedback to the source of data cells (in the interconnect network) may be required. Data traffic may have different priorities within the data traffic class. Priority based service-class are assumed but where guaranteed services might be implemented on a higher level in the distributed router.

*3) In general, the arrival of IP-datagrams in a LAN environment is considered to be self-similar [11]. The traffic in the studied router may be similar to this. The arrival model of the control traffic cells is more directly related to arrival of IP datagrams than the arrival model of data traffic cells.

As can be seen in Table 16.3, all nodes communicate with all other nodes with equal probability. The data units in the network are called cells and are 48 – 64 bytes long. Speed is of concern in this application. Checking deadlines of e.g. control traffic cells consumes much time. Instead we assume that it is acceptable if cells are treated in FCFS (first come, first served) order but are differentiated by the three traffic classes in Table 16.2, and by priority in the case of data traffic. More or less predictable throughput is also assumed, depending on which traffic class.

The control packets contain information about how an IP-datagram requires to be handled at the destination module and how the source and destination ports of the interconnect network should be set up for the following data packets. Information in the control cells also co-ordinate

functions in the interconnect network and are therefore not necessarily directly associated with data cells. The data cells will contain some ID that associates it with a control cell that arrived previously, together with destination and source ports in the interconnect network

16.6 Case definition and simulator setup

In all simulations the channel “wiring” is the same, i.e. the same definition of logical channels are used and each channel goes to the same node(s). However, the load of a set of channels may change for each data point. In [9], the full case definition is described. All three simulations are based on the “straight pipeline” case, as described in [9]. Each slot (smallest simulated time unit) is equivalent to $1 \mu\text{s}$ and corresponds to 1 kByte of data. Table 16.2 contains values and assumptions for three cases. The “straight pipeline” RSP case is assumed in the following simulations.

The traffic in the RSP case definition consists of three main types (in decreasing order of timeliness requirements): Control traffic, data traffic and other traffic. These types are conveniently services with the three data transports services offered by the CCR-EDF network protocol. The control traffic requires guarantees of timeliness and is therefore is serviced by the LRTC service. Typical qualities of the three types of traffic in the RSP case definition, found in [9], are summarised in Table 16.2.



Figure 16.9: The “wiring” of BE-channels. Note that each arrow is a unicast channel.

Figure 16.9 depicts the “wiring” of the Best effort (BE) channels. These communicate the bulk of the data, in a pipelined fashion from node to node. The traffic is periodic with a period of 16 ms, i.e. 16 000 slots, and the default payload is 10 MByte, i.e. 10000 slots. During simulation, the payload may be varied from 1 MByte – 16 Mbyte. Best effort traffic has lower priority than LRTC, but higher priority than NRT traffic, see also Table 16.1. If the BE channels are used by themselves in a system, the set of BE channels have a maximum throughput of 10 packets per slot with 16 Mbyte payload. This is the theoretical limit and is also found by simulation.

Figure 16.10 depicts the wiring of LRTCs. The case study has a fixed set of LRTCs. These are organised as master / slave communication.

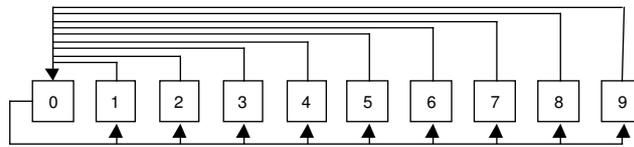


Figure 16.10: The "wiring" of the LRTCs. Node 0 is the master node and the others are slaves.

The data is control information and therefore the amount is less than BE traffic. The traffic is periodic with a period of $100g \mu s$ i.e. 100 slots, and default payload is 1 kByte, i.e. one slot. During simulation, the payload may be varied from 1 kByte to 16 kByte. The LRTC traffic class is a guaranteed service with the highest priority. When used by themselves in a system, the set of LRTCs have a maximum throughput of 1.1 packets per slot per channel with 11 kByte payload per packet. This result is found in simulation 3 performed in Section 7.3 and is also further explained here.

	[packets per slot]
Total throughput with the default load of the RSP case (Observe that the system is not saturated here)	5.97
Default LRTC throughput	0.10
Default BE throughput	5.26
Maximum throughput of LRTC traffic only	1.11
Maximum throughput of BE traffic only	10

Table 16.4: Important results, values taken from simulations. Observe that packets per slot refers to any and all types of traffic in the system unless otherwise stated. Observe also that more complex combinations such as maximum BE throughput with default LRTC load are not stated here.

The NRT (non real time) traffic is not organised in channels with fixed destinations. From each source the destination is uniformly distributed. Every node is a NRT source of equal load. The deadline of NRT traffic is fixed to 100 slots. In the simulator, NRT traffic arrival is always poisson distributed. NRT traffic always has the lowest priority. When used by itself in a system, the NRT traffic will have a maximum throughput of 2 packets per slot and will use all available capacity.

All “wirings” of channels, etc. are done according to the case study definition [9]. The case definition also states the traffic loads under normal operation. With the mix of traffic described in the case study, the total throughput (for all traffic types) is 5.97 packets per slot (see Table 16.4). In Simulation 2, Section 7.2 it will be seen that the load of the network can be increased to achieve an even higher throughput.

Each “simulation” consists of several (usually 16) data points. Each data point is an execution of the simulator with fixed parameters. To attain the curve, a parameter is varied and several runs later, the result is the data points that make up the curves in the presented figures.

Periodic traffic channels in the simulator are treated as follows. All traffic that a channel will send during one period will be generated and queued for sending at the start of the period. There is no “intelligent” functionality that smoothes the incoming traffic over the whole period. This lack of smoothing affects the maximum latency that a packet endures during a period. If the incoming traffic was smoothed, then the maximum would be closer to the average latency.

16.7 Simulations

In this section three main simulations are presented. Other simulations have also been done. These may be referred to only briefly, e.g. as numeric results, and not presented in full with diagram and discussion.

The first simulation concerns the latency and packet loss of BE traffic under increasing load of BE traffic. The second and third simulations have the mixture of traffic described in the case definition. In the second and third, the load of the BE traffic and LRTC traffic, respectively, is varied while the other is fixed. The effect on the mixture of traffic is studied. Important results from the simulations are summarised in Table 16.4. The “default” throughputs in the table are according to the load-levels found in the case study [9]. Observe that NRT traffic in the RSP case always strives to use all left over available capacity. Therefore it is meaningless to cite the throughput of this class.

16.7.1 Simulation 1

The “straight pipeline” of the case study definitions was implemented in the simulator and tested with different loads of BE traffic. The network setup for the simulation is as follows:

- All LRTCs at constant load: 1 data packet with a period of 100 slots period (the same as the RSP case definition).
- The load of each BE channel is varied in the interval 1 MByte – 16 MByte (6 % - 100 %).

- No Non Real time traffic in the system

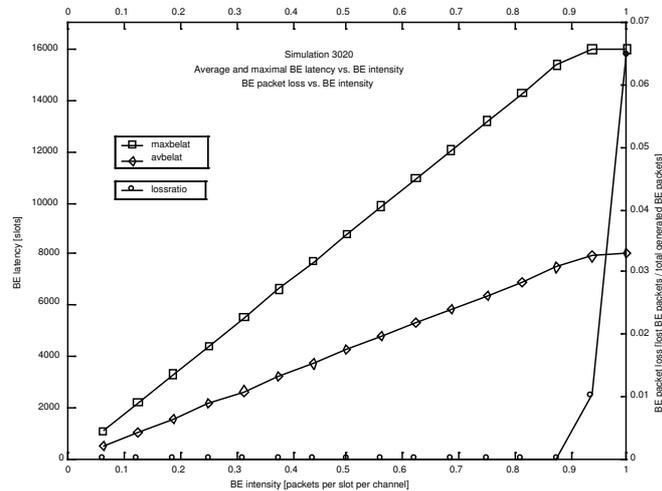


Figure 16.11: The figure shows the BE-traffic latency and BE-packet loss vs. the intensity of the BE-traffic. Regarding BE latency, observe that the deadline of the BE packets is 16000 slots.

The result of the simulation is shown in Figure 16.11. The figure shows three plots: Maximum and average BE latency vs. BE intensity and BE packet loss vs. BE traffic intensity. See Section 5.4 for a deeper discussion on traffic intensity. The maximum BE latency is the maximum latency that any BE packet was subject to during the simulation (for the respective intensity of BE traffic). As the BE traffic intensity increases, so does the maximum latency. The trend holds until the network cannot accept more BE traffic. At maximal BE intensity, 1 packet per slot per channel, the theoretical throughput (with no other traffic in the system) is 1 packet per slot per channel (total of 10 packets per slot for all BE channels). However, in the simulation, higher priority traffic also contends for capacity (LRTC traffic), and the throughput of BE traffic is therefore lower than the theoretical value. This can be observed since packets are lost at high intensity levels of BE traffic. At this point the trend of the latency curve changes and BE packets begin to be lost. The packet latency curve levels out (does not continue to increase) at a value of 16 000 slots. This is because the latency cannot be higher than the deadline of the packets, which is 16 000 slots. When a packet is queued for longer than its deadline, the packet is removed and considered lost. A maximum latency of 16 000 slots might seem to

be long but one should remember that 16 Mbyte of data translates to 16 000 needed slots / packets per channel.

Regarding BE intensity, observe that the x-axis in Figure 16.11 indicates the ratio per BE channel, not total BE intensity. The total BE intensity would at its peak approach 10, i.e. the number of nodes in the system. Observe also that the average latency of the BE traffic is always roughly half that of the maximum throughput.

16.7.2 Simulation 2

In this simulation, all three types of traffic are generated during the simulation. How the traffic types effect each other is studied. In short the BE traffic is constant and the LRTC traffic is varied. The network set-up for the simulation is as follows:

- The LRTC set and behaviour of the set is the same as in simulation 1, i.e. constant.
- The set of best effort channels and behaviour of these are the same as in simulation 1, i.e. varied in the interval 1 MByte – 16 Mbyte (6 % - 100 %).
- The intensity of NRT data is enough to saturate the network. This means that although the other traffic classes have priority, NRT traffic will always be sent as soon as there is an opportunity. The intensity of the NRT traffic is constant throughout the simulation.

Figure 16.9 shows the result of the simulation. Observe that the concept of “throughput ceiling” is dependent on the traffic pattern, (see Section 8). In the figure, it can be seen that the highest priority traffic (LRTC) is not affected by the increasing level of BE traffic. Also note that the throughput of the NRT traffic decreases as the intensity of BE increases. In other words, prioritisation of different traffic classes in the simulator works as specified in the protocol.

As can be seen in the figure, the maximum total throughput (before any BE traffic is dropped) is approximately 7.7 packets per slot. The total throughput is the combined throughput of all traffic types.

16.7.3 Simulation 3

In this simulation, all three types of traffic are generated during the simulation. How the traffic types effect each other is studied. In short the BE traffic is constant and the LRTC traffic is varied. The network setup for the simulation is as follows:

- The load of the BE set is constant at 4 000 packets (4 MByte) per period of 16 000 slots (16 ms), (less load than the default RSP case).
- The load of LRTC traffic is varied between 1-16 KBytes per channel with a period of 100 slots.
- The intensity of NRT data is enough to saturate the network. This means that although the other traffic classes have priority, NRT traffic will always be sent as soon as there is an opportunity. The intensity of the NRT traffic is constant throughout the simulation.

Figure 16.13 shows the result of the simulation. Observe again that the concept of “throughput ceiling” is dependent on the traffic pattern, (see Section 8). As can be seen in the figure, the maximum total throughput (before any LRTC traffic is dropped) is approximately 2.8 packets per slot. Observe that in a real implementation of the network there would be admission control of LRTC traffic. Thus it would be impossible to reach a level where LRTC traffic is lost. The total throughput is the combined throughput of all traffic types. The maximum throughput of LRTC traffic (before packets begin to be dropped) is approximately 1.1 packets per slot per channel. This can be found in Figure 16.13. When LRTC traffic begins to be dropped, the payload is 11 kByte per period, i.e. the LRTC intensity is 0.11 packets per slot per channel (11 packets / 100 slots per channel). Here the LRTC throughput is 1.1 packets per slot.

As can be seen in the figure, the total throughput drops as the LRTC intensity increases. This is because there is decreasingly less traffic in the system that takes advantage of spatial reuse (pipelining), i.e. BE traffic. This phenomenon is further discussed in Section 8. As expected, throughput for the two lower priority traffic classes decrease as LRTC intensity is increased.

16.8 Discussion on throughput ceiling

In simulation 2 and 3 a concept of total system throughput is used. This is a measure of the number of data packets that are sent during each slot. Each data packet takes one slot to transmit. When comparing the throughput of the different traffic classes, it must be taken into account that the classes have different traffic patterns and therefore have different throughput ceilings. This “100%”-level varies depending on the traffic pattern. In a system, the throughput ceiling is constant and does not change unless the traffic pattern does so.

The following example illustrates this. The worst case is when all communication is destined one node upstream. Here the throughput

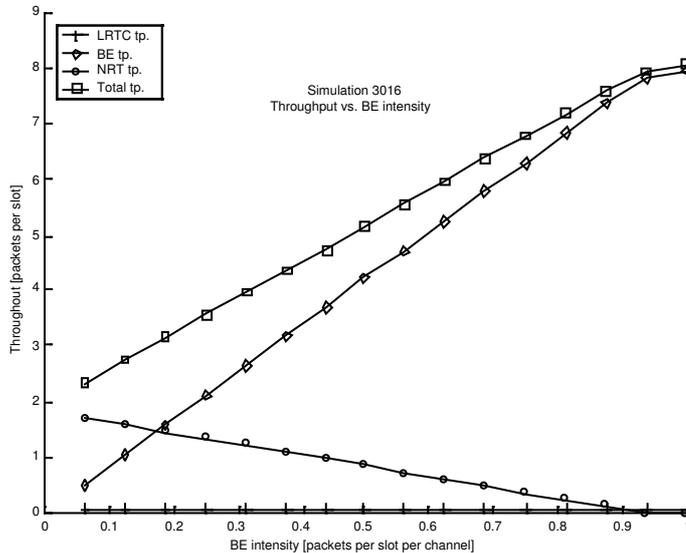


Figure 16.12: The throughput of the different traffic classes change as the BE intensity is increased. LRTC traffic load is constant.

can be at most one. I.e. all traffic is destined to one node upstream, and no traffic is destined to any other node in the network. Therefore no pipelining of non-overlapping transmissions can take place. The best case is when all communication is destined one node down stream. Here the throughput can be at most N , where N is the number of nodes in the network. In this case, the pipelining capability of the network is fully utilised. For traffic that is well-specified in channels, it is easy to find a value for throughput. However, if traffic is, e.g. poisson distributed, then the throughput can only be known statistically. The four different scenarios of data communications patterns are shown as explained below. Observe that they are discussed in increasing order of throughput.

- Scenario one occurs when all traffic is destined “furthest around the ring” i.e. to the node’s upstream neighbour. This communication pattern does not suit the network topology under discussion (unidirectional pipelined ring). The level of throughput achieved is equivalent to that of a shared media network (e.g. a standard shared ring or bus), i.e. one data packet per slot.
- Scenario two occurs when the destination of all traffic is evenly distributed, i.e. on average the packets will travel half way around the

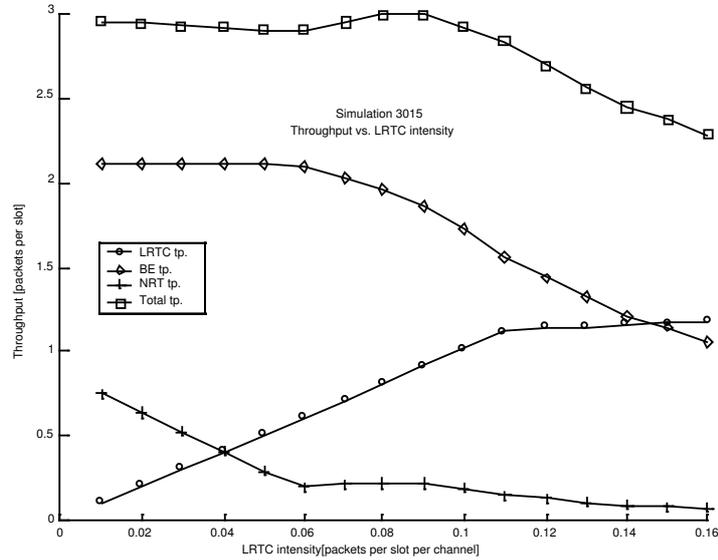


Figure 16.13: Throughput of different traffic when BE and NRT traffic stays constant as LRTC traffic varies.

ring. In this case the average throughput is two packets per slot. This is possible because of the pipelining feature of the network, where several transmissions may take place in non-overlapping segments.

- Scenario three is the radar case currently being discussed. Here, a large part of the communication is pure pipelined (the BE traffic), which is advantageous for total throughput. Therefore the total throughput will be larger than two. Observe that the total throughput depends on the traffic pattern. We have seen in the simulations that the total throughput with the radar case is 5.97 packets per slot, (see Table 16.4).
- Scenario four occurs when all traffic is destined to the next downstream neighbour. Here the pipelining feature of the network is optimally utilised. Throughput will be N packets per slot, where N is the number of nodes in the network.

16.9 Conclusions

The function of the CCR-EDF protocol has been verified by simulation. Also, the concept of having different traffic classes to differentiate between traffic has been tested in simulation, and shown to work. Results from the simulations of the radar signal processing case study show that the CCR-EDF network is an effective choice. Two applications for SANs, with support for heterogeneous real-time communication, has been described. For each application, a case study has been defined.

Acknowledgement

This work is part of M-NET, a project financed by SSF (Swedish Foundation for Strategic Research) through ARTES (A Real-Time network of graduate Education in Sweden).

Bibliography

- [1] C. Bergenheim and M. Jonsson, "Fibre-ribbon ring network with inherent support for earliest deadline first message scheduling" *Proc. International Parallel and Distributed Processing Symposium.*, , (IPDPS 2002), Fort. Lauderdale, FL, USA, 2002 15-19 Apr., pp. 92 - 98
- [2] M. Jonsson, "Two fibre-ribbon ring networks for parallel and distributed computing systems," *Optical Engineering*, vol. 37, no. 12, pp. 3196-3204, Dec. 1998.
- [3] M. Jonsson, A. hlander, M. Taveniku, and B. Svensson, "Time-deterministic WDM star network for massively parallel computing in radar systems," *Proc. Massively Parallel Processing using Optical Interconnections (MPPOI'96)*, Maui, HI, USA, Oct. 27-29, 1996, pp. 85-93.
- [4] M. Taveniku, A. hlander, M. Jonsson, and B. Svensson, "A multiple SIMD mesh architecture for multi-channel radar processing," *Proc. International Conference on Signal Processing Applications & Technology (ICSPAT'96)*, Boston, MA, USA, Oct. 7-10, 1996, pp. 1421-1427.
- [5] J. A. Stankovic, "Misconceptions about real-time computing," *Computer*, vol. 21, no. 10, pp. 10-19, Oct. 1988.
- [6] M. Jonsson, B. Svensson, M. Taveniku, and A. hlander, "Fiber-ribbon pipeline ring network for high-performance distributed computing systems," *Proc. International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'97)*, Taipei, Taiwan, Dec. 18-20, 1997, pp. 138-143.
- [7] P. C. Wong and T.-S. P. Yum, "Design and analysis of a pipeline ring protocol," *IEEE Transactions on communications*, vol. 42, no. 2/3/4, pp. 1153-1161, Feb./Mar./Apr. 1994.

[8] M. Jonsson, C. Bergenhem, and J. Olsson, "Fibre-ribbon ring network with services for parallel processing and distributed real-time systems," *Proc. ISCA 12th International Conference on Parallel and Distributed Computing Systems (PDCS-99)*, Fort Lauderdale, FL, USA, Aug. 18-20, 1999, pp. 94-101

[9] C. Bergenhem, M. Jonsson, B. Gördén, and A. hlander, "Heterogeneous real-time services in high-performance system area networks - application demands and case study definitions," *Technical Report IDE - 0254, School of Information Science, Computer and Electrical Engineering (IDE), Halmstad University*, 2002.

[10] S. Agelis, S. Jacobsson, M. Jonsson, A. Alping, and P. Lignander, "Modular interconnection system for optical PCB and backplane communication," *Proc. Workshop on Massively Parallel Processing (WMPP'2002) in conjunction with International Parallel and Distributed Processing Symposium (IPDPS'02)*, Fort Lauderdale, FL, USA, April 19, 2002.

[11] W. E. Leland, M. S. Taqqu, W. Willinger, D. V. Wilson, "On the self-similar nature of Ethernet traffic," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1-15, Feb. 1994.

Chapter 17

Wireless Real-Time Communication Using Deadline Dependent Coding

By **Elisabeth Uhlemann**[†], **Lars K. Rasmussen**[‡]
and **Per-Arne Wiberg**[†]

[†]Computer and Communications Laboratory
Halmstad University

Email: {Elisabeth.Uhlemann,Pelle.Wiberg}@ide.hh.se

[‡]Institute for Telecommunications Research
University of South Australia

Email: Lars.Rasmussen@unisa.edu.au

The constant evolution of wireless communication, and all the applications this enables, is rapidly increasing our demands on the performance of communication networks. As the transmission speed increases, entirely new applications and services, like for example video streaming, suddenly becomes interesting for wireless systems as well. The expectations of the general user with respect to performance of wireless applications are guided by the current quality of traditional wireline systems. This naturally implies a considerable challenge when designing wireless communication systems. Many of these new wireless applications are based on packet transmissions and are subject to time-critical constraints. The objective of the deadline dependent coding (DDC) communication protocol presented here is therefore to develop an efficient and fault tolerant real-time link layer foundation, enabling critical deadline dependent communication over unreliable wireless channels.

17.1 Introduction

The data rates of future wireless communications networks are expected to increase significantly in order to meet the rapidly growing demands of both telephony and advanced data communications services. The distinction between the two is vanishing as a variety of services are now being offered in common networks, leading to mixed communication traffic with different, competing quality of service (QoS) requirements. Data services such as emailing and web browsing are sensitive to transmission errors, but relatively tolerant to transmission delay. In contrast, telephony at the current level of voice quality is delay sensitive but relatively error tolerant. New services, enabled by higher data rates, may be sensitive to both transmission delay and transmission errors.

Advanced multimedia applications, such as DVD quality audio and video, have strict requirements on delay variations and transmission errors in order to maintain acceptable user perception. Another example is found in Japan where the main mobile operator (NTT-DoCoMo) envisions that an advanced “Intelligent Transportation System” is to be an integral part of a future fourth generation cellular mobile communications network [OYN00]. A vital task here is collision prevention where involved vehicles may be alerted when a potential collision situation is predicted. Every minute, one person in the world dies in a car crash. Auto accidents will also injure at least ten million people this year – two or three million of them seriously. All told, the hospital bills, the damaged property and other costs will add up to three percent of the world’s gross domestic product [Jon01], where naturally, the losses that matter the most are not even captured by these statistics. The ultimate solution is to keep cars from smashing into each other in the first place. Advanced vehicle control systems for increased safety is a potential solution to this problem.

Real-time wireless applications are also found within private communications networks, for example within industry intended for automating potentially dangerous moments in manufacturing and goods transportation. Other industrial applications may include measurement and control of moving objects, e.g., rotating parts and high mobility objects, communication to and from vehicles in a factory automation situation, temporary product lines and communication between cooperating embedded systems.

The example real-time applications discussed above all depend on real-time constraints in the sense that excess delay beyond a specified deadline degrades the quality of service. In some cases, like collision prevention in traffic safety systems, exceeding the proposed deadline not only degrades the quality of service, but renders the transmitted information useless. In addition to being subject to a real-time deadline,

the service must also be delivered correctly with high probability since information delivered before the deadline, but in error, may have severe consequences. The deadline dependent coding (DDC) communication protocol [UARW00] presented here lets the timeliness and the reliability of delivered information constitute the QoS parameters required by the application. The values of these QoS parameters are then transformed into actions to be taken by the link layer protocol in terms of coding and retransmission strategies specifically required when using an unreliable channel. Hence, the DDC protocol is explicitly intended for critical real-time applications over a wireless channel. The objective is to develop an efficient and fault-tolerant real-time communication protocol for critical deadline dependent communication over unreliable wireless channels.

17.1.1 Background

Network protocols such as IP [KR00] have traditionally been designed for a best-effort approach regarding both timeliness and information reliability. The protocol suite TCP/IP is a joint transport and network layer protocol suite that provides reliable communication through error detection and retransmission strategies, but offers only best-effort timeliness. Recently, QoS implementations extending into new IP standards have made it possible to introduce priority algorithms, e.g. IPv6 [LGW00]. Also the use of the UDP protocol in place of TCP has made it possible to provide real-time multimedia communication at best-effort information reliability. These extensions are based on modifications to protocols originally designed for different purposes, which means that most existing protocol standards can only guarantee timeliness or information reliability, but not both. Wireless ATM [Aca96] is a potential alternative for providing both timeliness and reliability. However, in many cases, ATM operates in conjunction with an overlaid TCP/IP stack, disabling the critical real-time capabilities of ATM [KR00].

Besides TCP/IP there are a number of real-time communication protocols such as [ZSR90, KSZ99, ACZD94] that strives to ensure delivery prior to deadlines, but that are best-effort protocols in terms of information reliability and consequently do not give any guarantees or explicit prediction on the probability of correct delivery when a wireless channel is considered. Recent protocols offering negotiable QoS in terms of timeliness requirements have been suggested in e.g., [AAS00, KS01]. However, reliability in terms of error in the delivered information is still not explicitly guaranteed in these protocols, since they assume that a reliable channel is directly available. In a mobile wireless environment, the encountered communication conditions are quite challenging compared to wireline communication in terms of signal degradation [Rap96]. In particular, the inherent error rate for a realistic averaged signal-to-noise

ratio (SNR) is both significantly higher and also time varying when a wireless environment is considered. In addition, most wireless applications must work under strict power constraints as they are operating through battery powered devices. The concepts of channel coding [HW99] to cope with the high error rate and powerful retransmission schemes [LC83] to provide diversity must therefore be introduced in order to obtain a reliable channel.

In OSI layered networks [KR00], the responsibility for QoS enforcement is almost entirely placed within the transport layer. In a network supporting a significant amount of critical real-time communication, this may not be efficient in terms of average or maximum delay or in terms of the overall efficient use of network resources. A more effective organization could be to imbue the QoS enforcement over multiple layers in the protocol stack. An erroneous packet may then be dealt with as early as possible, avoiding unnecessary excess delays caused by referring packet quality assessments to higher network layers at the receiving end. A promising innovation based on these principles is the design of DDC protocols. In addition, the development of resource allocation strategies based on the QoS requirements, as done in DDC protocols, introduces critical real-time constraints into additional functionalities in the lower network layers, such as channel coding and retransmission strategies. A DDC protocol provides a new and flexible approach, enabling truly negotiable and enforceable QoS that substantially increase the number of accepted service requests at high network load using a wireless channel.

17.1.2 Organization of the Paper

In this paper, we focus mainly on the link layer aspects of the DDC communications protocol, suggested in [UARW00, Uhl01], for point-to-point communications, demonstrating the flexible real-time communications features of the approach. In a wireless data packet communications system, there are many protocol aspects that introduce delays. For example, issues such as formatting the message, queuing the message while waiting for medium access, transmitting the message over the wireless medium, notifying the receiver of message arrival, and deformatting the message, just to name a few. Here, we only look at the time it takes to transmit the message over the medium and the time it takes to decode it so that it can be ready for delivery to the receiver. The principles are, however, easily extended to other cases.

The following section describes the special concerns introduced when real-time communication is to take place over a wireless channel. Next, we describe the probabilistic view adopted of the real-time constraints and the resulting QoS parameters. Thereafter the DDC protocol is described, with its retransmission strategies, the chosen channel code and

finally how the QoS parameters affect the error control strategy of the DDC protocol and consequently the level of fault tolerance chosen by the application.

17.2 Wireless Communication

In a typical wireless communication system, the channel conditions vary with time, and thus the quality of received frames that have been transmitted over the channel is not constant. The transmitted signals are perturbed by additive thermal noise, frequency and time selective fading, and interference from other transmitters. Not only does the distance between the transmitter and the receiver cause a transmission time delay but also the received energy of the transmitted signal, and hence the corresponding performance, declines with distance. Relative mobility between transmitters and receivers causes the channel conditions to change accordingly, as well as introduces relative frequency smearing in terms of Doppler effects. Fixed or moving obstacles in the propagation environment may enter into the line of sight, increasing the self-interference in terms of multiple signal reflections accumulated at the receivers and as a consequence further amplify these effects, [Rap96]. Interference can also be caused by unintentional transmitters, such as microwave ovens or by transmitters from other systems, e.g., a radio-controlled car may interfere with a commercial radio broadcast.

Due to the time-varying nature of the wireless channel, it is difficult to define a single meaningful performance measure for an arbitrary packet transmission. With large channel variations, average performance may not provide a suitable measure of quality. Assuming that channel conditions are constant for the duration of a packet transmission, the channel quality can in some cases be described by the corresponding SNR encountered by the specific packet. Consequently, the performance is often measured in terms of probability of error, P_e , or bit error rate (BER), i.e., the average error behavior for a given SNR, and then depicted over a range of such SNRs in an SNR versus BER plot.

Such a dynamic environment entails a challenge for wireless communication, especially when real-time constraints are imposed. The inherent consequence is a relatively high error rate, making the wireless channel unreliable in comparison to copper wire local loops or optical wireline channels. This has limited the extensive use of wireless access in real-time systems.

The BER of the transmitted frame can be improved by the use of a code, i.e., the inclusion of redundant bits in the frame. These redundant bits, also called parity bits, are chosen to provide a certain level of fault tolerance against errors. The relationship between the original number

of bits in the frame and the total number of transmitted bits including redundancy is called the code rate. The lower the code rate, the more redundant bits have been included; and the more redundant bits that are included the stronger the code becomes and hence the better performance can be obtained [LC83]. At the same time as additional parity bits will improve the performance in terms of BER, it will also require more resources in terms of more bandwidth, more energy and more time required to transmit and decode. Consequently, the manners in which the parity bits are chosen, greatly influence the performance and hence have to be carefully considered. By choosing a specific code, we inherently chose the manner in which the redundant bits are assigned, and hence different codes have different code rates, different performances and are also specialized on different things. Some codes are intended for packet transmission, some for continuous streams. Some are good at coping with bursts of errors, others with randomly distributed errors.

Considering a wireless radio channel, bandwidth is limited since the radio spectrum is a limited natural resource. A fully utilized frequency band cannot easily be complemented by additional resources. It is not possible to add an extra fiber or an extra bus to increase the capacity as can be done with a wireline channel. The radio spectrum is assigned according to strictly enforced rules and consequently additional bandwidth may be costly or may not exist at all. Furthermore, wireless devices are often battery powered and therefore the transmitted signal power should be limited to prolong battery life. The battery also puts restrictions on the maximal computational complexity that can be used in each of the end nodes. Moreover, given that we have a certain bandwidth to use in our system, a limited transmit power also limits the inherent interference generated by the other transmitters present in the local wireless multiple access system.

The channel capacity formulated by Shannon [Sha48] incorporates into one composite parameter the effects of channel parameters such as thermal noise, constrained bandwidth, and limited signal power. The channel capacity is a fundamental upper limit for the achievable data rate over channels described by these parameters. The significance of channel capacity is that as long as the communication rate is kept below the channel capacity, an arbitrarily low error rate can in principle be obtained if infinitely long signals are used. From coding theory, we know that most finite-length codes are good, provided they are sufficiently long. Decoding complexity, however, generally increases exponentially with code length and hence may prohibit the use of codes beyond a certain length. When a real-time communication system is used, we are not only concerned with decoding complexity but also transmission time. The question is how well we can perform when complexity requirements in terms of time to decode and time to transmit have to be considered.

Unfortunately, the channel capacity only states what performance is theoretically possible to achieve, but it does not say what codes to use in order to achieve this performance. Therefore, it has traditionally been a gap between the achievable capacity obtained by using codes of a manageable decoding complexity and the theoretical limit. However, in 1993 a novel approach to error control coding revolutionized the area of coding theory. The so-called turbo codes [BGT93] almost completely closed the gap between the theoretical limit and the achievable capacity obtained using manageable practical implementations. Turbo codes are based on concatenated codes which can be decoded iteratively using low complexity, increasing only linearly with code length. Although this decoding algorithm is still sub-optimal, it is able to essentially avoid any performance loss. Even if a capacity approaching code is used and even if the SNR is high, no communication system and no code is ever 100% error free; they are probabilistic by nature and hence liable to contain errors.

17.3 Probabilistic View

A communication failure in a real-time system can be divided into information error and temporal error. By information error is meant that the communication channel corrupts the information in such a way that the receiver is unable to interpret it or interprets it in an erroneous way. Temporal errors occur when the communication system fails to deliver the information in time. This implies that the application sets up a deadline for the communication system before which it has to deliver. The literature in the real-time field often discusses two different classes of systems, hard and soft real-time systems. In a hard real-time system, late delivery cannot be tolerated. In contrast, in a soft real-time system a certain low probability of late delivery is tolerated, but leads to performance degradation in terms of the real-time constraints. A crucial issue that has to be thoroughly addressed before a wireless system can be accepted for applications requiring real-time communication is the reliability aspects.

QoS negotiations are a powerful way of handling varying demands on the network. QoS parameters used today are, for example: bandwidth, jitter and error rate [Tan96]. All these parameters reflect demands on average behavior of the network. For industrial systems, however, the demands are often of worst case nature. Therefore, we exploit a probabilistic view of the real-time constraints that focuses on the worst case behavior in order to provide a systematic approach for the development of efficient wireless real-time communication protocols. This means that it is not meaningful to talk about hard or soft real-time requirements.

Instead we talk about a deadline for delivery and the probability of success in delivering correct information before this deadline. Thus, we use two parameters: deadline for delivery, t_{DL} , and the probability of correct delivery before the deadline, P_d . They can be viewed as QoS parameters, leading to a probabilistic view of real-time communication. It follows that a protocol layer can negotiate values of these parameters with an upper or a lower layer, thus enabling flexible admission control that provides a trade-off between the worst case delivery time and the quality of the delivered information. One of the objectives of the real-time communication protocol is to maximize the probability that the communication system will be able to accept the transmission request with the required values of the QoS parameters. The values of these parameters are requested by the application using the communication system. The value of P_d controls how reliable the transfer must be in a real-time perspective and consequently, it does not say anything about delivery of correct information after the designated deadline. Note that a hard real-time system defined using these QoS parameters will have $P_d = 1$, a situation that cannot be achieved in any physical communication system due to the noisy channel conditions.

It is worth noting that these QoS parameters are useful for real-time as well as non-real-time applications. An application sending emails may for example require a P_d as high as possible but can increase t_{DL} if negotiations require a lower QoS level in order to be able to accept the request. If a packet of image data intended for video streaming is to be sent, the deadline is relatively tight but P_d can be moderate, because an incorrect delivery (or no delivery) will appear as image noise. A packet arriving too late will disturb the viewing more. For a control application, it is important that correct information reaches its destination before a tight deadline with a fairly low error probability. This implies that more bandwidth is required for this kind of transmission to be accepted.

17.4 Deadline Dependent Coding

The main idea behind DDC is to make all components in the link control protocol *deadline dependent*. To provide a controllable measure of QoS, a probabilistic view of the real-time constraints is adopted. We thus consider the probability P_d of correct delivery prior to reaching the deadline t_{DL} . Correct delivery implies that a certain target error rate limit, P_e , is met. This can be in terms of average packet error rate, or average bit error rate within the packet. Note that the target error rate can never be equal to zero due to the presence of channel noise.

Generally, by using longer code words, i.e., by adding more redundancy, we will achieve a higher P_d – but we will, in turn, be requiring

more time, Fig. 17.1. Traditionally, a fixed time has been used when scheduling the transmission time over a medium. In contrast, we make this time variable depending of the required quality (error rate). Besides maximizing the probability of delivering the required information before a given deadline, the protocol should also attempt to minimize the required bandwidth, the transmitted energy and the average time required to successfully deliver the information. The QoS parameters t_{DL} and P_d are therefore mapped onto a retransmission protocol which plays the role of maximizing the probability of delivery with the required reliability before the deadline. It follows that DDC strives to maximize the probability of delivery over an unreliable wireless channel.

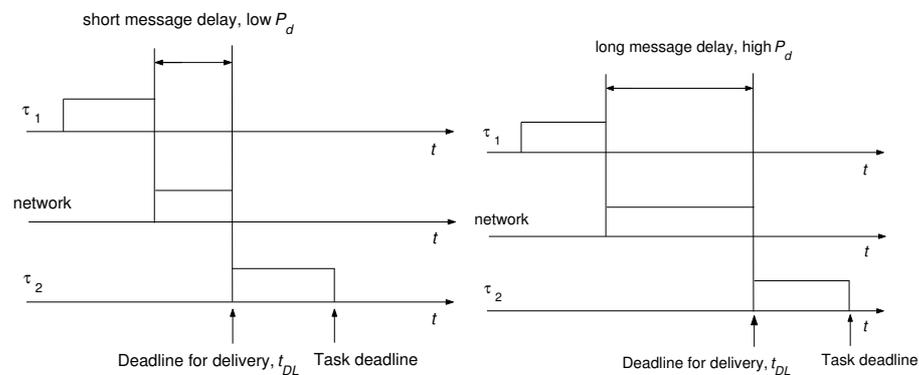


Figure 17.1: By adding more redundancy, we will achieve a higher P_d – but we will, in turn, be requiring more time.

17.4.1 Hybrid Automatic Repeat Request

In a packet-based system, an Automatic Repeat request (ARQ) scheme [LCM84] can be used. Whenever a packet arrives, the receiver may choose to reject it, and instead request a retransmission through a feedback channel. To determine whether or not a retransmission should be requested, the receiver checks the quality of the received packet, usually by means of an error detection code. A hybrid ARQ (HARQ) scheme, first suggested in [WH60], uses an error control code in conjunction with the retransmission scheme. Consequently, the receiver first tries to decode the received code word and only requests a retransmission if the quality of the decoded information is unacceptable, i.e., if the decoder output is below a certain reliability threshold, Fig. 17.2. There are different methods of determining whether a decoder output is sufficiently reliable and hence different criteria for requesting a retransmission. The choice of method significantly affects the character of the retransmission scheme.

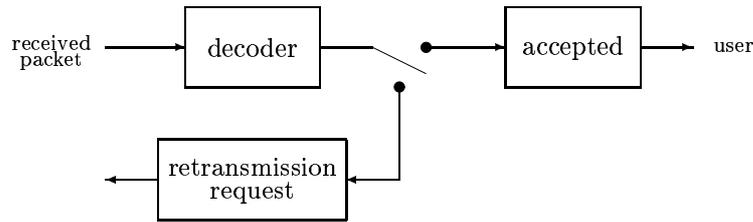


Figure 17.2: If the quality of the decoded packet is too low – a retransmission is requested.

In a non-ARQ scheme, the packet decoding process is either successful with a certain probability, P_c , or the received packet is interpreted as another valid (but erroneous) code word with a certain probability, P_e , resulting in a block error. In an HARQ scheme, we choose not to accept the decoded information if the quality is below a specific threshold and instead request a retransmission. This results in a certain *probability of retransmission* here denoted P_{ARQ} . If the threshold determining the quality of the decoded information is high, leading to a high information reliability, P_{ARQ} will be high. If P_{ARQ} is increased, consequently P_c and P_e are reduced since $P_c + P_e + P_{ARQ} = 1$. We would like P_e to be as small as possible since a block error does not result in a retransmission and hence there is no way of correcting the error. Although we may be able to minimize P_e , a large P_{ARQ} will result in numerous retransmissions, yielding a very slowly increasing P_c and a low throughput. Consequently, we want the QoS parameters, P_d and t_{DL} to guide our choice of retransmission threshold.

In a pure HARQ scheme, rejected packets are discarded. Previously received packets may, however, be used for so-called packet combining, in order to improve performance. There are two major types of packet combining, diversity combining [Sin77] and code combining [Cha85]. The choice of packet combining technique is related to the choice of error control code, but the goal should always be to use all the received observables in the decoding process.

One of the main components in a DDC protocol is the retransmission scheme. However, since we specifically consider a time-limited channel, we must limit the maximum number of retransmissions, hence the ARQ system becomes truncated [ML00]. The maximum number of retransmissions allowed is chosen according to the deadline for delivery, t_{DL} and will provide an upper bound on the communication time. It has been argued that a retransmission scheme is not to be used in a real-time system [LMM02] – however as long as there is an upper limit on the maximum number of retransmissions allowed, the delay is finite and controllable.

A retransmission scheme is continuously adapting to instantaneous channel conditions. A series of retransmissions is initiated when the channel is bad, thus contributing to the robustness of the protocol, while only a negligible number of retransmissions are required when the channel is well behaved. We may therefore use only the required amount of redundancy suitable for the current channel condition and thereby saving energy and bandwidth resources as well as reducing the amount of multiple access interference. Rather than designing the system based on the worst possible channel conditions, increasingly more channel resources are allocated as the deadline approaches, in order to meet the probabilistic requirements for delivery before the deadline.

17.4.2 Concatenated Codes with Iterative Decoding

Concatenated codes using iterative decoding is a way of providing long codes with manageable decoding complexity [For66]. A turbo code is basically a concatenated system of two simple component codes connected through an interleaver in order to create a very long code and thus also a very strong code [BGT93]. Optimal decoding of such a system is NP-hard and infeasible due to the length of the code. However, the attractive characteristic of a concatenated system is that iterative *a posteriori* probability (APP) decoding based on exchanging soft reliability information provides a low complexity sub-optimal technique for decoding. Given certain conditions, the iterative decoding technique performs close to the fundamental Shannon limit [BGT93]. In general, concatenated codes using iterative decoding provide significant performance improvements at reasonable complexity investments. In addition, only two relatively simple decoders for the component codes are needed and are reused several times, in contrast to the very complex decoder for the total code used only once.

Using concatenated codes in an HARQ scheme also elevates the corresponding performance to levels close to fundamental limits. When no retransmissions are allowed, the packet is iterated between the component decoders based on exchanging updated APP information. This is illustrated in Fig. 17.3, where the dashed box indicates the iterative decoder when no retransmissions are allowed.

The exchange of information first follows the upper path of arrows from left to right, and then back again along the lower path from right to left, constituting one iteration. In a retransmission protocol, we should use as input all the received packets pertaining to the same information bits, thereby using all received information, [NS97]. This is also illustrated with an example in Fig. 17.3 where we have connected two iterative decoders pertaining to two different received copies of the same packet. The connection represents the exchange of information between

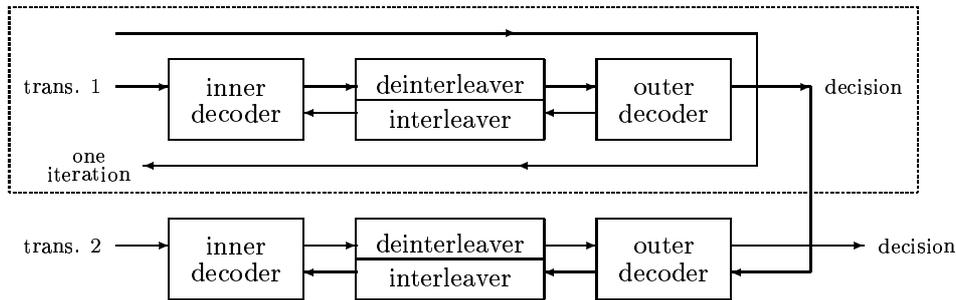


Figure 17.3: Iterative decoder for concatenated codes used in an HARQ scheme.

two received packets, leading to effective packet combining. The approach is also known as code combining [Cha85] and it can significantly speed up the convergence of the decoding process and enhance the performance in terms of BER [UARW03]. Clearly, a faster decoding process, involving fewer retransmissions and fewer iterations, translates directly into real-time benefits.

The complexity of the iterative decoder is increasing linearly with the number of iterations as opposed to exponentially with the code length. Initially, there is much to be gained from iterating, but in most cases the performance reaches a point of diminishing returns. The number of iterations needed for convergence varies between packets and is generally not known. A common approach for stopping the iterative decoding process is to allow for a fixed number of iterations. This may lead to unnecessary iterations, or to performance degradation if the process is terminated prematurely. Applying a performance based stopping criterion these problems can be addressed.

A stopping rule based on thresholding the average APP information has been found efficient [RGT01]. The stopping criterion is intended to stop the iterative process once the mean APP information is above the chosen threshold – however, if the decoding algorithm does not converge to the optimal value for a specific packet due to excessive noise, this may never happen. As we are dealing with real-time communication we must have an upper limit on the time to decode a packet. Since the decoding complexity, and hence the time required to deliver a packet with sufficient quality, is directly related to the number of iterations, we consequently need to have an upper limit on the number of iterations. Examining the convergence behavior in a concatenated system, it is generally noticed that for a majority of packets, convergence is observed after a fixed number of iterations. Consequently, we set the maximum allowed number of iterations accordingly. A non-negligible number of

packets may, however, converge after less iterations and therefore the performance based stopping criterion is still very relevant.

The criterion used for stopping the iterative decoding process is based on the convergence behavior for the iterative decoder. Naturally, the same criterion can also be used as a retransmission criterion so that if the stopping criterion is not fulfilled after the maximum allowed number of iterations a retransmission is requested. The stopping criterion together with the upper limit on the number of iterations and the number of retransmissions yield an upper bound on the decoding complexity thus also the time to decode the information.

Concatenated codes using iterative decoding also provides the opportunity to perform so-called erasure decoding. This means that if a specific bit is completely unknown, i.e., no APP information is received, the iterative decoder simply assigns an *a priori* probability of 0.5 and proceeds with the decoding process. This in turn implies that we do not need to transmit all the parity bits in the code directly. Hence, we can puncture away some bits and instead include them in a potential retransmission [RM00]. Not only does this yield an extremely efficient code combining technique, it also reduces the decoding complexity, since a retransmission can simply be included in the ongoing iterative decoding process. In other words, if we chose a very long *mother code* and simply let each transmission constitute a new part of this mother code, the decoder can just start iterating and when additional parts of the mother codes arrives the iterative process does not have to restart.

A time and safety critical application benefits from the long powerful concatenated codes yielding reliable communication, while the processing time of the iterative decoder is kept low. The iterative decoding algorithm also gives the opportunity to always deliver something to the receiver just before the deadline, i.e., we can offer a fast tentative response and progressively provide iterative refinements. This last-minute delivery can also be complemented by a measure of reliability or quality of the delivered information based on the current APP information.

A potential drawback using long codes is that a sufficiently large information block is required for the encoding process to be successful. This means that even though long codes have good performance when channel conditions are bad, they will not be good if they are made up of redundant information alone. Rather, a significant portion of the resulting code word needs to be made up of pure information. However, since concatenated codes can be made systematic, i.e., the redundant bits are appended to the unscrambled information frame, any headers and preambles required by the application can be used as part of the information frame.

In effect, iterative decoding within ARQ protocols provides an entirely new array of possibilities for adaptability in terms of complexity

versus decoding speed and reliability, i.e., critical reliability and timing constraints can be readily evaluated as a function of available system resources and complexity. This in turn enables quantifiable QoS and thus negotiable QoS. Services requests can therefore be accepted, rejected or re-negotiated depending on available resources. The resulting protocol is termed concatenated hybrid automatic repeat request deadline depending coding (CHARQ-DDC).

17.4.3 Mapping of the QoS Parameters to the CHARQ-DDC Protocol

The retransmission strategy, i.e., what and how much will be transmitted in each transmission, the length of the concatenated code, the APP threshold used as a stopping criterion, the maximum allowed number of iterations and the maximum allowed number of retransmissions are all components and features to be assigned according to the QoS parameters.

The factor that mainly influences the number of parity bits that needs to be transmitted, or equivalently the length of the mother code is P_d . We know that after having transmitted all available parity bits of the mother code, we will have a certain P_e , which is related to P_d as $P_d = 1 - P_e$. If the parity bits necessary to achieve a certain P_d are more than what we have time to transmit before t_{DL} the request cannot be accepted. Further, the level of the APP threshold used as stopping and retransmission criterion is also dictated by P_d and it will affect P_{ARQ} .

Assuming that we have assigned a mother code and an APP threshold, the maximum number of allowed transmissions dictates the partition of the mother code into transmission packets. The more redundancy that is included in an early transmission, the lower the P_{ARQ} , but at the same time the risk of submitting unnecessary redundancy increases. The more retransmissions that are allowed, the higher average code rate can be obtained and hence also implying less interference to other nodes. So, even though P_d may only require that we use half of the parity bits that would be possible to transmit before t_{DL} , we should still use the available time window to accommodate for as many retransmissions as possible, i.e. only send a small portion of parity bits with each packet to avoid unnecessary interference. So, t_{DL} tells us how many retransmissions we can accommodate for. Depending on system specific parameters, such as distance between transmitter and receiver the penalty in time introduced by an extra transmission may differ.

It should be noted that once a request has been accepted by the CHARQ-DDC protocol, the probability of delivering a packet, with the required QoS parameters P_d and t_{DL} , is guaranteed to hold. If the requested P_d can be achieved after the number of iterations and retrans-

mission given by t_{DL} , the request is accepted. If the request is rejected, the application can renegotiate by stating a lower value on one or both of the QoS parameters. Depending on the application in question one may choose to prolong the deadline or reduce the requested P_d . If, for example, we have a sensor value that is over-sampled but the correctness is critical, one may choose to prolong the deadline. A multimedia application may, alternatively, choose to reduce the requested P_d in order to meet the required deadline.

In Fig. 17.4 the P_d is plotted as a function of time. In this figure we have not taken into account the possible variations in transmission time of a packet, as these factors will affect the schemes compared in the figure equally. Further, the time it takes to make each transmission may vary due to the distance between the sender and receiver. Similarly, the time it takes to make one iteration is most likely hardware dependent and thus system specific. These statistical variations should therefore be part of the constraints imposed on the protocol design, affecting the CHARQ-DDC protocol through P_d and t_{DL} . Consequently, we have here adopted the concept of logical time, meaning that we do not put an absolute value on the time to perform one iteration, but instead report the number of iterations needed. The true transmission time merely corresponds to a scaling of the x -axis in some system specific manner.

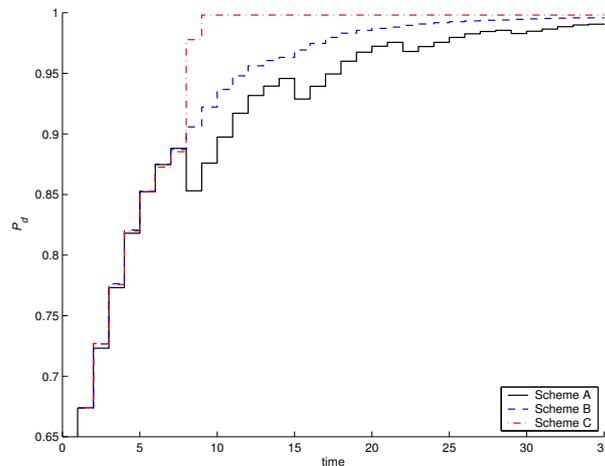


Figure 17.4: Different CHARQ-DDC systems plotted as a function of the QoS parameters.

All the schemes in Fig. 17.4 have curves with a staircase characteristic. The length of a step in the staircase depends on the time to do one iteration and in case of a retransmission, the transmission time together with the time to do one iteration. The schemes in Fig. 17.4 make a retransmission if the stopping criterion is not fulfilled after seven

iterations. The height of each step yields the performance improvement. For each iteration some additional bits can be successfully decoded and results in a new step. The schemes in Fig. 17.4 compare different packet combining techniques and as a result the performance are the same for the first seven iterations since no packet combining can be done until a retransmission has occurred. Consequently, iteration number eight is the first iteration of a newly received packet, and this is where we can see a difference between the schemes. Scheme A, which is a pure HARQ scheme discarding all erroneous copies of a message, will actually lose in performance in the beginning when a retransmission is requested. This is due to the fact that the whole packet is deemed erroneous and hence even correctly decoded bits will be discarded. Scheme B and C uses different packet combining techniques and their gain at the eighth iteration is considerable, especially for scheme C which uses the optimal packet combining technique for the particular code in question.

It should be noted that at every step there is an increased probability that the packet is correct and retransmissions will cease. Hence, using all allowed retransmissions and iterations is in fact the worst case. Most of the time we will stop earlier because the stopping criterion has been fulfilled. The BER should be considered as a mean of all information packets regardless of when the stopping criterion for each trial is fulfilled.

If simple, cheap transmitters and receivers are required, e.g., a mobile sensor with limited battery supply, the mapping of the QoS parameters onto the CHARQ-DDC protocol may be done using a look up table. If the transmitter and receiver can be more costly, the mapping can be done adaptively based on the current estimated channel conditions. The latter will of course enable additional flexibility by continuously adapting to channel variations.

17.5 Conclusions

The resulting CHARQ-DDC protocol presented here is based on CHARQ techniques, creating a flexible and dependable scheme to meet real-time constraints. Each packet has a certain t_{DL} , and P_d , required by the user or the application. These two parameters will be translated into a maximum number of retransmissions allowed and the amount of redundancy to be used.

The scheme results in improved fault-tolerance in presence of communication errors and permits a flexible admission control scheme capable of providing a trade-off between the worst-case delivery time and the quality of the delivered information. The DDC scheme differs from existing communication protocols in a number of ways. Most importantly, DDC explicitly uses the deadline to control the transmission suite and

also strives to maximize the probability of correct delivery over an unreliable channel.

Analysis of a protocol model indicates that it is possible to transmit time critical information in a mobile wireless system with very low error probabilities in an industrial environment.

The mapping of the QoS parameters to the CHARQ-DDC scheme may be done using a look-up table or, alternatively, by using adaptive strategies.

Bibliography

- [AAS00] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin. QoS negotiation in real-time systems and its application to automated flight control. *IEEE Transactions on Computers*, 49(11):1170–1183, November 2000.
- [Aca96] A. Acampora. Wireless ATM: a perspective on issues and prospects. *IEEE Personal Communications*, 3:8–17, August 1996.
- [ACZD94] A. G. Argawal, B. Chen, W. Zhao, and S. Davari. Guaranteeing synchronous message deadlines with the timed token medium access control protocol. *IEEE Transactions on Computers*, 43(3):327–339, March 1994.
- [BGT93] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: turbo codes. In *Proc. International Conference on Communications*, pages 1064–1070, Geneva, Switzerland, May 1993.
- [Cha85] D. Chase. Code combining - a maximum-likelihood decoding approach for combining an arbitrary number of noisy packets. *IEEE Transactions on Communications*, 33(5):385–393, May 1985.
- [For66] Jr. Forney, G. David. *Concatenated Codes*. M.I.T. Press, Cambridge, MA, 1966.
- [HW99] C. Heegard and Stephen B. Wicker. *Turbo Coding*. Kluwer Academic Press, 1999.
- [Jon01] Willie D. Jones. Keeping cars from crashing. *IEEE Spectrum*, 38(9):40–45, September 2001.
- [KR00] J. F. Kurose and K. W. Ross. *Computer Networking - A Top-Down Approach Featuring the Internet*. Addison-Wesley, 2000.

- [KS01] J. Kim and K. G. Shin. Performance evaluation of dependable real-time communication with elastic QoS. In *Proc. International Conference on Dependable Systems and Networks*, pages 295–303, Gothenburg, Sweden, July 2001.
- [KSZ99] S.-K. Kweon, K. G. Shin, and Q. Zheng. Statistical real-time communication over ethernet for manufacturing automation systems. In *Proc. IEEE Real-Time Technology and Applications Symposium*, pages 192–202, Phoenix, AZ, May 1999.
- [LAMM02] C. Ladas, R. M. E. Amiee, M. Mahdavi, and G.A. Manson. Class based selective-ARQ scheme for high performance TCP and UDP over wireless links. In *Proc. International Workshop on Mobile and Wireless Communications Network*, pages 311–315, September 2002.
- [LC83] Shu Lin and Jr. Costello, Daniel J. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Englewood Cliffs, N.J., 1983.
- [LCM84] Shu Lin, Jr. Costello, Daniel J., and M. J. Miller. Automatic-repeat-request error control schemes. *IEEE Communications Magazine*, 22(12):5–16, December 1984.
- [LGW00] A. Leon-Garcia and I. Widjaja. *Communication Networks*. McGraw-Hill, 2000.
- [ML00] Esa Malkamäki and H. Leib. Performance of truncated type-II hybrid ARQ schemes with noisy feedback over block fading channels. *IEEE Transactions on Communications*, 48(9):1477–1487, September 2000.
- [NS97] K. R. Narayanan and Gordon L. Stüber. A novel ARQ technique using the turbo coding principle. *IEEE Communications Letters*, 1(2):49–51, March 1997.
- [OYN00] S. Ohmori, Y. Yamao, and N. Nakajima. The future generations of mobile communications based on broadband access technologies. *IEEE Communications Magazine*, pages 134–142, December 2000.
- [Rap96] T. S. Rappaport. *Wireless Communications - Principles and Practice*. Prentice-Hall, 1996.
- [RGT01] Andrew C. Reid, T. Aaron Gulliver, and Desmond P. Taylor. Convergence and errors in turbo-decoding. *IEEE Transactions on Communications*, 49(12):2045–2051, December 2001.

- [RM00] Douglas N. Rowitch and Laurence B. Milstein. On the performance of hybrid FEC/ARQ systems using rate compatible punctured turbo (RCPT) codes. *IEEE Transactions on Communications*, 48(6):948–959, June 2000.
- [Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and pp. 623–656, October 1948.
- [Sin77] P. Sindhu. Retransmission error control with memory. *IEEE Transactions on Communications*, 25(5):423–429, May 1977.
- [Tan96] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall, Inc., New Jersey, 3 edition, 1996.
- [UARW00] Elisabeth Uhlemann, Tor M. Aulin, Lars K. Rasmussen, and Per-Arne Wiberg. Deadline dependent coding - a framework for wireless real-time communication. In *Proc. International Conference on Real-Time Computing Systems and Applications*, pages 135–142, Cheju Island, South Korea, December 2000.
- [UARW03] Elisabeth Uhlemann, Tor M. Aulin, Lars K. Rasmussen, and Per-Arne Wiberg. Packet combining and doping in concatenated hybrid ARQ schemes using iterative decoding. In *Proc. IEEE Wireless Communications and Networking Conference*, pages 849–854, New Orleans, LA, March 2003.
- [Uhl01] Elisabeth Uhlemann. *Hybrid ARQ using serially concatenated block codes for real-time communication - an iterative decoding approach*. Lic. eng. thesis, Chalmers University of Technology, Gothenburg, Sweden, October 2001.
- [WH60] J. M. Wozencraft and M. Horstein. Digitalised communication over two-way channels. In *Proc. Fourth London Symposium on Information Theory*, London, U.K., September 1960.
- [ZSR90] W. Zhao, J. A. Stankovic, and K. Ramamritham. A window protocol for transmission of time-constrained messages. *IEEE Transactions on Computers*, 39(9):1186–1203, September 1990.

Chapter 18

Real-Time Communication for Industrial Embedded Systems Using Switched Ethernet

By **Hoai Hoang** and **Magnus Jonsson**

School of Information Science

Computer and Electrical Engineering

Halmstad University

Email: {Hoai.Hoang,Magnus.Jonsson}@ide.hh.se

This paper focuses on developing and analyzing support for real-time traffic over a switched Ethernet network without any hardware or protocol modifications. Network architecture with full-duplex switched Ethernet and end-nodes has been assumed. The switch and the end nodes control the real-time traffic with Earliest Deadline First (EDF) scheduling on the frame level. No modification to the Ethernet standard is needed in the network that supports both real-time and non-real-time TCP/IP communication. The switch is responsible for admission control where feasibility analysis is made for each link between source and destination. All added traffic handling to support real-time communication is positioned in a thin layer (RT layer) added between the Ethernet layer and the TCP/IP suite. This assures adaptation to the surrounding protocol standards. The RT layer manages traffic on the basis of virtual connections, denoted as RT channels, as well as packet level scheduling. RT channels are created between end-nodes prior to any occurrence of real-time traffic. The comparison of two deadline-partitioning schemes, to partition the delay budget over the links for a path, is also presented. While SDPS (Symmetric Deadline Partitioning Scheme) is straightforward to implement, ADPS (Asymmetric Deadline Partitioning Scheme) is devised in order to have a more

flexible feasibility test. ADPS shows promises in removing bottlenecks from links, especially when a master-slave communication is considered as the traffic pattern, and ADPS proves to be a better choice than SDPS.

18.1 Network architecture

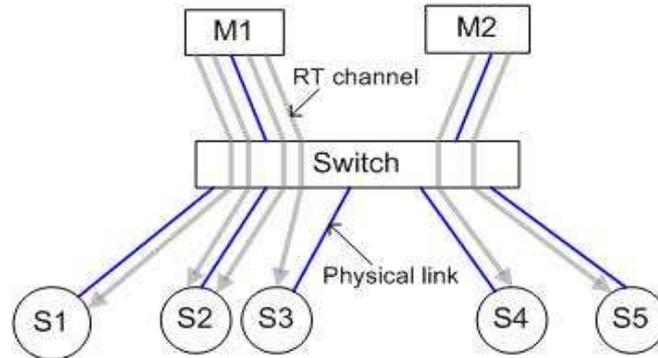


Figure 18.1: Network configuration with master-slave traffic pattern over RT channels

We assume a network with the star topology, which includes a full-duplex switched Ethernet and the end-nodes. Both the switch and the end-nodes have an RT layer added to support guarantees for real-time traffic. All the end-nodes are connected to the switch. The RT channel is described as a logical connection between two nodes in the network allowing the nodes to communicate with each other (see Section 18.2). Regular non-real-time traffic is supported at the same time. End-nodes have the capability of controlling outgoing traffic from the nodes using the Earliest Deadline First (EDF) algorithm. The switch has the same capability, and is also responsible for the admission control. The proposed network is well suited for, e.g., master-slave communication, in which the end-nodes are divided into master nodes and slave nodes according to Figure 18.1.

18.2 Real-time traffic handling

18.2.1 Real-time layer interaction

The function and interaction with the RT layer etc is shown in Figure 18.2. When an application wants to setup an RT channel, it interacts directly with the RT layer (1). The RT layer then sends a question to the RT channel management software in the switch (18.2). Outgoing

real-time traffic from the end-node uses UDP and is put in a deadline-sorted queue in the RT layer (18.3). Outgoing non-real-time traffic from the end-node typically uses TCP and is put in a FCFS-sorted (First Come First Serve) queue in the RT layer (4). In the same way, there are two different output queues for each port on the switch too (18.5) (see Figure 18.2).

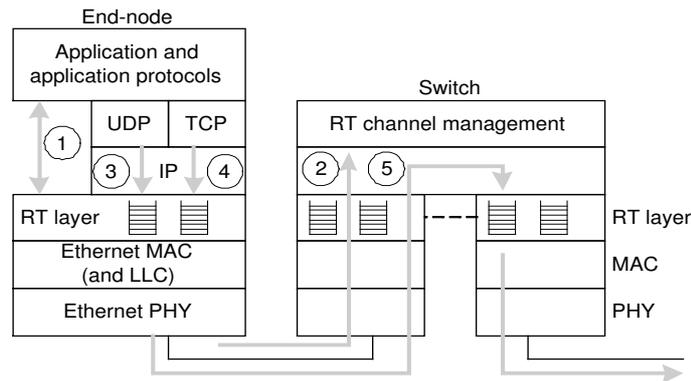


Figure 18.2: Layers and output queues.

18.2.2 Real-time channel establishment

Each RT channel is a virtual connection between two nodes in the system. The network has capability to add RT channels dynamically. The network guarantees to deliver each generated message with a bounded delay over the RT channel. An RT channel with index i is characterized by $\{P_i, C_i, d_i\}$, where P_i is the period of data, C_i is the amount of data per period, and d_i is the relative deadline used for the end-to-end EDF scheduling. All P_i , C_i , and d_i are expressed as the number of maximal sized frames.

The creation of an RT channel consists of request and acknowledgment communication where the source node, the destination node, and the switch agree on the channel establishment. When a node wants to establish an RT channel, it sends a RequestFrame to the switch, which includes (see Figure 18.3): source and destination node MAC and IP addresses and $\{P_i, C_i, d_i\}$. The connection ID field is set to a source-node unique ID for the ability to distinguish the response in the case of several requests. The RT channel ID field is not set with a valid value yet. When receiving a RequestFrame, the switch will calculate the feasibility of the traffic schedule between the requesting node and the switch, and between the switch and the destination node (admission control). If the switch finds the schedule feasible, the RequestFrame is then forwarded to

the destination node, after adding a network unique ID in the RT channel ID field. The destination node responds with a ResponseFrame (see Figure 18.4) to the switch telling whether the establishment is accepted or not. The switch will then, after taking notation of the response, forward the ResponseFrame to the source node. If the switch did not find the requested RT channel feasible to schedule, the RequestFrame is not forwarded to the destination node. Instead, a ResponseFrame is sent directly to the source node reporting the rejection.

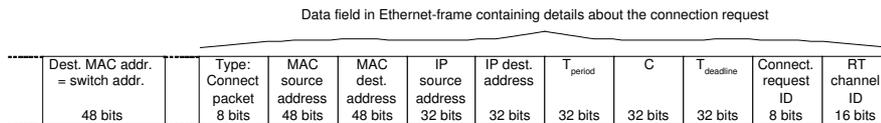


Figure 18.3: Request Frame

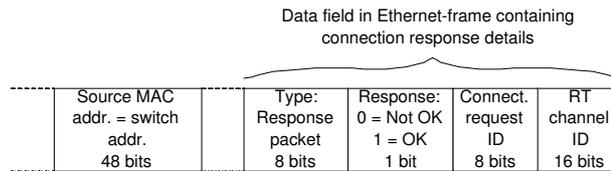


Figure 18.4: Response Frame

The RT layer in an end-node prepares outgoing real-time IP datagrams by changing the IP header (in the data part of an Ethernet frame) before allowing the Ethernet layers to forward the datagram to the switch. The IP source address and the 16 most significant bits of the IP destination address, 48 bits together, are set to the absolute deadline of the frame. The 16 least significant bits of the IP destination are set to the RT channel ID for the RT channel to which the frame belongs. The Type of Service (ToS) field is always set to value 255. Other values than 255 in the ToS field can be used for future services.

18.3 Deadline scheduling and feasibility test

18.3.1 Scheduling Algorithm

As mentioned above, both the switch and the end-nodes use the EDF algorithm to control the outgoing flow of RT traffic. An RT channel must traverse two physical links, one from the source node to the switch (*uplink*), and one from the switch to the destination node (*downlink*).

The relative deadline of a RT channel is divided into two parts: d_{iu} and d_{id} , as the guaranteed worst-case time to deliver on the *uplink* and *downlink*, respectively. The source node controls the traffic flow on the *uplink* and the switch controls the *downlink* part. The network guarantees to deliver each generated message with a bounded delay:

$$T_{max_delay,i} = d_i + T_{latency} \quad (18.1)$$

Where $T_{latency}$ is a value determined by the medium propagation delay and the medium access time. Consider a system consisting of one centralized switch and several end-nodes with identical connections to the switch. In such a system, $T_{latency}$ can be considered as a system specific constant.

18.3.2 Feasibility Analysis

The switch checks the feasibility of accepting a new RT channel, using an EDF theory modified to reflect the characteristics of the Ethernet network proposed. Each part of the RT channel can be looked upon as a periodic task, and the corresponding link would constitute a CPU or processing system (from a scheduling point of view). The capacity, C_i , would be the worst-case-execution-time (WCET) for the task. Furthermore, because the system is full duplex, each link would organize two independent CPUs, one executing the download parts of all channels traversing the link, and the other executing the upload parts. Following are some definitions that are used throughout this chapter.

- *The Utilization factor:* According to basic EDF theory [2] the utilization of periodic real-time traffic is defined as:

$$U = \sum C_i/P_i \quad (18.2)$$

- *The Hyperperiod:* The Hyperperiod for a set of periodic tasks is defined as the length of time from when all tasks' periods start at the same time, until they start at the same time again.
- *The BusyPeriod:* A BusyPeriod is any interval of time in which a link is not idle.
- *The workload function $h(n, t)$:* is the sum of all the capacities of the tasks with absolute deadline less than or equal to t , running on link n , where t is the number of timeslots elapsed from the start of the hyperperiod. It is calculated as follows:

$$h(n, t) = \sum \left(1 + \left\lfloor \frac{t - d_i}{P_i} \right\rfloor \right) C_i \quad (18.3)$$

- *A feasible link:* A feasible link is a link with a set of channels traversing it that can be feasibly scheduled using EDF.
- *The system state:* The system state, denoted SS , is defined by the pair: $\{N, K\}$, where N is the set of nodes connected to the system, and K is the set of RT channels currently active.
- *A feasible system:* A feasible system is a system state (SS) with every link in the system being feasible.

Following the above discussion, in relation to the new definitions, the problem for the switch to conduct a test to determine if the channel can be added is equivalent to testing if the new state is still feasible, given that the new channel has been added. The feasibility test of a link is done in two steps, each step being a test of its own, as shown below.

- *First Constraint:* The utilization of the link has to be less than or equal to one (100%)
- *Second Constraint:* For all values of t , the workload function $h(n, t)$ has to be less than or equal to t

Liu and Layland [2] showed that the first constraint is enough for the RT channels, which has relative deadline equal to their period. In such a scenario, it is enough for the switch to check utilization only, to determine if a RT channel can be added or not. The second constraint, in the form given above, does not lend itself out particularly well to computation. It is shown in [6] how to reduce the time and memory complexity of the second constraint check. If $h(n, t) \leq t$ in the first busy period of the *hyperperiod* in the proposed schedule to come, then $h(n, t) \leq t$ for all t . The following *upperbound* would therefore be an improvement of the algorithm noted above:

$$t, \text{ such that } 1 \leq t \leq \text{BusyPeriod}(n) \quad (18.4)$$

where $\text{BusyPeriod}(n)$ is the first *BusyPeriod* in the schedule at the start of the hyper-period, on link n . Furthermore, one does not need to check every integer from the first timeslot, but only the integers t where

$$t \in \bigcup_{i=1}^Q \{mP_i + d_i : m = 0, 1, \dots\} \quad (18.5)$$

assuming that Q denotes the number of RT channels traversing the considered link in the considered direction.

18.4 Deadline partitioning schemes

Here we review the method of looking at links as processing units, where each link has tasks to perform. This method is devised in the interest of forcing the test of system feasibility, down to the level of successive tests on links. For this approach to work, it is necessary to derive two supposed tasks from each channel. A pair of supposed tasks for the *upload* and *download* part of a channel, T_{iu} and T_{id} , is defined as:

$$T_{iu} = \{Source_i, P_i, C_i, d_{iu}\} \quad (18.6)$$

$$T_{id} = \{Destination_i, P_i, C_i, d_{id}\} \quad (18.7)$$

Where $Source_i$ and $Destination_i$ are the source and destination nodes for the channel and d_{iu} and d_{id} are deadlines for the tasks on the *uplink* part and *downlink* part respectively. Obtaining such tasks is accomplished by partitioning the deadline of the channel into two parts: d_{iu} and d_{id} where

$$D_i = d_{iu} + d_{id} \quad (18.8)$$

$$d_{iu}, d_{id} \geq C_i ; (\text{if } D_i \geq 2C_i) \quad (18.9)$$

Condition (8) must be upheld, because otherwise the channel as a whole will be different. If one divides a task into separate smaller tasks, it should follow then that if the original task had a deadline, then the sum of the subtasks' deadlines must equal this larger deadline. Condition (9) should be upheld because otherwise the partitioning will automatically yield a non-EDF-feasible situation. The deadline cannot be allowed to be shorter than the capacity, because the capacity is the WCET of the supposed tasks. We can also assure ourselves that if $D_i < 2C_i$ then the channel cannot, by definition, be EDF-feasible for a store-and-forward switch.

A deadline-partitioning scheme (DPS) is defined as: DPS is a function that maps the deadlines d_i of all the channels in the system into two deadlines d_{iu}, d_{id} such that Equation (8) is upheld for each RT channel.

The presence of a DPS gives us the freedom to create d_{iu} and d_{id} from every channel i . In fact, the availability of a DPS is not optional, but the system cannot operate without a DPS. There are different ways of looking at DPSs, but the most mathematically satisfying one is as a multi-dimensional function. The dimension of the function is then

$$dim = size(K) \quad (18.10)$$

where K is the set of channels in the system state.

We can make the DPS more agreeable as a function, by turning it into a vector field, with the range of its elements fixed between 0 and 1. To start out with, the function would not generate scalars, but it would be *dim* number of pairs of deadlines, $\{d_{iu}, d_{id}\}$. We now take steps to change this function. First, we normalize with the original deadline, d_i for each corresponding pair. Because of (8) this would mean that we would have pairs, ranging from 0 to 1. The output would look like:

$$\begin{aligned} U_{part,i} &= d_{iu}/d_i \\ D_{part,i} &= d_{id}/d_i \end{aligned} \quad (18.11)$$

where U_{part} and D_{part} are the factors of d_i to get d_{iu} and d_{id} , respectively. But because of (8) we conclude that:

$$U_{part,i} = 1 - D_{part,i} \quad (18.12)$$

This means that both $U_{part,i}$ and $D_{part,i}$ contain all the information by themselves. We can now write a DPS in the general form:

$$U_{part} = DPS(systemstate) \quad (18.13)$$

18.4.1 SDPS (Symmetric Deadline Partitioning)

In [1], it was proposed to partition the deadline of the channels into two equal parts, i.e. to split it in half. Following the notation introduced above, this would imply that

$$d_{iu} = d_{id} = d_i/2 \quad (18.14)$$

$$U_{part,i} = D_{part,i} = 1/2 \quad (18.15)$$

We define this approach as a Symmetric DPS (SDPS). It is easily seen that condition (8) is upheld under this function. We can also note that the SDPS only depends on the *size(K)* of the system state. In the view of DPSs as vector fields this means that the SDPS is represented by a vector of *size(K)* number of elements, with each element being constant, equal to 0.5. Obviously, as the SDPS doesn't take into consideration what the system looks like, we should be able to propose a better DPS.

18.4.2 ADPS (Asymmetric Deadline Partitioning Scheme)

With bottlenecks we mean links with a greater number of channels traversing them than other links. We say that bottlenecks have a higher link-load, which we propose to be defined in the following manner. We define the linkload:

- *The LinkLoad (LL)* of a link is the number of channels traversing it, which is the same as the number of tasks running on the link.

A logical approach in the case of a bottleneck is for the system to partition deadlines for the channels that traverse the bottleneck, so that a high fraction of the deadlines for the channels can be found in the tasks of the bottleneck. Obviously, the SDPS does not do anything to relieve bottlenecks, as the SDPS, as stated above, is invariant of the System State.

- *The ADPS* is a DPS devised to distribute, when possible, the deadline of channels, to where it is most needed, i.e. where the *LL* is greatest. We define *ADPS* as:

$$U_{part,i} = LL(Source_i)/(LL(Source_i) + LL(Destination_i)) \quad (18.16)$$

$$D_{part,i} = LL(Destination_i)/(LL(Source_i) + LL(Destination_i)) \quad (18.17)$$

We do an experiment with the network configuration of 10 master nodes and 50 slave nodes. To compare the result between symmetric and asymmetric deadline partitioning, in this simulation, every requested channel have the same parameters: $C_i = 3$, $P_i = 100$, $d_i = 40$. The result showed in Figure 18.5 proved that we get much better result with asymmetric deadline partitioning scheme.

18.5 Conclusions

We present a switched Ethernet based network concept supporting real-time communication with guaranteed bit rate and worst-case delay for periodic traffic. Two deadline-partitioning schemes are presented. While SDPS is straightforward to implement, ADPS is devised in order to have a more flexible feasibility test. ADPS in particular, shows promise to relieve bottlenecked links. When the master-slave communication is considered as the traffic pattern, ADPS proves to be a better choice than SDPS.

Future work into this area should include investigating the use of more complex network topologies, i.e, networks, consisting of many interconnected Switches and links having a shared medium. Alternative communication models and scheduling algorithms could be explored as well.

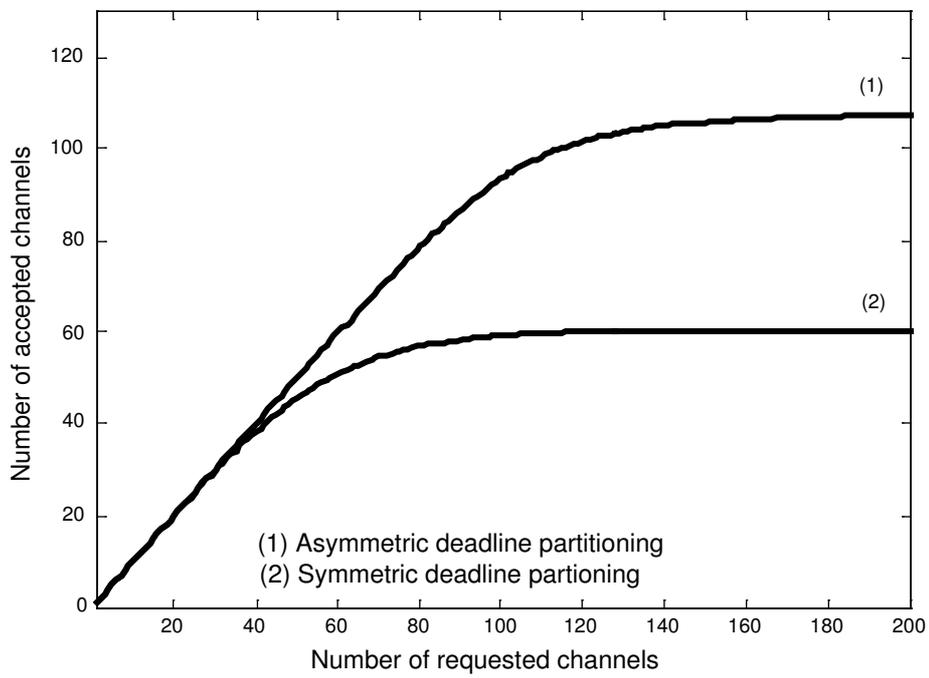


Figure 18.5: Number of accepted channels when all requested channels have the same characteristics $C_i = 3$, $P_i = 100$, $D_i = 40$.

Bibliography

[1] H. Hoang, M. Jonsson, U. Hagström, and A. Kallerdahl, "Switched real-time Ethernet with earliest deadline first scheduling - protocols and traffic handling", *Proc. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'2002) in conjunction with International Parallel and Distributed Processing Symposium (IPDPS'02)*, Fort Lauderdale, FL, USA, April 15-16, 2002.

[2] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in hard real-time traffic environment", *Journal of the Association for Computing Machinery*, vol. 20, no. 1, Jan. 1973.

[3] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal of Selected Areas in Communications*, vol. 8, no. 3, pp. 368-379, Apr. 1990.

[4] Q. Zheng and K. G. Shin, "On the ability of establishing real-time channels in point-to-point packet-switched networks," *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 1096-1105, Feb./Mar./Apr. 1994.

[5] C. M. Krishna and K. G. Shin, "Real-time systems," *McGraw-hill international edition*.1997

[6] J. A. Stankovic, M. Spuri, K. Ramamritham, G. C. Buttazzo, "Deadline Scheduling for Real-time Systems - EDF and Related Algorithms," *Kluwer Academic Publishers*, 1998.

Part IV

**Multiprocessor Real-Time
Systems**

Chapter 19

Introduction

By **Per Stenström**

Department of Computer Science and Engineering

Chalmers University of Technology

Email: `pers@ce.chalmers.se`

19.1 Background

Ever since the beginning of the era of electronic computers, there has been an immense creativity as far as inventing new computer applications which has pushed computer performance to higher levels. In fact, today's desktops are more than a million times faster than the first electronic computer and today's microprocessors consume an area that is about one million smaller than the computer room which hosted ENIAC back in 1946.

Computer performance often means the performance or response time as perceived by the application user – application performance. Sometimes we are interested in carrying out a single task within a certain amount of time because it needs to interact with a user or the environment. In one example, a computer is used to control an industrial process and the response time may not exceed a certain threshold as dictated by the physical characteristics of the system we want to control. Such response time guarantees are often referred to as *hard real-time requirements*.

In another example, a computer generates graphics images in real-time in a computer game. For a human user to perceive the image sequence as realistic, each image frame must have a sufficient level of realism and the the frame rate must exceed about 25 frames per second. While this could also translate into a hard real-time guarantee of producing an image frame each 40 milliseconds, slight fluctuation of both

the realism of each frame as well as the frame rate is usually acceptable. Such performance guarantees are often referred to as *soft real-time requirements*.

It is clear that the time it takes to carry out a computational task can exhibit more or less stringent real-time requirements. In fact, since either users or physical systems interact with most computer applications, they are subject to real-time requirements; thus, meeting response time guarantees is important and key to enable new applications.

Another measure of performance is the number of tasks carried out per time unit – task throughput. Many of today’s applications are throughput oriented. As one concrete example, imagine everytime you enter your banking card into an automatic teller machine. Your data is then matched against a huge database of accounts in your bank before the transaction is carried out. Concurrently, a large number of independent transactions are in process. Clearly, for the server machine to meet the soft real-time requirements, each transaction must be carried out in a certain amount of time which translates into a certain *throughput* of transactions per time unit. This class of throughput-oriented applications is increasing because of the wider use of the internet and mobile computing. Web search engines, online banking, and e-commerce services are some of the many examples of throughput-oriented applications.

The main question facing computer architects and technologists are how to bring computer performance – be it response time or throughput – to higher levels. Application performance, as dictated by benchmarking suites such as SPEC, has increased by 50-60% annually [1]. While clock frequency improvements as dictated by Moore’s law are responsible for a 30% annual performance growth the rest is attributable to advances in computer architecture and compilation methods.

Advances in computer architecture and compilation methods have mainly been concerned with exploiting more parallelism without changing the programming model; we still program a computer using high-level languages that have only undergone evolutionary changes, if any. As a result, exploitation of parallelism has been carried out at the instruction-set architecture level by trying to maintain as many instructions in flight in the microprocessor as possible.

Such superscalar processors typically issue a handful instructions each processor cycle. If an instruction is dependent of the execution of another, it is typically put on a hold until the dependence is resolved. Meanwhile, other independent instructions are executed. Unfortunately, dependence checking results in performance-costly bookkeeping and communication. As we shrink the technology even further, it will be increasingly difficult to push performance to higher levels by ex-

exploiting more *instruction-level parallelism* – much in part because of the increased ratio of wire-delay versus gate-delay [2].

The computer architecture community in both industry and academia has responded to this trend by departing from the so-far prevailing ‘centralized’ superscalar paradigm and is now investigating microarchitectures that consist of multiple decentralized and simpler computational structures, or cores. The overall idea is to either at compile time or runtime partition the computation into coarser chunks – called threads – and schedule these threads across the many cores. Architectural variations of this general approach include clustered microarchitectures [3], tiled architectures [4], and multi-core architectures [5]. Over time, it is reasonable to see a convergence among the many ideas being pursued. Meanwhile, we will see a renaissance for parallel processing. In this chapter, this renewed interest in parallel processing manifests itself in multiprocessors that are rapidly moving onto the chip.

The idea behind multi-core architectures is to integrate many compute engines on a single chip. As a result instead of further exploiting instruction-level parallelism, the hope is to exploit parallelism also at a coarser level which we will refer to as *thread-level parallelism*. Such microprocessor chips are referred to as *chip multiprocessors*. Because of the shift in the application domain to throughput-oriented computing, the computer manufacturers in the high-end domain are announcing products containing chip-multiprocessors with up to four cores per chip. The number of cores per chip will however naturally increase with the advancements dictated by Moore’s law.

This trend was quite expected when ARTES was initiated in 1998. Multiprocessors had been around and was the de-facto platform for servers ever since the beginning of the 90s. In addition, the success of the internet made it quite clear that throughput computing was going to be more dominant. As a result, a research cluster around multiprocessors in real-time systems was initiated that focussed on issues ranging from applications via software engineering and parallelization methodologies to architectural issues for multiprocessors. This chapter provides a background in multiprocessor technology as well as summarizing the research contributions made exemplified by three accompanying articles.

19.2 Multiprocessors: Technology Overview

Multiprocessors connect multiple processors, where each of them executes a single thread-of-control, to a logically shared memory. Thus, the model offered to the software is simply a collection of processors that can access all data in a single address space. From an implementation point of view, this model found great appeal in the 80s as it was

then quite clear that microprocessors would eventually surpass the performance of supercomputers. As a result, that decade witnessed many commercial implementations of small-scale multiprocessors [6] and triggered research in scalable solutions that had an impact in the 90s when several so called distributed shared-memory machines impacted on the commercial market [9]. Let's review the technological developments in some more detail.

Architecture Abstractions, Implementations, and Trends

The processors in a multiprocessor all access a physically shared memory through normal load and store operations. When several processors execute their programs, they will be serviced by the memory as though there was a single monolithic memory. As a result, this memory will serve each memory request on a first-come-first serve basis; however, in this global sequence of memory requests, the memory requests issued by each processor appears in the order they were specified by the program executed on that processor. This essential semantic property is called *sequential consistency* and was formulated by Lamport in 1979 [11].

Since it is obviously not practical from a performance point of view to implement the physical memory in a multiprocessor as a single monolithic memory, memory is usually partitioned into multiple memory modules and caches are attached to each processor to preserve memory bandwidth in addition to shield the processor from the longer access latencies to the shared memory [7]. However, attaching private caches to each processor offer a complication as it is now possible for multiple copies of a shared location to co-exist in the many caches. To guarantee consistency across the multiple caches, a cache coherence protocol is employed. Conceptually, such a protocol makes sure that a memory request always returns the latest value of the requested location, where latest is specified by the global sequence of memory requests as seen by the logical monolithic memory. To correctly implement such a protocol to implement the semantical interface of sequential consistency in a performance-efficient way occupied many researchers, including the author of this chapter, for some time. Effective solutions were eventually offered and are an integral part of any multiprocessor that has been shipped since the late 80s.

The first generation of multiprocessors interconnected the processors with their private caches to the memory through a bus and employed a broadcast-based so called snoopy-cache consistency protocol to implement a sequentially consistency machine model. However, because bus bandwidth couldn't keep pace with the performance growth of microprocessors and because of their intrinsic difficulty to scale up to larger configurations, research into scalable multiprocessors took off in

the late 1980s. Many projects at major universities, as well as in industry demonstrated that it is indeed possible to scale multiprocessors to large processor counts by employing scalable cache coherence solutions and interconnect technologies. Notable commercial efforts are the Sun Wildfire implementation [12] as well as SGI Origin 2000 [13]. As mentioned already, we are at a point when this technology can fit on a die. The first attempt to make this case was done in the Hydra project at Stanford [5]. Chip multiprocessor technology is now here to stay with the commercial offerings available. Nevertheless, while the programming model seems quite intuitive, it is quite challenging to exploit the performance potential of multiprocessors. Let's review state-of-the-art in the next section.

Parallelization Strategies and Trends

There are essentially four approaches to exploit the thread-level parallelism in a multiprocessor:

- User-level parallelization
- Compiler parallelization
- Run-time or dynamic parallelization
- A combination of the above

Because multiprocessors leverage on microprocessors, they can leverage on the software technologies available for single-processor systems. As a result, commonly used programming languages and software development environments can be exploited. However, to develop a parallel program that solves a problem by breaking down its sequential implementation into threads, programming languages must be extended. Such support consists of constructs to start and terminate threads and to synchronize them via the shared memory through synchronization constructs such as barriers and critical sections. Standardization efforts have led to the emergence of OpenMP which provides libraries for most popular programming languages.

Unfortunately, to take a sequential application and parallelize it is a challenging task. First, the inherent thread-level parallelism has to be identified and exposed. Then, the parallel units of execution have to be partitioned across virtual threads that eventually will execute on physical processors. Data dependences then have to be respected by orchestrating the code with synchronization constructs. Finally, a mapping strategy is needed to pin virtual threads onto the physical processors. The resulting speedup of the execution time depends on how well one can balance the available threads across processors and how

much they will have to communicate with each other as communication involves costly memory accesses that will have to bypass the caches. Moreover, since all memory is not close to the processor as is the case in a sequential processor, to exploit memory locality is even more involved in eventually achieving high application performance. In other words, how much parallelism can be exposed depends on the inherent available parallelism in the application as well as the performance characteristics of the architecture of the multiprocessors. Until now, manual parallelization efforts have only been pursued by a smaller part of the programming community that are engaged in application development for server applications.

In order to automate the parallelization effort, research into parallelizing compilers have been active for many decades [8]. Program loops have been the main target for compilers to harvest thread-level parallelism. Typically, a parallelizing compiler aims at partitioning the loop iterations across a number of virtual threads. In order to not violate correctness, it is important for the compiler to prove that there are no data dependences across such virtual threads. If the accesses to the involved data structures depend on input data, it may not be possible to precisely ascertain that there are no data dependences. As a result, compiler approaches are in general conservative and cannot expose all parallelism. As a result, they have only proven successful for some codes. Apart from data dependency analysis, automated approaches to find parallelism have to address all the issues involved in manual approaches discussed above; namely – exposing, partitioning, orchestrating, and mapping of parallel threads.

Recently, there has been a focus on run-time methods to aid the compilers to expose more parallelism by detecting data dependence violations (see [10] and the references therein). Such methods, referred to as *thread-level data speculation*, detect at run-time if two threads access the same variable of which at least one is a write access. Then the two threads may be data dependent and the execution of one of them has to be dropped. To guarantee correctness, one option is then to re-execute the threads sequentially. By offering such a 'safety-net' against data dependence violations, the compiler can more aggressively expose parallel threads by also considering threads for parallel execution that are *likely* to not cause any dependence violation. In this chapter, one of the articles considers the execution time speedup from a very simple approach to expose thread-level parallelism using such methods.

With the onset of multi-core architectures, it will be more acute to come up with programming environments for parallelization. As these architectures become widespread, there will be a greater incentive to parallelization efforts. Research into methods to simplify parallel programming will be more relevant than ever. Because there is no sin-

gle approach that has been successful, a combined approach where the user/compiler/run-time interact with each other may be a fruitful path to eventually make parallel programming widespread.

Applications and Trends

Traditionally, the type of applications that can take advantage of multiprocessors fall in one of the following categories [8]:

- Scientific applications
- Engineering applications
- Multimedia applications
- Transaction-oriented applications

Many real-life applications have a lot of inherent thread-level parallelism and have been adapted for efficient exploitation of the parallelism in multiprocessors. Computational science has traditionally been a driver of parallel machines. As a result, many problems involving solving complex physical models of various aspects of science have been adapted for parallel execution. Good examples involve meteorological models to predict weather; modelling of currents in the oceans, wind-tunnel models, etc. Interestingly, many of the computation problems that have showed up in computational science now show up in everyday applications. Virtual reality models in which objects interact with each other and sometimes collide require the computational power of problems in computational science and are feasibly hosted on small-scale multiprocessors to expose a sufficient level of realism.

Other examples of applications with inherent thread-level parallelism involve various modeling needed in engineering work. Crash simulation is a particularly computationally challenging application. In circuit design there is an immense need of modeling too. By nature the objects being modelled can be broken down into fairly independent threads which can take advantage of the parallelism available in multiprocessors.

Apart from modelling, and as mentioned in the first section of this introduction, transaction-oriented applications are growing in importance with the more widespread use of the Internet. Typically, independent transactions are expediated by the application so that a close to linear speedup with respect to the number of processors is achieved. As multiprocessors become more widespread, it is likely that its use extend to also other application domains.

19.3 Articles in this Chapter

The three accompanying articles focus on the performance interaction between applications and multiprocessor platforms. The first article by Warg and Stenström focuses on techniques to make it possible to transparently extract parallelism from programs compiled for sequential machines. They show that hard-to-parallelize code can enjoy a speedup of a factor of two by speculative execution techniques known from the literature.

The second article by Lundberg and Haggander argues that techniques that simplify maintenance of software often are at odds with techniques that improve software performance for multiprocessors. Through studies of industrial software systems on multiprocessor platforms, they propose design guidelines that can simplify software maintenance and at the same time preserve performance goals.

The last article, by Karlsson et al. focuses on how memory systems impact on the performance for typical Java-based multiprocessor server workloads. Previous studies of on-line transaction processing systems on multiprocessor servers have shown that the design of the memory system is critical to performance. The workloads considered in their study - Java-based middle-ware workloads - expose a quite different behavior. On the positive side, they have significantly smaller footprints. On the other hand, their interaction with the memory hierarchy close to the processor is quite diverse.

Bibliography

- [1] John Hennessy and David Patterson. *Computer Architecture: A Quantitative Approach*, Third Edition, Morgan Kaufmann Publ., 2003.
- [2] V. Agrawal, M. S. Hrishikesh, S. W. Keckler, and D. Burger. Clock Rate Versus IPC: The End of the Road for Conventional Architectures. In *Proceedings of the 27th International Symposium on Computer Architecture*, pages 248–259, June 2000.
- [3] S. Palacharla, N.P. Jouppi, and J.E. Smith. Complexity-Effective Superscalar Processors. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 206–218, June, 1997.
- [4] M. B. Taylor et al. Evaluation of the Raw Microprocessor: an Exposed-Wire-Delay Architecture for ILP and Streams In *Proceedings of the 31st International Symposium on Computer Architecture*, pages 2–13, June, 2004.

- [5] L. Hammond et al. The Stanford Hydra CMP. In *IEEE Micro*, pages 72–84, Dec. 2000.
- [6] P. Stenström. Reducing Contention in Shared-Memory Multiprocessors, in *IEEE Computer*, Vol 21, No 11, pages. 26–37, November 1988.
- [7] P. Stenström. A Survey of Cache Coherence Schemes for Multiprocessors, in *IEEE Computer*, Vol 23, No 6, pages 12–24, June 1990.
- [8] P Stenström, Erik Hagersten, David Lilja, Margaret Martonosi, and Madan Venugopal. Trends in Shared-Memory Multiprocessing, in *IEEE Computer*, Vol. 30, No. 12, pages 44–50, December 1997.
- [9] V. Milutinovic and P. Stenström. Opportunities and Challenges for Distributed Shared-Memory Multiprocessors. Guest Editors' Introduction, in *Proceedings of the IEEE*. Vol. 87 No 3, pages 399–404, March 1999.
- [10] P. Rundberg and P. Stenström. An All-Software Thread-Level Data Dependence Speculation System for Multiprocessors, in *Journal of Instruction-Level Parallelism*, Vol 3. Oct 2002.
- [11] L. Lamport. How To Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. In *IEEE Trans. on Computers* C-28(9): 690–691. 1979.
- [12] E. Hagersten and M. Koster. WildFire: a Scalable Path for SMPs. In *Proceedings of 5th International Symposium on High-Performance Computer Architecture*, pages 172–181, Feb. 1999.
- [13] D. Lenoski and J. Laudon. The SGI Origin: A CC-NUMA Highly Scalable Server. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 241–251, May, 1997.

Chapter 20

Limits on Speculative Module-level Parallelism in Imperative and Object-oriented Programs on CMP Platforms

By **Fredrik Warg** and **Per Stenström**
Department of Computer Engineering
Chalmers University of Technology
Email: {warg,pers}@ce.chalmers.se

This paper considers program modules, e.g. procedures, functions, and methods as the basic method to exploit speculative parallelism in existing codes. We analyze how much inherent and exploitable parallelism exist in a set of C and Java programs on a set of chip-multiprocessor architecture models, and identify what inherent program features, as well as architectural deficiencies, that limit the speedup. Our data complement previous limit studies by indicating that the programming style – object-oriented versus imperative – does not seem to have any noticeable impact on the achievable speedup. Further, we show that as few as eight processors are enough to exploit all of the inherent parallelism. However, memory-level data dependence resolution and thread management mechanisms of recent CMP proposals may impose overheads that severely limit the speedup obtained.

20.1 Introduction

While instruction-level parallelism has been a good source to boost performance of processors over a long period of time, this source is now getting exhausted. The major reasons are the difficulties in supporting huge instruction windows as well as in speculating beyond control-flow dependences. As a remedy, several recent papers have proposed support for speculative thread-level parallelism (STLP) in the context of chip-multiprocessors (CMPs) [HWO98, SM98]. A chip-multiprocessor with support for thread-level data speculation allows programs partitioned into threads to correctly execute in parallel even if it is not ascertained that the threads are indeed data independent. If it turns out that a data dependence violation occurs, the speculation support will detect it which permits the speculation system to abort and re-execute the threads in a way that respects sequential semantics.

The most popular form of STLP to exploit has been loop-level parallelism. While impressive parallelism can be obtained in numeric applications with loops that contain few loop-carried dependences, the poor parallelism coverage or lack of do-all loops in general integer applications severely limit this approach [OHL99]. On the other hand, module-level parallelism, i.e., parallelism across function, procedure, or method invocations, is potentially a more general and useful form of STLP. First, it is very simple to identify the thread boundaries; new threads are created at module invocations, and terminated when they reach a return. Second, we avoid the control dependence problem we encounter in, for instance, loop-level speculation. Presumably most importantly, however, is that modules are used frequently as the key abstraction mechanism in object-oriented programs in particular but also in imperative programming styles.

The first goal of this paper is to understand to what extent the programming style – imperative versus object-oriented – affects the *inherent* speculative module-level parallelism. We do this by considering a set of C and Java programs and carry out a speedup limit study assuming an idealized machine model. This model has an infinite number of processors, it supports perfect prediction on return as well as memory values, and it imposes no overhead on thread management or inter-thread communication. While Oplinger *et al.* [OHL99] present a limit study based on a similar idealized machine model, they didn't consider Java programs.

The most important result we gained from the experiments on the idealized model is that there is a fair amount of module-level parallelism in C and Java programs. The question is how to best exploit it in terms of appropriate architectural support. We separate out a number of concerns through a series of successively refined architectural models

as follows. The first issue we study is to what extent data dependences between threads (on return or memory values) limit the speedup obtained. More effective encapsulation of data in objects would speak in favor of an object-oriented style of programming. It is interesting to see whether this indeed will result in fewer data dependences and higher speedup limits when comparing C and Java programs.

Another motivation is to see whether simple value prediction schemes suffice or research into more sophisticated value prediction schemes are warranted. A third issue that we address is how much machine resources are needed to exploit the inherent parallelism. We address this issue by studying the speedup limit as a function of the number of processors and again whether the expected heavier use of modules in object-oriented programs would lead to more scalability. Finally, we also address to some extent how thread management overheads impact on the achievable speedup to see whether research into more effective support is warranted and what this support should target. One interesting aspect is how well the granularity of parallelism in terms of common module sizes matches the overheads incurred in recent CMP proposals. While [OHL99, OHW99, CO98] have also studied the potential of module-level parallelism in C and Java programs on CMP platforms, none of them has explicitly compared the nature of the module-level parallelism inherent in C and Java programs.

The main contribution of this paper is the insights into the inherent and architectural limits on the speedup for imperative versus object-oriented programs in a single consistent framework. Our most important findings are:

- Overall, we didn't notice any significant qualitative differences between C and Java programs suggesting that the programming style has a minor effect on the amount of parallelism to be exploited.
- The inherent module-level parallelism in applications is typically not more than four to eight suggesting that small-scale CMP or multi-threaded cores are enough to exploit all of the available parallelism.
- Most of the codes do not benefit from more advanced return value-prediction schemes than stride and last-value prediction suggesting that current predictors fare pretty well.
- The granularity of modules typically don't match the overheads in recent CMP proposals. In addition, the accuracy of memory value prediction schemes is a major inhibitor to decent speedups. This suggests that more research into better machine models and memory-value prediction schemes are warranted.

In the next section, we introduce the execution model and the series of architectural models used to identify the limits on module-level parallelism. Then in Section 20.3, the experimental setup along with the benchmark applications used are introduced. The experimental results are provided in Section 20.4. We put the work in perspective of related work in Section 20.5 along with an outlook before we conclude in Section 20.6.

20.2 Execution and architectural models

In this section, we first introduce the execution model as seen by the software and then introduce the machine models used to identify what the limits on achievable speedup assuming the execution model are.

20.2.1 Module-level execution model

The true advantage of module-level parallelism lies in its simplicity. In its simplest form, a new thread is spawned at each module (i.e., function, procedure, or method). To respect sequential semantics on a module invocation, the old thread of control executes the module, whereas a new thread is spawned that speculatively executes the code after the module call as shown in Figure 20.1. If another module call is encountered, a new speculative thread that executes the continuation of the module will once again be created. In order to respect sequential semantics, the new thread will be more speculative than the thread from which it was created (i.e. it would execute after the original thread in sequential execution), but retain the same relationship as its parent with respect to all other speculative threads. All threads must commit in sequential order; in other words, a thread cannot commit until all less speculative (earlier in sequential execution order) threads have already committed.

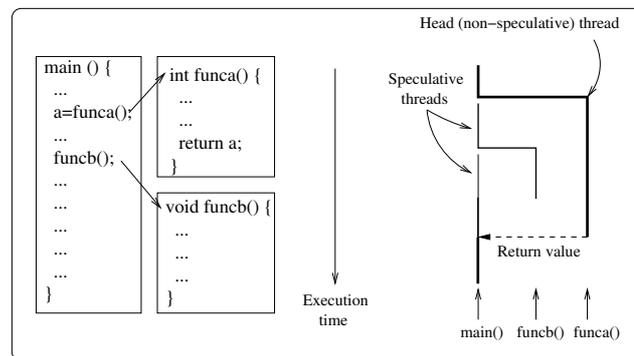


Figure 20.1: Execution model

20.2.2 Architectural models

The four models introduced gradually encounter more of the architectural limitations associated with recent CMP proposals. For each model, we first specify what limitations it encounters and then discuss what issues will be addressed with it.

Model 1: Inherent module-level parallelism

A speculative thread will successfully terminate as long as no data dependences are violated with threads that would precede it according to sequential semantics. Then the upper bound on the speedup is dictated by the control dependences between subsequent module invocations. We wish to understand the scalability of module-level parallelism in terms of how severely the control dependences set in. The first model therefore assumes a machine with an infinite number of processors and that no data dependences will be violated and cause rollbacks. In addition, thread management and inter-thread communication costs are zero.

This model provides important insights into the difference of imperative and object-oriented programming styles. One hypothesis is that an object-oriented style of programming would lead to more scalability in exploiting module-level parallelism. This is one of the hypotheses we will test using this model.

Model 2: Impact of data dependences

With this model, we are interested in how data dependences between threads limit the achievable speedup and whether proposed support in terms of forwarding and value prediction in the recent literature is enough.

There are two classes of data dependences: flow and name dependences. In this as well as in the subsequent architecture models, we assume that name (anti- and output) dependences can be resolved through renaming. In CMPs with speculation support this is done by keeping speculative state in the cache until the thread can commit, which involves flushing the speculative state back to the memory. As Steffan and Mowry have found [SM98], for the small-grain threads usually considered for CMPs, the available cache space seems sufficient to host the speculative state created.

On the other hand, flow dependences may have a severe impact on the achievable speedup through module-level speculation since a data dependence violation will result in a rollback. If a thread has computed a value before a more speculative thread reads it, the most recent value will be forwarded to the more speculative thread; but, if the value is computed after the more speculative thread performs the read, a flow

dependence violation occurs. After a violation, the more speculative (violating) thread will rollback execution in order to maintain correct sequential execution.

The model we assume is capable of perfect rollbacks, which means that the thread causing the violation will be able to restart execution exactly at the load instruction causing the violation. However, threads started by the violating thread after the erroneous instruction are squashed.

Flow dependences take two forms: flow dependences through memory and return values. To separate out the relative frequency of each category, we experiment with six alternatives:

- Heap accesses have either (1) perfect value prediction or (2) none at all. Perfect value prediction means that the prediction is always correct, and consequently we never get any memory-bound dependence violations.
- Value prediction for return values comes in three flavors: (1) Perfect return value prediction (RVP) is once again always correct; (2) stride RVP is supported by a table storing a last value and a stride value for each procedure; the table is of unbounded size. RVP buffers are updated in execution order, which is not necessarily in the same order as in the sequential execution. Additionally, it might happen that a finished thread updates the value predictor and then gets squashed, resulting in predictor pollution; one could say that the value predictor is speculatively updated. (3) The third option is no return value prediction.

With this model, we are able to answer questions related to the relative importance of memory versus return value flow dependence violations and how they relate to the programming style. One hypothesis would be that object-oriented programs tend to better encapsulate memory-bound flow dependences whereas dependences caused by return values become more critical. In addition, it is possible to pinpoint whether it would make sense to focus future research on more sophisticated value prediction schemes for STLP.

Model 3: Impact of limited processing resources

While the scale of CMPs will increase with increased integration, it may not make sense to charge too many resources to thread-level parallelism in trading off number of processors versus issue-width for example.

In the third model, we study to what extent the number of processors limit the speedup. When the number of available threads exceed

the number of processors, priority will be given to threads based on sequential execution order. New threads with higher priority will preempt more speculative threads if needed.

One problem noted in the Hydra project [HWO98] regards the speculative state stored in the caches. To avoid having to save the cache state, it is not possible to preempt a speculative thread until all the preceding speculative threads have committed. This may severely limit the speedup obtained for module-level speculation. If preemption is impossible, a new thread cannot be created when all processors are in use, unless a more speculative thread occupying one of the processors is squashed, wasting the work it has already done. An even more serious consequence would be load-imbalance problems. If a large thread is running, completed more speculative threads can neither commit, nor yield the processor, and therefore the processor will remain idle until the large thread finishes. We do not, however, impose this limitation as our aim is to establish an upper-bound on the available parallelism. While one would have to address this issue, it does not appear as a hard problem.

Model 4: Impact of thread-management overhead

In the preceding models, we have assumed that threads can be spawned, committed, and rolled back in zero time. On recently proposed CMPs such as Hydra, the overheads imposed by these operations are not negligible. To what extent the overheads have a significant impact on the speedup obtained is strongly connected to two application parameters: the number of flow dependence violations and the module granularities. Our goal with this model is to factor in these overheads to identify what mechanisms would have to be further researched to come up with machine models better adapted to module-level parallelism.

20.3 Methodology and benchmarks

In this section, we first explain how the simulations were done, and then present the benchmarks used in our experiments.

20.3.1 Simulation tools

All results presented in the following sections are obtained from our trace-driven simulation tool. The simulation tool implements all architectural models described in the previous section. The tool runs a program much as it would be run on a real machine with speculation support, that is, threads are run in parallel with run-time dependence checking. If a dependence violation is detected, the violating thread is rolled back and subsequent threads squashed. It is possible to set a

maximum number of active threads (number of processing elements) or to let every available thread run simultaneously.

As opposed to a real machine, only instructions of importance for the simulation are supported; i.e. module calls, loads and stores, returns (including the actual return value) and an instruction marking that the return value was used. These events are included in the traces. A virtual timer, which keeps track of the number of instructions executed between such events, is associated with each thread.

Traces with all information needed by the analysis tool were obtained by running the programs sequentially on a system-level instruction set simulator, Simics [MLM⁺98]. Simics makes it possible to run applications and OS in a simulated environment, and to capture memory accesses and register contents without introducing any overhead in the application. Another feature of Simics we use is to annotate the programs to make a call-out from the application to the memory system simulator to mark an event in the program in the same way as any memory system event. This feature was used to mark module calls and returns, as well as the first occurrence of return value use.

The simulated processor is a single-issue in-order SPARC v8. The memory system is assumed to be perfect, loads and stores are always available for use in the next clock cycle. This means that the simulated system always completes one instruction each cycle.

Realistic processor core and memory hierarchy models would affect the run-time of each module: ILP would decrease thread execution time on a modern superscalar core, and an imperfect memory hierarchy would increase execution time. There are many issues affected by the memory hierarchy, for instance the impact of inter-thread communication, speculative state management, context switch overhead, sharing overhead if separate caches are used, and increased bandwidth requirements because of speculation. To determine exactly how this would affect speedup, a cycle-accurate simulator of a CMP with speculation support is needed. For the time being, we have omitted these points of the design space in favor of what we consider to be more fundamental issues. Processor and memory hierarchy considerations are very important, however, and an interesting topic of future papers.

The programs were compiled with the GNU Compiler Collection (GCC) 2.95.2 with full optimizations. In a Java Virtual Machine (JVM) environment, the execution of a Java program includes class loading and verification, Just-In-Time (JIT) compilation and/or interpretation, and garbage collection. In our measurements, the Java programs are run without a JVM. Instead, they are compiled to native executables, which means neither class loading and verification, nor interpretation or JIT-compilation occurs. Furthermore, garbage collection has been disabled. Since our intention is to find the parallelism inherent in the applications,

and not to evaluate the Java run-time system, this method makes sure our measurements only contain code execution. In addition, it gives a fair comparison between Java (Object-Oriented) and C (Imperative) codes, since GCC can be used to compile all of the benchmarks. It should be noted, however, that the Java compiler is still under development, and optimizations are not as good as for the C compiler, so in comparison, the Java program instruction counts might be somewhat higher than what would be achieved with a production-quality compiler.

Only modules in the actual application are marked by the compiler. This means we do not speculate on library (or class library for Java) calls. Such functions are run inside the caller thread. Another noteworthy detail is that exceptions and I/O operations would inhibit the ability to run threads speculatively in a real machine. These events are rare in our benchmarks, so they have not been considered in the simulations. Artificial dependences through the stack from the sequential execution have been removed.

Figure 20.2 summarizes the simulation process: First the application is compiled with GCC, and annotations to make call-outs to Simics are inserted; then it is run on top of Simics, generating the trace; and finally the trace is 'executed' on the architectural models that collect the statistics we will present in the subsequent section.

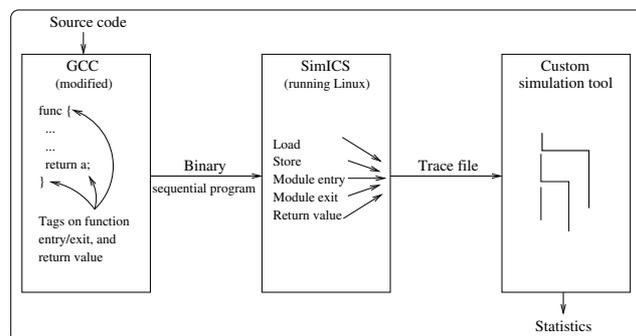


Figure 20.2: Our toolchain

20.3.2 The benchmarks

We have selected ten benchmarks, four written in C (imperative) and six written in Java (object-oriented). The C benchmarks are from the well-known SPECint95 benchmark suite and have been used in earlier STLP limit studies [OHL99, MG00]. From the eight SPECint95 benchmarks, we chose four that based on the earlier studies seem to represent typical behavior.

Table 20.1: The benchmark applications

<i>Name</i>	<i>#Instructions (dynamic)</i>	<i>#Modules (dynamic)</i>	<i>Avg. instr./mod. (dynamic)</i>	<i>#Modules (static)</i>
C Applications				
gcc	13M	54.5k	237	525
compress	1.4M	21k	67	8
go	1.4M	1.1k	1190	105
m88ksim	2.2M	0.5k	4767	34
Java Applications				
db	13M	4.9k	2644	52
deltablue	2.6M	12.5k	208	76
idea	35.7M	12k	2966	16
jess	16.3M	25.8k	633	484
neuralnet	4.2M	2.6k	1626	26

The **gcc** application is the GNU C compiler 2.5.2 compiling a single source file, and **compress** is the unix compress utility. **Go** is a simulation of the board game go, and **m88ksim** simulates a m88k processor, running the dhrystone benchmark on top of the simulated processor.

Three of the Java benchmarks are from SPEC JVM98. Unfortunately, the rest of the benchmarks in the suite did not include source code. Instead, we included two benchmarks from jBYTEmark (also used in [CO98]) and a constraint solver benchmark from Sun Labs. **db** simulates a simple database, **deltablue** is the constraint solver, and **Idea** does encryption and decryption with the IDEA block cipher algorithm. **Jess** is a simple expert system, and the final benchmark **neuralnet** is a back-propagation neural network simulation.

All these applications are general integer applications which have been found difficult to parallelize with conventional methods. Integer applications often suffer from poor parallelism coverage, lack of do-all loops, and complex data- and control flow.

In order to keep simulation times down, we had to restrict the size of the input data sets. While the data sets are small, there are still plenty of module calls to speculate on. We have tried to make sure this restriction does not affect the behavior of the programs (for instance resulting in large initialization phases); however, it cannot be ruled out that larger input sets could affect the result on some of the benchmarks.

Table 20.1 shows some statistics for each application, namely dynamic instruction and module counts, as well as average module size and static module count. However, it should be noted that the average module size can be a bit misleading; module sizes vary greatly. In Section 20.4.4 we will see that a majority of modules are less than 100 instructions in all but two of the programs.

20.4 Experimental results

In this section, we present the results of our experiments on the set of models described in Section 20.2.2 using the methodology in Section 20.3. We begin with studying the upper bound on the module-level parallelism in Section 20.4.1 followed by the impact of data dependences in Section 20.4.2, impact of limited processing resources in Section 20.4.3, and finally impact of thread-management overhead in Section 20.4.4.

20.4.1 Limits on the inherent parallelism

Figure 20.3 shows the speedup for our benchmark applications with perfect (i.e. always correct) value prediction both for return values and all memory loads. The harmonic mean for both groups (C and Java) of applications is also included. The speedup under ideal machine conditions is only limited by the module-level parallelism inherent in the program structure as constrained by the control dependences, i.e., how often and when modules are called. Figure 20.3 therefore serves as a fundamental limit for MLP, given the simplistic execution model where we begin speculation whenever a module call is encountered. The only way to find more MLP would be to speculate on module calls, i.e. speculatively call modules before execution has reached the point of the call, which may introduce the element of control-speculation. To some extent, it could also be possible to use compiler transformations to rearrange the module calls in a more advantageous way, i.e. to increase the overlap of module execution.

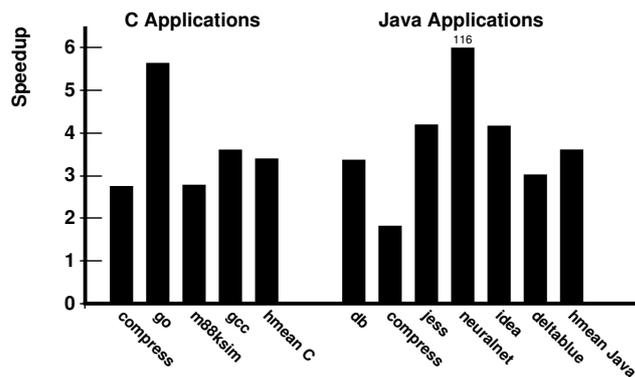


Figure 20.3: Speedup on the ideal machine with perfect memory and return value prediction

A noticeable fact is that the speedup without the impact of dependences is not spectacular, with a harmonic mean of 3.4 and 3.6 for the

C and Java applications, respectively. This means that the module calls are not arranged in such a way that there will be a large overlap of modules even if all calls are parallelized. A contributing reason as to why we do not get a large additive effect is that the majority of modules are small. However, if some of the parallelism can be extracted with reasonable effort, it might still be a useful proposition given the simplicity by which this parallelism can be extracted from existing programs.

We can also see that there is no significant difference between the Java and C application with respect to potential MLP, the mean speedup is almost exactly the same for both programming styles.

The reason for the high speedup (116) in NeuralNet is that a number of modules are called repeatedly inside a main loop, encompassing the entire program except for a short initialization phase. Thus, the main loop uncovers large amounts of MLP.

20.4.2 Impact of data dependences

The previous model predicted speedup under the assumption that value prediction on return as well as memory values is perfect. Perfect value prediction is of course not possible to attain, so the first question on our trek towards a realistic machine model is: how would value predictors with reasonable implementation complexity affect speedup?

In Figure 20.4, we show how memory load value prediction (MVP) affects performance. For each application the left bar, labeled (P), is the speedup with perfect MVP, while the right bar, labeled (N), indicates speedup with no MVP. The difference in height thus indicates the potential of memory load value prediction. The two rightmost bars once again show the harmonic mean.

The lack of memory value prediction has a substantial impact on some of the benchmarks. For instance, most of the massive potential in the NeuralNet benchmark disappears. NeuralNet contains numerous shared data structures that are continuously updated in each iteration of a main loop; therefore, this main loop is not possible to parallelize. The remaining parallelism comes from partial overlap of modules within a loop iteration. The key methods in NeuralNet do contain a good amount of loop-level parallelism, which cannot be exploited with the MLP-only approach. In [CO98], the authors have extracted module-level parallelism from this application by recoding it, converting loop-level parallelism to MLP.

The shaded vertical sections on each bar in Figure 20.4 show the impact of return value prediction (RVP). Three policies are presented: no RVP, stride RVP, and perfect RVP. For the no RVP policy, modules are still run speculatively, but a rollback always occurs to the point where the return value is used. The gap between no RVP and perfect

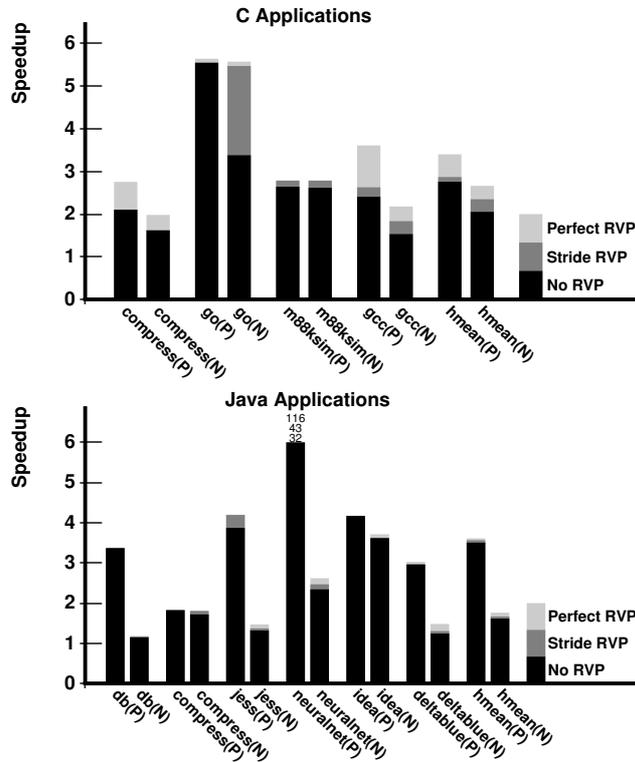


Figure 20.4: Value prediction: the left bar (P) for each application has perfect memory value prediction, the right bar (N) has no memory value prediction

RVP will reveal the potential benefits of return value prediction. We also included a known and computationally simple value predictor as an indication of the predictability of return values; the stride predictor, which predicts the next value as the last value plus the difference between the two last values. Another obvious candidate would be a last-value predictor, however, they both catch the most obvious case of a function that almost always returns the same value.

Return value prediction seems to make sense in some of the benchmarks, but surprisingly, in many of the programs most of the parallelism can be exploited without RVP, since a large portion of the modules either do not produce a return value at all (void modules), or produces a return value which is never used. If one would choose a scheme without RVP, a speculation system could catch and rollback modules in the cases where the return value is indeed used, but a better way would be to have the compiler mark all calls to void modules and those whose return value is not used, since this can be determined statically.

The simple stride value predictor has been observed to perform reasonably well in most applications, successfully predicting between 20% and 80% of return values in seven of the ten applications. Some applications, notably Idea and NeuralNet, did show a very large percentage of mispredictions. It turned out to be because of heavy use of a random number function during initialization (which is a small part of the total execution time). It would be useful to be able to selectively disable speculation for such cases, where obviously no value predictor can be expected to perform well.

The execution time in Idea is concentrated to one module which handles encryption/decryption. Most of the speedup we can see for this program is because of overlap of two iterations of encryption and decryption (four calls to this module). Although it is written in Java, it has the structure of an imperative program, which is not surprising considering the fact that it is converted from C. NeuralNet and Java Compress are also originally C programs.

Our initial belief was that the object-oriented (Java) programs would exhibit more parallelism than the imperative (C) at this point for two reasons: the object-oriented programming style encourages more frequent use of module calls, and the use of data encapsulation would result in fewer memory dependences. However, our results do not indicate any such difference (Figure 20.4). There seems to be a small difference when it comes to return values, the speedup of the C programs are slightly more affected by rollbacks due to return value mispredictions.

In summary, two important lessons can be learned from this experiment: a simple return value predictor will suffice in most cases, and a good memory load predictor would be very useful. In the rest of this paper, we will assume no value prediction on memory loads, and stride value prediction for return values, since we feel that this represents a design choice of reasonable complexity today.

20.4.3 Impact of limited processing resources

In Figure 20.5 we can see the speedup for our applications running on a machine with limited processing resources. The model is still ideal in the sense that we assume the processing elements (PEs) on an n -way machine can always be utilized executing the n least speculative threads. A more speculative thread will be preempted, without penalty, if a new less speculative thread arrives; execution will be resumed, however, where it was preempted the next time the thread can be rescheduled on a PE.

With this model, we can see that virtually all potential speedup can be utilized with only eight PEs. In fact, many of the benchmarks do not benefit significantly from more than four PEs. This is good news, since it shows that parallelism is in general not concentrated to a limited part

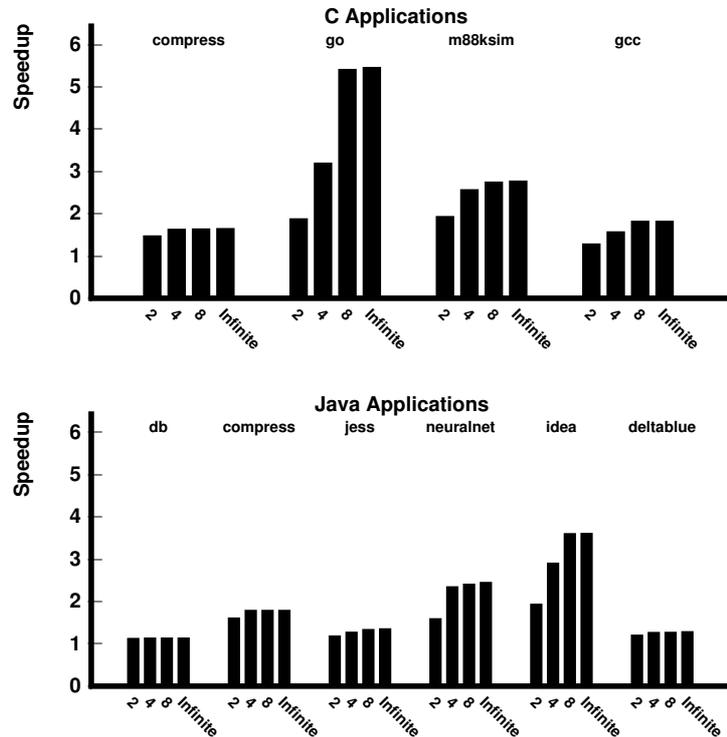


Figure 20.5: Speedup with 2, 4, 8 or an infinite number of PEs

of the execution; rather, a limited number of PEs are busy working most of the time.

This data suggests that all the module-level parallelism available in C and Java programs could potentially be exploited using chip multi-processors with relatively few processor cores. Again, there is not any big difference across C and Java programs.

20.4.4 Impact of thread-management overhead

Figure 20.6 shows speedup with overhead for speculation support. We have included three types of overhead: starting a new speculative thread, performing a rollback on misspeculation, and committing speculative state when a thread has successfully finished. A thread that has been squashed as part of a rollback will incur a new thread start overhead when it is called again.

In the figure, the three types of overhead are set to the same size; we ran simulations for 10, 100, or 1000 cycles. For the sake of comparison, we repeat the speedup for the no-overhead machine. The numbers are for an 8-way machine.

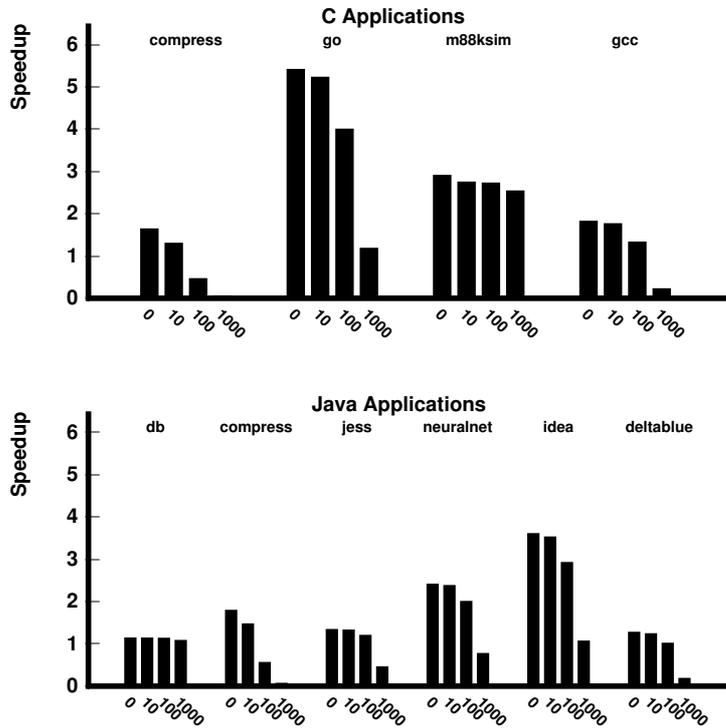


Figure 20.6: Speedup with thread-management overheads of 0, 10, 100 or 1000 cycles on an 8-way machine

The 100-cycle overhead simulations are interesting since they approximately correspond to the overheads reported for module speculation support in the Hydra CMP [HWO98]. At a 100-cycle overhead, we can already see a severe impact on the speedup for several applications, even a slowdown for both compress programs. When the overhead is increased to 1000 cycles, the compress programs are more than ten times slower than their sequential execution.

In order to find the reason for this, we do not need to look further than module size. Figure 20.7 reveals that for both C and Java compress, the majority of the modules are shorter than 20 cycles, and almost all are under 100 cycles. This means that thread-management overheads will dominate execution time, since each module will, at least, give rise to a thread start overhead when called, and a commit when it reaches return. On the other hand, some of the modules are very large, which explains why the average sizes presented in Table 20.1 are several thousand instructions for some of the programs. Note that with our single-issue, perfect memory machine, there is a one-to-one correspondence between the number of cycles and instructions.

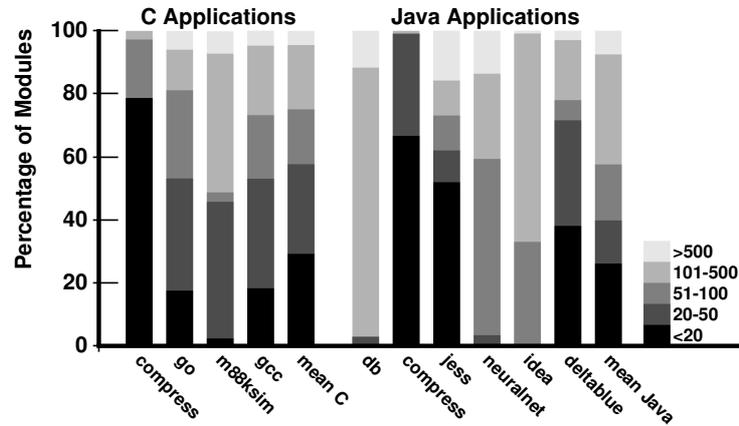


Figure 20.7: Percentage of modules (dynamic) of size <20, 20-50, 51-100, 101-500, or >500 cycles

We have observed that one of the side effects of increasing the number of processing elements is that the number of dependence violations will also increase. Therefore, with high thread overheads, the benefits of adding more processing elements will be smaller than indicated in Figure 20.5, in some cases we can even get a slowdown.

In Figure 20.8 we can see how the execution time is used. The execution time for a program in this figure is the total used time on all PEs added together. The simulations are run with 100-cycle thread-management overheads on eight PEs.

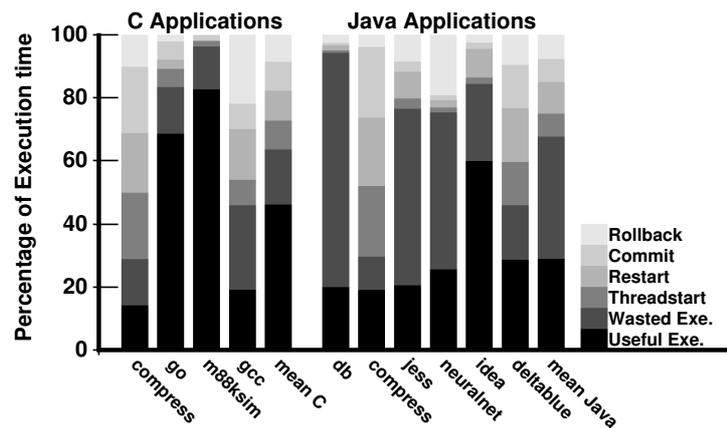


Figure 20.8: Part of execution time that is useful, wasted because of rollbacks, and used for thread-management. 100-cycle overheads on an 8-way machine

Useful execution is the part of the execution that was successful and committed; it is the part that corresponds to the sequential execution. Wasted time is the execution time that was thrown away because of a rollback or when the thread was squashed. Restart is the effect of additional thread start overhead for a thread that was squashed and must be started again. The remaining three categories show thread start, rollback, and commit overhead.

This figure points out one of the serious disadvantages of speculative execution. Only slightly more than 20% on average for Java programs, or 40% for C programs, of the processing time is useful execution. This is a disadvantage in a multitasking environment where other processes might make better use of the resources. It is also a problem from an energy-efficiency perspective. Wasted execution makes up a major part of the total processing time for most benchmarks. A conclusion would be that methods for minimizing the number of misspeculations, and thus wasted execution, is probably needed even if it is not necessary from the point of view of performance for a single-application.

It is clear from this experiment, however, that keeping overheads small is of utmost importance for module speculation support.

20.5 Related work

There is a large body of research in the recent literature that focuses on architectures and compilation techniques for speculative thread-level parallelism.

One of the first architecture proposals for thread-level speculation was done within the Multiscalar project [SBV95]. One of the novel features of this architecture is the address-resolution buffer [FS96] that validates and signals violations to data dependences between threads. Another noticeable speculative architecture proposal is the superthreaded architecture [TY96]. Several distributed approaches to implement support for thread-level speculation have also been presented in the framework of chip-multiprocessors [GVSS98, HWO98, SM98]. This study is based on the feasible inclusion of such a mechanism in chip multiprocessors.

Another important prerequisite for this study is the progress in value prediction done over the last few years. Value prediction enables speculation beyond the data flow limit. It was introduced by Lipasti *et al.* [LWS96] as a way to hide memory load latency by allowing data dependent instructions to execute in parallel. The predictability of data values was investigated in [SS97]. Others have followed up with a number of inventive prediction schemes such as the stride and last value predictors [LWS96] which we study in this paper.

This paper focuses on the opportunities and limitations of speculative module-level parallelism – a straight-forward method to extract thread-level parallelism out of existing software. Several recent papers have had similar goals. A limit study of the inherent loop-level as well as module-level parallelism in SPECint95 programs was recently published by Oplinger *et al.* [OHL99]. While disregarding architectural limitations in terms of thread management overheads, they found that there is ample module-level parallelism in the benchmark suite that can be exploited by multiprocessor or multithreaded processor cores of typically less than eight processors. In comparison with our study, they didn't address how important memory-level dependences are and did not look at Java applications. Moreover, they didn't study how much typical overheads in CMP architecture models would affect the achievable speedup.

In contrast, Chen and Olukotun [CO98] focus on Java programs. Their study is mostly aimed at the speedup obtained on the Hydra CMP proposal and does neither address the impact of various value prediction schemes nor how scalable the performance is. While they note that thread management overhead may have a severe impact on the speedup, they didn't analyze how it relates to the size of the modules. In a follow-up study by the same team based on SPECint95 programs [OHW99], they observe that thread-management overheads can be detrimental to the speedup obtained because of the penalties associated with misspeculations. As a remedy, they propose and evaluate schemes that select modules to speculate on depending on their likelihood to succeed.

Value prediction as a way to reduce dependence violations in thread-level data dependence speculation architectures has been investigated by Marcuello *et al.* [MTG99, MG00] in the context of their Clustered Speculative Multithreaded processor. They speculate on live input values to threads (values used but not defined within the thread) at thread start time. Some works mentioned earlier has also used value prediction for module return values [CO98, HWO98] and memory loads [OHL99]. Others who have used value prediction in conjunction with coarse-grained speculative architectures include [RJSS97, AD98, CW99]. However, to the best of our knowledge, our study is the first to address the limits on value prediction which pin-points whether there is room for improvements.

20.6 Conclusions

The goal of this study has been to understand the impact of the programming style – imperative versus object-oriented – on the inherent module-level speculative parallelism as well as how architectural defi-

ciencies in proposed chip-multiprocessor architectures affect the achievable speedup.

One would expect that object-oriented programs would make more heavy use of modules and would encapsulate many of the data dependences with a potential to expose more module-level parallelism. Contrary to this intuition, we found that there is not any significant difference between the inherent module-level parallelism in the C versus the Java programs that we studied. In both cases, we observed a speedup limit of about 3.5. In addition, the two suites representing the two programming styles were both sensitive to memory-level data dependences which suggests that progress in memory value prediction schemes are important to approach the maximum speedup. As for return-value prediction schemes, simple ones based on last- or stride-value fare pretty well across all applications.

When considering the impact of architecture-level constraints, we found that all of the inherent parallelism could be exploited by typically small multithreaded or multiprocessor cores with less than eight processors. However, a key inhibitor to reaching the speedup limit is the overheads imposed by thread management including the time to start (or restart), commit, or roll-back threads upon data dependence violations. Given the fairly small module sizes, speedup is severely affected when the overhead exceeds a hundred cycles. This calls for more efficient thread management mechanisms than what is currently known in the literature for chip-multiprocessor architectures.

Obviously, using MLP in more loosely coupled architectures is not an option.

In this study, we did not try to enforce a certain thread granularity, instead all modules were parallelized regardless of size. On realistic architectures, with various overheads, granularity is important. Methods to selectively apply module-level speculation would be needed. Our reason for using modules as the only source of parallelism was that they are control independent and easy to identify. Since the amount of MLP is limited, additional sources of parallelism are needed in order to achieve large performance gains using thread-level speculation techniques, but likely at the expense of increased complexity.

Acknowledgments

We would like to thank Peter Rundberg and Magnus Ekman of Chalmers University of Technology, for their valuable comments on earlier drafts of this paper. This research was supported by the ARTES/PAMP program of the Swedish Foundation for Strategic Research (SSF) and by equipment grants from Sun Microsystems Inc. and the Swedish Coun-

cil for Planning and Coordination of Research (FRN) under contract number 96238.

Bibliography

- [AD98] H. Akkary and M. A. Driscoll. A dynamic multithreading processor. In *Proceedings of the 31st Annual International Symposium on Microarchitecture (MICRO '98)*, pages 226–236. IEEE Computer Society, December 1998.
- [CO98] M. K. Chen and K. Olukotun. Exploiting method-level parallelism in single-threaded Java programs. In *Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques (PACT '98)*, pages 176–184. IEEE Computer Society, October 1998.
- [CW99] L. Codrescu and D. S. Wills. Architecture of the atlas chip-multiprocessor: Dynamically parallelizing irregular applications. In *Proceedings of the 1999 International Conference on Computer Design (ICCD '99)*, pages 428–435. IEEE Computer Society, October 1999.
- [FS96] M. Franklin and G. Sohi. Arb: A hardware mechanism for dynamic memory disambiguation. In *IEEE Transactions on Computers Vol. 45 No. 5*, pages 552–571. IEEE Computer Society, May 1996.
- [GVSS98] S. Gopal, T. Vijaykumar, J. Smith, and G. Sohi. Speculative versioning cache. In *Proceedings of the Fourth International Symposium on High-Performance Computer Architecture (HPCA '98)*, pages 195–206. IEEE Computer Society, February 1998.
- [HWO98] L. Hammond, M. Willey, and K. Olukotun. Data speculation support for a chip multiprocessor. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII '98)*, pages 58–69. ACM Press, October 1998.
- [LWS96] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen. Value locality and load value prediction. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII '96)*, pages 138–147. ACM Press, October 1996.

- [MG00] P. Marcuello and A. Gonzalez. A quantitative assessment of thread-level speculation techniques. In *Proceedings of the 14th International Conference on Parallel and Distributed Processing Symposium (IPDPS '00)*, pages 595–604. IEEE Computer Society, May 2000.
- [MLM⁺98] P. S. Magnusson, F. Larsson, A. Moestedt, B. Werner, F. Dahlgren, M. Karlsson, F. Lundholm, J. Nilsson, P. Stenström, and H. Grahn. SimICS/sun4m: A virtual workstation. In *Proceedings of the USENIX 1998 Annual Technical Conference*, pages 119–130. USENIX Association, June 1998.
- [MTG99] P. Marcuello, J. Tubella, and A. Gonzalez. Value prediction for speculative multithreaded architectures. In *Proceedings. 32nd Annual International Symposium on Microarchitecture (MICRO '99)*, pages 230–237. IEEE Computer Society, December 1999.
- [OHL99] J. T. Oplinger, D. L. Heine, and M. S. Lam. In search of speculative thread-level parallelism. In *Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques (PACT '99)*, pages 303–313. IEEE Computer Society, October 1999.
- [OHW99] K. Olukotun, L. Hammond, and M. Willey. Improving the performance of speculatively parallel applications on the hydra CMP. In *Proceedings of the 1999 International Conference on Supercomputing (ICS '99)*, pages 21–30. ACM Press, June 1999.
- [RJSS97] E. Rotenberg, Q. Jacobson, Y. Sazeides, and J. Smith. Trace processors. In *Proceedings of the 30th Annual International Symposium on Microarchitecture (MICRO '97)*, pages 138–148. IEEE Computer Society, December 1997.
- [SBV95] G. S. Sohi, S. E. Breach, and T. N. Vijaykumar. Multiscalar processors. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA '95)*, pages 414–425. ACM Press, June 1995.
- [SM98] J. G. Steffan and T. C. Mowry. The potential for using thread-level data speculation to facilitate automatic parallelization. In *Proceedings of the Fourth International Symposium on High-Performance Computer Architecture (HPCA '98)*, pages 2–13. IEEE Computer Society, February 1998.

- [SS97] Y. Sazeides and J. E. Smith. The predictability of data values. In *Proceedings of the 30th Annual International Symposium on Microarchitecture (MICRO '97)*, pages 248–258. IEEE Computer Society, December 1997.
- [TY96] J.-Y. Tsai and P.-C. Yew. The superthreaded architecture: Thread pipelining with run-time data dependence checking and control speculation. In *Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques (PACT '96)*, pages 35–46. IEEE Computer Society, October 1996.

Chapter 21

Trade-offs and conflicts between performance and maintainability in large multiprocessor based Real-Time Systems

By **Lars Lundberg** and **Daniel Häggander**

Department of Systems and Software

Blekinge Institute of Technology

Email: {llu,dha}@bth.se

21.1 Introduction

High performance in terms of throughput and response times is always desirable, and in some real-time applications it is imperative that the system responds within certain time limits. The use of shared memory multiprocessors, is one important way to improve performance in real-time systems. Performance is, however, not the only important quality when developing software for large real-time systems; reducing the cost and time to market is equally important. It is well known that roughly 80 percent of a typical software systems cost occurs after initial deployment, in the maintenance phase (see page 32 in [1]). It is, therefore, very important to adopt design techniques that will minimize the maintenance cost, i.e., we want to obtain high maintainability.

A number of design techniques are used in order to obtain high maintainability, e.g., object-orientation, frameworks and design patterns [6].

One common characteristic of design techniques that support maintainability is that they tend to split objects into small pieces, one reason for this is that there can be a large number of inheritance levels and object details and specifics are gradually introduced in the inheritance tree. The advantage of splitting objects into smaller pieces is that, the impact of design changes can be limited to only a small part of the object oriented design. However, another effect of splitting object into smaller pieces is that creating a new object at run time can result in a large number of memory allocations on the heap (one allocation for each piece). Studies of large industrial real-time applications show that programs developed for high maintainability tend to generate excessive use of dynamic memory, which can degrade the performance seriously, particularly on multiprocessors [8, 10].

The example above shows that the ambition to build maintainable software systems can result in poor, or sometimes very poor, performance. The opposite is, however, also true, i.e., the ambition to build systems with high performance (and availability) can result in poor, or very poor, maintainability. One reason for this is that in order to optimize performance the application code may become polluted with smart programming tricks. Another reason could be that the application programmer thinks that third party software, such as database management systems, will degrade performance and that the corresponding functionality should, for performance reasons, be included in the application code.

This means that there are situations where one has to make trade-offs and balance software performance (and availability) against maintainability and flexibility. There are, however, also situations where the conflicts can be avoided reduced or significantly.

We have during a period of six years looked at four large and performance demanding industrial applications. Based on our experiences we identify situations where conflicts occur. We also identify a number of techniques for handling these conflicts.

The rest of this chapter is organized in the following way. In Section 2 we give a brief description of the four applications that we have studied. Section 3 presents the different studies that we have done on the five applications and our experiences from these studies. In Section 4 we present we present different ways of handling these conflicts, i.e., our main results. Section 5 discusses our results and some related work. Our conclusions can be found in Section 6.

21.2 Industrial Cases

We have looked at four large industrial applications over a period of six years. All applications are large and performance demanding, i.e., performance and maintainability are important qualities in all applications. The applications are all from the telecommunication domain.

21.2.1 Billing Gateway (BGw)

The Ericsson Billing Gateway (BGw) connects the operators network with post processing systems. A post processing system can be any system interested in call data, e.g. billing systems or statistical systems.

Each phone call and other network service, such as sending an SMS, generates a number of Call Data Records (CDRs). The CDRs are sent from the switches and other network elements to the BGw. After being processed, the data is sent to post processing system, e.g., a billing system that generates a customer bill. The BGw may change the format of the CDRs. It may also filter out some unbillable CDRs and separate CDRs for roaming calls from CDRs for non-roaming calls. In some cases, a number of CDRs are assembled into one CDR by the BGw.

The BGw is written in C++ using object-oriented design techniques. The application consists of more than 100,000 lines of code. BGw is multithreaded and runs on Sun/Solaris multiprocessor servers with 1-32 processors depending on the performance requirements of the particular network operator. Scalability on multiprocessors is thus very important.

21.2.2 Fraud Control Center (FCC)

When operators first introduce cellular telephony in an area their primary concern is capacity, coverage and signing up customers. However, as their network matures lost revenues due to fraud becomes more important. One type of fraud is cloning fraud, where the caller uses a false or stolen subscriber identity in order to make free calls or be anonymous. The Ericsson FCC is part of an anti-fraud system that combats cloning fraud with real-time analysis of network traffic.

Software in the switching centers provides real-time surveillance of suspicious activities associated with a call. The idea is to identify potential fraud calls and have them terminated. One single fraud indication is, however, not enough for terminating a call. The FCC makes it possible to define certain criteria that have to be fulfilled before a call is terminated, e.g., a certain minimum number of fraud indications with a certain time interval.

The FCC is written in C++ using object-oriented design techniques and Sun Solaris threads. The target platform is Sun multiprocessors.

21.2.3 Data Monitoring (DMO)

In order to detect network problems immediately, the network operators collect performance data continuously in real-time. Examples of performance data are the number of uncontrolled call terminations and the number of failed hand-over for each cell in the network. The DMO application collects performance data in real-time. If certain parameters are outside of a user defined interval an alarm is generated. It is also possible to define an interval relative historic average values.

Two students groups were assigned the task of developing two DMO applications. The requirements were given by Ericsson. Each group consisted of four students working full time for 10 weeks. The groups had identical functional requirements. One group (DMO 1) was allowed to use a third party database system (Sybase), whereas the other group (DMO 2) had to use the standard Solaris file system. Both systems were going to run on Sun multiprocessors.

DMO 1 implemented the system as one multithreaded program, and DMO 2 implemented the system as a number of Solaris processes that communicated with the database server.

21.2.4 Prepaid Service Data Point (SDP)

The SDP (Service Data Point) is a system for handling prepaid calls in telecommunication networks. The basic idea is simple; each phone number is connected to an account (many phone numbers can in some rare cases be connected to the same account), and before a call is allowed the telecommunication switch asks the SDP if there are any money left on the account. This initial message in the dialogue is called First Interrogation. The call is not allowed if there are no, or too little, money on the account.

If there are money on the account the call is allowed and the estimated amount of money for a certain call duration (e.g. three minutes) is reserved in a special table in the database (the account should not be reduced in advance). When the time period (e.g. three minutes) has expired, the telecommunication switch sends a request for an additional three minutes to the SDP. This type of message is called Intermediate Interrogation. The SDP then subtracts the previous reservation from the account and makes a new reservation for the next three minutes. This procedure is repeated every third minute until the call is completed. The telecommunication switch sends a message (Final Report) to the SDP when the call is completed, and at this point the SDP calculates the actual cost for the last period (which may be shorter than three minutes) and removes this amount from the account. Calls that are shorter than three minutes will only result in two messages, i.e., a First Interrogation message and a Final Report message.

The SDP consists of about 170,000 lines of C++ code, and the system was very hard to maintain. For performance and availability reasons, the SDP runs on two Sun multiprocessor computers. The distribution is handled in the application code, i.e., no commercial cluster software (or hardware) is used.

21.3 Studies and Experiences

In this section we present the experiences and studies that we have done on the different applications. The presentation in this section summarizes the major results in a number of previous investigations. The complete report of each individual investigation is included in the reference list at the end of the chapter. The combined experiences from all of the studies, i.e., the major result in this chapter, are provided in Section 4.

21.3.1 Billing Gateway (BGw)

We did a number of multiprocessor performance evaluations on the BGw. Since the system is multithreaded and since there are a lot of parallelism (there can be up to 100 network elements connected to each BGw), we expected a good speedup. The actual speedup was, however, very disappointing [8]. In the first version the performance actually dropped when the number of processors increased because the dynamic memory management system, i.e., the global heap, was a major bottleneck.

It turned out that the excessive use of dynamic memory was caused by the designers ambition to create a flexible and maintainable system, i.e., this ambition lead to a fine grained object structure where anticipated changes are isolated within a small (sub-)object. One result of this is that each the processing of each CDR involves a large number of allocations and deallocations of dynamic memory.

It transpired that a parallel heap implementation, e.g., SmartHeap for SMPs [17], ptmalloc [7] or hoard [2], could solve the dynamic memory problem. We also evaluated application specific object pools. The application specific object pools resulted in much higher performance (roughly a factor of eight) compared to the parallel heap implementations. However, the parallel heap implementations resulted in an order of magnitude higher performance than the original implementation. The obvious drawback with the application specific object pools is that they are hard to maintain as new versions of the BGw application are developed.

We also did a study of the BGw where the system was broken up into components [13]. The reason for breaking the system into components was to decrease the maintainability cost, i.e., each change would be isolated to a component. Another effect of the component-based

approach was that the system could be easily split into a number of processes that could be distributed to different computers. It turned out that the component-based prototype had much less problems with dynamic memory contention than the original (monolithic) implementation, i.e., in this case there was no conflict between maintainability and performance.

21.3.2 Fraud Control Center (FCC)

The experiences from the FCC are similar to those from the BGw [9]. The FCC had major speedup problems due to the dynamic memory management system. We initially evaluated two ways of solving the dynamic memory problem in the FCC:

- We replaced the ordinary heap with a parallel heap (ptmalloc).
- We changed the design of the FCC from one multithreaded program to a number of (single-threaded) processes.

The performance characteristics of these two solutions were quite similar, and both solutions resulted in a much better speedup compared to the original implementation.

We then did a more detailed study of the FCC and found that most of the dynamic memory problems came from a parser object in the FCC [10]. Interviews with the designers showed that the excessive use of dynamic memory was again caused by their ambition to build a maintainable and flexible system. The interviews also showed that the designers were rather happy with the result, i.e., they thought that the parser was maintainable and could be easily adapted to new formats. The interviews also showed that the designers had not anticipated any performance problems due to their design.

We now implemented an alternative parser design. This design was very rigid (and thus small), i.e., one could not adapt it to other formats. Consequently, in order to support new formats the parser had to be completely redesigned. It turned out that the rigid parser had eight times higher performance than the original flexible parser. Even more interestingly, it turned out that the rigid parser was eight times smaller than the flexible parser. Small code size is one of the most important qualities for reducing the maintainability cost. State-of-the-art maintainability estimation techniques showed that the cost of writing a new small and rigid parser was probably less than the cost of adapting the large flexible parser to a new format. Consequently, the maintainability of the small and rigid approach was at least as good as the maintainability of the flexible approach. Consequently, in this case the conflict between performance and maintainability was based on myths and misconceptions, and could thus be avoided.

21.3.3 Data Monitoring (DMO)

It turned out that the DMO version (DMO 1) that used a commercial database system and Solaris processes for expressing parallel execution had better multiprocessor scale-up than the multithreaded version (DMO 2). Again, the reason for the poor scale up of the multithreaded version was problems with the dynamic memory. The single-processor performance of DMO 2 was, however, higher than the single-processor performance of DMO 1. The experiences from DMO also show that commercial database systems seem to be highly optimized and do not cause any performance problems on SMPs. The use of a commercial database system decreased the size of the application program and will thus surely decrease the maintainability cost.

21.3.4 Prepaid Service Data Point (SDP)

The BGw and FCC applications were originally optimized for high maintainability. The SDP is, however, highly optimized for high performance and availability. One implication of the optimization efforts was that no commercial DataBase Management Systems (DBMS) was used; instead of using a DBMS the designers implemented the database functionality as part of the application code. Another implication of the performance optimization was that the designers implemented fault tolerance, i.e., redundant distributed computers and components, as part of the application code, instead of using standard cluster solutions.

The SDP was very hard to maintain. In order to reduce the amount of code, and thus the maintenance cost and time to market for new versions of the SDP, we developed a version that used standard DBMS and cluster solutions. The code reduction was dramatic; the new version (with the same functionality) has about 15,000 lines of code, which is an order of magnitude less than the code size of the original SDP. The performance of the new version is also acceptable (it was in fact somewhat better than the performance of the original version).

In order to facilitate this kind of dramatic size reduction without jeopardizing the performance and availability requirements, we had to use high-end hardware and software. The most spectacular example was the use of a fault tolerant computer from Sun. We did, however, develop a simple set of guidelines and a design strategy that minimized the additional hardware and software cost; the hardware and third party software cost of the new SDP is only 20-30% the hardware and third party software cost of the old SDP.

21.4 Results

In this section we present the accumulated experiences from our studies.

The experiences from the rigid versus flexible parser in the FCC and the component-based BGw show that there need not always be conflicts between maintainability and performance. The DMO experience shows that there was a conflict between single-processor performance and maintainability, but this conflict is reduced on multiprocessors since the more maintainable version (DMO 1) has better multiprocessor scale-up. The other examples show, however, that there seem to be situations where the ambition of obtaining high maintainability results in unacceptable poor performance, and the ambition to obtain high performance results in acceptable poor maintainability.

The discussion in the previous section shows that we have investigated three different ways of handling such conflicts:

- *By defining design methods and guidelines for obtaining acceptable performance without seriously degrading maintainability.*
Based on experience from a number of large industrial applications, we have defined a set of ten design guidelines and a process defining how to apply these guidelines to the application [11]. The process is primarily designed for migrating existing applications, which have been developed for single-processors, to multiprocessors. The process can, however, be useful also when developing new multiprocessor applications. The design guidelines are very effective, but some designers do not like the additional restriction of having a set of guidelines, and they may in some cases ignore the guidelines; failure to follow the guidelines may result in serious performance problems.
- *By developing resource allocation algorithms and implementation techniques that guarantee acceptable performance for object-oriented programs that are designed for maximum maintainability.*
We have successfully used this approach on object-oriented programs which tend to use excessive dynamic memory due to design optimizations with respect to maintainability. The dynamic memory problem could to, some extent, be reduced using an optimized dynamic memory handler, e.g., SmartHeap for SMP [17], ptmalloc [7], and hoard [2]. We have, however, found that these techniques are not always sufficient. The experiences from the application specific object pools in the BGw show that there is room for significant performance improvements. Inspired by this observation, we have developed an automatic method and that can obtain better performance than the parallel heap approaches. The performance improvement is obtained by using compile time information, which

is not possible in the dynamic heap approach. The result of this line of research is a technique called Amplify [20].

- *By using modern execution platforms, e.g., multiprocessors and disk-arrays, that will guarantee acceptable performance without sacrificing the maintainability aspect.*

The experiences from the SDP, FCC and DMO projects show that modern DBMS are often efficient, and it is difficult to improve performance by implementing the database functionality as part of the application program. Moreover, commercial DBMS often use multiprocessors and disk-arrays very efficiently. The effect of this is that one can often obtain the desired level of performance by buying a larger multiprocessor, which is generally a very cost-effective alternative to complicated performance optimizations in the application code.

We have also developed guidelines on how to partition the application into different parts and then apply the high-end hardware and third party software to the parts where it is really needed. In short, the guidelines say that one should first separate the performance critical parts from the non-performance critical parts. The performance critical parts should then further be divided into a state-less part and a state-full part, where one should try to keep the state-full part as small as possible. The high-end hardware and third party software components are then concentrated to the state-full and performance critical part. These guidelines minimize the extra cost for hardware and third party software, and in the SDP application the additional cost was limited to 20-30% considering the significant reduction of the maintainability cost.

21.5 Discussion

Others have also identified the need for design trade-offs. John Hennessy writes *Performance – long the centerpiece – needs to share the spotlight with availability, maintainability, and other quality attributes* in his article on The Future of System Research [12]. The Architecture Trade-off Analysis Method (ATAM), proposed by Kazman et al. [16], addresses the need for making trade-offs between different quality attributes. One important difference between our guidelines and ATAM is that the ATAM work concentrates on identifying so called trade-off points, i.e., design decisions that will affect a number of quality attributes. There are no guidelines in ATAM on how to modify the software architecture.

Conflicts between quality attributes such as performance and maintainability have sometimes been attacked early in the design process, at

the requirements level. Boehm and In have developed a knowledge-based tool that helps users, developers, and customers analyze requirements and identify conflicts among them [4].

Previous experiences with flexible database systems [5] show that there are similar trade-offs between maintainability and flexibility on the one hand and performance on the other hand also for applications outside of the telecommunication domain.

A large research project on trade-offs between different software qualities, such as performance, maintainability, time-to-market and usability is performed at Blekinge Institute of Technology. The project (called BESQ [3]) involves four senior researchers and more than 10 Ph.D. students. One of the subprojects within BESQ suggests a novel approach to balancing performance and maintainability when developing and porting real-time operating system kernels for multiprocessors [15]. The fundamental idea in that case is to increase maintainability by using the same kernel code for both the multiprocessor and the single-processor version of the real-time kernel. The multiprocessor adaptations are done in a separate layer outside of the kernel.

Some of the dynamic memory problems were caused by the fine-grained object-oriented design. State-of-the-art design techniques are based on design pattern, and up to recently the design patterns did not consider performance and multiprocessor issues. There are a number of recent papers that identify this problem and discuss new and adapted design patterns. Douglas C. Schmidt is one of the leading researchers in this area. More information about the research activities in this filed can be found on his homepage [18].

A lot of researchers have looked at efficient implementations of dynamic memory, Wilson et al have written an excellent survey [19]. The main focus in this area has, however, been memory allocators for sequential programs. Some programming languages, e.g., Java, use garbage collection for unused dynamic memory. Extensive research has been done on garbage collection techniques, incremental and concurrent garbage collection in particular [14].

21.6 Conclusions

To sum up, we have during a period of six years studied four large industrial real-time applications where performance and maintainability are important quality attributes. The average size of these applications is approximately 100,000 lines of code. The experiences show that there can indeed be conflicts between maintainability and performance. Some of these conflicts are based on myths and misconceptions, but some are inherent. The inherent conflicts can in some cases be handled by using

modern hardware and/or optimized resource allocation algorithms and techniques. In some other cases, one must obtain a reasonable trade-off between maintainability and performance, and we have defined a set of guidelines and a simple development process for obtaining such trade-offs.

The fact that performance and maintainability are, to some extent, exchangeable puts software (and hardware) real-time performance engineering in a new and interesting perspective. The relevant performance related question is not only if the system meets its performance requirements using a certain software design and architecture on a certain hardware and operating system platform. An equally interesting question is if the system can be made more maintainable by changing the software architecture and compensating this with modern hardware and/or optimized resource allocation algorithms and techniques.

Bibliography

- [1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.
- [2] E.D. Berger and R.D. Blumofe, *Hoard: A Fast Scalable, and Memory Efficient Allocator for Shared Memory Multiprocessors*, <http://www.hoard.org>.
- [3] *BESQ, Blekinge Engineering Software Qualities*, <http://www.ipd.bth.se/besq/>
- [4] B. Boehm and H. In, *Identifying Quality-Requirement Conflicts*, IEEE Software, March 1996.
- [5] Wolfgang Diestelkamp and Lars Lundberg, *Performance Evaluation of a Generic Database System*, International Journal of Computers and Their Applications, Vol. 7, No. 3 September 2000. ISSN 1076-5204.
- [6] E. Gamma, R. Helm, R. Johnson, and Vilssides, *Design Patterns*, Addison-Wesley, 1997.
- [7] W. Gloger, *ptmalloc homepage*, <http://www.malloc.de/en/index.html>
- [8] D. Haggander and L. Lundberg, *Optimizing Dynamic Memory Management in a Multithreaded Application Executing on a Multiprocessor*, in Proceedings of the 27th International Conference on Parallel Processing, Minneapolis, USA, August 1998.

- [9] D. Haggander and L. Lundberg, *Memory Allocation Prevented Telecommunication Application to be Parallelized for Better Database Utilization*, in Proceedings of the 6th International Australasian Conference on Parallel and Real-Time Systems, Melbourne Australia, November 1999.
- [10] D. Haggander, PO Bengtsson, J. Bosch, and L. Lundberg, *Maintainability Myth Causes Performance Problems in SMP Applications*, in Proceedings of the 6th Asian-Pacific Conference on Software Engineering, Takamatsu, Japan, December 1999.
- [11] D. Haggander and L. Lundberg, *A Simple Process for Migrating Server Applications to SMPs*, Journal of Systems and Software, Vol 57/1, May 2001, pp. 31-43.
- [12] J. Hennessy, *The Future of System Research*, IEEE Computer, August 1999
- [13] H. Hermansson, M. Johansson, and L. Lundberg, *A Distributed Component Architecture for a Large Telecommunication Application*, in Proceedings of the 7th Asian-Pacific Conference on Software Engineering, Singapore, December 2000.
- [14] R. Jones and R. Lins, *Garbage Collection: Algorithms for automatic dynamic memory management*, John Wiley and Sons, 1998.
- [15] S. Kagstrom, L. Lundberg, H. Grahn, *A novel method for adding multiprocessor support to a large and complex uniprocessor kernel*, in Proceedings of IPDPS 2004, April 2004, to appear.
- [16] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, *The Architecture Tradeoff Analysis Method*, Tech. Report, CMU/SEI-98-TR-008.
- [17] Microquill, *SmartHeap for SMP*,
<http://www.microquill.com/smartheapsmp/index.html>
- [18] D. Schmidt, *Douglas C. Schmidt homepage*,
<http://www.cs.wustl.edu/schmidt>
- [19] P. Wilson, M. Johnstone, M. Neely, D. Boles, *Dynamic storage allocation: A survey and critical review*, in Proceedings of the 1995 International Workshop on Memory Management, Kinross, Scotland, 1995 (Springer Verlag)
- [20] D. Haggander, P. Liden, and L. Lundberg, *A Method for Automatic Optimization of Dynamic Memory Management in C++*, in Proceedings of the International Conference on Parallel Processing (ICPP), Valencia, Spain, September, 2001.

Chapter 22

Memory System Behavior of Java-Based Middleware

By **Martin Karlsson**[†], **Kevin E. Moore**[‡],
Erik Hagersten[†], and **David A. Wood**[‡]

[‡]Department of Computer Sciences
University of Wisconsin
Email: {kmoore,david}@cs.wisc.edu

[†] Division of Computer Systems
Department of Information Technology
Uppsala University
Email: {martink,eh}@it.uu.se

Java-based middleware, and application servers in particular, are rapidly gaining importance as a new class of workload for commercial multiprocessor servers. SPEC has recognized this trend with its adoption of SPECjbb2000 and the new SPECjAppServer2001 (ECperf) as standard benchmarks. Middleware, by definition, connects other tiers of server software. SPECjbb is a simple benchmark that combines middleware services, a simple database server, and client drivers into a single Java program. ECperf more closely models commercial middleware by using a commercial application server and separate machines for the different tiers. Because it is a distributed benchmark, ECperf provides an opportunity for architects to isolate the behavior of middleware. In this paper, we present a detailed characterization of the memory system behavior of ECperf and SPECjbb using both commercial server hardware and Simics full-system simulation. We find that the memory footprint and primary working sets of these workloads are small compared to other commercial workloads (e.g., on-line transaction processing), and that a large fraction of the working sets are shared between processors. We observed two key differences between ECperf and SPECjbb that highlight the importance of isolating the behavior of the middle tier. First, ECperf has a larger instruction footprint, resulting in much

higher miss rates for intermediate-size instruction caches. Second, SPECjbb's data set size increases linearly as the benchmark scales up, while ECperf's remains roughly constant. This difference can lead to opposite conclusions on the design of multiprocessor memory systems, such as the utility of moderate sized (i.e., 1 MB) shared caches in a chip multiprocessor.

22.1 Introduction

Architects have long considered On-Line Transaction Processing (OLTP) and Decision Support Systems (DSS) as important workloads for multiprocessor servers. The recent shift toward 3-tier and N-tier computing models has created a large and rapidly-growing market for Java-based middleware, especially application servers. Still, middleware workloads are not yet well understood, and there are few accepted benchmarks that measure the performance of middle-tier applications. This is due both to the recent emergence of middleware as a mainstream workload and to the fact that 3-tier workloads are by nature difficult to install, tune and run.

We present a detailed characterization of two Java-based middleware benchmarks, SPECjbb and ECperf (now SPECjAppServer2001 [SSGS01]), running on shared-memory multiprocessors. ECperf more closely resembles commercial middleware applications because it runs on top of a commercial application server and is deployed on a 3-tiered system. The distributed nature of ECperf also facilitates monitoring the behavior of each tier independently. ECperf, however, is difficult to install and run. It requires the coordination of several machines and several pieces of software. SPECjbb is also a Java middleware benchmark. It is an attractive alternative to ECperf because although it models a 3-tiered system, it is a single Java program that can be run on any Java Virtual Machine (JVM). SPECjbb includes many common features of 3-tiered systems in a single program running on a single machine.

The goal of this paper is to understand the memory system behavior of these middleware benchmarks, to gain insight into the behavior of Java-based middleware, and to provide useful data and analysis to memory systems designers targeting middle-tier servers. We focus on mid-range (up to 16 processor) shared-memory multiprocessors because many application servers target these systems. We also investigate whether or not the simple SPECjbb benchmark behaves similarly enough to the more complex ECperf to be considered representative of commercial middleware applications. We find that these Java-based middleware applications have moderate CPIs compared to previously-published commercial workloads (between 2.0 and 2.8 for ECperf). In particular, memory related stalls are low, with misses to main mem-

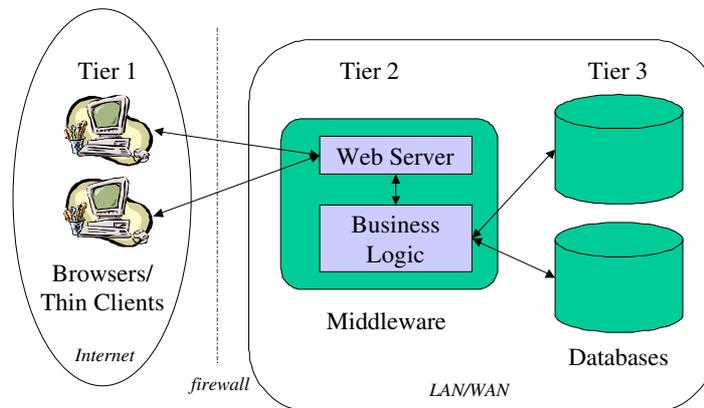


Figure 22.1: 3-Tiered System

ory accounting for as little as 15% of the data stall time and 5% of total execution time. Conversely, sharing misses occur frequently in both workloads, accounting for over 60% of second-level cache misses on larger systems. SPECjbb is similar to ECperf in many ways, but there are important differences between the two benchmarks. ECperf has a larger instruction working set, but a lower data cache miss rate. Furthermore, the memory footprint of ECperf remains nearly constant as the benchmark scales up, whereas the memory use of SPECjbb grows linearly with database size. We show that this difference can lead to opposite conclusions on some design decisions, like the utility of shared level-two caches in a chip multiprocessor.

22.2 Background

The emergence of the Internet and World Wide Web has triggered a shift in enterprise computing from a two-tiered, client-server architecture to a 3-tiered architecture (see Figure 22.1), where a Web browser is now used universally as a database client. For databases, connection to the Web allows users to access data without installing a client program. For Web pages, databases provide dynamic content and permanent storage. Software that connects databases to Web pages is known as "middleware." Much of the middleware used today is written in Java. Two of the most popular Java middleware architectures are Java Servlets and Enterprise Java Beans (EJB). The two are often used together, with Servlets implementing the presentation logic and EJB providing the business rules. Application servers host both Servlets and EJB and provide them with communication with both back-end databases and front-end web clients. Recently, Web-connected database applications have also been deployed

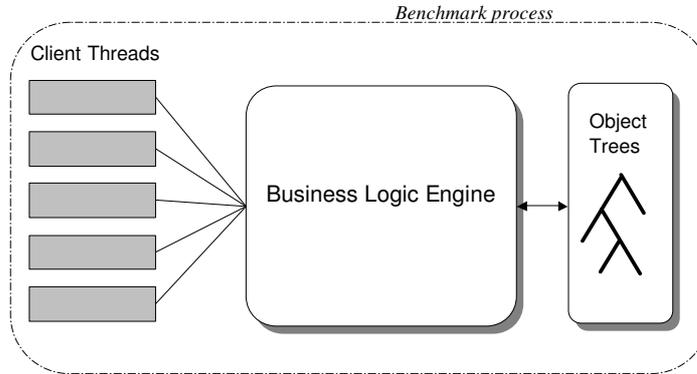


Figure 22.2: SPECjbb Overview

in an "N-Tier" architecture in which the presentation logic is separated from the business rules. The presentation logic can be implemented by stateless servers and is sometimes considered to be a first-tier application. N-Tiered architectures allow the application server to focus entirely on the business logic.

22.2.1 SPECjbb Overview

SPECjbb is a software benchmark designed to measure a system's ability to run Java server applications. Inspired by the On-Line Transaction Processing Benchmark TPC-C, SPECjbb models a wholesale company with a variable number of warehouses. Beyond the nomenclature and business model, however, there are few similarities between TPC-C and SPECjbb. TPC-C is intended to measure the performance of large-scale transaction processing systems, particularly databases. In contrast, SPECjbb was written to test the scalability and performance of JVMs and multiprocessor servers that run Java-based middleware. It emphasizes the middle-tier business logic that connects a back-end data store to a set of thin clients, and is implemented entirely in Java.

SPECjbb models a 3-tiered system, but to make the benchmark portable and easy to run, it combines the behavior of all 3 tiers into a single application (see Figure 22.2). Instead of using a commercial database engine like most real 3-tiered systems, SPECjbb stores its data in memory as trees of Java objects [SPE]. The SPECjbb specification calls for running the benchmark with a range of warehouse values. In an official SPECjbb run, the benchmark is run repeatedly with an increasing number of warehouses until a maximum throughput is reached. The benchmark is then run the same number of times with warehouse values starting at the maximum and increasing to twice that value. Therefore,

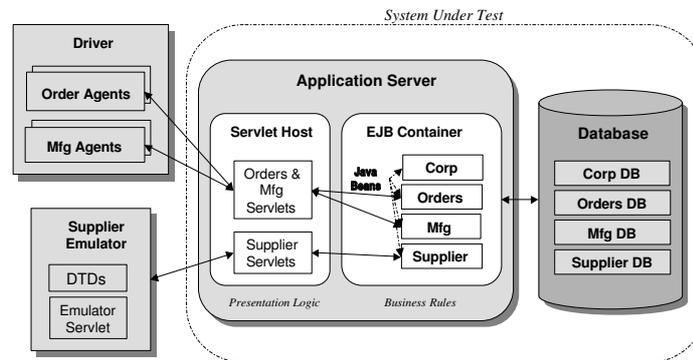


Figure 22.3: ECperf Setup

if the best throughput for a system comes with n warehouses, $2n$ runs are made. The benchmark score is the average of runs from n to $2n$ warehouses. This large number of separate benchmark runs would take prohibitively long in simulation. Therefore, in our simulation experiments, we selected 3 values for the number of warehouses to represent the range of values that would be included in a publishable SPECjbb result for our hardware configuration. In order to simplify our monitoring simulations, we report results from the steady state interval of SPECjbb running with the optimal number of warehouses at each system size.

22.2.2 ECperf Overview

ECperf is a middle-tier benchmark designed to test the performance and scalability of a real 3-tier system. ECperf models an on-line business using a "Just-In-Time" manufacturing process (products are made only after orders are placed and supplies are ordered only when needed). It incorporates e-commerce, business-to-business, and supply chain management transactions. The presentation layer is implemented with Java Servlets, and the business rules are built with EJB. The application is divided into the following four domains, which manage separate data and employ different business rules. The Customer Domain models the actions of customers who create, change and inquire about the status of orders. The customer interactions are similar to On-Line Transaction Processing (OLTP) transactions. The Manufacturing Domain implements the "Just-In-Time" manufacturing process. As orders are filled, the status of customer orders and the supply of each part used to fill the order are updated. The Supplier Domain models interactions with external suppliers. The parts inventory is updated as purchase orders are filled. Finally, the Corporate Domain tracks customer, supplier and parts information. The ECperf specification supplies the EJB compo-

nents that form the core of the application. These components implement the application logic that controls the interaction between orders, manufacturing and suppliers. In particular, that interaction includes submitting various queries and transactions to the database, and exchanging XML documents with the Supplier Emulator. Four separate agents participate in the ECperf benchmark, each of which is run on a separate machine or group of machines. Each of these parts is represented by a box in Figure 22.3.

Application Server The application server, shown in the center of Figure 22.3, hosts the ECperf Java Beans. Together, they form the middle tier of the system, which is the most important component to performance on ECperf.

Database The next most important part of the system, in terms of performance, is the database. Though ECperf does not overly stress the database, it does require the database to keep up with the application server and provide atomic transactions.

Supplier Emulator Suppliers are emulated by a collection of Java Servlets hosted in a separate web container.

Driver The driver is a Java program that spawns several threads that model customers and manufacturers. Each high-level action in ECperf, such as a customer making a new order, or a manufacturer updating the status of an existing order, is called a "Benchmark Business Operation," or "BBop." Performance on ECperf is measured in terms of BBops/minute. Although performance on ECperf is measured in terms of throughput, the benchmark specification requires that 90% of all transactions are completed within a fixed time [KMHW02][Pag]. In our experiments, however, we relaxed the response time requirement of ECperf and tuned our system to provide the maximum throughput regardless of response time.

22.2.3 Enterprise Java Beans

ECperf is implemented using Enterprise Java Beans (EJB), a part of the Java 2 Enterprise Edition (J2EE) standard. EJB are reusable Java components for server-side applications. In other words, they are building blocks for web-service applications. They are not useful until they are deployed on an application server. Inside the server, an EJB "container" hosts the beans and provides important services. In particular, EJB rely on their containers to manage connections to the database, control access to system resources, and manage transactions between

components. Often the container is also responsible for maintaining the persistent state of the beans it hosts. The application server controls the number of containers and coordinates the distribution of client requests to the various instances of each bean.

22.2.4 Java Servlets

Servlets are Java classes that run inside a dynamic web server. Servlets can communicate with a back-end database through the Java DataBase Connectivity (JDBC) API. Session information can be passed to Servlets either through browser cookies or URL renaming.

22.2.5 Java Application Servers

To host ECperf, we used a leading commercial Java-based application server. That server can function both as a framework for business rules (implemented in EJB) and as a host for presentation logic, including Java Servlets. As an EJB container, it provides required services such as database connections and persistence management. It also provides better performance and scalability than a naive implementation of the J2EE standard. Three important performance features of our particular server are thread pooling, database connection pooling, and object-level caching. The application server creates a fixed number of threads and database connections, which are maintained as long as the server is running. The application server allocates idle threads or connections out of these pools, rather than creating new ones and later destroying them when they are no longer needed. Database connections require a great deal of effort to establish and are a limited resource on many database systems. Connection pooling increases efficiency, because many fewer connections are created and opened. In addition, connection pooling allows the application server to potentially handle more simultaneous client sessions than the maximum number of open connections allowed by the database at any time. Thread pooling accomplishes the same conservation of resources in the Operating System that database connection pooling does in the database. Our experience tuning the application server showed that configurations with too many threads spend much more time in the kernel than those that are well tuned. Object-level caching increases performance in the application server because instances of components (beans) are cached in memory, thereby reducing database queries and memory allocations. The application server used in this study is one of the market leaders (we are not able to release the name due to licensing restrictions). In all of our experiments, a single instance of the application server hosted the entire middle tier. Many commercial application servers, including ours, provide a cluster-

ing mechanism that links multiple server instances running on the same or different machines. The scaling data presented in section 4 does not include this feature and only represents the scaling of a single application server instance, running in a single JVM.

22.3 Methodology

We used a combination of monitoring experiments on real hardware and detailed full-system simulation to measure the memory system behavior of our middleware workloads. The native hardware enabled us to perform our measurements on a complete run of the benchmarks while our simulation study offered us the opportunity to change the memory system parameters. On the native hardware, we used the Solaris tool `psrset` to restrict the application threads to only run on a subset of the processors available on the machine. The `psrset` mechanism also prevents other processes from running on processors within the processor set. This technique enabled us to measure the scalability of the applications and to isolate them from interference by other applications running on the host machine.

22.3.1 Hardware Setup

We ran both SPECjbb and the application server of ECperf on a Sun Enterprise 6000 server. The E6000 is a bus-based snooping multiprocessor with 16 248-MHz UltraSPARC II processors with 1 MB L2 caches and 2 GB of main memory. The UltraSPARC II processors are 4-wide and in-order issue. For ECperf, we ran the database on an identical Sun E6000, and the supplier emulator and driver were each run on a 500 MHz UltraSPARC IIe Sun Netra. All the machines were connected by a 100-Mbit Ethernet link.

22.3.2 Benchmark Tuning

Tuning Java server workloads is a complicated process because there are several layers of software to configure, including the operating system, the JVM, and the application itself. Tuning 3-Tier Java applications is more complicated still, because the application server and database must be properly configured as well.

Operating System (Solaris 8) We optimized Solaris for running large server programs by enabling Intimate Shared Memory (ISM), which increases the page size from 8 KB to 4 MB and allows sharing of page table entries between threads. This optimization greatly increases the TLB reach, which would otherwise be much smaller than the application

server's large heap.

JVM (HotSpot 1.3.1) We configured the JVM by testing various thread synchronization and garbage collection settings. We found that the default thread synchronization method gave us the best throughput on ECperf and SPECjbb. In all cases, the heap size was set to the largest value that our system could support, 1424 MB. We tuned the garbage collection mechanism in the virtual machine by increasing the size of the new generation to 400 MB. A large new generation leads to fewer, but longer, partial collections and better total throughput. Our multiprocessor simulations of SPECjbb were run with HotSpot 1.4.0. In order to be as consistent as possible with both our uniprocessor simulations and the multiprocessor simulations of ECPerf, we used the same heap and new generation sizes in all of our experiments.

Application Server For ECperf, we tuned the application server for each processor set size by running the benchmark repeatedly with a wide range of values for the size of the execution queue thread pool and the database connection pool. For each processor count, the configuration settings used were those that produced the best throughput.

Database ECperf uses a small database, which fit entirely in the buffer pool of our database server. We found that the performance of ECperf was unaffected by other database settings.

22.3.3 Simulation Environment

We used the Simics full-system simulator [MCJE⁺02] to simulate ECperf and SPECjbb running on several different system configurations. Simics is an execution-driven simulator that models a SPARC V9 system accurately enough to run unmodified Solaris 8. To determine the cache behavior of the applications without communication, we configured Simics to model a 1-processor E6000-like SPARC V9 system with 2 GB of main memory running Solaris 8. To run ECperf, we simulated four such machines connected by a simulated 100-Mbit Ethernet link. The reported cache statistics for ECperf were taken from the simulated machine that ran the application server. For these experiments we extended Simics with a detailed memory system simulator [MLLL02]. The memory system simulator allowed us to measure several cache performance statistics on a variety of caches with different sizes, associativities and block sizes. In order to evaluate the communication behavior of these workloads and their suitability to a shared-cache memory system, we also simulated multiprocessor configurations of each workload. We were not able to simulate a multi-tiered configuration of ECperf running on

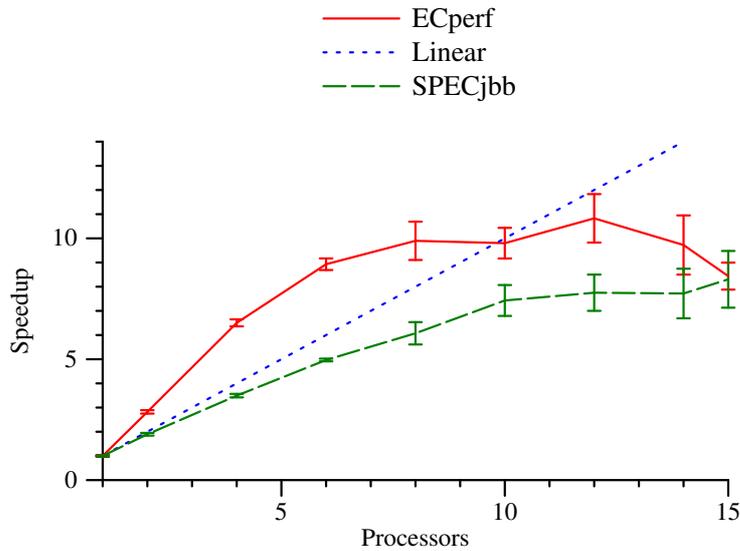


Figure 22.4: Throughput Scaling on a Sun E6000

a multiprocessor. Instead, we simulated a single 16-processor machine where the application server was bound to 8 processors. We then filtered out the memory requests from the other 8 processors, and fed only the requests from the application server processors to our memory system simulator. We use the methodology proposed by Alameldeen, et al. [AW03] to account for the inherent variability of multithreaded commercial workloads. We present the means and standard deviations (shown as error bars) for all measured and most simulated results.

22.4 Scaling Results

Java-based middleware applications, like most commercial workloads, are throughput-oriented. Understanding how these applications scale up to both larger multiprocessors and larger data sets is important for both hardware and software developers. In this section, we analyze how ECperf and SPECjbb scale on a Sun E6000 system. Despite our best efforts to tune these workloads, we were unable to even come close to achieving linear speedup. Figure 22.4 shows that ECperf scales super-linearly from 1 to 8 processors, but scales poorly beyond 12 processors. ECperf achieves a peak speedup of approximately 10 on 12 processors, then performance degrades for larger systems. SPECjbb scales up more gradually, leveling off after achieving a speedup of 7 on 10 processors. In the remainder of this section we present an analysis of the factors that contribute to the limitations on scaling. We find that both benchmarks experience significant idle time (approximately 25%) for systems with

10 or more processors, apparently due to contention for shared software resources. Memory system stalls are the second major factor, causing the average cycles per instruction to increase by as much as 40%. Finally, although garbage collection does impact performance, on larger systems it accounts for only a fraction of the difference between measured and linear speedup.

22.4.1 Resource Contention

We used a variety of Solaris measurement tools to identify the bottlenecks in ECperf and SPECjbb. Figure 22.5 shows a breakdown of the time spent in various execution modes as measured by the Solaris tool `mpstat`. The four modes are running the operating system (system), running the benchmark (user), stalled for I/O (I/O), and stalled for other reasons (idle). Figure 22.5 illustrates one important difference between ECperf and SPECjbb. ECperf spends significant time in the operating system, while SPECjbb spends essentially none. This is not surprising, since SPECjbb emulates all three tiers on a single machine, using memory-based communication within a single JVM. Conversely, ECperf uses separate machines for each tier, requiring communication via operating system-based networking code. For ECperf, the system time increase from less than 5% for a single-processor run, to nearly 30% for a 15-processor system. We hypothesize, but have been unable to confirm, that the increase in system time arises from contention in the networking code. Both workloads incur significant idle time for larger system sizes, reaching 25% for 15 processors. Some of this idle time is due to garbage collection. Like most currently available systems, the JVM we ran uses a single-threaded garbage collector. That is, during collection only 1 processor is active and all others wait idle. We estimated the fraction of idle time due to garbage collection by multiplying the fraction of processors that are idle during collection by the fraction of time spent performing garbage collection. Figure 22.5 shows that the bulk of the idle time is due to factors other than garbage collection. The increase in idle time with system size suggests that there is contention for shared resources in these benchmarks. The application server in ECperf shares its database connection pool between its many threads, and the object trees in SPECjbb are protected by locks, both of which could lead to contention in larger systems. However, the fact that the idle time increases similarly for both benchmarks indicates that the contention could be within the JVM.

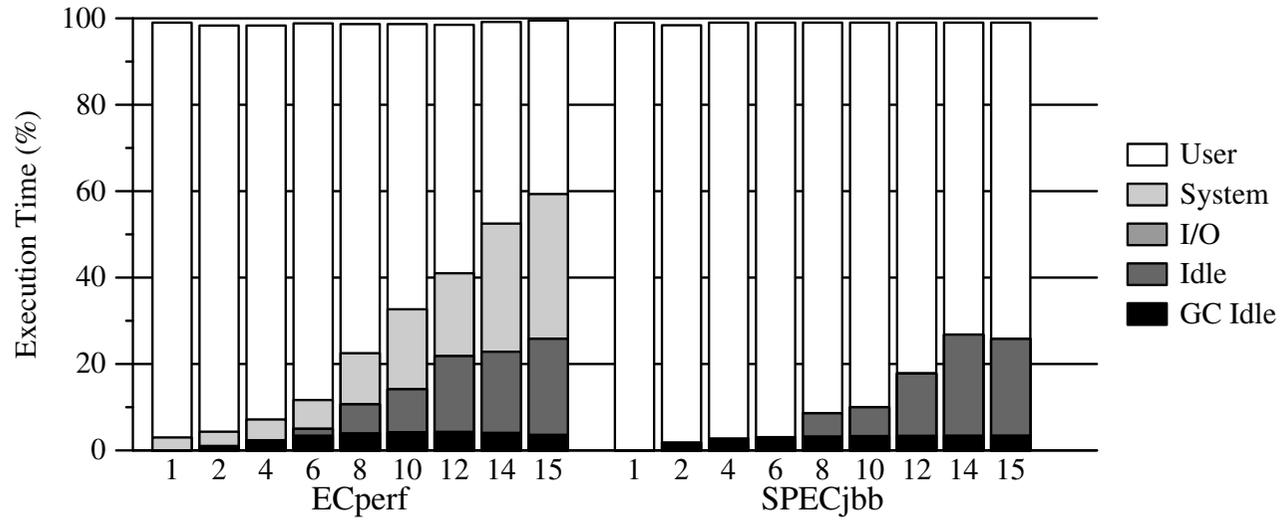


Figure 22.5: Execution Mode Breakdown vs. Number of Processors

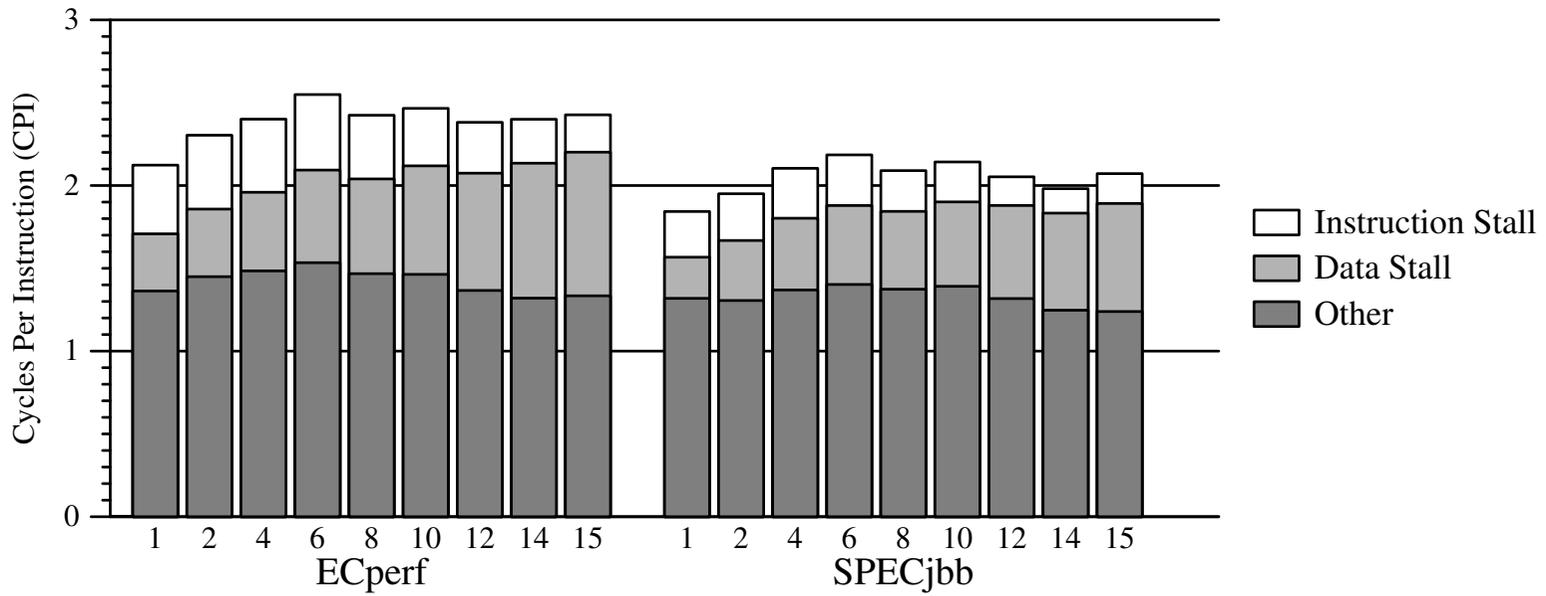


Figure 22.6: CPI Breakdown vs. Number of Processors

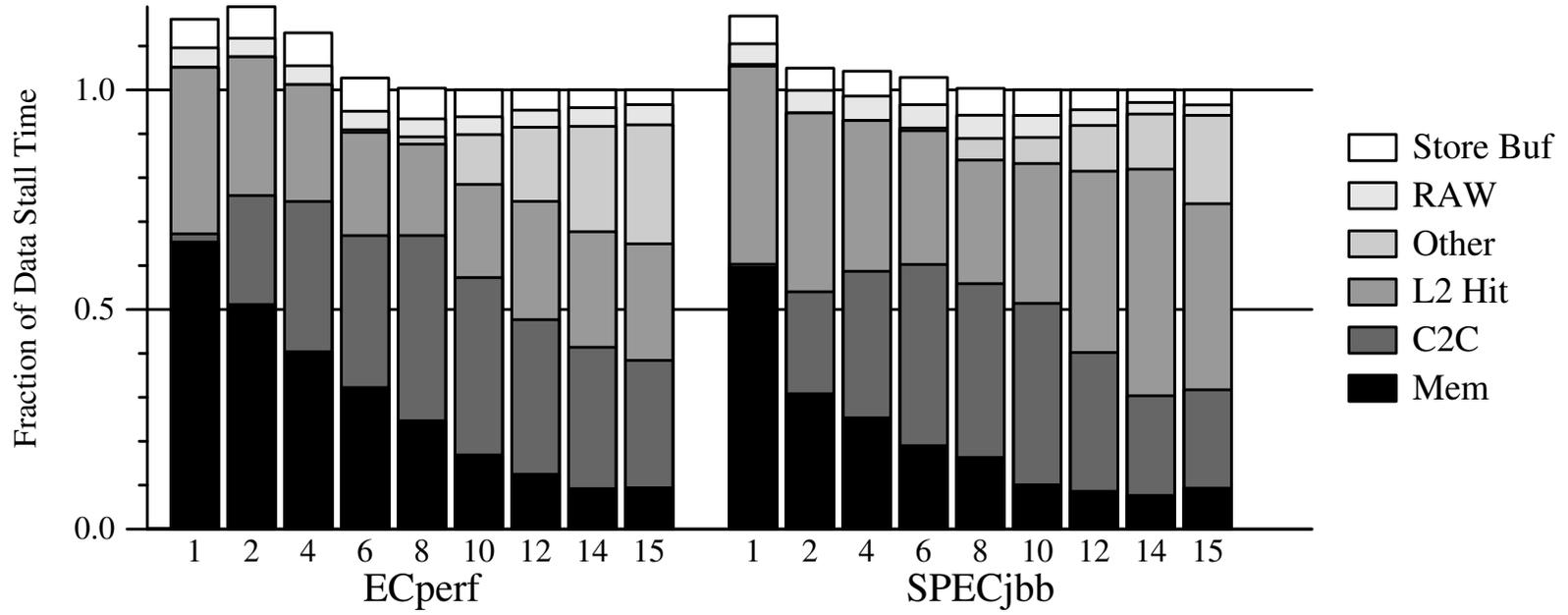


Figure 22.7: Data Stall Time Breakdown vs. Number of Processors

22.4.2 Execution Time Breakdown

Idle time alone explains at most half the degradation in speedup (75% non-idle time times 15 processors is approximately 11, not the 8 we observe). To identify other limits to scalability, we used the integrated counters on the UltraSPARC II processors to measure and breakdown the average cycles per instruction (CPI) across a range of system sizes. While CPI is not a good indicator of overall performance on multiprocessors —e.g., because of the effect of the idle loop— it gives a useful indication of where the time goes. Figure 22.6 presents the CPI, broken down into instruction stalls, data stalls, and other (which includes instruction execution and all non-memory-system stalls). The overall CPI ranges from 1.8 to 2.4 for SPECjbb and 2.0 to 2.8 for ECperf. These are moderate CPIs for commercial workloads running on in-order processors. Barroso, et al. report CPIs for Alpha 4100 systems of 1.3 to 1.9 for decision support database workloads and as high as 7 for a TPC-B on-line transaction processing workload. The CPI increases by roughly 40% and 33% for ECperf and SPECjbb, respectively, as the number of processors increase from 1 to 15. Assuming instruction path lengths remain constant (see Section 22.4.4.), the increase in CPI would account for most of the remaining performance degradation. Figure 22.6 also shows that data stall time is the main contributor to the increase in CPI. On a single processor run, data stall time accounts for only 15% and 12% for ECperf and SPECjbb, respectively. However for a 15-processor system, this increases to 35% and 25% for ECperf and SPECjbb, respectively. Figure 22.7 presents an approximate decomposition of the data stall time. Because some factors are estimated using frequency counts multiplied by published access times, the total does not always exactly sum to one. Approximately 60% of the data stall time is due to misses in the L2 cache, with the bulk of the remainder being L2 hits. Conversely, store buffer stalls, the cycles spent waiting for a full store buffer to be flushed, account for only 1% to 2% of the total execution time. Similarly, read-after-write hazard stalls, which occur if a load is not separated enough from a store, account for only 1% of the time.

22.4.3 Cache-to-Cache Transfer Ratio

Figure 22.7 also illustrates that cache-to-cache transfers represent a significant fraction of the data stall time for multiprocessor systems. For larger multiprocessors, cache-to-cache transfers account for nearly 50% of the total data stall time. Cache-to-cache transfers are an important factor because many multiprocessor systems take longer to satisfy a miss from a processor's cache than from main memory. On the E6000, the latency of a cache-to-cache transfer is approximately 40% longer than the latency of an access to main memory [GSSD00]. For NUMA memory

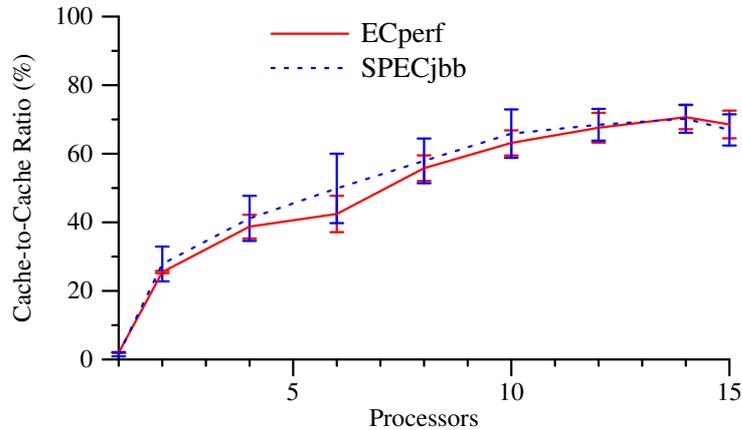


Figure 22.8: Cache-to-Cache Transfer Ratio

systems, this penalty is typically much higher-200-300% is not uncommon [CBJC02]-because of the indirection required by directory-based protocols. To dig deeper, we measured the cache-to-cache transfer ratio for SPECjbb and ECperf by counting the "snoop copyback" events reported in `cpustat`. In the UltraSPARC II processor, a snoop copyback event signifies that a processor has copied a cache line back to the memory bus in response to a request by another processor. Figure 22.8 shows that the fraction of L2 cache misses that hit in another cache starts at 25% for two processors and increases rapidly to over 60% for fourteen processors. This is comparable to the highest ratios previously published for other commercial workloads [BGB98]. Figure 22.8 also shows cache-to-cache transfers occur even for 1 processor. These transfers are possible because the operating system runs on all 16 processors, even when the application is restricted to a single processor. Snoop copybacks occur when the processor running the benchmark responds to a request from another processor running in the operating system.

22.4.4 Path Length

Comparing Figure 22.4 to Figure 22.6 reveals an apparent contradiction. ECperf scales super-linearly as the system size increases from 1 to 8 processors, even though the average CPI increases over the same range. This surprising result occurs because the instructions executed per BBop decreases even more dramatically over the same range (not shown). The decrease in instruction count more than compensates for the longer average execution time per instruction. We hypothesize that this drop is due to object-level caching in the application server. Constructive interference in the object cache allows one thread to re-use objects fetched by another thread.

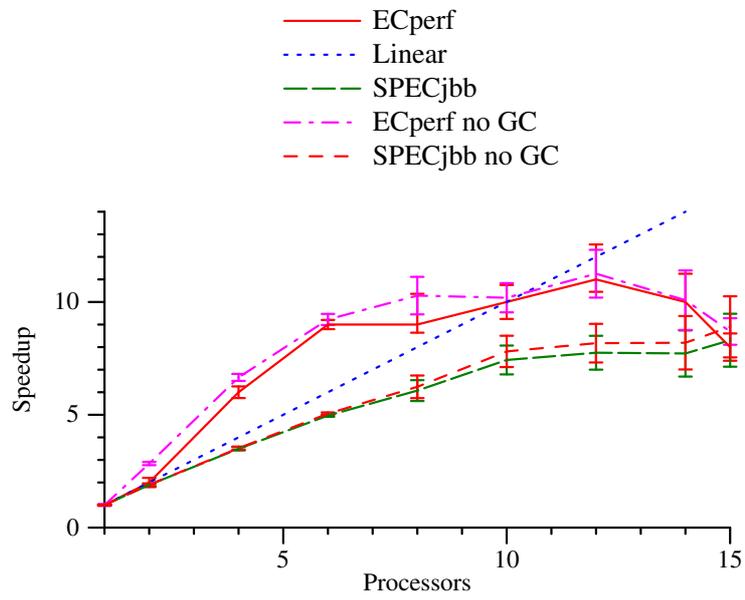


Figure 22.9: Effect of Garbage Collection on Throughput Scaling

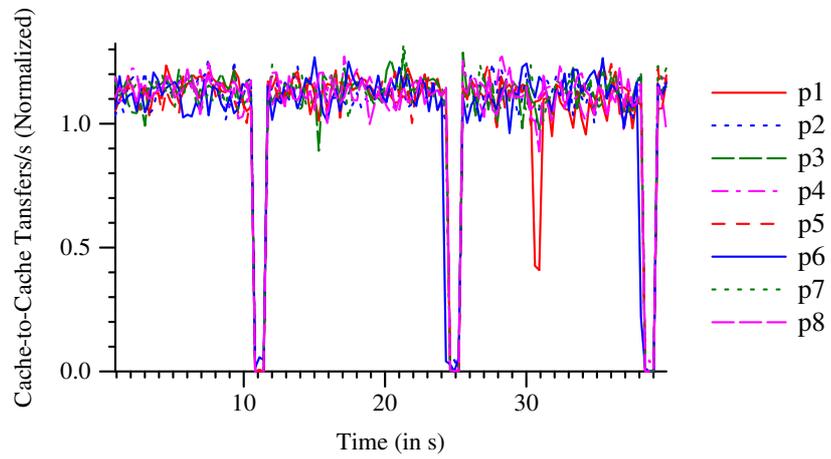


Figure 22.10: Cache-to-Cache Transfers Per Processor Per Second Over Time (Normalized)

22.4.5 Garbage Collection Effects

Both workloads spend a considerable amount of time doing garbage collection. To determine the impact of the collection time on scalability, we compared the measured speedup to the speedup with the garbage collection time factored out. That is, we subtracted the garbage collection time from the runtime of the benchmark and calculated speedup in the usual way. The solid lines in Figure 22.9 represent the speedup of ECperf and SPECjbb as measured. The dotted lines display the speedup of the benchmarks with the garbage collection time factored out. The difference in throughput with and without the garbage collection is small, but statistically significant for ECperf up to 6 processors. For SPECjbb and ECperf on larger systems, the difference is not statistically significant. We originally hypothesized that the high percentage of cache-to-cache transfers we observed in both SPECjbb and ECperf was due to garbage collection. Our JVM (HotSpot 1.3.1) uses a generational copying collector and is single-threaded. Therefore, during collection, all live new generation objects are copied by the collection thread regardless of which thread had created them and regardless of their location in the cache of a particular processor. For example, in a system that uses a simple MSI invalidation protocol, any new generation data in the M state cached at a processor that is not performing the collection will be read by the collector thread through a costly cache-to-cache transfer. This will result in the original copy of the data being invalidated. After the garbage collection is performed, the previous owner of the block will have to reacquire the block to access it. If the data is still residing in the garbage collector's cache, that access will result in another costly cache-to-cache transfer. Contrary to our hypothesis, the benchmark generates almost no cache-to-cache transfers during garbage collection. We counted the number of snoop copyback events every 100 ms during a run of SPECjbb. Figure 22.10 illustrates this dramatic drop in the cache-to-cache transfer rate during the 3 garbage collections that occurred in our measurement interval. The HotSpot 1.3.1 JVM has an option to trace the garbage collection in a program. We used that output to verify that the decreases in the snoop copyback rate occurred during garbage collection periods. Since our JVM uses a single-threaded garbage collector, only one processor is active during the collection. That by itself would explain a drop, but Figure 22.10 shows that the cache-to-cache transfer rate drops to almost zero during the garbage collection periods. Even the single processor which is performing the collection causes fewer cache-to-cache transfers.

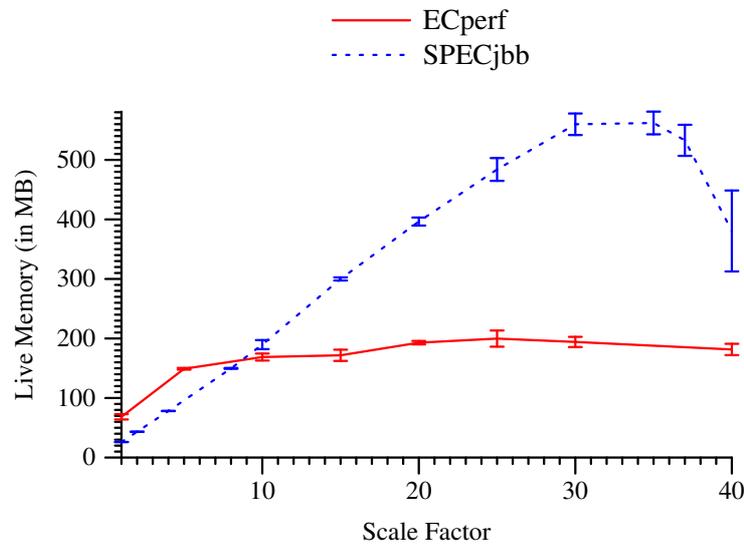


Figure 22.11: Memory Use vs. Scale Factor

22.4.6 Benchmark Scaling Differences

One of the most striking differences between SPECjbb and ECperf is the effect that scaling the benchmark size has on memory behavior. Like most commercial workload benchmarks, official measurements of SPECjbb and ECperf require that the benchmark size increase with input rate. In other words, faster systems must access larger databases. In SPECjbb, the input rate is set by the number of warehouses, which determines the number of threads in the program in addition to the size of the emulated database. ECperf has a similar scaling factor, the Orders Injection Rate. However, because the database and client drivers run on different machines, increasing the Orders Injection Rate has much less impact on the middle-tier memory behavior. Figure 22.11 shows the average heap size immediately after garbage collection in SPECjbb and ECperf. The size of the heap after collection is an approximation of the amount of live data. As the scale factor (i.e., warehouses) increases, SPECjbb's memory use increases linearly through approximately 30. Beyond 30 warehouses, the average live memory decreases because the generational garbage collector begins compacting the older generations. This slower collection process results in dramatic performance degradation (not shown). By contrast, the memory use of ECperf increases up to an Orders Injection Rate of approximately 6, then remains roughly constant through 40. This result suggests that using SPECjbb could lead memory system designers to overestimate the memory footprints of middleware applications on larger systems.

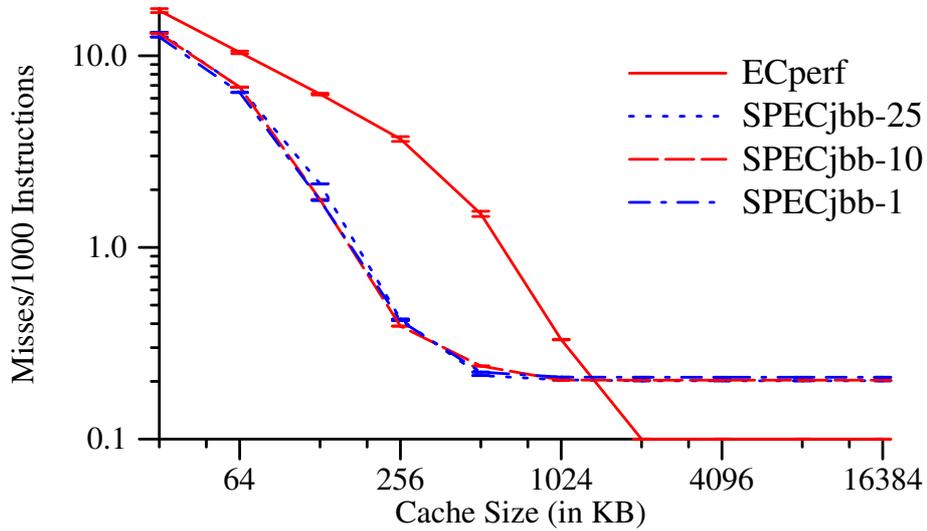


Figure 22.12: Instruction Cache Miss Rate

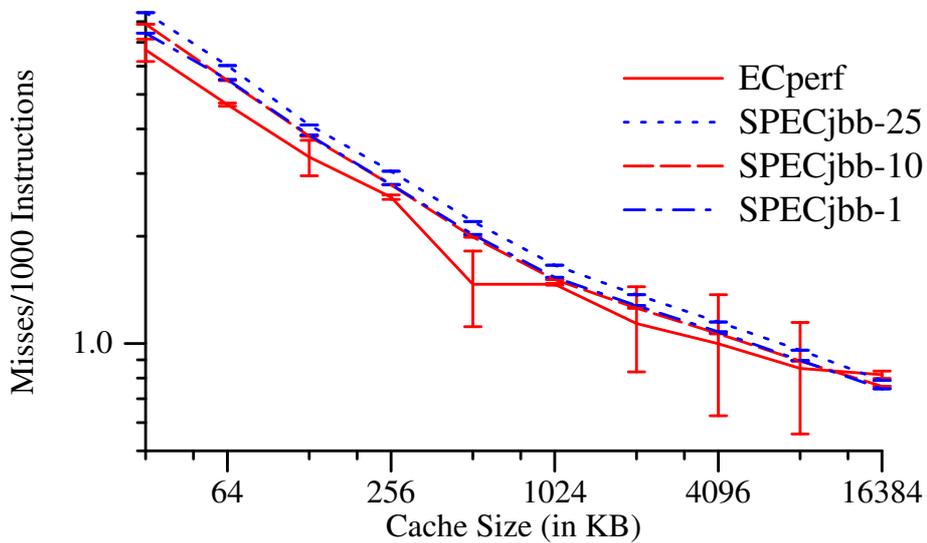


Figure 22.13: Data Cache Miss Rate

22.5 Cache Performance

The previous section showed that memory system stalls were a significant detriment to scalability on the Sun E6000. To understand this behavior more deeply, we used full-system simulation to evaluate a variety of memory system configurations. Our simulation results show that whereas the scaling properties of these workloads are similar, the cache behavior of ECperf is quite different from that of SPECjbb. ECperf has a small data set and a low data-cache miss rate. SPECjbb puts significantly more pressure on the data cache, particularly when it is configured with a large number of warehouses. Although ECperf has a smaller data cache miss rate than even the smallest configuration of SPECjbb, a higher fraction of its total memory is shared between threads. Wider sharing and a smaller data working set make shared-caches a more effective design for that workload.

22.5.1 Cache Miss Rates

Figure 22.12 and Figure 22.13 present the instruction and data cache miss rates, respectively, for a uniprocessor system with a range of cache sizes. All configurations assume split instruction and data caches, 4-way set associativity and 64-byte blocks. We simulated SPECjbb with three different scaling factors (1, 10, and 25 warehouses) to examine the impact of the larger memory sizes discussed in Section 22.4.6. These graphs demonstrate that both benchmarks place at most moderate demand on typical level one (L1) and level two (L2) caches. Typical L1 caches, falling in the 16 KB-64 KB range, exhibit miss rates of 10-40 misses per 1000 instructions. For typical L2 cache sizes of 1 MB and larger, the data miss rate falls to less than two misses per 1000 instructions. Instruction misses are even lower, falling well below one miss per 1000 instructions. The two benchmarks behave similarly, but do have two notable differences. First, ECperf has a much higher instruction cache miss rate for intermediate size caches (e.g., 256 KB). Second, the data miss rate for SPECjbb with one warehouse is roughly comparable to that for ECperf, but it increases by as much as 30% as the data set scales to 25 warehouses. This result is not surprising, given that SPECjbb's live data increases linearly with the number of warehouses (see Figure 22.11).

22.5.2 Communication Footprint

To provide insight into the communication behavior of the workloads, we measured the footprint of the data causing cache-to-cache transfers. As shown in Figure 22.14, all of the cache-to-cache transfers observed

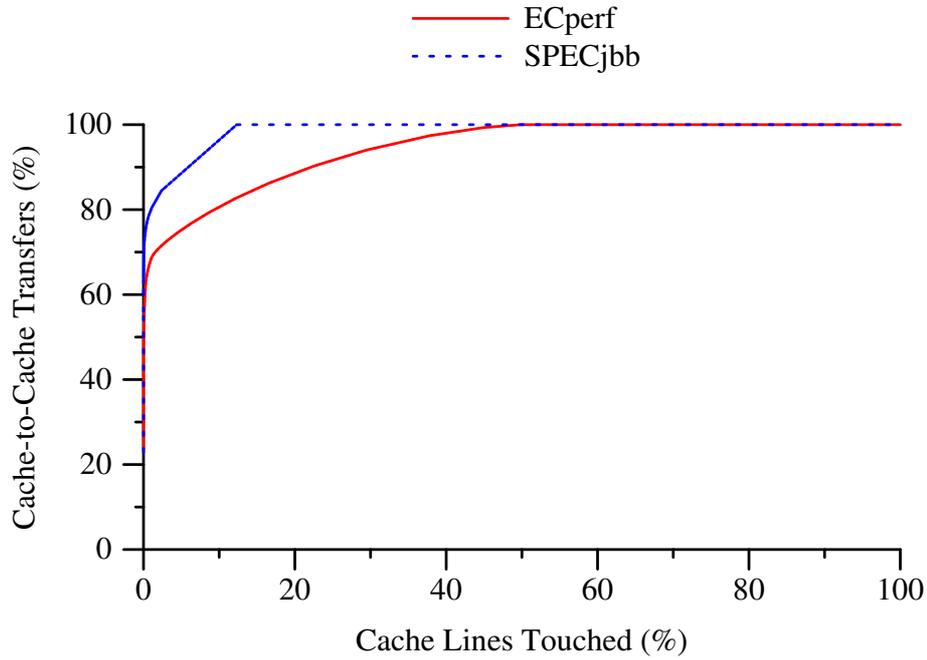


Figure 22.14: Distribution of Cache-to-Cache Transfers (64 Byte Cache Lines)

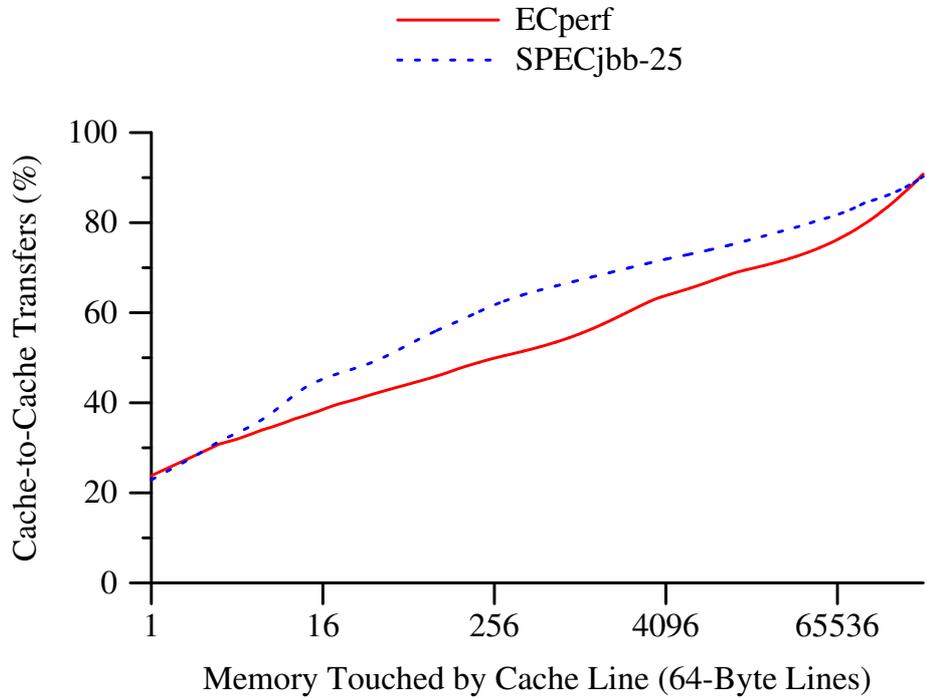


Figure 22.15: Distribution of Cache-to-Cache Transfers

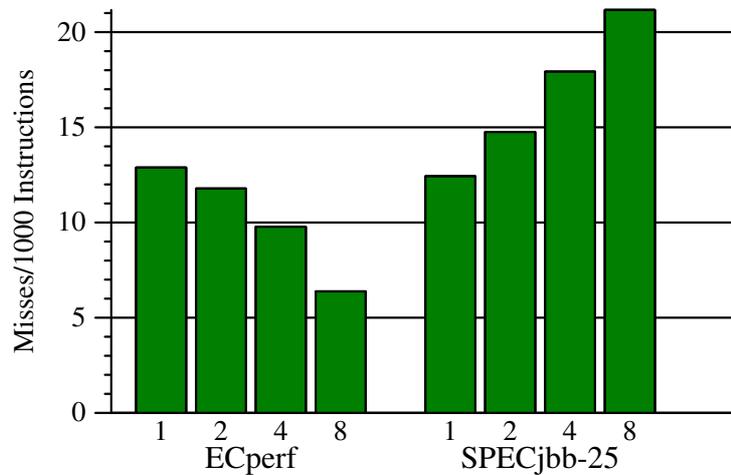


Figure 22.16: Cache Miss Rate on Shared Caches (Processors Per Shared 1 MB Cache)

in SPECjbb came from 12% of the cache lines touched during the measurement period, and over 70% came from the most active 0.1% of cache lines. For both benchmarks, a significant fraction of the communication is likely due to a few highly contended locks. The single cache line with the highest fraction of the cache-to-cache transfers accounted for 20% of the total for SPECjbb and 14% for ECperf. This resembles earlier findings for databases and OLTP workloads [GSSD00]. In contrast to SPECjbb, however, the most active 0.1% of cache lines in ECperf account for only 56% of the cache-to-cache transfers. Furthermore, the cache-to-cache transfers are spread over half of the touched cache lines. A major contributor to this difference between ECperf and SPECjbb is SPECjbb's emulated database. The object trees that represent the database are updated sparsely enough that they rarely result in cache-to-cache transfers. Figure 22.15 shows the cumulative distribution of cache-to-cache transfers versus the amount of data transferred (on a semi-log plot). This graph shows that even though SPECjbb has a larger total data set, ECperf has a larger communication footprint on an absolute, not just a percentage, basis.

22.5.3 Shared Caches

The high cache-to-cache transfer rates of these workloads suggest that they might benefit from a shared-cache memory system, which have become increasingly common with the emergence of chip multiprocessors (CMPs) [ONH⁺96]. Shared caches have two benefits. First, they eliminate coherence misses between the processors sharing the same

cache (private L1 caches will still cause coherence misses, but these can be satisfied on-chip much faster than conventional off-chip coherence misses). Second, compulsory misses may be reduced through inter-processor prefetching (i.e., constructive interference). The obvious disadvantage of shared caches is the potential increase in conflict and capacity misses. To evaluate shared caches, we used Simics to model an 8-processor SPARC V9 system with four different memory hierarchies. In the base case, each processor has a private 1 MB L2 cache, for a total of 8 caches. In the other three cases, the eight processors share one, two, and four 1 MB caches. The total size of all caches is the product of the cache size (i.e., 1 MB) and the number of caches. Figure 22.16 shows the data miss rates for ECperf and SPECjbb as the number of processors per cache increases. For ECperf, the benefit of reducing coherence misses more than makes up for the additional capacity and conflict misses. ECperf has the lowest data miss rate when all eight processors share a single cache, even though the aggregate cache size is 1/8 the size in the base case (i.e., private caches). Sharing had the opposite effect on SPECjbb. Even though SPECjbb had a significant fraction of cache-to-cache transfers, the larger data set size (due to the emulated database) results in an increase in overall miss rate for 1 MB shared L2 caches.

22.6 Related Work

Work This paper extends previous work by examining examples of an important emerging class of commercial applications, Java-based middleware. Cain, et al. describe the behavior of a Java Servlet implementation of TPC-W, which models an online bookstore [CRML01]. Though the Servlets in their implementation are also Java-based middleware, that workload is also quite different than ECperf, since it does not maintain session information for client connections in the middle tier. The Servlets share a pool of database connections in that implementation like the application server in ECperf. However, no application data is exchanged between the Servlets. Previous papers have presented the behavior of commercial applications. Among the most notable are those that describe the behavior of Database Management Systems (DBMS) running the TPC benchmarks, TPC-C and TPC-H [ADHW99][BGB98]. Ailamaki, et al. report that DBMS's spend much of their time handling level-1 instruction and level-2 data misses [ADHW99]. Barroso, et al. report that the memory system is a major factor in the performance of DBMS workloads, and that OLTP workloads are particularly sensitive to cache-to-cache transfer latency, especially in the presence of large second level caches [BGB98]. These studies demonstrate that the execution time of

DBMS is closely tied to the performance of the memory system. Other studies have also examined Java workloads. Luo and John present a characterization of VolanoMark and SPECjbb2000 [LJ01]. VolanoMark behaves quite differently than ECperf or SPECjbb because of the high number of threads it creates. In VolanoMark, the server creates a new thread for each client connection. The application server that we have used, in contrast, shares threads between client connections. As a result, the middle tier of the ECperf benchmark spends much less time in the kernel than VolanoMark. SPECjbb also has a much lower kernel component than VolanoMark. Marden, et al. compare the memory system behavior of a PERL CGI script and a Java Servlet [MLLL02]. Chow, et al. measure uniprocessor performance characteristics on transactions from the ECperf benchmark [CBJC02]. They present correlations between both the mix of transaction types and system configuration to processor performance. Shuf, et al. measure the cache performance of java benchmarks, including pBOB (now SPECjbb). They find that even fairly large L2 caches do not significantly improve memory system performance [SSGS01]. Their measurements, however, are limited to direct-mapped caches and uniprocessors, while we consider multiprocessors with 4-way set-associative caches. They also find that TLB misses are a major performance issue. Although we did not specifically measure TLB miss rates, we found that using the intimate shared memory (ISM) feature of Solaris, which increases the page size from 8 KB to 4 MB, increased performance of ECperf by more than 10%. Barroso, et al. [BGB98] and Olukotun, et al. [ONH⁺96] discuss the performance benefits of chip multiprocessors using shared caches. We extend their work by evaluating the impact of shared caches on SPECjbb and ECperf. 7.

22.7 Conclusions

In this paper, we have presented a detailed characterization of two popular Java-based middleware benchmarks. ECperf is a complex, multi-tier benchmark that requires multiple machines, a commercial database system, and a commercial application server. In contrast, SPECjbb is a single application that is trivial to install and run. The distributed nature of ECperf makes the installation and management of that benchmark more difficult, but it also provides an opportunity to isolate the behavior of each tier individually. We find that both workloads have low CPIs and low memory stall times compared to other important commercial server applications (e.g., OLTP). Running on the moderate size multiprocessors in our study, both workloads maintained small data working sets that fit well in the 1 MB second-level caches of our UltraSPARC II processors. More than half of all second-level cache misses on our larger systems hit

in the cache of another processor. SPECjbb closely approximates the memory behavior of ECperf except for two important differences. First, the instruction working set of SPECjbb is much smaller than that of ECperf. Second, the data memory footprint of SPECjbb is larger than that of ECperf, especially as the benchmark scales up for larger system sizes. The difference in behavior could lead memory system designers toward different conclusions. For example, our simulation results demonstrate that ECperf is particularly well suited to a shared-cache memory system even when the total cache size is limited to 1 MB. In contrast, the reduction in total cache capacity causes SPECjbb's performance to degrade. This study compares two middleware benchmarks, running on a specific combination of hardware, operating system, Java virtual machine, application server, and database system. Further study is needed to determine how well these results apply to other Java middleware and different versions of the underlying hardware and software. However, we believe that as middleware becomes better understood, it will prove increasingly important to isolate its behavior from the effects of other software layers.

Acknowledgments

We would like to thank Paul Caprioli and Michael Koster for introducing us to the ECperf benchmark. We also sincerely thank Jim Nilsson for providing us with the Sumo cache simulator, Akara Sucharitakul for his help configuring and tuning ECperf, as well as Alaa Alameldeen, Dan Sorin and Alvy Lebeck for their insightful comments.

Bibliography

- [ADHW99] Anastassia G. Ailamaki, David J. DeWitt, Mark D. Hill, and David A. Wood. Dbmss on a modern processor: Where does time go? In *In Proceedings of the 25th International Conference on Very Large Data Bases*, pages 266–277, 1999.
- [AW03] A. R. Alameldeen and D. A. Wood. Variability in Architectural Simulations of Multi-Threaded Workloads. In *Proceedings of the Ninth International Symposium on High Performance Computer Architecture (HPCA-9)*, Anaheim, California, USA, February 2003.
- [BGB98] Luiz A. Barroso, Kouros Gharachorloo, and Edouard Bugnion. Memory system characterization of commercial workloads. In *In Proceedings of the 25th Annual Interna-*

- tional Symposium on Computer Architecture*, pages 3–14, 1998.
- [CBJC02] Kingsum Chow, Manesh Bhat, Ashish Jha, and Colin Cunningham. Characterization of java application server workloads. In *IEEE 4th Annual Workshop on Workload Characterization in conjunction with MICRO-34*, pages 175–181, 2002.
- [CRML01] Harold W. Cain, Ravi Rajwar, Morris Marden, and Mikko H. Lipasti. An architectural evaluation of java tpc-w. In *Proceedings of the Seventh IEEE Symposium on High-Performance Computer Architecture*, pages 229–240, 2001.
- [GSSD00] Kouros Gharachorloo, Madhu Sharma, Simon Steely, and Stephen Von Doren. Architecture and design of alphaserver gs320. In *Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 13–24, 2000.
- [KMHW02] Martin Karlsson, Kevin E. Moore, Erik Hagersten, and David A. Wood. Memory characterization of the ecperfbenchmark. In *Second Annual Workshop on Memory Performance Issues (WMPI), in conjunction with ISCA-29*, 2002.
- [LJ01] Yue Luo and Lizy Kurian John. Workload characterization of multithreaded java servers. In *IEEE International Symposium on Performance Analysis of Systems and Software*, 2001.
- [MCJE+02] ”P. S. Magnusson, M. Christensson, D. Forsgren J. Eskilson, G. Hillberg, J. Hgberg, A. Moestedt F. Larsson, and B. Werner”. ”simics: A full system simulation platform”. *IEEE Computer*, February 2002.
- [MLLL02] Morris Marden, Shih-Lien Lu, Konrad Lai, and Mikko Lipasti. Memory system behavior in java and non-java commercial workloads. In *Proceedings of the Fifth Workshop on Computer Architecture Evaluation Using Commercial Workloads*, 2002.
- [ONH+96] Kunle Olukotun, Basem A. Nayfeh, L. Hammond, K. Wilson, and K.-Y. Chang. The case for a single-chip multiprocessor. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, 1996.

- [Pag] ECperf Home Page. Ecpref.
<http://java.sun.com/j2ee/ecperf/>.
- [SPE] SPEC. Spec jappserver development page.
<http://www.spec.org/osg/jAppServer/>.
- [SSGS01] Yefim Shuf, Mauricio J. Serrano, Manish Gupta, and Jaswinder Pal Singh. Characterizing the memory behavior of java workloads: A structured view and opportunities for optimizations. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 2001.

Part V

Real-Time Scheduling

Chapter 23

Introduction

By **Jan Jonsson**

Department of Computer Science and Engineering
Chalmers University of Technology
Email: `janjo@ce.chalmers.se`

Compared to the other sections in this book, which to a large extent deal with application level design issues, this chapter will focus on the basic algorithms used for the planning and scheduling of real-time applications on single and multiprocessor systems.

We start by presenting an overview of the underlying principles used for the execution of real-time processes. This not only encompasses a taxonomy for *when* (on-line vs off-line) and *how* (time table vs priority based) resource conflicts are handled; it also includes an account of what constraints (timing, preemption, migration, etc) are imposed on the processes during their execution.

We then present some fundamental techniques for generating timetables based on the application constraints and a system-wide optimization criterion. This includes a brief description of the search techniques branch-and-bound and simulated annealing.

Finally, we describe different techniques for priority-based scheduling. To that end, we present the well-established rate-monotonic, deadline-monotonic, and earliest-deadline-first priority-assignment methods. In conjunction with this, we also make an *a priori* performance assessment of the priority-based scheduling approaches.

Following the introduction, there will be a collection of papers from the ARTES initiative that can be considered to focus primarily on the problem areas described above.

23.1 Scheduling preliminaries

The main purpose of *scheduling* (or planning) is to assign time slots of the available computation resources — such as processors or other shared hardware/software objects — to the executable entities (tasks) in such a way that the constraints imposed on the system are met.

The resulting set of task-to-resource time slots is referred to as a *schedule*. If the schedule is generated before the tasks are available for execution, we refer to it as *off-line* scheduling. Such scheduling typically requires that complete knowledge about the task characteristics is available, for example that the tasks arrive periodically and execute with a known (worst-case) execution time. In contrast, if the schedule is generated while the tasks are executing we refer to it as *on-line* scheduling.

One class of constraints that regulate how the tasks will be scheduled include timing constraints (deadlines, periodicity), precedence constraints (enforced task ordering) and exclusion constraints (for access to shared resources). These constraints have in common that they are independent of the underlying execution model. Examples of constraints that regulate the actual execution of the tasks on the system hardware are preemption constraints (whether tasks are allowed to interrupt each other's execution or not) and migration constraints (whether a task is allowed to use several processors during its execution or not).

Also dictating the resulting schedule is the chosen scheduling objective. Many techniques only strive for meeting timing constraints whereas fewer techniques attempt to optimize one or more local or global performance measures (for example, task lateness, power consumption, reliability).

23.2 Time-table-based scheduling

For off-line scheduling, the mechanism used is the *time table*, which is a list of start-stop times for each task execution on a certain resource. For a system consisting of periodic tasks the list of start-stop times repeats itself throughout the life-time of the system. Since the time table will be completely known before the system is taken into mission, verifying whether all system constraints are met or not is a fairly easy task.

The easiest way to generate a time table is to use a priority-based algorithm and emulate the execution of the tasks so as to get start and stop times of each task's execution. This technique works quite well for systems with simpler constraints and can then generate schedules of quite good quality, in many cases the optimal schedule. For systems with more complex constraints or performance optimizations, an intelligent search algorithm must be used to find a schedule of the requested quality.

Such algorithms cannot only find optimal schedules whenever they exist, but can also be used for solving multi-objective search problems.

One of the most well-known search algorithm for schedule generation is the branch-and-bound approach (also known as A*). In this technique a search tree representing all possible partial or complete schedules is used, and the algorithm traverses that tree in search for an acceptable schedule. In order to not explore more solutions than necessary, the branches of the search tree is generated dynamically as the search progresses. Whenever the search algorithm has the possibility to explore a new branch in the tree, it tentatively evaluates the quality of that branch using clever prediction methods. If the quality of the branch is provably of poor quality or inferior to other branches already explored this new branch is dismissed and never explored in detail. On the other hand, if the new branch looks promising it is further explored and its true quality could be established. Several intelligent applications of the branch-and-bound approach to real-time scheduling has been proposed, for example by Shin, Ramamritham, Stankovic and others.

Another interesting approach to the off-line generation of schedules is the technique based on simulated annealing, a model for describing how molten metal forms into crystalline structures as it cools down. The adaption to real-time systems was originally elaborated by Burns et al and has since been used in several research efforts. In this approach, all possible solutions to the scheduling problem is represented as points in a search space. The difference between points in the close vicinity of each other is that a particular task is assigned to two different processors. Hence, moving from one point to a neighbor point means changing the assignment of a task. The quality of a point in search space is represented by its energy, where a lower energy means a better solution. On the case of application to the real-time problem domain the energy represents a weighted sum of different penalties that is incurred to the schedule represented by the point. The search for a good schedule is not completely deterministic like for the branch-and-bound technique, but relies to some extent on random jumps in search space. In all natural course of things, the algorithm jumps from one point in search space to another if the new point has a lower energy. However, with a certain likelihood, a neighbor point with a higher energy will be selected, thus allowing for jumps out of local minima in the search space. The probability of making such jumps is a function of the system temperature, a variable that represents the level of freedom in the search. The higher the temperature, the higher the likelihood of making jumps to points with higher energy. As the search progresses the temperature slowly decreases, thus making random jumps less and less likely until the system finally freezes with a final schedule.

23.3 Priority-based scheduling

A common execution mechanism for on-line scheduling is the use of *task priorities* — static or dynamic — to resolve conflicts whenever there are multiple tasks contending for the same resource. Since the actual task schedule will not be known until tasks have been executed, determining whether an on-line scheduling mechanism will meet the imposed constraints must include predicting the schedulability of the tasks. Depending on the actual mechanisms used for scheduling the tasks, this can be a simple problem to solve or it can be a problem that require significant effort.

When the priority-assignment itself is done off-line the priorities are typically derived once and then kept static for the rest of the time. Two well-known heuristics for assigning static priorities are the rate-monotonic and deadline-monotonic approaches. With the rate-monotonic heuristic, tasks with shorter periods are assigned higher priorities. When the deadline of a task is equal to its period, this heuristic has been shown to be optimal among all static-priority heuristics on a single-processor system [Liu and Layland]. As for performance prediction of the rate-monotonic heuristic, it has been shown that tasks will be guaranteed to meet their deadlines if the total requested processing capacity from n periodic tasks — and where the deadline of a task is equal to its period — does not exceed $n(2^{1/n} - 1)$ (the expression converges towards 69.3% as n becomes large). With the deadline-monotonic heuristic, it is no longer necessary to assume that the deadline of a task is equal to its period. For this case, the deadline-monotonic heuristics has been shown to be optimal among all static-priority heuristics on a single-processor system [Leung and Whitehead]. To predict the schedulability of the deadline-monotonic approach, a refined technique referred to as response-time analysis is used. In that technique, the worst-case completion time for each task is calculated and compared against the deadline of the task. This worst-case completion time will account for the execution interferences as caused by tasks in the system that have higher priority than the task being analysed.

If the task priorities can be derived on-line, the system's flexibility increases which is manifested in better performance in terms of schedulability. The most well-known dynamic-priority heuristic is the earliest-deadline-first approach where, at each point in time, the task with the closest (in time) deadline will receive the highest priority. As shown by Liu and Layland, this heuristic has been shown to be optimal among all dynamic-priority heuristics on a single-processor system when the deadline of a task is equal to its period. By the same people, it was also shown for the earliest-deadline heuristic that tasks will always be guaranteed to meet their deadlines as long the total requested processing capacity

from a set of periodic tasks does not exceed 100% on a single-processor system. For the case of single-processor systems where the task deadline is allowed to be shorter than its period, it is not possible to use performance prediction using processing capacity bounds. Instead, the more detailed (and elaborate) technique, called processor-demand analysis, is used to determine whether a set of periodic tasks will meet their deadlines or not. In this technique, a collection of time intervals are analysed with respect to the amount of processing capacity each task requests in each interval. Only the processing capacity of tasks with deadlines that fall within the interval is counted. If the accumulated processing capacities in each interval does not exceed the length of that interval all deadlines of the tasks will be met.

For multiprocessor systems that allow task migration, the priority-assignment techniques mentioned above cannot be used in an efficient way since there are no good methods for predicting their actual performance (in fact, Dhall has shown that no non-zero processing capacity bound exists for rate-monotonic multiprocessor scheduling). In order to achieve efficient resource usage on a multiprocessor system with migrative scheduling, a scheduling approach based on proportionate fairness (or p-fair scheduling for short) must be used. The p-fair scheduling technique achieves a very flexible and efficient technique that combines a refined deadline-based dynamic-priority mechanism with a concept of fairness among all tasks.

23.4 Included papers

This chapter contains four interesting research efforts from the ARTES initiative. Below is a short account of each of these efforts.

The paper "Average-Case Performance of Static-Priority Scheduling on Multiprocessors" by Björn Andersson deals with the issue of whether to partition processes in advance or allow full migration capabilities (aka global scheduling) in multiprocessor real-time systems. Here, simulation-based evaluations are used to show two important things. First, an average-case comparison using an idealized architecture showed that, if a system has a small number of processors, then global scheduling offers higher performance than partitioned scheduling. Second, an average-case comparison using a realistic architecture, showed that, for several combinations of preemption and migration costs, global scheduling offers higher performance.

The paper "Schedulability-Driven Communication Synthesis for Time-Triggered Embedded Systems" by Paul Pop *et al* deals with the issue of how to generate efficient time-tables for data- and control-dependent real-time applications that use interprocess communication.

This work presents an analysis for the network communication delays with four different message scheduling policies over a time-triggered communication channel. Optimization strategies for the synthesis of communication are developed, and the four approaches to message scheduling are compared using extensive experiments.

The paper "Generating Real-Time Schedules using Constraint Programming" by Cecilia Ekelin deals with the problem of generating efficient time-tables for real-time applications with multiple optimization criteria. This work demonstrates how to devise a scheduling algorithm based on *constraint programming* that facilitates both accurate system modeling and easy implementation without sacrificing either speed or quality. Constraint programming is a high-level software layer to the branch-and-bound search algorithm and allows the user to describe/model the system constraints in a mathematical fashion and use a powerful constraint propagation algorithm to reduce the complexity of the original problem. As demonstrated in this work, the constraint programming approach is able to produce good (near-optimal) schedules and in reasonable time when compared with other state-of-the-art algorithms. Furthermore, the underlying framework reduces the implementation efforts since constraints are modeled independently and previous knowledge on constraint propagation can be reused.

The last paper, "Utilization bounds of static-priority scheduling on multiprocessors" by Björn Andersson and Jan Jonsson, deals with the problem of finding good priority-assignment schemes for multiprocessor real-time systems with full migration capabilities. In this work, it is shown that every static-priority algorithm can miss deadlines on a multiprocessor although only close to 50% of the capacity is requested. The new algorithms presented in this work have been shown to have the following performance characteristics for the scheduling of periodic task: the processing capacity that can be requested *a priori* without missing a deadline cannot exceed 33% for migrative scheduling and 50% for non-migrative scheduling. In aperiodic scheduling, many performance metrics have been used in previous research. With the aperiodic model used in this work, the new algorithms in this work have been shown to possess the following performance characteristics: the processing capacity that can be requested without missing a deadline cannot exceed 50% for migrative scheduling and 31% for non-migrative scheduling.

Chapter 24

Average-case performance of static-priority scheduling on multiprocessors

By **Björn Andersson** and **Jan Jonsson**
Department of Computer Science and Engineering
Chalmers University of Technology
Email: janjo@ce.chalmers.se

This chapter deals with the problem of scheduling a set of tasks to meet deadlines on a computer with multiple processors. Static-priority scheduling is considered, that is, a task is assigned a priority number that never changes and at every moment the highest-priority tasks that request to be executed are selected for execution.

Many contemporary computers support static-priority scheduling using two different approaches: with task migration or without task migration. This chapter evaluates the performance of these approaches, using simulation of randomly-generated workloads on a range of different highly-abstracted architectural setups.

24.1 Introduction

Static-priority scheduling of tasks on shared-memory multiprocessors is typically solved using one of two different approaches based on how tasks are assigned to the processors at run-time. In *partitioned scheduling*, all instances of a task are executed on the same processor. The processor used for the execution of a task is determined before run-time by a partitioning algorithm. In *global scheduling*, a task is allowed to execute on any processor, even when resuming after having been preempted. Many operating systems for shared-memory multiprocessors support static-

priority scheduling with both the partitioned and global scheduling but it is not clear which one offers the best performance.

It is tempting to reason like this. For every schedule generated by a partitioning algorithm, there is a global scheduling algorithm that generates the same schedule, and hence global scheduling is no worse than partitioned scheduling. This reasoning is, however, false in the context of static-priority scheduling [LW82], hence making it non-obvious which approach offers the best performance. In fact, before 1999 when the research of our Artes project was performed, it was often claimed that in the context of static-priority scheduling, global scheduling using rate-monotonic performs poorly. These claims were supported by simulation experiments [LMM98a] or referring to an observation [DL78] where it was shown that global rate-monotonic could miss deadlines although close to 0% of the processing capacity is requested.

In this chapter, we compare the performance of global scheduling against partitioned scheduling, where global scheduling uses another priority-assignment scheme than rate-monotonic. We demonstrate that partitioned scheduling is *not necessarily the best approach*. To this end, we make two main research contributions.

- C1. We show that, on an idealized architecture, global scheduling offers higher average-case performance than partitioned scheduling. We also show that, even if a necessary and sufficient schedulability test is used for partitioning scheduling, then the best partitioned scheduling algorithm performs only slightly better than the global scheduling algorithm.

- C2. We show that, on a realistic architecture, global scheduling can provide higher performance than partitioned scheduling when both an improved dispatcher is used and the additional cost of a migration is no greater than the cost of a preemption.

The rest of this chapter is organized as follows. In Section 24.2, we review previous work in static-priority preemptive multiprocessor scheduling and define concepts and system models used. Section 24.3 shows their average-case behavior. In Section 24.4, we discuss and show the architectural impact on the average-case performance. We discuss other aspects of the scheduling problem in Section 24.5, and summarize our results in Section 24.6.

24.2 Background

Recall that there are two approaches to solve the multiprocessor scheduling problem, namely partitioned scheduling and global scheduling¹. In this section, we first discuss in more detail the capabilities and problems of these methods, and then define concepts and system models used in this chapter.

24.2.1 Previous work

Multiprocessor real-time scheduling differs from uniprocessor real-time scheduling in that we need to determine not only *when* a task should execute, but also on *which* processor to execute. In preemptive uniprocessor real-time scheduling, a task can be preempted by another task, and resume its execution later at a different time on the same processor. In preemptive multiprocessor real-time scheduling, a task that is preempted by another task still needs to resume its execution later at a different time, but the task may resume its execution on a different processor. Based on how the system resumes a task's execution, two fundamentally different methods can be used to implement preemptive multiprocessor scheduling, namely, partitioned and global scheduling.

Using partitioned scheduling, the tasks in the task set are divided in such a way that a task can only execute on its assigned processor. In this case, each processor has its own ready queue and tasks are not allowed to migrate between processors. Using global scheduling, all tasks reside in a global ready queue and can be dispatched to any processor. After being preempted, the task can resume its execution on any processor. The principles for partitioned scheduling and global scheduling are illustrated in Figure 24.1.

For static-priority preemptive scheduling of periodically-arriving tasks on a multiprocessor system, both partitioned scheduling and global scheduling have been addressed in previous research. Important properties were presented in a seminal paper by Leung and Whitehead [LW82]. In particular, they showed that the problem of deciding whether a task set is schedulable (that is, all tasks will meet their deadlines at run-time) is NP-hard for both global scheduling and partitioned scheduling. They also observed that no approach dominates the other in the sense that there are task sets which are schedulable with an optimal priority assignment with global scheduling, but are unschedulable with an optimal partitioning algorithm and conversely.

Among the two methods, partitioned scheduling has received the most attention in the research literature. The main reason for this is

¹Some authors refer to global scheduling as “the non-partitioned method” or “dynamic binding”.

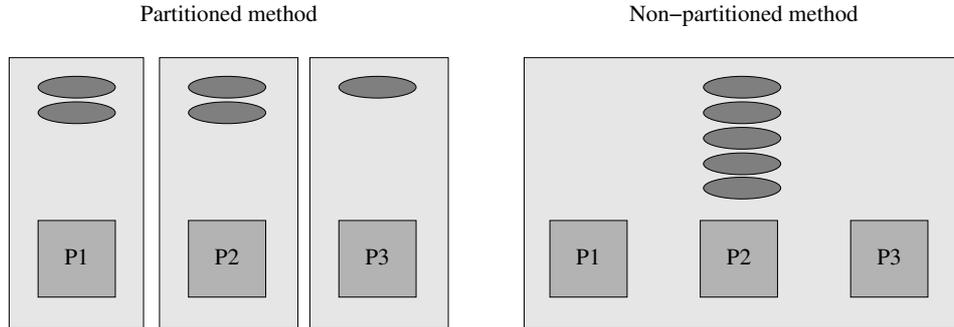


Figure 24.1: Using global scheduling a task can only execute on its assigned processor. Using global scheduling, a task can execute on any processor. Squares represent processors. Ellipses represent tasks.

that partitioned scheduling can easily be used to guarantee run-time performance (in terms of schedulability). By using a uniprocessor schedulability test as the admission condition when adding a new task to a processor, all tasks will meet their deadlines at run-time. Now, recall that partitioned scheduling requires that the task set has been divided into partitions, each having its own dedicated processor. Since an optimal solution to the problem of partitioning the tasks is believed to be computationally intractable, many heuristics for partitioning have been proposed, a majority of which are versions of the bin-packing algorithm² [Dha77, DL78, DD85, DD86, BLOS95, OS95a, OS95b, LMM98c, SVC98]. All of these bin-packing-based partitioning algorithms provide performance guarantees, they all exhibit fairly good average-case performance, and they can all be applied in polynomial time (using sufficient schedulability tests).

Global scheduling has received considerably less attention, mainly because of the following limitations. First, no efficient schedulability tests currently exist for global scheduling. Recall that, for partitioned scheduling, existing uniprocessor schedulability tests can be used. The only known necessary and sufficient schedulability test for global scheduling has an exponential time-complexity [Leu89]. The complexity can be reduced with sufficient schedulability tests to a polynomial [LMM98b, Lun98, Liu69] or pseudo-polynomial [Lun98] time

²The bin-packing algorithm works as follows: (1) sort the tasks according to some criterion; (2) select the first task and an arbitrary processor; (3) attempt to assign the selected task to the selected processor by applying a schedulability test for the processor; (4) if the schedulability test fails, select the next available processor; if it succeeds, select the next task; (5) goto step 3.

complexity. However, the test in [Liu69] becomes pessimistic when the number of tasks increases, and the tests in [LMM98b, Lun98] become pessimistic when the number of processors increases. During the years 1999-2003, several schedulability analysis techniques were developed [AL01, AAJ⁺02, LL03, AS03, Bak03]. Unfortunately, they are only sufficient but not necessary tests. The second reason why global scheduling has received less attention is that no efficient optimal priority-assignment scheme has been found for global scheduling. The rate-monotonic priority assignment (RM) [LL73], which is optimal on a uniprocessor, is not optimal for multiprocessors using global scheduling [Dha77, DL78, LW82]. Even worse, task sets with a very low utilization can be unschedulable with RM [Dha77, DL78]. We refer to the latter as *Dhall's effect*.

In a recent comparison of partitioned scheduling and global scheduling, it was claimed that, even if one is willing to use a necessary and sufficient schedulability test for global scheduling, global rate-monotonic is inferior to the partitioning algorithms [LMM98b, Section 3.5]. We believe that this comparison is not complete for two reasons. First, the state-of-art in global static-priority scheduling has made steady progress during the years 1999-2003. Currently, the algorithm that has achieved the greatest utilization bound is RM-US(0.37) [Lun02] and it meets all deadlines if the utilization is less than 0.37. In this chapter, we will however, only consider the priority-assignment scheme *adaptiveTkC* [AJ00]. We choose adaptiveTkC for two reasons: (i) the author believe (in the year of 2004) that adaptiveTkC offers greater performance than all published global static-priority scheduling algorithms, and (ii) this chapter is based on research that was performed in 1999 (and hence used adaptiveTkC). AdaptiveTkC assigns the highest priority to the task τ_i which has the least $T_i - k * C_i$, where $k = \frac{1}{2} \cdot \frac{m-1+\sqrt{5m^2-6m+1}}{m}$ (m is the number of processors). As shown in [AJ00], the value of k is selected to counter two effects that occur at $k = 0$ and $k \rightarrow \infty$. The case where $k = 0$ is equivalent to RM, which is known to suffer from Dhall's effect. The other case, $k \rightarrow \infty$, gives highest priority to the task with the longest execution time, which can make task sets unschedulable at arbitrary low utilization. The second reason why the previous evaluation is not complete is that it considered the cost of preemption and migration to be zero. It is not clear how preemption and migration costs will affect performance of partitioned scheduling and global scheduling. Because of these shortcomings, we believe that it is necessary to take a new look at the question of whether to partition or not to partition.

24.2.2 Concepts and System model

We consider the problem of scheduling a task set $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$ of n independent³, periodically-arriving real-time tasks on m identical processors. A task arrives periodically with a period of T_i . Each time a task arrives, a new *instance* of the task is created. Each instance has a constant execution time of C_i . Each task has a prescribed deadline, which is the time of the next arrival of the task. The *meta period* of the task set is the least common multiple of T_1, T_2, \dots, T_n .

For partitioned scheduling the system behaves as follows. Each task is assigned to a processor, and then assigned a local (for the processor), unique and fixed priority. With no loss of generality, we assume that the tasks on each processor are numbered in the order of decreasing priority, that is, τ_1 has the highest priority. On each processor, the task with the highest priority of those tasks which has arrived, but not completed, is executed.

For global scheduling, the system behaves as follows. Each task is assigned a global, unique and fixed priority. With no loss of generality, we assume that the tasks in τ are numbered in the order of decreasing priority, that is, τ_1 has the highest priority. Of all tasks that have arrived, but not completed, the m highest-priority tasks are executed⁴ in parallel on the m processors.

The *utilization* u_i of a task τ_i is $u_i = C_i/T_i$, that is, the ratio of the task's execution time to its period. The utilization U of a task set is the sum of the utilizations of the tasks belonging to that task set, that is, $U = \sum_{i=1}^n C_i/T_i$. Since we consider scheduling on a multiprocessor system, the utilization is not always indicative of the load of the system. This is because the original definition of utilization is a property of the task set only, and does not consider the number of processors. To also reflect the amount of processing capability available, we use the concept of *system utilization*, U_s , for a task set on m processors, which is the average utilization of each processor, that is, $U_s = U/m$. A task is *schedulable* if all its instances complete no later than their deadlines. A task set is schedulable if all its tasks are schedulable. *The least system utilization* of a scheduling algorithm is the minimum of the system utilization of unschedulable task sets.

To understand the basic performance characteristics, we initially assume the following simple system model.

- A1. Tasks are independent, arrive periodically, and can always be pre-empted. Hence, at every moment, a dispatcher determines which task to execute. In practice, some operating systems use tick

³That is, there are no precedence constraints on the arrival time of the tasks.

⁴At each instant, the processor chosen for each of the m tasks is arbitrary. If less than m tasks should be executed simultaneously, some processors will be idle.

driven scheduling, where the ready queue is only inspected at certain times.

- A2. Tasks do not require exclusive access to any other resource than a processor.
- A3. The cost of preemption is zero. We will use this assumption even if a task is resumed on another processor than the task was originally preempted on (that is, the cost of migration is also assumed to be zero).
- A4. The cost when a task arrives is zero because we consider cache misses that occur when a task arrives to be included in the execution time. We will use this assumption even if a task executes after arrival on a processor on which it has never executed. These assumptions should be reasonable since worst-case execution time analysis tools typically consider uninterrupted execution of a task that starts in an “empty state” (e.g., all cache lines are empty).

To investigate more realistic characteristics of the system, we will relax A3 later in this chapter (in Section 24.4).

24.3 Average-Case Behavior

In this section, we will conduct an average-case performance evaluation of global scheduling and partitioned scheduling, assuming an idealized architecture with no overhead of task preemption or migration. Our evaluation methodology, which will be described in Section 24.3.1, is based on simulation experiments using randomly-generated task sets. The rationale for using simulation of synthetic task sets is that it more easily reveals the average-case performance and robustness of a scheduling algorithm than can be achieved by scheduling a single application benchmark. Section 24.3.2 presents the results from the simulations.

24.3.1 Experimental setup

Unless otherwise stated, we conduct the performance evaluation using the following experimental setup.

Task sets are randomly generated and their scheduling is simulated with the respective method on $m = 4$ processors. The number of tasks, n , follows a uniform distribution with an expected value of $E[n] = 8$, a minimum of $0.5 E[n]$, and a maximum of $1.5 E[n]$. The period, T_i , of a task τ_i is taken from a set $\{100, 200, 300, 400, 500, \dots, 1600\}$, each number having an equal probability of being selected. The utilization, u_i , of a task τ_i follows a normal distribution with an expected value of

$E[u_i] = 0.5$ and a standard deviation of $stddev[u_i] = 0.4$. If $u_i < 0$ or $u_i > 1$, then a new u_i is generated. The execution time, C_i , of a task τ_i is computed from the generated utilization of the task, and the execution time is rounded down to the next lower integer. If the execution time becomes zero, then the task is generated again.

The priorities of tasks are assigned with the respective priority-assignment scheme, and, if applicable, tasks are partitioned and assigned to processors. All tasks arrive at time 0 and scheduling is simulated during one meta period⁵. Scheduling decisions are only taken when a task arrives or completes.

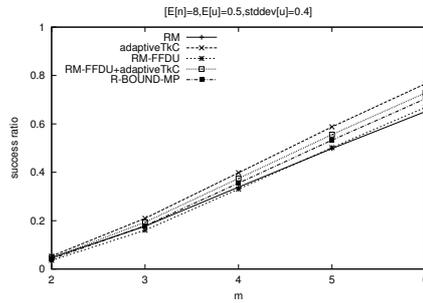
We use *success ratio* as the performance measure for average-case performance. The success ratio is the fraction of all generated task sets that are successfully scheduled with respect to an algorithm. The success ratio is computed for each point in a plot as an average of 2,000,000 task sets⁶. In the evaluation of partitioned scheduling, we consider a task set as successfully scheduled if and only if the number of processors required according to the bin-packing algorithm is no greater than m . In the evaluation of global scheduling, we consider a task set as successfully scheduled if the task set is schedulable, that is all task instances in the task set simulated during a meta period completed no later than their deadlines.

Two global priority-assignment schemes are evaluated, namely RM [LL73] and adaptiveTkC [AJ00]. Two bin-packing-based partitioning algorithms are studied, namely RM-FFDU [OS95b], and R-BOUND-MP [LMM98c]. The reason for selecting these two latter algorithms is that we have found that they are the partitioning algorithms which provide the best performance in our experimental environment (other algorithms, such as RRM-BF [OS95a] and RMGT [BLOS95] offer lower performance). For all partitioning algorithms, we use the corresponding sufficient schedulability test.

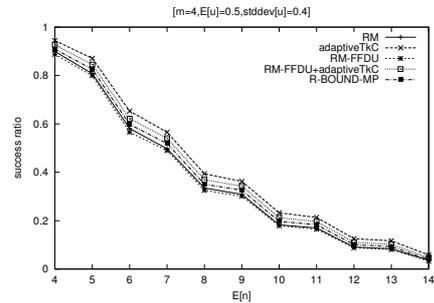
We have also evaluated a hybrid partitioned/global scheduling algorithm, which we will call *RM-FFDU+adaptiveTkC*. The reason for considering a hybrid solution is that we may be able to increase processor utilization with the use of tasks that are not partitioned, without jeopardizing the guarantees given to partitioned tasks. The RM-FFDU+adaptiveTkC scheme operates in the following manner. First, as many tasks as possible are partitioned with RM-FFDU on the given

⁵The reason for scheduling during a meta period is because for global scheduling, the response time of a task is not necessarily maximized when a task arrives at the same time as its higher priority tasks [AJ00]. We therefore select small values of $E[n]$ to avoid that the meta period grows too large, causing simulations to take too long time. We also select small values of m since m must be less than n to make the scheduling problem non-trivial.

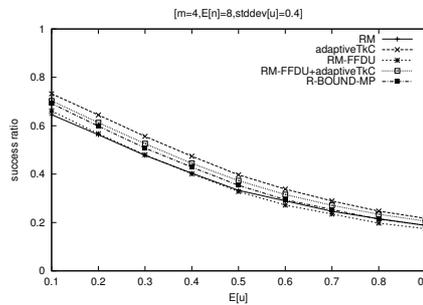
⁶With 95% confidence, we obtain an error of the success ratio that is less than 0.1%.



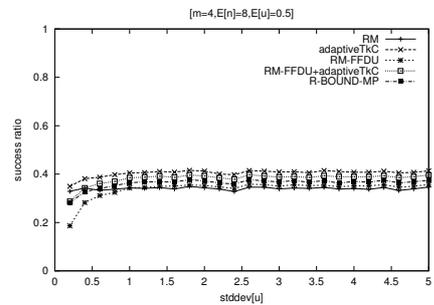
(a) Success ratio as a function of the number of processors.



(b) Success ratio as a function of the number of tasks.



(c) Success ratio as a function of the expected value of utilization of tasks.



(d) Success ratio as a function of the standard deviation of utilization of tasks.

Figure 24.2: Success ratio for different scheduling algorithms when the partitioning algorithms use a sufficient schedulability test (polynomial time complexity).

number of processors, and are given local priorities. Then, the remaining tasks (if any) are assigned global priorities according to the adaptiveTkC priority-assignment scheme. Each processor has a local ready queue for the partitioned tasks and there is a global ready queue for the tasks that are not partitioned. A processor executes a task from its local ready queue, if the local ready queue of that processor is non-empty. A processor executes a task from the global ready queue, if the local ready queue of that processor is empty.

24.3.2 Performance comparison

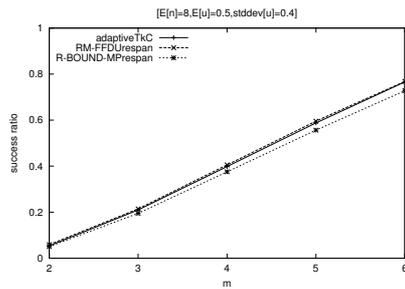
Figure 24.2 shows the results of the simulation experiment. We make three observations.

First, RM offers the worst performance. However, it is not as bad as suggested by previous studies [Dha77, DL78, DD85, DD86]. The reason for this is of course that Dhall's effect, although it exists, does not occur frequently. This observation also corroborates a recent study [LMM98b].

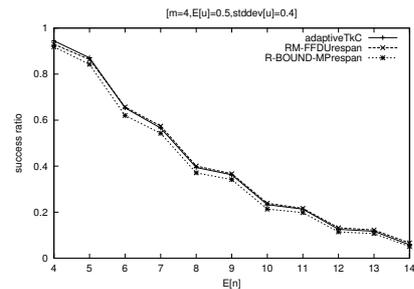
Second, adaptiveTkC offers the best performance, no matter how we vary the parameters (the number of processors, the expected value of the number of tasks, the expected value of task utilization and the standard deviation of the task utilization). The reason for this is that adaptiveTkC takes advantage of the salient property of global scheduling, namely the ability to schedule tasks in slots of unused time on different processors (as RM does).

Third, the hybrid partitioned/global algorithm consistently outperforms the corresponding partitioning algorithms. This indicates that such a hybrid scheme is a viable alternative to use in multiprocessor systems that mixes real-time tasks of different criticality. The reason for the good performance is that the hybrid partitioned/global algorithm can schedule all task sets that the corresponding partitioned scheduling algorithm can schedule, but the hybrid partitioned/global algorithm can also schedule some tasks in slots of unused time on different processors.

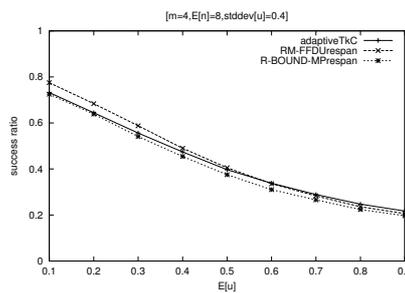
For all partitioning algorithms, sufficient schedulability tests were originally proposed to be used. Now, if we instead use a necessary and sufficient schedulability test based on response-time analysis [JP86] for the partitioning algorithms, the success ratio can be expected to increase, albeit at the expense of a significant increase in computational complexity. Recall that response-time analysis has pseudo-polynomial time complexity, while sufficient schedulability tests typically have polynomial time complexity. Figure 24.3 shows the simulation results when the partitioning algorithms use response-time analysis. Here, we make the following observation. RM-FFDU with response-time analysis only provides a slightly higher success ratio than adaptiveTkC while other partitioning algorithms still have a lower success ratio than adaptiveTkC. Note that this means that, even if the best partitioning algorithm (R-BOUND-MP) use response-time analysis, it still performs worse than adaptiveTkC. This should not come as a surprise, since for R-BOUND-MP, the performance bottleneck is the partitioning and not the schedulability test [LMM98c].



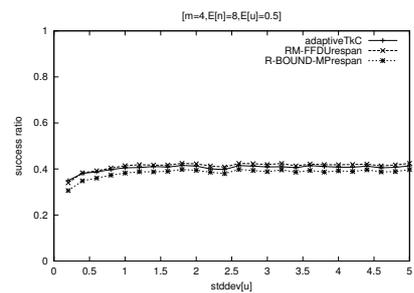
(a) Success ratio as a function of the number of processors.



(b) Success ratio as a function of the number of tasks.



(c) Success ratio as a function of the expected value of utilization of tasks.



(d) Success ratio as a function of the standard deviation of utilization of tasks.

Figure 24.3: Success ratio for different scheduling algorithms when the partitioning algorithms use a necessary and sufficient schedulability test (pseudo-polynomial time complexity).

Algorithm 14 Task-to-processor assignment algorithm for global scheduling.

Input: Let τ_{before} be the set of tasks that just executed on the m processors. On each processor p_i , a (possibly non-existing) task $\tau_{ij,before}$ executed. Let $\tau_{highest}$ be the set of tasks that are selected for execution.

Output: On each processor p_i , a (possibly non-existing) task $\tau_{ij,after}$ should execute.

```

1:  $E = \{p_i : (\tau_{ij,before} \neq \text{non-existing}) \wedge (\tau_{ij,before} \in \tau_{highest})\}$ 
2: for each  $p_i \in E$ 
3:   remove  $\tau_{ij,before}$  from  $\tau_{highest}$ 
4:    $\tau_{ij,after} \leftarrow \tau_{ij,before}$ 
5: for each  $p_i \notin E$ 
6:   if  $\tau_{highest} \neq \emptyset$ 
7:     select an arbitrary  $\tau_j$  from  $\tau_{highest}$ 
8:     remove  $\tau_j$  from  $\tau_{highest}$ 
9:      $\tau_{ij,after} \leftarrow \tau_j$ 
10:  else
11:     $\tau_{ij,after} \leftarrow \text{non-existing}$ 

```

24.4 Architectural Impact

From the results in Section 24.3, we observed that adaptiveTkC performed well under the assumption of an idealized architecture. Now, in order to evaluate the performance of a realistic system we must incorporate some architectural costs. In this section, we will study the impact of two architectural costs, namely *preemption* and *migration*.

It is tempting to believe that global scheduling causes a larger amount of (and more costly) preemptions⁷ than partitioned scheduling, and that this makes more task sets schedulable with partitioned scheduling than with global scheduling. We will now show that such statements cannot easily be made. First, we will propose a dispatcher for global scheduling that reduces the number of preemptions by analyzing the current state of the schedule. We will then show that the average number of preemptions generated by the best global scheduling algorithm using the new dispatcher is significantly less than that of the best partitioning algorithm. Finally, we will evaluate the schedulability of global scheduling and partitioned scheduling when the costs of preemption and migration are accounted for.

24.4.1 Improved Dispatcher

Recall that global static-priority preemptive scheduling only requires that, at each instant, the m tasks with the highest priorities are executed;

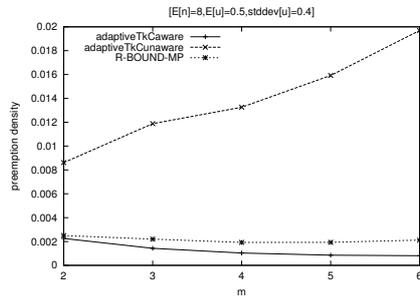
⁷A task is preempted if it has remaining execution time but does not immediately continue to execute on the same processor.

it does not require a task to run on a specific processor. As long as the cost of a preemption is zero, the task-to-processor assignment at each instant does not affect schedulability. However, on real computers the time required for a preemption is non-negligible. If the task-to-processor assignment is selected arbitrarily, it could happen that all m highest-priority tasks execute on another processor than they did the last time they were dispatched, even if these m tasks were the same ones that executed last. With the original dispatcher, this would have serious consequences for the schedulability of the system. Hence, to reduce the risk of unnecessary preemptions, we need a dispatcher that not only selects the m highest-priority tasks, but also selects a task-to-processor assignment such that the number of preemptions is minimized.

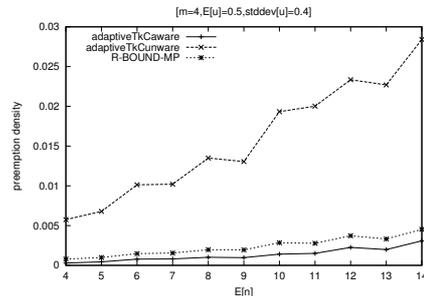
We now propose a heuristic for the task-to-processor assignment that will reduce the number of preemptions for global scheduling. The basic idea of the task-to-processor assignment algorithm is to determine which of the tasks that must execute now (that is, have the highest priority) have recently executed, and then try to execute those tasks on the same processor as in their previous execution. The algorithm for this is described in Algorithm 14. In the remainder of this chapter, we will refer to this dispatcher as the *preemption-aware* dispatcher. We let *adaptiveTkCunaware* denote adaptiveTkC together with a dispatcher where the task with the highest priority is assigned to the processor with the least index, unaware of the preemptions it will generate. Correspondingly, we let *adaptiveTkCaware* denote adaptiveTkC together with the preemption-aware dispatcher. In the remainder of this section, we will assume that, whenever global scheduling is used, adaptiveTkCaware is used. AdaptiveTkCunaware will only be used as a baseline algorithm.

24.4.2 Number of Preemptions

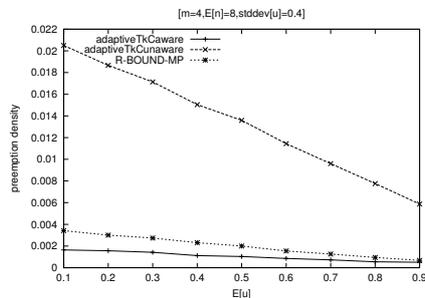
To demonstrate that the preemption-aware dispatcher is effective, we will start by showing its performance in terms of the number of preemptions generated. To this end, it would be natural to simulate scheduling and count the total number of preemptions generated during a meta period and use that number as a measure of the number of preemptions. However, if many task sets are simulated, and we take the sum of the preemptions in all task sets, then task sets with a large meta period would be biased in that they would have a higher impact on the total number of preemptions. To make each task set equally important, we have therefore chosen to use *preemption density* as the measure of the number of preemptions. Preemption density of a scheduled task set is defined as the number of preemptions generated during a meta period divided by the length of the meta period.



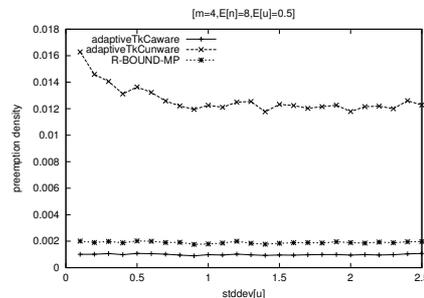
(a) Preemption density as a function of the number of processors.



(b) Preemption density as a function of the number of tasks.



(c) Preemption density as a function of the expected value of utilization of tasks.



(d) Preemption density as a function of the standard deviation of utilization of tasks.

Figure 24.4: Preemption density of different scheduling algorithms.

To reveal which of the two methods (partitioned or global scheduling) that generates the highest number of preemptions, we have simulated scheduling of randomly-generated task sets and computed the preemption density for each of these task sets. Since, in this subsection, we are only interested in the number of preemptions — not their impact on schedulability — we assume that the cost of preemption and the cost of migration is zero. We simulate scheduling using `adaptiveTkCaware`, `adaptiveTkCunaware` and `R-BOUND-MP` because they are the best (as demonstrated in Section 24.3.2) schemes for global scheduling and partitioned scheduling, respectively. We use the same experimental setup as described in Section 24.3.1. We varied the number of tasks and simulated 5,000 task sets for each point in a plot. We then computed the preemption density only for those task sets for which both `adaptiveTkCaware`, `adaptiveTkCunaware` and `R-BOUND-MP` were schedulable.

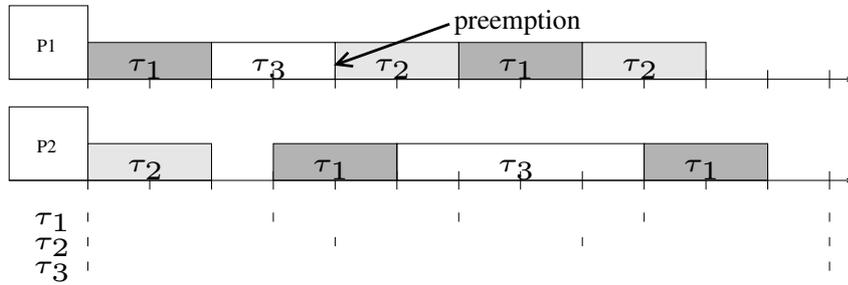
The results from the simulations are shown in Figure 24.4. We observe that, on average, the preemption density of the best global scheduling algorithm (`adaptiveTkCaware`) is lower than the preemption density of the best partitioned scheduling algorithm (`R-BOUND-MP`). The reason for this is that, for partitioned scheduling, an arriving higher-priority task must always preempt an executing lower-priority task on the same processor. With global scheduling, a higher priority task can sometimes execute on another idle processor, thereby avoiding a preemption. Figure 24.5 illustrates a situation where global scheduling with the preemption-aware dispatcher causes the number of preemptions to be less than the number of preemptions of a partitioned scheduling algorithm.

Note that, in Figure 24.4(a), increasing the number of processors gives a different effect on the preemption density for `adaptiveTkCaware` than for `adaptiveTkCunaware`. It can be explained as follows. When the number of processors increases, there will be more tasks executing in parallel and it is more likely that, during a time interval, a task completes its execution. Consider which processor a low-priority task executes on when a higher-priority task completes its execution. For `adaptiveTkCaware`, the lower-priority task will continue to execute on the same processor as it did before the higher priority task completed. Hence no preemption occurs. However, for `adaptiveTkCunaware` (as defined in Section 24.4.1), the lower priority task will continue its execution on another processor with a lower index. That is, with our definition, a preemption.

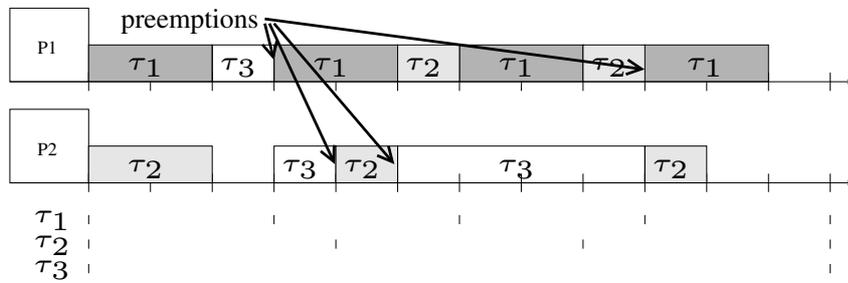
24.4.3 Average-Case Behavior

With the new dispatcher at hand, we are now in position to assess the schedulability of a system with non-negligible preemption and migration cost. We will model the preemption cost and migration cost of a uniform-memory access shared-memory multiprocessor with caches (depicted in Figure 24.6).

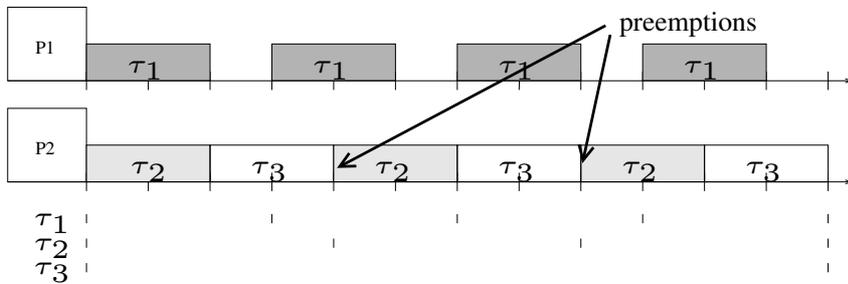
For the assumed system, the cost of a preemption includes the following terms: time to acquire the ready queue, time to save and restore a task's environment (e.g., registers, memory mapping) and the cost of cache misses due to cache reloading when the task resumes. In the evaluation in this section, we will assume that the time to acquire the ready queue and time to save and restore a task's state is zero. We do this for three reasons, namely (i) these costs depend heavily on the implementation (hardware/software) and mechanism to protect the ready queue (fine-grained/course-grained locking/wait-free/lock-free), (ii) it has been found that the time to execute kernel code constitute only a fraction of the total cost of a context switch [PR94], and (iii) the trend of faster



(a) AdaptiveTkCaware



(b) AdaptiveTkCunaware



(c) R-BOUND-MP

Figure 24.5: Preemptions for global scheduling and partitioned scheduling when scheduling the task set $\{(T_1 = 3, C_1 = 2), (T_2 = 4, C_2 = 2), (T_3 = 12, C_3 = 6)\}$ on $m = 2$ processors. Global scheduling, with adaptiveTkCaware, generates the least number of preemptions.

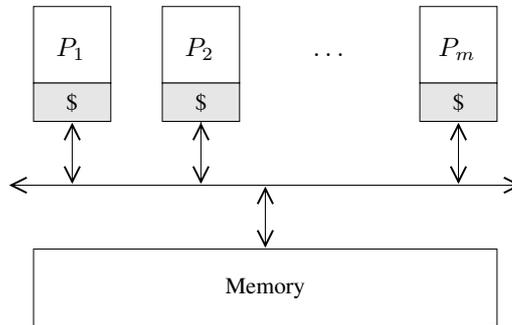


Figure 24.6: A uniform-memory access shared-memory multiprocessor. Each processor P_1, P_2, \dots, P_m shares the memory and the physical address space. Each processor has a level-1 and level-2 cache memory, together represented as a dollar sign.

processors and (relatively) slower memories will make the cache reload effect an even more dominant term in the cost of preemption.

Our simulated system model is now changed as follows. When a task is resumed, we add a cost of $preempt.rat \cdot E[u] \cdot E[T]$ to its remaining execution time because of cache reloading. If a task is resumed on another processor than it was preempted on we add yet another cost of $mig.rat \cdot E[u] \cdot E[T]$ to its remaining execution time because of cache reloading due to the migration. We use the same experimental setup as described in Section 24.3.1, but simulated 5,000 task sets for each possible combination of preemption ratio, migration ratio, and the number of processors.

The results from our simulations are shown in Figure 24.7. We observe that, when the migration and preemption costs are high, adaptiveTkCaware has higher success ratio than adaptiveTkCunaware. This indicates that the preemption-aware dispatcher is effective. We can also see that partitioned scheduling becomes more favorable when the migration ratio is large, because no migrations can occur for that algorithm. On the other hand, global scheduling becomes more favorable when the preemption ratio increases, because the preemption-aware dispatcher can reduce the number of preemptions. However, when there are only two processors available (see Figure 24.7(a)), the preemption-aware dispatcher becomes less effective, because it has only one alternative processor to try to schedule a task on to reduce the number of preemptions.

mig.rat	preempt. rat				
	1 %	5 %	10 %	20 %	50 %
1 %	0.05	0.04	0.03	0.02	0.01
	0.05	0.03	0.02	0.01	0.00
	0.04	0.04	0.03	0.02	0.01
5 %	0.05	0.04	0.03	0.02	0.01
	0.03	0.02	0.01	0.00	0.00
	0.04	0.04	0.03	0.02	0.01
10 %	0.05	0.04	0.03	0.02	0.01
	0.02	0.01	0.01	0.00	0.00
	0.04	0.04	0.03	0.02	0.01
20 %	0.04	0.03	0.02	0.02	0.01
	0.01	0.01	0.00	0.00	0.00
	0.04	0.04	0.03	0.02	0.01
50 %	0.02	0.02	0.01	0.01	0.01
	0.00	0.00	0.00	0.00	0.00
	0.04	0.04	0.03	0.02	0.01

(a) $m = 2$

mig.rat	preempt. rat				
	1 %	5 %	10 %	20 %	50 %
1 %	0.21	0.19	0.16	0.13	0.08
	0.19	0.11	0.07	0.02	0.00
	0.18	0.16	0.13	0.09	0.05
5 %	0.20	0.18	0.15	0.12	0.08
	0.12	0.08	0.04	0.01	0.00
	0.18	0.16	0.13	0.09	0.05
10 %	0.18	0.16	0.14	0.12	0.07
	0.07	0.05	0.02	0.01	0.00
	0.18	0.16	0.13	0.09	0.05
20 %	0.15	0.14	0.13	0.10	0.07
	0.02	0.01	0.01	0.00	0.00
	0.18	0.16	0.13	0.09	0.05
50 %	0.11	0.10	0.09	0.08	0.05
	0.00	0.00	0.00	0.00	0.00
	0.18	0.16	0.13	0.09	0.05

(b) $m = 3$

mig.rat	preempt. rat				
	1 %	5 %	10 %	20 %	50 %
1 %	0.40	0.37	0.33	0.29	0.21
	0.36	0.23	0.17	0.12	0.12
	0.35	0.32	0.27	0.22	0.16
5 %	0.38	0.35	0.32	0.28	0.21
	0.24	0.18	0.14	0.12	0.12
	0.35	0.32	0.27	0.22	0.16
10 %	0.36	0.33	0.31	0.26	0.20
	0.17	0.14	0.13	0.12	0.12
	0.35	0.32	0.27	0.22	0.16
20 %	0.31	0.30	0.27	0.24	0.20
	0.13	0.12	0.12	0.12	0.12
	0.35	0.32	0.27	0.22	0.16
50 %	0.24	0.23	0.22	0.21	0.18
	0.12	0.12	0.12	0.12	0.12
	0.35	0.32	0.27	0.22	0.16

(c) $m = 4$

mig.rat	preempt. rat				
	1 %	5 %	10 %	20 %	50 %
1 %	0.76	0.72	0.67	0.60	0.49
	0.67	0.45	0.36	0.34	0.34
	0.69	0.62	0.54	0.46	0.38
5 %	0.73	0.69	0.65	0.58	0.49
	0.45	0.37	0.35	0.34	0.34
	0.69	0.62	0.54	0.46	0.38
10 %	0.69	0.66	0.62	0.56	0.48
	0.36	0.35	0.34	0.34	0.34
	0.69	0.62	0.54	0.46	0.38
20 %	0.63	0.60	0.57	0.53	0.47
	0.34	0.34	0.34	0.34	0.34
	0.69	0.62	0.54	0.46	0.38
50 %	0.52	0.50	0.49	0.47	0.44
	0.34	0.34	0.34	0.34	0.34
	0.69	0.62	0.54	0.46	0.38

(d) $m = 6$

Figure 24.7: Success ratio for scheduling algorithms as a function of the number of processors, preemption costs and migration costs. Each square shows adaptiveTkCaware (upper), adaptiveTkCunaware (middle), R-BOUND-MP (lower).

24.5 Discussion

From the previous sections, we have learned that the relative performance of partitioned scheduling and global scheduling varies depending on the system models and performance measures used. However, the decision whether to partition or not to partition may also need to consider other aspects than comparing success ratio. Below we list some interesting aspects.

Global scheduling is the best way of maximizing the resource utilization when a task's actual execution time is much lower than its stated worst-case execution time [LMM98b]. This situation can occur when the execution time of a task depends highly on user input or sensor values. Since partitioned scheduling is guided by the worst-case execution time during the partitioning decisions, there is a risk that the actual resource usage will be lower than anticipated, and thus wasted if no dynamic exploitation of the spare capacity is made. Our suggested hybrid approach hence offers one solution for exploiting processors efficiently, while at the same time providing guarantees for those tasks that require so. The hybrid solution proposed in this chapter applies partitioned scheduling to the task set until all processors have been filled. The remaining tasks are then scheduled using global scheduling. An alternative approach would be to partition only the tasks that have hard real-time constraints, and then let the tasks with soft constraints be scheduled by global scheduling.

Partitioned scheduling has the advantage of having a low computational complexity of the schedulability test, something that is important for, e.g., online admission tests and QoS negotiation [AAS97]. There is also a complexity associated with the algorithm that reschedules a task set that has changed. Global scheduling has lower computational complexity for rescheduling than partitioned scheduling if partitioned scheduling use the response-time analysis. However, even if sufficient schedulability tests are used for the partitioned method, then global scheduling has a slightly lower computational complexity of rescheduling ($O(n \log n)$ for adaptiveTkC) than partitioned scheduling ($O(nm + n \log n)$ for R-BOUND-MP and $O(nm + n \log n)$ for RM-FFDU). For systems that reschedule frequently, but do not rely on schedulability tests, e.g., feedback control scheduling [LSTS99], global scheduling, with its slightly higher success ratio and its slightly lower computational complexity of rescheduling, becomes a viable alternative. Also, since these systems are likely to be used when the execution time varies, the advantage of reclaiming resources in global scheduling gives a further performance increase.

24.6 Conclusions

Before 1999, it was believed that global scheduling was inferior to partitioned scheduling. In this paper, we have shown through three evaluations that by using a better priority assignment scheme and an improved dispatcher, such a belief is not necessarily true. First, an average-case comparison using an idealized architecture, showed that, if a system has a small number of processors, then global scheduling offers higher performance than partitioned scheduling. Second, an average-case comparison using a realistic architecture, showed that, for several combinations of preemption and migration costs, global scheduling offers higher performance.

Bibliography

- [AAJ⁺02] T. Abdelzaher, B. Andersson, J. Jonsson, V. Sharma, and M. Nguyen. The aperiodic multiprocessor utilization bound for liquid tasks. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium*, San José, California, September 24–27 2002.
- [AAS97] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin. QoS negotiation in real-time systems and its application to automated flight control. In *Proc. of the IEEE Real-Time Technology and Applications Symposium*, pages 228–238, Montreal, Canada, June 9–11, 1997.
- [AJ00] B. Andersson and J. Jonsson. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. In *Proc. of the IEEE Real-Time Systems Symposium – Work-in-Progress Session*, Orlando, Florida, November 27–30, 2000. Also in TR-00-10, Dept. of Computer Engineering, Chalmers University of Technology.
- [AL01] T. Abdelzaher and C. Lu. Schedulability analysis and utilization bounds for highly scalable real-time services. In *Proc. of the IEEE Real-Time Technology and Applications Symposium*, pages 15–25, Taipei, Taiwan, May 30–1, 2001.
- [AS03] T. Abdelzaher and V. Sharma. A synthetic utilization bound for aperiodic tasks with resource requirements. In *Proc. of the 15th EuroMicro Conference on Real-Time Systems*, pages 141–150, Porto, Portugal, July 2–4 2003.

- [Bak03] T. P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proc. of the IEEE Real-Time Systems Symposium*, Cancun, Mexico, December 3–5 2003.
- [BLOS95] A. Burchard, J. Liebeherr, Y. Oh, and S.H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers*, 44(12):1429–1442, December 1995.
- [DD85] S. Davari and S.K. Dhall. On a real-time task allocation problem. In *19th Annual Hawaii International Conference on System Sciences*, pages 8–10, Honolulu, Hawaii, 1985.
- [DD86] S. Davari and S.K. Dhall. An on-line algorithm for real-time task allocation. In *Proc. of the IEEE Real-Time Systems Symposium*, volume 7, pages 194–200, New Orleans, LA, December 1986.
- [Dha77] S. K. Dhall. *Scheduling Periodic-Time-Critical Jobs on Single Processor and Multiprocessor Computing Systems*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1977.
- [DL78] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, January/February 1978.
- [JP86] M. Joseph and P. Pandya. Finding response times in a real-time system. *Computer Journal*, 29(5):390–395, October 1986.
- [Leu89] J. Y.-T. Leung. A new algorithm for scheduling periodic, real-time tasks. *Algorithmica*, 4(2):209–219, 1989.
- [Liu69] C. L. Liu. Scheduling algorithms for multiprocessors in a hard real-time environment. In *JPL Space Programs Summary 37-60*, volume II, pages 28–31. 1969.
- [LL73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.
- [LL03] L. Lundberg and H. Lennerstad. Global multiprocessor scheduling of tasks using time-independent priorities. In *Proc. of the IEEE Real Time Technology and Applications Symposium*, volume 9, Washington, DC, May 27–30, 2003.

- [LMM98a] S. Lauzac, R. Melhem, and D. Mossé. Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. In *Proc. of the EuroMicro Workshop on Real-Time Systems*, pages 188–195, Berlin, Germany, June 17–19, 1998.
- [LMM98b] S. Lauzac, R. Melhem, and D. Mossé. Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. In *10th Euromicro Workshop on Real Time Systems*, pages 188–195, Berlin, Germany, June 17–19, 1998.
- [LMM98c] S. Lauzac, R. Melhem, and D. Mossé. An efficient RMS admission control and its application to multiprocessor scheduling. In *Proc. of the IEEE Int'l Parallel Processing Symposium*, pages 511–518, Orlando, Florida, March 1998.
- [LSTS99] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Design and evaluation of a feedback control EDF scheduling algorithm. In *Proc. of the IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1–3, 1999.
- [Lun98] L. Lundberg. Multiprocessor scheduling of age constraint processes. In *5th International Conference on Real-Time Computing Systems and Applications*, Hiroshima, Japan, October 27–29, 1998.
- [Lun02] L. Lundberg. Analyzing fixed-priority global multiprocessor scheduling. In *Proc. of the IEEE Real Time Technology and Applications Symposium*, volume 8, pages 145–153, San José, California, September 24–27, 2002.
- [LW82] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, December 1982.
- [OS95a] Y. Oh and S. H. Son. Allocating fixed-priority periodic tasks on multiprocessor systems. *Real-Time Systems*, 9(3):207–239, November 1995.
- [OS95b] Y. Oh and S. H. Son. Fixed-priority scheduling of periodic tasks on multiprocessor systems. Technical Report 95-16, Department of Computer Science, University of Virginia, March 1995.
- [PR94] V. Padmanabhan and D. Roselli. The real cost of context switching. Technical report, CS Division, University of California at Berkeley, November 1994.

- [SVC98] S. Sáez, J. Vila, and A. Crespo. Using exact feasibility tests for allocating real-time tasks in multiprocessor systems. In *10th Euromicro Workshop on Real Time Systems*, pages 53–60, Berlin, Germany, June 17–19, 1998.

Chapter 25

Schedulability-Driven Communication Synthesis for Time Triggered Embedded Systems

By **Paul Pop**, **Petru Eles** and **Zebo Peng**
Department of Computer and Information Science
Linköping University
Email: {paupo,petel,zpe}@ida.liu.se

We present an approach to static priority preemptive process scheduling for the synthesis of hard real-time distributed embedded systems where communication plays an important role. The communication model is based on a time-triggered protocol. We have developed an analysis for the communication delays with four different message scheduling policies over a time-triggered communication channel. Optimization strategies for the synthesis of communication are developed, and the four approaches to message scheduling are compared using extensive experiments.

25.1 Introduction

Depending on the particular application, an embedded system has certain requirements on performance, cost, dependability, size, etc. For hard real-time applications the timing requirements are extremely important: failing to meet a deadline can potentially lead to a catastrophic event. Thus, in order to function correctly, an embedded system implementing such an application has to meet its deadlines. One of the typical application areas for such systems is that of safety-critical automotive applications, e.g. drive-by-wire, brake-by-wire [bWC98].

Due to the complexity of embedded systems, hardware/software co-synthesis environments are developed to assist the designer in finding the most cost effective solution that, at the same time, meets the design requirements [YW98].

In this chapter we concentrate on the schedulability analysis and communication synthesis of embedded hard real-time systems which are implemented on distributed architectures. Process scheduling is based on a static priority preemptive approach while the bus communication is statically scheduled.

Preemptive scheduling of independent processes with static priorities running on single-processor architectures has its roots in the work of Liu and Layland [LL73]. The approach has been later extended to accommodate more general computational models and has also been applied to distributed systems [TC94]. The reader is referred to [ABD⁺95, BLMSV98, SR93] for surveys on this topic. In [LPW99] an earlier deadline first strategy is used for non-preemptive scheduling of processes with possible data dependencies. Preemptive and non-preemptive static scheduling are combined in the cosynthesis environment described in [DLJ97, DJ98]. In many of the previous scheduling approaches researchers have assumed that processes are scheduled independently. However, this is not the case in reality, where process sets can exhibit both data and control dependencies. Moreover, knowledge about these dependencies can be used in order to improve the accuracy of schedulability analyses and the quality of produced schedules. One way of dealing with data dependencies between processes with static priority based scheduling has been indirectly addressed by the extensions proposed for the schedulability analysis of distributed systems through the use of the *release jitter* [TC94]. Release jitter is the worst case delay between the arrival of a process and its release (when it is placed in the run-queue for the processor) and can include the communication delay due to the transmission of a message on the communication channel.

In [TC94] and [YW98] time offset relationships and phases, respectively, are used in order to model data dependencies. Offset and phase are similar concepts that express the existence of a fixed interval in time between the arrivals of sets of processes. The authors show that by introducing such concepts into the computational model, the pessimism of the analysis is significantly reduced when bounding the time behaviour of the system. The concept of dynamic offsets has been later introduced in [GH98] and used to model data dependencies [GH99].

Currently, more and more real-time systems are used in physically distributed environments and have to be implemented on distributed architectures in order to meet reliability, functional, and performance constraints. However, researchers have often ignored or very much simplified aspects concerning the communication infrastructure. One typi-

cal approach is to consider communication processes as processes with a given execution time (depending on the amount of information exchanged) and to schedule them as any other process, without considering issues like communication protocol, bus arbitration, packaging of messages, clock synchronization, etc. These aspects are, however, essential in the context of safety-critical distributed real-time applications and one of our objectives is to develop a strategy which takes them into consideration for process scheduling. Many efforts dedicated to communication synthesis have concentrated on the synthesis support for the communication infrastructure but without considering hard real-time constraints and system level scheduling aspects [CB95, DIJ95, NG94]. Lower level communication synthesis aspects under timing constraints have been addressed in [OB98, KM98]. We have to mention here some results obtained in extending real-time schedulability analysis so that network communication aspects can be handled. In [THW94], for example, the CAN protocol is investigated while the work reported in [EHS97] considers systems based on the ATM protocol. Analysis for a simple time-division multiple access (TDMA) protocol is provided in [TC94] that integrates processor and communication schedulability and provides a “holistic” schedulability analysis in the context of distributed real-time systems. The problem of how to allocate priorities to a set of distributed tasks is discussed in [GH95]. Their priority assignment heuristic is based on the schedulability analysis from [TC94].

In this chapter we consider the time-triggered protocol (TTP) [KB03] as the communication infrastructure for a distributed real-time system. However, the research presented is also valid for any other TDMA-based bus protocol that schedules the messages statically based on a schedule table like, for example, the SAFEbus [HD92] protocol used in the avionics industry. [Rus01] provides a comparison of bus architectures for safety-critical embedded systems, concluding that TTP “is unique in being used for both automobile applications, where volume manufacture leads to very low prices, and aircraft, where a mature tradition of design and certification for flight-critical electronics provides strong scrutiny of arguments for safety.”

In this chapter, we consider that processes are scheduled according to a static priority preemptive policy. TTP has been classically associated with non-preemptive static scheduling of processes, mainly because of fault tolerance reasons [Kop97]. In [PEP99, EDPP00] we have addressed the issue of non-preemptive static process scheduling and communication synthesis using TTP. However, considering preemptive priority based scheduling at the process level, with time triggered static scheduling at the communication level, can be the right solution under certain circumstances [LA99]. A communication protocol like TTP provides a global time base, improves fault-tolerance and predictability. At

the same time, certain particularities of the application or of the underlying real-time operating system can impose a priority based scheduling policy at the process level.

The contribution of this chapter is threefold. First we develop a schedulability analysis for distributed processes with preemptive priority based scheduling considering a TTP-based communication infrastructure. As our second contribution, we have proposed four different approaches to message scheduling using static and dynamic message allocation. Finally, the third contribution is showing how the parameters of the communication protocol can be optimized in order to fit the communication particularities of a certain application. Thus, based on our approach, it is not only possible to determine if a certain task set implemented on a TTP-based distributed architecture is schedulable, but it is also possible to select a particular message passing strategy and also to optimize certain parameters of the communication protocol. By adapting the communication infrastructure to certain particularities of the task set we increase the likelihood of producing an implementation which satisfies all time constraints.

The chapter is divided into seven sections. The next section presents the architectures considered for system implementation. The computational model assumed and formulation of the problem are presented in Section 25.3, and Section 25.4 presents the schedulability analysis for each of the four approaches considered for message scheduling. The optimization strategy is presented in Section 25.5, and the four approaches are evaluated in Section 25.6. The last section presents our conclusions.

25.2 System Architecture

25.2.1 Hardware Architecture

We consider architectures consisting of nodes connected by a broadcast communication channel. Every node consists of a TTP controller, a CPU, a RAM, a ROM and an I/O interface to sensors and actuators (Figure 25.1). A node can also have an ASIC (Application Specific Integrated Circuit) in order to accelerate parts of its functionality.

Communication between nodes is based on the TTP [KB03]. TTP was designed for distributed real-time applications that require predictability and reliability (e.g, drive-by-wire). It integrates all the services necessary for fault-tolerant real-time systems.

The communication channel is a broadcast channel, so a message sent by a node is received by all the other nodes. The bus access scheme is time-division multiple-access (TDMA) (Figure 25.2). Each node N_i can transmit only during a predetermined time interval, the so called TDMA slot S_i . In such a slot a node can send several messages packaged

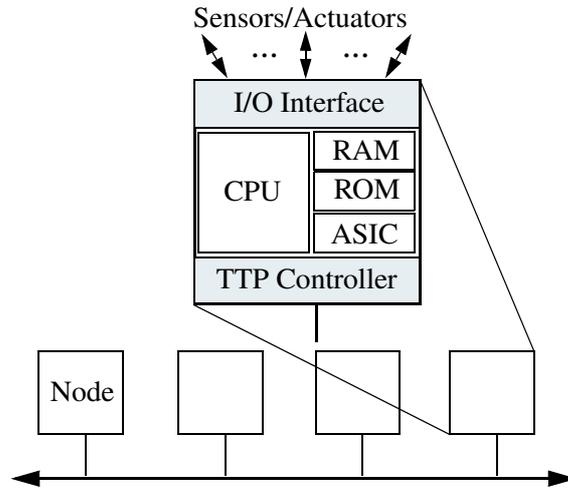


Figure 25.1: System Architecture

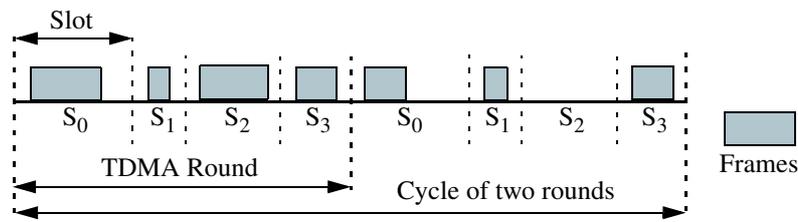


Figure 25.2: Bus Access Scheme

in a frame. A sequence of slots corresponding to all the nodes in the architecture is called a TDMA round. A node can have only one slot in a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically. The sequence and length of the slots are the same for all TDMA rounds. However, the length and contents of the frames may differ.

Every node has a TTP controller that implements the protocol services and runs independently of the nodes CPU (Figure 25.3). Communication with the CPU is performed through a so called message base interface (MBI) which is usually implemented as a dual ported RAM.

The TDMA access scheme is imposed by a so called message descriptor list (MEDL) that is located in every TTP controller. The MEDL basically contains: the time when a frame has to be sent or received, the address of the frame in the MBI and the length of the frame. MEDL serves as a schedule table for the TTP controller which has to know when to send or receive a frame to or from the communication channel.

The TTP controller provides each CPU with a timer interrupt based on a local clock synchronized with the local clocks of the other nodes.

Clock synchronization is done by comparing the a priori known time of arrival of a frame with the observed arrival time. Thus, TTP provides a global time-base of known precision, without any overhead on the communication.

25.2.2 Software Architecture

We have designed a software architecture which runs on the CPU in each node and which has a real-time kernel as its main component. Each kernel has a so called tick scheduler that is activated periodically by the timer interrupts and decides on activation of processes, based on their priorities. Several activities, like polling of the I/O or diagnostics, also take place in the timer interrupt routine.

In order to run a predictable hard real-time application, the overhead of the kernel and the worst case administrative overhead (WCAO) of every system call have to be determined. Our schedulability analysis takes into account these overheads and also the overheads due to message passing.

The message passing mechanism is illustrated in Figure 25.3, where we have three processes, P_1 to P_3 . P_1 and P_2 are mapped on node N_0 that transmits in slot S_0 , and P_3 is mapped on node N_1 that transmits in slot S_1 . Message m_1 is transmitted between P_1 and P_2 that are on the same node, while message m_2 is transmitted from P_1 to P_3 between the two nodes.

Messages between processes located on the same processor are passed through shared protected objects. The overhead for their communication is accounted for by the blocking factor, using the analysis from [SRL90] for the priority ceiling protocol.

Message m_2 has to be sent from node N_0 to node N_1 . Thus, after m_2 is produced by P_1 it will be placed into an outgoing message queue, called *Out*. The access to the queue is guarded by a priority-ceiling semaphore. A so called transfer process (denoted by T in Figure 25.3) moves the message from the *Out* queue into the MBI.

How the message queue is organized and how the message transfer process selects the particular messages and assembles them into a frame, depend on the particular approach chosen for message scheduling (see Section 25.4). The message transfer process is activated, at certain a priori known moments, by the tick scheduler in order to perform the message transfer. These activation times are stored in a message handling time table (MHTT) available to the real-time kernel in each node. Both the MEDL and the MHTT are generated off-line as result of the schedulability analysis and optimization which will be discussed later. The MEDL imposes the times when the TTP controller of a certain node has to move frames from the MBI to the communication channel. The

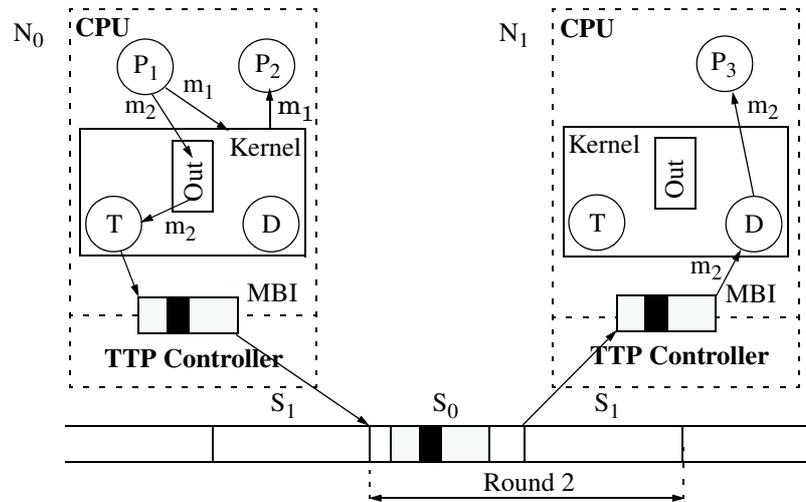


Figure 25.3: Message Passing Mechanism

MHTT contains the times when messages have to be transferred by the message transfer process from the *Out* queue into the MBI, in order to be broadcasted by the TTP controller. As result of this synchronization, the activation times in the MHTT are directly related to those in the MEDL and the first table results directly form the second one.

It is easy to observe that we have the most favourable situation when, at a certain activation, the message transfer process finds in the *Out* queue all the “expected” messages which then can be packed into the immediate following frame to be sent by the TTP controller. However, application processes are not statically scheduled and availability of messages in the *Out* queue cannot be guaranteed at fixed times. Worst-case situations have to be considered, as will be shown in Section 25.4.

Let us consider Figure 25.3 again. There we assumed a context in which the broadcasting of the frame containing message m_2 is done in the slot S_0 of *Round2*. The TTP controller of node N_1 knows from its MEDL that it has to read a frame from slot S_0 of *Round 2* and to transfer it into its MBI. In order to synchronize with the TTP controller and to read the frame from the MBI, the tick scheduler on node N_1 will activate, based on its local MHTT, a so called delivery process, denoted with D in Figure 25.3. The delivery process takes the frame from the MBI and extracts the messages from it. For the case when a message is split into several packets, sent over several TDMA rounds, we consider that a message has arrived at the destination node after all its constituent packets have arrived. When m_2 has arrived, the delivery process copies it to process P_3 which will be activated. Activation times for the delivery process are fixed in the MHTT just as explained earlier for the message transfer process.

The number of activations of the message transfer and delivery processes depends on the number of frames transferred, and it is taken into account in our analysis as also is the delay implied by the propagation on the communication bus.

25.3 Problem Formulation

We model an application as a set of processes. Each process P_i is allocated to a certain processor, and has a known worst case execution time C_i , a period T_i , a deadline D_i , and a uniquely assigned priority. We consider a preemptive execution environment, which means that higher priority processes can interrupt the execution of lower priority processes. A lower priority process can block a higher priority process (e.g., it is in its critical section), and the blocking time is computed according to the priority ceiling protocol. Processes exchange messages, and for each message m_i we know its size S_{m_i} . A message is sent once in every n_m invocations of the sending process, with a period $T_m = n_m T_i$ inherited from the sender process P_i , and has a unique destination process. Each process is allocated to a node of the distributed system and messages are transmitted according to the TTP.

We are interested to synthesize the MEDL of the TTP controllers (and, as a direct consequence, also the MHTTs) so that the process set is schedulable on an as cheap (slow) as possible processor set.

The next section presents the schedulability analysis for each of the four approaches considered for message scheduling, under the assumptions outlined above. In Section 25.5, the response times calculated using this schedulability analysis are combined in a cost function that measures the “degree of schedulability” of a given design alternative. This “degree of schedulability” is then used to drive the optimization and synthesis the MEDL and the MHTTs.

25.4 Schedulability Analysis

Under the assumptions presented in the previous section, [TC94] integrate processor and communication scheduling and provide a “holistic” schedulability analysis in the context of distributed real-time systems with communication based on a simple TDMA protocol. The validity of this analysis has been later confirmed in [PGH97]. The analysis belongs to the class of response time analyses, where the schedulability test is whether the worst case response time of each process is smaller or equal than its deadline. In the case of a distributed system, this response time also depends on the communication delay due to messages. In [TC94] the analysis for messages is done in a similar way as for processes: a

message is seen as an unpreemptable process that is “running” on a bus.

The basic idea in [TC94] is that the release jitter of a destination process depends on the communication delay between sending and receiving a message. The release jitter of a process is the worst case delay between the arrival of the process and its release (when it is placed in the run-queue for the processor). The communication delay is the worst case time spent between sending a message and the message arriving at the destination process.

Thus, for a process $d(m)$ that receives a message m from a sender process $s(m)$, the release jitter is

$$J_{d(m)} = r_{s(m)} + a_m + r_{deliver} + T_{tick} \quad (25.1)$$

where $r_{s(m)}$ is the response time of the process sending the message, a_m (worst case arrival time) is the worst case time needed for message m to arrive at the communication controller of the destination node, $r_{deliver}$ is the response time of the delivery process (see Section 25.2.2), and T_{tick} is the jitter due to the operation of the tick scheduler. The communication delay for a message m (also referred to as the “response time” of message m) is

$$r_m = a_m + r_{deliver} \quad (25.2)$$

where a_m itself is the sum of the access delay Y_m and the propagation delay X_m . The access delay is the time a message queued at the sending processor spends waiting for the use of the communication channel. In a_m we also account for the execution time of the message transfer process (see Section 25.2.2). The propagation delay is the time taken for the message to reach the destination processor once physically sent by the corresponding TTP controller. The analysis assumes that the period T_m of any message m is longer or equal to the length of a TDMA round, $T_m \geq T_{TDMA}$ (see Figures 25.2 and 25.4).

The pessimism of this analysis can be reduced by using the notion of offset in order to model precedence relations between processes [Tin94]. The basic idea is to exclude certain scenarios which are impossible due to precedence constraints. By considering dynamic offsets the tightness of the analysis can be further improved [GH98, GH99]. Similar results have been reported in [YW98]. In [PEP00] we have shown how such an analysis can be performed in the presence of not only precedence constraints but also control dependencies between processes, where the activation of processes depends on conditions computed by so-called “disjunction processes”. In the present chapter our attention is concentrated on the analysis of network communication delays and on optimization of message passing strategies. In order to keep the discussion focused we present

our analysis starting from the results in [TC94]. All the conclusions this research apply as well to the developments addressing precedence relations proposed, for example, in [GH98, GH99].

Although there are many similarities with the general TDMA protocol, the analysis in the case of TTP is different in several aspects and also differs to a large degree depending on the policy chosen for message scheduling.

Before going into details for each of the message scheduling approaches proposed by us, we analyze the propagation delay and the message transfer and delivery processes, as they do not depend on the particular message scheduling policy chosen. The propagation delay X_m of a message m sent as part of a slot S , with the TTP protocol, is equal to the time needed for the slot S to be transferred on the bus (this is the slot size expressed in time units, see Figure 25.4). This time depends on the number of bits which can be packed into the slot and on the features of the underlying bus.

The overhead produced by the communication activities must be accounted for not only as part of the access delay for a message, but also through its influence on the response time of processes running on the same processor. We consider this influence during the schedulability analysis of processes on each processor. We assume that the worst case computation time of the transfer process (T in Figure 25.3) is known, and that it is different for each of the four message scheduling approaches. Based on the respective MHTT, the transfer process is activated for each frame sent. Its worst case period is derived from the minimum time between successive frames.

The response time of the delivery process (D in Figure 25.3), $r_{deliver}$, is part of the communication delay (Equation 25.2). The influence due to the delivery process must be also included when analyzing the response time of the processes running on the respective processor. We consider the delivery process during the schedulability analysis in the same way as the message transfer process.

The response times of the communication and delivery processes are calculated, as for all other processes, using the arbitrary deadline analysis from [TC94].

The four approaches we propose for scheduling of messages using TTP differ in the way the messages are allocated to the communication channel (either statically or dynamically) and whether they are split or not into packets for transmission. The next subsections present the analysis for each approach as well as the degrees of liberty a designer has, in each of the cases, for optimizing the MEDL. First, we discuss each approach in the case when the arrival time a_m of a message m is smaller or equal with its period T_m . Then, in Section 25.4.5, we present the generalization for the case $a_m > T_m$.

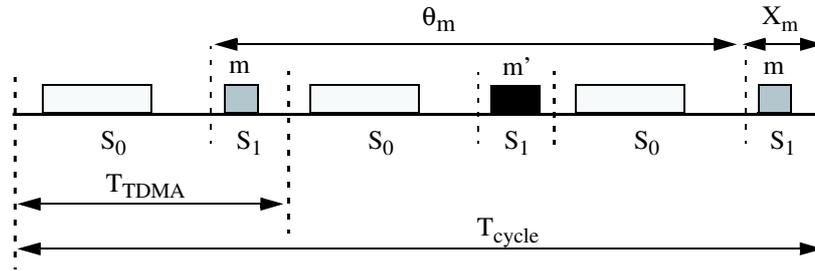


Figure 25.4: Worst case arrival time for SM

25.4.1 Static Single Message Allocation (SM)

The first approach to scheduling of messages using TTP is to statically (off-line) schedule each of the messages into a slot of the TDMA cycle, corresponding to the node sending the message. This means that for each message we decide off-line to allocate space in one or more frames, space that can only be used by that particular message. In Figure 25.4 the frames are denoted by rectangles. In this particular example, it has been decided to allocate space for message m in slot S_1 of the first and third rounds. Since the messages are dynamically produced by the processes, the exact moment a certain message is generated cannot be predicted. Thus, it can happen that certain frames will be left empty during execution. For example, if there is no message m in the *Out* queue (see Figure 25.3) when the slot S_1 of the first round in Figure 25.4 starts, that frame will carry no information. A message m produced immediately after slot S_1 has left, could then be carried by the frame scheduled in the slot S_1 of the third round.

In the SM approach, we consider that the slots can hold each maximum one single message. This approach is well suited for application areas, like safety-critical automotive electronics, where the messages are typically short and the ability to easily diagnose the system (fewer messages in a frame are easier to observe) is critical. In the automotive electronics area messages are typically a couple of bytes, encoding signals like vehicle speed. However, for applications using larger messages, the SM approach leads to overheads due to the inefficient utilization of slot space when transmitting smaller size messages.

As each slot carries only one fixed, predetermined message, there is no interference among messages. If a message m misses its allocated frame it has to wait for the following slot assigned to m . The worst case access delay Y_m for a message m in this approach is the maximum time between consecutive slots of the same node carrying the message m . We denote this time by θ_m , illustrated in Figure 25.4, where we have a system cycle of length T_{cycle} , consisting of three TDMA rounds.

In this case, the worst case arrival time a_m of a message m becomes $\theta_m + X_m$. Therefore, the main aspect influencing schedulability of the messages is the way they are statically allocated to slots, which determines the values of θ_m . θ_m , as well as X_m , depend on the slot sizes which in the case of SM are determined by the size of the largest message sent from the corresponding node plus the bits for control and CRC, as imposed by the protocol.

As mentioned before, the analysis in [TC94], done for a simple TDMA protocol, assumes that $T_m \geq T_{TDMA}$. In the case of static message allocation with TTP (the SM and MM approaches), this translates to the condition $T_m \geq \theta_m$.

During the synthesis of the MEDL, the designer has to allocate the messages to slots in such a way that the process set is schedulable. Since the schedulability of the process set can be influenced by the synthesis of the MEDL only through the θ_m parameters, these are the parameters which have to be optimized.

Let us consider the simple example depicted in Figure 25.5, where we have three processes, P_1 , P_2 , and P_3 running each on a different processor. When process P_1 finishes executing it sends message m_1 to process P_3 and message m_2 to process P_2 . In the TDMA configurations presented in Figure 25.5, only the slot corresponding to the CPU running P_1 is important for our discussion and the other slots are represented with light gray. With the configuration in Figure 25.5a, where the message m_1 is allocated to the rounds 1 and 4 and the message m_2 is allocated to rounds 2 and 3, process P_2 misses its deadline because of the release jitter due to the message m_2 in *Round2*. However, if we have the TDMA configuration depicted in Figure 25.5b, where m_1 is allocated to the rounds 2 and 4 and m_2 is allocated to the rounds 1 and 3, all the processes meet their deadlines.

25.4.2 Static Multiple Message Allocation (MM)

This second approach is an extension of the first one. In this approach we allow more than one message to be statically assigned to a slot and all the messages transmitted in the same slot are packaged together in a frame. As for the SM approach, there is no interference among messages, so the worst case access delay for a message m is the maximum time between consecutive slots of the same node carrying the message m , θ_m . It is also assumed that $T_m \geq \theta_m$.

However, this approach offers more freedom during the synthesis of the MEDL. We have now to decide also on how many and which messages should be put in a slot. This allows more flexibility in optimizing the θ_m parameter. To illustrate this, let us consider the same example depicted in Figure 25.5. With the MM approach, the TDMA configuration can

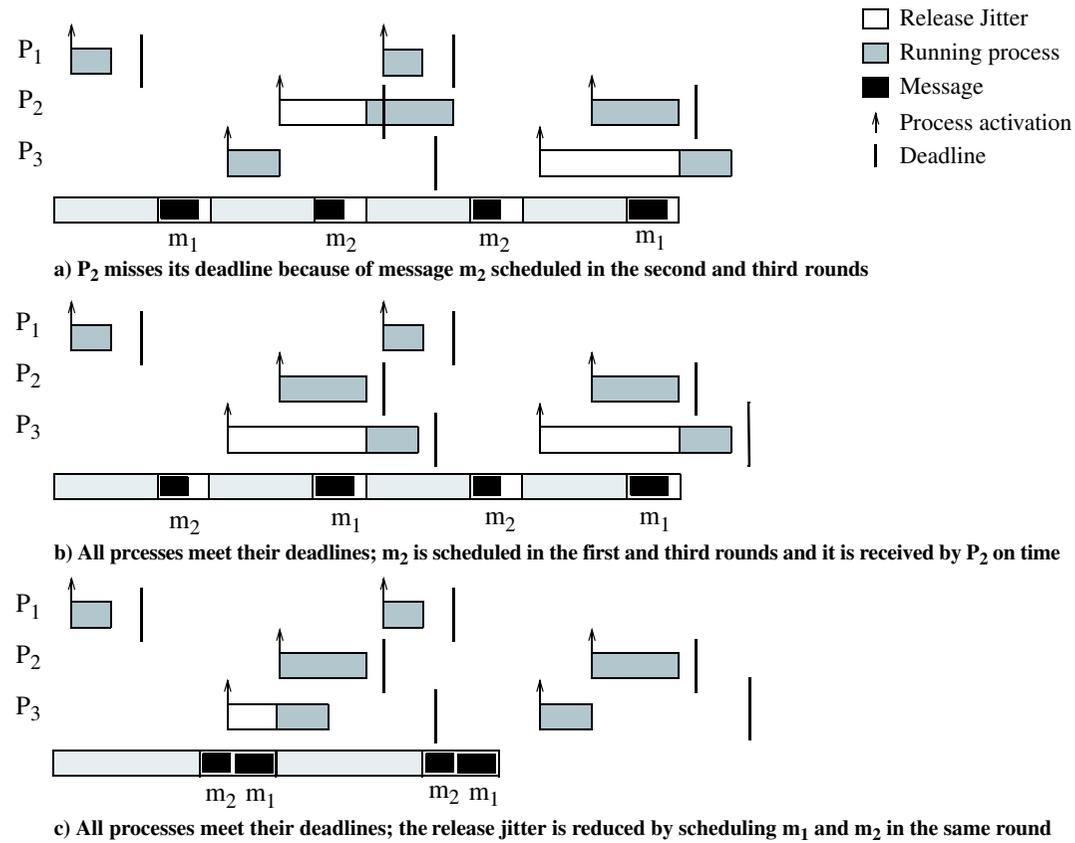


Figure 25.5: Optimizing the MEDL for SM and MM

be arranged as depicted in Figure 5c, where the messages m_1 and m_2 are put together in the same slot in the rounds 1 and 2. Thus, the deadline is met and the release jitter is further reduced compared to the case presented in Figure 25.5b where process P_3 was experiencing a large release jitter.

25.4.3 Dynamic Message Allocation (DM)

The previous two approaches have statically allocated one or more messages to their corresponding slots. This third approach considers that the messages are dynamically allocated to frames, as they are produced.

Thus, when a message is produced by a sender process it is placed in the *Out* queue (Figure 25.3). Messages are ordered according to their priority. At its activation, the message transfer process takes a certain number of messages from the head of the *Out* queue and constructs the frame. The number of messages accepted is decided so that their total size does not exceed the length of the data field of the frame. This length is limited by the size of the slot corresponding to the respective processor. Since the messages are sent dynamically, we have to identify them in a certain way so that they are recognized when the frame arrives at the delivery process. We consider that each message has several identifier bits appended at the beginning of the message.

Since we dynamically package messages into frames in the order they are sorted in the queue, the access delay to the communication channel for a message m depends on the number of messages queued ahead of it.

The analysis in [TC94] bounds the number of queued ahead *packets* of messages of higher priority than message m , as in their case it is considered that a message can be split into packets before it is transmitted on the communication channel. We use the same analysis but we have to apply it for the number of *messages* instead of packets. We have to consider that messages can be of different sizes as opposed to packets which always are of the same size.

Therefore, the total *size* of higher priority messages queued ahead of a message m , in the worst case, is:

$$I_m = \sum_{\forall j \in hp(m)} \left\lceil \frac{r_{s(j)}}{T_j} \right\rceil S_j \quad (25.3)$$

where S_j is the size of the message m_j , $r_{s(j)}$ is the response time of the process sending message m_j , and T_j is the period of the message m_j .

Further, we calculate the worst case time that a message m spends in the *Out* queue. The number of TDMA rounds needed, in the worst case, for a message m placed in the queue to be removed from the queue

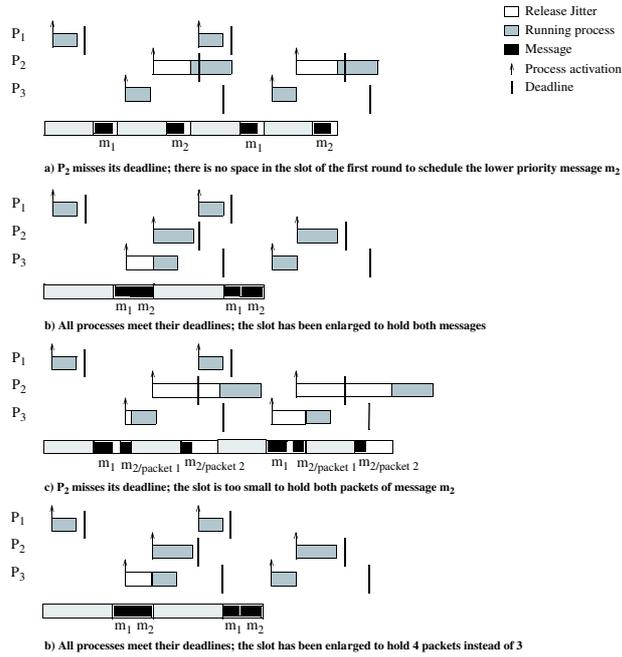


Figure 25.6: Optimizing the MEDL for DM and DP

for transmission is

$$\left\lceil \frac{S_m + I_m}{S_s} \right\rceil \quad (25.4)$$

where S_m is the size of the message m and S_s is the size of the slot transmitting m (we assume, in the case of DM, that for any message x , $S_x \leq S_s$). This means that the worst case time a message m spends in the *Out* queue is given by

$$Y_m = \left\lceil \frac{S_m + I_m}{S_s} \right\rceil T_{TDMA} \quad (25.5)$$

where T_{TDMA} is the time taken for a TDMA round.

Since the size of the messages is given with the application, the parameter that will be optimized during the synthesis of the MEDL is the slot size. To illustrate how the slot size influences schedulability, let us consider the example in Figure 25.6 where we have the same setting as for the example in Figure 25.5. The difference is that we consider message m_1 having a higher priority than message m_2 and we schedule the messages dynamically as they are produced. With the configuration in Figure 25.6a message m_1 will be dynamically scheduled first in the slot of the first round, while message m_2 will wait in the *Out* queue until the next round comes, thus causing process P_2 to miss its deadline. However, if we enlarge the slot so that it can accommodate both messages, message

m_2 does not have to wait in the queue and it is transmitted in the same slot as m_1 . Therefore, P_2 will meet its deadline as presented in Figure 25.6b. However, in general, increasing the length of slots does not necessarily improve schedulability, as it delays the communication of messages generated by other nodes.

25.4.4 Dynamic Packet Allocation (DP)

This approach is an extension of the previous one, as we allow the messages to be split into packets before they are transmitted on the communication channel. We consider that each slot has a size that accommodates a frame with the data field being a multiple of the packet size. This approach is well suited for the application areas that typically have large message sizes. By splitting messages into packets we can obtain a higher utilization of the bus and reduce the release jitter. However, since each packet has to be identified as belonging to a message, and messages have to be split at the sender and reconstructed at the destination, the overhead becomes higher than in the previous approaches.

The worst case time a message m spends in the *Out* queue is given by the analysis in [TC94] which is based on similar assumptions as those for this approach:

$$Y_m = \left\lceil \frac{p_m + I_m}{S_p} \right\rceil T_{TDMA} \quad (25.6)$$

where p_m is the number of packets of message m , S_p is the size of the slot (in number of packets) corresponding to m , and

$$I_m = \sum_{\forall j \in hp(m)} \left\lceil \frac{r_{s(j)}}{T_j} \right\rceil p_j \quad (25.7)$$

where p_j is the number of packets of a message m_j .

In the previous approach (DM) the optimization parameter for the synthesis of the MEDL was the size of the slots. With this approach we can also decide on the packet size which becomes another optimization parameter. Consider the example in Figure 25.6c where messages m_1 and m_2 have a size of 6 bytes each. The packet size is considered to be 4 bytes and the slot corresponding to the messages has a size of 12 bytes (3 packets) in the TDMA configuration. Since message m_1 has a higher priority than m_2 , it will be dynamically scheduled first in the slot of the first round and it will need 2 packets. In the third packet the first 4 bytes of m_2 are placed. Thus, the remaining 2 bytes of message m_2 have to wait for the next round, causing process P_2 to miss its deadline. However, if we change the packet size to 3 bytes and keep the same size of 12 bytes for the slot, we have 4 packets in the slot corresponding to the CPU running P_1 (Figure 25.6d). Message m_1 will

be dynamically scheduled first and will need 2 packets in the slot of the first round. Hence, m_2 can be sent in the same round so that P_2 can meet its deadline.

In this particular example, with one single sender processor and the particular message and slot sizes as given, the problem seems to be simple. This is, however, not the case in general. For example, the packet size which fits a particular node can be unsuitable in the context of the messages and slot size corresponding to another node. At the same time, reducing the packets size increases the overheads due to the transfer and delivery processes.

25.4.5 Arbitrary Arrival Times

The analysis presented in the previous sections is valid only in the case the arrival time a_m of a message m is smaller or equal with its period T_m . However, in the case $a_m > T_m$ the “arbitrary deadline” analysis from (Lehoczky 1990) has to be used. Lehoczky uses the notion of “level- i busy period” which is the maximum time for which a processor executes tasks of priority greater than or equal to the priority of process P_i . In order to find the worst case response time, a number of busy periods have to be examined.

The same analysis can be applied for messages. Thus, the worst case time message m takes to arrive at the communication controller of the destination node is determined in [TC94] using the arbitrary deadline analysis [Leh90], and is given by:

$$a_m = \max_{q=0,1,2,\dots} (w_m(q) + X_m(q) - qT_m) \quad (25.8)$$

where $Y_m(q) = w_m(q) - qT_m$ is the access delay, $X_m(q)$ is the propagation delay, and T_m is the period of the message. In the previous equation, q is the number of busy periods being examined, and $w_m(q)$ is the width of the level m busy period starting at time qT_m .

The approach to message scheduling in [TC94] is similar to the DP approach and considers that the messages are dynamically allocated to frames as they are produced, and that they can be split into packets before they are transmitted. Thus, the access delay (the time a message queued at the sending processor spends waiting for the use of the communication channel) is determined as being:

$$Y_m(q) = \left\lceil \frac{(q+1)p_m + I_m(w(q))}{S_p} \right\rceil T_{TDMA} - qT_m \quad (25.9)$$

with

$$I_m(w) = \sum_{\forall j \in hp(m)} \left\lceil \frac{w + r_{s(j)}}{T_j} \right\rceil p_j \quad (25.10)$$

where the variables have the same meaning as in the analysis presented in the previous section for the DP approach, and w is a function of m and q and denotes the width of the busy period.

Further, in order to determine the propagation delay $X_m(q)$ of a message m we have to observe that in the case of DP the messages can be split into packets, and thus the transmission of a message can span over several rounds. The analysis in [TC94] determines the propagation delay based on the number of packets that need to be taken from the queue over the time $w_m(q)$ in order to guarantee the transmission of the last packet of m . Thus, the propagation delay depends on the number of busy periods q being examined.

However, in the other three approaches, namely DM, MM and SM, messages cannot be split into packets and the transmission of a message is done in one single round. Therefore, the propagation delay X_m is equal to the slot size in which message m is being transmitted, and thus is not influenced by the number of busy periods being examined.

The access delay in the DM, MM and SM approaches is obtained through a particularization of the analysis for DP presented above. Thus, in the case of DM, in which messages are allocated dynamically but cannot be split into packets, we have:

$$Y_m(q) = \left\lceil \frac{(q+1)S_m + I_m(w(q))}{S_s} \right\rceil T_{TDMA} - qT_m \quad (25.11)$$

with

$$I_m(w) = \sum_{\forall j \in hp(m)} \left\lceil \frac{w + r_{s(j)}}{T_j} \right\rceil S_j \quad (25.12)$$

where the number of packets p_m and the size of the slot (measured in packets) S_p are replaced with the message size S_m and slot size S_s , respectively.

The analysis becomes even simpler in the case of the two static approaches. Since the allocation of messages to slots is statically performed in the SM and MM approaches there is no interference from other (higher priority) messages but only from later transmissions of the same message. Thus, the interference term I_m due to higher priority messages is null, and the access delay for both SM and MM is:

$$Y_m(q) = (q+1)\theta_m - qT_m. \quad (25.13)$$

25.5 Optimization Strategy

Our problem is to analyze the schedulability of a given process set and to synthesize the MEDL of the TTP controllers (and consequently the

MHTTs) so that the process set is schedulable on an as cheap as possible architecture. The optimization is performed on the parameters which have been identified for each of the four approaches to message scheduling discussed before. In order to guide the optimization process, we need a cost function that captures the “degree of schedulability” for a certain MEDL implementation. Our cost function is similar to that in [TBW92] in the case an application is not schedulable (f_1). However, in order to distinguish between several schedulable applications, we have introduced the second expression, f_2 , which measures, for a feasible design alternative, the total difference between the response times and the deadlines:

$$\text{cost}(\text{optimization parameters}) = \begin{cases} f_1 = \sum_{i=1}^n \max(0, R_i - D_i), & \text{if } f_1 > 0 \\ f_2 = \sum_{i=1}^n R_i - D_i, & \text{if } f_1 = 0 \end{cases} \quad (25.14)$$

where n is the number of processes in the application, R_i is the response time of a process P_i , and D_i is the deadline of a process P_i . If the process set is not schedulable, there exists at least one R_i that is greater than the deadline D_i , therefore the term f_1 of the function will be positive. In this case the cost function is equal to f_1 . However, if the process set is schedulable, then all R_i are smaller than the corresponding deadlines D_i . In this case $f_1 = 0$ and we use f_2 as the cost function, as it is able to differentiate between two alternatives, both leading to a schedulable process set. For a given set of optimization parameters leading to a schedulable process set, a smaller f_2 means that we have improved the response times of the processes, so the application can be potentially implemented on a cheaper hardware architecture (with slower processors and/or bus, but without increasing the number of processors or buses).

The response time R_i is calculated according to the arbitrary deadline analysis [TC94] based on the release jitter of the process (see Section 25.4), its worst case execution time, the blocking time, and the interference time due to higher priority processes. They form a set of mutually dependent equations which can be solved iteratively. As shown in [TC94], a solution can be found if the processor utilization is less than 100%.

For a given application, we are interested to synthesize a MEDL such that the cost function is minimized. We are also interested to evaluate in different contexts the four approaches to message scheduling, thus offering the designer a decision support for choosing the approach that best fits his application.

The MEDL synthesis problem belongs to the class of exponential complexity problems, therefore we are interested to develop heuristics that are able to find accurate results in a reasonable time. We have

developed optimization algorithms corresponding to each of the four approaches to message scheduling. A first set of algorithms presented in Section 25.5.1 is based on simple and fast greedy heuristics. In Section 25.5.2 we introduce a second class of heuristics which aims at finding near-optimal solutions using the simulated annealing (SA) algorithm.

25.5.1 Greedy Heuristics

We have developed greedy heuristics for each of the four approaches to message scheduling. The main idea of the heuristics is to minimize the cost function by incrementally trying to reduce the communication delay of messages and, by this, the release jitter of the processes.

The only way to reduce the release jitter in the SM and MM approaches is through the optimization of the θ_m parameters. This is achieved by a proper placement of messages into slots (see Figure 25.5).

The `OptimizeSM` algorithm presented in Figure 25.7 starts by deciding on a size ($size_{s_i}$) for each of the slots. The slot sizes are set to the minimum size that can accommodate the largest message sent by the corresponding node (lines 1–4 in Figure 25.7). In this approach a slot can carry at most one message, thus slot sizes larger than this size would lead to larger response times.

Then, the algorithm has to decide on the number of rounds, thus determining the size of the MEDL. Since the size of the MEDL is physically limited, there is a limit to the number of rounds (e.g., 2, 4, 8, 16 depending on the particular TTP controller implementation). However, there is a minimum number of rounds *MinRounds* that is necessary for a certain application, which depends on the number of messages transmitted (lines 5–9). For example, if the processes mapped on node N_0 send in total 7 messages then we have to decide on at least 7 rounds in order to accommodate all of them (in the SM approach there is at most one message per slot). Several numbers of rounds, *RoundsNo*, are tried out by the algorithm starting from *MinRounds* up to *MaxRounds* (lines 15–31).

For a given number of rounds (that determine the size of the MEDL) the initially empty MEDL has to be populated with messages in such a way that the cost function is minimized. In order to apply the schedulability analysis that is the basis for the cost function, a *complete* MEDL has to be provided. A complete MEDL contains at least one instance of every message that has to be transmitted between the processes on different processors. A *minimal complete* MEDL is constructed from an empty MEDL by placing one instance of every message m_i into its corresponding empty slot of a round (lines 10–14). In Figure 25.5a, for example, we have a MEDL composed of four rounds. We get a minimal complete MEDL, for example, by assigning m_2 and m_1 to the slots in

```

OptimizeSM
01  -- set the slot sizes
02  for each node  $N_i$  do
03       $size_{S_i} = \max(\text{size of messages } m_j \text{ sent by node } N_i)$ 
04  end for
05  -- find the min. no. of rounds that can hold all the messages
06  for each node  $N_i$  do
07       $nm_i = \text{number of messages sent from } N_i$ 
08  end for
09   $MinRounds = \max(nm_i)$ 
10  -- create a minimal complete MEDL
11  for each message  $m_i$ 
12      find  $round$  in  $[1..MinRounds]$  that has an empty slot for  $m_i$ 
13      place  $m_i$  into its slot in  $round$ 
14  end for
15  for each  $RoundsNo$  in  $[MinRounds..MaxRounds]$  do
16      -- insert messages in such a way that the cost is minimized
17      repeat
18          for each process  $P_i$  that receives a message  $m_i$  do
19              if  $D_i - R_i$  is the smallest so far then  $m = m_{P_i}$  end if
20          end for
21          for each  $round$  in  $[1..RoundsNo]$  do
22              place  $m$  into its corresponding slot in  $round$ 
23              calculate the  $CostFunction$ 
24              if the  $CostFunction$  is smallest so far then
25                   $BestRound = round$ 
26              end if
27              remove  $m$  from its slot in  $round$ 
28          end for
29          place  $m$  into its slot in  $BestRound$  if one was identified
30      until the  $CostFunction$  is not improved
31  end for
end OptimizeSM

```

Figure 25.7: Greedy Heuristic for SM

```

OptimizeDM
01  for each node  $N_i$  do
02       $MinSize_{S_i} = \max(\text{size of messages } m_j \text{ sent by node } N_i)$ 
03  end for
04  -- identifies the size that minimizes the cost function
05  for each slot  $S_j$ 
06       $BestSize_{S_j} = MinSize_{S_j}$ 
07      for each  $SlotSize$  in  $[MinSize_{S_i}..MaxSize]$  do
08          calculate the CostFunction
09          if the CostFunction is best so far then
10               $BestSize_{S_j} = SlotSize_{S_j}$ 
11          end if
12      end for
13       $size_{S_j} = BestSize_{S_j}$ 
14  end for
end OptimizeDM

```

Figure 25.8: Greedy Heuristic for DM

rounds 3 and 4, and letting the slots in rounds 1 and 2 empty. However, such a MEDL might not lead to a schedulable system. The “degree of schedulability” can be improved by inserting instances of messages into the available places in the MEDL, thus minimizing the θ_m parameters. For example, in Figure 25.5a by inserting another instance of the message m_1 in the first round and m_2 in the second round leads to P_2 missing its deadline, while in Figure 25.5b inserting m_1 into the second round and m_2 into the first round leads to a schedulable system.

Our algorithm repeatedly adds a new instance of a message to the current MEDL in the hope that the cost function will be improved (lines 16–30). In order to decide an instance of which message should be added to the current MEDL, a simple heuristic is used. We identify the process P_i which has the most “critical” situation, meaning that the difference between its deadline and response time, $D_i - R_i$, is minimal compared with all other processes. The message to be added to the MEDL is the message $m = m_{P_i}$ received by the process P_i (lines 18–20). Message m will be placed into that round (*BestRound*) which corresponds to the smallest value of the cost function (lines 21–28). The algorithm stops if the cost function cannot be further improved by adding more messages to the MEDL.

The `OptimizeMM` algorithm is similar to `OptimizeSM`. The main difference is that in the MM approach several messages can be placed into a slot (which also decides its size), while in the SM approach there can be at most one message per slot. Also, in the case of MM, we have to

take additional care that the slots do not exceed the maximum allowed size for a slot.

The situation is simpler for the dynamic approaches, namely DM and DP, since we only have to decide on the slot sizes and, in the case of DP, on the packet size. For these two approaches, the placement of messages is dynamic and has no influence on the cost function. The `OptimizeDM` algorithm (see Figure 25.8) starts with the first slot $S_i = S_0$ of the TDMA round and tries to find that size ($BestSize_{S_i}$) which corresponds to the smallest *CostFunction* (lines 4–14 in Figure 25.8). This slot size has to be large enough ($S_i \geq MinSize_{S_i}$) to hold the largest message to be transmitted in this slot, and within bounds determined by the particular TTP controller implementation (e.g., from 2 bits up to $MaxSize = 32$ bytes). Once the size of the first slot has been determined, the algorithm continues in the same manner with the next slots (lines 7–12).

The `OptimizeDP` algorithm has also to determine the proper packet size. This is done by trying all the possible packet sizes given the particular TTP controller. For example, it can start from 2 bits and increment with the “smallest data unit” (typically 2 bits) up to 32 bytes. In the case of the `OptimizeDP` algorithm the slot size has to be determined as a multiple of the packet size and within certain bounds depending on the TTP controller.

25.5.2 Simulated Annealing Strategy

We have also developed an optimization procedure based on a simulated annealing (SA) strategy. The main characteristic of such a strategy is that it tries to find the global optimum by randomly selecting a new solution from the neighbors of the current solution. The new solution is accepted if it is an improved one. However, a worse solution can also be accepted with a certain probability that depends on the deterioration of the cost function and on a control parameter called temperature [Ree93].

In Figure 25.9 we give a short description of this algorithm. An essential component of the algorithm is the generation of a new solution x starting from the current one x^{now} (line 5 in Figure 25.9). The neighbours of the current solution x^{now} are obtained depending on the chosen message scheduling approach. For SM, x is obtained from x^{now} by inserting or removing a message in one of its corresponding slots. In the case of MM, we have to take additional care that the slots do not exceed the maximum allowed size (which depends on the controller implementation), as we can allocate several messages to a slot. For these two static approaches we also decide on the number of rounds in a cycle (e.g., 2, 4, 8, 16; limited by the size of the memory implementing the MEDL). In the case of DM, the neighboring solution is obtained by

```

SimulatedAnnealing
01  construct an initial TDMA round  $x^{now}$ 
02   $temperature = \text{initial temperature } TI$ 
03  repeat
04    for  $i = 1$  to  $temperature \text{ length } TL$ 
05      generate randomly a neighboring solution  $x'$  of  $x^{now}$ 
06       $delta = CostFunction(x') - CostFunction(x^{now})$ 
07      if  $delta < 0$  then  $x^{now} = x'$ 
08      else
09        generate  $q = \text{random}(0, 1)$ 
10        if  $q < e^{-delta / temperature}$  then  $x^{now} = x'$  end if
11      end if
12    end for
13     $temperature = \alpha * temperature$ 
14  until stopping criterion is met
15  return solution corresponding to the best  $CostFunction$ 
end SimulatedAnnealing

```

Figure 25.9: The Simulated Annealing Strategy

increasing or decreasing the slot size within the bounds allowed by the particular TTP controller implementation, while in the DP approach we also increase or decrease the packet size.

For the implementation of this algorithm, the parameters TI (initial temperature), TL (temperature length), α (cooling ratio), and the stopping criterion have to be determined. They define the so called cooling schedule and have a strong impact on the quality of the solutions and the CPU time consumed. We were interested to obtain values for TI , TL and α that will guarantee the finding of good quality solutions in a short time. In order to tune the parameters we have first performed very long and expensive runs on selected large examples and the best ever solution, for each example, has been considered as the near-optimum. Based on further experiments we have determined the parameters of the SA algorithm, for different sizes of examples, so that the optimization time is reduced as much as possible but the near-optimal result is still produced. These parameters have then been used for the large scale experiments presented in the following section. For example, for the graphs with 320 nodes, TI is 300, TL is 500 and α is 0.95. The algorithm stops if for three consecutive temperatures no new solution has been accepted.

25.6 Experimental Results

For evaluation of our approaches we first used sets of processes generated for experimental purpose. We considered architectures consisting of 2, 4, 6, 8 and 10 nodes. 40 processes were assigned to each node, resulting in sets of 80, 160, 240, 320 and 400 processes. 30 process sets were generated for each of the five dimensions. Thus, a total of 150 sets of processes were used for experimental evaluation. Worst case computation times, periods, deadlines, and message lengths were assigned randomly within certain intervals. For the communication channel we considered a transmission speed of 256 kbps. The maximum length of the data field in a slot was 32 bytes and the frequency of the TTP controller was chosen to be 20 MHz. All experiments were run on a Sun Ultra 10 workstation.

For each of the 150 generated examples and each of the four message scheduling approaches we have obtained the near-optimal values for the cost function (Equation 25.14) as produced by our SA based algorithm (see Section 25.5.2). For a given example these values might differ from one message passing approach to another, as they depend on the optimization parameters and the schedulability analysis which are particular for each approach. Figure 25.10 presents a comparison based on the average percentage deviation of the cost function obtained for each of the four approaches, from the minimal value among them. The percentage deviation is calculated according to the formula:

$$deviation = \frac{cost_{approach} - cost_{best}}{cost_{best}} \times 100. \quad (25.15)$$

The DP approach is, generally, able to achieve the highest degree of schedulability, which in Figure 25.10 translates in the smallest deviation. In the case the packet size is properly selected, by scheduling messages dynamically we are able to efficiently use the available space in the slots, and thus reduce the release jitter. However, by using the MM approach we can obtain almost the same result if the messages are carefully allocated to slots as does our optimization strategy.

Moreover, in the case of larger process sets, the static approaches suffer significantly less overhead than the dynamic approaches. In the SM and MM approaches the messages are uniquely identified by their position in the MEDL. However, for the dynamic approaches we have to somehow identify the dynamically transmitted messages and packets. Thus, for the DM approach we consider that each message has several identifier bits appended at the beginning of the message, while for the DP approach the identification bits are appended to each packet. Not only the identifier bits add to the overhead, but in the DP approach, the transfer and delivery processes (see Figure 25.3) have to be activated

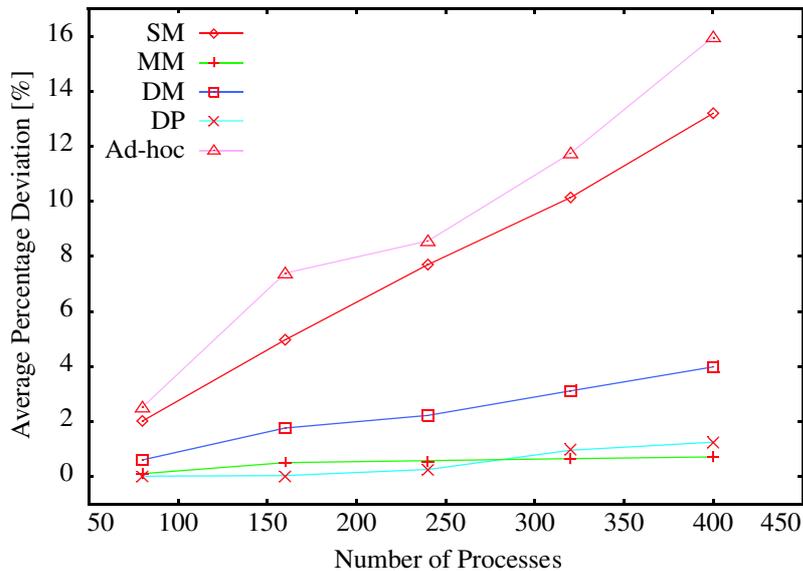


Figure 25.10: Comparison of the Four Approaches to Message Scheduling

at each sending and receiving of a packet, and thus interfere with the other processes. Thus, for larger applications (e.g. process sets of 400 processes), MM outperforms DP, as DP suffers from large overhead due to its dynamic nature. DM performs worse than DP because it does not split the messages into packets, and this results in a mismatch between the size of the messages dynamically queued and the slot size, leading to unused slot space that increases the jitter. SM performs the worst as it does not permit much room for improvement, leading to large amounts of unused slot space. Also, DP has produced a MEDL that resulted in schedulable process sets for 1.33 times more cases than the MM and DM. MM, in its turn, produced two times more schedulable results than the SM approach.

Together with the four approaches to message scheduling a so called ad-hoc approach is presented. The ad-hoc approach performs scheduling of messages without trying to optimize the access to the communication channel. The ad-hoc solutions are based on the MM approach and consider a design with the TDMA configuration consisting of a simple, straightforward allocation of messages to slots. The lengths of the slots were selected to accommodate the largest message sent from the respective node. Figure 25.10 shows that the ad-hoc alternative is constantly outperformed by any of the optimized solutions. This demonstrates that significant gains can be obtained by optimization of the parameters defining the access to the communication channel.

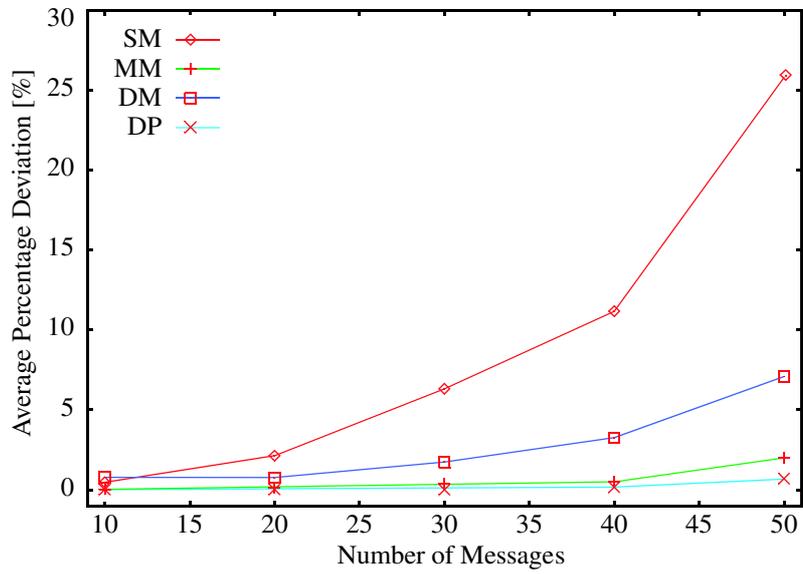


Figure 25.11: Four Approaches to Message Scheduling: The Influence of the Message Sizes

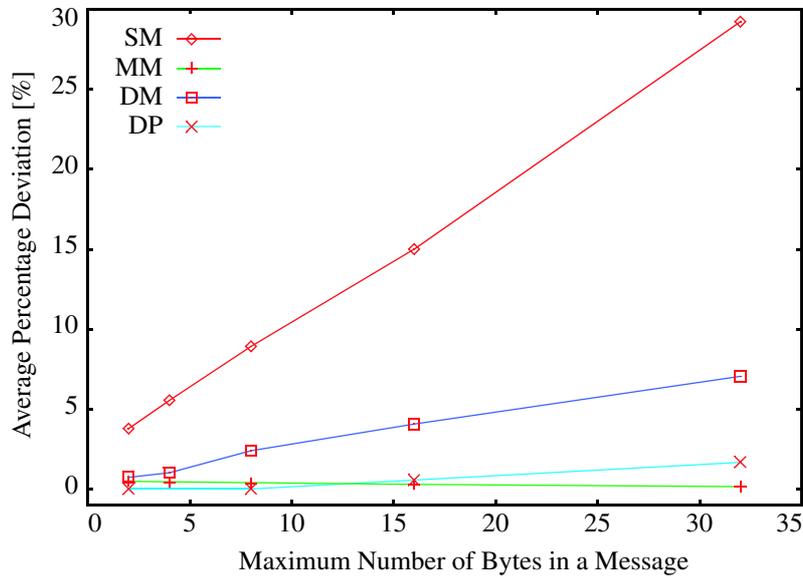


Figure 25.12: Four Approaches to Message Scheduling: The Influence of the Message Sizes

		80 procs.	160 procs.	240 procs.	320 procs.	400 procs.
SM	avg.	0.12%	0.19%	0.50%	1.06%	1.63%
	max.	0.81%	2.28%	8.31%	31.05%	18.00%
MM	avg.	0.05%	0.04%	0.08%	0.23%	0.36%
	max.	0.23%	0.55%	1.03%	8.15%	6.63%
DM	avg.	0.02%	0.03%	0.05%	0.06%	0.07%
	max.	0.05%	0.22%	0.81%	1.67%	1.01%
DP	avg.	0.01%	0.01%	0.05%	0.04%	0.03%
	max.	0.05%	0.13%	0.61%	1.42%	0.54%

Table 25.1: Percentage Deviations for the Greedy Heuristics compared to SA

Next, we have compared the four approaches with respect to the number of messages exchanged between different nodes and the maximum message size allowed. For the results depicted in Figures 25.11 and 25.12 we have assumed sets of 80 processes allocated to 4 nodes. Figure 25.11 shows that, as the number of messages increases, the difference between the approaches grows while the ranking among them remains the same. The same holds for the case when we increase the maximum allowed message size (Figure 25.12), with a notable exception: for large message sizes MM becomes better than DP, since DP suffers from the overhead due to its dynamic nature.

We were also interested in the quality of our greedy heuristics. Thus, we have run all the examples presented above, using the greedy heuristics and compared the results with those produced by the SA based algorithm. Table 25.1 shows the average and maximum percentage deviations of the cost function values produced by the greedy heuristics from those generated with SA, for each of the graph dimensions. All the four greedy heuristics perform very well, with less than 2% loss in quality compared to the results produced by the SA algorithms. The execution times for the greedy heuristics were more than two orders of magnitude smaller than those with SA. Although the greedy heuristics can potentially find solutions not found by SA, for our experiments, the extensive runs performed with SA have led to a design space exploration that has included all the solutions produced by the greedy heuristics.

The above comparison between the four message scheduling alternatives is mainly based on the issue of schedulability. However, when choosing among the different policies, several other parameters can be of importance. Thus, a static allocation of messages can be beneficial from the point of view of testing and debugging and has the advantage of simplicity. Similar considerations can lead to the decision not to split messages. In any case, however, optimization of the bus access scheme is highly desirable.

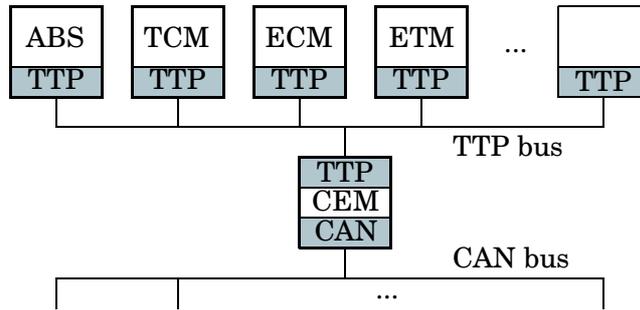


Figure 25.13: Hardware Architecture for the CC

25.6.1 Case Study

A typical safety critical application with hard real-time constraints, to be implemented on a TTP based architecture, is a vehicle cruise controller (CC). We have considered a CC system derived from a requirement specification provided by the industry. The CC described in this specification delivers the following functionality: it maintains a constant speed for speeds over 35 km/h and under 200 km/h, offers an interface (buttons) to increase or decrease the reference speed, and is able to resume its operation at the previous reference speed. The CC operation is suspended when the driver presses the brake pedal. The specification assumes that the CC will operate in an environment consisting of several nodes interconnected by a TTP channel (Figure 25.13). There are five nodes which functionally interact with the CC system: the Anti Blocking System (ABS), the Transmission Control Module (TCM), the Engine Control Module (ECM), the Electronic Throttle Module (ETM), and the Central Electronic Module (CEM). It has been decided to distribute the functionality (processes) of the CC over these five nodes. The transmission speed of the communication channel is 256 kbps and the frequency of the TTP controller was chosen to be 20 MHz. We have modeled the specification of the CC system using a set of 32 processes and 17 messages. All the experiments were run on a Sun Ultra 10 workstation. For the given application the ad-hoc approach was unable to produce a schedulable solution. Then, we ran the simulated annealing strategy in order to determine if the CC system is schedulable with each of the four approaches to message scheduling. The SM approach failed to produce a schedulable solution after 304 seconds optimization time. However, with the other three approaches schedulable solutions could be produced. The smallest cost function resulted for the DP approach (taking 730 seconds of optimization time), followed in this order by MM (1193 seconds optimization time) and DM (648 seconds optimization time).

	Percentage Deviation from SA	Execution Time (sec)
SM	0.4%	23.08
MM	0.8%	87.98
DM	0.5%	21.37
DP	0.3%	104.75

Table 25.2: Results for the Cruise Controller Example

Using the greedy strategies similar results have been obtained with significantly shorter optimization times. As with the SA-based algorithm, SM failed to produce a schedulable solution, while the other approaches succeeded. Table 25.2 presents the percentage deviations of the cost function obtained with the greedy algorithms from the values obtained with SA, and the execution times.

25.7 Conclusions

In this chapter we have considered hard-real time systems implemented on distributed architectures consisting of several nodes interconnected by a communication channel. Processes are scheduled using a preemptive policy with fixed priorities. The bus access is statically scheduled according to the time triggered protocol. We have presented an approach to schedulability analysis with special emphasis on the impact of the communication infrastructure and protocol on the overall system performance.

We have considered four different approaches to message scheduling over TTP, that were compared based on the issue of schedulability. It has been also shown how the parameters of the communication protocol can be optimized in order to fit the communication particularities of a certain application. By optimizing the bus access scheme significant improvements can be obtained with regard to the “degree of schedulability” of the system. Experimental results show that our optimization heuristics are able to efficiently produce good quality results.

Bibliography

- [ABD⁺95] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Journal of Real-Time Systems*, 8(2-3):173–198, March-May 1995.
- [BLMSV98] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-Vincentelli. Scheduling for embedded real-time sys-

- tems. *IEEE Design and Test of Computers*, pages 71–82, January–March 1998.
- [bWC98] X by Wire Consortium. X-by-wire: Safety related fault tolerant systems in vehicles. Technical report, 1998. <http://www.vmars.tuwien.ac.at/projects/xbywire/>.
- [CB95] P. Chou and G. Borriello. Interval scheduling: Fine-grained code scheduling for embedded systems. In *Proceedings of the 32nd ACM/IEEE conference on Design automation*, pages 462–467. ACM Press, 1995.
- [DIJ95] J.-M. Daveau, T. B. Ismail, and A. A. Jerraya. Synthesis of system-level communication by an allocation-based approach. In *Proceedings of the 8th international symposium on System Synthesis*, pages 150–155. ACM Press, 1995.
- [DJ98] B. P. Dave and N. K. Jha. Cohra: Hardware-software cosynthesis of hierarchical heterogeneous distributed systems. In *IEEE Transactions on CAD*, volume 17, pages 900–919, 1998.
- [DLJ97] B. P. Dave, G. Lakshminarayana, and N. K. Jha. Cosyn: Hardware-software co-synthesis of embedded systems. In *Proceedings of the 34th annual conference on Design Automation*, pages 703–708. ACM Press, 1997.
- [EDPP00] P. Eles, A. Doboli, P. Pop, and Z. Peng. Scheduling with bus access optimization for distributed embedded systems. *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems*, 8(5):472–491, October 2000.
- [EHS97] H. Ermedahl, H. Hansson, and M. Sjödin. Response-time guarantees in atm networks. In *Proceedings of the Real-Time Systems Symposium*, pages 274–284, 1997.
- [GH95] J. J. Gutiérrez García and M. González Harbour. Optimized priority assignment for tasks and messages in distributed hard real-time systems. pages 124–132, 1995.
- [GH98] J. C. Palencia Gutiérrez and M. González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real Time Systems Symposium*, pages 26–37, December 1998.
- [GH99] J. C. Palencia Gutiérrez and M. González Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proceedings of the*

- 20th IEEE Real-Time Systems Symposium*, page 328. IEEE Computer Society, 1999.
- [HD92] K. Hoyme and K. Driscoll. Safebus. *IEEE Aerospace and Electronic Systems Magazine*, 8(3):34–39, 1992.
- [KB03] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.
- [KM98] P. V. Knudsen and J. Madsen. Integrating communication protocol selection with partitioning in hardware/software codesign. In *Proceedings of the 11th international symposium on System synthesis*, pages 111–116. IEEE Computer Society, 1998.
- [Kop97] H. Kopetz. *Real-Time Systems*. Kluwer Academic Publishers, 1997.
- [LA99] H. Lönn and J. Axelsson. A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications. In *Proceedings of the 11th Euromicro Conference on Real-time Systems*, pages 142–149. IEEE Computer Society Press, Jun 1999.
- [Leh90] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of 11th IEEE Real-Time Symposium*, pages 201–209, 1990.
- [LL73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):47–61, January 1973.
- [LPW99] C. Lee, M. Potkonjak, and W. Wolf. Synthesis of hard real-time application specific systems. *Design Automation of Embedded Systems*, 4:215–242, 1999.
- [NG94] S. Narayan and D. Gajski. Synthesis of system-level bus interfaces. In *Proceedings of the European Design and Test Conference*, pages 395–399, 1994.
- [OB98] R. B. Ortega and G. Borriello. Communication synthesis for distributed embedded systems. In *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 437–444. ACM Press, 1998.
- [PEP99] P. Pop, P. Eles, and Z. Peng. Scheduling with optimized communication for time-triggered embedded systems. In *Proceedings of the 7th International Workshop on Hardware-Software Co-Design*, pages 178–182, May 1999.

- [PEP00] P. Pop, P. Eles, and Z. Peng. Schedulability analysis for systems with data and control dependencies. In *Proceedings of the Euromicro Conference on Real Time Systems*, pages 201–208, 2000.
- [PGH97] J. C. Palencia, J. J. Gutiérrez Garca, and M. González Harbour. On the schedulability analysis for distributed hard real-time systems. In *Proceedings of the Euromicro Conference on Real Time Systems*, pages 136–143, 1997.
- [Ree93] C. R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications, 1993.
- [Rus01] J. Rushby. Bus architectures for safety-critical embedded systems. *Springer-Verlag Lecture Notes in Computer Science*, 2211:306–323, 2001.
- [SR93] J. A. Stankovic and K. Ramamritham. IEEE Computer Society Press, 1993.
- [SRL90] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.
- [TBW92] K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: an np-hard problem made easy. *Real-Time Systems*, 4(2):145–165, 1992.
- [TC94] K. W. Tindell and J. Clark. Holistic schedulability analysis for distributed real-time systems. *Euromicro Journal on Microprocessing and Microprogramming (Special Issue on Parallel Embedded Real-Time Systems)*, 40:117–134, 1994.
- [THW94] K. W. Tindell, H. Hansson, and A. J. Wellings. Analysing real-time communications: Controller area network (can). In *Proceedings of the Real-Time Systems Symposium*, pages 259–263, 1994.
- [Tin94] K. W. Tindell. Adding time-offsets to schedulability analysis. Technical Report YCS 221, Department of Computer Science, University of York, January 1994.
- [YW98] T.-Y. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(11):1125–1136, 1998.

Chapter 26

Generating Real-Time Schedules using Constraint Programming

By **Cecilia Ekelin**

Department of Computer Engineering
Chalmers University of Technology
Email: cekelin@ce.chalmers.se

In real-time systems that contain a large variety of application constraints, such as embedded systems, the most common scheduling approach is to resolve potential conflicts off-line by generating a fixed time-table for the execution of the tasks. Unfortunately, the generation of such a schedule is an NP-complete problem which implies that the runtime complexity of an exact scheduling algorithm is exponential in the worst case. Moreover, the amount of different constraints that must be considered makes it hard to devise a reasonably efficient algorithm without sacrificing solution quality or model accuracy. In this work we show how a scheduling algorithm based on constraint programming is able to tackle these difficulties.

26.1 Introduction

Scheduling of a real-time systems means deciding when the different tasks in the system should execute such that the application constraints – deadlines in particular – always are met. If the application contains no other constraints besides deadlines the problem is simple enough to be solved sufficiently fast during runtime. However, if constraints concerning for example shared resources and task communication exist, it is not known how to select the next task to be scheduled (such that the problem is feasible) without investigating all possible choices. Since

this procedure is too time consuming to be done on-line (during runtime) it is typically done off-line (before the systems is started) and the resulting schedule is stored as a time-table. Naturally, this approach assumes that all tasks in the system are known beforehand. However, in for example embedded systems this is often not a limitation since the system behavior is highly application-specific which means that the operating environment is well-known. Advantages of the time-table approach include that arbitrary constraints can be handled and that the schedule can be optimized regarding some criteria. However, the runtime and implementation complexity of the scheduling algorithm may still be troublesome. Therefore, most proposed algorithms have to make trade-offs between the system model and the schedule quality. In this paper, we show how to devise a scheduling algorithm based on *constraint programming* that facilitates both accurate system modeling and easy implementation without sacrificing either speed or quality.

26.2 Related Work

Many different time-table based scheduling-algorithms have been proposed but the basic ideas they are built on can be divided into four categories:

- **Heuristic algorithms** Schedules are constructed using “rules-of-thumb” for how to hopefully satisfy the constraints. Typically the implementation complexity grows a lot with the amount of different constraints since it becomes harder to find heuristics that work well for all constraints. Therefore the system model has to be rather simple. Since heuristic algorithms do not consider all possibilities their runtime complexity is polynomial but in contrast they cannot guarantee feasibility. See for example [Ram95].
- **Local-search algorithms** To avoid relying on sophisticated heuristics, these algorithms operate by making small changes to an existing (initial) schedule. With a certain probability, an altered schedule is then selected as the next schedule to serve as a basis for the changes. Whenever the schedule is regarded as good enough or the runtime has reached a predefined value, the algorithm stops. An advantage of this approach is that the changes can be applied independently which makes the algorithm fairly easy to implement. However, the algorithm parameters require fine tuning to increase the probability that a feasible schedule is found. Common variants of these algorithms include simulated annealing (SA) and genetic algorithms (GA). See for example [TBW92, NS00].

- **Exhaustive-search algorithms** Since the addressed scheduling problem is NP-complete the only known method to guarantee feasibility is to consider all possible schedules. However, this does not mean that the schedules actually have to be generated since many can be determined as infeasible by using some (local) reasoning about the constraints. Thus, although the algorithm runtime is exponential in the worst case, it may be experienced as polynomial for typical problem instances. Variants include tailored branch-and-bound algorithms (B&B) as well as approaches based on general-purpose constraint programming (CP). See for example [Xu93, JS97] (B&B) and [SW00, SGK00] (CP).
- **Mathematic algorithms** Here the scheduling problem is formulated as an optimization problem using mathematical equations and solution methods. However, since the scheduling problem is discrete and also contain constraints that are hard to express as equations, this approach has limited applicability.

We believe that the constraint-programming technique is the most promising because i) it is based on a theoretical foundation ii) it allows optimality but also acknowledges heuristics iii) it supports arbitrary constraints. Despite these properties, the potential of constraint programming has not yet been discovered by the real-time community.

26.3 Constraint Programming

The problems addressed with the constraint programming technique are called *constraint satisfaction problems (CSP)*. Formally, a CSP is a triple $\langle Z, D, C \rangle$ where Z is a set of variables $\{z_1, \dots, z_n\}$, D a set of domains $\{d_1, \dots, d_n\}$ such that $z_i \in d_i$ and C a set of constraints over Z and D . A solution to a CSP is an assignment of values from the domains in D to the variables in Z such that the constraints in C are satisfied.

The general scheme for constraint propagation is as follows. Assume $Z = \{z_1, \dots, z_n\}$, $D = \{d_1, \dots, d_n\}$ and $C = \{c(z_i, z_j)\}$ where c is some constraint on z_i and z_j .¹ Fix a variable, for instance z_i . Then, for each $v_z \in d_i$, check if there exists some $v_y \in d_j$ such that the constraint c holds when $z_i = v_z$ and $z_j = v_y$. If no such v_y exists, delete v_z from d_i since clearly $z_i = v_z$ is not part of any solution. This procedure is repeated for all variables and constraints until a fix-point is reached (that is, the propagation is unable to reduce the domains further.) If some $d_i = \emptyset$ the problem is infeasible. Unfortunately, constraint propagation is not complete (the domains may still contain values that are not part of

¹The concept can be generalized to n -ary constraints as well.

a solution) which is why a subsequent search phase is required. Furthermore, constraint propagation is performed during search since selecting a value to a variable is equal to adding a constraint.

Since it is possible to express most CSP:s using general-purpose domains and constraints, for example, integers and arithmetics, several commercial constraint-systems are now available. In this work, we will use SICStus Prolog [Lab95] and its constraint solver for finite domains [COC97]. Using such a system, the major challenge for the constraint programmer is to figure out how to express the problem in variables and constraints, and come up with suitable heuristics for the assignment of values to variables. In fact, these two issues should be carefully addressed because they greatly influence the search performance.

26.4 Problem Description

The input for a task assignment and scheduling algorithm is a description of the real-time application \mathcal{T} , the available hardware architecture \mathcal{N} , constraints for how the application may use the hardware \mathcal{C} and design objectives that are to be optimized \mathcal{O} . We assume \mathcal{N} to represent a heterogeneous system consisting of m nodes η_1, \dots, η_m , which are connected via a single *communication bus*, and each node contains one *processor*. The system also has a number of globally shared *resources* ρ_r which can be accessed by at most one task at a time. The application \mathcal{T} includes n periodic tasks τ_1, \dots, τ_n , that execute on the processors and possibly communicate by *message passing*. The worst-case execution time of task τ_i on processor η_p is $c(i, p)$ and the size of a message sent from τ_i to τ_j is $message_size(i, j)$. Each periodic task is invoked at regular intervals of length $period(i)$ starting at time zero. We use τ_i^k to denote the k^{th} invocation of τ_i . Each task invocation must complete its execution within a time interval of length $deadline(i)$ where $deadline(i) \leq period(i)$. Task execution is non-preemptive and migration is not allowed. Due to the periodicity and phasing of the tasks it is sufficient to consider a schedule of length equal to the least-common multiple (*LCM*) of all task periods, referred to as the *lcp*. The schedule for each node is represented as a (cyclic) time-table containing the start times for the tasks assigned to that node. A similar time-table is also required for the messages being sent on the bus. The schedule must satisfy all constraints in \mathcal{C} and should be close to optimal regarding \mathcal{O} .

26.5 Real-Time Scheduling using Constraint Programming

Here we show how to model and solve the real-time assignment and scheduling problem as a CSP.

26.5.1 Constraint Model

Variables For our problem, there are three types of variables that represent initially unknown problem properties, namely, the start time, $start(\tau_i^k)$, of each task instance, the node, $node(\tau_i)$, to which each task is assigned, and the transmission start time, $start(\tau_i^k, \tau_j^l)$, of a message sent from task τ_i^k to τ_j^l . In addition, we use the following set of support variables: the (worst-case) execution time, $execution_time(\tau_i)$, of a task (which depends on the assigned node) and the actual communication delay of a message, $delay(\tau_i, \tau_j)$, (which differs for inter- and intra-node communication).

Since the tasks are periodic, the start time of each task is restricted to fall within certain intervals, that is,

$$start(\tau_i^k) \in [(k-1) \cdot period(i), k \cdot period(i)].$$

For the assignment we have $node(\tau_i) \in [1, m]$, while for the messages we use $start(\tau_i^k, \tau_j^l) \in [0, lcp]$.

Constraints The constraint solver that we use supports both simple arithmetic constraints as well as so-called symbolic constraints that are applicable for a wide range of problems. While many of the symbolic constraint constructs we use are specific to the SICStus framework, similar constructs are available in most other constraint systems. The constraints below are those that will be used in the algorithm evaluation. However, additional constraints can be modeled with, in most cases, minimum effort. This is both due to that arbitrary constraint constructs can be devised and that the modeling of the already expressed constraints is not affected.

Using the `element` constraint, the actual execution time of a task τ_i^k is expressed as `element(node(τ_i), [execution_time($i, 1$), ..., execution_time(i, m)], execution_time(τ_i))`. This construct is merely shorthand for $\forall p : node(\tau_i) = p \Rightarrow execution_time(\tau_i) = execution_time(i, p)$. Task deadlines impose $start(\tau_i^k) + execution_time(\tau_i) \leq deadline(i, k)$ where $deadline(i, k) = (k-1) \cdot period(i) + deadline(i)$.

A non-preemptive task execution can be modeled using the `disjoint2` constraint, which given a set of tuples (x_i, w_i, y_i, h_i) , representing rectangles with the bottom left corner in position (x_i, y_i) , width w_i and height h_i , ensures that these rectangles do not overlap. That is, we represent each task as a rectangle where the x -dimension corresponds to time and

the y -dimension to the nodes. This is possible since non-preemptive means that the executions of tasks on the same node must never overlap in time. Hence, we get `disjoint2` (`[(start(τ_i^k), execution_time(τ_i), node(τ_i), 1)]`).

For message transmission the model depends on the communication protocol used. Note that we will not know in advance which messages will actually be sent on the bus since this depends on the task assignment. If the communicating tasks are located on the same node, the message passing is instantaneous and does not involve the bus. Hence, the communication delay² $delay(\tau_i, \tau_j) = c_{speed} \cdot message_size(i, j) \cdot B_{i,j}$ where $B_{i,j} \Leftrightarrow node(\tau_i) \neq node(\tau_j)$.³ This means that some of the messages will not have to be considered by the communication protocol, which means that we do not have to find values for those $start(\tau_i^k, \tau_j^l)$ variables. In order to avoid that search, we impose the constraint $start(\tau_i^k, \tau_j^l) = B_{i,j} \cdot (start(\tau_i^k) + execution_time(\tau_i) + offset(\tau_i^k, \tau_j^l))$, where $offset(\tau_i^k, \tau_j^l)$ is a support variable that is constrained by $0 \leq offset(\tau_i^k, \tau_j^l) \leq start(\tau_j^l) - (start(\tau_i^k) + execution_time(\tau_i) - delay(\tau_i, \tau_j))$. This will make $start(\tau_i^k, \tau_j^l) = 0$ for intra-node communication which avoids a search for its value and also serves a purpose in the modeling of message transmissions as will be shown below. Sender tasks should precede receiver tasks with a distance that allows the message to arrive on time. To ensure this behavior the constraint $start(\tau_i^k) + execution_time(\tau_i) + delay(\tau_i, \tau_j) \leq start(\tau_j^l)$ is imposed.⁴ However, this constraint assumes that the only factor that affects the message transmission is the speed of the bus. In a more accurate model it is necessary to express the communication protocol in more detail. For instance, if contention-based message passing is used, the messages have to be scheduled (non-preemptively) on the bus, similar to the tasks. But unlike tasks, that will eventually execute, some message transmissions may not use the bus. Therefore, they cannot be modeled using the `disjoint2` constraint, since some of the corresponding rectangles $(start(\tau_i^k, \tau_j^l), delay(\tau_i, \tau_j), 0, 1)$ will be transformed into “lines” $(0, 0, 0, 1)$ which are not permitted by the `disjoint2` constraint. Instead, we use the constraint `serialized` (`[start(τ_i^k, τ_j^l), [delay(τ_i, τ_j)]]`), which expresses the same constraint but only in one dimension and using a different propagation algorithm.

As mentioned, the system may contain resources other than processors that can be used by the tasks. We assume that these resources are dynamic and global. That is, they are only required during the execu-

²Without loss of generality, we assume a normalized bus data rate, $c_{speed} = 1$.

³Hence, the purpose of $B_{i,j} \in \{0, 1\}$ is to represent the truth value of a constraint, a procedure which is known as *reification*.

⁴Note that when $delay(\tau_i, \tau_j) = 0$ this constraint is reduced to a precedence constraint.

tion of a task and have limited *capacities* which restrict the simultaneous resource usage. In the modeling of resources we use the **cumulative** constraint which, given a number of tasks, durations and resource usages, ensures that the combined resource usage of the tasks at no point in time exceeds the capacity of the resource. In our case this constraint can be used as **cumulative**([*start*(τ_i^k)], [*execution_time*(τ_i)], [*amount_used*(i, r)], *capacity*(r)).

If the system requires explicit locality constraints it must be ensured that the task is only assigned to permitted processors. That is, $node(\tau_i) \neq p$ if η_p is a forbidden node for task τ_i or $node(\tau_i) = p$ if τ_i should be pre-assigned to η_p .

Objectives We will consider two different optimization objectives in our experimental studies. The first objective is minimizing the *maximum jitter*, f_{jitter} , in the application. The jitter for a task instance τ_i^k is defined as,

$$jitter(\tau_i^k) = \begin{cases} |start(\tau_i^k) - start(\tau_i^{k-1}) - period(i)|, & \text{if } k > 1 \\ |start(\tau_i^k) + lcp - period(i) - start(\tau_i^{\frac{lcp}{period(i)}})|, & \text{if } k = 1 \end{cases}$$

Then $f_{jitter} \geq jitter(\tau_i^k)$. The second objective we will consider is the *maximum lateness*, $f_{lateness}$. Similar to jitter, lateness is defined using $lateness(\tau_i^k) = start(\tau_i^k) + execution_time(\tau_i) - deadline(i, k)$ where $f_{lateness} \geq lateness(\tau_i^k)$.

It should be mentioned that optimization is performed by repetitively solving the same problem but with increasing bounds on the objective function. That is, if f' is the value of the best solution found so far, the problem is constrained as $f < f'$.

26.5.2 Search

Search for a good assignment of values to variables involves three main activities. First, there must be suitable *guidance* heuristics. That is, information on which variable to assign next and what value to assign this variable. Second, there must be some reasoning to detect *inferiority*. That is, to recognize when none of the values available for a variable actually will be possible. Third, it must be possible to decide *optimality* for a solution. In short, these three activities control where to go, where not to go and when (where) to stop.

Guidance Guidance is also known as search heuristics. Within constraint programming a number of general-purpose heuristics exist that work well on a wide range of problems. However, even better results

are obtained if the heuristics are aware of the structure of the specific problem. Within the real-time community many such heuristics have been devised for the real-time assignment and scheduling problem. We have therefore combined and evaluated some promising heuristics, see [EJ01]. In short, it turns out that a good task-to-processor *assignment* heuristic is to assign a task to the same processor as the task it communicates the most with [Ram95]. The reasons are that the bus otherwise quickly becomes the bottleneck and a reduced transmission delay increases the slack for the task executions involved in the communication. For task *scheduling* the general heuristic “least smallest domain-value” is suitable, breaking ties using “smallest domain, most constrained”. It should also be mentioned that the task assignment takes place before the task scheduling since the node assignment determines more unknown problem properties – all according to the general rule that difficult variables should be assigned early on.

Inferiority Reasoning about inferior choices is already somewhat present in the CP approach due to the constraint propagation. However, the propagation is limited to the way the problem is modeled. For example, due to that variables are represented as integers, the domain values are subject to enumeration. That is, each value is understood to represent a different choice (solution). However, it may be the case that some values in reality represent the same choice as in “any value but x”. In our case, this problem manifests itself in the task assignment. That is, if a task is to be assigned a previously unused processor it does not matter exactly which processor that is selected (assuming that they are identical) since the outcome will be the same. However, this will not automatically be detected during search which means that already investigated choices will be made over and over again resulting in a large slowdown. The problem appears due to *symmetries* in the problem formulation and is well studied within the constraint community. Although a number of general techniques to handle symmetries have been proposed, the major question – how to identify what constitutes a symmetry in your specific problem – is still rather open. We have found that a technique for detecting and discarding symmetries in graph-coloring problems almost directly translates to the task assignment problem. By incorporating this symmetry exclusion mechanism into the search, it is possible to obtain a runtime behavior that is independent of the number of nodes in the system [EJ01].

Optimality Theoretically, the problem of finding a feasible static schedule is NP-complete. This implies that once a solution is found (guessed) feasibility can be checked in polynomial time by simply comparing the constraints with the proposed solution. In contrast, finding an optimal

schedule (regarding some objective) is NP-hard since, even if a feasible solution is proposed, we cannot check optimality in polynomial time. That is, we have to compare the given schedule with all other possibilities. However, if we somehow knew that the optimal schedule could not have an objective value less than f_{lb} we could potentially discard a large amount of schedules. In particular if $f_{lb} = f'$ we would not have to search any more schedules at all since obviously the one we have found is optimal. Clearly, a *lower bound* measure f_{lb} would be very useful. This kind of measure is typically obtained by solving a simplified version of the original problem which enables an algorithm with low computational complexity to be used. Despite their potential, these kinds of lower bounds have so far not been used in the context of real-time scheduling. Therefore, we have developed several such estimates within this area (e.g., [EJ02]) but also a general technique which we will demonstrate in this work.

26.6 Experimental Evaluation

To evaluate the performance of our CP approach we have applied the algorithm to synthetic (randomly generated) tasks sets. On the same tasks sets we have also applied existing implementations of SA and B&B. The measures used to compare the scheduling algorithms are the solution quality (number of feasible solutions and average objective value) together with the runtime behavior (average time to find feasible/near-optimal solutions). Due to that the implementations of SA and B&B support different application constraints and optimization objectives, we cannot directly compare SA and B&B. Therefore, the algorithm comparison is divided into two sections.

26.6.1 SA vs. CP

The SA algorithm we have used for our experiments is based on the framework presented in [NS00]. In our slightly modified version of that algorithm, the objective is to minimize the maximum jitter (f_{max_jitter}) and the algorithm is capable of handling global dynamic resources and precedence constraints. (Communication is modeled through precedence constraints and shared resources.) Scheduling is non-preemptive. It is possible to configure the algorithm to search only for feasible schedules by modifying the energy function which represents the quality measure of a schedule. It is assumed that all tasks are pre-assigned to the processors, meaning that the assignment phase is omitted. For the internal algorithm parameters we have used $\alpha = 0.9$ and $T_0 = \frac{2 \cdot \max\{period(i)\}}{\ln(\alpha^{-1})}$ as suggested in [NS00]. As reported in [NS00], the computational complexity of this algorithm is $O(n^4)$.

Parameter	Study			
	A	B	C	D
Period	{16, 32, 64, 128, 256}	{15, 30, 50, 180, 200}	{100, 150, 300}	{100, 150, 300}
Number of tasks	20	10	30	30
<i>Average number of task invocations</i>	112	322	59	58
Deadline/Period ratio	0.25-1	0.25-1	0.25-1	0.25-1
Execution times	1-5	4-8	5-10	10-20
<i>Average utilization</i>	0.617	0.508	0.735	0.707
<i>Number of infeasible task sets</i>	47	83	14	87
Number of processors	2	2	2	4
Number of tasks/Resource	2	2	2	4
Task resource-usage probability	0.4	0.4	0.4	0.8

Table 26.1: Configuration parameters for the SA task sets.

Experimental Setup We have conducted four studies, labeled A, B, C and D, which follow the layout used in [NS00]. The task sets used in the studies comprise periodic tasks which are pre-assigned to the processors. The system also contains shared resources that are used by the tasks. The main difference between the studies is the selection of the periods and thus the degree of utilization. Task sets in Study A have harmonic periods and a medium utilization. In contrast, Study B task sets have less harmonic periods resulting in a larger *lcp* and a lower utilization than Study A. Task sets in Study C and D have mixed periods and a higher utilization than A and B. In addition, Study D task sets contain a larger number of resource usage constraints. The configuration parameters for the generated task sets are shown in Table 26.1.

Due to the potentially very long runtimes of CP we abort an experiment after one hour and, in that case, report the best result found so far.

Experimental Results Table 26.2 shows the performance of SA and CP for finding a first feasible schedule. Table 26.3 shows similar data (for SA and CP) when performing optimization. Note that for CP the same algorithm configuration is used both to find a first solution and to find an optimal solution. In contrast, the energy function for SA will be different in the two cases since the algorithm needs to know whether to prioritize low jitter or schedulability. Hence, for SA the number of solved instances may differ depending on the objective. For this reason, and because SA cannot decide optimality, *optimal* for SA in Table 26.3 simply means the number of solved instances.

As can be seen clearly in the tables, CP outperforms SA not only in the number of solved instances but also in solution quality and experienced runtime. For example, in Study A, CP is able to find (and

Study	Result	Algorithm	
		SA	CP
A	solved	45/53	53/53
	value	47.5	9.4
	runtime (geo)	59 s	2 s
B	solved	16/17	17/17
	value	105.1	36.2
	runtime (geo)	6 min	43 s
C	solved	85/86	86/86
	value	73.4	50.2
	runtime (geo)	5 s	0.6 s
D	solved	12/13	13/13
	value	64.8	66.1
	runtime (geo)	44 s	0.6 s

Table 26.2: Experimental results for SA vs. CP (first found)

guarantee) an optimal solution to all feasible instances (100%) whereas SA only finds a solution to 64% of the instances. Furthermore, CP finds these optimal solutions more than ten times faster than SA finds a (feasible or infeasible) solution which clearly demonstrates the effectiveness of CP. The fact that the average value for CP sometimes is larger than for SA is due to that the average is not calculated on the same set of values since CP finds more solutions than SA. (There are single instances though where SA finds a better solution than CP.)

Note that for the runtimes, the geometric mean is used rather than the arithmetic. The reason is that, for the arithmetic mean to be useful the runtime distribution has to be normal. However, it is known that the runtime distribution of CP typically is heavy-tailed [GSCK00] since it depends more on the hardness of a particular instance rather than the problem size. Therefore, the geometric mean is a better representation of the expected runtime.

26.6.2 B&B vs. CP

The implementation of B&B that we have used is available in the tool GAST [Jon98] configured to generate schedules with the objective of minimizing the maximum lateness ($f_{max_lateness}$). GAST is capable of handling communication constraints and locality constraints and also offers the possibility to have the algorithm terminate as soon as a first feasible schedule is found. Scheduling is non-preemptive, non-migrative and contention-based communication is used. We have chosen the best algorithmic parameters for the addressed scheduling problem according to the recommendations in [Jon99]. To that end, the complexity of the

Study	Result	Algorithm	
		SA	CP
A	optimal	34/53	53/53
	value	2.2	0.094
	runtime (geo)	4 min	15 s
B	optimal	6/17	6/17
	value	7.4	10.8
	runtime (geo)	22 min	9 min
C	optimal	79/86	55/86
	value	3.5	1.4
	runtime (geo)	2 min	24 s
D	optimal	10/13	8/13
	value	10.2	14.6
	runtime (geo)	2 min	20 s

Table 26.3: Experimental results for SA vs. CP (optimization)

(dynamic) lower-bound function is $O(n^2)$. The implementation also uses symmetry exclusion similar to that of CP.

Experimental Setup We have conducted three different studies, labeled A, B and C. The task sets used in the studies comprise tasks with equal periods. The tasks are not pre-assigned to the processors and they communicate via message passing. The main difference between the studies is the ratio of number of processors to the amount of communication constraints. Study A task sets have a comparably large amount of processors and communication constraints. In Study B, these numbers are divided by two whereas, in Study C, only the number of processors is reduced. The configuration parameters for the generated task sets are shown in Table 26.4.

Again, for CP, an experiment as a maximum time limit of one hour. For B&B we use a similar limit but on the number of searched (partial) schedules (10^6).

Experimental Results Table 26.5 shows the performance of B&B and CP for the problem of finding a first feasible solution. Table 26.6 shows similar data for finding an optimal solution.

As can be seen in the tables, CP outperforms B&B in terms of runtime when an optimal solution is needed (Table 26.6). For example, in Study B, CP is able to find good solutions more than ten times faster than B&B on the average. In terms of schedule quality, CP is bit worse than B&B in finding good solutions but significantly better in deciding optimality. This interesting relationship can be explained by taking a closer look at the results from the experiments. It turns out that CP

Parameter	Study		
	A	B	C
Period	100	100	100
Number of tasks	15	15	15
Execution times	10–20	5–15	5–15
<i>Average utilization</i>	<i>0.562</i>	<i>0.756</i>	<i>0.746</i>
<i>Number of infeasible task sets</i>	<i>0</i>	<i>0</i>	<i>0</i>
Communication probability	0.1	0.05	0.1
<i>Average number of communication relations</i>	<i>10</i>	<i>6</i>	<i>11</i>
Message sizes	1–5	1–5	1–5
Number of processors	4	2	2

Table 26.4: Configuration parameters for the B&B task sets.

Study	Result	Algorithm	
		B&B	CP
A	solved	82/100	94/100
	value	-3	-11
	runtime (geo)	0.2 s	0.2 s
B	solved	100/100	100/100
	value	-5	-13
	runtime (geo)	0.1 s	0.1 s
C	solved	98/100	100/100
	value	-5	-9
	runtime (geo)	0.1 s	0.2 s

Table 26.5: Experimental results for B&B vs. CP (first found)

has a greater variability in the solution quality. That is, for some instances the rather simple heuristics employed, are not accurate enough to guide the search in the right direction. In contrast, B&B contains more elaborate heuristics. However, when CP ends up in the right place, its inferiority techniques are apparently stronger than in B&B. Having said this, it is somewhat surprising that CP finds better first solutions than B&B.

26.6.3 General Lower-Bound

So far we have not considered how lower-bounds can improve the ability of CP to decide optimality. Although lower-bound algorithms tend to be tailored for the optimization objective we have identified a general lower-bound strategy, based on constraint propagation, for a certain class of optimization objectives.

Study	Result	Algorithm	
		B&B	CP
A	optimal	20/100	63/100
	value	-34	-30
	runtime (geo)	2 min	6 s
B	optimal	0/100	1/100
	value	-24	-23
	runtime (geo)	14 min	2 s
C	optimal	0/100	47/100
	value	-25	-23
	runtime (geo)	15 min	21 s

Table 26.6: Experimental results for B&B vs. CP (optimization)

Strategy As mentioned, the objective function is merely modeled as a constrained variable. Hence, (after propagation) we have $f \in [f_{lb}, f_{ub}]$. However, as propagation is not complete, it is quite likely that for an optimal solution, $f^* > f_{lb}$. In order to get a tighter lower bound lb we can impose $f_{lb} \geq f$ and check whether the problem becomes infeasible. Unfortunately, the propagation of this constraint back to the variables constituting the objective function is likely to be rather weak, resulting in a poor improvement of the search performance. However, if the objective function $f = \max\{x_i\}$ where x_i is some variable, we can instead introduce the constraint directly on the x_i variables. In fact, all objective functions fit this description although it may only be one x_i variable to consider. Hence, our lower-bound strategy starts with $lb = f_{lb}$ and then imposes $\forall i : lb \geq x_i$. Now, this constraint may either be successfully added (in which case we have not gained anything) or it may render the problem infeasible. If the problem becomes infeasible after propagation we know that $f > lb$ and we repeat the procedure with $lb + 1$ and so on.

Evaluation We have now applied the general lower-bound algorithm in the experiments described previously. As expected, for the SA task sets very few improvements could be detected. The reason is that the corresponding x_i variable $jitter(\tau_i^k)$ depends on two problem variables ($start(\tau_i^k)$ and $start(\tau_i^{k-1})$) which makes the propagation weak. In contrast, $lateness(\tau_i^k)$ only depends on $start(\tau_i^k)$ which implies better propagation. This can also be seen in Table 26.7.

As can be seen in the tables, using the lower-bound significantly increases the number of instances where optimality can be verified (within the one hour time-limit), and sometimes at shorter runtimes. For instance, in Study B the fraction of optimal solutions increased from 1% to 56%. However, the quality of the solutions experience no improvements. This indicates that, as suspected, the optimal solutions are indeed *found*

Study	Result	Algorithm	
		CP	CP+lb
A	optimal	63/100	65/100
	value	-30	-30
	runtime (geo)	6 s	6 s
B	optimal	1/100	56/100
	value	-23	-23
	runtime (geo)	2 s	5 s
C	optimal	47/100	78/100
	value	-23	-23
	runtime (geo)	21 s	9 s

Table 26.7: Results for using the general lower-bound.

rather quickly but optimality *verification* is very time-consuming without a tight lower-bound.

26.7 Summary and Future Work

The purpose of this work is to show the potential of constraint programming for static real-time scheduling. The problem of devising an algorithm to generate such schedules is not only the computational complexity, but also the implementation complexity that comes with all the application constraints that should be handled. As demonstrated in this work, the constraint programming approach is able to produce good (near-optimal) schedules and in reasonable time when compared with other state-of-the-art algorithms. Furthermore, the underlying framework reduces the implementation efforts since constraints are modeled independently and previous knowledge on constraint propagation can be reused.

Future extensions of this work would draw even more on the strengths of the constraint propagation. A rather natural extension would be to include more degrees of freedom in the system model. Today, a popular such extension is to introduce energy awareness through voltage scaling. That is, a task is extended with a variable $speed(\tau_i^k)$ that express what speed level the task should execute on. This is under the assumption that a processor has l different speed levels with varying energy characteristics. The energy consumed by a task then depends on its speed level. (See [EJ03].)

Another kind of extension would be to record the domain reductions performed by the constraint propagation in order to get an understanding of how the application constraints affect the schedule generation. Using such a trace, the system designer may analyze the schedule to

find bottlenecks or other undesired effects of the application constraints that may require the system to be redesigned. (See [Eke04].)

Bibliography

- [COC97] M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In H. Glaser et al., editors, *Proc. of the Int'l Symposium on Programming Languages: Implementations, Logics, and Programs*, volume 1292 of *Lecture Notes in Computer Science*, pages 191–206, Southampton, UK, September 3–5, 1997. Springer Verlag.
- [EJ01] C. Ekelin and J. Jonsson. Evaluation of search heuristics for embedded system scheduling problems. In *Proc. of the International Conference on Principles and Practice of Constraint Programming*, pages 640–654, Paphos, Cyprus, November 16–December 1, 2001.
- [EJ02] C. Ekelin and J. Jonsson. A lower-bound algorithm for minimizing network communication in real-time systems. In *Proc. of the Int'l Conf. on Parallel Processing*, pages 343–351, Vancouver, Canada, August 18–21, 2002.
- [EJ03] C. Ekelin and J. Jonsson. Energy-aware static scheduling of heterogeneous embedded real-time systems. Tech. Rep. 03-13, Dept. of Computer Engineering, Chalmers University of Technology, S-412 96 Göteborg, Sweden, 2003.
- [Eke04] C. Ekelin. Glass-box scheduling. Tech. Rep. 04-XX, Dept. of Computer Engineering, Chalmers University of Technology, S-412 96 Göteborg, Sweden, 2004.
- [GSCK00] C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1-2):67–100, 2000.
- [Jon98] J. Jonsson. GAST: A flexible and extensible tool for evaluating multiprocessor assignment and scheduling techniques. In *Proc. of the Int'l Conf. on Parallel Processing*, pages 441–450, Minneapolis, Minnesota, August 10–14, 1998.
- [Jon99] J. Jonsson. Effective complexity reduction for optimal scheduling of distributed real-time applications. In *Proc. of the IEEE Int'l Conf. on Distributed Computing Systems*, pages 360–369, Austin, Texas, May 31 –June 5, 1999.

- [JS97] J. Jonsson and K. G. Shin. A parametrized branch-and-bound strategy for scheduling precedence-constrained tasks on a multiprocessor system. In *Proc. of the Int'l Conf. on Parallel Processing*, pages 158–165, Bloomingdale, Illinois, August 11–15, 1997.
- [Lab95] Intelligent Systems Laboratory. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, 1995.
- [NS00] M. D. Natale and J. A. Stankovic. Scheduling distributed real-time tasks with minimum jitter. *IEEE Trans. on Computers*, 49(4):303–316, April 2000.
- [Ram95] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Trans. on Parallel and Distributed Systems*, 6(4):412–420, April 1995.
- [SGK00] R. Szymanek, F. Gruian, and K. Kuchcinski. Digital systems design using constraint logic programming. In *Proc. of the Practical Application of Constraint Technology and Logic Programming*, Manchester, England, April 10–12, 2000.
- [SW00] K. Schild and J. Würtz. Scheduling of time-triggered real-time systems. *Constraints*, 5(4):335–357, October 2000.
- [TBW92] K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: An NP-hard problem made easy. *Real-Time Systems*, 4(2):145–165, June 1992.
- [Xu93] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. on Software Engineering*, 19(2):139–154, February 1993.

Chapter 27

Static-priority scheduling on multiprocessors

By **Björn Andersson**

Department of Computer Engineering

Chalmers University of Technology

Email: `ba@ce.chalmers.se`

This chapter deals with the problem of scheduling a set of tasks to meet deadlines on a computer with multiple processors. Static-priority scheduling is considered, that is, a task is assigned a priority number that never changes and at every moment the highest-priority tasks that request to be executed are selected for execution.

The performance metric used is the capacity that tasks can request without missing a deadline. It is shown that every static-priority algorithm can miss deadlines although close to 50% of the capacity is requested. The new algorithms in this chapter have the following performance. In periodic scheduling, the capacity that can be requested without missing a deadline is: 33% for migrative scheduling and 50% for non-migrative scheduling. In aperiodic scheduling, many performance metrics have been used in previous research. With the aperiodic model used in this chapter, the new algorithms in this chapter have the following performance. The capacity that can be requested without missing a deadline is: 50% for migrative scheduling and 31% for non-migrative scheduling.

27.1 Introduction

Static-priority scheduling is a widely-used to schedule a set of tasks to meet deadlines in computer systems. Unfortunately, scheduling theory of static-priority scheduling is currently only well-developed for uniprocessor system. This chapter deals with multiprocessors. In particular, this chapter answers the question:

How much of the capacity of a multiprocessor system can be requested without missing a deadline when static-priority preemptive scheduling is used?

Of course, the answer to this question depends on the scheduling algorithms used. With a poor scheduling algorithm, it can happen that close to 0% of the capacity is requested and a deadline is still missed. For this reason, this chapter aims to design scheduling algorithms that maximize the capacity that can be used without missing a deadline. We will discuss three aspects related to the capacity of static-priority scheduling on multiprocessors: scheduling algorithms, schedulability analysis and robustness.

Scheduling algorithms A scheduling algorithm uses *global scheduling* if a task is allowed to be executed on any processor even when it is resumed after having been preempted. In global static-priority scheduling, the way an algorithm can affect when a task is executed is to assign priorities. Hence, an important research question is to assign priorities so that all tasks meet their deadlines. And, if a task misses a deadline with this priority assignment, we want this to happen only because the capacity that was requested was large.

A multiprocessor scheduling algorithm uses the *partitioned method* if a task is assigned to a processor and the task is allowed to be executed only on that processor. In partitioned static-priority scheduling, two algorithms are needed in order to schedule tasks: (i) a task-to-processor assignment algorithm and (ii) an algorithm to assign priorities to a task and these priorities are only used locally on each processor. As we will see, assigning priorities to tasks running on a uniprocessor is straightforward. The main challenge in partitioned scheduling is thus to assign tasks to processors.

Schedulability analysis Whether tasks meet their deadlines or not, depends not only on how much capacity is requested; two different workloads may request the same capacity, but the arrival times of tasks are different and the individual execution times of tasks are different so that one workload meets all deadlines whereas the other workload does not. Schedulability analysis techniques that incorporate knowledge of arrival times and execution times can often be used to guarantee that deadlines are met even if the capacity that is requested is high. In addition, such a schedulability analysis is often helpful when proving that a scheduling algorithm meets all deadlines if less than a certain capacity is requested, regardless of task arrival times.

In static-priority scheduling, the scheduling of a task is unaffected by its lower priority tasks. Hence, the problem in schedulability analysis

is to compute how much a task can be delayed by higher priority tasks. Typically, we attempt to find not exactly how much a task is delayed by the execution of higher priority tasks but rather an upper bound on that delay. We do so because (i) if a task arrives at multiples times, for example periodically, then it may be delayed by different amounts at different times or (ii) the execution times are not known exactly but there is a known upper bound on them.

The approach to schedulability analysis taken in this chapter is based on computing the capacity that is requested and compare it to the minimum capacity that can be requested without missing a deadline. This kind of schedulability analysis has the drawback of being very pessimistic, in the sense that many workloads could actually meet their deadlines but our schedulability test cannot guarantee that at design time. However, our schedulability test offers the following advantages: (i) execution times do not necessarily have to be known, it may be possible to measure the capacity that is requested, and (ii) it is computationally efficient in that the number of steps required to give a yes/no answer is proportional to the number of tasks, even when a task arrives periodically.

Robustness Finding the greatest capacity that can be requested without missing a deadline is one mean to achieve robustness in that if less of the capacity is requested then deadlines are met and if execution times decrease or arrival periods increase then deadlines are still met. But there are scheduling algorithms such that if they are applied to some workloads then deadlines are met but changing the workload in an intuitively positive way, like decreasing the execution times, leads to a missed deadline. These workloads are called *anomalies* and naturally finding the existence of anomalies in a scheduling algorithm is interesting because they show that intuitively positive changes, like upgrading to a faster processor, can be dangerous.

27.1.1 Assumptions

Every scientific study is based on a model and that model has its own assumptions. There is often a trade-off between choosing a model which is on the one hand (i) expressive (to allow many applications to be used) and realistic (to describe something which is close to problems that designers in the industry face) and on the other hand a model which is (ii) simple enough to allow reasoning. In this chapter, we will make the following assumptions:

- A1. The deadlines are given as requirements to the scheduling algorithm, that is, the scheduling algorithm is not permitted to change the deadlines.

- A2. The characteristics of the workload (arrival times, periods and execution times) are given as requirements to the scheduling algorithm, that is, the scheduling algorithm is not permitted to change that.
- A3. If all tasks meet their deadlines then the scheduling algorithm has succeeded; if a task misses a deadline, even if it finishes just a little later than the deadline, then the scheduling algorithm has failed.
- A4. Tasks do not request any other resources than a processor.
- A5. The arrival times of tasks are independent, that is, the execution of one task does not affect the arrival of another task.
- A6. The execution time of a task is not a variable with an upper and lower bound. It is a constant, but different tasks may have different execution times. The execution or absence of execution of a task may of course affect the finishing time of other tasks, but it does not affect the execution time of other tasks.
- A7. Preemption is permitted at any time and has no associated overhead. When we speak of preemption, we mean that the execution of a task is suspended and its state is saved in such a way that the task can resume its execution in the same location in the program. For scheduling algorithms that allow task migration (that is, global scheduling), no overhead is associated with migration, even if a task resumes on another processor than the one on which it was preempted.
- A8. There are no faults in hardware or software.
- A9. The speed of a processor does not change and cannot be changed.
- A10. A task cannot execute on two or more processors simultaneously, and a processor cannot execute two or more tasks simultaneously.

27.1.2 Contributions in this chapter

The problem of finding how much of the capacity that tasks can request without missing a deadline is well studied. (See [And03] for a discussion.)

The main contribution of this chapter to the state of the art in static-priority preemptive multiprocessor scheduling is to present how much of the capacity that tasks can request without missing a deadline. The contributions are illustrated in Table 27.1.

- C1. It is shown that regardless of whether partitioned scheduling or global scheduling is used, and regardless of whether tasks arrive

		arrival pattern	
		periodic scheduling	aperiodic scheduling
preemptive	partitioned	0.41 \rightarrow 0.50	0.00 \rightarrow 0.31
	global scheduling	0.00 \rightarrow 0.33	0.00 \rightarrow 0.50

Table 27.1: The contributions of this chapter. The figures show, for state-of-the-art algorithms, the capacity that can be requested without missing a deadline. The figure to the left of the arrow is the capacity prior to the work in this chapter, whereas the figure to the right of the arrow is the capacity resulting from the work in this chapter.

periodically or aperiodically, there are workloads that request just a little over 50% of the capacity and yet it is impossible to design a static-priority scheduling algorithm to meet all deadlines.

- C2. For global periodic scheduling, it is shown how to design an algorithm that meets all deadlines if 33% or less of the capacity is requested. This result is significant because before this research was performed, the best algorithm in global static-priority scheduling could miss deadlines even when the fraction of the capacity that the workload requested approached 0% [DL78].
- C3. For partitioned periodic scheduling, it is shown how to design an algorithm that meets all deadlines if 50% or less of the capacity is requested. This result is significant because, as stated above, no static-priority scheduling algorithm can guarantee that a fraction of the capacity greater than 50% can be used without missing a deadline. The best partitioned static-priority scheduling algorithm could only guarantee that 41% could be requested without missing a deadline [OB98, LDG01].
- C4. For global aperiodic scheduling, it is shown how to design an algorithm that meets all deadlines if 50% or less of the capacity is requested. This result is significant because, as stated above, no static-priority scheduling algorithm can guarantee that a fraction of the capacity greater than 50% can be used without missing a deadline. Other work that uses our definition of capacity has focused on a more restricted type of priority-assignment schemes and they can guarantee that all deadlines are met if 35% of the capacity is requested [LL03].
- C5. For partitioned aperiodic scheduling, it is shown how to design an algorithm that meets all deadlines if 31% or less of the capacity is requested. There is no previous work that uses definition of capacity that is used in this chapter.

C6. It is shown that scheduling anomalies can happen in several previously known preemptive multiprocessor scheduling algorithms for global and partitioned scheduling. This chapter also presents a partitioned scheduling algorithm that is free from anomalies. Previously, anomalies were only known in non-preemptive scheduling [Gra69] and scheduling with preemption but restricted migration [HL94].

The concept of “capacity” is intentionally left undefined because a clear definition depends on the system model used — the system model is different in periodic and aperiodic scheduling. System models will therefore be given later in this chapter.

The remainder of this chapter is structured as follows. First, periodic scheduling is introduced with motivation, notation and system model. We will also see issues in the design of static-priority scheduling algorithms for periodic scheduling. After that follows three sections that list new algorithms and present their performance. Second, the presentation of aperiodic scheduling is analogous; Aperiodic scheduling is introduced and then two sections list new algorithms and their performance. In both periodic- and aperiodic scheduling, we deal with both global and partitioned scheduling. After the results on periodic- and aperiodic scheduling follows a chapter, Conclusions, that gives the implications of the results in this chapter.

We will omit some proofs and some other discussion that would interrupt the main thread of the chapter. See [And03] for details.

27.2 Introduction to periodic scheduling

27.2.1 Motivation

Many applications in feedback-control theory, signal processing and data acquisition require equidistant sampling, making the scheduling of periodic tasks especially interesting. Other applications, such as interactive computer graphics and tracking, do not necessarily require periodicity but do require that tasks execute “over and over again”; periodic scheduling is one way to achieve this as well. It would be desirable that a task could be scheduled so that it executed periodically. Some algorithms, for example pinwheel scheduling [BL98], can do this for restricted task sets, but unfortunately this problem is in general impossible to solve (see Appendix A in [And03]). For this reason we will focus on periodically *arriving* tasks. In such a system a task arrives (requests to execute), periodically, but its execution is only approximately periodic.

The remainder of this section is organized as follows. Section 27.2.2 states the system model that we will use. Issues in the design of uni-

and multiprocessor scheduling algorithms are discussed in Section 27.2.3. After this section follows the Sections 27.3–27.6 which present the main results: the design of scheduling algorithms, their capacities, and their robustness.

27.2.2 System model

The system model of the periodic scheduling problem that we study is well established in previous research. It is as follows:

We consider the problem of scheduling a task set $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$ of n independent¹, periodically-arriving real-time tasks on m identical processors. A task arrives periodically with a period of T_i . Each time a task arrives, a new *instance*² of the task is created. We denote the k^{th} instance of the task by $\tau_{i,k}$, where $k \in \mathbf{Z}^+$. A task is *runnable* at time t if an instance of τ_i has arrived but this instance has not yet been completed. Each instance has a constant execution time of C_i . Each task instance has a prescribed deadline, D_i time units after its arrival. If D_i is not written out, then it is assumed that $D_i = T_i$, that is the deadline is equal to the time of the next arrival of the task. The *response time* of an instance of a task τ_i is the time from its arrival to the time when it has completed C_i units of execution. The response time, R_i , of a task τ_i is the maximum response time of all instances of that task. The *interference* of an instance of a task τ_i is its response time minus its execution time. The interference, I_i , of a task τ_i is the maximum interference of all instances of that task.

The *utilization*, u_i , of a task τ_i is $u_i = C_i/T_i$, that is, the ratio of the task's execution time to its period. The utilization, U , of a task set is the sum of the utilizations of the tasks belonging to that task set, that is, $U = \sum_{i=1}^n C_i/T_i$. Since we consider scheduling on a multiprocessor system, the utilization is not always indicative of the load of the system. This is because the original definition of utilization is a property of the task set only and does not consider the number of processors. To also reflect the amount of processing capability available, we use the concept of *system utilization*, U_s , for a task set on m processors, which is the average utilization of each processor, that is, $U_s = U/m$. Note that utilization and system utilization describe how much the task set stresses the computer system without referring to any particular time or time interval. It can happen that the system utilization of a task set is less than 100% but there are still time intervals during which all processors are busy.

¹That is, the execution of one task does not affect the arrival of another task.

²In Part I, about periodic scheduling, we use the concepts: *job*, *instance* and *task instance* synonymously.

A task is *schedulable* with respect to an algorithm if all its instances complete no later than their deadlines when scheduled by that algorithm. A task set is schedulable if all its tasks are schedulable. The *utilization bound* of a scheduling algorithm is a figure such that if the system utilization is less than or equal to the utilization bound then all deadlines are met. With this definition, every scheduling algorithm has the utilization bound of 0%, so when we speak of the utilization bound of an algorithm we usually mean the greatest utilization bound that we are able to prove for an algorithm or the greatest utilization bound that is possible. A task set is *feasible* with respect to a class of algorithms if there is any algorithm in the class that can schedule the task set to meet all deadlines. When we say feasible without mentioning which class we mean, then it is understood that the class is: all scheduling algorithms that could possibly exist that satisfy the two very reasonable constraints that: (i) a task cannot execute on two or more processors simultaneously and (ii) a processor cannot execute on two or more tasks simultaneously.

A schedulability test is a condition which tells whether a task set meets its deadlines. A schedulability test with a condition such that if it is true then it implies that all deadlines are met is called a *sufficient* schedulability test. A schedulability test with a condition such that if all deadlines are met then the condition is true is called a *necessary* schedulability test. A schedulability test that is both sufficient and necessary is called *exact*.

In partitioned scheduling, the system behaves as follows. Each task is assigned to a processor and then assigned a local (for the processor) and static priority. With no loss of generality, we assume that the tasks on each processor are numbered in the order of decreasing priority, that is, τ_1 has the highest priority. On each processor, the task with the highest priority of those tasks that have arrived but not completed is executed, using preemption if necessary.

In global scheduling, the system behaves as follows. Each task is assigned a global, unique and static priority. With no loss of generality, we assume that the tasks in τ are numbered in the order of decreasing priority, that is, τ_1 has the highest priority. Of all tasks that have arrived but not completed, the m highest-priority tasks are executed, using preemption and migration if necessary³ in parallel on the m processors.

We assume that C_i and T_i are real numbers such that $0 < C_i \leq T_i$. Let S_i denote the time when τ_i arrives for the first time. We assume that S_i is part of the description of the scheduling problem — S_i cannot be chosen by the scheduling algorithm and S_i cannot be chosen by the designer. When S_i cannot be chosen by a designer there are two models,

³At each instant, the processor chosen for each of the m tasks is arbitrary. If less than m tasks should be executed simultaneously, some processors will be idle.

the *synchronous model*, where $\forall i : S_i = 0$ and the asynchronous model, where S_i is arbitrary. Unless otherwise stated, we use the most general model, the asynchronous task model. In the asynchronous model, the scheduling algorithm only uses T_i and C_i in its decisions on how to assign priorities — S_i are not used, and a task set is deemed schedulable only if it meets all deadlines for every choice of S_i .

27.2.3 Design issues in periodic scheduling

Uniprocessor scheduling

It is desirable that a scheduling algorithm causes deadline misses only when it is impossible to meet deadlines. Such an algorithm is said to be optimal⁴. Earliest-Deadline-First (EDF) [LL73] is one of these optimal scheduling algorithms for uniprocessor preemptive scheduling of periodic tasks⁵. EDF assigns priorities in the following way. At time t , let d_i denote the time of the deadline (in our model, the time of the next arrival) of task τ_i . The priority of task τ_i is computed as: $\text{prio}(\tau_i) = 1/d_i$. (Tasks with a high $\text{prio}(\tau_i)$ are chosen over those with a low $\text{prio}(\tau_i)$.) EDF will not be discussed further in the context of periodic scheduling because EDF is not a static-priority scheduling algorithm when it is used in periodic scheduling (the priority of different instances of the same task may be different). Unfortunately, no static-priority scheduling algorithm is optimal (see Example 2).

Example 2 Consider two tasks to be scheduled on one processor. The tasks have the following characteristics: $T_1 = 5, C_1 = 2$ and $T_2 = 7, C_2 = 3 + \epsilon$. It is assumed that $0 < \epsilon \ll 1$. If τ_1 is given the highest priority, then τ_2 misses a deadline (shown in Figure 27.1(a)). On the other hand, if τ_2 is given the highest priority, then τ_1 misses a deadline (shown in Figure 27.1(b)). Hence, no static-priority scheduling algorithm can meet all deadlines. However, note that EDF meets all deadlines (shown in Figure 27.1(c)). We can conclude that no static-priority scheduling algorithm is optimal.

This illustration assumed that the tasks arrived at the same time when they arrived for the first time, but this argument remains valid even when the first arrival of a task is arbitrary [LL73]. In this example, the utilization was $\frac{2}{5} + \frac{3+\epsilon}{7} \approx 0.83$ but a deadline was missed. It may appear strange that a deadline can be missed despite the fact that less than 100% of the capacity is requested. In this example, the reason is that, at some instants, a task with a deadline further away in the future is forced, due to static-priority scheduling, to receive the highest priority (this is illustrated at time $t = 5$ in Figure 27.1(a)). \square

⁴Some authors call it *universal*.

⁵It is optimal for many other models too.

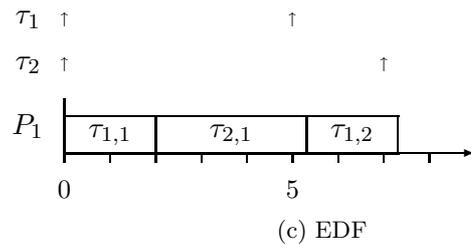
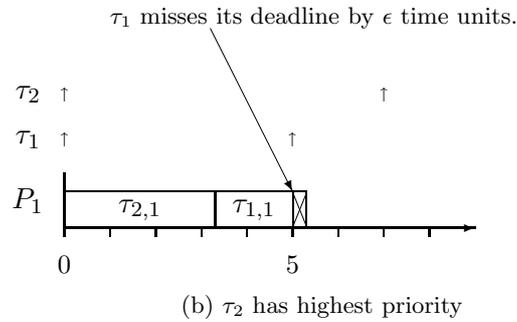
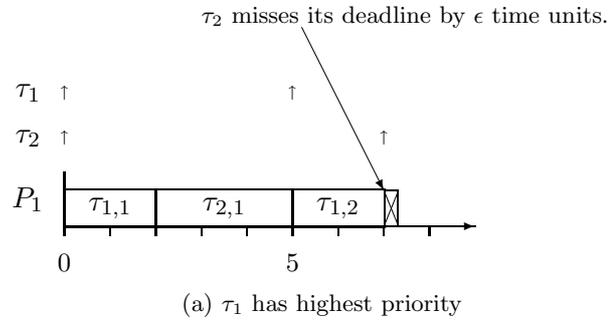


Figure 27.1: Static and dynamic priority scheduling on a uniprocessor. Arrows indicate the arrival times of tasks. With static-priority scheduling, only two priority assignments are possible in this example: τ_1 has the highest priority (shown in Figure 27.1(a)) or τ_2 has the highest priority (shown in Figure 27.1(b)). Either way, deadlines are missed. But with a dynamic priority scheduling algorithm, Earliest-Deadline-First (shown in Figure 27.1(c)), deadlines are met.

Although no static-priority scheduling algorithm is optimal it is still worth finding optimal static-priority assignment schemes. A static-priority scheme is optimal if a task misses a deadline only when there is no static-priority assignment scheme which can meet all deadlines. One optimal priority-assignment scheme is rate-monotonic (RM) [LL73]. It assigns a priority such that $prio(\tau_i) = 1/T_i$.

In schedulability analysis, it is interesting to find for each task the instant when its response time is maximized because, if the deadline of a task is met when it arrived at that instant, then all other deadlines of that task will be met as well. Such an instant is called a *critical instant*. For RM, we know that:

Theorem 13 ([LL73]) *One critical instant of a task scheduled by RM is when it arrives at the same time as its higher priority tasks.*

Based on this result, various schedulability conditions, too numerous to deal with here (see [Fid98] for an excellent survey), have been developed. The two most basic ones, the response-time analysis and the utilization-based test, are presented here.

The response-time analysis is a technique for computing the response times of tasks (see Theorem 14).

Theorem 14 ([JP86]) *If and only if $\forall i : R_i \leq T_i$ then all deadlines are met.*

The response time is the solution to the equation:

$$R_i = C_i + \sum_{j \in hp(i)} \lceil \frac{R_i}{T_j} \rceil \cdot C_j$$

Here, $hp(i)$ denotes the set of tasks that have a higher priority than τ_i . The equation can be solved iteratively, with the following procedure:

$$R_i^0 = 0$$

$$R_i^{k+1} = C_i + \sum_{j \in hp(i)} \lceil \frac{R_i^k}{T_j} \rceil \cdot C_j$$

When $R_i^{k+1} = R_i^k$, then $R_i = R_i^k$.

The utilization-based test is a technique that computes the utilization of a task set and compares it to the utilization bound (see Theorem 15).

Theorem 15 ([LL73]) *If RM is used and $\sum_{i=1}^n \frac{C_i}{T_i} \leq n \cdot (2^{1/n} - 1)$ then all deadlines are met.*

The response-time analysis is necessary and sufficient whereas the utilization-based test is sufficient but not necessary. However, the utilization-based test has lower computational complexity.

In certain models, RM is not optimal. For example, when tasks are given offsets [Goo03] (that is, the first time of arrival can be when all tasks do not necessarily arrive at the same time, and this first time can be chosen by the scheduling algorithm) or tasks are scheduled non-preemptively, or a task can be blocked (for example, waiting for a lower priority task that has locked a critical section). However, there is an optimal priority-assignment scheme for these models as well. This scheme, called Audsley's scheme [Aud91, ATB93], is based on the assumption that, although the question of whether a task meets its deadlines depends on its higher priority tasks; the relative priority orders within these higher priority tasks are unimportant. The main idea of Audsley's scheme is to iterate through all tasks and apply a schedulability test on each task asking the question: can this task be assigned the lowest priority? If the answer is yes, then one iterates through the remaining tasks and asks: can this task be assigned the second lowest priority, and so on.

RM and Audsley's priority assignment scheme have been extended to various models [KAS93, BTW95] but a discussion of this is beyond the scope of this chapter; our aim here is to understand the basic properties of static-priority scheduling on a uniprocessor so that we can design algorithms for multiprocessors.

The fact that a task cannot execute on two or more processors simultaneously poses a problem in multiprocessor scheduling. The approaches addressed in this chapter, partitioning and global scheduling, deal with this problem in different ways.

Partitioned scheduling

Recall that, in partitioned scheduling, a task is assigned to a processor and a task is not allowed to migrate. Once a task has been assigned to a processor, the constraint that a task cannot execute on two or more processors disappears. However, during task assignment, it is possible that the accumulated available processor capacity on all processors is large but no single processor has enough available capacity to execute the task.

A common solution to the task-assignment problem is to use a bin-packing algorithm [DL78]. Here, a task is first tentatively assigned to the processor with the lowest index, but if a schedulability test cannot guarantee that the task can be assigned there, then the task is tentatively assigned to the next processor with a higher index and so on. This has achieved a utilization bound of 0.41 [OB98, LGDG00].

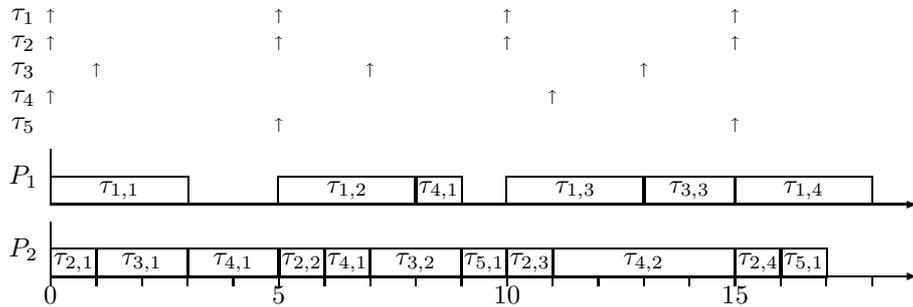


Figure 27.2: When tasks are not in-phase, unexpected task instances may contribute to the amount of time units that are executed. In this case, more than one instance of τ_4 will affect the execution of τ_5 despite that fact that the period of τ_4 is longer than that of τ_5 .

Global scheduling

Recall that, in global scheduling, a task is put in a queue of runnable tasks that is shared by all processors and, at every moment, the m highest priority runnable tasks are selected for execution on the m processors. Since global scheduling does not reduce the multiprocessor scheduling problem to many uniprocessor scheduling problems, as partitioned scheduling does, the fact that a task cannot execute on two or more processors gives rise to many interesting and unexpected effects that complicate the design of priority assignment algorithms and schedulability analysis techniques.

In schedulability analysis, one technique is to compute how many time units of execution higher priority tasks can perform during a time interval. In uniprocessor scheduling, it holds that during a time interval of length L , a task τ_i can execute for at most $\lceil L/T_i \rceil$ times and hence it can execute for at most $\lceil L/T_i \rceil \cdot C_i$ time units. However, in global multiprocessor scheduling, this number can be higher, as illustrated by Example 3.

Example 3 Consider the following five periodic tasks to be scheduled on two processors: $(T_1 = 5, C_1 = 3)$, $(T_2 = 5, C_2 = 1)$, $(T_3 = 6, C_3 = 2)$, $(T_4 = 11, C_4 = 4)$, $(T_5 = 10, C_5 = 2)$. Here, we assume that τ_3 arrives at time 1 and τ_5 arrives at time 5 (and all other tasks at time 0). For this particular case of task arrival times, the amount of execution from the four high-priority tasks in the interval $[5, 15)$ are: 6 for τ_1 , 2 for τ_2 , 4 for τ_3 , and 6 for τ_4 (see Figure 2.2). Task τ_5 is delayed by 9 time units due to the execution of higher priority tasks, which causes τ_5 to miss its deadline at time 15 (since $T_5 = 10$ and $C_5 = 2$). It is worth noting that more than one instance of τ_4 will contribute to the amount of

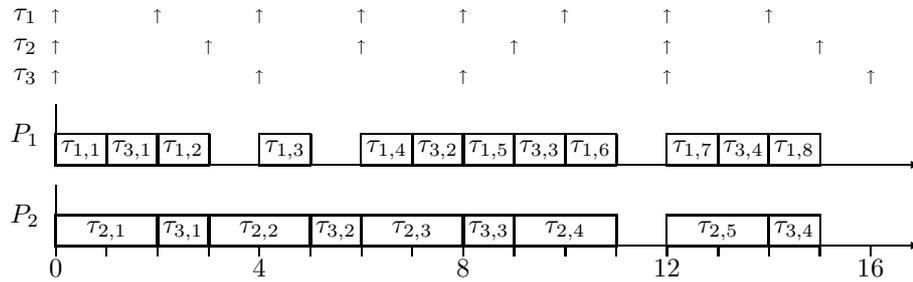


Figure 27.3: A critical instant does not always occur when a task arrives at the same time as all its higher-priority tasks. While the amount of execution from higher-priority tasks is equal for the first two instances of τ_3 , the delay is higher for the second instance despite the fact that tasks arrive at the same time for the first instance.

execution that delays τ_5 in the interval, despite that fact that the period of τ_4 is longer than that of τ_5 . \square

In global multiprocessor scheduling, it is not only the amount of execution of higher priority tasks that delays a lower priority task; the delay also depends on whether these higher priority tasks execute at the same time. This leads to additional phenomena for which assumptions that we were able to make in uniprocessor scheduling do not hold in global multiprocessor scheduling. The following observation (also reported by other researchers [LMM98a, Lun98]) describes one of these phenomena.

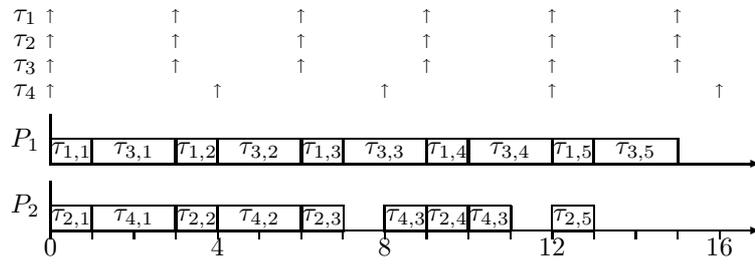
Observation 1 (Critical instant)

For static-priority preemptive global multiprocessor scheduling, there exist task sets where a critical instant of one of the tasks does not occur when it arrives at the same time as its higher-priority tasks.

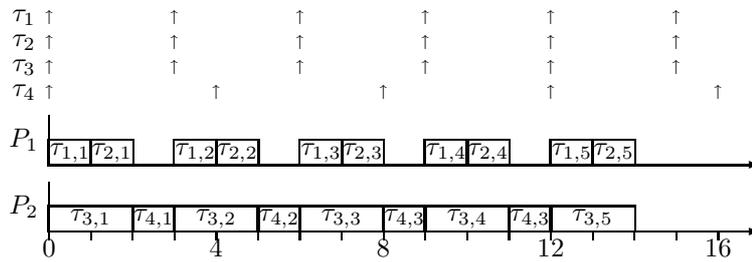
Example 4 Consider the following three periodic tasks: $(T_1 = 2, C_1 = 1)$, $(T_2 = 3, C_2 = 2)$, $(T_3 = 4, C_3 = 2)$. These tasks can be scheduled on two processors (see Figure 3). The first instance of τ_3 has a response time of $R_{3,1} = 3$ when it arrives at the same time as τ_1 and τ_2 . However, the second instance of τ_3 has a response time of $R_{3,2} = 4$ although τ_1 and τ_2 do not both arrive at the same time as τ_3 . \square

Example 3 and Observation 1 implies that the response-time calculation in Theorem 14 cannot be extended from uniprocessor scheduling to multiprocessor scheduling in a straightforward manner.

It is easy to show (as we will do in Chapter 27.3) that RM is not optimal in global multiprocessor scheduling. In fact, the utilization bound of RM is zero for global multiprocessor scheduling [Dha77, DL78], so it



(a) Task set schedulable



(b) Task set unschedulable

Figure 27.4: When the priority order of the higher-priority tasks τ_2 and τ_3 are swapped, the first instance of τ_4 becomes unschedulable (misses its deadlines by two time units). This is because τ_4 barely meets its deadline and the interference during the first task period increases (from 2 to 3).

is clear that better priority-assignment schemes should be sought. Based on our knowledge of priority assignment in uniprocessor scheduling, it may be tempting to use Audsley’s priority assignment scheme in global multiprocessor scheduling. Recall that Audsley’s priority assignment scheme assumes that the question of whether a task meets its deadline can be answered without knowing the relative priority orders among the higher priority tasks. Unfortunately, this assumption does not hold in global scheduling as shown by Observation 2.

Observation 2 (Dependence on the order of the higher-priority tasks)

For static-priority preemptive global multiprocessor scheduling, there exist task sets for which the response time of a task depends not only on the characteristics (that is, T_i and C_i) of its higher-priority tasks but also on the relative priority order of the higher-priority tasks.

The following example illustrates this phenomenon.

Example 5 Consider the following four periodic tasks: $(T_1 = 3, C_1 = 1)$, $(T_2 = 3, C_2 = 1)$, $(T_3 = 3, C_3 = 2)$, $(T_4 = 4, C_4 = 2)$. If priorities are assigned to these tasks according to RM (and τ_3 is given lower priority than both τ_1 and τ_2) and the first task instances arrive at the same time, the tasks can be scheduled on two processors (see Figure 27.4(a)). However, if we swap the priority order of τ_2 and τ_3 , task τ_4 misses a deadline (see Figure 27.4(b)). \square

Observation 2 implies that Audsley's priority-assignment scheme cannot be extended from uniprocessor scheduling to multiprocessor scheduling in a straightforward manner.

27.3 Global scheduling

27.3.1 Introduction

In global scheduling, the only way the scheduling algorithm can affect whether tasks meet their deadlines is to assign priorities. A natural choice is to use RM, but it unfortunately has a utilization bound of zero, as illustrated in Example 6.

Example 6 ([Dha77, DL78]) Consider $m+1$ periodic tasks that should be scheduled on m processors using RM. Let tasks τ_i (where $1 \leq i \leq m$) have $T_i = 1, C_i = 2\epsilon$, and the task τ_{m+1} have $T_{m+1} = 1 + \epsilon, C_{m+1} = 1$. All tasks arrive at the same time when they arrive for the first time; let us call this time $t = 0$. Tasks τ_i (where $1 \leq i \leq m$) will execute immediately when they arrive and complete their execution 2ϵ units later. τ_{m+1} then executes from time that is, $1 - \epsilon$ time units. τ_{m+1} needs to execute 1 time unit, however, so it misses its deadline. that is, $1 - 2\epsilon$ time units. τ_{m+1} needs to execute 1 time unit, however, so it misses its deadline. By letting $m \rightarrow \infty$ and $\epsilon \rightarrow 0$, we have a task set with a system utilization of zero, but a deadline is still missed. \square

Since RM can perform poorly in global scheduling, there is a need for a better priority assignment scheme — a priority assignment scheme with a utilization bound that is greater than zero. This section presents the RM-US approach that was invented for global scheduling to achieve a utilization bound greater than zero. We will do so by presenting the **RM-US(m/(3m-2))** scheme, the first published algorithm that used the RM-US approach.

Organization of this section. The remainder of this section is organized as follows. In Section 27.3.2, we briefly describe two major results that we will be using in the remainder of this section. In Section 27.3.3,

we present Algorithm **RM-US**($m/(3m-2)$), our static-priority multiprocessor algorithm for scheduling arbitrary periodic task systems, and prove that Algorithm **RM-US**($m/(3m-2)$) successfully schedules any periodic task system with utilization $\leq m^2/(3m-2)$ on m identical processors. Finally, Section 27.4 gives an upper bound on the utilization bound of priority-assignment schemes in global scheduling.

27.3.2 Results we will use

Some very interesting and important results in real-time multiprocessor scheduling theory were obtained in the mid 1990's. We will make use of two of these results in this section; these two results are briefly described below.

Resource augmentation. It has previously been shown [BKM⁺92, BKM⁺91, BHS94] that on-line real-time scheduling algorithms tend to perform extremely poorly under overloaded conditions. Phillips, Stein, Torng, and Wein [PSTW97] explored the use of *resource-augmentation* techniques for the on-line scheduling of real-time jobs⁶; the goal was to determine whether an on-line algorithm, if provided with faster processors than those available to a clairvoyant algorithm, could perform better than is implied by the bounds derived in [BKM⁺92, BKM⁺91, BHS94]. Although we are not studying on-line scheduling in this section — all the parameters of all the periodic tasks are assumed a priori known — it nevertheless turns out that a particular result from [PSTW97] will prove very useful to us in our study of static-priority multiprocessor scheduling. We present this result below.

The focus of [PSTW97] was the scheduling of individual jobs, and not periodic tasks. Accordingly, let us define a **job** $J_j = (r_j, e_j, d_j)$ as being characterized by an arrival time r_j , an execution requirement e_j , and a deadline d_j , with the interpretation that this job needs to execute for e_j units over the interval $[r_j, d_j)$. (Thus, the periodic task $\tau_i = (C_i, T_i, S_i)$ generates an infinite sequence of jobs with parameters $(S_k + k \cdot T_i, C_i, S_k + (k + 1) \cdot T_i)$, $k = 0, 1, 2, \dots$; in the remainder of this section, we will often use the symbol τ itself to denote the infinite set of jobs generated by the tasks in periodic task system τ .)

Let I denote any set of jobs. For any algorithm A and time instant $t \geq 0$, let $W(A, m, s, I, t)$ denote the amount of work done by algorithm A on jobs of I over the interval $[0, t)$, while executing on m processors of speed s each. A **work-conserving** scheduling algorithm is one that never idles a processor while there is some active job awaiting execution.

⁶Resource augmentation as a technique for improving the performance on on-line scheduling algorithms was formally proposed by Kalyanasundaram and Pruhs [KP95].

Theorem 1 (Phillips et al.) For any set of jobs I , any time-instant $t \geq 0$, any work-conserving algorithm A , and any algorithm A' , it is the case that

$$W(A, m, (2 - \frac{1}{m}) \cdot s, I, t) \geq W(A', m, s, I, t). \quad (27.1)$$

That is, an m -processor work-conserving algorithm completes at least as much execution as any other algorithm, if provided processors that are $(2 - 1/m)$ times as fast.

Predictable scheduling algorithms. Ha and Liu [HL94] have studied the issue of predictability in the multiprocessor scheduling of real-time systems from the following perspective.

Definition 1 (Predictability) Let A denote a scheduling algorithm, and $I = \{J_1, J_2, \dots, J_n\}$ any set of n jobs, $J_j = (r_j, e_j, d_j)$. Let f_j denote the time at which job J_j completes execution when I is scheduled by algorithm A .

Now, consider any set $I' = \{J'_1, J'_2, \dots, J'_n\}$ of n jobs obtained from I as follows. Job J'_j has an arrival time r_j , an execution requirement $e'_j \leq e_j$, and a deadline d_j (i.e., job J'_j has the same arrival time and deadline as J_j , and an execution requirement no larger than J_j 's). Let f'_j denote the time at which job J_j completes execution when I is scheduled using algorithm A . Scheduling algorithm A is said to be **predictable** if and only if for any set of jobs I and for any such I' obtained from I , it is the case that $f'_j \leq f_j$ for all j .

Informally, Definition 1 recognizes the fact that the specified execution-requirement parameters of jobs are typically only *upper bounds* on the actual execution-requirements during run-time, rather than the exact values. For a predictable scheduling algorithm, one may determine an upper bound on the completion-times of jobs by analyzing the situation under the assumption that each job executes for an amount equal to the upper bound on its execution requirement; it is guaranteed that the actual completion time of jobs will be no later than this determined value.

Since a periodic task system generates a set of jobs, Definition 1 may be extended in a straightforward manner to algorithms for scheduling periodic task systems: an algorithm for scheduling periodic task systems is predictable iff for any periodic task systems $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$ it is the case that the completion time of each job when every job of τ_i has an execution requirement exactly equal to C_i is an upper bound on the completion time of that job when every job of τ_i has an execution requirement of at most C_i , for all i , $1 \leq i \leq n$.

Ha and Liu define a scheduling algorithm to be **priority driven**⁷ if and only if it satisfies the condition that *for every pair of jobs J_i and J_j , if J_i has higher priority than J_j at some instant in time, then J_i always has higher priority than J_j* . Notice that any global static-priority algorithm for scheduling periodic tasks satisfies this condition, and is hence priority-driven. However, the converse is not true in that not all algorithms for scheduling periodic tasks that meet the definition of priority-driven are global static-priority algorithms (e.g., notice that the earliest deadline first scheduling algorithm, which schedules at each instant the currently active job whose deadline is the smallest, is a priority-driven algorithm, but is not a static-priority algorithm).

The result from the work of Ha and Liu [HL94] that we will be using can be stated as follows.

Theorem 2 (Ha and Liu) Any priority-driven scheduling algorithm is predictable.

27.3.3 Algorithm **RM-US(m/(3m-2))**

We now present Algorithm **RM-US(m/(3m-2))**, a static-priority global scheduling algorithm for scheduling periodic task systems, and derive a utilization-based sufficient schedulability condition for Algorithm **RM-US(m/(3m-2))**; in particular, we will prove that any task system τ satisfying $U(\tau) \leq m^2/(3m-2)$ will be scheduled to meet all deadlines on m unit-speed processors by Algorithm **RM-US(m/(3m-2))**. This is how we will proceed. In Section 27.3.3, we will consider a restricted category of periodic task systems, which we call “light” systems; we will prove that the multiprocessor **rate-monotonic** scheduling algorithm (we will henceforth refer to the multiprocessor rate-monotonic algorithm as Algorithm **RM**), which is a global static-priority algorithm that assigns tasks priorities in inverse proportion to their periods, will successfully schedule any light system. Then in Section 27.3.3, we extend the results concerning light systems to arbitrary systems of periodic tasks. We extend Algorithm **RM** to define a global static-priority scheduling algorithm which we call Algorithm **RM-US(m/(3m-2))**, and prove that Algorithm **RM-US(m/(3m-2))** successfully schedules any periodic task system with utilization at most $m^2/(3m-2)$ on m identical processors.

“Light” systems

Definition 2 A periodic task system τ is said to be a *light system on m processors* if it satisfies the following two properties

⁷The word “priority-driven” is synonymous to our word “job-static priority”.

Property P1: For each $\tau_i \in \tau$, $U_i \leq \frac{m}{3m-2}$

Property P2: $U(\tau) \leq \frac{m^2}{3m-2}$

We will consider the scheduling of task systems satisfying Property P1 and Property P2 above, using the rate-monotonic scheduling algorithm (Algorithm **RM**).

Theorem 3 Any periodic task system τ that is light on m processors will be scheduled to meet all deadlines on m processors by Algorithm **RM**.

Proof: Let us suppose that ties are broken by Algorithm **RM** such that τ_i has greater priority than τ_{i+1} for all i , $1 \leq i < n$. Notice that whether jobs of τ_k meet their deadlines under Algorithm **RM** depends only upon the jobs generated by the tasks $\{\tau_1, \tau_2, \dots, \tau_k\}$, and are completely unaffected by the presence of the tasks $\tau_{k+1}, \dots, \tau_n$. For $k = 1, 2, \dots, n$, let us define the task-set $\tau^{(k)}$ as follows:

$$\tau^{(k)} \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \dots, \tau_k\}.$$

Our proof strategy is as follows. We will prove that Algorithm **RM** will schedule $\tau^{(k)}$ in such a manner that all jobs of the lowest-priority task τ_k complete by their deadlines. Our claim that Algorithm **RM** successfully schedules τ would then follow by induction on k .

Lemma 3.1 Task system $\tau^{(k)}$ is feasible on m processors each of computing capacity $(\frac{m}{2m-1})$.

Proof: Since $m \geq 2$, notice that $3m-2 > 2m-1$. Since $U_i \leq \frac{m}{3m-2}$ for each task τ_i (by Property P1 above), it follows that

$$U_i \leq \frac{m}{2m-1} \tag{27.2}$$

Similarly from $U(\tau) \leq \frac{m^2}{3m-2}$ (Property P2 above) and $\tau^{(k)} \subseteq \tau$, it can be derived that

$$\sum_{\tau_i \in \tau^{(k)}} U_i \leq \frac{m^2}{2m-1}. \tag{27.3}$$

As a consequence of Inequalities 27.2 and 27.3 we may conclude that $\tau^{(k)}$ can be scheduled to meet all deadlines on m processors each of computing capacity $(\frac{m}{2m-1})$: the processor-sharing schedule (which we will henceforth denote **OPT**), which assigns a fraction U_i of a processor to τ_i at each time-instant bears witness to the feasibility of $\tau^{(k)}$.

End proof (of Lemma 3.1)

Since $\frac{m}{2m-1} \times (2 - \frac{1}{m}) = 1$, it follows from Theorem 1, the existence of the schedule OPT described in the proof of Lemma 3.1, and the fact that Algorithm **RM** is work-conserving, that

$$W(\mathbf{RM}, m, 1, \tau^{(k)}, t) \geq W(\text{OPT}, m, \frac{m}{2m-1}, \tau^{(k)}, t) \quad (27.4)$$

for all $t \geq 0$; i.e., *at any time-instant t , the amount of work done on $\tau^{(k)}$ by Algorithm **RM** executing on m unit-speed processors is at least as much as the amount of work done on $\tau^{(k)}$ by OPT on $m \frac{m}{2m-1}$ -speed processors.*

Lemma 3.2 All jobs of τ_k meet their deadlines when $\tau^{(k)}$ is scheduled using Algorithm **RM**.

Proof: Let us assume that the first $(\ell - 1)$ jobs of τ_k have met their deadlines under Algorithm **RM**; we will prove below that the ℓ 'th job of τ_k also meets its deadline. The correctness of Lemma 3.2 will then follow by induction on ℓ , starting with $\ell = 1$.

The ℓ 'th job of τ_k arrives at time-instant $S_k + (\ell - 1)T_k$, has a deadline at time-instant $S_k + \ell T_k$, and needs C_k units of execution. From Inequality 27.4 and the fact that the processor-sharing schedule OPT schedules each task τ_j for $U_j \cdot \max(0, S_k + (\ell - 1)T_k - S_j)$ units over the interval $[0, S_k + (\ell - 1)T_k)$, we have

$$W(\mathbf{RM}, m, 1, \tau^{(k)}, S_k + (\ell - 1)T_k) \geq \sum_{j=1}^k (U_j \cdot \max(0, S_k + (\ell - 1)T_k - S_j)) \quad (27.5)$$

Also, at least $\sum_{j=1}^{k-1} (U_j \cdot \max(0, S_k + (\ell - 1)T_k - S_j))$ units of this execution by Algorithm **RM** was of tasks $\tau_1, \tau_2, \dots, \tau_{k-1}$ — this follows from the fact that exactly $(\ell - 1)T_k U_k$ units of τ_k 's work has been generated prior to instant $S_k + (\ell - 1)T_k$; the remainder of the work executed by Algorithm **RM** must therefore be generated by $\tau_1, \tau_2, \dots, \tau_{k-1}$.

The cumulative execution requirement of all the jobs generated by the tasks $\tau_1, \tau_2, \dots, \tau_{k-1}$ that arrive prior to the deadline of τ_k 's ℓ 'th job is bounded from above by

$$\begin{aligned} & \sum_{j=1}^{k-1} \left\lceil \frac{\max(0, S_k + \ell T_k - S_j)}{T_j} \right\rceil C_j \\ & < \sum_{j=1}^{k-1} \frac{\max(0, S_k + \ell T_k - S_j)}{T_j} C_j + C_j \end{aligned} \quad (27.6)$$

As we have seen above (the discussion following Inequality 27.5) at least $\sum_{j=1}^{k-1} (U_j \cdot \max(0, S_k + (\ell - 1)T_k - S_j))$ of this gets done prior to time-instant $S_k + (\ell - 1)T_k$; hence, at most

$$\sum_{j=1}^{k-1} \left(U_j \cdot (\max(0, S_k + \ell T_k - S_j) - \max(0, S_k + (\ell - 1)T_k - S_j)) \right) + \sum_{j=1}^{k-1} C_j \quad (27.7)$$

remains to be executed *after* time-instant $S_k + (\ell - 1)T_k$. We will now show that $\max(0, S_k + \ell T_k - S_j) - \max(0, S_k + (\ell - 1)T_k - S_j) \leq T_k$ by considering the following cases:

1. $S_k + (\ell - 1)T_k - S_j < 0$

(a) $S_k + \ell T_k - S_j < 0$

Using the inequalities of this case yields:

$$\begin{aligned} \max(0, S_k + \ell T_k - S_j) - \max(0, S_k + (\ell - 1)T_k - S_j) \\ = 0 - 0 = 0 \end{aligned}$$

(b) $S_k + \ell T_k - S_j \geq 0$

Using the inequalities of this case yields:

$$\begin{aligned} \max(0, S_k + \ell T_k - S_j) - \max(0, S_k + (\ell - 1)T_k - S_j) \\ = S_k + \ell T_k - S_j - 0 \\ = T_k + S_k + (\ell - 1)T_k - S_j \\ < T_k \end{aligned}$$

2. $S_k + (\ell - 1)T_k - S_j \geq 0$

(a) $S_k + \ell T_k - S_j < 0$

This case cannot happen, because $T_k > 0$.

(b) $S_k + \ell T_k - S_j \geq 0$

Using the inequalities of this case yields:

$$\begin{aligned} \max(0, S_k + \ell T_k - S_j) - \\ \max(0, S_k + (\ell - 1)T_k - S_j) \\ = (S_k + \ell T_k - S_j) - (S_k + (\ell - 1)T_k - S_j) \\ = T_k \end{aligned}$$

Applying $\max(0, S_k + \ell T_k - S_j) - \max(0, S_k + (\ell - 1)T_k - S_j) \leq T_k$ on Inequality 27.7 yields that at most

$$\left(T_k \cdot \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j \right) \quad (27.8)$$

remains to be executed *after* time-instant $S_k + (\ell - 1)T_k$.

The amount of processor capacity left unused by $\tau_1, \dots, \tau_{k-1}$ during the interval $[S_k + (\ell - 1)T_k, S_k + \ell T_k)$ is therefore no smaller than

$$m \cdot T_k - \left(T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j \right) \quad (27.9)$$

Since there are m processors available, the cumulative length of the intervals over $[S_k + (\ell - 1)T_k, S_k + \ell T_k)$ during which $\tau_1, \dots, \tau_{k-1}$ leave at least one processor idle is minimized if the different processors tend to idle simultaneously (in parallel); hence, a lower bound on this cumulative length of the intervals over $[S_k + (\ell - 1)T_k, S_k + \ell T_k)$ during which $\tau_1, \dots, \tau_{k-1}$ leave at least one processor idle is given by $(m \cdot T_k - (T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j)) / m$, which equals

$$T_k - \frac{1}{m} \left(T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j \right) \quad (27.10)$$

For the ℓ 'th job of τ_k to meet its deadline, it suffices that this cumulative interval length be at least as large as τ_k 's execution requirement; i.e.,

$$\begin{aligned} T_k - \frac{1}{m} \left(T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j \right) &\geq C_k \\ \equiv \frac{C_k}{T_k} + \frac{1}{m} \left(\sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} \frac{C_j}{T_k} \right) &\leq 1 \\ \Leftarrow &\quad (\text{Since } T_k \geq T_j \text{ for } j < k) \\ U_k + \frac{1}{m} \left(2 \sum_{j=1}^{k-1} U_j \right) &\leq 1 \end{aligned} \quad (27.11)$$

Let us now simplify the lhs of Inequality 27.11 above:

$$\begin{aligned}
& U_k + \frac{1}{m} \left(2 \sum_{j=1}^{k-1} U_j \right) \\
& \leq U_k + \frac{1}{m} \left(2 \sum_{j=1}^k U_j - 2U_k \right) \\
& \leq \text{(By Property P2 of task system } \tau \text{)} \\
& \quad U_k \left(1 - \frac{2}{m} \right) + \frac{2m}{3m-2} \\
& \leq \text{(By Property P1 of task system } \tau \text{)} \\
& \quad \frac{m}{3m-2} \left(1 - \frac{2}{m} \right) + \frac{2m}{3m-2} \tag{27.12} \\
& = 1 \tag{27.13}
\end{aligned}$$

From Inequalities 27.11 and 27.13, we may conclude that the ℓ 'th job of τ_k does meet its deadline.

End proof (of Lemma 3.2)

The correctness of Theorem 3 follows from Lemma 3.2 by induction on k , with $k = m$ being the base case (that $\tau_1, \tau_2, \dots, \tau_m$ meet all their deadlines directly follows from the fact that there are m processors available in the system).

End proof (of Theorem 3)

Arbitrary systems

In Section 27.3.3, we saw that Algorithm **RM** successfully schedules any periodic task system τ with utilization $U(\tau) \leq m^2/(3m-2)$ on m identical processors, *provided each $\tau_i \in \tau$ has a utilization $U_i \leq m/(3m-2)$* . We now relax the restriction on the utilization of each individual task; rather, we permit any $U_i \leq 1$ for each $\tau_i \in \tau$. That is, we will consider in this section the static-priority global scheduling of any task system τ satisfying the condition

$$U(\tau) \leq \frac{m^2}{3m-2} .$$

For such task systems, we define the static priority-assignment scheme Algorithm **RM-US(m/(3m-2))** as follows.

Algorithm RM-US(m/(3m-2)) assigns (static) priorities to tasks in τ according to the following rule:

if $U_i > \frac{m}{3m-2}$ **then** τ_i has the highest priority (ties broken arbitrarily)

if $U_i \leq \frac{m}{3m-2}$ then τ_i has rate-monotonic priority.

Example 1 As an example of the priorities assigned by Algorithm **RM-US(m/(3m-2))**, consider a task system

$$\begin{aligned} \tau \stackrel{\text{def}}{=} & \{ \tau_1 = (T_1 = 7, C_1 = 1), \tau_2 = (T_2 = 10, C_2 = 2), \\ & \tau_3 = (T_3 = 20, C_3 = 9), \\ & \tau_4 = (T_4 = 22, C_4 = 11), \tau_5 = (T_5 = 25, C_5 = 2) \} \end{aligned}$$

to be scheduled on a platform of 3 identical unit-speed processors. The utilizations of these five tasks are ≈ 0.143 , 0.2, 0.45, 0.5, and 0.08 respectively. For $m = 3$, $m/(3m - 2)$ equals $3/7 \approx 0.4286$; hence, tasks τ_3 and τ_4 will be assigned highest priorities, and the remaining three tasks will be assigned rate-monotonic priorities. The possible priority assignments are therefore as follows (highest-priority task listed first):

$$\tau_3, \tau_4, \tau_1, \tau_2, \tau_5$$

or

$$\tau_4, \tau_3, \tau_1, \tau_2, \tau_5$$

□

Theorem 4 Any periodic task system τ with utilization $U(\tau) \leq m^2/(3m-2)$ will be scheduled to meet all deadlines on m unit-speed processors by Algorithm **RM-US(m/(3m-2))**.

Proof: Assume that the tasks in τ are indexed according to the priorities assigned to them by Algorithm **RM-US(m/(3m-2))**. First, observe that since $U(\tau) \leq m^2/(3m-2)$, while each task τ_i that is assigned highest priority has U_i strictly greater than $m/(3m-2)$, there can be at most $(m-1)$ such tasks that are assigned highest priority. Let k_o denote the number of tasks that are assigned the highest priority; i.e., $\tau_1, \tau_2, \dots, \tau_{k_o}$ each have utilization greater than $m/(3m-2)$, and $\tau_{k_o+1}, \dots, \tau_n$ are assigned priorities rate-monotonically. Let $m_o \stackrel{\text{def}}{=} m - k_o$.

Let us first analyze the task system $\hat{\tau}$, consisting of the tasks in τ each having utilization $\leq m/(3m-2)$:

$$\hat{\tau} \stackrel{\text{def}}{=} \tau \setminus \tau^{(k_o)} .$$

The utilization of $\hat{\tau}$ can be bounded from above as follows:

$$\begin{aligned}
U(\hat{\tau}) &= U(\tau) - U(\tau^{(k_o)}) \\
&< \frac{m^2}{3m-2} - k_o \cdot \frac{m}{3m-2} \\
&= \frac{m(m-k_o)}{3m-2} \\
&\leq \frac{(m-k_o) \cdot (m-k_o)}{3(m-k_o)-2} \\
&= \frac{m_o^2}{3m_o-2}
\end{aligned} \tag{27.14}$$

Furthermore, for each $\tau_i \in \hat{\tau}$, we have

$$U_i \leq \frac{m}{3m-2} \leq \frac{m_o}{3m_o-2} . \tag{27.15}$$

From Inequalities 27.14 and 27.15, we conclude that $\hat{\tau}$ is a periodic task system that is light on m_o processors. Hence by Theorem 3, $\hat{\tau}$ can be scheduled by Algorithm **RM** to meet all deadlines on m_o processors.

Now, consider the task system $\tilde{\tau}$ obtained from τ by replacing each task $\tau_i \in \tau$ that has a utilization U_i greater than $m/(3m-2)$ by a task with the same period, but with utilization equal to one:

$$\tilde{\tau} \stackrel{\text{def}}{=} \hat{\tau} \cup \left(\cup_{(C_i, T_i, S_i) \in \tau^{(k_o)}} \{(T_i, T_i, S_i)\} \right) .$$

Notice that Algorithm **RM-US(m/(3m-2))** will assign identical priorities to corresponding tasks in τ and $\hat{\tau}$ (where the notion of “corresponding” is defined in the obvious manner). Also notice that when scheduling $\tilde{\tau}$, Algorithm **RM-US(m/(3m-2))** will devote k_o processors exclusively to the k_o tasks in $\tau^{(k_o)}$ (these are the highest-priority tasks, and each have a utilization equal to unity) and will be executing Algorithm **RM** on the remaining tasks (the tasks in $\hat{\tau}$) upon the remaining $m_o = (m - k_o)$ processors. As we have seen above, Algorithm **RM** schedules the tasks in $\hat{\tau}$ to meet all deadlines; hence, Algorithm **RM-US(m/(3m-2))** schedules $\tilde{\tau}$ to meet all deadlines of all jobs.

Finally, notice that an execution of Algorithm **RM-US(m/(3m-2))** on task system τ can be considered to be an instantiation of a run of Algorithm **RM-US(m/(3m-2))** on task system $\tilde{\tau}$, in which some jobs — the ones generated by tasks in $\tau^{(k_o)}$ — do not execute to their full execution requirement. By the result of Ha and Liu (Theorem 2), it follows that Algorithm **RM-US(m/(3m-2))** is a *predictable* scheduling algorithm, and hence each job of each task during the execution of Algorithm **RM-US(m/(3m-2))** on task system τ completes no later than the corresponding job during the execution of Algorithm **RM-US(m/(3m-2))** on task system $\tilde{\tau}$. And, we have already seen above

that no deadlines are missed during the execution of Algorithm **RM-US(m/(3m-2))** on task system $\tilde{\tau}$.

End proof (of Theorem 4)

27.4 Bound on utilization bounds

We can show an upper bound on the best possible system utilization bound for any static-priority multiprocessor scheduling algorithm. Consider the task set

$$\tau \stackrel{\text{def}}{=} \{\tau_1 = (L, 2L - 1), \tau_2 = (L, 2L - 1), \dots, \\ \tau_m = (L, 2L - 1), \tau_{m+1} = (L, 2L - 1)\}$$

to be scheduled on m processors (L is a positive integer) when all tasks arrive at time 0. For this task set, the system utilization is $L/(2L - 1) + (L/(2L - 1))/m$. For global static-priority scheduling, deadlines will be missed for this task set because all m highest priority tasks will execute at the same time and occupy L time units during $[0, 2L - 1)$. There will be $L - 1$ time units available for a lower priority tasks, but the lowest priority task needs L time units and thus misses its deadline. By letting $L \rightarrow \infty$ and $m \rightarrow \infty$, the task set is unschedulable at a system utilization of $1/2$. Consequently, *the utilization guarantee bound for any global static-priority multiprocessor scheduling algorithm cannot be higher than $1/2$ of the capacity of the multiprocessor platform.*

This bound of 0.5 applies to all global static-priority algorithms; it applies even to very complex algorithms such as algorithms that enumerate all possible orders of priorities of tasks. However, if we consider simpler algorithms, algorithms (such as **RM-US(m/(3m-2))**) that assign a priority to a task based on only information of that task — not other tasks, then the utilization bound that we can achieve is even lower. This is illustrated by Theorem 16.

Theorem 16 *If the priorities of global traditional static-priority scheduling are assigned according to the function $\text{prio}(\tau_i) = f(T_i, C_i)$ and if the function $f(T_i, C_i)$ is scale invariant, that is $f(T_i, C_i) < f(T_j, C_j) \Leftrightarrow f(A * T_i, A * C_i) < f(A * T_j, A * C_j) \forall A > 0$, then the utilization bound is no greater than $\sqrt{2} - 1$.*

Proof: See [And03]. □

We can conclude that although our algorithm **RM-US(m/(3m-2))** only achieved a utilization bound of 0.33, it is not too far from what can be achieved given that it only takes a limited amount of information into account in its decisions.

27.5 Partitioned scheduling

27.5.1 Introduction

Before this research was performed, the partitioned method was well explored [Dha77, DL78, DD85, DD86, BLOS95, OS95a, OS95b, LMM98b, SVC98, LW82, OB98] and it was known that no partitioned scheduling algorithm can have a utilization bound greater than 0.50 but the best utilization bound known so far was 41% [OB98, LDG01], leaving room for improvements.

In this section, we show that an algorithm, called R-BOUND-MP-NFR, has a utilization bound of 50%. We hence close the problem.

The remainder of this section is organized as follows. Section 27.5.2 gives a background on partitioned scheduling. We propose an algorithm and prove its utilization bound, first in Section 27.5.3 where the periods of tasks are restricted and later in Section 27.5.4 where the periods of tasks are not restricted.

27.5.2 Background on partitioned scheduling

Recall that the partitioned method divides tasks into partitions, each having its own dedicated processor. Unfortunately, the problem of deciding whether a schedulable partition exists is NP-complete [LW82]. Therefore many heuristics for partitioning have been proposed, a majority of which are versions of the bin-packing algorithm⁸. These bin-packing algorithms rely on a schedulability test in order to know whether a task can be assigned to a processor or not. This reduces our problem from partitioning a set of tasks to meet deadlines into the problem of partitioning a set of tasks such that, on every processor, the schedulability test can guarantee that all tasks on that processor meet their deadlines. As a schedulability test, a natural choice is to use the knowledge that: if $\sum_{i=1}^{n_p} C_i/T_i \leq n_p \cdot (2^{1/n_p} - 1)$ and rate-monotonic is used to schedule tasks on processor p then all deadlines are met. (We let n_p denote the number of tasks assigned to processor p .) This schedulability test is often used, but as shown in Example 7 below, this bound is not tight enough to allow us to design a multiprocessor scheduling algorithm with a utilization bound of 50%.

Example 7 Consider $m + 1$ tasks with $T_i = 1$ and $C_i = \sqrt{2} - 1 + \epsilon$ to be scheduled on m processors. For this system, there must be a

⁸The bin-packing algorithm works as follows: (1) sort the tasks according to some criterion; (2) select the first task and an arbitrary processor; (3) attempt to assign the selected task to the selected processor by applying a schedulability test for the processor; (4) if the schedulability test fails, select the next available processor; if it succeeds, select the next task; (5) goto step 3.

processor p which is assigned two tasks. On that processor the utilization is $\sum_{i=1}^{n_p} C_i/T_i = 2 \cdot (\sqrt{2} - 1 + \epsilon)$ which is greater than $2 \cdot (\sqrt{2} - 1)$. Hence, there is no way to partition tasks so that all tasks can be guaranteed by this schedulability test to meet deadlines. We can do this reasoning for every m and every ϵ . By letting $\epsilon \rightarrow 0$ and $m \rightarrow \infty$ we can see that the utilization bound for algorithms that are based on this schedulability test cannot be greater than $\sqrt{2} - 1$, which is approximately 41%. \square

Note that the task set in Example 7 could actually be guaranteed by a necessary and sufficient schedulability test to meet deadlines (provided that ϵ is not too large). It is known that if all tasks are harmonic⁹ then the uniprocessor utilization bound is 100%¹⁰, and then the task set in Example 7 could be assigned with two tasks on one processor. A uniprocessor schedulability test that could exploit this information could allow a multiprocessor scheduling algorithm to achieve a utilization bound of 50%. This is what we will do in the following.

R-BOUND [LMM98b] is a uniprocessor schedulability test which exploits harmonicity. Let r_p denote the fraction between the maximum and the minimum period among the tasks assigned to processor p . If we restrict our attention to the case in which $\forall p : 1 \leq r_p < 2$ (we will relax this restriction later), we have the following theorem.

Theorem 17 (Lauzac, Melhem and Mossé[LMM98b]) *Let $B(r_p, n_p) = n_p(r_p^{1/n_p} - 1) + 2/r_p - 1$. If $\sum_{i=1}^{n_p} C_i/T_i \leq B(r_p, n_p)$ and rate-monotonic is used to schedule tasks on processor p then all deadlines are met.*

R-BOUND-MP is a previously known multiprocessor scheduling algorithm that exploits R-BOUND [LMM98b]. R-BOUND-MP combined R-BOUND with a first-fit bin-packing algorithm. To show which utilization bound a partitioned scheduling algorithm can achieve, we will design two derivatives of R-BOUND-MP. First, we will consider an algorithm R-BOUND-MP-NFRNS (R-BOUND-MP with next-fit-ring noscaling) and prove its utilization bound when $1 \leq \frac{\max_{\tau_i \in \tau} T_i}{\min_{\tau_i \in \tau} T_i} < 2$. (τ denotes the set of all n tasks.) Then we will consider the algorithm R-BOUND-MP-NFR (R-BOUND-MP with next-fit-ring) and prove its utilization bound when periods are not restricted.

⁹In a harmonic task set, the periods T_i and T_j of any two tasks τ_i and τ_j are related as follows: either T_i is an integer multiple of T_j , or T_j is an integer multiple of T_i .

¹⁰This is easy to see by dropping the ceiling in the equations/inequalities in exact schedulability tests [JP86, LSD89].

27.5.3 Restricted periods

In this section, we assume that $1 \leq \frac{\max_{\tau_i \in \tau} T_i}{\min_{\tau_i \in \tau} T_i} < 2$ holds. Clearly it means that no matter how we assign tasks to processors, it holds that $\forall p : 1 \leq r_p < 2$ and hence Theorem 17 can be used. We will use the algorithm R-BOUND-MP-NFRNS. It works as follows: (i) sort tasks in ascending order of periods, that is, the task with the shortest period is considered first, (ii) use Theorem 17 as a schedulability test on each uniprocessor, (iii) assign tasks with the next-fit bin-packing algorithm and (iv) when a task cannot be assigned to processor m , try to assign it to processor 1, if this does not work then declare FAILURE. If the algorithm terminates and has partitioned the whole task set then the algorithm declares SUCCESS.

Example 8 illustrates the workings of our algorithm R-BOUND-MP-NFRNS.

Example 8 Consider 4 tasks with $\{(T_1 = 1.1, C_1 = 0.935), (T_2 = 1.3, C_2 = 0.26), (T_3 = 1.2, C_3 = 0.084), (T_4 = 1, C_4 = 0.1)\}$ to be scheduled on 2 processors using R-BOUND-MP-NFRNS. The algorithm sorts the tasks in ascending order of periods. Reordering the tasks yields: $\{(T_4 = 1, C_4 = 0.1), (T_1 = 1.1, C_1 = 0.935), (T_3 = 1.2, C_3 = 0.084), (T_2 = 1.3, C_2 = 0.26)\}$. We can compute the utilizations of tasks: $u_4 = 0.1, u_1 = 0.85, u_3 = 0.07$ and $u_2 = 0.2$.

The current processor is processor 1. Tasks are now assigned in order. τ_4 is assigned to processor 1. Then an attempt is made to assign τ_1 to processor 1, but it fails because the $T_1/T_4 = 1.1$, and $n_1 = 2$ gives a utilization bound of 0.915 for these two tasks, and the sum of utilization of these two tasks is 0.95. Hence τ_1 is assigned to processor 2.

Now, processor 2 is the current processor. An attempt is made to assign τ_3 processor 2, and it succeeds because $T_3/T_1 = 1.2/1.1 = 1.09$, and $n_2 = 2$ gives a utilization bound of 0.922 for these two tasks, and the sum of utilization of these two tasks is 0.92.

Processor 2 is still the current processor. An attempt is made to assign τ_2 to processor 2, but it fails because $\max(T_1, T_3, T_2)/\min(T_1, T_3, T_2) = 1.3/1.1 = 1.18$ and $n_2 = 3$ gives a utilization bound of 0.86 for these three tasks, and the sum of utilization of these three tasks is 1.12. Since processor 2 is the last processor and τ_2 failed, we make an attempt to assign τ_2 to the first processor, that is, processor 1. This succeeds because $T_2/T_4 = 1.3/1 = 1.3$ and $n_1 = 2$ gives a utilization bound of 0.818 for these two tasks, and the sum of utilization of these two tasks is 0.3. Hence τ_2 is assigned to processor 1. \square

Theorem 18 (Utilization bound of R-BOUND-MP-NFRNS) If R-BOUND-MP-NFRNS is used and $T_1 \leq T_2 \leq \dots \leq T_n$ and $T_n/T_1 < 2$

Algorithm 15 Scale Task Set.

Input: A task set τ . **Output:** Another task set τ' .

- 1: $q = T_1$
- 2: for each $i \in \tau$
- 3: $q = \max(q, T_i)$
- 4: end for
- 5: for each $i \in \tau$
- 6: $T_i' = T_i \cdot 2^{\log_2(q/T_i)}$
- 7: $C_i' = C_i \cdot 2^{\log_2(q/T_i)}$
- 8: end for
- 9: sort tasks in τ' in increasing period
- 10: return τ'

and $\frac{1}{m} \sum_{i=1}^n u_i \leq 1/2$, then R-BOUND-MP-NFRNS will find a partitioning (declare SUCCESS).

Proof: See [And03]. □

27.5.4 Not restricted periods

In this section, we will see that if task periods are not restricted as they were in the previous section, Section 27.5.3, then it is possible to scale the periods and execution times of all tasks such that the restriction holds. This is meaningful because we will use a theorem which claims that, if the scaled task set meets all deadlines, then the task set which is not scaled also meets its deadlines.

Consider two task sets, τ and τ' . τ is not restricted. τ' is computed from τ according to Algorithm 15. Note that Algorithm 15 does not change the utilization of tasks. In addition we know that:

Theorem 19 (Lauzac, Melhem and Mossé[LMM98b]) *Given a task set τ , let τ' be the task set resulting from the application of the algorithm Scale Task Set to τ . If τ' is schedulable on one processor using rate-monotonic scheduling, then τ is schedulable on one processor with rate-monotonic scheduling.*

Now let R-BOUND-MP-NFR (R-BOUND-MP with next-fit-ring) be an algorithm which works as follows. First, each task in τ is transformed according to Algorithm 15 into τ' and the tasks in τ' are then assigned according to R-BOUND-MP-NFRNS. We can see that every task in τ has a corresponding task in τ' , so τ_i is assigned to the processor where τ_i' is.

We are now ready to state our utilization bound of R-BOUND-MP-NFR when tasks are not restricted.

Theorem 20 (Utilization bound of R-BOUND-MP-NFR) *If R-BOUND-MP-NFR is used and $\sum_{i=1}^n u_i \leq m/2$, then R-BOUND-MP-NFR will find a partitioning (declare SUCCESS).*

Proof: The proof is by contradiction. Suppose that the theorem was false. Then there would exist a task set τ with $\sum_{i=1}^n u_i \leq m/2$ which failed. The first thing that R-BOUND-MP-NFR does is to scale¹¹ the task set, so a scaled task set τ' will also declare failure when scheduled by R-BOUND-MP-NFRNS. Since u_i of a task does not change when it is scaled, we have that τ' (which failed) has $\sum_{i=1}^n u_i \leq m/2$. But this is impossible according to Theorem 18. \square

27.6 Anomalies

27.6.1 Introduction

Analysis techniques for real-time systems often require exact knowledge of task characteristics, but this is usually not available, for example: the execution time of a task depends on input data (which is unknown) or the arrival time of a task depends on when an external event occurs (which is unknown). Fortunately, upper and lower bounds are often known, so in order to give guarantees that deadlines are met, an often-used approach is to make assumptions. For example: (i) assume that a task meets its deadline if it did so when all tasks executed at their maximum execution time or (ii) assume that a task meets its deadline if it did so when all tasks arrived at their maximum arrival frequency. Situations where these assumptions do not hold are referred to as scheduling anomalies, and their existence jeopardizes timeliness or complicates the design process.

Anomalies neither occur in popular preemptive uniprocessor scheduling algorithms, such as rate-monotonic (RM) and earliest-deadline-first (EDF) [LL73, Mok00], nor in multiprocessor systems with the partitioned scheduling and each processor using an anomaly-free uniprocessor scheduling algorithm. Anomalies can occur in both uniprocessor and multiprocessor systems [GL89, Mok00, SRS95, LS99, Gra69, HL94] due to non-preemptive scheduling or due to restricted task migration because decreasing the execution time of a task changes the schedule and that can constrain future scheduling choices. However, in preemptive global scheduling and in partitioned scheduling where tasks are repartitioned when the task set changes, it is not known whether scheduling anomalies exist.

In this section, we study execution-time and period anomalies in preemptive multiprocessor scheduling algorithms. Our objective is to

¹¹This does not change u_i .

find anomalies and avoid them without introducing too much additional pessimism in the analysis.

The remainder of this section is organized as follows. Section 27.6.2 discusses what a scheduling anomaly is. Section 27.6.3 shows examples of anomalies in preemptive multiprocessor scheduling. Section 27.6.4 discusses strategies for avoiding anomalies. Section 27.6.5 describes a new algorithm that does not suffer from anomalies and Section 27.6.6 generalizes this algorithm.

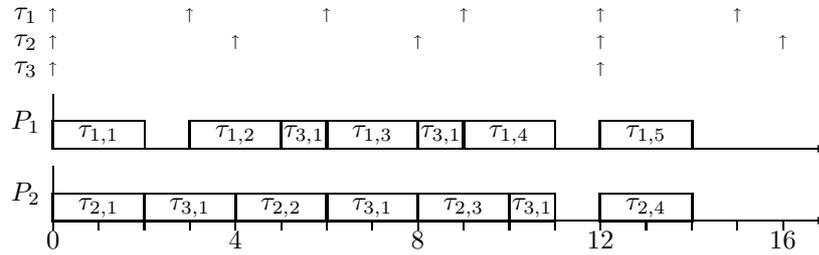
27.6.2 What is a scheduling anomaly?

In theory of science, an anomaly is an event that contradicts a theory, hence putting the prevailing paradigm up for a test [Kuh62]. In this chapter, and in particular in this section, we will use the word “anomaly” in another way. We say that a scheduling algorithm suffers from an anomaly if an intuitively positive change in the task set causes a task to miss a deadline when it met its deadline before the change occurred.

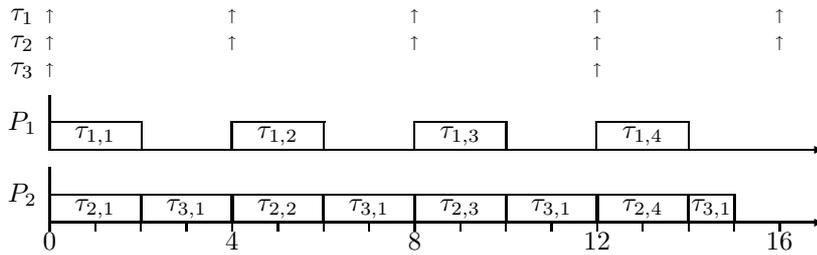
What is a change? Since the task set is only described by T_i , C_i , and S_i , we mean that one or many tasks changed their T_i , C_i , or S_i .

What is an intuitively positive change? An intuitively positive change is a change that decreases the utilization of one or many tasks in the task set. That is, T_i increases, or C_i decreases. If S_i changes then it is not an intuitively positive change.

What does the scheduling algorithm do when the intuitively positive change happens? I conceive of two ways. One way is that the priority assignment and task assignments remain the same. Another way is that, when the task set changes, the algorithm that assigns priorities and/or assign tasks to processors is run again, hence causing possibly new scheduling decisions. If the period is changed then both ways are reasonable; a change in period may be a consequence of inaccuracies in the clock (first way) or of an algorithm choosing another sampling frequency (second way). If the execution time is changed then the first way is most likely; an execution time was smaller than the maximum execution time because the program executed another path. However, the second way is possible as well, because some programs can give different Quality-of-Service by changing their execution time and then the modification of the execution could be known to the scheduling algorithm. In this chapter, if any of these two ways leads to a deadline miss, then we will say that the scheduling algorithm suffers from anomalies.



(a) Task set schedulable



(b) Task set unschedulable

Figure 27.5: When τ_1 increases its period from 3 to 4, the first instance of τ_3 becomes unschedulable (misses its deadline by four time units). This is because τ_3 already barely meets its deadline and the delay from higher priority tasks during the first task period increases by 2 (from 4 to 6).

27.6.3 Examples of anomalies

This section shows that anomalies can occur in many existing preemptive multiprocessor scheduling algorithms. For different scheduling algorithms there are different reasons why anomalies occur. However, if many scheduling algorithms are similar, and the cause of their anomalies is the same, we present only one example.

Period anomalies in global scheduling One reason why anomalies can occur in global scheduling is that an increase in period causes tasks to arrive at different times. These different times do not affect schedulability directly, and the schedule generated when the period increases, performs less work on the processors. However, the execution can be distributed differently. This change in distribution of execution causes more instants when all processors are busy, and this delays a lower priority task even more.

This can happen in global static-priority scheduling (see Observation 3).

Observation 3 *For static-priority preemptive global multiprocessor scheduling, there exist task sets that meet all deadlines with one priority assignment but if the period of a task increases and priorities remain the same then a task misses its deadline.*

Example 9 *Consider the following three periodic tasks: $(T_1 = 3, C_1 = 2)$, $(T_2 = 4, C_2 = 2)$, $(T_3 = 12, C_3 = 7)$. These tasks can be scheduled on two processors (see Figure 27.5(a)). Here $S_1 = S_2 = S_3$ but even if S_i is arbitrary then the task set is still schedulable. However, if we increase the period of τ_1 from 3 to 4 (but not change the relative priority order), the resulting task set misses a deadline (see Figure 27.5(b)).*

A similar but different reason for why period anomalies can occur in global scheduling is that an increase in period causes tasks to arrive at different times. These different times make tasks perform less work on the processors, and the execution is not distributed so that all processors are busy at the same time more frequently. However, just the fact that the arrival times are different causes a task to miss its deadline. This can happen in global static-priority scheduling, but we will not discuss that here. Instead, we will look at a more interesting case, to show that these anomalies are not specific to static-priority scheduling. We will look at the case in which there are no restrictions on the scheduling algorithm: preemption at any time, migration at any time and priorities can vary at any time. Priorities vary as a function of time in such a way that a deadline is only missed if it is impossible to vary the priorities to meet deadlines; this is called optimal scheduling. Observation 4 illustrates this.

Observation 4 *In global optimal scheduling, where the deadline $D_i < T_i$, there exist schedulable synchronous task sets (that is $S_1 = S_2 = \dots = S_n$) such that, if the period of a task increases, a task misses a deadline.*

Example 10 *Consider the following three periodic tasks: $(T_1 = 4, D_1 = 2, C_1 = 1)$, $(T_2 = 5, D_2 = 3, C_2 = 3)$, $(T_3 = 10, D_3 = 8, C_3 = 7)$ to be scheduled using global optimal scheduling on two processors. The tasks are schedulable if $s_1 = s_2 = s_3$ (Figure 27.6(a)). However, if T_1 is increased to 5, the resulting task set is no longer schedulable because it is not possible to construct any schedule that makes the tasks meet their deadlines. The reason is that τ_2 must execute immediately when it arrives, because τ_2 would otherwise miss a deadline. τ_1 must execute 1 time unit within $[0, 2)$ and 1 time unit within $[5, 7)$. Figure 27.6(b) illustrates the situation in which τ_3 starts to execute at time 0 and at time 5. Regardless of when τ_1 executes within these intervals, the two first instances of τ_1 will execute at the same time as τ_2 executes. That is, within the interval $[0, 8)$ there are at least 2 time units when both*

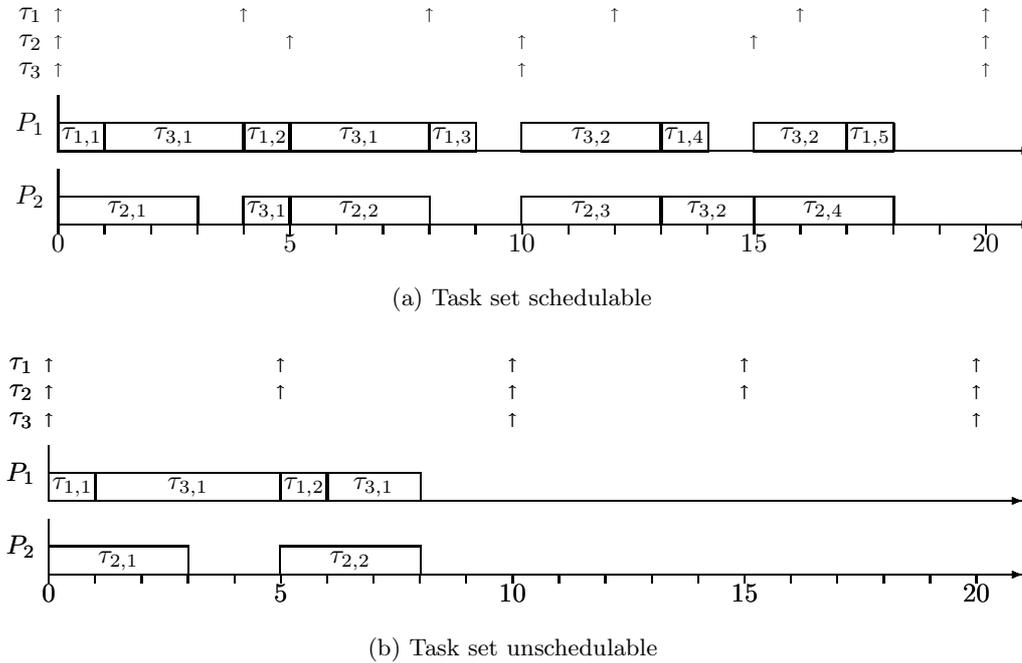


Figure 27.6: Period anomaly in global optimal scheduling. Increasing the period of τ_1 from 4 to 5 causes the second instance of τ_1 to miss its deadline.

τ_1 and τ_2 execute. That is, during $[0, 8)$ there are 6 time units or less available for τ_3 to execute. But τ_3 needs to execute 7 time units in the interval $[0, 8)$. Hence τ_3 misses its deadline.

Note that, although an unschedulable synchronous task set is also an unschedulable asynchronous task set, the fact that a scheduling algorithm suffers from anomalies of a synchronous task set does not necessarily imply that there exist asynchronous task sets that suffer from anomalies.

Period-based anomalies in bin-packing schemes Recall that, with the partition method, bin-packing is a common technique for assigning tasks to processors (see for example [DL78, OB98, LDG01]). All partitioning schemes that we will discuss use bin-packing. Bin-packing algorithms work as follows: (1) sort the tasks according to some criterion; (2) select the first task and an arbitrary processor; (3) attempt to assign the selected task to the selected processor by applying a schedulability test for the processor; (4) if the schedulability test fails, select the next available processor; if it succeeds, select the next task; (5) go to step 3. Step 1, sorting, can be performed by sorting tasks (i) in ascending order

of periods, (ii) by decreasing utilization or (iii) to make the period of a task harmonic to the period of its subsequence task. Step 3 can be performed by attempting to assign a task to the processor that the previous task was assigned on (next-fit) or make attempts on all processors that have at least one task assigned to them but make the attempts in order of processor index (first-fit), or make attempts on all processors that have at least one task assigned but select the processor that is heavily loaded but that still can host the task that is to be assigned (best-fit). Next-fit and decreasing utilization will be discussed in more detail in Section 27.6.5. Partitioning schemes based on bin-packing never miss deadlines, but they declare failure because a schedulability test in the scheduling algorithm cannot guarantee the task to meet deadlines. For that reason, we will say that the scheduling algorithm suffers from an anomaly iff there is a task set for which the algorithm declares success, but there is at least one task such that, if its utilization is decreased, then the algorithm declares failure.

The original bin-packing problem did not address processors and tasks, but rather putting items in bins, where items correspond to tasks and bins correspond to processors. For systems, in which bin sizes do not depend on the item sizes, the existence of bin packing anomalies has been shown for first-fit and first-fit decreasing [Gra72]. Since EDF scheduling on a uniprocessor has a utilization bound of 1, which does not depend on the task sets, there clearly exist anomalies for partitioned EDF. In the remainder of this section, we will discuss static-priority scheduling using partitioning.

One reason for the anomaly in bin-packing is that, if the period increases, the schedulability test used becomes more pessimistic. That cannot happen if the schedulability test is a utilization-based test, where the utilization bound does depends on the number of processors (and hence does not depend on execution times or periods). However, one way of improving the schedulability of partitioning schemes is to make the utilization bound dependent on the periods of the tasks. R-BOUND-MP [LMM98b] is one such bin-packing-based partitioning scheme. Then anomalies can occur.

Observation 5 *For the partitioning scheme, R-BOUND-MP [LMM98b], there exist task sets that can be guaranteed to meet their deadlines, but if the period of a task increases, a task is not guaranteed to meet its deadline.*

Example 11 *Consider the following three periodic tasks: $(T_1 = 1, C_1 = 1)$, $(T_2 = 2, C_2 = 1)$, $(T_3 = 4, C_3 = 2)$ to be scheduled using R-BOUND-MP on two processors. During task set transformation in R-BOUND-MP, the task set will be changed to $(T_1' = 4, C_1' = 4)$, $(T_2' = 4, C_2' =$*

2), $(T_3' = 4, C_3' = 2)$. The task set transformation is done such that if τ' is schedulable, then τ is schedulable. Then continue to run the algorithm. τ_1' will be assigned to processor P_1 . τ_2' will be attempted to be assigned to processor P_1 , but the schedulability test fails because processor P_1 is utilized to 100%. τ_2' is then tested to be assigned to processor P_2 , and that succeeds because no other tasks are yet assigned to processor P_2 . Then τ_3' is tested to be assigned to processor P_1 , but the schedulability test fails, so τ_3' is tested to be assigned to processor P_2 . The schedulability test, *R-BOUND*, succeeds because the ratio between maximum and minimum period of the task set assigned to P_2 is 1, and then the utilization bound according to *R-BOUND* is 100% on that processor. τ_3' is assigned to P_2 , and hence the task set is guaranteed to be schedulable according to *R-BOUND-MP*.

However, if we increase the period of τ_1 from 4 to 5, the resulting task set is no longer guaranteed by *R-BOUND-MP* to be schedulable. To see this, we can run the algorithm *R-BOUND-MP*. The task set is transformed to $(T_1' = 4, C_1' = 4)$, $(T_2' = 4, C_2' = 2)$, $(T_3' = 5, C_3' = 2)$. When assigning τ_1' and τ_2' , the algorithm *R-BOUND-MP* behaves as previously, that is τ_1' is assigned to processor P_1 and τ_2' is assigned to processor P_2 . Then τ_3' is attempted to be assigned to processor P_1 and that attempt fails, so τ_3' is attempted to be assigned to processor P_2 . Now *R-BOUND-MP* behaves differently. The schedulability test fails because the ratio between periods is $5/4 = 1.25$, and thereby *R-BOUND* can only guarantee a task set that has a utilization that is no greater than 85%. The utilization of τ_2' and τ_3' is 90%. Hence *R-BOUND-MP* cannot guarantee the task set to be schedulable.

It would be tempting to think that if a necessary and sufficient schedulability test is used then these anomalies cannot occur. However, anomalies can still occur for partitioning schemes that sort tasks according to periods, because when periods are changed, the order of how tasks are assigned to processors also changes. One such technique is RMFFS improved by using a necessary and sufficient schedulability test. RMFFS [DL78] is a first-fit bin-packing algorithm that originally used a schedulability test that was similar to a utilization based test. Since we use a necessary and sufficient schedulability test, we can be sure that, if the partitioning scheme fails, then a task will actually miss a deadline.

Observation 6 *For the partitioning scheme, RMFFS improved by using a necessary and sufficient schedulability test, there exist task sets that can be guaranteed to meet their deadlines, but if the period of a task increases, a task misses its deadline.*

Example 12 Consider the following four periodic tasks: $(T_1 = 2, C_1 = 1), (T_2 = 3, C_2 = 2), (T_3 = 6, C_3 = 3), (T_4 = 7, C_4 = 1)$ to be scheduled on two processors using RMFFS improved by using a necessary and sufficient schedulability test. RMFFS will first sort the task set according to its periods. That does not change the task set. τ_1 will be assigned to processor P_1 and τ_2 will be assigned to processor P_2 . τ_3 is tested to be assigned to processor P_1 and it succeeds. τ_4 is tested to be assigned to processor P_2 and it succeeds. Hence the task set can be guaranteed.

If the period of τ_3 is increase by 2, we obtain the task set: $(T_1 = 2, C_1 = 1), (T_2 = 3, C_2 = 2), (T_3 = 8, C_3 = 3), (T_4 = 7, C_4 = 1)$. RMFFS will first sort the task set according to its periods. That yields the task set: $(T_1 = 2, C_1 = 1), (T_2 = 3, C_2 = 2), (T_4 = 7, C_4 = 1), (T_3 = 8, C_3 = 3)$. τ_1 will be assigned to processor P_1 and τ_2 will be assigned to processor P_2 . τ_4 is assigned to processor P_1 , but τ_3 cannot be assigned to processor P_1 (because then the utilization would be 1.017, and τ_3 cannot be assigned to processor P_2).

It turns out that all previously published partitioning schemes for static priority preemptive scheduling, suffer from period anomalies as long as repartitioning is done when the task set changes. The reasons for the anomalies are the two reasons given so far, or a similar reason as for the execution-time anomaly in the next paragraph.

Execution-time anomalies in bin-packing Because a decrease in execution time of a task can make the schedulability test succeed when it would otherwise fail, the partitioning can become different when subsequent tasks are assigned to processors, making the task set miss deadlines.

Observation 7 For the partitioning scheme, RMFFS improved by using a necessary and sufficient schedulability test, there exist task sets that can be guaranteed to meet their deadlines, but if the execution time of a task decreases, a task misses its deadline.

Example 13 Consider the following four periodic tasks: $(T_1 = 5, C_1 = 3), (T_2 = 7, C_2 = 4), (T_3 = 8, C_3 = 3), (T_4 = 10, C_4 = 4)$ to be scheduled on two processors using RMFFS improved by using a necessary and sufficient schedulability test. Tasks are sorted according to their periods. That does not change the task set. τ_1 will be assigned to processor P_1 and τ_2 will be assigned to processor P_2 . τ_3 is tested to be assigned to processor P_1 , but fails. τ_3 is then tested to be assigned to processor P_2 , and succeeds. τ_4 is tested to be assigned to processor P_1 , and succeeds.

If the execution time of τ_1 is decreased by 1, we obtain the task set: $(T_1 = 5, C_1 = 2), (T_2 = 7, C_2 = 4), (T_3 = 8, C_3 = 3), (T_4 = 10, C_4 = 4)$.

Tasks are sorted according to their periods. That does not change the task set. τ_1 will be assigned to processor P_1 and τ_2 will be assigned to processor P_2 . τ_3 is tested to be assigned to processor P_1 , and succeeds. τ_4 is tested to be assigned to processor P_1 , but fails. Then τ_4 is tested to be assigned to processor P_2 , but fails again.

Similar execution-time anomalies can occur for many other partitioning schemes such as R-BOUND-MP, R-BOUND-MPrespan, RM-FFDU and RM-FFDUrespan. R-BOUND-MPrespan differs from R-BOUND-MP only in that R-BOUND-MPrespan uses a necessary and sufficient schedulability test. RM-FFDUrespan is defined analogously.

27.6.4 Solutions

Having observed that anomalies can occur for preemptive multiprocessor scheduling, the question arises of how to deal with them. We conceive the following approaches:

- Perform system adjustments such that anomalies cannot occur. For example, if the system suffers from period anomalies, but it does not suffer from execution-time anomalies, then we can perform system adjustments on execution times.
- Use a scheduling algorithm that is designed to dynamically detect anomalies and avoid them.
- Accept only such task sets that cannot suffer from anomalies.
- Use a scheduling algorithm that is designed so that anomalies cannot occur.

Since the first approach transfers the problem of anomalies to another parameter, we only discuss the last three approaches below.

Designed to detect anomalies When designing algorithms to determine whether a certain property holds (e.g. whether there exists an offset assignment [Goo03] or whether there exists a schedule [BCPV96] that causes a given task set to meet deadlines), it is often the case that only solutions that are of the same granularity (a multiple of the greatest common divisor) as parameters describing the problem instance need to be explored. If that assumption holds, and if the parameters describing the problem instance are bounded, then the number of computational steps (computational complexity) of an algorithm is bounded by simply enumerating all combinations (which are bounded). Unfortunately, such an approach is not always possible in anomaly detection (Example 14).

Example 14 For global static-priority preemptive scheduling, the following asynchronous task set is schedulable $(T_1 = 8, C_1 = 4), (T_2 = 20, C_2 = 12), (T_3 = 32, C_3 = 20)$ on two processors, assuming global RM. Any combination of increases in period or decreases in execution times that are multiples of 4 causes the task set to be schedulable. However, increasing T_1 by a value less than the granularity (for example, 1) makes the task set unschedulable. This example is also applicable to optimal priority assignment.

As the example shows, it seems difficult to design a necessary and sufficient condition to determine whether a task set suffers from scheduling anomalies. However, as will be described below, sufficient conditions for anomaly-free task sets are easier to design.

Accept tasks If a scheduling algorithm has a utilization bound, then task sets with a utilization lower than or equal to the utilization bound can neither suffer from execution-time anomalies nor period anomalies. Consequently, in order to avoid anomalies, accept only task sets that have a utilization lower than or equal to this bound. Unfortunately, such an approach introduces additional pessimism, since there are anomaly-free task sets with a utilization that is higher than the bound.

We can also use a schedulability test such that if the task set can be guaranteed to meet deadlines according to this schedulability test then the task set is also anomaly-free, assuming that the priority order does not change. Theorem 21 does this.

Theorem 21 (Circumventing anomalies) Consider global static-priority scheduling of periodic tasks. If, for each task τ_i in a task set, there exists a $R_i^{ub} \leq T_i$ such that:

$$C_i + \frac{1}{m} \sum_{j \in hp(i)} \left(\left\lfloor \frac{R_i^{ub}}{T_j} \right\rfloor \cdot C_j + 2C_j \right) \leq R_i^{ub} \quad (27.16)$$

then the task set meets all deadlines. In addition, if all tasks met their deadlines and tasks increase their period and the relative priority order does not change, then all deadlines will continue to hold.

Proof: For this lemma, we need to show that two conditions are satisfied:

1. The task set meets its deadlines.

This property has been proven by [LMM98a].

2. Increasing periods does not jeopardize schedulability.

It is clear that the expression $\left\lfloor \frac{R_i^{ub}}{T_j} \right\rfloor \cdot C_j + 2C_j$ is non-increasing as T_j increases. Hence, if Equation 27.16 holds and T_j increases, then Equation 27.16 still holds. This implies that the task set remains schedulable even when T_j increases.

If T_i increases, neither the left-hand side nor the right-hand side (R_i^{ub}) of Equation 27.16 will change. Thus, the task set remains schedulable when T_i increases.

□

Designed to avoid anomalies We conceive three ways of designing scheduling algorithms to avoid anomalies: optimal scheduling, no repartitioning and anomaly-free repartitioning.

Scheduling algorithms that are optimal for $D_i = T_i$ must¹² have the property that $\sum_{i=1}^n \frac{C_i}{T_i} \leq m \Rightarrow$ schedulable. Obviously, such algorithms do not suffer from anomalies. However, as we saw in Observation 4, scheduling algorithms that are optimal for $D_i \neq T_i$ can still suffer from anomalies.

Scheduling algorithms, that partition a task set and apply an anomaly-free uniprocessor scheduling algorithm on each processor and do not repartition the task set when the task set changes do not suffer from anomalies. Unfortunately such an approach can cause deadlines to be missed at a system utilization of zero (Example 15).

Example 15 Consider a task set of $m + 1$ tasks where $(T_i = 1, C_i = 0.5 + \epsilon)$ for $i = 1..m - 1$ and $(T_i = 1, C_i = 0.5 - \epsilon/2)$ for $i = m$ and $i = m + 1$ to be scheduled on m processors. No matter how partitioning and scheduling is done (assuming that we use the partitioned method), the only way that this task set can be made to meet its deadlines is to partition τ_m and τ_{m+1} to the same processor, let us say processor P_m . Now, assume that the execution times of tasks changes so that $(T_i = 1, C_i = \epsilon)$ for $i = 1..m - 1$ and $(T_i = 1, C_i = 0.5 + \epsilon)$ for $i = m$ and $i = m + 1$. Assuming that $m \rightarrow \infty$ and $\epsilon \rightarrow 0$ and that repartitioning is not allowed, then the task set is unschedulable with a system utilization of zero.

Anomaly-free bin-packing can be achieved using a next-fit policy [Mur88]. It is easy to see that such a policy can also be used in real-time scheduling for partitioning of tasks that are scheduled by EDF on each uniprocessors because in EDF each processor has a utilization bound of

¹²In the synchronous case, when task characteristics are integers, PF [BCPV96] is optimal. In the asynchronous case, when characteristics are real numbers, a simple polynomial time algorithm can be designed by using Theorem 1 in [Hor74]. These two algorithms can schedule all task sets with a utilization $\leq m$.

1 and hence can be thought of as a bin with a size that does not depend on the size of the elements. However, from the perspective of real-time static-priority scheduling that result is not fully satisfactory because (i) the algorithm in [Mur88] does not have a system utilization bound, and (ii) a straightforward extension to static priority scheduling does not perform well. The next-fit scheduling algorithm by Dhall [DL78] can be shown to suffer from period anomalies because tasks are sorted according to periods before assignment. Consider, for example, $(T_1 = 2, C_1 = 1), (T_2 = 4, C_2 = 1), (T_3 = 5, C_3 = 4)$ to be scheduled on 2 processors, and then increase T_2 by 2 time units. One straightforward extension from bin-packing to partitioned multiprocessor scheduling is as follows. Apply the next-fit by [Mur88] (that is, no sorting) but allow a task to be assigned to a processor if the utilization of the tasks that are assigned to the processor and the task that is added is no greater than the Liu and Layland utilization bound [LL73]. Otherwise, assign the task to the next empty processor. However, such an approach can fail with a system utilization of $(1/2) \cdot \ln 2 \approx 0.35$ (see [And03]).

Hence there is a need for a static-priority partitioning scheme that: (i) does not suffer from anomalies and (ii) has a utilization bound that is no lower than the best partitioning schemes. The second item means that we should strive for a utilization bound of 0.5 because this is what we achieved in section Section 27.5. However, this turns out to be difficult, and when I started my research, the best utilization bound was 0.41, so achieving 0.41 is sufficient to meet our goal.

27.6.5 Anomaly-free partitioning

In this section, we will propose a partitioning scheme RM-DU-NFS that avoids both period and execution-time anomalies, while still providing a system utilization bound that is no lower than what the best published partitioning schemes could achieve before I started my research. RM-DU-NFS is described in Algorithm 16.

In the remainder of this section, we will define a certain property SD that is used in a set of lemmas. These lemmas are used to prove that RM-DU-NFS suffers neither from period anomalies nor execution-time anomalies. Finally, we prove the system utilization bound of RM-DU-NFS.

Definition 14 *An assignment of the task set τ to processors is SD (schedulable with decreasing utilization on each processor) if and only if: $\forall p \in [1..m - 1]: \min_{\tau_j \in \tau^p} C_j/T_j \geq \max_{\tau_j \in \tau^{p+1}} C_j/T_j$ and $\forall p \in [1..m]: \sum_{\tau_j \in \tau^p} C_j/T_j \leq n_p \cdot (2^{1/n_p} - 1)$.*

Algorithm 16 Anomaly-free partitioning: RM-DU-NFS.

Input: τ and m
Output: assigned_processor(τ_i)

- 1: Sort tasks (rearrange indices) such that $u_1 \geq u_2 \geq u_3 \geq \dots \geq u_n$
- 2: $i := 1$ $j := 1$
- 3: for each $p \in [1..m]$ $n_p := 0$
- 4: for each $p \in [1..m]$ $util_on_processor_p := 0$
- 5: while $i \leq n$ loop
- 6: $util_bound := (n_j + 1) \cdot (2^{1/(n_j+1)} - 1)$
- 7: if $util_on_processor_j + u_i \leq util_bound$
- 8: assigned_processor(τ_i) = j
- 9: $n_j := n_j + 1$
- 10: $util_on_processor_j := util_on_processor_j + u_i$
- 11: else
- 12: if $j=m$ then
- 13: declare failure
- 14: else
- 15: $j := j + 1$
- 16: assigned_processor(τ_i) = j
- 17: $n_j := n_j + 1$
- 18: $util_on_processor_j := util_on_processor_j + u_i$
- 19: endif
- 20: $i := i + 1$
- 21: end while
- 22: declare success

Lemma 6 *RM-DU-NFS declares success for a given $\tau \Leftrightarrow$ there exists an assignment of τ that is SD.*

Follows directly from the algorithm.

Lemma 7 *Consider two task sets τ^{before} and τ^{after} , where τ^{after} differs from τ^{before} only in that there is a task τ_i^{after} such that $u_i^{after} \leq u_i^{before}$. If there exists an assignment of τ^{before} that is SD then there exists an assignment of τ^{after} that is SD.*

Proof: If tasks in τ^{after} have the same assignment as tasks in τ^{before} , then $\forall p \in [1..m]: \sum_{\tau_j \in \tau^{after,p}} C_j/T_j \leq n_p \cdot (2^{1/n_p} - 1)$. However, with such an assignment of τ^{after} , τ^{after} is not necessarily SD, because it could be that τ_i^{after} has a lower utilization than tasks assigned a processor with higher index. We will now show that by swapping the assignment of tasks, it is possible to achieve an assignment of τ^{after} that is SD.

Consider those tasks $\tau_j \in \tau^{before}$ that satisfy $u_i^{before} \geq u_j \geq u_i^{after}$. Those tasks are assigned to the processors P_k, \dots, P_l . For each of the processors $P_g \in \{P_{k+1}, \dots, P_l\}$, move the task with the highest utilization from processor P_g to P_{g-1} . Also move task τ_i^{before} from P_k to processor P_l . The number of tasks on each processor is unaffected, so the utilization bounds of each processor are unaffected. The utilization

of tasks on each of the processors $P_g \in \{P_k, \dots, P_l\}$ does not increase, and for the other processors the utilization does not change. This new assignment of τ_{after} also satisfies $\min_{\tau_j \in \tau^p} C_j/T_j \geq \max_{\tau_j \in \tau^{p+1}} C_j/T_j$. Hence the new assignment of τ_{after} is SD. \square

Lemma 8 Consider two task sets τ^{before} and τ^{after} , where τ^{after} differs from τ^{before} only in that there is a task τ_i^{after} such that $u_i^{after} \leq u_i^{before}$. If RM-DU-NFS declares success for τ^{before} , then RM-DU-NFS declares success for τ^{after} .

Proof: By applying Lemma 6 and Lemma 7, we can reason as follows:

RM-DU-NFS declares success for τ^{before} . \Rightarrow
 There exists an assignment of τ^{before} that is SD. \Rightarrow
 There exists an assignment of τ^{after} that is SD. \Rightarrow
 RM-DU-NFS declares success for τ^{after} .

Lemma 9 Consider a task set τ such that RM-DU-NFS has declared success when τ is applied. If u_i is decreased for any subset of τ , then RM-DU-NFS will declare success.

Proof: Apply Lemma 8 repeatedly for each task that decreased its utilization. \square

Theorem 22 Consider a task set τ such that RM-DU-NFS has declared success when τ is applied. If T_i is increased or C_i is decreased for any subset of τ , then RM-DU-NFS will declare success.

Proof: Apply Lemma 9. \square

In the remainder of this section, we will prove the utilization bound of RM-DU-NFS. First, a lemma proves the utilization bound assuming that the tasks are sorted, then a theorem states the same utilization bound holds even if tasks were not sorted.

Lemma 10 If $\sum_{i=1}^n \frac{C_i}{T_i} \leq (\sqrt{2} - 1) \cdot m$ and $u_1 \geq u_2 \geq u_3 \geq \dots \geq u_n$ then executing lines 2-25 in Algorithm 16 (RM-DU-NFS) will declare success.

Proof: See [And03]. \square

Theorem 23 If $\sum_{i=1}^n \frac{C_i}{T_i} \leq (\sqrt{2} - 1) \cdot m$ then algorithm RM-DU-NFS will declare success.

Proof: Apply Lemma 10. \square

27.6.6 Generalization

The anomaly-free partitioning scheme presented in Section 27.6.5 can actually be generalized in the following way. Consider a general partitioning scheme, *xx-DU-NFS*, that is analogous to Algorithm 16, but uses instead an arbitrary uniprocessor scheduling algorithm *xx* with a utilization bound of $utilization_bound = A$. The proofs can then be extended for that so that the algorithms are anomaly-free and the system utilization bound is $A/2$. We then obtain two special cases: *RM-DU-NFS* with a system utilization bound of $\sqrt{2} - 1$ and *EDF-DU-NFS* with a system utilization bound of 0.5.

27.7 Introduction to aperiodic scheduling

27.7.1 Motivation

In some applications, tasks are requested to execute as a result of external events and the time when these events occur cannot be controlled by an application designer. For example, buttons are pushed, requests arrive at a server, or emergency conditions are detected (such as: this vehicle is going to crash within 0.5s). Such situations call for aperiodic real-time scheduling algorithms.

In other applications, such as computer graphics, multimedia and control loops, service should be delivered repeatedly. A straightforward solution is to use a periodic scheduling algorithm. However, periodic scheduling is a special case of aperiodic scheduling, so an aperiodic scheduling algorithm can solve this as well. An advantage of using aperiodic scheduling to give service repeatedly is that an aperiodic task gives more flexibility, which enables the computer to give better service because it can adapt to information that is available only at run-time (one such example can be found in [MFFR02]). Such an approach is useful when the application is designed not to require equidistant sampling and execution times can be controlled to achieve different Quality-of-Service levels.

When scheduling aperiodic tasks, it is common to distinguish between online scheduling and offline scheduling. Offline scheduling means that the whole task set, including future task arrivals, is known initially. Online scheduling means that initially, the task set is not known to the scheduling algorithm, but the characteristics of a task are revealed to the scheduling algorithm when the task arrives. This part of the chapter considers online scheduling of aperiodic tasks. If the characteristics of tasks are known initially, then our scheduling algorithms can ignore this information and schedule tasks anyway.

The remainder of this chapter is organized as follows. Section 27.7.2 describes the system model that we use. Section 27.7.3 discusses issues in the design of scheduling algorithms for aperiodic tasks. After this section follows the Sections 27.8 and 27.9 which present the main results: the design of scheduling algorithms and their capacities.

27.7.2 System model

We consider the problem of scheduling a task set τ of aperiodically-arriving real-time tasks on m identical processors. A task τ_i has an arrival time A_i , an execution time C_i and a deadline D_i , that is, the task requests to execute C_i time units during the time interval $[A_i, A_i + D_i)$. For convenience, we call $A_k + D_k$ the absolute deadline of the task τ_k and call D_k the relative deadline of the task τ_k . We assume that C_i and D_i are positive real numbers such that $C_i \leq D_i$ and A_i are a real numbers. With no loss of generality we can assume that $0 = A_1 \leq A_2 \leq \dots \leq A_n$. We let the set of *current* tasks at time t be defined as $V(t) = \{\tau_k : A_k \leq t < A_k + D_k\}$.

The *utilization*, u_i , of a task τ_i is $u_i = C_i/D_i$. The utilization at time t is $U(t) = \sum_{\tau_i \in V(t)} u_i$. Since we consider scheduling on a multiprocessor system, the utilization is not always indicative of the load of the system because the original definition of utilization is a property of the current tasks only and does not consider the number of processors. Therefore, we use the concept of *system utilization*, $U_s(t) = U(t)/m$. The finishing time f_i of a task τ_i is the earliest time when task τ_i has executed C_i time units. If $f_i \leq A_i + D_i$, then we say that task τ_i meets its deadline.

When we study global job-static priority scheduling, the system behaves as follows. Each task is assigned a global priority. Of all tasks that have arrived, but have not finished, the m highest-priority tasks are executed¹³ in parallel on the m processors. When we study partitioned scheduling, the system behaves as follows. When a task arrives, it is immediately assigned to a processor; the task is then only allowed to execute on the processor to which it is assigned. On each processor, the task with the highest priority of those tasks which have arrived, but have not finished, is executed.

Common to all scheduling algorithms that we propose is that they are not allowed to use information about the future, that is, at time t , they are not allowed to use A_i , D_i or C_i of tasks with $A_i > t$. However, at the moment when a task τ_i arrives, that is at time A_i , then C_i and D_i are immediately known to the scheduling algorithm.

We will analyze the performance of our scheduling algorithm using a *utilization bound* such that if the system utilization is, at every time,

¹³At each instant, the processor chosen for each of the m tasks is arbitrary. If less than m tasks are to be executed simultaneously, some processors will be idle.

less than or equal to this utilization bound, then all deadlines are met. Our objective is to design a scheduling algorithm with a high utilization bound.

27.7.3 Design issues in aperiodic scheduling

The design of an algorithm for scheduling aperiodic tasks on a multiprocessor needs to consider the same issues as in periodic scheduling (see Section 27.2.3). However, two aspects of aperiodic scheduling are important and noteworthy.

First, it is desirable to design an optimal aperiodic scheduling algorithm, that is, one that only misses a deadline when it is impossible to meet all deadlines. EDF is an optimal scheduling algorithm for a uniprocessor. For a multiprocessor, designing such algorithms is possible in (i) periodic scheduling [Hor74], and (ii) in aperiodic scheduling when all task arrival times, deadlines and periods are known (a slight modification of the algorithm in [LM81] can do this)¹⁴. However, these algorithms are of little use in the context of this part of the chapter because they both assume (i) dynamic-priority scheduling and (ii) task parameters are known before run-time. In addition, it is known that in online scheduling of aperiodic tasks (which is the subject of this part of the chapter), an optimal algorithm cannot exist [DM89, Mok83].

Second, tasks “disappear” when their deadlines expire. This implies that a processor that was busy at one time, because it executes a task that has a long execution time, may be idle for a long duration at a later time. This has effects on the design and analysis of our new scheduling algorithms, as we will see in Chapter 27.8 and Chapter 27.9.

27.8 Global scheduling

27.8.1 Introduction

In this section, we study global multiprocessor scheduling algorithms and their utilization bounds for aperiodic tasks where future arrivals are unknown. In particular, we extend a previously proposed job-static¹⁵ priority scheduling algorithm for periodic tasks with migration capability to aperiodic scheduling and show that it has a utilization bound of 0.5. This bound is close to the best achievable for a job-static priority scheduling algorithm. With an infinite number of processors, no job-static priority scheduling algorithm can perform better. We also propose a simple admission controller which guarantees that admitted

¹⁴Other optimal algorithms have been proposed later [DM89, Theorem 8] and [BCPV96] but they assume that task parameters are integers.

¹⁵Recall that job-static and task-static are synonymous in aperiodic scheduling.

tasks meet their deadlines and for many workloads, it admits tasks so that the real utilization can be kept above the utilization bound.

The remainder of this section is organized as follows. Section 27.8.2 presents the **EDF-US($m/(2m-1)$)** algorithm and Section 27.8.3 gives the derivation of its utilization bound.

27.8.2 Design of EDF-US($m/(2m-1)$)

EDF-US($m/(2m-1)$) means Earliest-Deadline-First Utilization-Separation with separator $m/(2m-1)$. We say that a task τ_i is heavy if $u_i > m/(2m-1)$ and a task is light if $u_i \leq m/(2m-1)$. **EDF-US($m/(2m-1)$)** assigns priorities so that all heavy tasks receive higher priority than all light tasks. The relative priority order among the heavy tasks is arbitrary, but the relative priority order among the light tasks is given by EDF. The rationale for this separation of heavy and light tasks is that, if heavy tasks received a low priority, then heavy tasks could miss deadlines even if there is ample capacity available.

27.8.3 Utilization bound of EDF-US($m/(2m-1)$)

In order to derive a utilization bound for **EDF-US($m/(2m-1)$)** when used with aperiodic tasks, we will look at the case with periodic tasks [SB02]. There, it was shown that if all tasks are light and the system utilization is always no greater than $m/(2m-1)$, then EDF schedules all tasks to meet their deadlines. If every heavy task was assigned its own processor, the light tasks executed on the remaining processors and the system utilization of all these tasks was no greater than $m/(2m-1)$, then all deadlines would also hold. It was also shown that, even if a light task was allowed to execute on a processor where a heavy task executes when the heavy task does not execute, then deadlines continue to hold. The reason why this technique works for periodic tasks is that the number of current tasks never changes at run-time because, when a deadline of a current task has expired, a new current task with the same execution time and deadline arrives.

However, in aperiodic scheduling, the number of heavy tasks is not necessarily the same at all times. Hence, the number of processors that are available for light tasks may vary with time. For this reason, we will prove (in Lemma 14) a schedulability condition of EDF for light tasks when the number of processors varies. We will do this in two steps. First, we will prove that OPT, an optimal scheduling algorithm, meets deadlines. Second, we will prove (in Lemma 13) that if any scheduling algorithm meets deadlines, then EDF will also do so if EDF is provided faster processors. The second step is proven by using a result

(Lemma 12) that tells how much work a scheduling algorithm does. To do this, we need to define work, OPT and a few other concepts.

Since we study scheduling on identical processors, all processors in a computer system have the same speed, denoted s , but two different computer systems may have processors of different speeds. If the speed of a processor is not explicitly written out, it is assumed that $s = 1$. A processor that is busy executing tasks during a time interval of length l does $l \cdot s$ units of work. This means that if a processor of speed s starts to execute a task with execution time $s \cdot l$ at time 0, then the task has finished its execution without interruptions at time l . Let $W(A, m(t), s, \tau, t)$ denote the amount of work done by the task set τ during the time interval $[0, t)$ scheduled by algorithm A when the number of processors varies according to $m(t)$ and each processor runs with speed s . We assume that $m(t)$ changes at the time instants denoted $change_1, change_2, \dots$ and let m_{UB} be a number such that $\forall t : m(t) \leq m_{UB}$. For convenience, we will say that a computer system has $m(t)$ processors when we mean that the number of processors varies as a function $m(t)$, where t is time.

Let OPT denote an algorithm that executes every current task τ_i , $L \cdot C_i/D_i$ time units in every time interval of length L . It implies that a task τ_i will execute C_i time units in every time interval of length D_i . In particular, it implies that τ_i will execute C_i time units in the interval $[A_i, A_i + D_i)$, and hence it meets its deadline. One can see that if $\forall t: U(t) \leq m(t)$ then OPT will succeed in scheduling every current task, τ_i , $L \cdot C_i/D_i$ time units in every time interval of length L and hence they meet their deadlines. We first show that there exists an algorithm OPT that has these properties and that never executes a task on two or more processors simultaneously. To that end, we can make use of Theorem 1 in [Hor74], which we repeat below, and for convenience have rewritten to use our notation and made a trivial rewriting.

Lemma 11 *If there are m processors and n tasks, with all $A'_i = 0, D'_i = K > 0$, and if preemption is allowed, but no task may be processed on two machines simultaneously, then there exists a schedule which finishes all tasks by K if, and only if*

- (a) $\frac{C'_i}{D'_i} \leq 1$ for each task τ_i and
- (b) $\sum_{i=1}^n \frac{C'_i}{D'_i} \leq m$

Proof: See [Hor74]. □

To see how this result can be used for our purposes, we proceed as follows. We first split an arbitrary time interval of length L into a set $\{[s_1, e_1), [s_2, e_2), \dots, [s_l, e_l)\}$ of time intervals with $s_{j+1} = e_j$ such that in each time interval $[s_j, e_j)$ the number of processors does not change and the number of current tasks does not change. We also split a task τ_i into

l_i subtasks $\tau_{i,1}, \tau_{i,2}, \dots, \tau_{i,l_i}$, such that $\tau_{i,j}$ has $A_{i,j} = s_j, D_{i,j} = e_j - s_j$ and $C_{i,j} = (e_j - s_j) \cdot \frac{C_i}{D_i}$. Since Lemma 11 assures us that each subtask meets its deadline, then it holds that every task meets its deadline. We conclude that there is an algorithm OPT that never executes a task on two or more processors simultaneously and it can guarantee that, if $\forall t: U(t) \leq m(t)$, then all deadlines are met.

Later when we prove the condition of schedulability for EDF, we will need the following lemma that tells us how much work a work-conserving scheduling algorithm performs compared to any scheduling algorithm. Such a condition was proven by [PSTW97] but it was assumed in that work that the number of processors did not vary; therefore, we will remove this limitation. We say that a scheduling algorithm is *work-conserving* if it never leaves a processor idle when there are tasks available for execution. For our purposes, **EDF-US(m/(2m-1))** is work-conserving.

Lemma 12 *Consider scheduling on $m(t)$ processors. Let A be an arbitrary work-conserving scheduling algorithm and let A' be an arbitrary scheduling algorithm. Then we have:*

$$W(A, m(t), (2 - \frac{1}{m_{UB}}) \cdot s, \tau, t) \geq W(A', m(t), s, \tau, t)$$

Proof: The proof is by contradiction. Suppose that it is not true; i.e., there is some time-instant by which a work-conserving algorithm A executing on $(2 - 1/m_{UB}) \cdot s$ -speed processors has performed strictly less work than some other algorithm A' executing on speed- s processors.

Let $\tau_j \in \tau$ denote a task with the earliest arrival time such that there is some time-instant t_0 satisfying

$$W(A, m(t), (2 - \frac{1}{m_{UB}}) \cdot s, \tau, t_0) < W(A', m(t), s, \tau, t_0)$$

and the amount of work done on task τ_j by time-instant t_0 in A is strictly less than the amount of work done of τ_j by time-instant t_0 in A' . One such τ_j must exist, because there is a time $t < t_0$ such that $W(A, m(t), (2 - \frac{1}{m_{UB}}) \cdot s, \tau, t) = W(A', m(t), s, \tau, t)$. For example, $t = 0$ gives one such equality. By our choice of A_j , it must be the case that

$$W(A, m(t), (2 - \frac{1}{m_{UB}}) \cdot s, \tau, A_j) \geq W(A', m, s, \tau, A_j)$$

Therefore, the amount of work done by A' over $[A_j, t_0)$ is strictly more than the amount of work done by A over the same interval. The fact that the amount of work done on τ_j in $[A_j, t_0)$ in A is less than the amount of work done on τ_j in $[A_j, t_0)$ in A' , implies that τ_j does not finish before t_0 .

Let a be the maximum number in $[A_j, t_0)$ such that

$$W(A, m(t), (2 - \frac{1}{m_{UB}}) \cdot s, \tau, a) \geq W(A', m(t), s, \tau, a)$$

Notice that $a < t_0$. Such a must exist — $a = r_j$ gives one. We also know that $a < t_0$.

Consider two cases:

1. There is no k such that $change_k > a$

Then let $b = t_0$.

2. There is a k such that $change_k > a$

- (a) There is no k such that $change_k > a$ and $change_k \leq t_0$

Then let $b = t_0$.

- (b) There is no k such that $change_k > a$ and $change_k \leq t_0$

Then let $b = \min(change_k : change_k > a)$.

We will now study the time interval $[a, b)$, and let us summarize its properties:

$$a < b$$

$$W(A, m(t), (2 - \frac{1}{m}) \cdot s, \tau, a) \geq W(A', m(t), s, \tau, a)$$

$$W(A, m(t), (2 - \frac{1}{m}) \cdot s, \tau, b) < W(A', m(t), s, \tau, b)$$

$$\forall t \in [a, b) \text{ it holds that } m(t) = m(a) \leq m_{UB}$$

τ_i has not finished at time b in the schedule generated by A

Let $exec(t)$ denote the number of processors that were busy at time t in the case that tasks were scheduled by algorithm A . Let x denote the cumulative length of time over the interval $[a, b)$ during which $m(t) = exec(t)$. Let $y = (b - a) - x$, that is, the length of time over the interval $[a, b)$ during which A idles some processor.

We make the following two observations.

- Since A is a work-conserving scheduling algorithm, τ_j , which has not finished by instant b in the schedule generated by A , must have executed for at least y time units by time b in the schedule generated by A ; while it could have executed for at most $(x+y)$ time units in the schedule generated by A' ; therefore,

$$(x + y) > (2 - \frac{1}{m_{UB}}) \cdot y$$

- The amount of work done by A over $[a, b]$ is at least:

$$\left(2 - \frac{1}{m_{UB}}\right) \cdot s \cdot (m(a) \cdot x + y).$$

while the amount of work done by A' over this interval is at most

$$m(a) \cdot s \cdot (x + y)$$

therefor it must be the case that

$$m(a) \cdot (x + y) > \left(2 - \frac{1}{m_{UB}}\right) \cdot (m(a) \cdot x + y).$$

By adding $m(a) - 1$ times Inequality 27.17 to Inequality 27.17, we get

$$\begin{aligned} & (m(a) - 1) \cdot (x + y) + m(a) \cdot (x + y) > \\ & (m(a) - 1) \cdot \left(2 - \frac{1}{m_{UB}}\right) \cdot y + \left(2 - \frac{1}{m_{UB}}\right) \cdot (m(a) \cdot x + y) \\ & \equiv (2m(a) - 1) \cdot (x + y) > \left(2 - \frac{1}{m_{UB}}\right) \cdot m(a) \cdot (x + y) \\ & \Rightarrow (2m(a) - 1) \cdot (x + y) > \left(2 - \frac{1}{m(a)}\right) \cdot m(a) \cdot (x + y) \\ & \equiv (2m(a) - 1) \cdot (x + y) > (2m(a) - 1) \cdot (x + y) \end{aligned}$$

which is a contradiction. \square

We can now present a lemma that can be used to indirectly determine whether EDF meets deadlines.

Lemma 13 *Let A' denote an arbitrary scheduling algorithm. Let π' denote a computer platform of $m(t)$ processors and let π denote a computer platform that has, at every time, the same number of processors as π' , but the processors of π have speed $2 - \frac{1}{m_{UB}}$. If A' meets all deadlines of τ on π' , then EDF meets deadlines of τ on π .*

Proof: Since EDF is a work-conserving scheduling algorithm we obtain from Lemma 12 that for every t :

$$W(EDF, m(t), \left(2 - \frac{1}{m_{UB}}\right) \cdot s, \tau, t) \geq W(A', m(t), s, \tau, t)$$

Let $d_i = A_i + D_i$ and let $\tau^k = \{\tau_1, \tau_2, \dots, \tau_k\}$, where tasks are ordered in EDF priority, that is, $d_1 \leq d_2 \leq \dots \leq d_k$. We will prove the lemma using induction on k .

Base case If $k \leq m$, this implies that a task is not delayed by another task and hence all deadlines hold.

Induction step We make two remarks. First, the scheduling of tasks $\tau_1, \dots, \tau_k \in \tau^{k+1}$ is the same as the scheduling of tasks $\tau_1, \dots, \tau_k \in \tau^k$, so we need to prove only that τ_{k+1} meets its deadline. Second, since τ_{k+1} has the lowest priority according to EDF and there is no task with lower priority, τ_{k+1} , will do all work that the higher priority tasks do not do.

From the lemma we know that:

*all tasks in τ^{k+1} meet their deadlines
when scheduled by A' on π'*

We can now reason as follows:

*all tasks in τ^{k+1} meet their deadlines
when scheduled by A' on $\pi' \Rightarrow$*

$$W(A', m(t), 1, \tau^{k+1}, d_{k+1}) = \sum_{j=1}^{k+1} C_j \stackrel{\text{use Lemma 12}}{\Rightarrow}$$

$$W(EDF, m(t), 2 - \frac{1}{m_{UB}}, \tau^{k+1}, d_{k+1}) \geq \sum_{j=1}^{k+1} C_j \Rightarrow$$

*τ_{k+1} executes at least C_{k+1} time units before
 d_{k+1} when scheduled by EDF on $\pi \Rightarrow$
 τ_{k+1} meets its deadline when
scheduled by EDF on $\pi \Rightarrow$
all tasks in τ^{k+1} meet their
deadlines when scheduled by EDF on π*

□

The following lemma is a schedulability condition for EDF on a variable number of processors.

Lemma 14 *Consider EDF scheduling on $m(t)$ processors. If $\forall t: U(t) \leq m(t) \cdot \frac{m_{UB}}{2m_{UB}-1}$ and $C_i/D_i \leq \frac{m_{UB}}{2m_{UB}-1}$ then all tasks meet their deadlines.*

Proof: From the properties of OPT we know that:

If a task set is such that $\forall t: U(t) \leq m(t)$ and $C_i/D_i \leq 1$,
then OPT meets all deadlines.

Applying Lemma 13 yields:

If a task set is such that $\forall t: U(t) \leq \overline{m(t)}$ and $C_i/D_i \leq 1$ and processors have the speed of $2 - \frac{1}{m_{UB}}$, then EDF meets all deadlines.

Scaling the speed of processors yields:

If a task set is such that $\forall t: U(t) \leq \overline{m(t)} \cdot m_{UB}/(2m_{UB} - 1)$ and $C_i/D_i \leq m_{UB}/(2m_{UB} - 1)$ and processors have the speed of 1, then EDF meets all deadlines.

□

To be able to prove utilization bounds of task sets that have not only light tasks but also heavy tasks, we introduce two new terms and present a lemma from previous research. Let $heavy(t)$ denote the number of current tasks at time t that have $C_i/D_i > \frac{m}{2m-1}$ and let $U_{light}(t)$ denote the sum of utilization of all current tasks at time t that have $C_i/D_i \leq \frac{m}{2m-1}$.

We will make use of a result by Ha and Liu [HL94], that states how the finishing time of a task is affected by the variability of execution times of tasks in global job-static priority scheduling. Let f_i denote the finishing time of task τ_i , f_i^+ denote the finishing time of task τ_i when all tasks execute at their maximum execution time and f_i^- denote the finishing time of task τ_i when all tasks execute at their minimum execution time. Lemma 15 presents the result that we will use.

Lemma 15 *For global scheduling where the priority orders of tasks does not change when the execution times change, it holds that:*

$$f_i^- \leq f_i \leq f_i^+$$

Proof: See Corollary 3.1 in [HL94].

□

We can now design a schedulability condition for EDF-US($m/2m-1$).

Lemma 16 *Consider EDF-US($m/(2m-1)$) scheduling on m processors. If $\forall t: U_{light}(t) \leq (m - heavy(t)) \cdot m/(2m - 1)$ and $\forall t: heavy(t) \leq m - 1$, then all tasks meet their deadlines.*

Proof: The tasks with $C_i/D_i > \frac{m}{2m-1}$ meet their deadlines because they receive the highest priority and there are at most $m - 1$ of them. It remains to be proven that tasks with $C_i/D_i \leq \frac{m}{2m-1}$ meet their deadlines. Consider two cases.

- All tasks with $C_i/D_i > \frac{m}{2m-1}$ have $C_i = D_i$,

The tasks with $C_i/D_i \leq \frac{m}{2m-1}$ experience it as if there were $m - heavy(t)$ processors available for them to execute on, and according to Lemma 14 the tasks meet their deadlines.

- Of the tasks with $C_i/D_i > \frac{m}{2m-1}$, there is a subset of tasks that have $C_i < D_i$.

If this subset of tasks had $C_i = D_i$, then according to the first case, all deadlines would hold. Reducing C_i of tasks with $C_i/D_i > \frac{m}{2m-1}$ does not affect priority order so according to Lemma 15 all deadlines continue to hold.

□

Now we have all the lemmas at our disposal for stating our final theorem.

Theorem 24 *Consider **EDF-US(m/(2m-1))** scheduling on m processors. If $\forall t: U(t) \leq m \cdot m/(2m-1)$ then all tasks meet their deadlines.*

Proof: It follows from $\forall t: U(t) \leq m \cdot m/(2m-1)$ that: $\forall t: U_{light}(t) \leq (m - heavy(t)) \cdot m/(2m-1)$ and $\forall t: heavy(t) \leq m-1$. Applying Lemma 16 gives the theorem. □

Theorem 24 states that **EDF-US(m/(2m-1))** has a utilization bound of $m/(2m-1)$. For a large number of processors this bound approaches $1/2$. In Example 16 we show that an upper bound on the utilization bound of every job-static priority scheduling algorithm is $0.5 + 0.5/m$, which demonstrates that **EDF-US(m/(2m-1))** is close to the best possible performance and with an infinite number of processors, no job-static priority scheduling algorithm can perform better than **EDF-US(m/(2m-1))**.

Example 16 *Consider $m+1$ aperiodic tasks that should be scheduled on m processors using job-static priority global scheduling. All tasks have $A_i = 0, D_i = 1, C_i = 0.5 + \epsilon$, so at every instant during $[0, 1)$, the system utilization is $0.5 + \epsilon + \frac{0.5+\epsilon}{m}$. Because of job-static priority scheduling, there must be a task with lowest priority, and that priority order is not permitted to change. That task executes when its higher priority tasks do not execute. Hence the lowest priority task executes $0.5 - \epsilon$ time units during $[0, 1)$, but it needs to execute $0.5 + \epsilon$ time units, so it misses its deadline. We can do this reasoning for every $\epsilon > 0$ and for every m , so letting $\epsilon \rightarrow 0$ and $m \rightarrow \infty$ gives us that:*

There are task sets that always have a system utilization arbitrarily close to $1/2 + 1/(2m)$, but no job-static priority scheduling algorithm can meet all its deadlines.

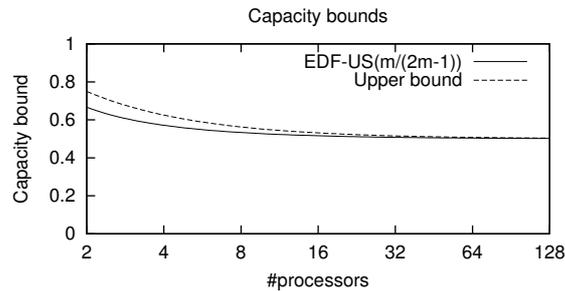


Figure 27.7: Utilization bounds for **EDF-US(m/(2m-1))** and an upper bound on the utilization bound of all job-static priority scheduling algorithms.

27.9 Partitioned scheduling

27.9.1 Introduction

In this section, we study multiprocessor scheduling algorithms and their utilization bounds for aperiodic tasks where future arrivals are unknown. We propose a job-static¹⁶ priority algorithm for tasks without migration capabilities and prove that it has a utilization bound of 0.31. No algorithm for tasks without migration capabilities can have a utilization bound greater than 0.50.

Section 27.9.2 discusses aperiodic partitioned scheduling and Section 27.9.3 presents our new results: an algorithm to assign a task to a processor and the proof of its utilization bound.

27.9.2 Partitioned scheduling

In partitioned scheduling, a task is immediately assigned to a processor when the task arrives, and the task does not migrate, effectively making a multiprocessor behave as a set of uniprocessors. Algorithms that assign a task to a processor require knowledge of whether a task can be assigned to a processor and meet its deadline. We will make use of the following result¹⁷:

Theorem 25 *Consider EDF scheduling on a uniprocessor. If $\forall t: U(t) \leq 1$, then all tasks meet their deadlines.*

Proof: Before proving this theorem, we will establish two claims:

1. If $\forall t: U(t) \leq 1$, then a uniprocessor sharing algorithm (called **OPT**) meets all deadlines.

¹⁶Recall that job-static and task-static are synonymous in aperiodic scheduling.

¹⁷A more general theorem is available in [DL97].

This follows from the observation that a processor sharing algorithm attempts to execute an arbitrary active task τ_i for $u_i \cdot \epsilon$ time units during every time interval of length ϵ within $[A_i, A_i + D_i)$. Since $\forall t: U(t) \leq 1$, OPT succeeds in executing an arbitrary task τ_i for $u_i \cdot \epsilon$ time units during every time interval of length ϵ within $[A_i, A_i + D_i)$. One such interval is $[A_i, A_i + D_i)$ and it has a length of D_i . In this interval, an arbitrary task τ_i is executed for $u_i \cdot \epsilon = u_i \cdot D_i = C_i$ time units. Hence OPT meets all deadlines.

2. If any scheduling algorithm meets all deadlines then EDF will also do so.

This follows from the optimality of EDF on a uniprocessor [Der74].

We can now reason as follows:

$$\begin{aligned} \forall t : U(t) \leq 1 & \xRightarrow{\text{use claim 1}} \\ \text{OPT meets all deadlines} & \xRightarrow{\text{use claim 2}} \\ \text{EDF meets all deadlines} & \end{aligned}$$

□

Intuitively, Theorem 25 reduces the partitioned multiprocessor scheduling problem to the design of an algorithm that assigns tasks to processors in order to keep the utilization on each processor at every moment to be no greater than 1.

When assigning tasks to processors, it is tempting to choose load balancing, but one can see that it can perform poorly in that it can miss deadlines even when only a small fraction of the capacity is requested. Example 17 illustrates that the utilization bound for load balancing is zero.

Example 17 Consider $m+1$ aperiodic tasks that should be scheduled on m processors using load balancing. We define load balancing as: assign an arriving task to a processor such that, after the task has been assigned to a processor, the utilization of the processor that has the maximum processor utilization is minimized. Let the tasks τ_i (where $1 \leq i \leq m$) have $D_i = 1, C_i = 2\epsilon$ and $A_i = i \cdot \epsilon^2$, and let the task τ_{m+1} have $D_{m+1} = 1 + \epsilon, C_{m+1} = 1$ and $A_{m+1} = (m+1) \cdot \epsilon^2$. The tasks τ_i (where $1 \leq i \leq m$) will be assigned one processor each due to load balancing. When τ_{m+1} arrives, it cannot be assigned to any processor to meet its deadline. By letting $m \rightarrow \infty$ and $\epsilon \rightarrow 0$, we have a task set that requests an arbitrarily small fraction of the capacity but still a deadline is missed.

In periodic scheduling, a common solution is to use bin-packing algorithms [DL78]. Here, a task is first tentatively assigned to the processor with the lowest index, but if a schedulability test cannot guarantee that

the task can be assigned there, then the task is tentatively assigned to the next processor with a higher index and so on. This avoids the poor performance of load balancing [OB98, LGDG00].

27.9.3 EDF-FF

We will now apply these ideas in aperiodic scheduling by proposing a new algorithm EDF-FF and analyzing its performance. Although the work by [OB98, LGDG00] proved a utilization bound for the periodic case, their proof is not easily generalized because, in our problem with aperiodic tasks, a task “disappears” when its deadline expires. When discussing EDF-FF we need to define the following concepts. The utilization of processor p at time t is $\sum_{\tau_i \in V_p} C_i/D_i$, where $V_p = \{\tau_k : (A_k \leq t < A_k + D_k) \wedge (\tau_k \text{ is assigned to processor } p)\}$. A processor p is called *occupied* at time t if there is at least one task that is both current at time t and that is assigned to processor p . A processor that is not occupied is called *empty*. Let $transition_p(t)$ be the latest time $\leq t$ such that processor p makes a transition from being empty to being occupied at time $transition_p(t)$. If processor p has never been occupied, then $transition_p(t)$ is $-\infty$.

EDF-FF means schedule tasks according to Earliest-Deadline-First on each uniprocessor and assign tasks using First-Fit. EDF-FF works as follows. When a task τ_i arrives it is assigned to the occupied processor with the lowest $transition_p(A_i)$ that passes the schedulability condition of Theorem 25. Otherwise the task is assigned to an arbitrary empty processor (if no empty processor exists then EDF-FF declares failure). Because of Theorem 25, we know that if EDF-FF does not declare failure, then all deadlines are met. If two or more tasks arrive at the same time, there is a tie in which task should be assigned first, and there could also be a tie in finding which processor is the one with the least transition. However, if there are tasks that arrive at the same time, then we can assign an order to them such that, for every pair of these tasks, we can say that one task τ_i arrives earlier than another τ_j . One such order could be to use the index of tasks. To illustrate this, consider two tasks, τ_1 and τ_2 , with $A_1 = 0$ and $A_2 = 0$. The tasks have $(C_1 = 0.6, D_1 = 1)$ and $(C_2 = 0.7, D_2 = 1)$. It is clear that EDF-FF will not assign τ_1 and τ_2 to the same processor, but if we did not have a tie breaking scheme we would not know whether EDF-FF would produce the assignment τ_1 to processor 1 (and consequently τ_2 to processor 2) or τ_2 to processor 2 (and consequently τ_1 to processor 1). Moreover, it would not be clear whether $transition_1(t) < transition_2(t)$, for $0 < t < 0.6$. To resolve this, we can choose an order such that A_1 is earlier than A_2 . This order implies that τ_1 is assigned to processor 1 and τ_2 is assigned to processor 2 and $transition_1(t) < transition_2(t)$ for $0 < t < 0.6$. In the remainder

of this section, if $A_i = A_j$, but it has been chosen that τ_i arrives before τ_j , then we will write $A_i < A_j$. The reason why this works is that the algorithm EDF-FF and its analysis does not depend on the absolute value of the arrival times; only the order is important.

Theorem 26 analyzes the performance of EDF-FF by computing its utilization bound B . We will see that $B \geq 0.31$ and hence EDF-FF performs significantly better than load balancing.

Theorem 26 *Consider scheduling on $m \geq 4$ processors using EDF-FF. Let B be a real number which is a solution to the equation*

$$m \cdot B = m \cdot (1 - B - B \cdot \ln \frac{m-1}{B \cdot m-1}) + \ln \frac{m-1}{B \cdot m-1} \quad (27.17)$$

Then we know that:

1. There is exactly one solution B , and it is in the interval $(2/m, 1]$.
2. If $\forall t: U(t) \leq m \cdot B$ then EDF-FF does not declare failure.

Proof: The theorem has two claims, so we first prove claim 1 and then prove claim 2.

1. There is exactly one solution B , and it is in the interval $(2/m, 1]$.

Let us introduce the function f (which is simply a manipulation of Equation 27.17):

$$f(B) = m \cdot (1 - 2B - B \cdot \ln \frac{m-1}{B \cdot m-1}) + \ln \frac{m-1}{B \cdot m-1}$$

We only need to prove that there is exactly one solution B to $f(B) = 0$. Noting that:

$$\begin{aligned} \frac{\partial f}{\partial B} &= -(1 + \ln \frac{m-1}{B \cdot m-1}) \cdot m < 0 \\ \lim_{B \rightarrow 2/m} f(B) &= (m-4) + (m-2) \cdot \ln(m-1) > 0 \\ \lim_{B \rightarrow 1} f(B) &= -m < 0 \end{aligned} \quad (27.18)$$

makes it possible to draw the conclusion that there is exactly one solution, and it is in the interval $(2/m, 1]$.

2. If $\forall t: U(t) \leq m \cdot B$ then EDF-FF does not declare failure.

We will first derive a lower bound of the utilization of a task that was not assigned to one of the l occupied processors with the least *transition_p*. Then we will assume that claim 2 in Theorem 26 was wrong, i.e. that there exists a task set with $\forall t: U(t) \leq m \cdot B$

for which EDF-FF declares failure. We will use the result of the lower bound of the utilization of a task to prove a lower bound on the utilization at the time when EDF-FF declared failure. This is finally used to derive a contradiction, which proves the correctness of claim 2 in Theorem 26.

A lower bound of the utilization of a task Consider a task τ_i with utilization u_i that arrives but is not assigned to the l occupied processors with the least $transition_p(A_i)$. If one of the l occupied processors had a utilization that was $1 - u_i$ or less, then τ_i would have been assigned to it, but that did not happen. Consequently, each of the l occupied processors with the least $transition_p(A_i)$ has a utilization greater than $1 - u_i$. Hence we have:

$$U(t = A_i) > l \cdot (1 - u_i) + u_i \quad (27.19)$$

From the theorem we obtain:

$$U(t = A_i) \leq B \cdot m \quad (27.20)$$

Combining Inequality 27.19 and Inequality 27.20 yields:

$$l \cdot (1 - u_i) + u_i < B \cdot m$$

Rearranging:

$$l - l \cdot u_i + u_i < B \cdot m$$

Rearranging again (for $l \geq 2$).

$$l - B \cdot m < (l - 1) \cdot u_i$$

Rearranging again (for $l \geq 2$).

$$\frac{l - B \cdot m}{l - 1} < u_i$$

We also know that $u_i > 0$. Hence we have (for $l \geq 2$):

$$\max(0, \frac{l - B \cdot m}{l - 1}) < u_i \quad (27.21)$$

A lower bound of the utilization at failure Suppose that claim 2 in Theorem 26 was wrong. Then there must exist a task set with $\forall t: U(t) \leq m \cdot B$ for which EDF-FF declares failure. Let τ_{failed} denote the task that arrived and caused the failure. If there was one processor that was empty at time $A_{failure}$, then τ_{failed} could have been assigned there and EDF-FF would not have

declared failure. For this reason, we know that all processors must have been occupied at time $A_{failure}$.

Let us choose the indices of processors so that $transition_1(A_{failed}) < transition_2(A_{failed}) < \dots < transition_m(A_{failed})$. Every task that was current at time A_{failed} and that was assigned processor j must have arrived during $[transition_j(A_{failed}), A_{failed}]$, because processor j was empty just before $transition_j(A_{failed})$, so any task that was current before $transition_j(A_{failed})$ and that was assigned to processor j must have had its absolute deadline earlier than $transition_j(A_{failed})$. When a task $\tau_{arrived}$ arrived during $[transition_j(A_{failed}), A_{failed}]$ and was assigned to processor j , there were at least $j - 1$ occupied processors ($processor_1, \dots, processor_{j-1}$) with a lower $transition_p(A_{arrived})$, so applying Inequality 27.21 (with $l = j - 1$) gives (for $j \geq 3$):

$$\max(0, \frac{j - 1 - B \cdot m}{j - 2}) < u_{arrived} \quad (27.22)$$

Since all processors are occupied at time $A_{failure}$, for every processor $j \in [1..m]$, there is at least one current task assigned to processor j that satisfies Inequality 27.22. For this reason (for $j \geq 3$) the utilization of processor j at time A_{failed} must satisfy:

$$U_j(t = A_{failed}) > \max(0, \frac{j - 1 - B \cdot m}{j - 2}) \quad (27.23)$$

where $U_j(t = A_{failed})$ denotes the utilization of processor j at time A_{failed} .

We also know that the task τ_{failed} was not assigned to the m occupied processors with the least $transition_p(A_{failed})$, so applying Inequality 27.21 with $l = m$ gives:

$$u_{failed}(t = A_{failed}) > \max(0, \frac{m - B \cdot m}{m - 1}) \quad (27.24)$$

When EDF-FF failed, the utilization of all current tasks is the same as the utilization of all current tasks that have been assigned processors plus the utilization of the task that just arrived. Hence:

$$U(t = A_{failed}) = (\sum_{j=1}^m U_j(t = A_{failed})) + u_{failed}$$

Applying Inequality 27.23 and Inequality 27.24 and using $\sum_{j=1}^m U_j(t = A_{failed}) > \sum_{j=3}^m U_j(t = A_{failed})$ yields:

$$U(t = A_{failed}) > \left(\sum_{j=3}^m \max\left(0, \frac{(j-1) - B \cdot m}{(j-2)}\right) \right) + \max\left(0, \frac{m - B \cdot m}{m-1}\right) \quad (27.25)$$

Rewriting (see Algebraic manipulation 1 in Appendix G in [And03] for details) gives us:

$$U(t = A_{failed}) > m - 1 - \sum_{k=1}^{m-1} \min\left(1, \frac{B \cdot m - 1}{k}\right) \quad (27.26)$$

It is worth noting that every term in the sum of Inequality 27.26 is non-negative because, from the first claim in the theorem, we have $B > 1/m$, which can be rewritten as $B \cdot m - 1 > 0$. We will now compute an upper bound on $\sum_{k=1}^{m-1} \min\left(1, \frac{B \cdot m - 1}{k}\right)$. Clearly we have:

$$\sum_{k=1}^{m-1} \min\left(1, \frac{B \cdot m - 1}{k}\right) = \sum_{k=1}^1 \min\left(1, \frac{B \cdot m - 1}{k}\right) + \sum_{k=2}^{m-1} \min\left(1, \frac{B \cdot m - 1}{k}\right)$$

Observing that the series: $\min\left(1, \frac{B \cdot m - 1}{k}\right)$ is non-increasing with respect to k and that $\sum_{k=1}^1 \min\left(1, \frac{B \cdot m - 1}{k}\right) = \min\left(1, \frac{B \cdot m - 1}{1}\right) \leq 1$ yields:

$$\sum_{k=1}^{m-1} \min\left(1, \frac{B \cdot m - 1}{k}\right) \leq 1 + \int_{k=1}^{m-1} \min\left(1, \frac{B \cdot m - 1}{k}\right)$$

Rewriting (see Algebraic manipulation 2 in Appendix G in [And03] for details) gives us:

$$\sum_{k=1}^{m-1} \min\left(1, \frac{B \cdot m - 1}{k}\right) \leq 1 + B \cdot m - 1 - 1 + (B \cdot m - 1) \cdot \ln \frac{m-1}{B \cdot m - 1} \quad (27.27)$$

m	4	5	6	10	100	1000	∞
B	0.38	0.36	0.35	0.34	0.32	0.31	0.31...

Table 27.2: B for different number of processors.

Using Inequality 27.27 in Inequality 27.26 yields:

$$\begin{aligned}
 U(t = A_{failed}) &> m - 1 \\
 &\quad - (1 + B \cdot m - 1 - 1 + \\
 &\quad (B \cdot m - 1) \cdot \ln \frac{m - 1}{B \cdot m - 1})
 \end{aligned}$$

Simplifying yields:

$$\begin{aligned}
 U(t = A_{failed}) &> m - B \cdot m \\
 &\quad - B \cdot m \cdot \ln \frac{m - 1}{B \cdot m - 1} + \\
 &\quad \ln \frac{m - 1}{B \cdot m - 1}
 \end{aligned}$$

Simplifying again:

$$\begin{aligned}
 U(t = A_{failed}) &> m \cdot (1 - B - B \cdot \ln \frac{m - 1}{B \cdot m - 1}) + \\
 &\quad \ln \frac{m - 1}{B \cdot m - 1}
 \end{aligned}$$

Since $U(t = A_{failed}) \leq m \cdot B$ it must have been that

$$m \cdot B > m \cdot (1 - B - B \cdot \ln \frac{m - 1}{B \cdot m - 1}) + \ln \frac{m - 1}{B \cdot m - 1}$$

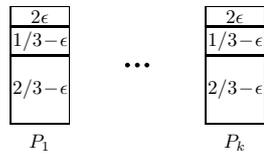
But this is impossible, because we have chosen B such that $m \cdot B = m \cdot (1 - B - B \cdot \ln \frac{m-1}{B \cdot m - 1}) + \ln \frac{m-1}{B \cdot m - 1}$.

□

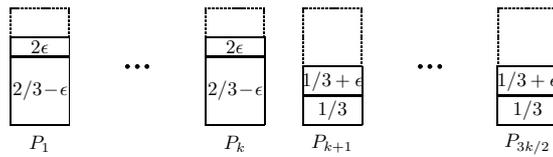
Different values of B are shown in Table 2. When m approaches infinity, then B is the solution to $1 - 2B + B \ln B = 0$. This is where our utilization bound of 0.31 came from.

Our analysis of utilization bounds of EDF-FF is not necessarily tight but one can see that no analysis of utilization bounds of EDF-FF can, in general, obtain a utilization bound that is greater than 0.42. This is illustrated in Example 18.

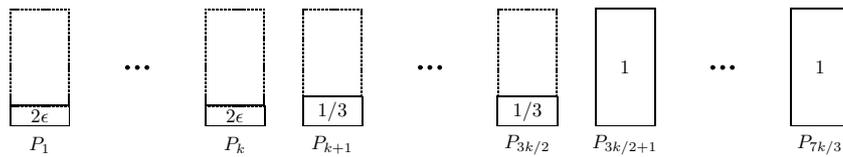
Example 18 (Adapted from [CGJ83]) Let $m = 7k/3$, where k is divisible by 6. First k tasks with $u_i = 2/3 - \epsilon$ arrive, followed by k



(a) Tasks arrive and are assigned to processors P_1, \dots, P_k , but the deadlines of tasks have not yet expired.



(b) The deadlines of tasks with utilization $1/3 - \epsilon$ expire and new tasks arrive and are assigned to processors $P_{k+1}, \dots, P_{3k/2}$.



(c) The deadlines of tasks expire and tasks with utilization one arrive and are assigned to processors $P_{3k/2+1}, \dots, P_{7k/3}$.

Figure 27.8: No analysis of EDF-FF can give a utilization bound greater than 0.42.

tasks with $u_i = 1/3 - \epsilon$ and then k tasks with $u_i = 2\epsilon$. Figure 27.8(a) illustrates the packing of current tasks.

The deadline of tasks with $u_i = 1/3 - \epsilon$ expires, so we remove them. $k/2$ tasks with $u_i = 1/3$ and $k/2$ tasks with $u_i = 1/3 + \epsilon$ arrive in the sequence $1/3, 1/3 + \epsilon, 1/3, 1/3 + \epsilon, \dots, 1/3, 1/3 + \epsilon$. Figure 27.8(b) illustrates the packing of current tasks.

The deadline of tasks with $u_i = 2/3 - \epsilon$ expires, so we remove them. The deadline of tasks with $u_i = 1/3 + \epsilon$ expires, so we remove them too. $5k/6$ tasks with $u_i = 1$ arrive. Now, at least one task is assigned to every processor. Figure 27.8(c) illustrates the packing of current tasks. Finally, a task with $u_i = 1$ arrives, but it cannot be assigned to any processor, so EDF-FF fails. If we let $\epsilon \rightarrow 0$ and $k \rightarrow \infty$ (and consequently $m \rightarrow \infty$), then we have that $\forall t : U_s(t) \leq 3/7$, but EDF-FF still fails.

Although one could design other partitioning schemes, those are unlikely to offer any significant performance improvements. This is because in dynamic bin-packing (where items arrive and depart), it is known that first-fit offers performance (in terms of competitive ratio) not too far from the best that one could hope for [CGJ83, page 230]. Nevertheless, it is known [LGDG00] that no partitioned multiprocessor scheduling algorithm can achieve a utilization bound greater than 0.5.

For computer platforms where task migration is prohibitively expensive, one could conceive of other approaches than partitioning. For example, one could conceive of scheduling algorithms where an arriving task is assigned to a global runnable queue, and when it has started to execute, it is assigned to that processor and does not migrate. However, such approaches suffer from scheduling anomalies [HL94] and they still cannot achieve a utilization bound greater than 0.50 (to see this, consider the same example as given in [LGDG00]).

We conclude that, although it may be possible to design and analyze scheduling algorithms that do not migrate tasks and make these algorithms achieve greater utilization bounds, these algorithms will not improve the performance as much as the EDF-FF did (from 0 to 0.31).

27.10 Conclusions

This chapter addressed the problem:

How much of the capacity of a multiprocessor system can be requested without missing a deadline when static-priority preemptive scheduling is used?

To this end, scheduling algorithms were designed and their performance were proven. The results imply the following:

11. **Global static-priority scheduling on multiprocessors is worth considering in multiprocessor-based real-time systems.** Before this research was performed, it was believed [Dha77, DL78] that global static-priority scheduling was unfit because global static-priority scheduling with the rate-monotonic priority assignment has a utilization bound of zero. With a new priority assignment scheme, we have seen that a greater utilization bound of $1/3$ can be achieved. The new priority-assignment scheme was based on the RM-US approach. After that, the RM-US approach has been used [Lun02] to design a better priority assignment scheme for periodic scheduling with a utilization bound of 0.37, that is, the greatest utilization bound that a global static-priority scheduling algorithm can have that is based on the RM-US approach. Of

course, other priority assignment schemes are conceivable, but we saw that a class of algorithms, which seem to cover all possible simple ones, have a utilization bound of 0.41 or lower. This implies two interesting topics for future research: design a simple algorithm that achieves a utilization bound of 0.41; design a non-simple algorithm that achieves the utilization bound of 0.50.

- I2. **In partitioned scheduling, dynamic priorities in uniprocessor scheduling do not offer any increase in utilization bounds as compared to static priorities.** A static-priority partitioned scheduling algorithm were presented that could achieve a utilization bound of 0.50, and since every partitioned scheduling algorithm has a utilization bound of 0.50 or less, we can conclude that dynamic priorities do not offer any increase in utilization bounds as compared to static priorities.
- I3. **Anomalies exist in many preemptive multiprocessor scheduling algorithms.**

Several examples showed that anomalies exist in many preemptive multiprocessor scheduling algorithms. However, it is possible to design an anomaly-free partitioned scheduling algorithm with a utilization bound of 0.41. This clearly implies that if the utilization is low enough and this algorithm is used then anomalies can be avoided. The question of whether it is possible to design an anomaly-free partitioned scheduling algorithm with a utilization bound of 0.50 was left open. If the answer is yes, then it shows that, for a scheduling algorithm, the requirement of having a high utilization bound does not conflict with the requirement of being anomaly-free.

Bibliography

- [And03] B. Andersson. *Static-priority scheduling on multiprocessors*. Ph.D. thesis, School of Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden, September 2003. Defended on Sept 29, 2003.
- [ATB93] N. Audsley, K. Tindell, and A. Burns. The end of the line for static cyclic scheduling? In *Proc. of the EuroMicro Workshop on Real-Time Systems*, pages 36–41, Oulu, Finland, June 22–24, 1993.
- [Aud91] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, Dept. of Computer Science, University of York, York, England YO1 5DD, December 1991.
- [BCPV96] S. Baruah, N. Cohen, G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, June 1996.
- [BHS94] S. Baruah, J. Haritsa, and N. Sharma. On-line scheduling to maximize task completions. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 228–237, San Juan, Puerto Rico, 1994.
- [BKM⁺91] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. On-line scheduling in the presence of overload. In *Proc. of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 100–110, San Juan, Puerto Rico, October 1991.
- [BKM⁺92] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4(2):125–144, June 1992.
- [BL98] S. Baruah and S.-S. Lin. Pfair scheduling of generalized pinwheel task systems. *IEEE Transactions on Computers*, 47(7):812–816, July 1998.
- [BLOS95] A. Burchard, J. Liebeherr, Y. Oh, and S.H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers*, 44(12):1429–1442, December 1995.
- [BTW95] A. Burns, K. Tindell, and A. Wellings. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Trans. on Software Engineering*, 21(5):475–480, May 1995.
- [CGJ83] E. G. Coffman, M. R. Garey, and D. S. Johnson. Dynamic bin packing. *SIAM Journal of Applied Mathematics*, 12(2):227–258, May 1983.
- [DD85] S. Davari and S.K. Dhall. On a real-time task allocation problem. In *19th Annual Hawaii International Conference on System Sciences*, pages 8–10, Honolulu, Hawaii, 1985.
- [DD86] S. Davari and S.K. Dhall. An on-line algorithm for real-time task allocation. In *Proc. of the IEEE Real-Time Systems Symposium*, volume 7, pages 194–200, New Orleans, LA, December 1986.

- [Der74] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *IFIP Congress*, pages 807–813, 1974.
- [Dha77] S. Dhall. *Scheduling Periodic-Time-Critical Jobs on Single Processor and Multiprocessor Computing Systems*. Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1977.
- [DL78] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, January/February 1978.
- [DL97] Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *Proc. of the IEEE Real-Time Systems Symposium*, volume 18, pages 308–319, San Francisco, California, December 3–5, 1997.
- [DM89] M. L. Dertouzos and A. K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Trans. on Software Engineering*, 15(12):1497–1506, December 1989.
- [Fid98] C. J. Fidge. Real-time schedulability tests for preemptive multi-tasking. *Real-Time Systems*, 14(1):61–93, January 1998.
- [GL89] D. W. Gillies and J. W.-S. Liu. Greed in resource scheduling. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 285–294, Santa Monica, California, December 5–7, 1989.
- [Goo03] J. Goossens. Scheduling of offset free systems. *Real-Time Systems*, 24(2):239–258, March 2003.
- [Gra69] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, March 1969.
- [Gra72] R. L. Graham. Bounds on multiprocessor scheduling anomalies and related packing problem. In *Proceedings AFIPS Spring Joint Computer Conference*, pages 205–217, 1972.
- [HL94] R. Ha and J. W.-S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proc. of the IEEE Int'l Conf. on Distributed Computing Systems*, pages 162–171, Poznan, Poland, June 21–24, 1994.
- [Hor74] W. A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21(1):177–185, March 1974.
- [JP86] M. Joseph and P. Pandya. Finding response times in a real-time system. *Computer Journal*, 29(5):390–395, October 1986.
- [KAS93] D. I. Katcher, H. Arakawa, and J. K. Strosnider. Engineering and analysis of fixed priority schedulers. *IEEE Trans. on Software Engineering*, 19(9):920–934, September 1993.
- [KP95] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *36th Annual Symposium on Foundations of Computer Science*, pages 214–223, Milwaukee, Wisconsin, 1995.
- [Kuh62] T. S. Kuhn. *The structure of scientific revolutions*. 1962.

- [LDG01] J. M. López, J. L. Díaz, and D. F. García. Minimum and maximum utilization bounds for multiprocessor RM scheduling. In *Proc. of the EuroMicro Conference on Real-Time Systems*, pages 67–75, Delft, The Netherlands, June 13–15 2001.
- [LGDG00] J.M. López, M. García, J.L. Díaz, and D.F. García. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Proc. of the 12th EuroMicro Conference on Real-Time Systems*, pages 25–33, Stockholm, Sweden, June 19–21, 2000.
- [LL73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.
- [LL03] L. Lundberg and H. Lennerstad. Global multiprocessor scheduling of tasks using time-independent priorities. In *Proc. of the IEEE Real Time Technology and Applications Symposium*, volume 9, Washington, DC, May 27–30, 2003.
- [LM81] E. L. Lawler and C. U. Martel. Scheduling periodically occurring tasks on multiple processors. *Information Processing Letters*, 12(1):9–12, February 1981.
- [LMM98a] S. Lauzac, R. Melhem, and D. Mossé. Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. In *Proc. of the EuroMicro Workshop on Real-Time Systems*, pages 188–195, Berlin, Germany, June 17–19, 1998.
- [LMM98b] S. Lauzac, R. Melhem, and D. Mossé. An efficient RMS admission control and its application to multiprocessor scheduling. In *Proc. of the IEEE Int'l Parallel Processing Symposium*, pages 511–518, Orlando, Florida, March 1998.
- [LS99] T. Lundqvist and P. Stenström. Timing anomalies in dynamically scheduled microprocessors. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 12–21, Scottsdale, Arizona, December 1–3, 1999.
- [LSD89] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average behavior. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 166–171, Santa Monica, California, December 5–7, 1989.
- [Lun98] L. Lundberg. Multiprocessor scheduling of age constraint processes. In *Proc. of the International Conference on Real-Time Computing Systems and Applications*, pages 42–47, Hiroshima, Japan, October 27–29, 1998.
- [Lun02] L. Lundberg. Analyzing fixed-priority global multiprocessor scheduling. In *Proc. of the IEEE Real Time Technology and Applications Symposium*, volume 8, pages 145–153, San José, California, September 24–27, 2002.
- [LW82] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, December 1982.

- [MFFR02] P. Marti, J. M. Fuertes, G. Fohler, and K. Ramamritham. Improving quality-of-control using flexible timing controls: Metric and scheduling issues. In *Proc. of the IEEE Real-Time Systems Symposium*, Austin, Texas, December 3–5, 2002.
- [Mok83] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1983.
- [Mok00] A. K. Mok. Tracking real-time systems requirements, December 12–14, 2000. Invited talk at the International Conference on Real-Time Computing Systems and Applications.
- [Mur88] F. D. Murgolo. Anomalous behavior in bin packing algorithms. *Discrete Applied Mathematics*, 21(3):229–243, October 1988. North Holland.
- [OB98] D. Oh and T. P. Baker. Utilization bounds for n -processor rate monotone scheduling with static processor assignment. *Real-Time Systems*, 15(2):183–192, September 1998.
- [OS95a] Y. Oh and S. H. Son. Allocating fixed-priority periodic tasks on multiprocessor systems. *Real-Time Systems*, 9(3):207–239, November 1995.
- [OS95b] Y. Oh and S. H. Son. Fixed-priority scheduling of periodic tasks on multiprocessor systems. Technical Report 95-16, Department of Computer Science, University of Virginia, March 1995.
- [PSTW97] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *Proc. of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 140–149, El Paso, Texas, May 4–6 1997.
- [SB02] A. Srinivasan and S. Baruah. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84(2):93–98, October 2002.
- [SRS95] C. Shen, K. Ramamritham, and J. A. Stankovic. Resource reclaiming in multiprocessor real-time systems. *IEEE Trans. on Parallel and Distributed Systems*, 4(4):382–397, April 1995.
- [SVC98] S. Sáez, J. Vila, and A. Crespo. Using exact feasibility tests for allocating real-time tasks in multiprocessor systems. In *10th Euromicro Workshop on Real Time Systems*, pages 53–60, Berlin, Germany, June 17–19, 1998.

Part VI
Real-Time and Control

Chapter 28

Introduction

By **Karl-Erik Årzén**

Department of Automatic Control

Lund University

Email: karlerik@control.lth.se

Real-Time Systems as a research discipline has always had strong connections to control engineering. A lot of the early work in real-time systems in the beginning of the 1970s was motivated by feedback control applications. Today almost all controllers are implemented on digital form in a computer using ASICs and FPGAs. A successful implementation of a computer control system requires a good understanding of both control theory and real-time systems.

28.1 Control Implementation

Task modeling is one area where control applications often are used as a motivating example. By tradition the hard task model is commonly associated with control applications, i.e., periodic tasks with a fixed period, a known worst-case bound on the execution time (WCET), and a hard deadline. This task model has contributed to providing a separation between the control community and the real-time scheduling community. The separation has allowed the control community to focus on its own problem domain without worrying about how scheduling is being done, and it has released the scheduling community from the need to understand what impact scheduling has on the stability and performance of the plant under control. From a historical perspective, the separated development of control and scheduling theories for computer-based control systems has produced many useful results and served its purpose. However, the separation has also had negative effects. The

two communities have partly become alienated. This has led to a lack of mutual understanding between the fields.

Upon closer inspection it is quite clear that many of the assumptions of the simple model are too restrictive. First, the assumptions do not allow us to use low-cost general purpose hardware and off-the-shelf operating systems, which in general are not able to give any guarantees on determinism. These systems are, typically, designed to achieve good average performance rather than guaranteed worst-case performance. They often introduce significant non-determinism in task scheduling. For computation intensive high-end applications, the large variability in execution time caused by modern hardware architecture also becomes visible. The effect of this on the control loop is jitter in sampling period and control delay (input-output latency).

The assumptions of the simple model are also overly restrictive with respect to the characteristics of many control loops. Many control loops are not periodic, or they may switch between a number of different fixed sampling periods. Control loop deadlines are not always hard. The hard task model is seldom something that is imposed by the plant under control. As all control engineers know, most plants are quite forgiving with respect to jitter in sampling periods and input-output latency. The use of feedback further increases the temporal robustness of the system. Instead it is often the control designer which imposes the hard task model. This allows him to apply the theory of sampled, or digital, control systems, which assumes periodic sampling and constant input-output latencies, to provide guarantees on stability and performance. It is also in many cases possible to compensate on-line for the variations by, e.g., recomputing the controller parameters. Obtaining an accurate value for the WCET is generally a difficult problem. Measuring WCET always implies the risk of underestimation, whereas analytical execution time analysis tools are still rare. An alternative may be to instead measure the actual execution time every task invocation and to adjust the task parameters accordingly. Finally, it is also possible to consider control systems that are able to do a tradeoff between the available computation time, i.e., how long time the controller may spend calculating the new control signal, and the control loop performance.

There is a need for more general scheduling models that better fit the nature and needs of advanced control algorithms. The optimality of computer control is subject to the limitations of available computing resources, especially in advanced applications where we want to control fast plant dynamics and to use sophisticated state estimation and control algorithms. On the other hand, the true objective of real-time scheduling for control is to allocate limited computing resources in such a way that the state estimation and control algorithms can ensure the system's stability and optimize the system's performance.

There is also a need for an increased understanding of which level of determinism that a given control loop requires in order to meet given control objectives. Is it necessary to use a time-triggered approach or will an event-based approach perform satisfactorily? How large latencies and what level of latency jitter can be tolerated? Is it OK to now and then skip a sample. These issues are strongly related to aperiodic systems and event-based sampling. Analysis of event-based sampled systems is considerably harder than for time-based sampled systems. For certain system setups, it has however been shown that event-based sampling can be more efficient than time-based sampling. Systems with mixed continuous and discrete variables, such as event-triggered sampled systems, are also studied in *hybrid control systems*. This is one of the most rapidly increasing fields in control theory today.

28.2 Co-Design Tools

The resource constraints in, e.g., small embedded applications, increase the need for co-design of the control system and computing system, including integration of control and real-time scheduling design. Co-design of control and computing systems is not a new topic. In the early days of computer control limited computer resources was a general problem, not only a problem for embedded controllers. For examples, the issues of limited word length, fixed-point calculations, and the results that this has on resolution was something that was well-known among control engineers in the 1970s. However, as computing power has increased these issues have received decreasing attention.

In order for integrated control and real-time scheduling to be a reality it is necessary to have computer tools for simulation, analysis, and synthesis. During recent years such tools have begun to emerge. These tools makes it possible to analyze how the temporal nondeterminism effect control performance and they allow co-simulation of controllers, the plant under control, the real-time computing system, and the network communication, down to a granularity of individual context switches and network packet arrivals.

28.3 Control of Real-Time Computing Systems

Another area where control and real-time systems intersects is for control of real-time computing systems. Applications of real-time computing have gradually evolved from closed embedded systems to complex, distributed open heterogeneous platforms operating in unpredictable poorly modeled environments such as, e.g., the Internet. Hard guarantees are impractical on such platforms since load and resource capacities

are very difficult to predict. Yet, many modern applications require some form of performance assurances, which may include guarantees on timeliness, bandwidth, data consistency, or jitter. Traditional approaches for providing these performances, such as resource pre-allocation and a priori knowledge of worst case execution conditions are no longer applicable.

Feedback control is a well-established and mathematically well-founded theory that is ideal for handling uncertainties. Using control-based approaches for modeling, analysis, and design of real-time computing systems is currently receiving increased attention as a promising foundation for controlling the uncertainty in large and complex real-time systems. Areas that are studied include feedback architectures for adaptive real-time computing, theory for performance guarantees under uncertainty, control-theoretical models of dynamic real-time systems, application of control theory for controlling timing behavior, and optimal, robust, or adaptive feedback control in real-time systems. The use of control has the potential to increase flexibility, while preserving dependability and efficiency. The area where most work currently is being done is for control of resource allocation, with control of web-servers as a common example.

28.4 Control in ARTES

The intersection between control and real-time computing has also been an important area within ARTES. The focus of the project “Integrated Control and Scheduling” has been methods for practical management of hard real-time demands in embedded real-time control software. Two approaches have been followed. One idea is to combine control theory and scheduling theory in such a way that the nominal requirements on hard deadlines for control systems can be relieved. The approach taken is based on using dynamic feedback from the scheduler to the controllers and from the controllers to the scheduler. The second, complementary, idea is to use attribute grammars and incremental semantic analysis to carry out on-line interactive analysis of the worst-case timing of the software.

The project “Real-time software for versatility, scalability and reconfigurability in complex embedded feedback control systems” has had the goal to provide a software architecture suitable for scalable, maintainable and reconfigurable mechatronic systems controlled by an embedded distributed computer system. The project is focused on multi-degree-of-freedom mechanical systems of a complex configuration, found in e.g. legged locomotion systems. The specific characteristics of the software architecture are its ability to support hierarchies of control levels; ex-

tensive mode shifting; predictable real-time execution, synchronization and communication; reconfiguration including scaling of the mechanical system, and, finally, partitioning and allocation of software functions for close to optimal resource utilization.

In the project “Automatic Control in Distributed Applications (AIDA2)” the goals have been to provide a modeling framework, a prototype tool-set and an accompanying method to support the design of machine control systems that are increasingly implemented in embedded distributed computer systems. The automotive industry is facing an increasing extent of functionality that is implemented in software on interconnected electronic control units. The project “Functional Integration and Interference in Embedded Control Systems” has investigated functional integration and interference from both a control functionality and computer implementation perspective. The project has studied control methods, architectures, and real-time system services by which different control functions can be integrated in a cooperative manner to achieve a common goal, including methods for integration of and arbitration between control functions of different safety criticality.

28.5 Article Overview

This section contains two articles covering some of the subjects from the control-related projects in ARTES.

The first paper, “Tools for Real-Time Control System Co-Design” by Henriksson, Redell, El-Khoury, Cervin, Törngren, and Årzén provides an overview of the co-design tools that have been developed in the ARTES projects. The paper gives a brief overview of four tools/toolkits: Jitterbug, TrueTime, AIDA, and XILO. Jitterbug and TrueTime are two Matlab-based toolboxes for analysis and simulation of real-time control systems from Lund University. The tools can be used at early design stages to determine how sensitive controllers are to scheduling-induced delays and jitter. They can also be used at the implementation stage for trade-off analysis between the tasks. Furthermore, TrueTime can be used as an experimental platform for research on flexible scheduling. At KTH the AIDA tool-set and XILO have been developed for analysis and design of networked embedded control systems. The tool-set is based on a modeling framework allowing functional requirements and various implementation abstractions to be represented. AIDA supports end-to-end timing behavior and facilities for fault injection and robustness experiments.

The second paper, “The Control Server Model for Codesign of Real-Time Control Systems” by Cervin and Eker, presents the control server, a real-time scheduling mechanism tailored to control and signal process-

ing applications. A control server creates the abstraction of a control task with a specified period and a fixed input-output latency shorter than the period. Individual tasks can be combined into more complex components without loss of their individual guaranteed fixed latency properties. I/O occurs at fixed predefined points in time, at which inputs are read or controller outputs become visible. The control server model is especially suited for co-design of real-time control systems. The single parameter linking the scheduling design and the controller design is the task utilization factor. The proposed server is an extension of the constant bandwidth server, which is based on the earliest-deadline-first scheduling algorithm.

Chapter 29

Tools for Real-Time Control System Co-Design

By **Dan Henriksson, Ola Redell, Jad El-Khoury, Anton Cervin,
Martin Törngren, and Karl-Erik Årzén**

Department of Automatic Control
Lund University
Email: karlerik@control.lth.se

A survey of four co-design tools for joint analysis, simulation, and design of computer-based control systems developed within the Swedish ARTES programme is presented. The tools allow simultaneous treatment of the control aspects and the computing and communication aspects of the control problems. The tools are Jitterbug and TrueTime developed at the Department of Automatic Control LTH, Lund University, and Aida and Xilo developed at the Division of Mechatronics, Department of Machine Elements, KTH.

29.1 Introduction

Control systems are becoming increasingly complex from the perspectives of both control and computer science. Today, even seemingly simple embedded control systems often contain a multitasking real-time kernel and support networking. At the same time, the market demands that the cost of the system be kept at a minimum. Hence, many embedded control systems are subject to resource constraints, manifesting itself in limited CPU speed, memory, and network bandwidth of the target platform. In addition, a strong trend within industry today is to use commercially available information technology and commercial-off-the-shelf (COTS) components deeper and deeper in the real-time control systems.

Many computer-controlled systems are distributed systems consisting of computer nodes and a communication network connecting the various systems. It is not uncommon for the sensor, the actuator, and the control calculations to reside on different nodes in the system. One prominent example of this is modern automotive systems, which contain several embedded ECUs (electronic control units) used for various feedback control tasks, such as engine performance control, anti-lock braking, active stability control, exhaust emission reduction, and cruise control.

Within the individual nodes in the networked control loops, the controllers are often implemented as one or several tasks on a microprocessor with a real-time operating system. Often the microprocessor also contains tasks for other functions (e.g., communication and user interfaces). The operating system typically uses multiprogramming to multiplex the execution of the various tasks. The CPU time and the communication bandwidth can hence be viewed as shared resources for which the tasks compete.

Limited resources combined with non-optimized hardware and software components introduce nondeterminism in the real-time system. Digital control theory normally assumes equidistant sampling intervals and a negligible or constant control delay from sampling to actuation. However, this can seldom be achieved in practice in a resource-constrained system. For control systems this is of particular concern. Timing variations in sampling periods and latencies degrade the control performance and may in extreme cases lead to instability.

For optimal use of computing resources, the control algorithm and the control software designs need to be considered at the same time. For this reason, new, computer-based tools for real-time and control co-design are needed. The purpose of this article is to describe four such tools: Aida, JitterBug, TrueTime, and Xilo, which all have been developed within the ARTES programme. All four tools are based upon Matlab/Simulink from Mathworks.

29.1.1 A Historical Perspective

The need to support efficient digital implementation of control systems and co-design of control systems with the main implementation technology, real-time computer systems, became apparent during the 1970s and 1980s as microprocessor technology appeared and became more mature. Early efforts on real-time implementation environments and code generation can be found in the 1980s in the conferences Computer Aided Control Engineering (often called Computer Aided Control Systems Development, [1]).

As computer-aided engineering tools improved, it also became possible to develop tool support environments. Examples of relatively early efforts which in some way address real-time implementation of control systems include

- The Development Framework [2], which combined and to some extent integrated control design (in Simulink) with software engineering capabilities using a CASE tool (Software through Pictures)
- The GRAPE tool-set [3], which although developed for digital signal processing systems, provides similar capabilities and supports distributed systems (allocation, scheduling, partitioning)
- Efforts by Honeywell labs including MetaH and the Parallel Scalable Design Tool-set (PSDT) [4, 5]

In addition, in the real-time research community, a number of prototypical tools have been developed for schedule simulation, timing analysis and schedule generation, [6, 7].

29.1.2 Related Work

While numerous other tools exist that support either simulation of control systems (e.g., MATLAB/Simulink [8], Dymola [9]) or scheduling algorithms (e.g., STRESS [6], DRTSS [7], RTSIM [10]), very few tools have been developed that support co-simulation of control systems and real-time scheduling. The RTSIM simulator [10] has been extended with a numerical module (based on the Octave library) that supports simulation of continuous dynamics [11]. Compared to, e.g., TrueTime, there is no graphical control systems editor in this tool.

Ptolemy II is the third generation of software produced within the Ptolemy project [12, 13] at the University of California at Berkeley. Ptolemy II supports *heterogeneous*, *hierarchical* modeling, simulation, and design of concurrent systems, especially embedded systems. The focus is on complex systems mixing various technologies and operations. Simulation models are constructed under *models of computation* that govern the interaction of the components in the model. Different models of computation are used for modeling different types of systems. The abstraction provided by the model of computation also simplifies code generation from the Ptolemy models. The *timed multitasking* (TM) domain [14] adds the possibility to model fixed-priority scheduling of tasks with fixed execution times.

Orccad [15, 16, 17, 18] is a CAD system and approach aimed at the development of robotic systems from high-level specifications down to the implementation details. It deals with hybrid systems where

continuous-time aspects relating to control laws, must be merged with discrete-time aspects related to control switches and exception handling.

The Syndex tool supports rapid prototyping of reactive data-driven algorithms implemented on distributed heterogeneous hardware architectures [19, 20, 21, 22]. Syndex lets the user specify both the algorithm and the distributed hardware in a graphical environment, and then automates the mapping and scheduling of functions (called operations) and communications (called transfers) on the processors (operators) and communication buses (transformators). During the mapping and scheduling process, the hardware architecture can be refined to better match the algorithm needs. When a sufficiently good solution has been found, Syndex generates executable code that can be downloaded to the target hardware.

A more extensive survey of some of the tools above is available in [23].

29.1.3 Article Outline

The remainder of the article is organized as follows. Sections 29.2-29.5 describe the different tools mentioned above. Each tool is presented by an introductory overview that describes the main use and intentions of the tool. Each tool is visualized by a simple example. The overview is followed by a more detailed description of various aspects of the tool.

29.2 AIDA

The Aida toolset [24] is an environment for model-based design and analysis of real-time control systems. The most important feature of Aida is that it allows a user to take implementation effects into consideration when analyzing the performance of an automatic control system. Considered implementation effects include delays and time variations in the execution and scheduling of control functions and communication of data. The toolset also supports timing analysis of the real-time design such that an implemented solution can be shown to be schedulable and meet its timing constraints.

The toolset consists of a modelling environment, *Aidasign*, which interfaces with MATLAB/Simulink [25], and a response time analysis tool, *Aidalyze*. In the toolset, a controller is designed using MATLAB/Simulink, which is an environment familiar to control engineers that supports simulation based analysis of control performance. The real-time system design starts with the translation of the Simulink model to a *data-flow diagram* (DFD) in *Aidasign*. The timing aspects of the controller, such as sampling periods and delays then constitute requirements on the real-time system design. The functions and communica-

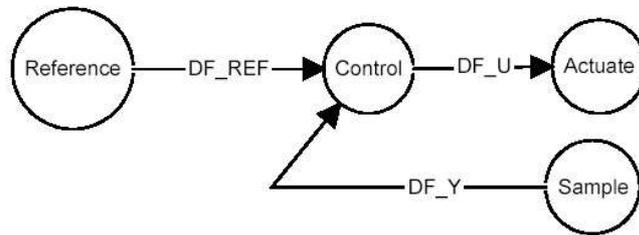


Figure 29.1: An example of an AIDA data flow diagram.

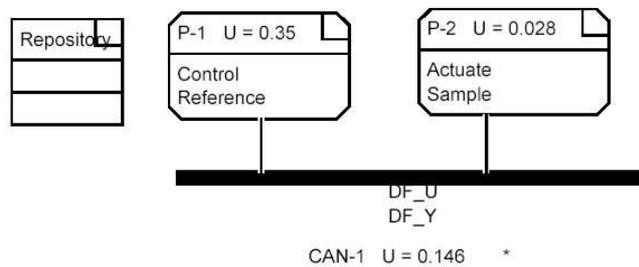


Figure 29.2: An example of an AIDA hardware structure diagram.

tion flows specified in the data-flow diagram form the basis for all further modelling in Aida. Apart from being generated from Simulink models, data-flow diagrams can be specified completely or in parts within Aida. Figure 29.1 shows an example of a data-flow diagram with four functions and the related data flows connecting them.

Another fundamental model in Aida is the *hardware structure diagram* (HSD), where the hardware architecture, in terms of processors and their interconnections via communication links, is designed. In the HSD the functions and data flows in the associated data-flow diagram(s) are mapped to processors and communication links, respectively. Figure 29.2 shows an HSD with two processors ($P-1$ and $P-2$) that are interconnected via a CAN-bus ($CAN-1$). The mapping of functions and data flows in Figure 29.1 is visualised. The utilisation (U) of each component is computed based on underlying models and the repository is used to temporarily store functions and data-flows that have not yet been mapped to any component.

Based on these two fundamental models and the mapping between them, a real-time implementation is designed. The design includes specification of operating system processes; their inter-communication in terms of messages; mapping of messages to CAN-frames; and triggering of process executions.

When the real-time system design has been completed, upper and lower bounds on the response times and release/response jitter (variations in release and response times) of the functions, processes and inter-process communications can be derived using the Aidalyze tool. These results can then be exported back to the control domain in the form of a Simulink model augmented with timing and execution order information. Hence, timing effects due to implementation can be incorporated in the control performance analysis through simulation of the generated Simulink diagram.

29.2.1 Development Scenarios Supported

AIDA is intended for one particular development scenario, but sub-scenarios can be followed as well. The major scenario starts in the control system design tool MATLAB/Simulink in which the data-flows in the control system are specified. The Simulink model is then imported to the Aida toolset in which a data-flow representation of the system is automatically generated. The data-flow model is augmented by the user with estimates of best- and worst-case execution times for functions and communication needs for the data-flows. The resulting model is the base for all other models in the tool-set.

Next, a real-time implementation of the control system is described using the models available in Aida. Given the model description of the implementation, a response time analysis is performed producing bounds on the response times of functions and processes. Finally, a transformed Simulink model can be generated, including delays according to the response time analysis results. The Simulink model can be used in simulation to test the control performance given the implementation induced delays.

29.2.2 Modeling Content

Apart from the modelling capabilities of Simulink, the Aida toolset includes the following models:

- *Data flow diagram* (DFD). Functions are specified and connected by data flow relations in the DFD. A function is parameterized by its minimum and maximum execution times while a data flow is simply described with the number of bytes that it communicates. See Figure 29.1.
- *Function timing and triggering diagram* (FTTD). The FTTD is used to describe the sequences of precedence-related functions (the control flow) and the triggering of such sequences using periodic (time) or aperiodic (event) triggers. Figure 29.3 shows an FTTD

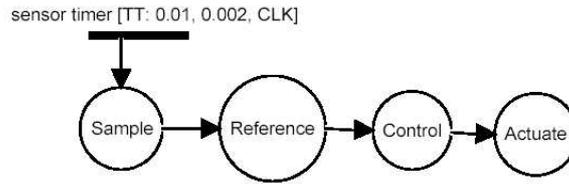


Figure 29.3: An example of an AIDA function timing and triggering diagram.

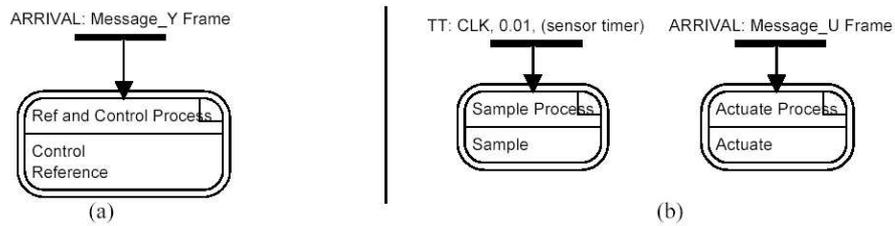


Figure 29.4: Process timing and triggering diagrams for processors P-1 (a) and P-2 (b) of Figure 29.2.

where the execution sequence of the functions in the example application of Figure 29.1 have been specified. The diagram also shows a time trigger (TT) named *sensortimer* used to trigger the execution of the sample function. The parameters of the trigger are: the period (0.01); the admissible jitter (0.002); and the name of the source clock (CLK). The FTTD can be interpreted together with the DFD as a way to set the requirements on the implementation and does not directly specify any part of the implementation. FTTDs are therefore not necessary for a complete system description.

- *Hardware structure diagram (HSD)*. As described above, the HSD is used to specify the hardware architecture as a network of processors interconnected by CAN buses. Processors are parameterized by a speed factor, used to scale the execution time of allocated functions. CAN buses are also associated with speed parameters, defining the communication speed on the bus. Furthermore, functions and data-flows are mapped to processors and buses in the HSD, as shown in Figure 29.2.
- *Process timing and triggering diagram (PTTD)*. For each processor in an HSD there is a PTTD that describes the triggering of the contained processes' execution. Process execution may be triggered by a precedence relationship (completion of another process); by

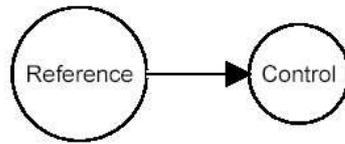


Figure 29.5: The process internal timing and triggering diagram of the Ref and Control process.

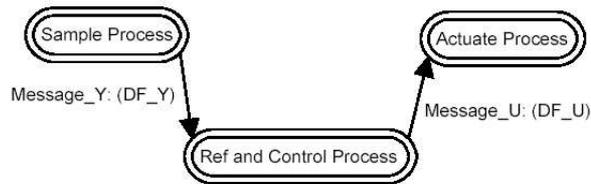


Figure 29.6: The process structure diagram for the example system.

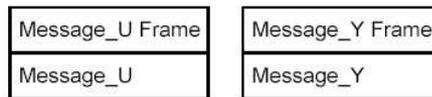


Figure 29.7: A communication link diagram defining the CAN-frames in the system.

the arrival of a CAN frame; or by time or event triggers. The PTTD is also used to specify the processes by mapping functions in a processor to different processes. A process is assigned a fixed priority for scheduling. Figure 29.4 shows the PTTDs for the two processors in Figure 29.2. The execution of the sample process is triggered by a time trigger with period 0.01 while the other two processes are triggered by arriving CAN-frames.

- *Process internal timing and triggering diagram (PiTTD)*. The PiTTD is used to define the execution sequence of functions within a process. It simply relates the functions included in a process in precedence order. Figure 29.5 shows the very simple PiTTD for the Ref and Control process executing in processor P-1.
- *Process structure diagram (PSD)*. The PSD is basically an implementation version of the DFD. It defines how processes communicate via messages. The messages are composed of data flows that are communicated between functions in different processes. Many data flows may be included in a single message, if these data flows have the same sending and receiving processes. The PSD for the

example application is shown in Figure 29.6. It defines two inter-process messages: `Message_U` and `Message_Y`.

- *Communication link diagram* (CLD). In the CLD the messages that were defined in the PSD are distributed on different CAN frames. One frame may include more than one message, but no more than 8 bytes in total. Figure 29.7 shows how the messages defined in Figure 29.6 are allocated to two different CAN frames. The arrivals of these frames are used to trigger the execution of the processes in the PTTDs in Figure 29.4.

The Simulink models are automatically transformed to Aida data-flow models when imported, and timing-augmented Simulink models are automatically generated from the Aida models.

The included tool Aidalyze may be used to perform response time analysis when an implementation model has been completely specified.

A consistency check, verifying the consistency of the information that is represented in multiple different Aida models, is performed when the user invokes an "update"-function for either model in Aidassign.

29.2.3 Implementation

Aidassign is completely developed in the Domain Modelling Environment (DoME) from Honeywell. DoME is a tool for development of new modelling languages in which new models are easily added. Hence, Aidassign is easily extended with more models when needed. Furthermore, tools performing automated tasks on the models, such as for example mapping of functions to processors, can easily be written in the Alter language which is an integral part of DoME.

Aidalyze is written in C++ for performance reasons. The source code is available and more algorithms for analysis can be added. Furthermore, other stand-alone tools written in other languages than Alter, can easily be added and their execution controlled from Aidassign.

29.3 Jitterbug

29.3.1 Tool Overview

Jitterbug [26, 27, 28] is a MATLAB-based *analysis* tool that makes it possible to compute a quadratic performance criterion for a linear control system under various timing conditions. The tool can also compute the spectral density of the signals in the system. Using the toolbox, one can easily and quickly assert how sensitive a control system is to delay, jitter, lost samples, etc., without resorting to simulation. The tool can also be used to investigate jitter-compensating controllers, aperiodic controllers,

and multi-rate controllers. The main contribution of the toolbox, which is built on well-known theory (LQG theory and jump linear systems), is to make it easy to apply this type of stochastic analysis to a wide range of problems.

Jitterbug offers a collection of MATLAB routines that allow the user to build and analyze simple timing models of computer-controlled systems. A control system is built by connecting a number of continuous-time and discrete-time systems. For each subsystem, optional noise and cost specifications may be given. In the simplest case, the discrete-time systems are assumed to be updated in order during the control period. For each discrete system, a random delay (described by a discrete probability density function) can be specified that must elapse before the next system is updated. The total cost of the system (summed over all subsystems) is computed algebraically if the timing model system is periodic or iteratively if the timing model is aperiodic.

Jitterbug is intended mainly as a research tool to evaluate different implementation strategies in terms of control performance. In that scenario a linear controller has been designed for a linear system and the tool will be used to evaluate how sensitive the closed-loop system is to various timing conditions imposed by the implementation.

Tool Architecture

Jitterbug consists of a collection of MATLAB functions that interface to the Matlab Control Systems Toolbox. These functions provide functionality to initialize Jitterbug, set up the timing and signal models that define a Jitterbug system, and to calculate the performance index.

In Jitterbug, a control system is described by two parallel models: a signal model and a timing model. The signal model is given by a number of connected, linear, continuous- and discrete-time systems. The timing model consists of a number of timing nodes and describes when the different discrete-time systems should be updated during the control period. Transitions between states in the timing model are performed depending on a chosen delay distribution.

The same discrete-time system may be updated in several timing nodes. It is possible to specify different update equations in the various cases. This can be used to model a filter where the update equations look different depending on whether or not a measurement value is available. It is also possible to make the update equations depend on the time since the first node became active. This can be used to model jitter-compensating controllers for example.

For some systems, it is desirable to specify alternative execution paths (and thereby multiple next nodes). In Jitterbug, two such cases can be modeled (see Fig. 29.8):

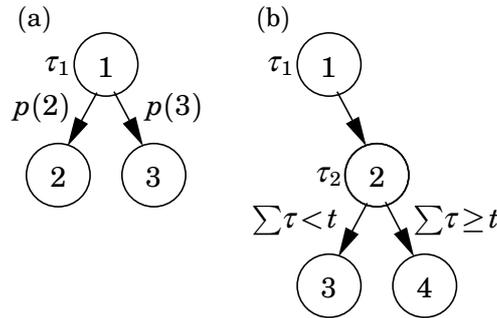


Figure 29.8: Alternative execution paths in a Jitterbug execution model: (a) random choice of path and (b) choice of path depending on the total delay from the first node.

- (a) A vector n of next nodes can be specified with a probability vector p . After the delay, execution node $n(i)$ will be activated with probability $p(i)$. This can be used to model a sample being lost with some probability.
- (b) A vector n of next nodes can be specified with a time vector t . If the total delay in the system since the node exceeds $t(i)$, node $n(i)$ will be activated next. This can be used to model time-outs and various compensation schemes.

29.3.2 Jitterbug Models

As mentioned above, Jitterbug can model most timing-related aspects of real-time control systems, such as constant and random delays, jitter in delays and sampling periods, and network issues such as lost samples.

However, to make the performance analysis feasible, Jitterbug can only handle a certain class of systems. The control system is built from linear systems driven by white noise, and the performance criterion to be evaluated is specified as a quadratic, stationary cost function. The timing delays in one period are assumed to be independent from the delays in the previous period. Also, the delay probability density functions are discretized using a time-grain that is common to the whole model.

Even though a quadratic cost function can hardly capture all aspects of a control loop, it can still be useful when one wants to quickly judge several possible controller implementations against each other. A higher value of the cost function typically indicates that the closed-loop system is less stable (i.e., more oscillatory), and an infinite cost means that the control loop is unstable. The cost function can easily be evaluated for a large set of design parameters and can be used as a basis in the control and real-time design.

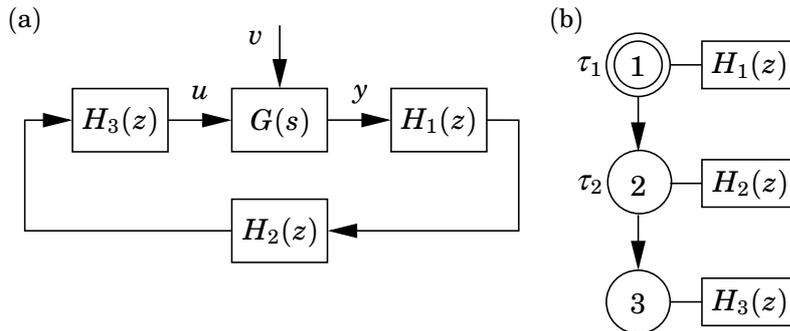


Figure 29.9: A simple Jitterbug model of a computer-controlled system: (a) signal model and (b) timing model. The process is described by the continuous-time system $G(s)$ and the controller is described by the three discrete-time systems $H_1(z)$, $H_2(z)$, and $H_3(z)$, representing the sampler, the control algorithm, and the actuator. The discrete systems are executed according to the periodic timing model.

As an illustration, an example of a Jitterbug model is shown in Figure 29.9, where a computer-controlled system is modeled by four blocks. The plant is described by the continuous-time system G , and the controller is described by the three discrete-time systems H_1 , H_2 , and H_3 . The system H_1 could represent a periodic sampler, H_2 could represent the computation of the control signal, and H_3 could represent the actuator. The associated timing model says that, at the beginning of each period, H_1 should first be executed (updated). Then there is a random delay τ_1 until H_2 is executed, and another random delay τ_2 until H_3 is executed. The delays could model computational delays, scheduling delays, or network transmission delays.

The Jitterbug commands used to define the control system of Figure 29.9 are given in Figure 29.10.

The process is modeled by the continuous-time system

$$G(s) = \frac{1000}{s(s+1)}.$$

and the controller is a discrete-time PD-controller implemented as

$$H_2(z) = -K \left(1 + \frac{T_d}{h} \frac{z-1}{z} \right),$$

The sampler and the actuator are described by the trivial discrete-time systems

$$H_1(z) = H_3(z) = 1,$$

The delays in the computer system are modeled by the two (possibly random) variables τ_1 and τ_2 . The total delay from sampling to actuation is thus given by $\tau_{tot} = \tau_1 + \tau_2$.

<code>G = 1000/(s*(s+1));</code>	Define the process
<code>H1 = 1;</code>	Define the sampler
<code>H2 = -K*(1+Td/h*(z-1)/z);</code>	Define the controller
<code>H3 = 1;</code>	Define the actuator
<code>Ptau1 = [...];</code>	Define delay probability distribution 1
<code>Ptau2 = [...];</code>	Define delay probability distribution 2
<code>N = initjitterbug(delta,h);</code>	Set time-grain and period
<code>N = addtimingnode(N,1,Ptau1,2);</code>	Define timing node 1
<code>N = addtimingnode(N,2,Ptau2,3);</code>	Define timing node 2
<code>N = addtimingnode(N,3);</code>	Define timing node 3
<code>N = addcontsys(N,1,G,4,Q,R1,R2);</code>	Add plant, specify cost and noise
<code>N = adddiscsys(N,2,H1,1,1);</code>	Add sampler to node 1
<code>N = adddiscsys(N,3,H2,2,2);</code>	Add controller to node 2
<code>N = adddiscsys(N,4,H3,3,3);</code>	Add actuator to node 3
<code>N = calcdynamics(N);</code>	Calculate internal dynamics
<code>J = calccost(N);</code>	Calculate the total cost

Figure 29.10: This MATLAB script shows the commands needed to compute the performance index of the control system defined by the timing and signal models in Figure 29.9.

Using the defined Jitterbug model it is straight-forward to investigate, e.g., how sensitive the control loop is to slow sampling and constant delays (by sweeping over suitable ranges for these parameters), and random delays with jitter compensation. For more details and other illustrative examples (including multi-rate control, overrun handling, and notch filter implementations), see [28].

29.3.3 Extensibility

The use of Jitterbug assumes knowledge of sampling period and latency distributions. This information can be difficult to obtain without access to measurements from the true target system under implementation. Also, the analysis cannot capture all the details and nonlinearities (especially in the real-time scheduling) of the computer system. Therefore, the obvious extension of the analysis provided by Jitterbug is to resort to *simulation*.

Jitterbug is available for download at

<http://www.control.lth.se/~lincoln/jitterbug/>

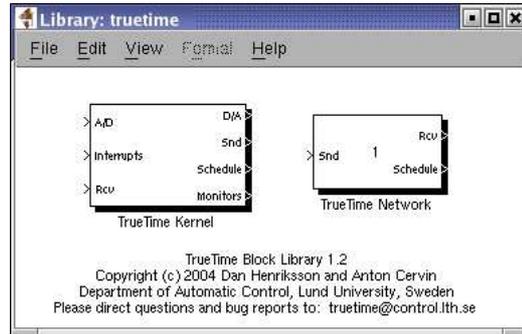


Figure 29.11: The TrueTime block library. The Schedule and Monitor outputs display the allocation of common resources (CPU, monitors, network) during the simulation.

29.4 TrueTime

29.4.1 Tool Overview

TrueTime [26, 29, 30, 31] is a MATLAB/Simulink-based tool that facilitates simulation of the temporal behavior of a multitasking real-time kernel executing controller tasks. The tasks are controlling processes that are modeled as ordinary continuous-time Simulink blocks. TrueTime also makes it possible to simulate models of standard MAC layer network protocols, and their influence on networked control loops.

In TrueTime, kernel and network Simulink blocks are introduced, the interfaces of which are shown in Figure 29.11. The kernel blocks are event-driven and execute code that models, e.g., I/O tasks, control algorithms, and network interfaces. The scheduling policy of the individual kernel blocks is arbitrary and decided by the user. Likewise, in the network, messages are sent and received according to the chosen network model.

The level of simulation detail is also chosen by the user—it is often neither necessary nor desirable to simulate code execution on instruction level or network transmissions on bit level. TrueTime allows the execution time of tasks and the transmission times of messages to be modeled as constant, random, or data-dependent. Furthermore, TrueTime allows simulation of context switching and task synchronization using events or monitors.

In addition to the block library in Figure 29.11, TrueTime provides a collection of C++ functions with corresponding MATLAB MEX-interfaces. Some functions are used to configure the simulation by creating tasks, interrupt handlers, monitors, timers, etc. The remaining functions are real-time primitives that are called from the task code during execution. These include functions for A/D-D/A conversion, changing

task attributes, entering and leaving monitors, sending and receiving network messages, and more.

TrueTime is configured in a C++ or MATLAB m-file, called an initialization script. Likewise, task and interrupt handler code is defined by C++ functions or MATLAB m-files according to a pre-specified format. The possibility for graphical modeling has been avoided to make the tool more general and more connected to the real implementation code.

The Kernel Block

The kernel block is a MATLAB S-function that simulates a computer with a simple but flexible real-time kernel, A/D and D/A converters, a network interface, and external interrupt channels. The kernel executes user-defined tasks and interrupt handlers. Internally, the kernel maintains several data structures that are commonly found in a real-time kernel: a ready queue, a time queue, and records for tasks, interrupt handlers, monitors and timers that have been created for the simulation.

An arbitrary number of tasks can be created to run in the TrueTime kernel. Tasks may also be created dynamically as the simulation progresses. Tasks are used to simulate both periodic activities, such as controller and I/O tasks, and aperiodic activities, such as communication tasks and event-driven controllers. Aperiodic tasks are executed by the creation of task instances (jobs).

Each task is characterized by a number of static (e.g., relative deadline, period, and priority) and dynamic (e.g., absolute deadline and release time) attributes. In accordance with the Real-Time Specification for Java (RTSJ) [32], it is furthermore possible to attach two overrun handlers to each task: a deadline overrun handler (triggered if the task misses its deadline) and an execution time overrun handler (triggered if the task executes longer than its worst-case execution time).

Interrupts may be generated in two ways: externally (associated with the external interrupt channel of the kernel block) or internally (triggered by user-defined timers). When an external or internal interrupt occurs, a user-defined interrupt handler is scheduled to serve the interrupt.

The execution of tasks and interrupt handlers is defined by user-written code functions. These functions can be written either in C++ (for speed) or as MATLAB m-files (for ease of use). Control algorithms may also be defined graphically using ordinary discrete Simulink block diagrams.

Simulated execution occurs at three distinct priority levels: the interrupt level (highest priority), the kernel level, and the task level (lowest

priority). The execution may be preemptive or non-preemptive; this can be specified individually for each task and interrupt handler.

At the interrupt level, interrupt handlers are scheduled according to fixed priorities. At the task level, dynamic-priority scheduling may be used. At each scheduling point, the priority of a task is given by a user-defined priority function, which is a function of the task attributes. This makes it easy to simulate different scheduling policies. For instance, a priority function that returns a priority number implies fixed-priority scheduling, whereas a priority function that returns the absolute deadline implies earliest-deadline-first scheduling. Predefined priority functions exist for rate-monotonic, deadline-monotonic, fixed-priority, and earliest-deadline-first scheduling.

The Network Block

The network block is event-driven and executes when messages enter or leave the network. When a node tries to transmit a message, a triggering signal is sent to the network block on the corresponding input channel. When the simulated transmission of the message is finished, the network block sends a new triggering signal on the output channel corresponding to the receiving node. The transmitted message is put in a buffer at the receiving computer node.

A message contains information about the sending and the receiving computer node, arbitrary user data (typically measurement signals or control signals), the length of the message, and optional real-time attributes such as a priority or a deadline.

The network block simulates medium access and packet transmission in a local area network. Six simple models of networks are currently supported: CSMA/CD (e.g. Ethernet), CSMA/AMP (e.g. CAN), Round Robin (e.g. Token Bus), FDMA, TDMA (e.g. TTP), and Switched Ethernet. The propagation delay is ignored, since it is typically very small in a local area network. Only packet-level simulation is supported, i.e., it is assumed that higher protocol levels in the kernel nodes have divided long messages into packets.

Configuring the network block involves specifying a number of general parameters, such as transmission rate, network model, and probability for packet loss. Protocol-specific parameters that need to be supplied include, e.g., the time slot and cyclic schedule in the case of TDMA.

Currently support for wireless networks is being added to TrueTime. So far, the IEEE 802.11 b/g and the IEEE 805.15.4 protocols have been implemented.

Development Scenarios Supported

The main use of TrueTime is for simultaneous simulation of all aspects of distributed real-time control applications. By co-simulation of continuous process dynamics, task execution in real-time kernels, and network communication, it is possible to evaluate the performance of control loops subject to the constraints of the target system.

In a typical scenario, a controller design has been performed (without considering implementation constraints) and is about to be implemented on the target system. In this scenario, TrueTime can be used to evaluate different real-time implementations, and the effects of CPU and network scheduling, task attributes, etc, on the control performance.

For a given implementation architecture, TrueTime may also be used to obtain temporal statistics that can be used as constraints in the design of the controller.

In the optimal scenario, however, the controller and architectural designs are performed at the same time. Here, TrueTime provides a convenient framework for integrated control and real-time design.

TrueTime is also used as an experimental platform for research on flexible approaches to real-time implementation and scheduling of controller tasks. One example is feedback scheduling [33, 34] where feedback is used in the real-time system to dynamically distribute resources according to the current situation in the system.

TrueTime may be used in all stages of the development process, from the early stages and system specifications, during the actual system construction, and finally for testing and validation.

Activities Supported

TrueTime makes it possible to simulate the temporal behavior of the computer architecture (e.g., scheduling policies and network protocols) and its effect on the control performance. Standard scheduling policies may be used, e.g., priority-based preemptive scheduling and earliest-deadline-first scheduling, but it is also straight-forward to define arbitrary user-defined policies. Task overrun strategies may be evaluated and easily implemented using the TrueTime overrun handlers.

TrueTime can also be used as an experimental platform for research on co-design of control algorithms and computer resource scheduling mechanism. It is possible to study dynamic compensation schemes that adjust the controller on-line based on measurements of actual timing variations, i.e., treat the temporal uncertainty as a disturbance and manage it with feed-forward or gain scheduling. It is also easy to implement new more flexible approaches to dynamic scheduling, e.g., feedback scheduling [33] of CPU time and communication bandwidth and quality-of-service (QoS) based scheduling, in the TrueTime CPU kernel.

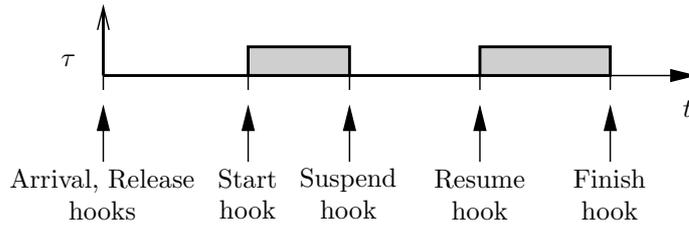


Figure 29.12: TrueTime scheduling hooks.

TrueTime may also be used only as a scheduling simulator, without being connected to any continuous-time processes. This can be used to get information of the timing of the real-time system, and various scheduling policies can be evaluated in terms of deadline misses and response times.

Being developed in Simulink, TrueTime allows for traditional control system assessment in terms of performance, stability and robustness. Compared to normal control system development in Simulink, TrueTime also considers the constraints imposed by the implementation platform.

29.4.2 Tool Architecture

TrueTime is primarily intended to be used together with MATLAB/Simulink. However, the TrueTime kernel actually implements a complete event-based kernel and Simulink is only used to interface the kernel and the tasks with the continuous-time processes.

TrueTime is written in C++ and consists of two Simulink S-functions for the kernel and network block, and a collection of C++ functions for the initialization commands and real-time primitives. All TrueTime objects, such as tasks, interrupt handlers, monitors, timers, and events, are defined by C++ classes. These classes as well as the real-time primitives may easily be extended by the user to add more functionality.

The Simulink engine is used only for timing and interfacing with the rest of the model (the continuous dynamics). Since it is written in C++, it should thus be easy to port the block code to other simulation environments, provided these environments support event detection (zero-crossing detection).

29.4.3 Tool Inputs

TrueTime is initialized in a script for each kernel block (node). In this script, the user specifies the scheduling policy of the kernel, creates tasks and assigns task attributes (period, priority, deadlines, etc), and creates any other objects for the simulation (interrupt handlers, timers, monitors, mailboxes, etc). The execution of each task and handler is defined

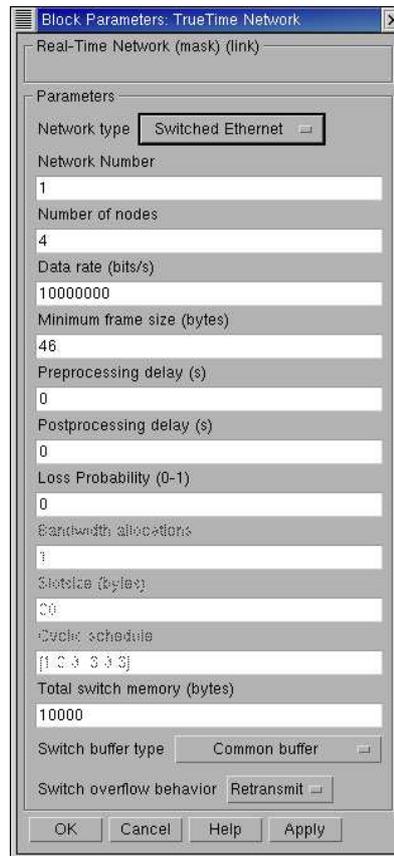


Figure 29.13: The dialog of the TrueTime Network block.

by a code function (see Modeling Content below) with constant or random execution time. It is also possible to specify a simulated time associated with context switches.

Furthermore, to facilitate arbitrary dynamic scheduling mechanisms, it is possible to attach small pieces of code (*hooks*) to each task. These hooks are executed at different stages during the simulation, as shown in Figure 29.12.

The network block is configured through the block mask dialog, see Figure 29.13. The following network parameters are common to all models; number of nodes in the network, data rate (bits/s), minimum frame size (bytes), pre- and post-processing delay, and loss probability. Protocol-specific attributes include slot sizes for TDMA, and buffer size and buffer type for switched Ethernet.

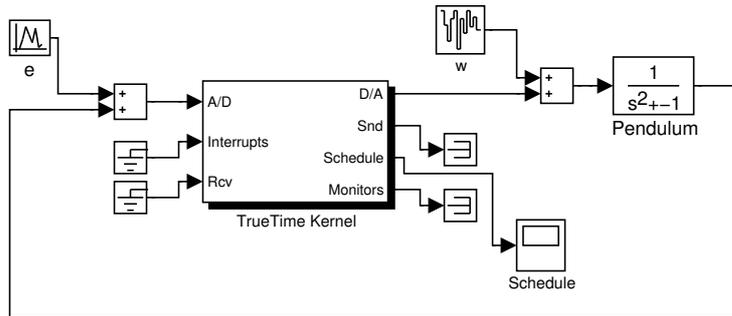


Figure 29.14: A TrueTime computer block connected to a continuous pendulum process.

29.4.4 Tool Outputs

Depending on the simulation a number of different output graphs are generated by the TrueTime blocks. Each kernel block will produce two graphs; a computer schedule and a monitor graph, and the network block will produce a network schedule. The computer schedule will display the execution trace of each task and interrupt handler during the course of the simulation. If context switching is simulated, the graph will also display the execution of the kernel.

There will be one execution trace for each task and handler. If the signal is high this means that the task is running. A medium signal indicates that the task is ready but not running (preempted), whereas a low signal means that the task is idle. In an analogous way the network schedule shows the transmission of messages over the network, with the states representing sending (high), waiting (medium), and idle (low). The monitor graph shows which tasks that have been holding the different monitors during the simulation.

It is also possible to create logs for each tasks. These will log arbitrary task attributes, such as response times and latencies, during the simulation and write them to the MATLAB workspace after the simulation.

Plant and controller outputs are conveniently displayed and evaluated using the Simulink built-in outputs. It is also possible to dynamically evaluate for example quadratic performance functions, within Simulink.

29.4.5 Modeling Content

The TrueTime blocks are connected with ordinary Simulink blocks to form a real-time control system, see Figure 29.14.

Figure 29.15: Example of a simple TrueTime initialization function.

```

function example_init

ttInitKernel(2, 1, 'prioFP');

name   = 'ctrl';
offset = 0;
period = 0.005;
prio   = 2;
data.u = 0;
data.K = 2;

ttCreatePeriodicTask (name, offset,
                      period, prio, 'Pcontroller', data);

```

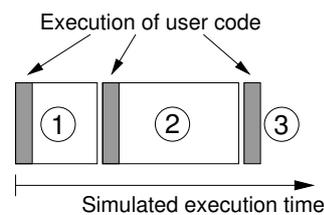


Figure 29.16: The execution of the code associated with tasks and interrupt handlers is modeled by a number of code segments with different execution times. Execution of user code occurs at the beginning of each code segment.

Before a simulation can be run it is necessary to initialize the individual kernel blocks. Initialization of a TrueTime kernel block involves specifying the number of inputs and outputs of the block, defining the scheduling policy, and creating tasks, interrupt handlers, events, monitors, etc for the simulation. This is done in an initialization script for each kernel block.

The initialization code in Figure 29.15 shows the minimum of initialization needed for a TrueTime simulation (e.g., corresponding to the simple simulation model in Figure 29.14). The kernel is initialized by providing the number of inputs and outputs and the scheduling policy using the function `ttInitKernel`. A periodic task is then created by the function `ttCreatePeriodicTask`. The execution of the task is given by the code function `Pcontroller`, described below.

The execution of tasks and interrupt handlers is defined by code functions. A code function is further divided into code segments according to the execution model in Figure 29.16. The code can interact with other tasks and with the environment at the beginning of each code segment. This execution model makes it possible to model input-output latencies, blocking when accessing shared resources, etc. The number of segments can be chosen to simulate an arbitrary time granularity of the code execution. Technically it would, e.g., be possible to simulate very fine-grained details occurring at the machine instruction level, such as race conditions. However, that would require a large number of code segments.

The simulated execution time of each segment is returned by the code function, and can be modeled as constant, random, or even data-dependent. The kernel keeps track of the current segment and calls the code functions with the proper argument during the simulation. Execution resumes in the next segment when the task has been running for the time associated with the previous segment. This means that preemption by higher-priority activities and interrupts may cause the actual delay between execution of segments to be longer than the execution time.

Figure 29.17 shows an example of a code function corresponding to the time line in Figure 29.16. The function implements a standard P-controller. In the first segment, the plant is sampled and the control signal is computed. In the second segment, the control signal is actuated and the controller states are updated. The third segment indicates the end of execution by returning a negative execution time.

The data structure `data` represents the local memory of the task and is used to store the control signal and measured variable between calls to the different segments. A/D and D/A conversion is performed using the kernel primitives `ttAnalogIn` and `ttAnalogOut`.

Note that the input-output latency of this controller will be *at least* 2 ms (i.e., the execution time of the first segment). However, if there is preemption from other high-priority tasks, the actual input-output latency will be longer.

TrueTime interrupt handlers is used to model code that is executed in response to interrupts. Interrupt handlers are scheduled with fixed priorities on a higher priority level than tasks. Interrupt handlers may be associated with timers, the network receive channel, external interrupt channels, or attached to tasks as overrun handlers. Timers can be one-shot or periodic.

TrueTime monitors are used to provide mutual exclusion and synchronization between tasks. Tasks waiting for monitor access are sorted according to their priority under the given scheduling policy. Standard priority inheritance is implemented as resource access policy. TrueTime

Figure 29.17: Example of a standard code function written in MATLAB code. The local memory of the controller task is represented by the data structure `data`. This stores the controller gain and the control signal between invocations of different code segments.

```
function [exectime, data] = Pcontroller(segment, data)
switch segment,
case 1,
    r = ttAnalogIn(1);
    y = ttAnalogIn(2);
    data.u = data.K*(r-y);
    exectime = 0.001;
case 2,
    ttAnalogOut(1, data.u);
    exectime = 0.001;
case 3,
    exectime = -1; % finished
end
```

events may be free or associated with monitors as condition variables. The event waiting queues are also priority-sorted.

29.4.6 Availability

TrueTime is available for download at

<http://www.control.lth.se/~dan/truetime/>

29.5 XILO

29.5.1 Tool Overview

XILO - standing for X-in-the loop simulation - is a prototypical toolset, built upon Simulink, developed to support detailed architectural design of distributed real-time control systems, [35]. The approach enables the co-simulation of functionality, from discrete-time control to logic, together with the controlled continuous-time processes and the behaviour of the computer system. In particular, modelling and simulation of distributed computer control systems is supported allowing analysis of timing and control system robustness. An emphasised feature of the tool is its multidisciplinary and integrated approach that combines the views of control and computer engineering into one view at an appropriate level of abstraction. The XILO toolset is composed of a number of libraries that lets a designer configure a distributed computer control system and

to allocate and partition the functionality as desired. Along with the basic toolset, an additional library that supports fault-injection in terms of bit-flips in all types of blocks, signals and constants has been developed, [36]. Some of the basic mechanisms of XILO have been reused in the Aida toolset [24].

29.5.2 Development Scenarios Supported

The usage of this toolset is similar in nature to the AIDA toolset [24] and TrueTime [26, 29]. The main emphasis is on architectural design on a rather detailed level, although usage in earlier stages is also possible. Given a control design, the resulting control performance can be evaluated for different computer system architectures and for different mappings to the architecture. Alternatively, it is possible to just use the simulation to get an idea about the timing of the computer system.

In the typical usage scenario, the control system has been designed, and it is of interest to study how it can be implemented on a single processor or in a distributed computer system. The hardware and software architecture could be fixed beforehand, or its design could also be guided by the results of XILO simulations. To use XILO, a new Simulink model is created, and by using the XILO block libraries the hardware and software architectures are defined. The functionality of the original control system model is then partitioned to the tasks (in general time or event-driven activities) of the nodes, inter-thread communication is added and the whole system is configured (e.g., by adding execution time information and priorities). Based on such a developed model in Simulink, alternative designs can be simulated and their behaviour compared. Hence, the toolset could be used to for example:

- compare and evaluate hardware architectures and function allocation
- compare and evaluate software architectures and task partitioning
- compare and evaluate different execution strategies, e.g. different triggering and scheduling approaches
- compare and evaluate control system designs

29.5.3 Tool Architecture

XILO is completely implemented within MATLAB/Simulink [25]. MATLAB/Simulink was chosen because of the relative ease with which it could be extended to model real-time implementations, because of its

support for the modelling of event-triggered and time-triggered systems, and because of its wide-spread use as a control design environment. Simulink allows for the integration of custom code into its models. Hence, it is possible to model the control application together with any encapsulating software.

On the other hand, there is a gap between the Simulink modelling level and the real-time system implementation. For example, in Simulink, the execution of a block takes zero time according to the simulation time, thus there is a need to provide a mechanism whereby durations due to execution or communication can be modelled. Moreover, the peculiarities of embedded system platforms (including, among other things, hardware, interrupts and real-time operating systems) need to be appropriately modelled and "instrumented" into Simulink to ensure that the simulation behaves as a real-time implementation, in particular considering scheduling and preemptions. That is, the original execution ordering undertaken by Simulink during simulation is not sufficient [37].

The handling of hybrid systems requires the simulator to have an event-triggered architecture, where all the activities in the system are event-triggered, and the Simulink triggering capabilities are used extensively. Note that this still allows for the modelling of purely time-triggered architectures, where time is viewed as an event. A combination of C-coded S-functions and Simulink blocks were used in the implementation. Each of the model components in the XILO libraries (further discussed below) is represented in the simulator as a Simulink block.

From the definition of the modelling content below it will be seen that there is extensive data exchange between the XILO library components. Taking the traditional Simulink approach of connecting blocks to exchange data is not favourable since this will certainly complicate the model for the simplest cases. Even worse, confusion will occur between data exchange of the application itself and that needed for the implementation of the underlying components. The approach taken in the simulator is to hide all data exchanges that do not form part of the representation model of the system. For example, although data exchange is necessary between a scheduler and each of the tasks on its processor, these links are not explicit in the model presentation since they do not contribute to the understanding of the model. The user need not be concerned with these hidden links since each component automatically reconfigures itself based on the presence of other blocks in the system. Also, the interface between these types of components within a node is well defined and fixed, allowing for the independent development of subtypes and variations in the internals of each of the components.

Table 29.1: Explicit XILO component attributes.

Component	Attributes
Communication link	Link Speed Communication Protocol
Scheduler	Scheduling Algorithm
Service Provider/ Hardware Unit	Service Response Policies Internal Buffer Sizes
Elementary Function	Execution Time

29.5.4 Tool Inputs

Each of the model components in the XILO libraries is represented in the simulator as a Simulink block. The drag-and-drop approach of blocks from libraries is used to build the models. Blocks are then customised through a graphical user interface. There are no restrictions on the types of standard Simulink blocks that can be used with the simulator models, except for those imposed by Simulink itself. This ensures a well-integrated environment with Simulink and the user does not need to learn a new tool. In order to use the tool according to the intended scenario, a control system model made in Simulink is needed.

Table 29.1 lists the explicit attributes that the user needs to specify for each of the components in the model. In addition, components contain implicit properties that can be automatically derived from their context. For example, a task implicitly identifies the list of elementary functions that are contained within it. Also, each component contains a unique identifier that distinguishes it from other components.

29.5.5 Tool Outputs

The simulator permits the user to monitor any variable in the system. Parameters that may be of interest for timing analysis include: task status, the times when particular events or activities within a task occur, or the time when a service request is serviced. Apart from these outputs, ordinary Simulink outputs can be used to for example store or visualize the behaviour of the controller and the controlled system.

29.5.6 Modeling Content

At the top level, the hardware topology of the whole system is modelled. This hardware structure consists of three types of components: the surrounding environment, communication links, and the computer nodes.

The environment is described using standard Simulink modelling techniques and blocks, typically to capture a model of the dynamics

of a mechanical system including sensors and actuators. It is necessary that the chosen environment model is integrable with the rest of the hardware model, meaning that it should be possible to actuate and sense appropriate signals of the mechanics.

A computer node connects to the system environment at various points. This connection is performed via Hardware Units such as pulse width modulators (PWM), analog to digital converters (ADC), and digital to analog converters (DAC) that reside in the node. A communication link provides data exchange facilities between computer nodes. It defines the protocols that handle the messages being sent between connected nodes. A communication link indirectly interacts with each connected node through communication controllers that reside in the node. The communication link performs the scheduling of messages requested from connected controllers, while the controller internally schedules its own messages. Such a setup allows for a representation of a multitude of node and link models to be connected, as well as allowing for a node to connect to one or more links, and vice versa. As an example, consider the implementation of the CAN bus. Requests to send messages on the bus are received by the bus from the controllers. These requests are only serviced if the bus is idle, and ongoing transmissions are not disturbed. Once the bus is idle, controllers arbitrate between each other to gain access to the bus according to the CAN protocol. The message transmission time depends on the bus bit rate and the message size including any protocol overheads. Note that the arbitration within a CAN controller is performed independently at the node level, and that this local scheduling is not part of the CAN protocol itself.

A node consists of the following types of components:

- one or more tasks from which the system software is built (the following section describes the task model)
- a task scheduler
- zero or more operating system Service Providers (SP) such as inter-task communication, task synchronisation and semaphores
- zero or more hardware units such as communication controllers, timers, ADCs and DACs
- a processor

The application functionality to be developed by the user is composed of application tasks, with services provided by the other components comprising the operating system. Software layers, which interface the application software to the system hardware and operating system, can be easily modelled and designed with this approach. For example,

system tasks, belonging to the operating system, can be developed to implement software drivers or high level network protocols.

Scheduler. A single task scheduler exists for each node in the system. The role of the scheduler role is to, when triggered, simply choose and activate a single task that is to run on the node processor at that time. The scheduler may be triggered by any of the service providers installed on that node, or by a timer reaching certain pre-defined points in time. A task list component also exists which holds certain information on each task such as its ID, current status, priority and any user-specific parameters. The scheduler only needs to interface to the task list in its decision making, and different scheduler models require different information about the tasks and hence, the task list model should be consistent with that of the scheduler.

This model allows for the modelling of a wide range of schedulers such as event/time triggered, static/dynamic, and off-line/on-line schedulers. Developing a new scheduler requires the implementation of the function that decides on the next running task, as well as the design of the data structures needed for each task in the task list. Using a particular scheduler simply requires the inclusion of the scheduler and its accompanying task list into the node, and any off-line customising is done by the user through the task list. As an illustrating example, consider the design of a fixed-priority preemptive scheduler. The task list stores, for each task, the user-specified static priority as well as its current status. A task status can be one of ready, running or blocked. The scheduler in this case is triggered every time a task changes its status in order to evaluate if a more legitimate ready task needs to run on the processor. A scheduler triggering may be caused by, for example, an interrupt or a service provider.

Service Provider. Examples of service providers are inter-task communication and task synchronisation. Although they vary in their functionality, these components have very similar features and interactions to the rest of the system. Essentially, a service provider (SP) responds to a service request from a task to perform certain activities. This activity may cause the calling task (or any other task, in general) to change status due to the internal state of the SP. The mechanism of making requests by a task and the response to these requests is fixed across all services. What varies is the interpretation of the requests and the way they are handled. Hence, developing new services simply requires the definition of the internal states, and the functionality to handle the various types of possible requests. All SPs have access to the task list and are able to trigger the processor scheduler.

As an example, consider an inter-task communication service implemented as a first-in/first-out, block-on-full service. When a task requests to send a message, it simply sends the data to the specific SP. Normally,

the SP places the data in the FIFO buffer. However, if the buffer is full, the SP changes the task status to blocked, and triggers the scheduler. When the buffer is available again, the SP changes the task status to ready and triggers the scheduler.

Hardware Unit. Hardware units may be fully embedded in the computer node (such as a floating point processor) and hence only interfacing with the processor, or they could lie on the border (such as an ADC) and provide an interface between the processor and the surrounding environment. From the task perspective, the interface to a hardware unit is similar to that of a service provider in that a request is made for a service which the unit provides. Hence, it is important to match the requests to the correct service. However, a hardware unit differs from a service provider in that it has no direct access to the internal data structures of the OS such as the scheduler or task list. Instead, the unit may cause processor interrupts that tasks in the system need to handle appropriately. This model naturally facilitates the masking of these units by developing unit drivers (consisting of system tasks) that encapsulate the hardware units and handle the generated interrupts. As a simple example, consider a hardware unit implementing a CAN communication controller. A task requesting to send a message over CAN makes a request for service by directly accessing the hardware registers. The unit in turn communicates with the associated CAN communication link, and upon receiving a message from the link, produces a hardware interrupt (if so configured). The CAN controller performs the local scheduling of simultaneous transmission requests, e.g. FIFO or priority based.

A task is modelled as a single sequence of *elementary functions* (EF). Each EF is assumed to take a specified non-zero amount of time to execute. We can draw a simple task as shown in Figure 29.18, illustrating the precedence relationship between the elementary functions. The EF can be either user-specific (octagonal block representation) or an operating system service request (rectangular block). When first triggered, the task is made ready to run on the processor. During its lifetime, and depending on the system activities and the scheduler being used, a task runs on the processor at different time slots. The directed link between two EFs within a task indicates passing control from the source to the destination EF, triggering the destination EF to begin its execution. The currently activated EF terminates once the task executes for a time period that is equivalent to the EF execution time, since the EF was first activated. When the control is passed to the last block, the task is terminated.

This simple model can be extended in order to provide looping and branching of the elementary functions, as shown in Figure 29.19. Once triggered, the Branch block (B) produces an output trigger in one, and only one of its outputs, based on internal logic. The Merge block (M)

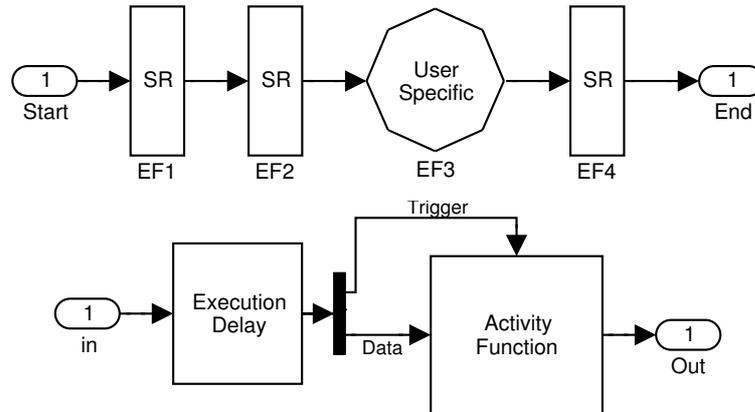


Figure 29.18: (top) The task model consisting of a sequence of elementary functions (EF). (bottom) The internal model of an EF.

produces a trigger on the output as soon as any of its inputs is triggered. These mechanisms do not consume any processor time, and are only intended to be used for high level modes of operations of the application.

Tasks may be initially triggered as soon as the node is booted, or they may be configured to be triggered by an interrupt. The first is typical for many application tasks, while the latter can be used to model system interrupt handlers. It is also necessary to have at least a single task (the system idle task) in the system that is initiated during boot time, and that may never terminate.

Elementary Function (EF). The internal model of an EF is shown in the bottom part of Figure 29.18. The input to an EF (either user-specific or an operating system service request) consists of the elementary function trigger and its data. When an EF is first triggered, the input data is captured and when its specified execution time elapses, an output trigger is produced together with output data. By definition, an elementary function may be preempted at any instance in its execution. Non-preemptive EFs can be implemented as a subset of normal EFs, by implicitly surrounding each elementary function with service requests that disable and enable preemption.

29.5.7 Extensibility

XILO is completely developed in Simulink. Given the rich API of Simulink and possibilities to integrate custom-code there are many possibilities open for extensions. For example, it would be possible to develop platform models at different levels of abstraction, and to support different kinds of platforms (e.g. different networks and unsynchronized nodes). In addition, the idea with the toolset was to also support the

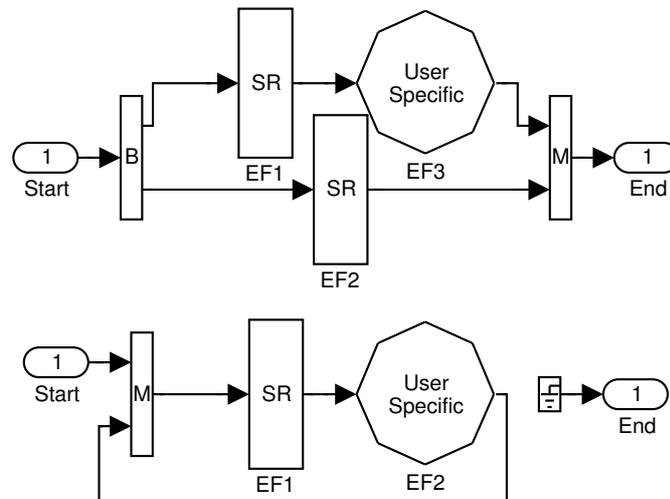


Figure 29.19: Examples of more complex task structures such as branching (top) or looping (bottom).

definition and analysis of different approaches for error detection and handling. Such facilities have been studied but are currently only partially implemented in the toolset. It is straightforward to implement fault-injection in Simulink related to different fault-models, including permanent and transient faults, and component failure modes of different kinds, from crash to asymmetric failures. For experiences with such models in Simulink see [37, 36]. In [36] we describe the development of a Simulink library that models transient hardware faults, in particular single bit-flips, and its use in control system robustness evaluation. As well as being executable, the models described above are representative enough that the collection of task models from each of the nodes in the system can serve as a basis for a detailed software model, which could be translated to a more traditional and familiar form, if desired. Sufficient level of detail is available to generate pseudo-code for each task in the system, and configuration information indicating the services needed for each node. Integration with commercial tools such as code generators is also possible.

Availability

XILO has been developed in-house at the Division of Mechatronics at KTH; available upon request.

Bibliography

- [1] Control Systems Society, “CACSD history,” Home page, <http://www.robotic.dlr.de/control/cacsd/cacsd/history.shtml>, 2004.
- [2] J. M. Bass, A. R. Browne, M. S. Hajji, D. G. Marriott, P. R. Croll, and P. J. Fleming, “Automating the development of distributed control software,” *IEEE Parallel and Distributed Technology: Systems and Technology*, vol. 2, pp. 9–19, 1994.
- [3] R. Lauwereins, M. Engels, M. Ad, and J. a. Peperstraete, “Grape-II: A system-level prototyping environment for dsp applications,” *IEEE Computer*, vol. 28, pp. 35–43, 1995.
- [4] S. Vestal, “Integrating control and software views in a cace/case toolset,” in *Proceedings of the IEEE/IFAC Joint Symposium on Computer-Aided Control System Design*, Tucson, Arizona, 1994, pp. 353–358.
- [5] D. Bhatt, V. Thomas, and J. Shackleton, “A methodology and toolset for the design of parallel embedded systems,” *ACM SIG-PLAN OOPS Messenger*, vol. 7, pp. 5–12, 1996.
- [6] N. Audsley, A. Burns, M. Richardson, and A. Wellings, “STRESS—A simulator for hard real-time systems,” *Software—Practice and Experience*, vol. 24, no. 6, pp. 543–564, June 1994.
- [7] M. F. Storch and J. W.-S. Liu, “DRTSS: A simulation framework for complex real-time systems,” in *Proc. 2nd IEEE Real-Time Technology and Applications Symposium*, 1996.
- [8] MathWorks, *Simulink: A Program for Simulating Dynamic Systems—User’s Guide*. The MathWorks Inc, 2001.
- [9] D. Brück, H. Elmqvist, S. E. Mattsson, and H. Olsson, “Dymola for multi-engineering modeling and simulation,” in *Proc. 2nd International Modelica Conference*, 2002.
- [10] A. Casile, G. Buttazzo, G. Lamastra, and G. Lipari, “Simulation and tracing of hybrid task sets on distributed systems,” in *Proc. 5th International Conference on Real-Time Computing Systems and Applications*, 1998.
- [11] L. Palopoli, L. Abeni, and G. Buttazzo, “Real-time control system analysis: An integrated approach,” in *Proc. 21st IEEE Real-Time Systems Symposium*, 2000.

- [12] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, and H. Zheng, "Overview of the Ptolemy project," Department of Electrical Engineering and Computer Science, University of California Berkeley, CA, Tech. Rep. UCB/ERL M03/25, July 2003.
- [13] Ptolemy Project, "Ptolemy II," Home page, <http://ptolemy.eecs.berkeley.edu/>, 2004.
- [14] J. Liu and E. Lee, "Timed multitasking for real-time embedded software," *IEEE Control Systems Magazine*, vol. 23, no. 1, Feb. 2003.
- [15] D. Simon, B. Espiau, E. Castillo, and K. Kapellos, "Computer-aided design of a generic robot controller handling reactivity and real-time control issues," *IEEE Transactions on Control Systems Technology*, vol. 1, no. 4, December 1993.
- [16] D. Simon, R. Pissard-Gibollet, K. Kapellos, and B. Espiau, "Synchronous composition of discretized control actions: Design, verification, and implementation with Orcad," in *Proceedings of the 6th International Conference on Real-Time Control Systems and Applications*, Hong Kong, December 1999.
- [17] D. Simon and A. Girault, "Synchronous programming of automatic control applications using Orcad and Esterel," in *Proceedings of the 40th IEEE Conference on Decision and Control, CDC'01*, Orlando, USA, December 2001.
- [18] D. Simon, B. Espiau, K. Kapellos, and R. Pissard-Gibollet, "Orcad: Software engineering for real-time robotics," *A Technical Insight, Robotica, Special Issues on Languages and Software in Robotics*, vol. 15, no. 1, pp. 111–116, March 1997.
- [19] N. Pernet and Y. Sorel, "Optimized implementation of distributed real-time embedded systems mixing control and data processing," in *Proceedings of the ISCA 16th International Conference: Computer Applications in Industry and Engineering (CAINE-2003)*, Las Vegas, USA, November 2003.
- [20] T. Grandpierre, C. Lavarenne, and Y. Sorel, "Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors," in *Proceedings of the 7th International Workshop on Hardware/Software Co-design*, Rome, Italy, March 1999.
- [21] C. Lavarenne, O. Seghrouchni, Y. Sorel, and M. Sorine, "The Syndex software environment for real-time distributed systems design and implementation," in *Proceedings of the European Control Conference*, Grenoble, France, July 1991.

- [22] J. Forget, C. Lavarenne, and Y. Sorel, “Syndex v6 – user manual,” Tech. Rep., April 2004.
- [23] D. Henriksson, O. Redell, J. El-Khoury, M. Törngren, and K.-E. Årzén, “Tools for real-time control systems co-design — a survey,” Department of Automatic Control, Lund Institute of Technology, Sweden, Tech. Rep. ISRN LUTFD2/TFRT--7612--SE, Apr. 2005.
- [24] O. Redell, J. El-Khoury, and M. Törngren, “The AIDA tool-set for design and implementation analysis of distributed real-time control systems,” *Journal of Microprocessors and Microsystems*, vol. 28, no. 4, pp. 163–182, May 2004.
- [25] The Mathworks, “MATLAB and Simulink for technical computing,” Home page, <http://www.mathworks.com>, 2005.
- [26] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén, “How does control timing affect performance?” *IEEE Control Systems Magazine*, vol. 23, no. 3, pp. 16–30, June 2003.
- [27] B. Lincoln and A. Cervin, “Jitterbug: A tool for analysis of real-time control performance,” in *Proceedings of the 41st IEEE Conference on Decision and Control*, Las Vegas, NV, Dec. 2002.
- [28] A. Cervin and B. Lincoln, “Jitterbug 1.1—Reference manual,” Department of Automatic Control, Lund Institute of Technology, Sweden, Tech. Rep. ISRN LUTFD2/TFRT--7604--SE, Jan. 2003.
- [29] D. Henriksson, A. Cervin, and K.-E. Årzén, “TrueTime: Real-time control system simulation with MATLAB/Simulink,” in *Proceedings of the Nordic MATLAB Conference*, Copenhagen, Denmark, Oct. 2003.
- [30] D. Henriksson and A. Cervin, “TrueTime 1.1—Reference manual,” Department of Automatic Control, Lund Institute of Technology, Tech. Rep. ISRN LUTFD2/TFRT--7605--SE, Mar. 2003.
- [31] D. Henriksson, A. Cervin, and K.-E. Årzén, “TrueTime: Simulation of control loops under shared computer resources,” in *Proceedings of the 15th IFAC World Congress on Automatic Control*, Barcelona, Spain, July 2002.
- [32] G. Bollella, B. Brosgol, P. Dibble, S. Furr, J. Gosling, D. Hardin, and M. Turnbull, *The Real-Time Specification for Java*. Addison-Wesley, 2000.
- [33] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, “Feedback-feedforward scheduling of control tasks,” *Real-Time Systems*, vol. 23, no. 1–2, pp. 25–53, July 2002.

- [34] D. Henriksson, A. Cervin, J. Åkesson, and K.-E. Årzén, “Feedback scheduling of model predictive controllers,” in *Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium*, San Jose, CA, Sept. 2002.
- [35] J. El-Khoury and M. Törngren, “Towards a toolset for architectural design of distributed real-time control systems,” in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, London, England, December 2001.
- [36] J. Norberg and M. Törngren, “Fault injection into control algorithms,” Department of Machine Design, KTH, Sweden, Tech. Rep. TRITA-MMK 2003:37, ISSN 1400-1179, ISRN KTH/MMK/R-03/11-SE, 2003.
- [37] M. Törngren, J. El-Khoury, M. Sanfridsson, and O. Redell, “Modelling and simulation of embedded computer control systems: Problem formulation,” Department of Machine Design, KTH, Stockholm, Sweden, Tech. Rep. TRITA-MMK 2001:3, ISSN 1400-1179, ISRN KTH/MMK/R-01/3-SE, 2001.

Chapter 30

The Control Server Model for Codesign of Real-Time Control Systems

By **Anton Cervin**[†] and **Johan Eker**[‡]

[†]Department of Automatic Control
Lund University
Email: anton@control.lth.se

[‡]Ericsson Mobile Platforms
Email: johan.eker@emp.ericsson.se

The paper presents the control server, a real-time scheduling mechanism tailored to control and signal processing applications. A control server creates the abstraction of a control task with a specified period and a fixed input-output latency shorter than the period. Individual tasks can be combined into more complex components without loss of their individual guaranteed fixed-latency properties. I/O occurs at fixed predefined points in time, at which inputs are read or controller outputs become visible. The control server model is especially suited for codesign of real-time control systems. The single parameter linking the scheduling design and the controller design is the task utilization factor. The proposed server is an extension of the constant bandwidth server, which is based on the earliest-deadline-first scheduling algorithm. The server has been implemented in a real-time kernel and has also been validated in control experiments on a ball and beam process.

30.1 Introduction

The design of a real-time control system is essentially a codesign problem. Decisions made in the real-time design affect the control design, and vice versa. For instance, the choice of scheduling policy influences the latency distributions in the control loops, and, ideally, this should be taken into account in the control design. At the same time, the performance requirements of the individual control loops place demands on the real-time system with regard to sampling periods, latencies, and jitter.

Traditional scheduling models give poor support for codesign of multi-threaded real-time control systems. One difficulty lies in the nonlinearity in scheduling mechanisms such as rate-monotonic (RM) or earliest-deadline-first (EDF) scheduling: a small change in a task parameter—e.g., period, execution time, deadline, or priority—may give rise to unpredictable results in terms of input-output latency and jitter. This is crucial, since the performance of a controller depends not only on its sampling period, but also on the latency and the jitter. In the control design, it is straight-forward to account for a constant latency, while it is difficult to address varying or unknown delays.

In the seminal Liu and Layland paper [1], it is assumed that I/O is performed periodically by hardware functions, introducing a one-sample delay in all control loops closed over the computer. This scheme does provide a quite nice separation between the scheduling design and the control design. From a scheduling perspective, the controller can be described by a periodic task with a period T , a computation time C , and a deadline $D = T$. From a control perspective, the controller will have a sampling period of T and a constant latency $L = T$. This allows the control design and the real-time design to be carried out in relative isolation.

However, the one-sample latency degrades the control performance and is ultimately a waste of resources. A common alternative implementation is therefore to perform the I/O requests within the task loop and output the control signal as soon as possible in each period (e.g., [2, 3]). At this point, however, the design problem becomes very complicated. The I/O jitter and latency of a controller are now affected by variations in its own execution time as well as interference from higher-priority tasks (which in turn depend on the variations in the task execution times, the phasing of the periodic tasks, the arrival pattern of sporadic tasks, etc.). In the best case, it may be possible to derive formulas for the worst-case and best-case response times of the tasks (e.g., [4, 5]), but this information is still not sufficient to accurately predict the performance of the controllers. Furthermore, as argued in [6], with standard RM and EDF scheduling it can be difficult to map task importance into priorities

and/or deadlines. These algorithms also perform poorly if tasks deviate from their assumed behavior or if the CPU should become overloaded.

30.1.1 Model Overview

This paper presents a novel computational model for control tasks, called the control server. The primary goal of the model is to facilitate simple codesign of flexible real-time control systems. In particular, the model should provide

- (R1) isolation between unrelated tasks,
- (R2) short input-output latencies,
- (R3) minimal sampling jitter and input-output jitter,
- (R4) a simple interface between the control design and the real-time design,
- (R5) predictable control and real-time behavior, also in the case of overruns, and
- (R6) the possibility to combine several tasks (components) into a new task (component) with predictable control and real-time behavior.

Requirement (R1) is fulfilled by the use of constant bandwidth servers (CBSs) [7]. The servers make each task appear as if it was running on a dedicated CPU with a given fraction of the original CPU speed. To facilitate short latencies (requirement (R2)), a task may be divided into a number of *segments*, which are scheduled individually. A task may only read inputs (from the environment or from other tasks) at the beginning of a segment and write outputs (to the environment or to other tasks) at the end of a segment. All communication is handled by the kernel and is hence not prone to jitter (requirement (R3)).

Requirements (R4)–(R6) are addressed by the combination of bandwidth servers and statically scheduled communication points. For periodic tasks with constant execution times, the model creates the illusion of a perfect division of the CPU, equivalent to the Generalized Processor Sharing (GPS) algorithm [8]. The model makes it possible to analyze each task in isolation, from both scheduling and control points of view. Like ordinary EDF, schedulability of the task set is simply determined by the total CPU utilization (ignoring context switches and the I/O operations performed by the kernel). The performance of a controller can also be viewed as a function of its allotted CPU share. These properties make the model very suitable for feedback scheduling applications.

Furthermore, the model makes it possible to combine two or several communicating tasks into a new task. The new task will consume a

fraction of the CPU equal to the sum of the utilization of the constituting tasks. The new task will have a predictable I/O pattern, and, hence, also predictable control performance. Control tasks may thus be treated as *real-time components*, which can be combined into new components.

30.1.2 Related Work

The constant bandwidth server (CBS) [7] was originally proposed as a means to bound the utilization of soft or aperiodic real-time. A CBS creates the abstraction of a virtual CPU with a given capacity (or *bandwidth*) U_s . Tasks executing within the CBS cannot consume more than the reserved capacity. Hence, from the outside, the CBS will appear as an ordinary EDF task with a maximum utilization of U_s . The time granularity of the virtual CPU abstraction is determined by the *server period* T_s .

In [9], a variant of the CBS server, called CBS^{hd}, is used to schedule control tasks with varying execution times. In the case of an execution overrun, the current period is extended and the CBS budget is recharged in small increments until the task finishes.

Minimizing jitter using high-priority tasks or interrupt handlers has been suggested in various settings, e.g., [10, 2, 11]. Disadvantages of the approach include a more complex implementation and more runtime overhead. Also, reducing jitter means increasing the average input-output latency. In [12], a design procedure that minimizes input-output jitter using high-priority input and output tasks is presented. Task attribute assignment under both FP and EDF scheduling is considered. Another option to reduce input-output jitter is to use non-preemptive scheduling. Given that the control algorithm has a constant execution time, this will make the input-output latency constant. The drawback is that the scheduling design becomes more complicated.

Giotto [13] is an abstract programming model for the implementation of embedded control systems. Similar to our model, I/O and communication are time-triggered and assumed to take zero time, while the computations inbetween are assumed to be scheduled in real-time. A serious drawback with the model is that a minimum of one sample input-output latency is introduced in all control loops. Also, Giotto does not address the scheduling problem.

Within the Ptolemy project, e.g., [14], a computational domain called Timed Multitasking (TM) has been developed [15]. In the model, tasks (or *actors* in the terminology of Ptolemy) may be triggered by both periodic and aperiodic events. Inputs are read when the task is triggered and outputs are written at the specified task deadline. The computations inbetween are assumed to be scheduled by a fixed-priority dispatcher. In

the case of a deadline overrun, an overrun handler may be called. Again, the scheduling problem is not explicitly addressed by the model.

30.2 The Model

The control server model assumes an underlying real-time operating system with an EDF scheduler. To guarantee isolation, all tasks in the system must belong to either one of two categories:

- Control server tasks, suitable for control loops and other periodic activities with high demands for input/output timing accuracy.
- Tasks served by ordinary CBS servers, including aperiodic, soft and non-real-time tasks.

30.2.1 Control Server Tasks

A control server task τ_i is described by

- a CPU share U_i ,
- a period T_i ,
- a release offset ϕ_i ,
- a set of $n_i \geq 1$ segments $S_i^1, S_i^2, \dots, S_i^{n_i}$ of lengths $l_i^1, l_i^2, \dots, l_i^{n_i}$ such that $\sum_{j=1}^{n_i} l_i^j = T_i$,
- a set of inputs I_i (associated with physical inputs or shared variables), and
- a set of outputs O_i (associated with physical outputs or shared variables).

Associated with each segment S_i^j are

- a subset of the task inputs, $I_i^j \in I_i$,
- a code function f_i^j , and
- a subset of the task outputs, $O_i^j \in O_i$,

The segments can be thought of as a static cyclic schedule for the reading of inputs, the writing of outputs, and the release of jobs. At the beginning of a segment S_i^j , i.e., when $t = \phi_i + \sum_{k=1}^{j-1} l_i^k \pmod{T_i}$, the inputs I_i^j are read and a job executing f_i^j is released. At the end of the segment, i.e., when $t = \phi_i + \sum_{k=1}^j l_i^k \pmod{T_i}$, the outputs O_i^j are written.

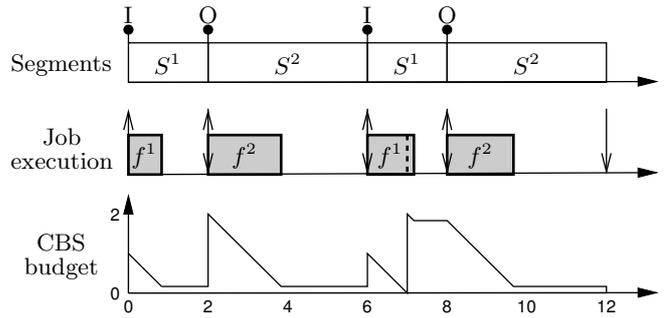


Figure 30.1: Example of a control server task executing alone. The up arrows indicate job releases and the down arrows indicate deadlines. The overrun at $t = 7$ causes the deadline to be postponed to the end of the next segment.

The jobs produced by a control server task τ_i are served on a first-come, first-served basis by a dedicated, slightly modified CBS with the following attributes:

- a server bandwidth equal to the CPU share U_i ,
- a dynamic deadline d_i ,
- a server budget c_i , and
- a segment counter m_i .

The server is initialized with $c_i = m_i = 0$ and $d_i = \phi_i$. The rules for updating the server are as follows:

1. During the execution of a job, the budget c_i is decreased at unit rate.
2. If $c_i = 0$, or, if a new job arrives at time r and $d_i = r$, then
 - the segment counter is updated, $m_i := \text{mod}(m_i, n_i) + 1$,
 - the deadline is moved, $d_i := d_i + l_i^{m_i}$, and
 - the budget is recharged to $c_i := U_i l_i^{m_i}$.

The rules are somewhat simplified compared to the original CBS rules [7] due to the predictable pattern of release times and deadlines. The only real difference from an ordinary CBS is that here a “dynamic server period”, equal to the current segment length, $l_i^{m_i}$, is used.

Fig. 30.1 shows an example of a control server task with two segments executing alone. This is a typical model of a control algorithm, which has been split into two parts, Calculate Output and Update State. The lengths of the segments are 2 and 4 units respectively, and the task

CPU share is $U = 0.5$. At the beginning of the first segment, an input is read, and at the end of the first segment, an output is written. The two first jobs consume less than their budgets (which are 1 and 2 units respectively), while the third job has an overrun at time 7. This causes the deadline to be moved to the end of the next segment and the budget to be recharged to 2 units (hence “borrowing” budget from the fourth job). In this example, the latency is constant and equal to 2 units (the length of the first segment) despite the variation in the job execution times.

30.2.2 Aborted Computations

The default behavior of the control server is to allow budget recharging across the task period. In the case of constant overruns, this will cause the task deadline to be postponed repeatedly and eventually reach infinity. For some applications, a better choice may be to abort the task when the total budget in the period has been exhausted. The choice of whether to abort the task in case of a period budget overrun may be specified separately for each control server task.

30.2.3 Communication and Synchronization

The communication between tasks and the environment requires some amount of buffering. When an input is read at the beginning of a segment, the value is stored in a buffer. The value in the buffer is then read from user code using a real-time primitive. The read operation is non-blocking and non-consuming, i.e., a value will always be present in the buffer and the same value can be read several times. Similarly, another real-time primitive is used to write a new output value. The value is stored in a buffer and is written to the output at the end of the relevant segment.

Communication between tasks is handled via shared variables. If an input is associated with a shared variable, the value of the variable is copied to the input buffer at the beginning of the relevant segment. Similarly, if an output is associated with a shared variable, the value in the output buffer is copied to the shared variable at the end of the relevant segment. Interrupts are assumed to be disabled when accessing buffers and shared data.

If two tasks should write to the same physical output or shared variable at the same time, the actual write order is undefined. More importantly, if one task writes to a shared variable and another task reads from the same variable at the same time, *the write operation takes place first*. The offsets can hence be used to line up tasks such that the

output from one task is immediately read by another task, minimizing the end-to-end latency.

The use of buffers and non-blocking read and write operations allow tasks with different periods to communicate. The periods of two communicating tasks need not be harmonic, even if this makes most sense in typical applications. However, for the kernel to be able to accurately determine if a read and write operation really occurs simultaneously, the offsets, periods, and segment lengths of a set of communicating tasks need to be integer multiples of a common tick size. For this purpose, communicating tasks are gathered into *task groups*. This is described further in the implementation section.

30.2.4 Scheduling Properties

From a schedulability point of view, a control server task with the CPU share U_i is equivalent to a CBS server with the bandwidth U_i . In [16], it is shown that a CBS can never demand more than its reserved bandwidth. By postponing the deadline when the budget is exhausted, the loading factor of the jobs served by the CBS can never exceed U_i . The same argument holds for the modified CBS used in the control server model. A set of CBS and control server tasks is thus schedulable if and only if

$$\sum U_i \leq 1. \quad (30.1)$$

If the segment lengths of a control server task τ_i are chosen such that

$$l_i^j = C_i^j / U_i, \quad (30.2)$$

where C_i^j denotes the worst-case execution time (WCET) of the code function f_i^j , overruns will never occur (i.e., the budget will never be exhausted before the end of the segment), and all latencies will be constant. For tasks with large variation in their execution time, it can sometimes be advantageous to assign segment lengths that are shorter than those given by (30.2). This means that some deadlines will be postponed and that the task may not always produce a new output in time, delaying the output one or more periods. An example of when this can actually give better control performance (for a given value of U_i) is given later.

30.3 Control and Scheduling Codesign

As stated in the introductory chapter of the thesis, the control and scheduling codesign problem can be formulated as follows: Given a set of processes to be controlled and a computer with limited resources, a set of controllers should be designed and scheduled as real-time tasks such that the overall control performance is optimized. With dynamic

scheduling algorithms such as EDF and RM, the general design problem is extremely difficult due to the complex interaction between task parameters, control parameters, schedulability, and control performance.

30.3.1 Control Performance

With our model, the link between the scheduling design and the control design is the CPU share U . Schedulability of a task set is simply determined by the total CPU utilization. The performance (or *cost*) J of a controller executing in a real-time system can—roughly speaking—be expressed as a function of the sampling period T , the input-output latency L_{io} , the sampling jitter J_s , and the input-output jitter J_{io} :

$$J = J(T, L_{io}, J_s, J_{io}). \quad (30.3)$$

Assuming that the first segment contains the Calculate Output part of the control algorithm, and that the segment lengths are chosen according to (30.2), execution under the Control Server implies

$$\begin{aligned} T &= \sum l^k = \sum C^k/U, \\ L_{io} &= l^1 = C^1/U, \\ J_s &= 0, \\ J_{io} &= 0. \end{aligned} \quad (30.4)$$

The only independent variable in the expressions above is U . The control performance can thus be expressed as a function of U only:

$$J = J(U). \quad (30.5)$$

Assuming a linear controller, a linear plant, and a quadratic cost function, the performance of the controller for different values of U can easily be computed using the Matlab toolbox Jitterbug [17].

The elimination of the jitter has several advantages. First, it is easy to design a controller that compensates for a constant delay. Second, the performance degradation associated with the jitter is removed. Third, it becomes possible to accurately predict the performance of the controller. These properties are exploited in the codesign examples below.

30.3.2 Codesign Example 1: Importance of Reducing Latency

Consider optimal control of the integrator process

$$\frac{dx(t)}{dt} = u(t) + v_c(t). \quad (30.6)$$

Table 30.1: Relative CPU demand of the integrator controller for different relative latencies to give the same performance.

L/T	CPU demand
1	1.00
0.5	0.72
0.25	0.58
0.1	0.50

Here, x is the state (which should be controlled to zero), u is the control signal, and v_c is a continuous-time white noise disturbance with zero mean and unit variance. A discrete-time controller is designed to minimize the continuous-time cost function

$$J = \lim_{t \rightarrow \infty} \frac{1}{t} \int_0^t x^2(s) ds. \quad (30.7)$$

Dividing the control computations into two segments and choosing the segment lengths in proportion to WCET of the parts, the control server model will generate equidistant sampling with the interval T and a constant latency L . The cost for the optimal, delay-compensating controller can be shown to be

$$J(T, L) = \frac{3 + \sqrt{3}}{6} T + L (\approx 0.79T + L). \quad (30.8)$$

(For details, see Appendix A.) It can be noted that, in this case, the cost grows linearly with both the sampling interval and the latency. Furthermore, for a fixed value of J (i.e., a specified level of performance), T is determined by L . This implies that a controller with a short latency will be less CPU-demanding than a controller with a long latency. In Table 1, the relative CPU demand of the integrator controller has been computed for different values of the relative latency L/T . The case $L/T = 1$ corresponds to a Liu and Layland implementation with a one sample delay. As the latency is reduced (by, e.g., a suitable division of the control algorithm into a Calculate Output segment and an Update State segment), the CPU demand of the controller can be decreased.

30.3.3 Codesign Example 2: Optimal Period Selection

In this example we study the problem of optimal sampling period selection for a set of control loops. This type of codesign problem first appeared in [18]. In those cases, however, the scheduling-induced latency and jitter was ignored.

Suppose for instance that we want to control three identical integrator processes,

$$\frac{dx(t)}{dt} = u(t) + v_c(t), \quad (30.9)$$

where x is the state, u is the control signal, and v_c is a continuous-time white noise process with zero mean and unit variance. A discrete-time controller is designed to minimize the continuous-time cost function

$$J = \lim_{T \rightarrow \infty} \frac{1}{T} \int_0^T x^2(t) dt. \quad (30.10)$$

assuming the sampling period h and a constant input-output latency L . The cost of the optimal controller is given by

$$J(h, L) = \frac{3 + \sqrt{3}}{6} h + L \quad (30.11)$$

(see Appendix I). The assumed design goal is to select sampling periods h_1, h_2, h_3 such that a weighted sum of the cost functions,

$$J_{tot} = w_1 J(h_1, L_1) + w_2 J(h_2, L_2) + w_3 J(h_3, L_3), \quad (30.12)$$

is minimized subject to the utilization constraint

$$U = \frac{C}{h_1} + \frac{C}{h_2} + \frac{C}{h_3} \leq 1.$$

Here, C is the (constant) total execution time of the control algorithm. Assigning segment lengths proportional to the parts of the algorithm, the control server model implies the same relative latency $a = L/h$ for all controllers. Using (30.11) the objective function (30.12) can be written

$$J_{tot} = \left(\frac{3 + \sqrt{3}}{6} + a \right) (w_1 h_1 + w_2 h_2 + w_3 h_3).$$

The solution to the optimization problem is

$$h_1 = b/\sqrt{w_1}, \quad h_2 = b/\sqrt{w_2}, \quad h_3 = b/\sqrt{w_3},$$

where $b = C(\sqrt{w_1} + \sqrt{w_2} + \sqrt{w_3})$. (For more general problems numerical optimization must be performed.) Contrary to [18] (where RM or EDF scheduling is assumed), our model allows for the latency and the (non-existent) jitter to be accounted for in the optimization.

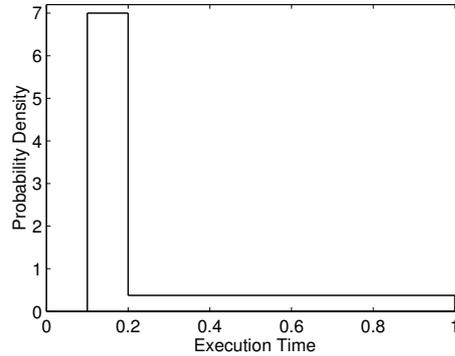


Figure 30.2: Assumed execution time probability distribution of the integrator controller.

30.3.4 Codesign Example 3: Overrun Handling

For controllers with large variations in their execution time, it can sometimes be pessimistic to select task periods (and segment lengths) according to the WCETs. The intuition is that, given a task CPU share, it may be better to sample often and occasionally miss an output, than to sample seldom and always produce an output. With our model, it becomes easy to predict the worst-case effects (i.e., assuming that the rest of the CPU is fully utilized) of such task overruns.

Again consider the integrator controller. For simplicity, it is assumed that the controller is implemented as a single segment, i.e., we have $L_{io} = T$ if no overrun occurs, and that the assigned CPU share is $U = 1$. The controller is designed for a constant latency of $L_{io} = T$. Now assume that the execution time of the controller is given by the probability distribution in Fig. 30.2.

Two cases are compared. First, budget recharging across the period is allowed. If an overrun occurs, the computation continues in the next period, where an output is eventually produced. In the second case, it is the task is aborted in the case of an overrun. This means that no new output is produced, and a new computation is started in the next period. The control performance in the different cases has been computed using Jitterbug [17]. In Fig. 30.3, the cost (30.7) has been computed for different values of the task period. When period overruns are allowed, the optimal cost is lower, $J = 1.55$, and occurs when $T = 0.5$. If task abortions are used, the optimal cost $J = 1.67$ occurs for $T = 0.76$. In this example, the default control server behavior with budget recharging across the task period yields the best control performance.

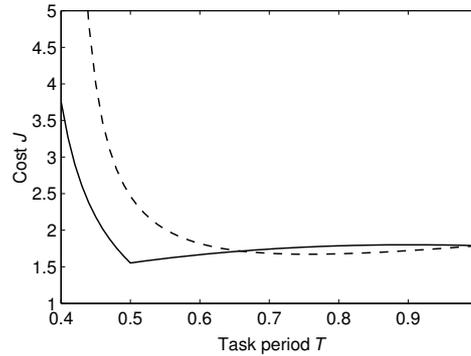


Figure 30.3: Cost as a function of the task period for the integrator controller, in the case of budget recharging across the period (full) and aborted computations (dashed). In both cases, assigning a task period shorter than the worst-case execution time can give better control performance.

30.4 Control Server Tasks as Real-Time Components

As argued in the previous section, given a control algorithm with known execution time C (divided into one or several segments), the sampling period T , the latency L_{io} , and the control performance J can be expressed as functions of the CPU share U . The predictable control and scheduling properties allows a control server task to be viewed as a scalable real-time component.

Consider for instance the PID (proportional-integral-derivative) controller component in Fig. 30.4. The controller has two inputs: the reference value r and the measurement signal y , and one output: the control signal u . The U knob determines the CPU share. An ordinary software component (see, e.g., [19]) would only specify the functional behavior, i.e., the PID algorithm. The specification for our real-time component includes the resource usage and the timely behavior. Assuming an implementation where the execution time of the Calculate Output part is $C^1 = 3.3$ ms and the execution time of the Update State part is

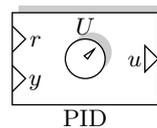


Figure 30.4: A PID controller component. The U knob determines both the scheduling and the control properties of the component.

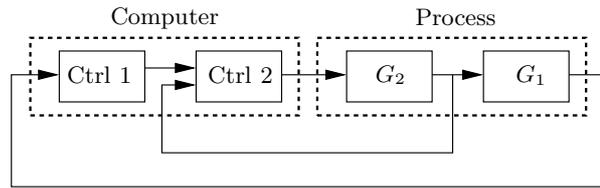


Figure 30.5: Cascaded controller structure.

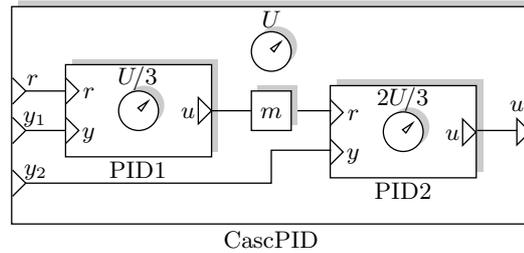


Figure 30.6: A cascaded PID controller component.

$C^2 = 10.0$ ms, the specification of the PID component could look something like this:

- Parameters: U (CPU share), K , T_i , T_d (PID parameters)
- Execution time: $C = 13.3$ ms
- Task period: $T = C/U$
- Latency: $L_{io} = C^1/U = T/4$
- Algorithm: $u = K \left((r - y) + \int \frac{1}{T_i} (r - y) dt + T_d \frac{-dy}{dt} \right)$, discretized using backward difference with interval T

Note that our model guarantees that the controller will have the specified behavior, regardless of other tasks in the system.

Next, consider the composition of two PID controllers in a cascaded controller structure, see Fig. 30.5. In this very common structure, the inner controller is responsible for controlling the (typically) fast process dynamics G_2 , while the outer controller handles the slower dynamics G_1 . A cascaded controller component can be built from two PID components as shown in Fig. 30.6. In this case, it is assumed that the inner controller should have twice the sampling frequency of the outer controller (reflecting the speed of the processes). This is achieved by assigning the shares $U/3$ to PID1 and $2U/3$ to PID2, U being the CPU share of the composite controller. The end-to-end latency in the controller can be minimized by a suitable segment layout, see Fig. 30.7.

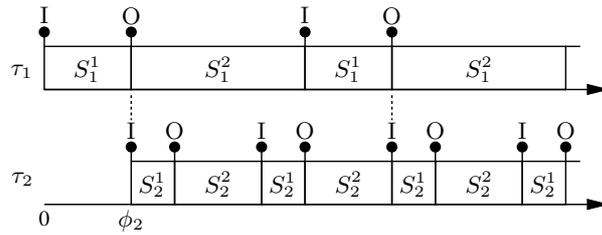


Figure 30.7: Segment layout in the cascaded PID controller. Task τ_2 is given an offset $\phi_2 = l_1^1$ such that the value written by S_1^1 is immediately read by S_2^1 .

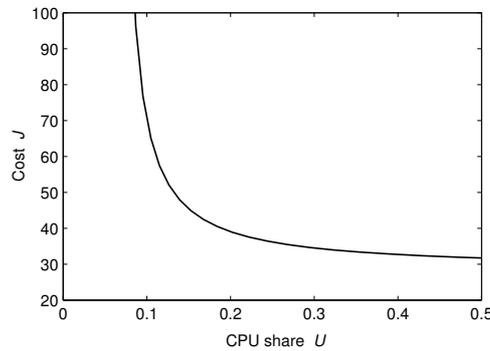


Figure 30.8: Control performance as a function of the CPU share for a cascaded PID controller.

The schedulability and performance of the cascaded controller will, again, only depend on the total assigned CPU share U . The resulting controller is a multi-rate controller and its performance can be computed using Jitterbug [17]. As an example, in Fig. 30.8, the cost J as a function of the CPU share U has been computed for a cascaded PID controlling a ball and beam process.

Note that such composition is not possible with ordinary threads, i.e., two communicating threads cannot be treated as one, neither from schedulability nor control perspectives.

30.5 Implementation

As a proof of concept, the computational model has been implemented in the public-domain real-time kernel STORK [20], developed at the Department of Automatic Control, Lund Institute of Technology. The original kernel is a standard priority-preemptive real-time kernel written in Modula-2, running on multiple platforms. For this project, the Mo-

Figure 30.9: Pseudo code for the modified real-kernel.

```

void schedule() { // Called by interrupt handler
    now = PowerPC.GetTB(); // Read hardware clock
    exectime = now - lastTime;
    if (task is associated with a CBS) {
        Decrease CBS budget by exectime;
        if (budget <= 0) {
            Update budget and deadline;
        }
    }
    for (each timer in the timer queue) {
        if (now >= expiry) {
            Run handler;
        }
    }
    for (each task in the time queue) {
        if (now >= release) {
            Move task to ready queue;
            if (task is associated with a CBS) {
                Update budget and deadline;
            }
        }
    }
    Make the first ready task the running task;
    if (task is associated with a CBS) {
        Set up CBS timer;
    }
    Determine next wake-up time;
    Set up new timer interrupt;
    lastTime = PowerPC.GetTB();
    Record context switch in log; // For traces
    Transfer control to the running task;
}

```

torola PowerPC was chosen because of its high clock resolution (40 ns on a 100 MHz processor).

The kernel was modified to use EDF as the basic scheduling policy, and high-resolution timers (hardware clock interrupts that trigger user-defined handlers) were introduced. For tracing purposes, the kernel measures the execution-time of each task. An outline of the kernel code is shown in Fig. 30.9.

A number of data structures for CBS servers, control server tasks, segments, inputs, and outputs, etc., were introduced, see the UML diagram in Fig. 30.10. The tasks in the ready queue are sorted according to their absolute deadlines. Tasks that are associated with a CBS inherit

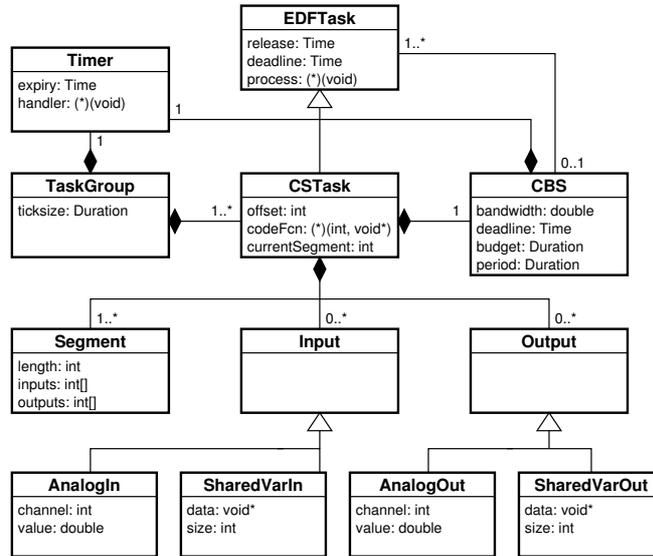


Figure 30.10: The various data structures in the implementation.

the deadline of the CBS. Note that several tasks may be served by the same CBS.

Each CBS is implemented using a timer. When a served task starts to execute, the expiry time of the timer is set to the time when the budget is expected to be exhausted. When the CBS is preempted or idle, the timer is disabled. A CBS that is associated with a control server task uses the segment information to determine how much the budget should be recharged and how much the deadline should be postponed.

30.5.1 Task Group Timing

For synchronization reasons, communicating control server tasks must share a common timebase and are gathered in task groups. Each task group uses a timer to trigger the reading of inputs, writing of outputs, and release of segments of tasks within the group. The structure of the task group timer interrupt handler is shown in Fig. 30.11.

Associated with each control server task is a semaphore that is used to handle the release of the segment jobs. Internally, every control server task is implemented as a simple loop, see Fig. 30.12.

30.5.2 API

The kernel provides a number of primitives for defining task groups, EDF tasks, CBS servers, control server tasks, inputs and outputs, etc. The code of a control server task is written according to a special format, here illustrated with a PID controller (written in Modula-2), see Fig. 30.13.

Figure 30.11: Pseudo code for the task group timing.

```

for (each task in the task group) {
    if (current segment is finished) {
        Write outputs; // (if any)
        Increase segment counter;
    }
}
for (each task in the task group) {
    if (a new segment should begin) {
        Read inputs; // (if any)
        Release segment job; // signal semaphore
    }
}
Determine next interrupt time;
Set up timer;

```

Figure 30.12: Pseudo code for the internal implementation of a control server task.

```

while (true) {
    Increase segment counter;
    Wait on semaphore;
    Call codeFcn(segment,data);
}

```

In the code, the kernel primitives `ReadInput` and `WriteOutput` are used to access the inputs and outputs associated with the segment. To prevent shared data from being corrupted, interrupts are disabled in the read and write primitives.

30.6 Control Experiments

The implementation of the control server was validated in a number of real-time control experiments on the ball and beam process, see Fig. 30.14. The objective of the control is to move the ball to a given position on the beam. The input to the process is the beam motor voltage, and the outputs are voltages representing the beam angle and the ball position.

The process is controlled by a multirate cascaded PID controller, where the inner controller executes at twice the frequency of the outer controller (see Figs. 30.5–30.7). The composite controller is assigned the CPU share $U = 0.5$. The execution time of the PID control algorithm

Figure 30.13: Code function in Modula-2 representing a control server task.

```

PROCEDURE PIDTask(seg: CARDINAL; data: PIDData);
VAR r, y, u: LONGREAL;
BEGIN
  CASE seg OF
    1: r := ReadInput(1);
       y := ReadInput(2);
       u := PID.CalculateOutput(data, r, y);
       WriteOutput(1, u);
       |
    2: PID.UpdateState(data);
  END;
END PIDTask;

```



Figure 30.14: The ball and beam process used in the control experiments.

is $C = 13.3$ ms, divided into a Calculate Output segment with $C^1 = 3.3$ ms and an Update State segment with $C^2 = 10$ ms. (To generate a high CPU load on the Power PC, busy cycles were inserted into the code.) The resulting sampling periods are $T_1 = 80$ ms for the outer PID controller and $T_2 = 40$ ms for the inner PID controller.

Also executing in the system is a sporadic task with an *assumed* minimum interarrival time of $T_{spor} = 20$ ms, and a worst-case execution time of $C_{spor} = 10$ ms. The actual execution time of the task is random and uniformly distributed between 2 and 10 ms. In the time interval 0 to 20, the actual interarrival time of the task is uniformly distributed between 20 and 40 ms. The average utilization of the sporadic task in this interval is 0.21. After $t = 20$ s, the interarrival time of the task suddenly decreases (hence violating the design assumptions) and

is from then on uniformly distributed between 5 and 10 ms. The new average utilization of the sporadic task is 0.83, causing the CPU to be overloaded.

The behavior of the system under rate-monotonic scheduling, earliest-deadline-first scheduling, and control server scheduling was studied. Under all policies, PID2 was released with an offset of 20 ms compared to PID1. The same random sequence of execution times for the sporadic task were used in all experiments. For each run, the execution trace (i.e., the task schedule) was logged, together with measurements of the ball position and the control signal.

30.6.1 Rate-Monotonic Scheduling

Under rate-monotonic scheduling, the sporadic task has the shortest assumed minimum interarrival time and is assigned the highest priority, while PID1 and PID2 are assigned low and medium priorities respectively. The task set is schedulable according to rate-monotonic theory.

The control performance under rate-monotonic scheduling is shown in Fig. 30.15, and a close-up of the execution trace is shown in Fig. 30.16. As expected, the control performance is good up to $t = 20$. When the sporadic task starts to misbehave, PID1 becomes completely blocked and the controller stops to work. The result is that the ball rolls off the beam.

30.6.2 Earliest-Deadline-First Scheduling

Under earliest-deadline-first scheduling, the tasks are scheduled according to their absolute deadlines. The PID tasks are assigned relative deadlines equal to their periods, while the sporadic task is assigned a relative deadline equal to its assumed minimum interarrival time.

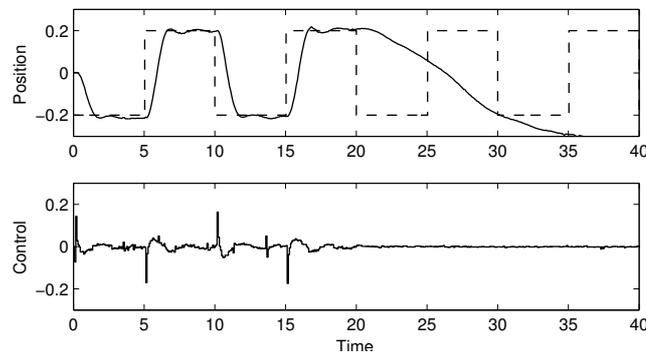


Figure 30.15: Control performance under rate-monotonic scheduling. The controller stops to work after $t = 20$.

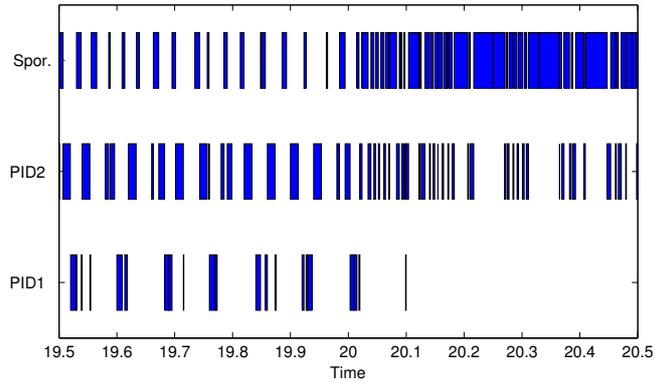


Figure 30.16: Execution trace under rate-monotonic scheduling. The sporadic task blocks the controller after $t = 20$.

The control performance under earliest-deadline-first scheduling is shown in Fig. 30.17, and a close-up of the execution trace is shown in Fig. 30.18. Again, the control performance is good up to $t = 20$. After that, the step responses are slower and more oscillatory, but the control system is still stable. The CPU overload caused by the sporadic task causes the sampling periods of the controller to be rescaled. This property of control tasks under overloaded EDF scheduling was first explained in [21].

30.6.3 Control Server Scheduling

Under control server scheduling, the cascade controller is assigned 50% of the processor, while the utilization of the sporadic task is bounded to 50% using a constant bandwidth server. The resulting control performance is shown in Fig. 30.19, and a close-up of the execution trace is

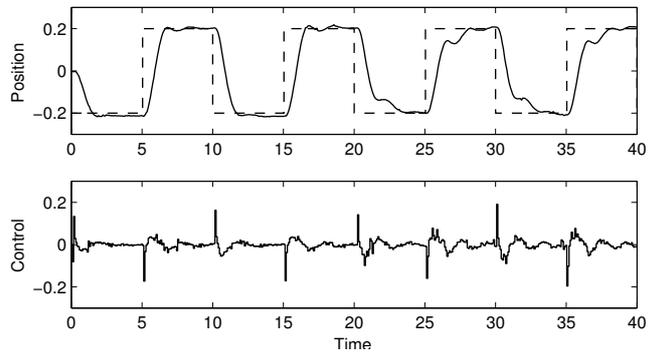


Figure 30.17: Control performance under earliest-deadline-first scheduling.

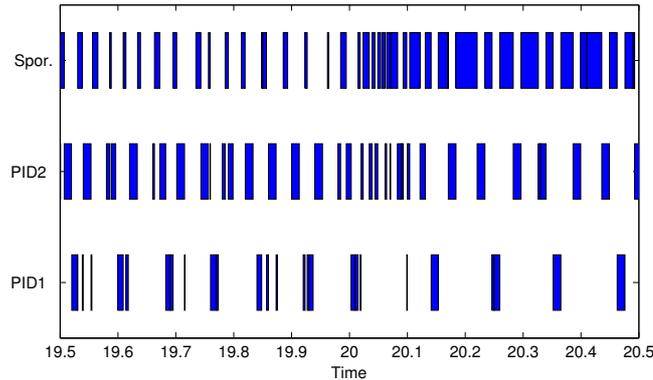


Figure 30.18: Execution trace under earliest-deadline-first scheduling.

shown in Fig. 30.20. Since the controller is no longer disturbed by the sporadic task, control performance is good throughout, and identical before and after $t = 20$. The schedule trace shows that the controller is no longer disturbed by the misbehaving sporadic task. Not shown in the trace is the fact that there is also no longer any I/O jitter.

30.7 Conclusion and Discussion of Future Work

This paper has presented the control server model, suitable for the implementation of feedback control tasks in embedded systems. Features of the model include small latency and jitter, something that is valuable from a control perspective. Based on the constant bandwidth server, the control server also provides isolation between unrelated tasks. A nice property of the control server is that both schedulability and control performance of a control task can be expressed through the task CPU

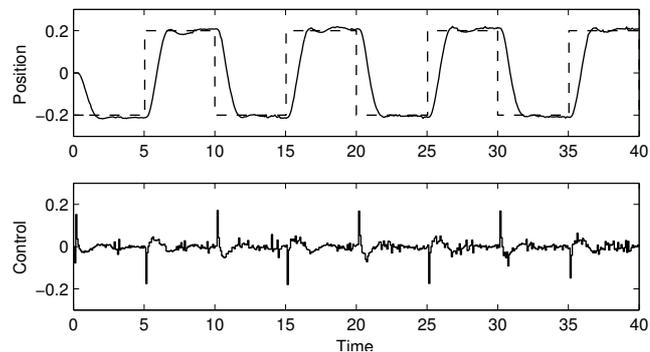


Figure 30.19: Control performance under control server scheduling. The performance is identical before and after $t = 20$.

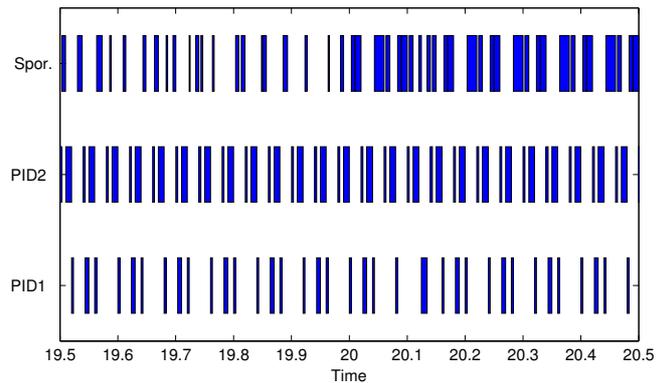


Figure 30.20: Execution trace under control server scheduling.

share. The control server has been implemented and tested in a real control application with good results.

The current work can be extended in many directions. One topic that needs further investigation is that of overrun handling. In Section 30.3.4, an example was given where budget recharging across the periods was shown to give better control performance than aborted computations. It is also possible to find examples where the opposite is true. Can any general conclusions can be drawn? The problem of overrun handling becomes especially intricate when the control algorithm has been divided into segments. For instance, should the Update State part be executed even if the Calculate Output part was aborted (or did not finish in time)? Are some controller realizations more sensitive to aborted computations than others? Can controllers be designed to be robust against period overruns?

To provide better performance, the constant bandwidth servers used could be modified to use a slack stealing algorithm. When the system is under-utilized, tasks would then be able to exceed their budgets without unnecessary deadline postponements. Interesting possibilities for slack stealing include the CASH algorithm [22] and the GRUB algorithm [23].

The analysis in the paper builds on the simplified assumption that all communication (including interrupt handlers and I/O) takes zero time. Possibilities for more detailed analysis of interrupt times under EDF scheduling are found in [1] (“mixed scheduling”) and in [24]. It would also be interesting to develop a variant of the control server for distributed systems. Components of the same controller could then be located at different nodes in a network. The communication times would now have to be taken into account, and a suitable network scheduling policy would have to be used.

The proposed server is based on EDF scheduling. Unfortunately, very few commercial real-time kernels support EDF scheduling today.

For backwards compatibility and industrial acceptance, it would be useful to develop a version of the control server which is based on priority-based scheduling. In fact, aperiodic task scheduling servers were originally developed for fixed-priority systems, [25], and a wealth of other algorithms have been developed since. Since EDF is an optimal scheduling policy, it cannot be expected that all results carry over to a fixed-priority setting.

Finally, automatic tuning of the control server parameters should be studied. It is often very hard to obtain a good estimates of the worst-case execution times of tasks. Using an on-line feedback mechanism, the segment lengths could be automatically adjusted such that overruns occur optimally often. Also, the division of the CPU among several competing control tasks could be performed by a feedback scheduler, as proposed in [21].

Appendix: Cost Calculation in Example 1

The delayed integrator process can be written

$$\frac{dx(t)}{dt} = u(t - L) + v_c(t), \quad 0 \leq L \leq T, \quad (30.13)$$

where L is the input-output latency, and T is the sampling interval. The cost function to be minimized can be written

$$J = \frac{1}{T} \mathbf{E} \left\{ \int_0^T x^2(t) dt \right\}. \quad (30.14)$$

Inserting the process description (30.13) into (30.14) gives

$$\begin{aligned} J &= \frac{1}{T} \mathbf{E} \left\{ \int_0^L \left(x(kT) + tu(kT - T) + \int_0^t v_c(s) ds \right)^2 dt \right. \\ &\quad \left. + \int_L^T \left(x(kT) + Lu(kT - T) + (t - L)u(kT) + \int_0^t v_c(s) ds \right)^2 dt \right\} \\ &= \frac{1}{T} \begin{bmatrix} x(kT) \\ u(kT - T) \\ u(kT) \end{bmatrix}^T \begin{bmatrix} Q_1 & Q_{12} \\ Q_{12}^T & Q_2 \end{bmatrix} \begin{bmatrix} x(kT) \\ u(kT - T) \\ u(kT) \end{bmatrix} + J_{samp}, \end{aligned} \quad (30.15)$$

where

$$Q_1 = \begin{bmatrix} T & \frac{L^2}{2} + (T-L)L \\ \frac{L^2}{2} + (T-L)L & \frac{L^3}{3} + (T-L)L^2 \end{bmatrix},$$

$$Q_{12} = \begin{bmatrix} \frac{(T-L)^2}{2} \\ \frac{(T-L)^2}{2} L \end{bmatrix}, \quad Q_2 = \frac{(T-L)^3}{3},$$

$$J_{samp} = \frac{1}{T} \int_0^T \int_0^t 1 \, ds \, dt = T/2.$$

Sampling the process (30.13) with the interval T gives

$$\begin{bmatrix} x(kT + T) \\ u(kT) \end{bmatrix} = \Phi \begin{bmatrix} x(kT) \\ u(kT - T) \end{bmatrix} + \Gamma u(kT) + v(kT), \quad (30.16)$$

where

$$\Phi = \begin{bmatrix} 1 & L \\ 0 & 0 \end{bmatrix}, \quad \Gamma = \begin{bmatrix} T - L \\ 1 \end{bmatrix},$$

and v is a discrete-time white noise process with covariance

$$R = \begin{bmatrix} T & 0 \\ 0 & 0 \end{bmatrix}.$$

Introduce the positive definite matrix S . The controller that minimizes the cost satisfies the algebraic Riccati equation (e.g. [3])

$$S = \Phi^T S \Phi + Q_1 - (\Phi^T S \Gamma + Q_{12}) (\Gamma^T S \Gamma + Q_2)^{-1} (\Gamma^T S \Phi + Q_{12}^T), \quad (30.17)$$

with the solution

$$S = \begin{bmatrix} L + \frac{\sqrt{3}T}{6} & \frac{L}{6} (3L - \sqrt{3}T) \\ \frac{L}{6} (3L - \sqrt{3}T) & \frac{L^3}{3} - \frac{\sqrt{3}L^2T}{6} \end{bmatrix}.$$

The optimal control law is

$$u(kT) = -K \begin{bmatrix} x(kT) \\ u(kT - T) \end{bmatrix}, \quad (30.18)$$

where

$$K = (Q_2 + \Gamma^T S \Gamma)^{-1} (\Gamma^T S \Phi + Q_{12}^T) = \begin{bmatrix} \frac{1}{T} \frac{\sqrt{3} + 3}{2 + \sqrt{3}} & \frac{L}{T} \frac{\sqrt{3} + 3}{2 + \sqrt{3}} \end{bmatrix},$$

and the optimal cost is given by

$$J = \frac{1}{T} \text{tr} SR + J_{samp} = \frac{3 + \sqrt{3}}{6} T + L. \quad (30.19)$$

Bibliography

- [1] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *Journal of the ACM*, vol. 20, no. 1, pp. 40–61, 1973.
- [2] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. Gonzalez Hrbour, *A Practitioner’s Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publisher, 1993.
- [3] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems*. Prentice Hall, 1997.
- [4] N. Audsley, K. Tindell, and A. Burns, “The end of the line for static cyclic scheduling,” in *Proc. 5th Euromicro Workshop on Real-Time Systems*, 1993.
- [5] O. Redell and M. Sanfridson, “Exact best-case response time analysis of fixed priority scheduled tasks,” in *Proc. 14th Euromicro Conference on Real-Time Systems*, Vienna, Austria, June 2002.
- [6] K. Jeffay and S. Goddard, “Rate-based resource allocation models for embedded systems,” in *Proc. First International Workshop on Embedded Software*, 2001.
- [7] L. Abeni and G. Buttazzo, “Integrating multimedia applications in hard real-time systems,” in *Proc. 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, 1998.
- [8] A. Parekh and R. Gallager, “A generalized processor sharing approach to flow control in integrated services networks: the single node case,” *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, 1993.
- [9] M. Caccamo, G. Buttazzo, and L. Sha, “Elastic feedback control,” in *Proc. 12th Euromicro Conference on Real-Time Systems*, Stockholm, Sweden, June 2000, pp. 121–128.
- [10] C. D. Locke, “Software architecture for hard real-time applications: Cyclic vs. fixed priority executives,” *Real-Time Systems*, vol. 4, pp. 37–53, 1992.
- [11] W. Halang, “Achieving jitter-free and predictable real-time control by accurately timed computer peripherals,” *Control Engineering Practice*, vol. 1, no. 6, pp. 979–987, 1993.

- [12] P. Balbastre, I. Ripoll, and A. Crespo, "Control task delay reduction under static and dynamic scheduling policies," in *Proc. 7th International Conference on Real-Time Computing Systems and Applications*, 2000.
- [13] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," in *Proc. First International Workshop on Embedded Software*, 2001.
- [14] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorfer, S. Sachs, and Y. Xiong, "Taming heterogeneity—the Ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.
- [15] J. Liu and E. Lee, "Timed multitasking for real-time embedded software," *IEEE Control Systems Magazine*, vol. 23, no. 1, Feb. 2003.
- [16] L. Abeni, "Server mechanisms for multimedia applications," Scuola Superiore S. Anna, Pisa, Italy, Tech. Rep. RETIS TR98-01, 1998.
- [17] B. Lincoln and A. Cervin, "Jitterbug: A tool for analysis of real-time control performance," in *Proceedings of the 41st IEEE Conference on Decision and Control*, Las Vegas, NV, Dec. 2002.
- [18] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On task schedulability in real-time control systems," in *Proc. 17th IEEE Real-Time Systems Symposium*, Washington, DC, 1996, pp. 13–21.
- [19] I. Crnkovic and M. Larsson, Eds., *Building Reliable Component-Based Software Systems*. Artech House Publishers, 2002.
- [20] L. Andersson and A. Blomdell, "A real-time programming environment and a real-time kernel," in *National Swedish Symposium on Real-Time Systems*, ser. Technical Report No 30 1991-06-21, L. Aspplund, Ed. Uppsala, Sweden: Dept. of Computer Systems, Uppsala University, 1991.
- [21] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, "Feedback-feedforward scheduling of control tasks," *Real-Time Systems*, vol. 23, no. 1, July 2002.
- [22] M. Caccamo, G. Buttazzo, and L. Sha, "Capacity sharing for overrun control," in *Proc. IEEE Real-Time Systems Symposium*, Orlando, Florida, 2000.
- [23] G. Lipari and S. Baruah, "Greedy reclamation of unused bandwidth in constant-bandwidth servers," in *Proc. Euromicro Conference on Real-Time Systems*, Stockholm, Sweden, 2000.

- [24] K. Jeffay and D. L. Stone, “Accounting for interrupt handling costs in dynamic priority systems,” in *Proc. 14th IEEE Real-Time Systems Symposium*, 1993.
- [25] B. Sprunt, L. Sha, and J. Lehoczky, “Aperiodic task scheduling for hard real-time systems,” *Real-Time Systems*, vol. 1, no. 1, 1989.

Appendix A

Projects and Courses

ARTES main efforts were the research projects and courses. The projects are presented in Table 1 and the courses in Table 2.

Abbreviations of university names:

BTH	Blekinge Institute of Technology
CTH	Chalmers University of Technology
HH	Halmstad University
HIS	University of Skövde
KTH	Royal Institute of Technology
LiU	Linköping University
LU	Lund University
MDH	Mälardalen University
SICS	Swedish Institute of Computer Science
UU	Uppsala University.

Table A.1: The ARTES (1-26b) and PAMP (27-34) project titles, leaders, affiliation, funding, start and students. No 35 was special funding from SSF to support a Professor Erik Hagersten at Uppsala University.

	Project titel	Projekt leader	Univ.1	Funding (kk)	Start	Graduate student(s)
1	Hardware-Software Co-Design of Real-Time Systems	Zebo Peng	LiU	2293	98-03-01	Paul Pop
2	Incremental Iterative Scheduling	Hans Hansson	MDH	825	98-07-01	Jukka Mäki-Turja
3	Integrated Control and Scheduling	Karl-Erik Årzén, Klas Nilsson, Ola Dahl	LU	4617	98-05-01	Anton Cervin, Sven Gestegård Robertz and Patrik Persson
4	Design of Heterogeneous Multiprocessor Systems for Real-Time Applications	Petru Eles	LiU	1775	99-09-01	Sorin Manolache
5	Real-time software for versatility, scalability and reconfigurability in complex embedded feedback control systems	Jan Wikander	KTH	2340	99-07-01	De Jiu Chen

	Project titel	Projekt leader	Univ.1	Funding (kkkr)	Start	Graduate student(s)
6	A tool environment for the development of embedded systems	Christer Norström, Hans Hanson	MDH	2080	99-01-01	Anders Wall
7	TATOO-Test and testability of distributed real-time systems	Hans Hansson	MDH	840	98-10-01	Henrik Thane
8	Applications of wait/lock-free protocols to real-time systems	Hans Hansson, Philippas Tsigas and Marina Papatriantafilou	CTH	3202	99-03-01	Håkan Sundell, Björn Allvin and Yi Zhang
9	Automatic Control in Distributed Applications (AIDA 2)	Martin Törngren	KTH	1650	00-02-14	Jad El-Khoury
10	Identification of Complexity-Reduction Techniques for Optimal Scheduling in Embedded Distributed Real-Time Systems	Jan Jonsson	CTH	2227	99-05-01	Cecilia Ekelin
11	Node-level Fault Tolerance for Fixed Priority Scheduling	Johan Karlsson	CTH	2140	99-04-01	Joakim Aidemark
12	Hierarchical Design and Analysis of Timed Systems	Wang Yi	UU	2120	99-01-01	Alexandre David

	Project titel	Projekt leader	Univ.1	Funding (kkkr)	Start	Graduate student(s)
13	New directions in Symbolic Model Checking for Real-Time Systems	Parosh Abdulla	UU	1880	99-08-01	Julien D'orso
14	Flexible reliable timing constraints	Gerhard Fohler	MDH	1612	00-02-01	Damir Iovic
15	Methods for Integration of Heterogeneous Real-Time Services into High-Performance Networks	Magnus Jonsson and Bertil Svensson	HH	1200	00-01-01	Carl Bergenhem
16	Real-time Mobile Communication	Per-Arne Wiberg and Bertil Svensson	HH	3360	00-01-01	Elisabeth Uhlemann and Urban Bilstrup
17	TETReS: Testing of Event-Triggered Real-Time Systems	Sten Andler and Jonas Mellin	HIS	2880	00-01-01	Birgitta Lindstrm and Robert Nilsson
18	Functional Integration and Interference in Embedded Control Systems	Martin Törngren and Jan Wikander	KTH	1100	00-12-01	Jonas Norberg
19	Distributed Real-Time Systems with Minimal Energy Consumption: Analysis and Synthesis	Krzysztof Kuchcinski	LU	1200	00-09-01	Flavius Gruian

	Project titel	Projekt leader	Univ.1	Funding (kk)	Start	Graduate student(s)
20	TATOO-Test and Testability of Distributed Real-Time Systems	Henrik Thane	MDH	1200	00-09-01	Anders Pettersson
21	Switched Real-Time Communication for Industrial Applications	Magnus Jonsson and Bertil Svensson	HH	1200	01-02-13	Hoai Hoang
22	Pre-Implementation Analysis of Distributed Control Systems - PICADOR	Martin Törngren, Ola Redell, Jan Wikander	KTH	1200	00-09-01	Ola Redell
23	RATAD - Reliability and Timing Analysis of Distributed Systems	Hans Hansson, Christer Norström	MDH	1200	00-11-01	Thomas Nolte
24	Extension of Flexible Reliable Timing Constraints	Gerhard Fohler	MDH	1200	00-09-01	Radu Dobrin
25	Real-Time Response and Control of Autonomous Agents	Nancy E. Reed	LiU	600	00-09-01	Paul Scerri
26a	Embedded Databases for Embedded Real-Time Systems (COMET)	Christer Norström, Jörgen Hansson	MDH	1200	00-11-01	Aleksandra Tesanovic
26b	Embedded Databases for Embedded Real-Time Systems (COMET)	Jörgen Hansson, Christer Norström	LiU	1200	00-12-01	Dag Nyström

	Project titel	Projekt leader	Univ.1	Funding (kk)	Start	Graduate student(s)
27	Simulation Concepts to Model Real-Time and Dependability Properties of Symmetric Multiprocessor Systems	Lars Albertsson	SICS	1850	99-05-01	Lars Albertsson
28	Predictable Parallel Protocol Processing	Per Gunningberg och Mats Björkman	UU	1200	00-06-01	Thiemo Voigt
29	Design Strategies for Real-Time High-Performance Multimedia Applications on Multiprocessors	Per Stenström and Jan Jonsson	CTH	3680	99-01-01	Jonas Lext, Björn Andersson and Ulf Assarsson
30	Design Guidelines and Visualization Support for Developing Parallel Real-Time Applications	Lars Lundberg	BTH	3121	99-01-01	Daniel Häggander and Magnus Broberg
31	Categorized and Specialized Caching for SMPs	Erik Hagersten	UU	1200	00-06-01	Martin Karlsson
32	Software Distributed Shared Memory - New Applications and Scalability	Mats Brorsson	KTH	948	01-01-01	Nguyen Phan and Nguyen Thai

	Project titel	Projekt leader	Univ.1	Funding (kkkr)	Start	Graduate student(s)
33	Techniques for Module-Level Speculative Parallelization on Shared-Memory Multiprocessors	Per Stenström and Lennart Pettersson	CTH	600	00-09-01	Fredrik Warg
34	Support for Real-Time 3-D Graphics on Future Mobile Terminals under Energy/Area Constraints	Per Stenström, Fredrik Dahlgren.	CTH	1440	00-12-01	Magnus Ekman
35	Professor in Computer Architecture	Erik Hagersten	UU	5980	99-04-01	
		SUM=		68360		

Table A.2: Courses supported by ARTES 1997-2003 and 2004-2005 by ARTES++. Note: University abbreviations are explained in Table A.1. The European Summer School on Embedded Systems (ESSES) was a joint Sweden - Korea -Denmark - ARTIST effort including 3 months of courses and workshops from July 14 until October 10, 2003.

Year, Term	Course name	Teacher(s)	Univ.	Pts.	Registered	Passed
1997, Autumn	Distributed Real-Time Systems	Sten F. Andler	HS	4	10	
	Design of Software for Embedded Real-time Control Systems	Martin Törngren	KTH	4	9	9
	Modelling and Analysis of Real-Time Systems	Hans Hansson and Wang Yi	UU	6	16	9
1998, Spring	Hardware/Software Codesign	Zebo Peng	LiU	4	18	9
1998, Autumn	Parallel and Distributed Real-Time Systems	Jan Jonsson	CTH	5	22	18
	Design of Software for Embedded Real-time Control Systems	Martin Törngren	KTH	4	7	5
2000, Spring	Computer clusters	Lars Lundberg	BTH	3		
	Component-based Software Engineering	Ivica Crnkovic	MDH	5	32	29

Year, Term	Course name	Teacher(s)	Univ.	Pts.	Registered	Passed
	Entreprenörskurs in Cooperation with VISIT	Magnus Klofsten			25	
2000, Autumn	Safety Critical Systems	Simin Nadjm-Therani	LiU	4	18	13
2001, Spring	Parallel and Distributed Real-Time Systems	Jan Jonsson	CTH	5	35	31
	Real-time Computer Control Systems	Martin Törngren	KTH	5	15	15
2002, Summer	Specification-based Software Testing	Sten F. Andler	HS	4	4	4
2002, Spring	Formalisms, Algorithms and Tools in Formal Methods for Real-Time	Hans Hansson and Wang Yi	MDH	3	15	11
2002, Autumn	Safety Critical Computer Control Systems	Martin Törngren	KTH	3	14	13
	Concurrency Theory and Time	Hans Hansson	MDH	3	12	12
2003	ESSES summer school					
2004, Spring	Parallel and Distributed Real-Time Systems	Jan Jonsson	CTH	5	6	6
	Embedded Control Systems	Karl-Erik Årzén	LU	5	13	13
	Formal Modelling and Analysis of Real-Time Systems	Paul Pettersson and Wang Yi	UU	5	20	13

Year, Term	Course name	Teacher(s)	Univ.	Pts.	Registered	Passed
2004, Autumn	Design of Embedded Real-Time Systems	Martin Törngren	KTH	5	15	14
	Safety Critical Systems	Simin Nadjm-Therani	LiU	5	6	5
2005, Spring	Research Planning 2004	Hans Hansson	UU	3	10	6
	Cooperating Embedded Systems	Tony Larsson	HH	5	11	11
	Advanced Component-Based Software Engineering	Ivica Crnkovic	MDH	5	11	10
	Storage Systems for Embedded Systems	Sang Lyul Min	UU	3	9	8
2005, Autumn	Introduction to Systems Thinking and its Application	Bud Lawson	HS	5	17	11
	Hardware/Software Codesign	Zebo Peng	LiU	5	7	n.a.
	Research Planning 2005	Hans Hansson	MDH	3	9	6
	Advanced Real-Time Scheduling	Gerhard Fohler	MDH	5	5	n.a.
2006, Spring	Parallel and Distributed Real-Time Systems	Jan Jonsson	CTH	5	4	n.a.
	Embedded Control Systems	Karl-Erik Årzén	LU	5	2	n.a.

Year, Term	Course name	Teacher(s)	Univ.	Pts.	Registered	Passed
2006, Autumn	Formal Modelling and Analysis of Real-Time Systems	Paul Pettersson and Wang Yi	UU	5	5	n.a.
	Real-Time Communication	Magnus Jonsson	HH	5	n.a.	n.a.
	Distributed Real-Time and Database Systems	Sten F. Andler	HS	5	n.a.	n.a.
	Design of Embedded Real-Time Systems	Martin Törngren	KTH	5	n.a.	n.a.
SUM=	35 course occasions			151	402	281

Appendix B

Theses by ARTES Real-Time Graduate Students

This appendix presents the theses of the graduate students supported by ARTES, together with a list of theses by students within the network not funded by ARTES.

B.1 Students funded by ARTES

The following is a lists in alphabetical order of the students supported by ARTES, indicating their projects (more details about the project are available in Appendix A.1) and theses. Abstracts of the most recent thesis of each student is provided in the subsequent section.

Aidemark, Joakim, ARTES project no. 11, CTH

-Licentiate, December 17, 2001, On the Design and Validation of Dependable Real-Time Systems.

-PhD, December 16, 2004, Node-level Fault Tolerance for Fixed Priority Scheduling

Andersson, Björn, ARTES project no. 29, CTH.

-Licentiate, June 8, 2001, Insights on Non-Partitioned Fixed-Priority Preemptive Scheduling.

-PhD, September 29, 2003, Static-priority scheduling on multiprocessors.

Assarsson, Ulf, ARTES project no. 29, CTH.

-Licentiate, May 28, 2001, View Frustum Culling and Animated Ray Tracing: Improvements and methodological Considerations.

-PhD, October 10, 2003, A Real-Time Soft Shadow Volume Algorithm.

- Bergenheim, Carl**, ARTES project no. 15, HH.
-Licentiate, December 19, 2002, Protocols with Heterogeneous Real-Time Services for High-Performance Embedded Networks .
- Bilstrup, Urban**, ARTES project no. 16, HH.
-Licentiate, June 7, 2005, Design Space Exploration of Wireless Multihop Networks.
- Broberg, Magnus**, ARTES project no. 30, BTH.
-Licentiate, June, 1999, An Approach for Performance Tuning of Multithreaded Applications on Multiprocessors.
-PhD, May 3, 2002, Performance Prediction and Improvement Techniques for Parallel Programs in Multiprocessors.
- Cervin, Anton**, ARTES project no. 3, LU.
-Licentiate, May 26, 2000, Towards the Integration of Control and Real-Time Scheduling Design.
-PhD, April 25, 2003, Integrated Control and Real-Time Scheduling.
- Chen, De Jiu**, ARTES project no. 5, KTH.
-Licentiate, March 30, 2001, Architecture for Systematic Development of Mechatronics Software Systems.
-PhD, June 9, 2004, Systems Modeling and Modularity Assessment for Embedded Computer Control Applications.
- David, Alexandre**, ARTES project no. 12, UU.
-Licentiate, October 12, 2001, Practical Verification of Real-time Systems.
-PhD, November 26, 2003, Hierarchical Modeling and Analysis of Real Time Systems.
- Dobrin, Radu**, ARTES project no. 24, MDH.
-Licentiate, May 2003, Transformation Methods for Off-line Schedules to Attributes for Fixed Priority Scheduling.
-PhD, September 27, 2005, Combining Off-line Schedule Construction and Fixed Priority Scheduling in Real-Time Computer Systems.
- d'Orso, Julien**, ARTES project no. 13, UU.
-PhD, November 25, 2003, New Directions in Symbolic Model Checking.
- Ekelin, Cecilia**, ARTES project no. 10, CTH.
-Licentiate, December 14, 2001, Scheduling of Embedded Real-Time Systems: A Constraint Programming Approach.
-PhD, May, 28, 2004, An Optimization Framework for Scheduling of Embedded Real-Time Systems.

Ekman, Magnus, ARTES project no. 34, CTH.

-Licentiate, September 12, 2003, Performance and Energy Aware Design Trade-Offs in Chip-Multiprocessors.

-PhD, December 20, 2004, Strategies to Reduce Energy and Resources in Chip Multiprocessor Systems.

El-Khoury, Jad, ARTES project no. 9, KTH.

-PhD, March 3, 2006, A Model Management and Integration Platform for Mechatronics Product Development.

Gestegård Robertz, Sven, ARTES project no. 3, LU.

-Licentiate, May 12, 2003, Flexible automatic memory management for real-time and embedded systems .

Gruian, Flavius, ARTES project no. 19, LU.

-PhD, December 17, 2002, Energy-Centric Scheduling for Real-Time Systems.

Hoang, Hoai, ARTES project no. 21, HH.

-Licentiate, May 27, 2003, Switched Real-Time Ethernet for Industrial Applications.

Häggander, Daniel, ARTES project no. 30, BTH.

-Licentiate, November 23, 1999, Software Design When Migrating to Multiprocessors.

-PhD, September 14, 2001, Software Design Conflicts, Maintainability versus Performance and Availability.

Isovic, Damir, ARTES project no. 14, MDH.

-Licentiate, June 8, 2001, Handling Sporadic Tasks in Real-time Systems - Combined Offline and Online Approach.

-PhD, November 9, 2004, Flexible Scheduling for Media Processing in Resource Constrained Real-Time Systems.

Karlsson, Martin, ARTES project no. 31, UU.

-Licentiate, September 26, 2003, Cache Memory Design Trade-offs for Current and Emerging Workloads.

-PhD, January 13, 2006, Memory System Design for Chip-Multiprocessors.

Manolache, Sorin, ARTES project no. 4, LiU.

-Licentiate, December 19, 2002, Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times.

-PhD, Dec 16, 2005, Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour.

Mäki-Turja, Jukka, ARTES project no. 2, MDH.

-PhD, May 27, 2005, Engineering Strength Response-Time Analysis — A Timing Analysis Approach for the Development of Real-Time Systems.

- Nolte, Thomas**, ARTES project no. 23, MDH.
-Licentiate, May 12, 2003, Reducing Pessimism and Increasing Flexibility in the Controller Area Network..
- Nyström, Dag**, ARTES project no. 26, MDH.
-Licentiate, May 12, 2003, COMET: A Component-Based Real-Time Database for Vehicle Control-Systems.
-PhD, October 26 2005, Data Management in Vehicle Control-Systems.
- Persson, Patrik**, ARTES project no. 3, LU.
-Licentiate, April 12, 2000, Predicting Time and Memory Demands of Object-Oriented Programs.
- Pettersson, Anders**, ARTES project no. 7, MDH.
-Licentiate, October 31, 2003, Analysis of Execution Behavior for Testing of Multi-Tasking Real-Time Systems.
- Pop, Paul**, ARTES project no. 1, LiU.
-Licentiate, June 14, 2000, Scheduling and Communication Synthesis for Distributed Real-Time Systems.
-PhD, June 16, 2003, Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems.
- Redell, Ola**, ARTES project no. 22, KTH.
-PhD, May 8, 2003, Response Time Analysis for Implementation of Distributed Control Systems.
- Scerri, Paul**, ARTES project no. 25, LiU.
-PhD, December 14, 2001, Designing Agents for Systems with Adjustable Autonomy.
- Sundell, Håkan**, ARTES project no. 8, CTH.
-Licentiate, March 6, 2002, Applications of Non-Blocking Data Structures to Real-Time Systems.
-PhD, Nov 5, 2004, Efficient and Practical Non-Blocking Data Structures.
- Tešanović, Aleksandra**, ARTES project no. 26, LiU.
-Licentiate, June 2, 2003, Towards Aspectual Component-Based Real-Time System Development.
- Thane, Henrik**, ARTES project no. 7, MDH.
-PhD, May 26, 2000, Monitoring, Testing and Debugging of Distributed Real-Time Systems.
- Uhlemann, Elisabeth**, ARTES project no. 16, HH.
-Licentiate, November 9, 2001, Hybrid ARQ Using Serially Concatenated Block Codes for Real-Time Communication - An Iterative Decoding Approach.
-PhD, October 22, 2004, Adaptive Concatenated Coding for Wireless Real-Time Communications.

Wall, Anders, ARTES project no. 6, MDH.

-Licentiate, Sept 15, 2000, A Formal Approach to the Analysis of Software Architectures for Real Time Systems.

-PhD, September 26, 2003, Architectural Modeling and Analysis of Complex Real-Time Systems.

Warg, Fredrik, ARTES project no. 33, CTH.

-Licentiate, June 6, 2003, Module-Level Speculative Execution Techniques on Chip Multiprocessors.

Voigt, Thiemo, ARTES project no. 28, UU.

-PhD, May 27, 2002, Architectures for Service Differentiation in Overloaded Internet Servers.

Zhang, Yi, ARTES project no. 8, CTH.

-PhD, June 12, 2003, Non-blocking Synchronization: Algorithms and Performance Evaluation.

B.2 Abstracts of theses of students funded by ARTES

The following is a list of abstracts of theses presented by students funded by ARTES. For each student, only the abstract of the most recent thesis is presented.

B.2.1 Aidemark, Joakim, ARTES project no. 11, CTH.

PhD, December 16, 2004, Node-level Fault Tolerance for Fixed Priority Scheduling.

This thesis deals with cost-effective design and validation of fault tolerant distributed real-time systems. Such systems play an increasingly important role in embedded applications such as automotive and aerospace systems. The cost of fault-tolerance is of primary concern in these systems, particularly for emerging applications like micro-satellites, unmanned air vehicles and active safety systems for road vehicles.

We address cost issues of fault tolerance from both a design and a validation perspective. From a design perspective, we investigate cost-effective techniques that can make systems more resilient to transient hardware faults. We propose a two-level approach to achieve fault-tolerance that combines system-level and node-level fault tolerance. Our approach relies on nodes that mask the effects of most transient faults and exhibit omission or fail-silent failures for permanent faults and transient faults that cannot be masked by the node itself. As only a subset of the faults is tolerated at the node level, we call this approach /light-weight node-level fault tolerance/, or light-weight NLFT. Tolerating transient faults at the node level is important in systems that rely

on duplicated nodes for fault tolerance, as it allows the system to survive transient faults also when one of the nodes have failed permanently. It also improves the robustness of the system when both nodes are affected by correlated or near coincident transient faults. We have implemented a real-time kernel that supports light-weight NLFT through time redundant execution of tasks and the use of software implemented error detection. The effectiveness of light-weight NLFT is evaluated both analytically and by extensive fault injection experiments.

The thesis also deals with the cost of fault tolerance from a validation perspective. Fault injection based validation of error handling mechanisms is a time consuming and costly activity. We present a fault injection tool that can be easily extended and adapted to different target systems making fault injection less time-consuming. We also propose an analytical technique for investigating how error coverage varies for different input sequences to a system. The analysis helps us identify interesting activation patterns, e.g., those that give extremely low, or high, error coverage

B.2.2 Andersson, Björn, ARTES project no. 29, CTH.

PhD, September 29, 2003, Static-priority scheduling on multiprocessors.

This thesis deals with the problem of scheduling a set of tasks to meet deadlines on a computer with multiple processors. Static-priority scheduling is considered, that is, a task is assigned a priority number that never changes and at every moment the highest priority tasks that request to be executed are selected for execution.

The performance metric used is the capacity that tasks can request without missing a deadline. It is shown that every static-priority algorithm can miss deadlines although close to 50% of the capacity is requested. The new algorithms in this thesis have the following performance. In periodic scheduling, the capacity that can be requested without missing a deadline is: 33% for migrative scheduling and 50% for non-migrative scheduling. In aperiodic scheduling, many performance metrics have been used in previous research. With the aperiodic model used in this thesis, the new algorithms in this thesis have the following performance. The capacity that can be requested without missing a deadline is: 50% for migrative scheduling and 31% for non-migrative scheduling.

B.2.3 Assarsson, Ulf, ARTES project no. 29, CTH.

PhD, October 10, 2003, A Real-Time Soft Shadow Volume Algorithm.

Rendering of shadows is a very important ingredient in three-dimensional graphics since they increase the level of realism and provide cues to spatial relationships. Area or volumetric light sources give rise to so called soft shadows, i.e., there is a smooth transition from no shadow to full shadow. For hard shadows, which are generated by point light sources, the transition is abrupt. Since all real light sources occupy an area or volume, soft shadows are more realistic than hard shadows. Fast rendering of soft shadows, preferably in real time, has been a subject for research for decades, but so far this has mostly been an unsolved problem.

Therefore, this thesis, which is based on five papers, focuses on how to achieve real-time rendering of soft shadows. The first four papers constitute the foundation and evolution of a new algorithm, called the soft shadow volume algorithm, and the fifth paper provides an essential proof for correctness and generality of this and some previous shadow algorithms.

The algorithm augments and extends the well-known shadow volume algorithm for hard shadows. Two passes are used, where the first pass consist of the classic shadow volume algorithm to generate the hard shadows (umbra). The second pass compensates to provide the softness (penumbra). This is done by generating penumbra wedges and rasterizing them using a custom pixel shader that for each rasterized pixel projects the hard shadow quadrilaterals onto the light source and computes the covered area.

A result of the thesis is an algorithm capable of real-time soft shadows that utilizes programmable graphics hardware. The algorithm produce high-quality shadows for area light sources and volumetric light sources. It also handles textured light sources, which currently is a very rare capability among real-time soft shadow algorithms. Even video textures are allowed as light sources.

B.2.4 Bergenhem, Carl, ARTES project no. 15, HH.

Licentiate, December 19, 2002, Protocols with Heterogeneous Real-Time Services for High-Performance Embedded Networks

Network protocols for applications that demand high performance and heterogeneous real-time services are presented. These protocols control the medium access to the network and offer additional features to the user, both different user services for traffic and services for parallel and distributed real-time processing. The network architecture assumed is a unidirectional pipelined optical ring.

Radar Signal Processing (RSP) is a typical application area. Such a system contains many computation nodes that are interconnected in order to co-operate and thereby achieve higher performance. The computing performance of the whole system is greatly affected by the choice of network. Computing nodes in a parallel computer for RSP should be tightly coupled, i.e., communications cost (e.g. latency) between nodes should be small, so that the whole system can be perceived as a single unit. This is possible if a suitable network with an efficient protocol is used.

There is an industrial need for new high-performance networks with support for the, often heterogeneous, real-time requirements found in (often embedded) applications such as RSP and other areas such as multimedia. The traffic this kind of network can be classified according to its requirements. The proposed protocols partition the traffic into three classes with distinctly different qualities. These classes are traffic with hard real-time demands, such as mission critical commands, traffic with soft real-time demands, such as process data (a deadline miss here only leads to decreased performance) and, finally, traffic with no real-time constraints at all. The contributions of the present thesis are protocols that integrate heterogeneous real-time services for the three traffic classes.

The performance of the proposed protocols is evaluated through simulations and analysis. It is shown that the protocol is an efficient choice for RSP systems. A brief survey of related technologies is included in the thesis. These are studied from the perspectives of application, architecture and user service.

B.2.5 Bilstrup, Urban, ARTES project no. 16, HH.

Licentiate, June 7, 2005, Design Space Exploration of Wireless Multihop Networks.

This thesis explores the feasible design space of wireless multihop networks and identifies fundamental design parameters. In the process of exploring it is important to ignore all details and instead take a holistic view. This means that all protocol details are overseen, all details of radio wave propagation models are overseen and the system is modelled strictly on an architectural level. From a theoretical information perspective, there is a limit to the capacity that a certain bandwidth and a certain signal-to-noise ratio at the receiver can provide. This limit is approximated as a volume in the time-frequency-space domain. A single transmission is represented as an occupied volume in this domain. A wireless multihop network covers a spatial area, and the question is how multiple numbers of transmission volumes can be fit into a given limited spatial area. This volume fitting should be done in order to maximize

the overall performance or to trade available resources to favour a specific characteristic in the wireless multihop network. The volume model is used for the design space exploration of a wireless multihop network. It is argued that the fault tolerance and the energy gain achieved in a multihop topology are its strength as compared to a single-hop architecture. It is further shown that the energy gain is achieved at the expense of delay and a greater end-to-end error probability. This indicates that these parameters must be very carefully balanced in order to gain in the global overall performance perspective. It can further be concluded that the overall spatial capacity is increased as a result of the spatial channel reuse in a multihop topology. On the other hand, it is also shown that the multihop topology introduces a rather stringent geometrical capacity limitation when the number of nodes of a wireless multihop network is increased. The dynamics (e.g. node mobility, changing radio channels etc.) of a large scale wireless multihop network is also a limiting factor. The nodes' mobility creates a knowledge horizon beyond which very little can be known about the present network topology.

B.2.6 Broberg, Magnus, ARTES project no. 30, BTH.

PhD, May 3, 2002, Performance Prediction and Improvement Techniques for Parallel Programs in Multiprocessors.

The performance of a computer system is important. One way of improving performance is to use multiprocessors with several processors that can work in parallel. Where multiprocessors are used, the programs must also be parallel in order to achieve high performance. However, it is not always easy to write parallel programs for multiprocessors; program developers need support in this area. Such support includes, for example, information regarding how well the parallel program scales-up when the number of processors increases and identification of performance bottlenecks; ideally, the result should be presented graphically. Bottlenecks arise both as a result of parallelization as well as traditional (sequential) code. Further, the developer may need to predict performance on other systems than the one used for development, since the development environment often is the (uni-processor) workstation on the developer's desk. One way of increasing the performance may be to bind threads on processors statically. Finding the optimal allocation is NP-hard and it is necessary to resort to heuristic algorithms. When heuristic algorithms are used we do not know how near/far we are from the optimal allocation. Finding a bound for the program's completion time shows what should be achievable using a heuristic algorithm.

In this thesis, I present techniques how to simulate a multiprocessor execution of a parallel program based on a monitored execution on a uni-processor. The result of the (simulated) multiprocessor execution is

graphically presented in order to give feedback to the developer. The techniques can be used for heuristic algorithms to find an allocation of threads to processors. Further, I show an algorithm that identifies the critical path of the parallel program on a multiprocessor, thereby identifying the segments that are worthwhile optimizing. I also show how to calculate a tight bound on the minimal completion time for the optimal allocation of threads to processors. Finally, I discuss the implications of the choice of simulation model. The techniques and algorithms described have been manifested in a prototype tool which I have used to perform empirical studies. The tool has been validated using a real multiprocessor.

B.2.7 Cervin, Anton, ARTES project no. 3, LU.

PhD, April 25, 2003, Integrated Control and Real-Time Scheduling.

The topic of the thesis is codesign of flexible real-time control systems. Integrating control theory and real-time scheduling theory, it is possible to achieve higher resource utilization and better control performance. The integration requires new tools for analysis, design, and implementation.

The problem of scheduling the individual parts of a control algorithm is studied. It is shown how subtask scheduling can reduce the input-output latency in a set of control tasks. Deadline assignment under different scheduling policies is considered.

A feedback scheduling architecture for control tasks is introduced. The scheduler uses feedback from execution-time measurements and feedforward from workload changes to adjust the sampling periods of a set of control tasks so that the combined performance of the controllers is optimized.

The Control Server, a novel computational model for real-time control tasks, is presented. The model combines time-triggered I/O with dynamic, reservation-based task scheduling. The model provides short input-output latencies and minimal jitter for the controllers. It also allows control tasks to be treated as scalable real-time components with predictable performance.

Two MATLAB-based toolboxes for analysis and simulation of real-time control systems have been developed. The Jitterbug toolbox evaluates a quadratic cost function for a linear control system with timing variations. The tool makes it possible to investigate the impact of delay, jitter, lost samples, etc., on control performance. The TrueTime toolbox facilitates detailed cosimulation of distributed real-time control systems. The scheduling and execution of control tasks is simulated in parallel with the network communication and the continuous process dynamics.

B.2.8 Chen, De Jiu, ARTES project no. 5, KTH.

PhD, June 9, 2004, Systems Modeling and Modularity Assessment for Embedded Computer Control Applications.

The development of embedded computer control systems (ECS) requires a synergetic integration of heterogeneous technologies and multiple engineering disciplines. With increasing amount of functionalities and expectations for high product qualities, short time-to-market, and low cost, the success of complexity control and built-in flexibility turn out to be one of the major competitive edges for many ECS products. For this reason, modeling and modularity assessment constitute two critical subjects of ECS engineering.

In the development of ECS, model-based design is currently being exploited in most of the sub-systems engineering activities. However, the lack of support for formalization and systematization associated with the overall systems modeling leads to problems in comprehension, cross-domain communication, and integration of technologies and engineering activities. In particular, design changes and exploitation of "components" are often risky due to the inability to characterize components' properties and their system-wide contexts. Furthermore, the lack of engineering theories for modularity assessment in the context of ECS makes it difficult to identify parameters of concern and to perform early system optimization.

This thesis aims to provide a more complete basis for the engineering of ECS in the areas of systems modeling and modularization. It provides solution domain models for embedded computer control systems and the software subsystems. These meta-models describe the key system aspects, design levels, components, component properties and relationships with ECS specific semantics. By constituting the common basis for abstracting and relating different concerns, these models will also help to provide better support for obtaining holistic system views and for incorporating useful technologies from other engineering and research communities such as to improve the process and to perform system optimization. Further, a modeling framework is derived, aiming to provide a perspective on the modeling aspect of ECS development and to codify important modeling concepts and patterns. In order to extend the scope of engineering analysis to cover flexibility related attributes and multi-attribute tradeoffs, this thesis also provides a metrics system for quantifying component dependencies that are inherent in the functional solutions. Such dependencies are considered as the key factors affecting complexity control, concurrent engineering, and flexibility. The metrics system targets early system-level design and takes into account several domain specific features such as replication and timing accuracy.

B.2.9 David, Alexandre, ARTES project no. 12, UU.

PhD, November 26, 2003, Hierarchical Modeling and Analysis of Real Time Systems.

UPPAAL is a tool for model-checking real-time systems developed jointly by Uppsala University and Aalborg University. It has been applied successfully in case studies ranging from communication protocols to multimedia applications. The tool is designed to verify systems that can be modeled as networks of timed automata. But it lacks support for systems with hierarchical structures, which makes the construction of large models difficult. In this thesis we improve the efficiency of UPPAAL with new data structures and extend its modeling language and its engine to support hierarchical constructs.

To investigate the limits of UPPAAL, we model and analyze an industrial fieldbus communication protocol. To our knowledge, this case study is the largest application UPPAAL has been confronted to and we managed to verify the models. However, the hierarchical structure of the protocol is encoded as a network of automata without hierarchy, which artificially complicates the model. It turns out that we need to improve performance and enrich the modeling language.

To attack the performance bottlenecks, we unify the two central structures of the UPPAAL engine, the passed and waiting lists, and improve memory management to take advantage of data sharing between states. We present experimental results that demonstrate improvements by a factor 2 in time consumption and a factor 5 in memory consumption.

We enhance the modeling capabilities of UPPAAL by extending its input language with hierarchical constructs to structure the models. We have developed a verification engine that supports modeling of hierarchical systems without penalty in performance. To further benefit from the structures of models, we present an approximation technique that utilizes hierarchy in verification.

Finally, we propose a new architecture to integrate the different verification techniques into a common framework. It is designed as a pipeline built with components that are changed to fit particular experimental configurations and to add new features. The new engine of UPPAAL is based on this architecture. We believe that the architecture is applicable to other verification tools.

B.2.10 Dobrin, Radu, ARTES project no. 24, MDH.

PhD, September 27, 2005, Combining Off-line Schedule Construction and Fixed Priority Scheduling in Real-Time Computer Systems.

Off-line scheduling and fixed priority scheduling (FPS) are often considered as complementing and incompatible paradigms. A number of

industrial applications demand temporal properties (predictability, jitter constraints, end-to-end deadlines, etc.) that are typically achieved by using off-line scheduling. The rigid off-line scheduling schemes used, however, do not provide for flexibility. On the other hand, FPS has been widely studied and used in a number of industrial applications, mostly due to its simple run-time scheduling and small overhead. It provides more flexibility, but is limited with respect to predictability, as actual start and completion times of executions depend on run-time events.

In this thesis we first show how off-line scheduling and FPS can be combined to get the advantages of both – the capability to cope with complex timing constraints while providing run-time flexibility. The proposed approach assume that a schedule for a set of tasks with complex constraints has been constructed off-line. We present methods to analyze the off-line schedule and derive FPS attributes such that the runtime FPS execution matches the off-line schedule. In some cases, i.e., when the off-line schedule can not be expressed directly by FPS, we split tasks into instances (artifacts) to obtain a new task set with consistent task attributes. Our method keeps the number of newly generated artifact tasks minimal.

At the same time, we investigate the behavior of the existing FPS servers to handle non-periodic events, while the complex constraints imposed on the periodic tasks are still fulfilled. In particular, we provide a solution to server parameter assignment to provide non-periodic events a good response time, while still fulfilling the original complex constraints on the periodic tasks.

Secondly, we apply the proposed method to schedule messages with complex constraints on Controller Area Network (CAN). We analyze an off-line schedule constructed to solve complex constraints for messages, e.g., precedence, jitter or end-to-end deadlines, and we derive attributes, i.e., message identifiers, required by CAN's native protocol. At run time, the messages are transmitted and received within time intervals such that the original constraints are fulfilled.

Finally, we propose a method to reduce the number of preemptions in legacy FPS systems consisting of tasks with priorities, periods and offsets. Unlike other approaches, our algorithm does not require modification of the basic FPS mechanism. Our method analyzes off-line a set of periodic tasks scheduled by FPS, detects the maximum number of preemptions that can occur at run-time, and reassigns task attributes such that the tasks are schedulable by the same scheduling mechanism while achieving a lower number of preemptions. In some cases, there is a cost to pay for achieving a lower number of preemptions, e.g., an increased number of tasks and/or reduced task execution flexibility. Our method provides for the ability to trade-off between the number of preemptions and the cost to pay.

B.2.11 d'Orso, Julien, ARTES project no. 13, UU.

PhD, November 25, 2003, New Directions in Symbolic Model Checking.

In today's computer engineering, requirements for generally high reliability have pushed the notion of testing to its limits. Many disciplines are moving, or have already moved, to more formal methods to ensure correctness. This is done by comparing the behavior of the system as it is implemented against a set of requirements. The ultimate goal is to create methods and tools that are able to perform this kind of verification automatically: this is called Model Checking.

Although the notion of model checking has existed for two decades, adoption by the industry has been hampered by its poor applicability to complex systems. During the 90's, researchers have introduced an approach to cope with large (even infinite) state spaces: Symbolic Model Checking. The key notion is to represent large (possibly infinite) sets of states by a small formula (as opposed to enumerating all members). In this thesis, we investigate applying symbolic methods to different types of systems:

Parameterized systems. We work within the framework of Regular Model Checking. In regular model checking, we represent a global state as a word over a finite alphabet. A transition relation is represented by a regular length-preserving transducer. An important operation is the so-called transitive closure, which characterizes composing a transition relation with itself an arbitrary number of times. Since completeness cannot be achieved, we propose methods of computing closures that work as often as possible.

Games on infinite structures. Infinite-state systems for which the transition relation is monotonic with respect to a well quasi-ordering on states can be analyzed. We lift the framework of well quasi-ordered domains toward games. We show that monotonic games are in general undecidable. We identify a subclass of monotonic games: downward-closed games. We propose an algorithm to analyze such games with a winning condition expressed as a safety property.

Probabilistic systems. We present a framework for the quantitative analysis of probabilistic systems with an infinite state-space: given an initial state s_{init} , a set F of final states, and a rational $\theta > 0$, compute a rational ρ such that the probability of reaching F from s_{init} is between ρ and $\rho + \theta$. We present a generic algorithm and sufficient conditions for termination.

B.2.12 Ekelin, Cecilia, ARTES project no. 10, CTH.

PhD, May, 28, 2004, An Optimization Framework for Scheduling of Embedded Real-Time Systems.

Embedded real-time systems - appearing in products such as cars and mobile phones - are nowadays common in our everyday lives. Despite this fact, the design process of such systems is still cumbersome due to the large variety of design constraints that must be considered to ensure a safe operation of the system. In particular, present scheduling techniques - that analyze the timing behavior of the system - typically assume a too limited model to truly represent the system. In addition, to make the system cost-effective its design should be optimized regarding performance measures such as resource utilization, energy consumption and robustness. Unfortunately, optimization in general is very time-consuming process, often without guarantee that the best solution will be found.

This thesis addresses these problems by proposing a scheduling framework that not only enables arbitrary design constraints to be modelled but also allows for design optimization. The framework is based on constraint programming, and this thesis presents how the problem of scheduling embedded real-time systems can be modeled and solved using this technique. In addition, a number of novel techniques for reducing the runtime of the optimization algorithm are presented. This includes the identification and exclusion of symmetries in the solution space as well as fast and tight estimates of how good a solution may get. Finally, this thesis contains a performance comparison between the proposed framework and other state-of-the-art scheduling algorithms. The evaluation shows that both the quality of the solutions and the optimization time is improved over previous approaches - in many cases the order of the solution time is reduced from minutes to seconds.

B.2.13 Ekman, Magnus, ARTES project no. 34, CTH.

PhD, December 20, 2004, Strategies to Reduce Energy and Resources in Chip Multiprocessor Systems.

A new architectural style known as chip multiprocessor (CMP) has recently emerged, where two or more processor cores are manufactured on the same die. This architectural style comes with many promises such as high performance for applications with much thread-level parallelism (TLP) and shorter design times due to its modularized design. Nevertheless, a new architectural paradigm also introduces new design challenges.

This thesis addresses the technical problem of how to design more efficient CMP systems in terms of energy and memory utilization. It

contributes with design strategies that fall into three categories, consisting of design principles to reduce energy dissipation and main memory resources without reducing performance, design recommendations to balance the exploited instruction level parallelism (ILP) and TLP in a CMP, and a methodology to reduce simulation time when evaluating future designs.

Two of the proposed design principles can together reduce the energy dissipation in the L1-caches and translation-lookaside buffers by 30%. Secondly, it is shown that it is possible to tolerate ten times longer access latency to 70% of the main memory, which can be exploited by compression techniques to reduce the amount of memory by 30%. A novel compression scheme applied to the entire main memory is proposed and evaluated and is shown to reduce the needed memory resources by 30%. These reductions do not have any significant negative effect on performance.

Further, the trade-off between TLP and ILP is studied for applications with an abundance of TLP under a fixed area constraint. Four different design points ranging from 16 single-issue cores to two eight-issue cores are evaluated. By choosing the design point with four cores with an issue width of four it is possible to achieve close to optimal performance while still enabling single-threaded applications to run well.

Finally, a sampling technique for single-processor simulation is applied to multiprocessors. For one class of applications the technique is more efficient for multiprocessors by reducing the number of simulation points linearly with the number of processors. Another statistical technique is then proposed both for single and multiprocessor systems and reduces the required number of simulation points by one order of magnitude.

B.2.14 El-Khoury, Jad, ARTES project no. 9, KTH.

PhD, March 3, 2006, A Model Management and Integration Platform for Mechatronics Product Development.

Mechatronics development requires the close collaboration of various specialist teams and engineering disciplines. Developers from the different disciplines use domain-specific tools to specify and analyse the system of interest. This leads to different views of the system, each targeting a specific audience, using that audience's familiar language, and concentrating on that audience's concerns. Successful system development requires that the views of all developers produced by the different tools are well integrated into a whole, reducing any risks of inconsistencies and conflicts in the design information specified.

This thesis discusses techniques of managing and integrating the views from various disciplines, taking better advantage of multidisci-

plinary, model-based, development. A Model Data Management (MDM) platform that generically manages models from the various domain-specific tools used in development is presented. The platform is viewed as a unification of the management functionalities typically provided by the discipline-specific PDM and SCM systems. The unification is achieved by unifying the kind of objects it manages - models. View integration is considered as an integral functionality of this platform.

In demonstrating the platform's feasibility, a generic version management functionality of models is implemented. In addition, model integration is investigated for the allocation of system functions onto the implementing hardware architecture. The proposed approach promotes the independent development of the views, allowing developers from each discipline to work concurrently, yet ensuring the completeness, correctness and analysis of any inter-view design decisions made.

The prototype MDM platform builds on existing technologies from each of the mechanical and software disciplines. The proposed MDM system is built based on a configurable PDM system, given its maturity and ability to manage model contents appropriately. At the same time, the version control functionality borrows ideas from the fine-grained version control algorithms in the software discipline.

The platform is argued to be feasible given the move towards model-based development in software engineering, bringing the discipline's needs closer to those of the hardware discipline. This leads the way for an easier and more effective integrated management platform satisfying the needs of both disciplines using a common set of mechanisms

B.2.15 Gestegård Robertz, Sven, ARTES proj. no. 3, LU.

Licentiate, May 12, 2003, Flexible automatic memory management for real-time and embedded systems.

The advent of safe languages like Java on the real-time systems scene motivates further research on efficient strategies for non-intrusive garbage collection and especially GC scheduling. This thesis presents new approaches to flexible and robust memory management from an engineering perspective and is a step towards write once — run anywhere with hard real-time performance.

The traditional approach to incremental GC scheduling, to perform garbage collection work in proportion to the amount of allocated memory, has drawbacks and in order to remedy this, a scheduling strategy, time-triggered GC, based on assigning a deadline for when the GC must complete its current cycle is proposed. It is shown that this strategy can give real-time performance that is equal to, or better than, that of an allocation-triggered GC. It is also shown that by using a deadline-based

scheduler, the GC scheduling and, consequently, the real-time performance, is independent of a complex and error-prone work metric.

Time-triggered GC also allows a more high-level view on GC scheduling as the GC cycle level rather than on each individual increment is considered. This makes it possible to schedule GC as any other thread. It also makes the time-triggered strategy well suited for auto-tuning and it is shown how an adaptive GC scheduler can be implemented.

A novel approach of applying priorities to memory allocation is introduced and it is shown how this can be used to enhance the robustness of real-time applications. The proposed mechanisms can also be used to increase performance of systems with automatic memory management by limiting the amount of garbage collection work.

The ideas brought forward in this thesis have been implemented and validated in an experimental real-time Java environment.

B.2.16 Gruian, Flavius, ARTES project no. 19, LU.

PhD, December 17, 2002, Energy-Centric Scheduling for Real-Time Systems.

Energy consumption is today an important design issue for all kinds of digital systems, and essential for the battery operated ones. An important fraction of this energy is dissipated on the processors running the application software. To reduce this energy consumption, one may, for instance, lower the processor clock frequency and supply voltage. This, however, might lead to a performance degradation of the whole system. In real-time systems, the crucial issue is timing, which is directly dependent on the system speed. Real-time scheduling and energy efficiency are therefore tightly connected issues, being addressed together in this work.

Several scheduling approaches for low energy are described in the thesis, most targeting variable speed processor architectures. At task level, a novel speed scheduling algorithm for tasks with probabilistic execution pattern is introduced and compared to an already existing compile-time approach. For task graphs, a list-scheduling based algorithm with an energy-sensitive priority is proposed. For task sets, off-line methods for computing the task maximum required speeds are described, both for rate-monotonic and earliest deadline first scheduling. Also, a run-time speed optimization policy based on slack re-distribution is proposed for rate-monotonic scheduling. Next, an energy-efficient extension of the earliest deadline first priority assignment policy is proposed, aimed at tasks with probabilistic execution time. Finally, scheduling is examined in conjunction with assignment of tasks to processors, as parts of various low energy design flows. For some of the algorithms given in the thesis, energy measurements were carried out on a real hardware platform con-

taining a variable speed processor. The results confirm the validity of the initial assumptions and models used throughout the thesis. These experiments also show the efficiency of the newly introduced scheduling methods.

B.2.17 Hoang, Hoai, ARTES project no. 21, HH.

Licentiate, May 27, 2003, Switched Real-Time Ethernet for Industrial Applications.

The research reported in this thesis has been focused on developing and analyzing how to support real-time traffic over a switched Ethernet network without any hardware or protocol modifications. The work has resulted in a proposed systems model, supporting both real-time and non real-time traffic. Currently this model is intended for a one-switch network, with no shared media. All added traffic handling to support real-time communication is positioned in a thin layer (RT layer) added between the Ethernet layer and the TCP/IP suite. This assures adaptation to the surrounding protocol standards. The RT layer manages traffic on the basis of virtual connections, denoted as RT channels, as well as packet level scheduling. RT channels are created between end-nodes prior to any occurrence of real-time traffic. Asymmetric deadline partitioning between the links of the RT channels is also proposed, in order to increase the number of possible RT channels.

B.2.18 Häggander, Daniel, ARTES project no. 30, BTH.

PhD, September 14, 2001, Software Design Conflicts, Maintainability versus Performance and Availability.

A major goal in software engineering is to reduce the cost of maintaining software systems. Finding design methods which make software more easily maintainable has thus been one of the most prioritized challenges during the past decade. While mainstream software design has concentrated on maintainability, other software disciplines e.g. high-performance computing and high-availability systems, have developed other design methods which primarily support the quality attributes that are more important in their areas. More recently, demands have been made for high performance and high availability in typical mainstream software. At the same time, traditional high-performance and high-availability systems tend to incorporate more advanced business functionality, i.e. different software disciplines have started to converge. The situation is not unproblematic since the software design methods developed for achieving performance and availability may have been developed with a limited influence from maintainability, and vice versa. It is thus important to identify and analyze emerging design conflicts.

In this thesis I have studied conflicts between maintainability design methods on the one hand, and performance and availability methods and techniques on the other. I present the results of four case-studies involving four different applications. It is a characteristic of these applications that half of the system can be regarded as a telecommunications system and the other as a typical main-stream system, i.e. all systems make high demands on performance and availability but also very high demands on high maintainability. In studying these applications, I have identified two major conflicts: granularity in dynamic memory usage and source code size. My results show that these two conflicts can cause problems of such amplitude that some applications become unusable. I found that conflicts in certain situations are inherent; in other cases they can be avoided - or at least reduced - by adjusting the design methods used. I have also shown that conflicts may quite simply be a matter of misconceptions. Ten guidelines have been combined into a simple process with the aim of helping software designers to avoid and reduce conflicts. A method which automatically reduces the dynamic memory conflict in object-oriented applications written in C++ has been developed, implemented and evaluated. Finally, I have defined optimal recovery schemes for high availability clusters.

B.2.19 Isovich, Damir, ARTES project no. 14, MDH.

PhD, November 9, 2004, Flexible Scheduling for Media Processing in Resource Constrained Real-Time Systems.

The MPEG-2 standard for video coding is predominant in consumer electronics for DVD players, digital satellite receivers, and TVs today. MPEG-2 processing puts high demands on audio/video quality, which is achieved by continuous and synchronized playout without interrupts. At the same time, there are restrictions on the storage media, e.g., limited size of a DVD disc, communication media, e.g., limited bandwidth of the Internet, display devices, e.g., the processing power, memory and battery life of pocket PCs or video mobile phones, and finally the users, i.e., human's ability of perceiving motion. If the available resources are not sufficient to process a full-size MPEG-2 video, then video stream adaptation must take place. However, this should be done carefully, since in high quality devices, drops in perceived video quality are not tolerated by consumers.

We propose real-time methods for resource reservation of MPEG-2 video stream processing and introduce flexible scheduling mechanisms for video decoding. Our method is a mixed offline and online approach for scheduling of periodic, aperiodic and sporadic tasks, based on slot shifting. We use the offline part of slot shifting to eliminate all types of complex task constraints before the runtime of the system. Then,

we propose an online guarantee algorithm for dealing with dynamically arriving tasks. Aperiodic and sporadic tasks are incorporated into the offline schedule by making use of the unused resources and leeways in the schedule. Sporadic tasks are guaranteed offline for the worst-case arrival patterns and scheduled online, where an online algorithm keeps track of arrivals of instances of sporadic tasks to reduce pessimism about future sporadic arrivals and improve response times and acceptance of firm aperiodic tasks. At runtime, our mechanism ensures feasible execution of tasks with complex constraints in the presence of additional tasks or overloads.

We use the scheduling and resource reservation mechanism above to flexibly process MPEG-2 video streams. First, we present results from a study of realistic MPEG-2 video streams to analyze the validity of common assumptions for software decoding and identify a number of misconceptions. Then, we identify constraints imposed by frame buffer handling and discuss their implications on the decoding architecture and timing. Furthermore, we propose realistic timing constraints demanded by high quality MPEG-2 software video decoding. Based on these, we present a MPEG-2 video frame selection algorithm with focus on high video quality perceived by the users, which fully utilize limited resources. Given that not all frames in a stream can be processed, it selects those which will provide the best picture quality while matching the available resources, starting only such decoding, which is guaranteed to be completed. As a final result, we provide a real-time method for flexible scheduling of media processing in resource constrained system. Results from study based on realistic MPEG-2 video underline the effectiveness of our approach.

B.2.20 Karlsson, Martin, ARTES project no. 31, UU.

PhD, January 13, 2006, Memory System Design for Chip-Multiprocessors.

The continued decrease in transistor size and the increasing delay of wires relative to transistor switching speeds led to the development of chip multiprocessors (CMPs). The introduction of CMPs presents new challenges and trade-offs to computer architects. In particular, architects must now balance the allocation of chip resources to each processor against the number of processors on a chip. This thesis deals with some of the implications this new kind of processors have regarding the memory system and proposes several new designs based on the resource constraints of CMPs. In addition, it includes contributions on simulation technique and workload characterization, which is used to guide the design of new processors and systems.

The memory system is the key to performance in contemporary computer systems. This thesis targets multiple aspects of memory system performance. To conserve bandwidth, and thereby packaging costs, a fine-grained data fetching strategy is presented that exploits characteristics of runahead execution. Two cache organizations are proposed: The RASCAL cache organization, which target capacity misses through selective caching and the Elbow cache that targets conflict misses by extending a skewed cache with a relocation algorithm. Finally, to reduce complexity and cost when designing multi-chip systems, a new trap-based system architecture is described.

When designing a new processor or memory system, simulations are used to compare design alternatives. It is therefore very important to simulate workloads that accurately reflect the future use of the system. This thesis includes the first architectural characterization studies of Java-based middleware, which is a workload that is an important design consideration for the next generation of processors and servers.

B.2.21 Manolache, Sorin, ARTES project no. 4, LiU.

PhD, Dec 16, 2005, Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour.

Embedded systems have become indispensable in our life: household appliances, cars, airplanes. power plant control systems, medical equipment. telecommunication systems. space technology, they all contain digital computing systems with dedicated functionality. Most of them, if not all. are real-time systems, i.e. their responses to stimuli have timeliness constraints.

The timeliness requirement has to be met despite some unpredictable, stochastic behaviour of the system. In this thesis, we address two causes of such stochastic behaviour: the application and platform-dependent stochastic task execution times, and the platform-dependent occurrence of transient faults on network links in networks-on-chip.

We present three approaches to the analysis of the deadline miss ratio of applications with stochastic task execution times. Each of the three approaches fits best to a different context. The first approach is an exact one and is efficiently applicable to mono-processor systems. The second approach is an approximate one, which allows for designer-controlled trade-off between analysis accuracy and analysis speed. It is efficiently applicable to multiprocessor systems. The third approach is less accurate but sufficiently fast in order to be placed inside optimisation loops. Based on the last approach, we propose a heuristic for task mapping and priority assignment for deadline miss ratio minimisation.

Our contribution is manifold in the area of buffer and time constrained communication along unreliable on-chip links. First. we intro-

duce the concept of communication supports, an intelligent combination between spatially and temporally redundant communication. We provide a method for constructing a sufficiently varied pool of alternative communication supports for each message. Second, we propose a heuristic for exploring the space of communication support candidates such that the task response times are minimised. The resulting time slack can be exploited by means of voltage and/or frequency scaling for communication energy reduction. Third, we introduce an algorithm for the worst-case analysis of the buffer space demand of applications implemented on networks-on-chip. Last, we propose an algorithm for communication mapping and packet timing for buffer space demand minimisation.

All our contributions are supported by sets of experimental results obtained from both synthetic and real-world applications of industrial size.

B.2.22 Mäki-Turja, Jukka, ARTES project no. 2, MDH.

PhD, May 27, 2005, Engineering Strength Response-Time Analysis — A Timing Analysis Approach for the Development of Real-Time Systems.

When developing computer systems that are part of larger systems, as in control systems for cars, airplanes, or medical equipment, reliability and safety is of major concern. Developers of these systems want to keep the production and development costs to a minimum while maximizing customer benefit by increasing the functionality of the product.

Increasing the number of functions, along with the added complexity that it entails, places a demand on better development methods, models, and tools. Response-Time Analysis (RTA) can be a useful method for these systems by able to guarantee a system's temporal behavior. This thesis presents new techniques aimed at improving currently existing RTA methods. Specifically, these techniques lead to the following improvements:

- The precision in the calculated response times are significantly higher than with previous methods, with typically 15% shorter response times.
- The analysis, itself, can be made more 100 times faster than with previous implementations.

By combining these independent techniques of precise (tight) response times and fast analysis, as shown in this thesis, one can get the benefits of both in a single analysis method. High precision response-time estimates enable either increased functionality within a given hardware cost, or a lower cost for a given functionality. Faster RTA will increase the

usefulness of RTA by enabling the use of RTA in development tools for real-time systems with hundreds, or even thousands of tasks.

RTA can be particularly beneficial for safety critical applications that have even higher requirements on reliability and safety, and often undergo expensive and lengthy certification processes. We illustrate the possible advantages by applying RTA for tasks with offsets in a real industrial context. The benefits consist of simplifying the development process as well as enabling an efficient resource usage.

B.2.23 Nolte, Thomas, ARTES project no. 23, MDH.

Licentiate, May 12, 2003, Reducing Pessimism and Increasing Flexibility in the Controller Area Network.

The Controller Area Network (CAN) is a widely used real-time communication network for automotive and other embedded applications. As new applications continue to evolve, the complexity of distributed CAN based systems increase. However, CAN's maximum speed of 1 Mbps remains fixed, leading to performance bottlenecks. In order to make full use of this scarce bandwidth, methods for increasing the achievable utilisation are needed.

Traditionally, real-time scheduling theory has targeted hard real-time systems, which most of the time are safety critical. Since these systems (by definition) are not allowed to have any timing flaws, analysis techniques need to take all possible scenarios of execution combinations and execution times of the system into consideration. This will result in a system that is configured for the worst possible scenario. Whether this scenario is likely, or even possible, in the real system is not considered. Hence, the result may be an unnecessarily expensive system, with potentially overly provisioned resources.

In this thesis we address two issues. In the first part, we investigate how to loosen up pessimistic real-time analysis in a controlled way, thereby allowing the designer to make well-founded trade-offs between the level of real-time guarantee and the system cost. Specifically, we investigate and model the bit-stuffing mechanism in CAN in order to retrieve representative distributions of stuff-bits, which we then use in the response time analysis instead of the worst-case values. We evaluate the validity of these stuff-bit distributions in case studies, and we integrate this representation of message frame length with the classical CAN worst-case response time analysis.

For the second part of the thesis, to increase CAN flexibility, we propose a novel way of scheduling the CAN. By providing server based scheduling, bandwidth isolation between users is guaranteed. This allows for efficient handling of sporadic and aperiodic message streams. Server based scheduling also has the potential to allow higher network

utilisation compared to CAN native scheduling. The performance and properties of server based scheduling of CAN

B.2.24 Nyström, Dag, ARTES project no. 26, MDH.

PhD, October 26 2005, Data Management in Vehicle Control-Systems.

As the complexity of vehicle control-systems increases, the amount of information that these systems are intended to handle also increases. This thesis provides concepts relating to real-time database management systems to be used in such control-systems. By integrating a real-time database management system into a vehicle control-system, data management on a higher level of abstraction can be achieved.

Current database management concepts are not sufficient for use in vehicles, and new concepts are necessary. A case-study at Volvo Construction Equipment Components AB in Eskilstuna, Sweden presented in this thesis, together with a survey of existing database platforms confirms this. The thesis specifically addresses data access issues by introducing; (i) a data access method, denoted database pointers, which enables data in a real-time database management system to be accessed efficiently. Database pointers, which resemble regular pointers variables, permit individual data elements in the database to be directly pointed out, without risking a violation of the database integrity. (ii) two concurrency-control algorithms, denoted 2V-DBP and 2VDBP- SNAP which enable critical (hard real-time) and non-critical (soft real-time) data accesses to co-exist, without blocking of the hard real-time data accesses or risking unnecessary abortions of soft real-time data accesses. The thesis shows that 2V-DBP significantly outperforms a standard real-time concurrency control algorithm both with respect to lower response-times and minimized abortions. (iii) two concepts, denoted substitution and subscription queries that enable service- and diagnostics-tools to stimulate and monitor a control-system during run-time.

The concepts presented in this thesis form a basis on which a data management concept suitable for embedded real-time systems, such as vehicle control-systems, can be built.

B.2.25 Persson, Patrik, ARTES project no. 3, LU.

Licentiate, April 12, 2000, Predicting Time and Memory Demands of Object-Oriented Programs.

Embedded computer systems are subject to a multitude of requirements. These include real-time requirements, that is, such computers must respond to external events within limited time. Many systems, such as satellites and telephone switches, must also operate unattended for long periods of time. They must not fail due to defective software.

Modern object-oriented programming languages, particularly Java, offer type safety, automatic memory management (garbage collection), dynamic loading of code, and object-oriented abstraction mechanisms. All these features, designed to increase software quality and flexibility, are highly desirable in embedded systems. Yet object-oriented languages are often avoided in such applications. One reason for this is that previous techniques for worst-case execution time (WCET) predictions are unsuitable for object-oriented languages. WCET predictions are necessary to guarantee fulfilment of real-time requirements.

We present techniques for predicting the WCET of programs in object-oriented languages. We also show how to predict the amount of memory required by an object-oriented program; such information is required for safe scheduling of real-time garbage collection. The techniques are mainly automatic (assisted by manual annotations) and benefit from integration with a compiler. They are being implemented in an interactive development environment for a subset of the Java programming language.

The presented techniques make object-oriented programming languages with garbage collection more predictable and thus more appropriate for hard real-time systems. The declarative implementation technique (reference attributed grammars) facilitates a clear and concise implementation suitable for our interactive environment. This interactivity allows timing problems, requiring revision of design or requirements, to be detected early.

B.2.26 Pettersson, Anders, ARTES project no. 7, MDH.

Licentiate, October 31, 2003, Analysis of Execution Behavior for Testing of Multi-Tasking Real-Time Systems.

An important issue in software testing is the ability to observe the execution of the software; this is especially true for real-time systems (RTS). RTS are difficult to observe, and the ability to test RTS is inherently low. Embedded RTS have few interfaces for observation and the execution of multi-tasking RTS is usually non-deterministic. As a consequence, testing of RTS cannot be exercised with existing tools for sequential programs. New tools and methods are necessary that enable observation of the system despite few interfaces while at the same time address the non-determinism issue.

The contribution in this thesis is three-folded: (1) we present a tool suite that allows deterministic testing of multi-tasking RTS, in which synchronization of tasks is resolved off-line or on-line. (2) We show by building a test bed how to use the tool suite. (3) We present the design and functionality of Asterix the Real-Time Kernel.

In (1) we propose an analysis tool that derives all possible system level control-flow paths of multi-tasking RTS in which synchronization between communicating tasks are resolved on-line by using the Priority Ceiling Emulation Protocol (PCEP; also know as the Immediate Inheritance Protocol). The analysis tool is an extension of an existing tool in which synchronization were resolved off-line by using release time offsets or priorities to separate the tasks in time.

Based on the number of derived control-flow paths test coverage criteria are defined, and estimations of test effort can be done early in the development of a system. In (2) we show how the defined test coverage criteria relate to the number of traversed control-flow paths during test execution. We also show how the estimation of tasks' execution times affects the analysis. The analysis tool is applied on multi-tasking RTS in which the tasks are synchronized off-line. The real-time applications are then exercised on the test bed using Asterix as the operating system.

In (3) we present a small-sized real-time kernel named Asterix that has support for software based instrumentation of kernel events as well as application usage of system calls. The major problem of software instrumentation is the change in execution behavior that occurs when a RTS is executed with or without the probes. In Asterix we avoid this probe-effect by leaving the probes in the kernel during normal operation.

Also, we present a literature survey covering the state-of-the-art in the field real-time systems testing.

B.2.27 Pop, Paul, ARTES project no. 1, LiU.

PhD, June 16, 2003, Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems.

Embedded computer systems are now everywhere: from alarm clocks to PDAs, from mobile phones to cars, almost all the devices we use are controlled by embedded computer systems. An important class of embedded computer systems is that of real-time systems, which have to fulfill strict timing requirements. As real-time systems become more complex, they are often implemented using distributed heterogeneous architectures.

The main objective of this thesis is to develop analysis and synthesis methods for communication-intensive heterogeneous hard real-time systems. The systems are heterogeneous not only in terms of platforms and communication protocols, but also in terms of scheduling policies. Regarding this last aspect, in this thesis we consider time-driven systems, event-driven systems, and a combination of both, called multi-cluster systems. The analysis takes into account the heterogeneous interconnected nature of the architecture, and is based on an application model that captures both the dataflow and the flow of control. The proposed

synthesis techniques derive optimized implementations of the system that fulfill the design constraints. An important part of the system implementation is the synthesis of the communication infrastructure, which has a significant impact on the overall system performance and cost.

To reduce the time-to-market of products, the design of real-time systems seldom starts from scratch. Typically, designers start from an already existing system, running certain applications, and the design problem is to implement new functionality on top of this system. Hence, in addition to the analysis and synthesis methods proposed, we have also considered mapping and scheduling within such an incremental design process.

The analysis and synthesis techniques proposed have been thoroughly evaluated using a solid experimental platform. Besides the evaluations, performed using a large number of generated example applications, we have also validated our approaches using a realistic case study consisting of a vehicle cruise controller.

B.2.28 Redell, Ola, ARTES project no. 22, KTH.

PhD, May 8, 2003, Response Time Analysis for Implementation of Distributed Control Systems.

Methods for performing response time analysis of real-time systems are important, not only for their use in traditional schedulability testing, but also for deriving bounds on output timing variations in control applications. Automatic control systems are inherently sensitive to variations in periodicity and end-to-end delays. Therefore, real-time performance needs to be considered during control design. For this purpose, any real-time analysis of a potential control implementation should produce results that can easily be used to examine how the implementation affects control performance. To find the maximum response time variation for a task, bounds on both minimum and maximum response times are needed. A tight bound on this maximum variation is useful in the analysis of control performance and can also be used to improve the results of some iterative response time analysis methods. In this thesis, three methods for response time analysis are developed.

While earlier research has focused on bounding maximum response times, one of the analysis methods in this thesis allows a computation of the minimum response times of independent fixed priority scheduled tasks. The analysis finds the largest lower bound of response times for such tasks, which leads to a tighter bound on the response time variations. A second analysis method allows exact computation of maximum response times for tasks whose arrival times are related by offsets. The method is a complement to schedule simulation based analysis, which it outperforms for systems with tasks that may experience release jitter.

A common design principle for distributed real-time systems is to let the completion of one task trigger the start of one or more successors. A third method supporting the analysis of tasks in such systems is described. The method extends and improves earlier methods as it allows a generalized system model and also results in tighter bounds than the original methods. This method has been implemented as part of a toolset that enables an integrated approach to the design and analysis of control systems and their implementation as distributed real-time systems. As part of the thesis, models for describing distributed control systems have been developed. The toolset, which is based on these models, uses the derived response time bounds in a control system performance analysis based on simulation. The use of the toolset is exemplified in a small case study.

B.2.29 Scerri, Paul, ARTES project no. 25, LiU.

PhD, December 14, 2001, Designing Agents for Systems with Adjustable Autonomy.

Agents are an artificial intelligence technique of encapsulating a piece of pro-active, autonomous, intelligent software in a module that senses and acts in its environment. As the technology underlying sophisticated multi-agent systems improves, such systems are being deployed in ever more complex domains and are being given ever more responsibility for more critical tasks. However, multi-agent technology brings with it not only the potential for better, more efficient systems requiring less human involvement but also the potential to cause harm to the system's human users. One way of mitigating the potential harm an intelligent multi-agent system can do is via the use of adjustable autonomy. Adjustable autonomy is the idea of dynamically changing the autonomy of agents in a multi-agent system depending on the circumstances. Decision making control is transferred from agents to users when the potential for costly agent errors is large.

We believe that the design of the agents in a multi-agent system impacts the difficulty with which the system's adjustable autonomy mechanisms are implemented. Some features of an agent will make the implementation of adjustable autonomy easier, while others will make it more difficult. The central contribution of this thesis is a set of guidelines for the design of agents which, if followed, lead to agents which make adjustable autonomy straightforward to implement. In addition, the guidelines lead to agents from which it is straightforward to extract useful information and whose autonomy may be changed in a straightforward manner. The usefulness of the guidelines is shown in the design of the agents for two systems with adjustable autonomy. The first system is EASE, which is used for creating intelligent actors for interactive

simulation environments. The second system is the E-Elves which is a multiagent system streamlining the everyday coordination tasks of a human organisation. An evaluation of the two systems demonstrates that following the guidelines leads to agents that make effective adjustable autonomy mechanisms easier to implement.

B.2.30 Sundell, Håkan, ARTES project no. 8, CTH.

PhD, Nov 5, 2004, Efficient and Practical Non-Blocking DataStructures.

This thesis deals with how to design and implement efficient, practical and reliable concurrent data structures. The design method using mutual exclusion incurs serious drawbacks, whereas the alternative non-blocking techniques avoid those problems and also admit improved parallelism. However, designing non-blocking algorithms is a very complex task, and a majority of the algorithms in the literature are either inefficient, impractical or both.

We have studied how information available in real-time systems can improve and simplify non-blocking algorithms. We have designed new methods for recycling of buffers as well as time-stamps, and have applied them on known non-blocking algorithms for registers, snapshots and priority queues.

We have designed, to the best of our knowledge, the first practical lockfree algorithm of a skip list data structure. Using our skip list construction we have designed a lock-free algorithm of the priority queue abstract data type, as well as a lock-free algorithm of the dictionary abstract data type.

We have designed, to the best of our knowledge, the first practical lockfree algorithm of a doubly linked list data structure. The algorithm supports well-defined traversals in both directions including deleted nodes. Using our doubly linked list construction we have designed a lock-free algorithm of the deque abstract data type. For the lock-free algorithms presented in this thesis, we have given correctness proofs of the strong consistency property called linearizability and the non-blocking properties.

We have made implementations for actual systems of the algorithms presented in this thesis and a representative set of related non-blocking as well as lock based algorithms in the literature. We have built a framework that combines the implementations in the form of a software library that offers a unified and efficient interface in combination with a portable design. We have performed empirical performance studies of the data structures presented in this thesis in comparison with related alternative solutions. The experiments performed on an extensive set of multi-processor systems show significant improvements for non-blocking

alternatives in general, and for the implementations presented in this thesis in particular.

B.2.31 Tešanović, Aleksandra, ARTES proj. no. 26, LiU.

Licentiate, June 2, 2003, Towards Aspectual Component-Based Real-Time System Development.

Increasing complexity of real-time systems and demands for enabling their configurability and tailorability are strong motivations for applying new software engineering principles such as aspect-oriented and component-based software development. The integration of these two techniques into real-time systems development would enable: (i) efficient system configuration from the components in the component library based on the system requirements, (ii) easy tailoring of components and/or a system for a specific application by changing the behavior (code) of the component by aspect weaving, and (iii) enhanced flexibility of the real-time and embedded software through the notion of system configurability and components tailorability.

In this thesis we focus on applying aspect-oriented and component-based software development to real-time system development. We propose a novel concept of aspectual component-based real-time system development (ACCORD). ACCORD introduces the following into real-time system development: (i) a design method that assumes the decomposition of the real-time system into a set of components and a set of aspects, (ii) a real-time component model denoted RTCOM that supports aspect weaving while enforcing information hiding, (iii) a method and a tool for performing worst-case execution time analysis of different configurations of aspects and components, and (iv) a new approach to modeling of real-time policies as aspects.

We present a case study of the development of a configurable real-time database system, called COMET, using ACCORD principles. In the COMET example we show that applying ACCORD does have an impact on the real-time system development in providing efficient configuration of the real-time system. Thus, it could be a way for improved reusability and flexibility of real-time software, and modularization of crosscutting concerns.

In connection with development of ACCORD, we identify criteria that a design method for component-based real-time systems needs to address. The criteria include a well-defined component model for real-time systems, aspect separation, support for system configuration, and analysis of the composed real-time system. Using the identified set of criteria we provide an evaluation of ACCORD. In comparison with other approaches, ACCORD provides a distinct classification of crosscutting concerns in the real-time domain into different types of aspects, and

provides a real-time component model that supports weaving of aspects into the code of a component, as well as a tool for temporal analysis of the weaved system.

B.2.32 Thane, Henrik, ARTES project no. 7, MDH.

PhD, May 26, 2000, Monitoring, Testing and Debugging of Distributed Real-Time Systems.

Abstract Testing is an important part of any software development project, and can typically surpass more than half of the development cost. For safety-critical computer based systems, testing is even more important due to stringent reliability and safety requirements. However, most safety-critical computer based systems are real-time systems, and the majority of current testing and debugging techniques have been developed for sequential (non real-time) programs. These techniques are not directly applicable to real-time systems, since they disregard issues of timing and concurrency. This means that existing techniques for reproducible testing and debugging cannot be used. Reproducibility is essential for regression testing and cyclic debugging, where the same test cases are run repeatedly with the intention of verifying modified program code or to track down errors. The current trend of consumer and industrial applications goes from single micro-controllers to sets of distributed micro-controllers, which are even more challenging than handling real-time per-see, since multiple loci of observation and control additionally must be considered. In this thesis we try to remedy these problems by presenting an integrated approach to monitoring, testing, and debugging of distributed real-time systems.

For monitoring, we present a method for deterministic observations of single tasking, multi-tasking, and distributed real-time systems. This includes a description of what to observe, how to eliminate the disturbances caused by the actual act of observing, how to correlate observations, and how to reproduce them.

For debugging, we present a software-based method, which uses deterministic replay to achieve reproducible debugging of single tasking, multi-tasking, and distributed real-time systems. Program executions are deterministically reproduced off-line, using information concerning interrupts, task-switches, timing, data accesses, etc., recorded at run-time.

For testing, we introduce a method for deterministic testing of multitasking and distributed real-time systems. This method derives, given a set of tasks and a schedule, all execution orderings that can occur at run-time. Each such ordering is regarded as a sequential program, and by identifying which ordering is actually executed during testing, techniques for testing of sequential software can be applied.

For system development, we show the benefits of considering monitoring, debugging, and testing early in the design of real-time system software, and we give examples illustrating how to monitor, test, and debug distributed real-time systems.

B.2.33 Uhlemann, Elisabeth, ARTES project no. 16, HH.

PhD, October 22, 2004, Adaptive Concatenated Coding for Wireless Real-Time Communications.

The objective of this thesis is to improve the performance of real-time communication over a wireless channel, by means of specifically tailored channel coding. The deadline dependent coding (DDC) communication protocol presented here lets the timeliness and the reliability of the delivered information constitute quality of service (QoS) parameters requested by the application. The values of these QoS parameters are transformed into actions taken by the link layer protocol in terms of adaptive coding strategies.

Incremental redundancy hybrid automatic repeat request (IR-HARQ) schemes using rate compatible punctured codes are appealing since no repetition of previously transmitted bits is made. Typically, IR-HARQ schemes treat the packet lengths as fixed and maximize the throughput by optimizing the puncturing pattern, i.e. the order in which the coded bits are transmitted. In contrast, we define an IR strategy as the maximum number of allowed transmissions and the number of code bits to include in each transmission. An approach is then suggested to find the optimal IR strategy that maximizes the average code rate, i.e., the optimal partitioning of n k parity bits over at most M transmissions, assuming a given puncturing pattern. Concatenated coding used in IR-HARQ schemes provides a new array of possibilities for adaptability in terms of decoding complexity and communication time versus reliability. Hence, critical reliability and timing constraints can be readily evaluated as a function of available system resources. This in turn enables quantifiable QoS and thus negotiable QoS. Multiple concatenated single parity check codes are chosen as example codes due to their very low decoding complexity. Specific puncturing patterns for these component codes are obtained using union bounds based on uniform interleavers. The puncturing pattern that has the best performance in terms of frame error rate (FER) at a low signal-to-noise ratio (SNR) is chosen. Further, using extrinsic information transfer (EXIT) analysis, rate compatible puncturing ratios for the constituent component code are found. The puncturing ratios are chosen to minimize the SNR required for convergence.

The applications targeted in this thesis are not necessarily replacement of cables in existing wired systems. Instead the motivation lies in the new services that wireless real-time communication enables. Hence,

communication within and between cooperating embedded systems is typically the focus. The resulting IR-HARQ-DDC protocol presented here is an efficient and fault tolerant link layer protocol foundation using adaptive concatenated coding intended specifically for wireless real-time communications.

B.2.34 Wall, Anders, ARTES project no. 6, MDH.

PhD, September 26, 2003, Architectural Modeling and Analysis of Complex Real-Time Systems.

Most automation systems and other large industrial software systems have long lifetimes, and customers expect these systems to be supported as long as they are in operation. Furthermore, software components in these systems may be reused in different products, e.g. using a software product line approach. Hence, the lifetime of software in individual systems may be very long; several decades or even longer.

Software that is used for a long time will be exposed to frequent changes as the system evolve over time, e.g. due to adding new functionality, error corrections, or changing the hardware platform. The larger and older the system is, the harder it becomes to foresee the consequences of changes.

In this thesis we present three different techniques for managing the evolution of large and complex real-time systems. The techniques are based on analytical modeling, predicting different quality properties, e.g. temporal correctness, by analyzing a model of the software. The first technique is a component model with analytical interfaces (ReFlex) that allows us to predict different properties of a component assembly, the second is a probabilistic modeling language which is analyzed by simulations (ART-FW), and the third technique is an extension of classical timed automata with a notion of real-time tasks (TAT).

Ideally, the analytical models should evolve together with the software. However, since new features are often added and the implementation is often changed without updating the model, the model becomes obsolete and predictions based on the model are no longer valid. By applying the techniques proposed in this thesis, we can re-introduce analyzability; Using ReFlex we can update the analytical aspects while re-designing the system. Unless ReFlex has been used in the earlier design, this will require a costly redesign of the complete system, but consistency between the analytical model and the implementation will be ensured. Using ART-FW or TAT the implementation will be kept untouched by introducing a separate model. The drawback is that an extra effort is required to keep the model consistent with the implementation.

We have applied ART-FW in the re-engineering activity of a large industrial system. The results indicate that the approach is indeed applicable on real systems

B.2.35 Warg, Fredrik, ARTES project no. 33, CTH.

Licentiate, June 6, 2003, Module-Level Speculative Execution Techniques on Chip Multiprocessors.

The objective of this thesis is to improve the performance of real-time communication over a wireless channel, by means of specifically tailored channel coding. The deadline dependent coding (DDC) communication protocol presented here lets the timeliness and the reliability of the delivered information constitute quality of service (QoS) parameters requested by the application. The values of these QoS parameters are transformed into actions taken by the link layer protocol in terms of adaptive coding strategies.

Incremental redundancy hybrid automatic repeat request (IR-HARQ) schemes using rate compatible punctured codes are appealing since no repetition of previously transmitted bits is made. Typically, IR-HARQ schemes treat the packet lengths as fixed and maximize the throughput by optimizing the puncturing pattern, i.e. the order in which the coded bits are transmitted. In contrast, we define an IR strategy as the maximum number of allowed transmissions and the number of code bits to include in each transmission. An approach is then suggested to find the optimal IR strategy that maximizes the average code rate, i.e., the optimal partitioning of $n-k$ parity bits over at most M transmissions, assuming a given puncturing pattern. Concatenated coding used in IR-HARQ schemes provides a new array of possibilities for adaptability in terms of decoding complexity and communication time versus reliability. Hence, critical reliability and timing constraints can be readily evaluated as a function of available system resources. This in turn enables quantifiable QoS and thus negotiable QoS. Multiple concatenated single parity check codes are chosen as example codes due to their very low decoding complexity. Specific puncturing patterns for these component codes are obtained using union bounds based on uniform interleavers. The puncturing pattern that has the best performance in terms of frame error rate (FER) at a low signal-to-noise ratio (SNR) is chosen. Further, using extrinsic information transfer (EXIT) analysis, rate compatible puncturing ratios for the constituent component code are found. The puncturing ratios are chosen to minimize the SNR required for convergence.

The applications targeted in this thesis are not necessarily replacement of cables in existing wired systems. Instead the motivation lies in the new services that wireless real-time communication enables. Hence, communication within and between cooperating embedded systems is

typically the focus. The resulting IR-HARQ-DDC protocol presented here is an efficient and fault tolerant link layer protocol foundation using adaptive concatenated coding intended specifically for wireless real-time communications.

B.2.36 Voigt, Thiemo, ARTES project no. 28, UU.

PhD, May 27, 2002, Architectures for Service Differentiation in Overloaded Internet Servers.

Web servers become overloaded when one or several server resources such as network interface, CPU and disk become overutilized. Server overload leads to low server throughput and long response times experienced by the clients.

Traditional server design includes only marginal or no support for overload protection. This thesis presents the design, implementation and evaluation of architectures that provide overload protection and service differentiation in web servers. During server overload not all requests can be processed in a timely manner. Therefore, it is desirable to perform service differentiation, i.e., to service requests that are regarded as more important than others. Since requests that are eventually discarded also consume resources, admission control should be performed as early as possible in the lifetime of a web transaction. Depending on the workload, some server resources can be overutilized while the demand on other resources is low because certain types of requests utilize one resource more than others.

The implementation of admission control in the kernel of the operating system shows that this approach is more efficient and scalable than implementing the same scheme in user space. We also present an admission control architecture that performs admission control based on the current server resource utilization combined with knowledge about resource consumption of requests. Experiments demonstrate more than 40% higher throughput during overload compared to a standard server and several magnitudes lower response times.

This thesis also presents novel architectures and implementations of operating system support for predictable service guarantees. The Nemesis operating system provides applications with a guaranteed communication service using the developed TCP/IP implementation and the scheduling of server resources. SILK (Scout in the Linux kernel) is a new networking stack for the Linux operating system that is based on the Scout operating system. Experiments show that SILK enables prioritizing and other forms of service differentiation between network connections while running unmodified Linux applications.

B.2.37 Zhang, Yi, ARTES project no. 8, CTH.

PhD, June 12, 2003, Non-blocking Synchronization: Algorithms and Performance Evaluation.

The thesis investigates non-blocking synchronization in shared memory systems, in particular in high performance shared memory multiprocessors and real-time shared memory systems. We explore the performance impact of non-blocking synchronization in high performance shared memory multiprocessors and the applicability of non-blocking synchronization in real-time systems.

The performance advantage of non-blocking synchronization over mutual exclusion in shared memory multiprocessors has been advocated by the theory community for a long time. In this work, we try to make non-blocking synchronization appreciated by application designers and programmers through a sequence of results. First, we develop a non-blocking FIFO queue algorithm which is simple and can be used as a building block for applications and libraries. The algorithm is fast and scales very well in both symmetric and non-symmetric shared memory multiprocessors. Second, we implement a fine-grain parallel Quicksort using non-blocking synchronization. Although fine-grain parallelism has been thought to be inefficient for computations like sorting due to synchronization overhead, we show that efficiency can be achieved by incorporating non-blocking techniques for sharing data and computation tasks in the design and implementation of the algorithm. Finally, we investigate how performance and speedup of applications would be affected by using non-blocking rather than blocking synchronization in parallel systems. We show that for many applications, non-blocking synchronization leads to significant speedup for a fairly large number of processors, while they never slow the applications down.

Predictability is the dominant factor in performance matrices of real-time systems and a necessary requirement for non-blocking synchronization in real-time multiprocessors. In this thesis, we propose two non-blocking data structures with predictable behavior and present an inter-process coordination protocol that bounds the execution time of lockfree shared data object operations in real-time shared memory multiprocessors. The first data structure is a non-blocking buffer for real-time multiprocessors. The buffer gives a way to concurrent real-time tasks to read and write shared data and allows multiple write operations and multiple read operations to be executed concurrently and has a predictable behavior. Another data structure is a special wait-free queue for real-time systems. We present efficient algorithmic implementations for the queue. These queue implementations can be used to enable communication between real-time tasks and non-real-time tasks in systems. The inter-process protocol presented is a general protocol which gives

predictable behavior to any lock-free data structure in real-time multiprocessors. The protocol works for the lock-free implementations in real-time multiprocessor systems in the same way as the multiprocessor priority ceiling protocol (MPCP) works for mutual exclusion in real-time multiprocessors. With the new protocol, the worst case execution time of accessing a lock-free shared data object can be bounded.

B.3 Theses of students without project support from ARTES

The following is a list, in alphabetical order, of theses presented by ARTES students without project support from ARTES. These students have been registered as ARTES Real-Time Graduate Students and have benefited from and participated in ARTES network and graduate school activities.

Ahlström, Kristina, CTH.

-Licentiate, December 6, 2000, On Conceptual Design of By-Wire Systems.

Annell, Tobias, UU.

-Licentiate, October 13, Code Synthesis for Timed Automata.

Andersson, Johan, MDH.

-Licentiate, June 16, 2005, Modeling the Temporal Behavior of Complex Embedded Systems - A Reverse Engineering Approach.

Andersson, Per, LU.

-PhD, June 17, 2005, Efficient Modelling and Synthesis of Data Intensive Reconfigurable Systems.

Askerdal, Örjan, CTH.

-Licentiate, December 8, 2000, Design and Evaluation Techniques for Detection and Coverage Estimation of Low-Level Errors.

-PhD, June 4, 2003, On Impact and Tolerance of Data Errors with Varied Duration in Microprocessors.

Bengtsson, Johan, UU.

-Licentiate, May 22, 2001, Efficient Symbolic State Exploration of Timed Systems: Theory and Implementation.

-PhD, June 7, 2002, Clocks, DBMs and States in Timed Systems.

Claesson, Vilgot, CTH.

-Licentiate, June, 1999, Cost Effective Communication Services for Applications in Distributed Time Triggered Real-Time Systems .

Cortés, Luis Alejandro, LiU.

-PhD, March 2, 2005, Verification and scheduling techniques for real-time embedded systems.

Curescu, Calin, LiU.

-Licentiate, May 6, 2003, Adaptive QoS-aware Resource Allocation for Wireless Networks.

-PhD, September 16, 2005, Utility-based optimisation of resource allocation for wireless networks.

Eker, Johan, LU.

-PhD, December 2, 1999, Flexible Embedded Control Systems - Design and Implementation.

El Shobaki, Mohammed, MDH.

-Licentiate, September 24 2004, On-Chip Monitoring for Non-Intrusive Hardware/Software Observability.

Enblom, Leif, MDH.

-Licentiate, November 28, 2003, Utilizing Concurrency to Gain Performance in an Industrial Automation System.

Engblom, Jakob, UU.

-PhD, April 19, 2002, Processor Pipelines and Static Worst-Case Execution Time Analysis.

Eriksson, Joakim, HS.

-Licentiate, December, 1998, Specifying and Managing Rules in an Active Real-Time Database System .

Ermedahl, Andreas, UU.

-PhD, June 3, 2003, A Modular Tool Architecture for Worst-Case Execution Time Analysis.

Falkenroth, Esa, LiU.

-PhD, June 8, 2000, Database Technology for Control and Simulation.

Fan, Xing, HH.

-Licentiate, November 17 2004, Real-time communication services for distributed computing over switched Ethernet.

Feldt, Robert, CTH.

-Licentiate, November, 1998, Using Genetic Programming to Systematically Force Software Diversity.

Fersman, Elena, UU.

-PhD, November 26, 2003, A Generic Approach to Schedulability Analysis of Real-Time Systems.

Forsberg, Kristina, CTH.

-PhD, June 10, 2003, Design Principles of Fly-By-Wire Architectures.

Fredriksson, Johan, MDH.

-Licentiate, April, 2005, Transformation of component models to real-time models.

Fröberg, Joakim, MDH.

-Licentiate, April 2, 2004, Engineering of Vehicle Electronic Systems: Business Requirements Reflected in Architecture.

Furunäs, Johan, MDH.

-Licentiate, December 11, 2001, Interprocess Communication Utilising Special Purpose Hardware.

Glimming, Johan, KTH.

-Licentiate, September 19, 2005, Dialgebraic Semantics of Typed Object Calculi.

Gäfvert, Magnus, LU.

-PhD, May 9, 2003, Topics in Modeling, Control, and Implementation in Automotive Systems .

Hansson, Jörgen, HS and LiU.

-PhD, September 21, 1999, Value-Driven Multi-Class Overload Management in Real-Time Database Systems.

Henriksson, Dan, LU.

-PhD, January 13, 2006, Resource-Constrained Embedded Control and Computing Systems.

Herzog, Erik, LiU.

-PhD, May, 7, 2004, An Approach to Systems Engineering Tool Data Representation and Exchange.

Hessel, Anders, UU.

-Licentiate, March 7, 2006, Model-based Test-Case Selection and Generation for Real-Time Systems.

Hiller, Martin, CTH.

-Licentiate, December 16, 1999 , Using Software to Handle Data Errors in Embedded Control Systems .

-PhD, 2002, A Software Profiling Methodology for Design and Assessment of Dependable Software.

Huselius, Joel, MDH.

-Licentiate, November 10, 2003, Preparing for Replay.

Jervan, Gert, LiU.

-PhD, May 20, 2005, Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems.

Johannessen, Per, CTH.

-Licentiate, November 30 2001, Design Methods for Safety Critical Automotive Architectures.

-PhD, November 25, 2004, On the Design of Electrical Architectures for Safety-Critical Automotive Systems.

Johnsson, Andreas, MDH.

-Licentiate, April, 2005, Bandwidth Measurements in Wired and Wireless Networks.

Johnsson, Charlotta, LU.

-PhD, March 25, 1999, A Graphical Language for Batch Control.

Karlsson, Magnus, CTH.

-PhD, December 3, 1999, Data Prefetching Techniques Targeting Single and a Network of Processing Nodes.

Larses, Ola, KTH.

-Licentiate, November 13, 2003, Dependable Architectures for Automotive Electronics; Philosophy, Theory and Practice.

Larsson, Fredrik, UU.

-Licentiate, May 30, 2000, Efficient Implementation of Model-Checkers for Networks of Timed Automata .

Larsson, Magnus, MDH.

-Licentiate, December 1, 2000, Applying Configuration Management Techniques to Component-Based Systems.

-PhD, March, 2004, Predicting Quality Attributes in Component-based Software Systems.

Lennvall, Tomas, MDH.

-Licentiate, May 2003, Handling Aperiodic Tasks and Overload in Distributed Off-line Scheduled Real-Time Systems.

-PhD, September 27, 2005, Adapting to Varying Demands in Resource Constrained Real-Time Devices.

Lin, Man, LiU.

-Licentiate, October 1997, Formal Analysis of Reactive Rule-Based Programs .

-PhD, January 21, 2000, Analysis and synthesis of Reactive Systems: A Generic Layered Architecture Perspective.

Lincoln, Bo, LU.

-PhD, May 17, 2003, Dynamic Programming and Time-Varying Delay Systems.

Lindgren, Markus, MDH.

-Licentiate, December 6, 2000, Measurement and Simulation Based Techniques for Real-Time Analysis.

Lundqvist, Kristina, UU.

-PhD, April 14, 2000, Distributed Computing and Safety Critical Systems in Ada .

Lundqvist, Thomas, CTH.

-Licentiate, June, 1999, A Static Timing Analysis Method for Programs on High-Performance Processors .

Lönn, Henrik, CTH.

-PhD, October 8, 1999, Synchronization and Communication Results in Safety-Critical Real-Time Systems .

Mellin, Jonas, HS.

-Licentiate, 1998, Predictable event monitoring. .
-PhD, June 3, 2004, Resource-Predictable and Efficient Monitoring of Events.

Meynard, Jean Paul, LiU.

-Licentiate, May 4, 2000, Control of industrial robots through high-level task programming .

Möller, Anders, MDH.

-Licentiate, January 28, 2005, Software Component Technologies for Heavy Vehicles.

Nilsson, Jim, CTH.

-Licentiate, October 15, 1999, Reducing Ownership Overhead in Transaction Processing Multiprocessor Servers.

Nilsson, Marcus, UU.

-Licentiate, December 4, 2000, Regular model checking.
-PhD, March 2, 2005, Regular Model Checking.

Orebäck, Anders, KTH.

-PhD, November, 19, A Component Framework for Autonomous Mobile Robots.

Petterson, Lennart, KTH.

-Licentiate, June, 1999 , Control System Architecture for a Walking Robot .

Petterson, Paul, UU.

-PhD, February 19, 1999, Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice.

Redell, Ola, KTH.

-Licentiate, May 7, 1998, Modelling of Distributed Real-Time Control Systems - An Approach for Design and Early Analysis .

Sandström, Kristian, KTH and MDH.

-Licentiate, April, 1999, Modeling and Scheduling of Control Systems .
-PhD, April 26, 2002, Enforcing Temporal Constraints in Embedded Control Systems.

Sanfridson, Martin, KTH.

-Licentiate, May 30, 2000, Timing problems in distributed control .
-PhD, June 4, 2004, Quality of Control and Real-time Scheduling - Allowing for time-variations in computer control systems.

Sebek, Filip, MDH.

-Licentiate, October 2002, Instruction Cache Memory Issues in Real-Time Systems.

Sivencrona, Håkan, CTH.

-Licentiate, November 16 2001, On Analysis and Design of Dependable Distributed Systems.

Sjödín, Mikael, UU.

-PhD, May 24, 2000, Predictable High-Speed Communications for Distributed Real-Time Systems.

Szentivanyi, Diana, LiU.

-Licentiate, December 13, 2002, Performance and Availability Tradeoffs in Fault-Tolerant Middleware.

-PhD, March 29, 2005, Performance Studies of Fault-tolerant Middleware.

Szymanek, Radoslaw, LU.

-PhD, June 16, 2004, Constraint-Driven Design Space Exploration for Memory-Dominated Embedded Systems.

Weckstn, Mattias, HH.

-Licentiate, April 20, 2004, Resource Budgeting as a Tool for Reduced Development Cost for Embedded Real-time Computer Systems.

Vera, Xavier, MDH.

-PhD, January 26, 2004, Cache and Compiler Interaction (how to analyze, optimize and time cache behavior).

Wilhelmsson, Jesper, UU.

-Licentiate, May 27, 2005, Efficient Memory Management for Message-Passing Concurrency.

Vinter, Jonny, CTH.

-Licentiate, December 14, 2001, Software-Implemented Error Detection and Recovery for Control Applications.

-PhD, June 9, 2005, On the Effects of Soft Errors in Embedded Control Systems .

Zeffner, Håkan, UU.

-Licentiate, May 30, 2005, Hardware-Software Tradeoffs in Shared-Memory Implementations.

Åkerholm, Mikael, MDH.

-Licentiate, February, 2005, A Software Component Technology for Vehicle Control Systems.

Åkesson, Knut, CTH.

-Licentiate, December 21, 1999, Recipe Coordination in Chemical Batch Processes.

Appendix C

Research groups

This appendix presents the academic research groups that participated in ARTES. In addition to the information provided below, Up to date web-links to the the web-pages of the participating groups are available at www.artes.uu.se/groups/

The presentation is given from north to south. In the presentation “++” is used to indicate that the group includes one or more ARTES++ graduate students, and numbers used to identify projects refer to the numbering in the list of projects in Appendix A.1. Also the listed principal investigators are the main senior researchers in the group involved in ARTES, i.e., there may be other PIs in the group.

C.1 Luleå University of Technology Department of Computer Science and Electrical Engineering.

C.1.1 Embedded Internet System Laboratory (EISLAB).

EISLAB covers research and education in the field of electronics, computer engineering and robotics. Major fields of research are, measurement and sensor technology, ultrasound technology, mixed mode electronics design, low-power ASIC design, embedded EMC, EIS architecture and autonomous robotics. Education emphasizes on electronics, computer engineering, EIS, measurement technology, EMC and robotics. The staff consists of 12 faculty, 2 senior lecturers, 20 Ph.D. students and 4 support staff.

Principal investigators: Johan Nordlander, Per Lindgren
ARTES projects: ++

C.2 Mid Sweden University, Department of Information Technology and Media, in Sundsvall.

C.2.1 Multimedia Communication Systems Lab.

The lab carries out research in the fields of computer networks and multimedia communication systems. The main research areas are 1) coding and transmission of multimedia, 2) radio resource management. The research involves application areas such as: Video-on-demand, 3D Video, WLAN, 3G, DVB, Interactive TV, and Broadband Wireless Access.”

Principal investigators: Tingting Zhang

ARTES projects: ++

C.2.2 The Electronics Design Division.

Computational electronics and photonics Optimization, analysis and design of electronic and photonic devices in new advanced materials using numerical tools and models based on so-called first principles. Electronics system design. Development of design methods and strategies for advanced digital systems with a focus on real time data processing in multimedia applications.

Principal investigators: Mattias O’Nils

ARTES projects: ++

C.3 Uppsala University, Department of Information Technology.

C.3.1 Modeling and Analysis of Real Time Systems.

The current research directions include: model-extraction, verification and testing of embedded real-time software, schedule synthesis and component-based design technology. The main activities have been the development of UPPAAL - a model-checker for timed systems, developed and maintained jointly with Aalborg University, Denmark, and TIMES - a tool for schedulability analysis and code generation guaranteeing timing constraints.

Principal investigators: Wang Yi, Paul Pettersson

ARTES projects: ++, 12

C.3.2 Communications Research.

In our research we study how computer systems and communication protocols should be designed in order to make efficient use of network capacity and meet the demands of emerging services, e.g. multimedia services. We focus on end-to-end protocol issues and the implementation of communication software running on end systems.

Principal investigators: Per Gunningberg

ARTES projects: 28

C.3.3 Algorithmic Program Verification.

The aim of the project is to develop new methods for algorithmic verification of computer systems. The main topics of research include:

- 1) To extend automatic verification techniques to new classes of systems with infinitely many states and with various data structures.
- 2) Incorporating techniques from SAT-solving, constraint-solving, abstract interpretation, symbolic execution, into verification methods to extend their efficiency and applicability.

Principal investigators: Parosh Abdulla, Bengt Jonsson.

ARTES projects: ++, 28

C.3.4 Uppsala Architecture Research Team.

Our research focus is to increase data processing speed through adopting architectures and coherent data replication. We are performing research in the following areas:

- shared-memory multiprocessor architectures
- Cache-Only Memory Architectures (COMA)
- dynamic adapting caching algorithms
- caching techniques in real time systems
- thread migration
- selective data replication
- access pattern categorization
- speculative execution
- dynamic speculative data delivery

Principal investigators: Erik Hagersten

ARTES projects: 31, 35

C.4 Mälardalen University, in Västerås, Dept. of Computer Science and Electronics

C.4.1 Programming language group

Focus: Worst-case execution time analysis, as well as design and analysis of languages for real-time and embedded systems.

Principal Investigator: Björn Lisper

ARTES projects: ++

C.4.2 Embedded Systems Software Engineering group

Focus: Embedded Systems Software Engineering (e.g. for automotive systems).

Principal Investigator: Christer Norström

ARTES projects: ++, 6, 26b

C.4.3 Industrial Software Engineering group

Focus: Software Engineering for industrial systems (e.g. for automation systems).

Principal Investigator: Ivica Crnkovic

ARTES projects: ++

C.4.4 Communication Performance and Small Systems group

Focus: Communication for small embedded devices, and traffic measurement and analysis.

Principal Investigator: Mats Björkman

ARTES projects: ++

C.4.5 Monitoring and testing group

Focus: Monitoring, testing, and debugging of real-time systems.

Principal Investigator: Henrik Thane

ARTES projects: 7,20

C.4.6 Predictably Flexible Real-Time Systems group

Focus: Static and dynamic real-time scheduling, combining flexibility and predictability.

Principal Investigators: Gerhard Fohler, Damir Isovica

ARTES projects: 14, 24, ++

C.4.7 Real-Time Systems Design Group

Focus: Design methods, architectures and communication for real-time systems.

Principal Investigator: Hans Hansson, Mikael Nolin

ARTES projects: 2, 23, ++

C.4.8 Artificial Intelligence and Intelligent Systems group

Focus: Methods and techniques for intelligent systems enabling experience reuse, learning and human computer collaboration.

Principal Investigator: Peter Funk

ARTES projects: ++

C.4.9 Sensors and Biomedical Engineering group

Focus: Sensors and measurement systems for biomedical applications

Principal Investigator: Ylva Bäcklund, Maria Lindén

ARTES projects: ++

C.5 Swedish Institute of Computer Science in Kista.

C.5.1 Computer and Network Architectures Laboratory.

The research focus of the Computer and Network Architecture laboratory is in the networking technology and the computing platform for advanced distributed applications on the Internet. The networking technology ranges from the design of hardware for IP routers over core Internet protocols in the network and transport layers and mobile networking to how distributed applications use the network service. Within computer architecture the lab has specialized in technology for full-system instruction set simulation. The focus in this area is now on using simulation technology for analyzing and debugging systems with real-time requirements.

Principal investigators: Bengt Ahlgren, Lars Albertsson

ARTES projects: 27

C.6 Royal Institute of Technology, Stockholm

C.6.1 Department of Machine Design, Division of Mechatronics, Embedded Control Systems research group.

The overall research goal is to provide a scientific basis through knowledge, methods and techniques for the development of future dependable and cost-efficient embedded control systems. The focus is on architectural design and interdisciplinary design techniques in the context of mechatronics - a specialization of systems engineering. The research areas include:

- Integrated control and embedded computer design; distributed control systems
- Architectural design: methodology and analytical techniques
- Model based development, specifically for model and tool integration, and embedded software
- Processes and work integration

Principal investigators: Martin Törngren, Jan Wikander

ARTES projects: 5, 9, 18, 22

C.6.2 Department of Microelectronics and Information Technology, The Department of Electronic.

Focus: Computer, and Software Systems, System Architecture and Methodology.

Principal investigators: Mats Brorsson

ARTES projects: 32

C.7 Linköping University, Department of Computer and Information Science,

C.7.1 Real-Time Systems Laborator (RTSLAB).

We conduct research in the areas of dependability and resource allocation in distributed systems. Among the topics addressed under the term dependability are:

- Formal analysis of safety and fault-tolerance
- Availabilty and survivability in critical infrastructures

Within resource allocation and distributed systems topics range over:

- Quality of service (QoS) guarantees for real-time databases
- Resource constrained embedded databases
- Optimal resource allocation in wireless mobile networks

RTSLAB combines the above expertise areas in projects treating configurable components in real-time, fault-tolerant, and safety-critical systems.

Principal investigators: Simin Nadjm-Tehrani and Jörgen Hansson

ARTES projects: 25, 26a, ++

C.7.2 Embedded Systems Laboratory (ESLAB)

ESLAB conducts research on the design and test of embedded systems, especially those consisting of tightly coupled hardware and software components. Special emphasis is placed upon the development of methods and tools for:

- specification
- modeling
- synthesis
- simulation
- design for test
- test synthesis
- hardware/software co-design

We are also concerned with the exploitation of systematic design and design automation techniques for industrial applications. Principal

investigators: Zebo Peng, Petru Eles

ARTES projects: 1, 4, ++

C.8 University of Skövde, School of Humanities and Informatics.

C.8.1 The Distributed Real-Time Systems Research Group.

The group is carrying out research in the area of distributed real-time systems and especially distributed real-time database systems and

timeliness testing of event-triggered real-time systems. The objectives of the Distributed Real-Time Systems research group is to study and solve the special synchronization, communication, and dynamic scheduling problems that appear in complex distributed real-time systems. In particular we study real-time database systems with soft, firm and hard deadlines, reactive mechanisms and event monitoring, as well as software timeliness testing based on techniques developed in this area. Principal investigator: Sten F. Andler

ARTES projects: 17, ++

C.9 Chalmers University of Technology, The Department of Computer Science and Engineering.

C.9.1 The Division of Computing Science, Distributed Computing and Systems Research Group,

The group is involved in research on i) Distributed and Parallel Computing and Systems, and ii) Information Visualization of Complex Systems, focusing on the following issues:

Distributed and Computing and Systems:

- Investigate new techniques for achieving high parallelism and fault-tolerance in distributed or parallel software.
- Efficient design of fundamental distributed and parallel data structures.
- Evaluate the performance of non-blocking synchronization in parallel application and system software (Lockless-Spark98, Lockless-mini-SPLASH2).
- Bridge the gap between research on interprocess synchronisation and use in application and systems software through case studies and the development of software libraries (NOBLE).
- Multi-peer information dissemination and consistency support
- Modelling and design of virtual objects for distributed collaborative environments

Information Visualization of Complex Systems:

- Visualization techniques to enhance understanding of the behaviour of distributed algorithms (LYDIAN).

- Effective visualization of of "cause and effect" relations (CausalViz).
- Three-dimensional user interfaces (3DWM).

Principal investigators: Marina Papatriantafidou and Philippas Tsigas
ARTES projects: 8

C.9.2 The Division of Computer Engineering, High-Performance Computer Architecture Group

Focus: design principles and methods for emerging general-purpose computer systems for industrial applications with high and predictable (real-time) performance demands. Target applications are e.g. transaction-processing, telecom, real-time 3D computer graphics as well as industrial process control. A current focus is on the exploitation of symmetric multiprocessor systems and high-performance microprocessors in these and other performance demanding application domains. The work in the group is carried out in a number of projects. The topics span issues in architecture, modeling, and scheduling and parallelization methods.

Principal investigator: Per Stenström
ARTES projects: 29, 33, 34, ++

C.9.3 The Division of Computer Engineering, Computer Graphics Research Group,

Focus: Soft Shadows - both real-time and non real-time, and real-time Ambient Occlusion Ray Tracing, Global Illumination, Real-time rendering algorithms, Parallelism.

Principal investigator: Ulf Assarsson
ARTES projects: 29

C.9.4 The Division of Computer Engineering, The Fault-tolerant Computing for Embedded Applications The FORCE group.

Our research area is fault-tolerant embedded computer systems. We investigate and design cost-effective hardware and software mechanisms for error detection and error recovery. We also develop experimental and analytical techniques for prediction of error coverage and validation of fault-tolerant systems.

Principal investigator: Johan Karlsson, Jan Jonsson
ARTES projects: 10, 11, 29, ++

C.10 Halmstad University, School of Information Science, Computer and Electrical engineering(IDE)

C.10.1 Computing and Communication lab.

Research is done within Embedded systems, in the first place within the CERES research profile.

Principal investigators: Bertil Svensson, Magnus Jonsson, Tony Larsson

ARTES projects: 15, 16, 21, ++

C.11 Blekinge Institute of Technology, School of Engineering

C.11.1 Parallel Architectures and Applications for Real-Time Systems (PAARTS) group.

Focus:

- Methods, guidelines and tools for cost-effective development of parallel applications for multiprocessors. This includes techniques for handling conflicts and tradeoffs between performance and maintainability.
- Methods for visualizing the behavior of a parallel program to the application developer.
- Availability aspects, including the use of cluster technology for obtaining high availability.
- NP-hard resource allocation algorithms, e.g. scheduling, is another interesting area. In this area we focus on mathematical techniques for establishing optimal performance bounds on NP-hard resource allocation problems, thus making it possible to compare the performance of heuristic resource allocation techniques with the optimal result.

Principal investigators: Lars Lundberg

ARTES projects: 30

C.12 Lund University

C.12.1 The Department of Computer Science, Embedded Systems Design Laboratory (ESDlab).

Research is focused on different aspects of embedded computer systems design. It involves a number of areas, such as real-time systems, distributed processing, (discrete) system modeling and analysis, compilation techniques, program and model transformations, and optimization methods. The big challenge of this research is to find efficient methods and tools which can be used in industry to enhance a design process, make time-to-market shorter and improve overall quality of the final products.

Principal investigators: Krzysztof Kuchcinski

ARTES projects: 19

C.12.2 The Department of Automatic Control, Control and Real-Time Computing group.

Focus: networked embedded control, real-time techniques in control system implementation, and control of real-time computing systems.

Principal investigator: Karl-Erik Årzén

ARTES projects: 3, ++

C.12.3 The Department of Computer Science, Software Development Environments group

The research is centered around a number of core areas within software development support, with central themes of integrated environments, object-oriented languages (in the tradition of Simula, BETA, and Java), and embedded systems such as industrial robots and mobile phones. The research method is focused on experimental implementation and development of theory that is of practical relevance. There is much interaction between the individual projects, with a strive for integrating the developed core technologies.

Principal investigator: Boris Magnusson, Klas Nilsson

Artes projects: 3, ++

Appendix D

Programme plan

ARTES programme plan for 1998-2002.

The whole plan including appendices is available at
<http://www.artes.uu.se/plan/plan.pdf>

ARTES



ARTES

**A network for Real-Time research and
graduate Education in Sweden**

Programme Plan
1998-2002

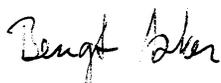
Preface

The ARTES national network officially started in January 1998, after almost three years of planning. The network has its roots in the Swedish National Association for Real-Time (SNART), which is an association formed in the early 90's to promote real-time systems research and related industry-academia cooperation. The launching of ARTES has been performed in a true SNART-spirit as a process with active involvement of both industry and academia.

Up-until-now, ARTES has sponsored three graduate courses, all given in a distributed fashion with participants from several universities and industry. Also, support for three projects, involving four graduate students has been decided. Additional graduate courses as well as research projects will be initiated during 1998. The purpose of this programme plan is to outline the activities and focus of ARTES during the years 1998-2002.

In addition to ARTES, the programme plan also covers PAMP which is an associated programme that is managed by ARTES. The administrative provisions for handling PAMP is included in this plan, whereas the PAMP research plan is provided separately in Appendix D.

This plan was approved by the ARTES board of directors April 21, 1998.



Bengt Asker
Chairman of the Board



Hans Hansson
Programme Director

Executive Summary

The Swedish Foundation for Strategic Research (Stiftelsen för Strategisk Forskning; SSF) has established the national research programme ARTES (A network for Real-Time research and graduate Education in Sweden). This programme plan presents ARTES activities and goals for the first five year period: 1998-2002. Additionally, a programme plan for the associated programme PAMP (Symmetric Multiprocessors in High-Performance Real-Time Applications) is provided in Appendix D. The PAMP research projects will be treated separately, whereas the PAMP graduate school will be integrated with ARTES. The PAMP administration and relation to ARTES is presented in the main sections of this plan.

ARTES is dedicated to Real-Time Systems research, which is multidisciplinary in that it (at least) includes aspects of Automatic Control, Computer Science, Computer Engineering and Electrical Engineering. There is also a multitude of application areas for Real-Time Systems, ranging from embedded micromechanical systems to world-wide communication systems. The area is of paramount importance to Swedish industry, since many Swedish companies are designing and manufacturing products with real-time systems as vital components.

Several of the strategic research centres supported by SSF (e.g., Computing & Communication Systems, Autonomous Systems, and the Excellence Center in Computer Science and Systems Engineering) have strong real-time orientations. The ARTES national network will complement these and other local programmes as well as strengthen Swedish real-time research in general by strengthening graduate education and providing an infrastructure for national and international cooperation.

The research in the ARTES programme is guided by the following twofold vision:

For the network:

To transfer knowledge and competence to Swedish industry that will allow it to first utilise the latest achievements in real time systems design.

Specifically for the research projects:

To reduce lead times for designing and modifying real time systems by an order of magnitude by year 2005.

Research in component based design together with a network for intense interactions between academic and industrial groups will be instrumental in fulfilling the above vision.

ARTES will be organised as a network of nodes, where each node typically is an academic or industrial research/development group. Presently, ARTES consists of 22 industrial and 12 academic nodes, encompassing more than 40 senior researchers (including 14 professors). The current annual budget for real-time systems related research and graduate education at the academic nodes is in the order of MSEK 60/year. During the planning period this budget is expected to increase to more than MSEK 90/year, partly due to ARTES, but also due to expected increase in funding from other sources.

ARTES will play an important role in coordinating the total educational, research and transfer of technology effort in the real-time systems area, thus guaranteeing synergetic effects among the various efforts, thereby maximising the total yield and nationwide substantially increase the real-time competence.

The goals of ARTES are

- to increase the number of PhDs and Licentiates which play important roles in development of industrial real-time applications and products,

- to increase the efficiency of graduate education,
- active industrial involvement in research and graduate education, as well as academic involvement in industry,
- to maximise synergy between the real-time components in strategic centers supported by SSF, as well as with other efforts in the area,
- to increase national and international cooperation in real-time systems research and education, and
- to provide a broad base for Swedish real-time systems research, and to make Swedish real-time systems research world leading in selected areas.

To reach these goals ARTES has the following four activities on its programme:

Research Projects involving cooperation between industrial and academic network nodes.

The projects should tie together research/development at different nodes to maximise synergy, but also concentrate efforts on areas of current and potential high industrial interest. ARTES (including PAMP) plans to finance the graduate studies for 43 students, of which 36 are expected to complete a PhD by 2005.

The ARTES graduate school which is concentrate on nationally providing graduate courses and proposing curricula for real-time related graduate education. A special effort is the annual ARTES *summer school*, which will be both a traditional academic summer school with tutorials and a meeting place for academic and industrial nodes, where projects, other cooperations and general issues are discussed.

A Mobility Programme to increase interaction between industrial and academic network nodes, as well as with internationally leading research groups.

Infrastructure Support to provide information and establishing various types of cooperations involving ARTES nodes, such as support to workshops and establishment of international cooperation.

The below table summarises the budget related to the SSF support for ARTES/PAMP (A/P) during the planning period 1998–2002. It should be noted that the reason for the relative high administrative cost for 1998 is that costs from the actual start of the programme in July 1997 are included.

(MSEK)	1998		1999		2000		2001		2002		Total	
Funding	A	P	A	P	A	P	A	P	A	P	A	P
SSF	5.0	2.0	10.0	4.0	15.0	5.0	15.0	5.0	15.0	5.0	60.0	21.0

Costs	A	P	A	P	A	P	A	P	A	P	A	P
Projects	2.1	1.2	5.8	3.6	9.8	3.7	11.6	3.4	10.4	2.6	39.7	14.5
Grad. School	0.8		1.1		1.4		1.9		1.9		7.1	
Mobility	0.4		0.7		0.8		0.8		0.8		3.5	
Infrastructure	0.4		0.6		0.7		0.8		0.8		3.3	
Adm.	1.5		1.1		1.1		1.1		1.1		5.9	
VAT (8%)	0.6		1.1		1.6		1.6		1.6		6.5	
Total	7.0		14.0		19.1		21.2		19.2		80.5	
Funding–Costs	0.0		0.0		0.9		-1.2		0.8		0.5	

Contents

1 Introduction	1
2 The ARTES Vision	1
3 Objectives	3
4 Impact	4
5 The ARTES Programme	5
5.1 Research Projects	5
5.2 The ARTES Graduate School	8
5.3 The ARTES Mobility Programme	10
5.4 Infrastructure	11
6 Milestones	11
7 Schedule of Events	12
8 Review and Evaluation	13
9 Organisation	13
9.1 The formal ARTES organisation	13
9.2 The informal ARTES network	16
10 Financial Plan and Budget	17
11 Collaborations	18
12 Other Plans and Policies	19
12.1 Continuation of successful programme	19
12.2 Future careers of young researchers	19
12.3 Recruitment of female students	19
12.4 Intellectual property and exploitation of results	19
12.5 Conditions for discontinuation of the programme	20
A The project process	21
B ARTES - The Academic Network	22
C Addresses	30
D PAMP - programme plan	32

1 Introduction

Real-time systems is an IT area of high relevance for current and emerging Swedish industry. The aim of the ARTES programme is to provide Swedish industry with competence and qualified personnel in this important area by focusing and developing Swedish real-time systems research and graduate education. This is to allow Swedish industry not only to stay competitive, but also to strengthen its competitiveness.

Real-Time Systems

In many applications, computer systems sense their environment and directly influence it through actions. Such systems are subject to the real-time constraints of the environments in which they operate. For example, an autonomous vehicle needs a control system that responds quickly enough to avoid collisions. This requirement for timely behaviour is the defining characteristic for *real-time systems*. Real-time systems must not only choose appropriate actions but also choose actions at appropriate times. Research in real-time systems addresses precisely this issue by developing methods for guaranteeing timely reaction. Real-time computing is not about building “fast” systems; it is about building systems predictably “fast enough” to act on their environments in well-specified ways.

Real-time systems provide an enabling technology for many important application areas, including: multimedia, telecommunications, robotics, process control, flexible manufacturing, avionics, vehicular systems, air-traffic control, nuclear power plants, medical equipment, and defence applications. In particular, almost all safety-critical systems are real-time systems. ARTES will serve to produce generic results and qualified personnel required in development of all these important applications. The benefits are likely to be immense since the results are potentially usable by so many industries critical to the Swedish economy.

Real-time computing is playing a key role in many sectors of Swedish industry. For instance, the automotive manufacturers can only stay competitive if they incorporate state-of-the-art real-time systems into their vehicles. In the future, distributed real-time control systems will replace and enhance many of the conventional control systems in automobiles, making them more efficient and improving public safety. Before distributed real-time systems can be used there are a number of significant research challenges that must be addressed, e.g. precise real-time response (to the microsecond), fault tolerance under strict timing requirements, maintainability, and testability, all under competitive pricing pressure. ARTES will facilitate the development and transition of current real-time technology in this and other application domains. In particular, the catalytic effects on graduate (and undergraduate) education will help industry solve the critical problem of finding qualified personnel.

Real-Time Systems Research

Real-Time Systems is a multidisciplinary research area, in that it includes (at least) aspects of Automatic Control, Computer Science, Computer Engineering and Electrical Engineering. There is also a multitude of application areas for Real-Time Systems, ranging from embedded micromechanical systems to world-wide communication systems.

2 The ARTES Vision

The research in the ARTES programme is guided by the following twofold vision:

For the network:

To transfer knowledge and competence to Swedish industry that will allow it to first utilise the latest achievements in real time systems design.

Specifically for the research projects:

To reduce lead times for designing and modifying real time systems by an order of magnitude by year 2005.

Research in component based design together with a network for intense interactions between academic and industrial groups will be instrumental in fulfilling the above vision.

It should be noted that, even though the above specific vision is focussed on the timing dimension of real-time systems development (speeding up the development and modification processes), we consider it to also implicitly cover other dimensions, such as

- **the quality/correctness dimension** with its focus on improving the quality of the design and the confidence in the correctness of the designed component/product. A plausible vision in this dimension is *provably correct designs*, meaning mathematical proofs of all relevant aspects of the design, including proving that requirements related to behaviour, timing and reliability of the designed component/product are met.
- **the size dimension** with its focus on handling systems substantially larger than current real-time systems. A possible vision in this dimension is to be able to conveniently design systems that are an order of magnitude larger the systems we are designing today.

One important basis for the ARTES vision is that most industrial real-time systems are built from components of various origin. It is not feasible to build a reasonably sized system with exclusively tailor made components. Some will be purchased, some will be inherited from existing systems, and they will not always make a perfect fit. Hence, industry has an urgent need for a better scientific foundation on how to develop this type of heterogeneous real-time systems.

This leads to a profile for ARTES with emphasis on a building block approach for designing real-time systems. The main problems typically arise in the early and late development phases, i.e. design and integration, respectively. Some relevant questions are:

- How to specify and design a heterogeneous real-time system?
- How to evaluate an early design architecture for a real-time system?
- What is required for the system to be able to grow and change?
- What are the implication for processes, methods and tools for real-time systems?

These problems are not unique for real-time systems, but solutions that are sufficient for other systems may not apply when requirements related to timing guarantees, predictability, availability and reliability are added. Specific aspects/topics that are important to consider include (but are not limited to):

- **Dependability**, which is a key issue in any computer/communication system that controls a critical application. Safety levels of 10^{-10} to 10^{-8} catastrophic failures per hour is often required. Dependability of that order can only be achieved by careful use of methods for errorless design in combination with tolerance and robustness against different types of hardware and software faults. Important research areas include: methods for fault detection, dependability in distributed systems, software fault-tolerance, and dependability validation.

- **Formal methods** are the use of mathematical techniques in the design and analysis of computer hardware and software. Formal methods allow properties of a computer system to be predicted from a mathematical model of the system by calculation. Application areas and research issues of formal methods include: execution-time analysis, schedulability analysis, specification of system requirements and behaviours, automatic code-generation, as well as verification of safety, real-time, and reliability properties.
- **Resource handling** concerns efficient handling of computer system resources to provide progress and proper sharing such that the system requirements are met. This problem is especially difficult for real-time systems, since by sharing (otherwise) unrelated processes may interfere such that their real-time requirements are violated. Resource handling includes both allocation (placement in space) and scheduling (sharing in time). Important research issues include: schedulability analysis, dynamic allocation and load-balancing, handling of overload situations, execution-time analysis, distributed and parallel system scheduling and allocation, taking control information into account, operating system support for predictable scheduling and allocation, as well as tool and language issues.
- **Distributed systems** are loosely coupled multicomputer system, geographically as well as logically distributed. A distributed real time system is complicated by the latency in communication, the need for distributed synchronisation, concurrent execution and that some computers may fail during normal operation. Central research in the area focus on how to make a distributed system predictable in the time domain under these complications. This involves both handling soft Quality of Service (QoS) type of requirements and strict hard requirements. Research fields and issues include: real-time and predictable communication systems, distributed scheduling, real-time platforms, dependability and fault-tolerance, formal methods, as well as programming systems for expressing distribution, concurrency, and real time requirements.
- **Real-time databases** are databases with database management systems that in addition to performing normal database functions are required to provide their responses in a timely manner. It is required that the system allows accurate estimates on the average and worst-case time taken to execute queries and operations. Due to the unpredictable behaviour of most database systems, real-time databases are often seen as only suitable in systems with soft real-time requirements. Research issues include: development of database systems that can be used also in systems with a mixture of soft and hard real-time requirements, handling of temporary overloads, increasing predictability (e.g., by distribution and replication), temporal databases for storing values and times, as well as active databases capable of taking actions in response to modifications of data.
- **Implementation of control systems** is the dominating application area for real-time technology. Real-time systems and control has developed into a recognised sub-field of automatic control. Application areas include industrial control systems for the process, manufacturing and power industry, embedded mechatronic systems, vehicular systems and aerospace systems. Research areas and issues include: distributed control systems, scheduling, operating systems and platforms, real-time languages, control that is robust against timing variations, models with both continuous and discrete elements, code generation from high-level specifications, as well as hardware and software architectures that support embedded open and yet efficient control system implementations.

3 Objectives

The objective of ARTES is to substantially increase the Swedish competence in real-time systems by strengthening real-time systems graduate education and research. The graduate students

should acquire knowledge in the multitude of aspects of real-time systems, including both theoretical and practical aspects, as well as skills in applying this knowledge in industrial type projects.

The specific goals of ARTES are

- to increase the number of PhDs and Licentiates which play important roles in development of industrial real-time applications and products. Concretely, the long term goal is 50 graduate degrees per year in ARTES related areas (current level is about 20 degrees per year), and that at least 80% of the graduates start an industrial career.
- to increase the efficiency of graduate education. The goal is that no ARTES graduate student should have a study-time exceeding the nominal times of 2 years for a licentiate and 4 years for a PhD (compensating for departmental duties and periods of industrial work).
- active industrial involvement in research and graduate education, as well as academic involvement in industry. Concretely, each ARTES supported project should have at least one active cooperation activity, e.g., in the form of an industrial engineer participating in the project, or a graduate student performing parts of his/her research in industry.
- to maximise synergy between the real-time components in strategic centers supported by SSF, as well as with other efforts in the area. Concretely, at least 50% of the ARTES supported projects should have a formalised or informal cooperation (e.g., in terms of joint papers) with related national programmes.
- to increase national and international cooperation in real-time systems research and education, and
- to provide a broad base for Swedish real-time systems research, and to make Swedish real-time systems research world leading in selected areas.

4 Impact

There is a substantial amount of real-time related research already in progress in Sweden, frequently involving close industrial cooperation (e.g., the initial ARTES application listed 150 active cooperations). However, these activities are only to a limited degree coordinated and their impact on graduate and undergraduate education is not matching the industrial demand for qualified personnel.

ARTES will be instrumental in achieving the necessary coordination between ongoing and planned real-time systems efforts, thereby achieving synergy which substantially increases the total yield. In particular, ARTES is focussed on coordinating graduate education, to provide industry with personnel qualified for the advanced technical development that will be required on the competitive markets of tomorrow.

The ARTES supported graduate education will be attractive to a wide group of students, not only to those that are primarily interested in academic research. Students interested in industrial careers will be attracted by the focus on industrial cooperation and industrially relevant research, together with the support and mechanisms provided for students to complete their graduate studies in time.

The coordination mechanism implemented by ARTES will give an outstanding platform for the ambition to identify uncovered research areas of strategic importance and accordingly establish

activities in these areas. The ARTES vision and profile will act as a basis for guiding and identifying required research.

ARTES also has a potential to be instrumental in the establishment of new industries. For instance, based on the yearly review of ongoing research, ARTES will identify research results which are commercially promising, and then assist in establishing contacts between the researchers and the appropriate financiers, industrialists and entrepreneurs to facilitate the creation of new ventures in the real-time systems area.

5 The ARTES Programme

ARTES is a network of nodes. As such, the main objective of ARTES is to play a catalytic role in stimulating cooperation. To ensure this, all ARTES activities are required to involve at least two nodes.

The ARTES programme is partitioned into the following four interrelated and mutually supportive sub-programmes:

1. Research Projects
2. The ARTES Graduate School
3. The ARTES Mobility Programme
4. Infrastructure support

5.1 Research Projects

An ARTES research project requires cooperation between individuals from different nodes, with the important objective to maximise synergy between local activities. Equally important is to provide relevant research problems and environments for graduate students.

A typical project includes at least one industry and several academic nodes. It typically considers several research areas, but is focussed on one application area. Preferably, the project applies research results to real industrial problems, or develops results conforming to industrial requirements.

The main evaluation criteria for a project are:

- The industrial relevance
- Its academic qualities
- How well it supports the ARTES vision and current profile
- The synergy it provides
- The degree of research cooperation with other SSF funded research programmes

The profile, which will develop as the network evolves, is an important mechanism for guiding and focussing the research to keep up with (and take a lead in) national and international industrial and academic developments.

Project requirements

An ARTES research project involves at least two ARTES nodes (where a node should be interpreted as a group active in the real-time systems area) of which at least one is industrial and at least one is academic. The industrial participation can take various forms, the major point being that the research is guaranteed to be industrially relevant and that exploitation possibilities are handled timely and effectively. Examples of industrial participation are:

- providing case studies and requirements,
- forming reference groups,
- active research work,
- providing equipment, and
- direct financing.

Projects are planned for 2 or 2+2 years, corresponding to the timing of licentiate and doctoral degrees. Since all projects are true collaborative efforts special emphasis is to be put on project leadership and organisation.

Project support is provided in *graduate student units*, one unit corresponding to the annual (marginal) cost for one full time graduate student including supervision, currently kSEK 500.

In order to consent to the 20-25% department duty rule for doctoral students at most universities the project application and later plans may include a time schedule taking department duty into account. This implies up to six months of delay per two year period.

All projects are presented annually to the ARTES network at the ARTES summer school and/or ARTES/SNART conference. All projects provide continuously updated project descriptions on the web.

Evaluation of Projects

Project applications are, as the general rule, evaluated by external experts and the ARTES board. In addition, there is a yearly review of projects and control points as outlined in Appendix A.

The following criteria are used in the evaluation of both the application and the results:

- scientific merits,
- industrial relevance, experience, participation and synergy,
- conformity with the ARTES vision and the real-time profile of the current call,
- mobility; both academia/industry and international,
- amount of interdisciplinary research,
- degree of research cooperation with other SSF funded research programmes,
- exploitation expectations,
- complementary activities and financing, critical mass,
- project management,

- multi node publications,
- case studies, application of research results,
- quality of presentations, and
- matching of objectives and resources.

The evaluation of a project application may result in approval, rejection or reformulation by the ARTES board. The latter meaning that the applicants are asked to reformulate their application. This is one of the ways in which the ARTES board can coordinate, synchronise and optimise activities within the network.

The process for ARTES projects is described in Appendix A.

Current projects

As a result of the first call for project proposals (with a November 10th 1997 deadline) support to the following three projects have been decided:

- *Integrated Control and Scheduling*

Supported with 2 graduate student units (initially for 2 years).

Project leader: Karl-Erik Arzén, Control Engineering, LTH.

This project is a cooperation between Automatic Control and Computer Science at Lund Institute of Technology, Sigma Exallon Systems AB, DDA Consulting AB, and Software Engineering Institute at Carnegie Mellon University.

The focus of the project is practical management of hard real-time demands in embedded software for real-time control. The project is based on two key ideas. The first idea is to combine control theory and scheduling theory in such a way that the nominal requirements on hard deadlines for control systems can be relieved. The approach taken is based on using dynamic feedback from the scheduler to the controllers and from the controllers to the scheduler. The second, complementary, idea is to use attribute grammars and incremental semantic analysis to carry out on-line interactive analysis of the worst-case timing of the software and generation of exception handling code for coping with unexpected delays. Combining these ideas forms a more complete and practical methodology than available today. The results will be packaged in tools that can be evaluated in industrial applications by the industrial partners.

- *Hardware Software Co-design*

Supported with 1 graduate student unit (initially for 2 years).

Project leader: Zebo Peng, Computer and Information Science Linköping University.

This is a project which is carried out at the Embedded Systems Laboratory (ESLAB), Linköping University. The project deals with system-level design methods and tools for mixed hardware/software systems, with special emphasis on real-time issues. Methods and tools for analysis of a given design based on a given architecture will be developed, as will methods for modifying a given architecture to obtain an optimal design of the system specification. The final objective is to develop techniques and tools to allow the designers to quickly explore different design alternatives and find cost-effective solutions of mixed hardware/software implementations.

The project will be performed in close cooperation with industry, including Saab Dynamics AB, Saab AB and Volvo Technological Development.

- *Incremental Static Scheduling*

Supported with 1 graduate student unit (initially for 2 years).

Project leader: Gerhard Fohler, Computer Engineering, Mälardalen university

This is a project performed at the Centre for Real-Time Systems at Mälardalen university college. The goal is to develop methods and interactive tools that allow incremental extension of existing trusted schedules. By this modular construction of schedules, component-based design and design modifications are facilitated, since adding or modifying tasks will not require reconstruction of the entire schedule.

The project is performed in cooperation with Volvo Construction Equipment Components.

5.2 The ARTES Graduate School

The purpose and ambition of the ARTES graduate school is twofold

- To offer a sufficient number of graduate courses nationwide to allow graduate student at all participating nodes to compose individual study programmes that will enable them to complete their graduate courses in 1-2 years, and
- To provide graduate students with industrially relevant and scientifically challenging theses topics, to be worked upon under competent supervision in stimulating academic and/or industrial environments.

It is worth pointing out that an additional important (but secondary) effect of the proposed concentrated graduate education effort will be that the real-time component in undergraduate education will be substantially strengthened. The reason for this is both that the interest, competence and awareness of the area will increase at universities, and that introductory graduate courses are expected to evolve into advanced undergraduate courses.

The graduate school will not be formally organised as a separate entity. Instead it will be built on increased utilisation and coordination of local courses offered, including those created through local graduate schools in related programmes supported by SSF. The obtained leverage will both increase quality in local programmes and improve cost effectiveness.

The Real-Time Graduate Student Network

An important task of the ARTES graduate school is to support interaction among graduate students and provide meeting places (e.g., the summer school) for discussions. To this end a *graduate student network* is formed. This network will be instrumental in strengthening the national real-time community, thereby providing a basis for continued long-term development in the real-time area. The network is built by admitting *Real-Time graduate students*.

A Real-Time graduate student is a PhD or licentiate student at a Swedish university which has applied to and been accepted by ARTES. The thesis subject of a Real-time graduate student is typically computer science, computer engineering, computer systems, industrial control systems, mechatronics, or automatic control. The thesis topic will be assessed by ARTES to have a strong connection to the real-time area. It is also expected that Real-time graduate students shall be well motivated to work in cooperation with industry during the education and have an ambition to work in industry after the education.

A Real-Time graduate student will have the possibility to be financially supported by ARTES by, for example, grants for travel and other purposes, and the possibility to be funded via ARTES-projects. He or she will also have priority admittance to ARTES-courses and other

common activities within ARTES. Thirdly, by being a Real-time graduate student, special supervision support can be arranged in an interdisciplinary way using the ARTES network

Acceptance to be an Real-Time graduate student is only possible if the candidate already has been admitted to a local licentiate or PhD programme at a Swedish university. Applications to and possible exclusions from being a Real-time graduate student within ARTES are handled by the ARTES board.

Real-Time Curricula

Graduate Education in ARTES is distributed on several traditional subjects - like automatic control, computer engineering or computer science. It is important that ARTES in connection with the Graduate School establishes areas of competence for real-time system professionals. This will simplify the recruitment of people in industry and the selection of subject areas/projects for academic research.

Establishing real-time curricula by describing competence areas will give a contributing support for the research and educational plans of graduate students in the real-time area

A final goal is to establish real-time systems as a new subject area at university level.

The ARTES summer school

A special and very important effort in establishing and maintaining the ARTES network will be the annual ARTES *summer school*. The summer school is not only a traditional academic summer school with tutorials given by internationally leading scientists, but it also provides a natural meeting place for academic and industrial nodes, where current and emerging ARTES projects and other cooperations are discussed, together with general discussions/presentations of issues of common interest, such as industrial needs and requirements. Additionally, representatives from related SSF-programmes are invited to discuss cooperation and other issues of mutual interest. The summer school will also provide opportunities for venture capitalists and start-up experts to discuss commercialisations with researchers and industrial partners.

In August 1997 ARTES organised the *Real-Time week* in cooperation with SNART. The programme for this week consisted of a one day tutorial introduction to real-time systems and research by Prof. Jack Stankovic, the *ARTES kick-off* which was a one day presentation/discussion of ARTES, and the two day bi-annual SNART symposium. The RT-week was quite successful, in that close to 200 persons attended (almost 50% from industry).

The first real ARTES summer school will take place outside Stockholm, August 17-21, 1998. The programme includes, in addition to presentations/discussions of the ARTES programme, three tutorial introductions (on reliability theory, formal methods, and real-time and control), together with presentation/discussions of the ARTES projects and PAMP, as well as discussions on industrial problems and issues. Also, representatives of related SSF programmes will be invited to present their programmes and discuss possible cooperations.

Supported courses

ARTES supports graduate courses in the area of real-time systems. The courses should be organised in a form which simplifies national participation. Four courses have been supported so far within ARTES:

1. *Design of Software for Embedded Real-time Control Systems* (KTH), December 1997 - February 1998.

2. *Distributed Real-Time Systems* (HS), September 1997 - December 1997.
3. *Modelling and Analysis of Real-Time Systems* (UU), November 1997 - February 1998
4. *Hardware/Software Co-design* (LiTH), Starts March 25, 1998

In these courses we have used different means to allow participation from geographically distributed nodes. The first course used web-techniques, the second teleconferencing, and the third was concentrated to a few meetings. An evaluation of using these different models for teaching these type of courses is underway.

Current plans for course autumn 1998 include courses on *Multiprocessing* and *Real-Time and Control*.

5.3 The ARTES Mobility Programme

The purpose of the mobility programme is to increase interactions between (industrial and academic) network nodes, as well as with internationally leading research groups. The mobility programme provides grants that facilitate different types of personal mobility, in particular it supports:

- Graduate student mobility
 - Industrial participation in graduate education (“industridoktorander”).
 - Graduate student involvement in industrial projects (“doktorandindustrialister”)
 - Scholarships for graduate education at internationally leading graduate schools
- Senior researcher mobility
 - To industry. Truly industrially relevant academic research requires leading researchers to have good knowledge and understanding of industrial problems and cultures.
 - To internationally leading research groups
- Academic fellowships (adjoint professorships) for senior industrialists.
- Research fellowships, to allow the invitation of
 - Post docs
 - Visiting professors
- Mobility between nodes, to encourage visits at other nodes.
- Travel grants.

ARTES has decided to give priority to mobility between industry and academia, and to give low priority to individual travel grants (since such support is available in most research projects).

Supported activities

This far, support with kSEK 150 for arranging a seminar and mobility in conjunction with a Nutek supported research project at KTH, Chalmers and LTH has been decided.

5.4 Infrastructure

The infrastructure activities aim at providing information and establishing various types of cooperations involving ARTES nodes, in particular the following is supported:

- Information about ARTES activities, which needs to be spread both within the network and externally. This is achieved by a the Web-site (URL: <http://www.docs.uu.se/artes/>), pamphlets, seminars and advertisements. As a special service to industry ARTES will provide a catalogue on current real-time activities and competences, as well as information about students that are about to graduate.
- To allow new nodes to enter the network, and to facilitate interaction with smaller companies, support for the academic part of *feasibility studies* is provided. Such a grant makes an academic available, free of charge for the company, during a shorter period (e.g. a couple of weeks). The feasibility study should typically lead to some concrete cooperation, either within ARTES or in some other form. These grants will motivate new companies to enter the network and substantially facilitate cooperation. This is especially valuable for small and medium sized companies, as they seldom have the resources (or contacts) to consult qualified senior academics/researchers.
- Workshops and conferences, including organisation of an annual ARTES conference. In contrast with the summer school, this conference will have the form of a regular academic conference with paper submissions, reviews etc., but be dedicated to presentations of research results from the ARTES network (and related efforts). All graduate students supported by ARTES are expected to submit papers to this conference. The first ARTES conference is planned for 1999. Additionally, support will be given to workshops on specific topics and international workshops/conferences held in Sweden.
- Establishment of international cooperation. Support will be provided for preparation of cooperation with internationally leading research groups, as well as for initiating international, e.g., within the forthcoming EU Framework 5 programme.

Topical Networks

An important objective of the ARTES network is to extend and strengthen the network itself. This will be done both by extending the size of the network and by forming sub-networks with specific interests in a particular topic.

One such topical sub-network is being formed for the topic *worst-case execution time analysis*. This network currently consists of a research group at DoCS in Uppsala and CUS in Västerås with funding from the Nutek-supported competence center ASTEC, a group at Computer engineering at Chalmers active in the PAMP programme, and the ARTES-supported *integrated control and scheduling* project in Lund. Current plans include to associate leading European and international groups to this topical sub-network.

Additional topical networks, e.g. in the areas of *real-time scheduling* and *real-time and control* are expected to be formed shortly.

6 Milestones

The most important milestones of the ARTES programme are the licentiate and PhD exams that will be completed, as outline in the table below (where A and P denotes ARTES and PAMP, respectively).

	1998		1999		2000		2001		2002		2003		2004		2005		Tot.	
	A	P	A	P	A	P	A	P	A	P	A	P	A	P	A	P	A	P
Admit. stud.	7	6	10	4	10	0	6	0	0	0	0	0	0	0	0	0	33	10
Stud. in pgm.	7	6	17	10	27	9	31	8	26	6	19	5	12	2	5	0	33	10
Lic. level	0	0	0	1	7	4	8	5	8	0	6	0	4	0	0	0	33	10
PhD exams	0	0	0	0	1	0	3	1	5	1	7	3	6	2	6	1	28	8

For the real-time systems area as a whole, we estimate currently approximately 75 graduate students, 20 licentiate degrees and 10 PhD degrees (1998). Corresponding estimates for 2000 (including the effects of ARTES) are 110, 40 and 20. See also the table on page 16.

Note that even though the table above only indicates the number of students that reach the licentiate level, we expect that most students will complete a licentiate exam., and we assume that approx. 80% of the students continue towards a PhD.

Specific technical milestones are given as an integrated part of the plans for the individual projects.

7 Schedule of Events

The below table presents scheduled ARTES-events, including announcements of call for project proposals.

Date	Activity
August 1997	ARTES kick-off meeting in Lund.
November 1997	Deadline for first call for project proposals (CFP).
March 1998	Start of first project.
May 1998	Deadline for first PAMP and second ARTES CFP.
June 1998	Decision related to 1st PAMP and 2nd ARTES CFP.
August 1998	ARTES summer school. Evaluation of first round of graduate courses.
Autumn 1998	Preliminary review/evaluation of ARTES. Graduate course 5-7. Third CFP (second PAMP CFP), including evaluation and decision.
Spring 1999	First ARTES evaluation. First ARTES conference.
Autumn 1999	ARTES summer school (in August). Annual review of project progress. Fourth CFP (third PAMP CFP). First review of ARTES and PAMP.
Spring 2000	Second ARTES conference.
Autumn 2000	ARTES summer school (in August). Annual review of project progress. Fifth CFP (fourth PAMP CFP).
Spring 2001	Second review of ARTES and PAMP. Third ARTES conference.
Autumn 2001	ARTES summer school (in August). Annual review of project progress. Sixth CFP.
Spring 2002	Fourth ARTES conference.
Autumn 2002	ARTES summer school (in August). Annual review of project progress. Seventh CFP. Third review of ARTES and PAMP.

8 Review and Evaluation

To assure and improve quality the ARTES programme, as well as individual ARTES activities, should be evaluated regularly. The following evaluation are planned:

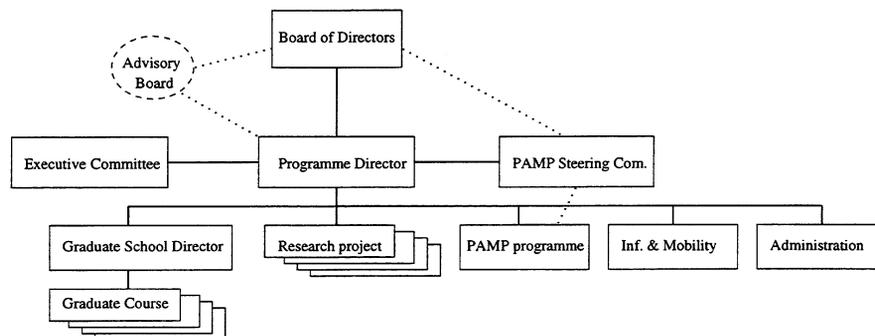
1. A bi-annual evaluation of the entire programme, performed by external experts in cooperation with SSF. A first such evaluation is planned for in January 1999.
2. The research projects will be evaluated as outlined in Section 5.1 and Appendix A.
3. The Graduate School, Mobility Programme and Infrastructure Support is, in addition to bi-annual external evaluation, annually evaluated by the ARTES board.

9 Organisation

ARTES has a formal organisation with a programme board etc., but more importantly an informal network of academic and industrial nodes which jointly are engaged in ARTES-projects and other ARTES-activities, as well as other forms of cooperation. The mission of the formal ARTES organisation is to support and encourage the informal network to focus their efforts on real-time systems, thereby reaching the goals of ARTES and striving towards the ARTES-vision. The following sections present the formal ARTES organisation as well as the informal network.

9.1 The formal ARTES organisation

The ARTES programme is organised in affiliation with Uppsala University as the formal receiver of the SSF funds. A board has the overall authority and responsibility of the programme, while day-to-day operation is delegated to an executive programme director.



Board of directors

The Vice-Chancellor at Uppsala University has in December 1997, after consultation with SSF, appointed the following *board of directors* with responsibility for, and decisive power over, the ARTES programme:

- Bengt Asker, Chairman of the board
- Bertil Emmertz, ABB Industrial Systems

- Kerstin Paulsson, Kockums
- Anders Romare, Volvo
- Jan Torin, Chalmers University of Technology
- Karl Johan Åström, Lund Institute of Technology
- Lars Österberg, ENEA OSE Systems

The task of the board is to

- decide on policies, research programme and activities of ARTES,
- propose *programme director*, appoint *advisory board* and *graduate school director*,
- report annually to SSF about the progress of the programme,
- approve the annual budget of the programme, and
- review the activities and organisation of the programme and take necessary actions.

Programme Director

The task of the *Programme Director* is to

- be responsible for building and maintaining the ARTES network,
- be responsible for marketing the ARTES network, externally as well as internally,
- be responsible for the administration of the programme,
- propose policies and activities to be supported to the board, i.e. it is the responsibility of the director to propose projects, programmes and activities to the board of directors, that will be responsible for making the final decision.
- coordinate and supervise the programme activities,
- review the quality of on-going work and submit material for review.

In February 1998 Hans Hansson (Uppsala University and Mälardalens högskola) was appointed programme director by Uppsala University (in consultation with SSF and by recommendation from the board of directors). However, Hansson has been acting programme director since July 1997. The current engagement (July'97–December'98) is on a 50% activity level.

To assist the programme director in running the programme, the board has appointed an executive committee and a graduate school director. In addition, to handle the associated PAMP programme a steering committee has been appointed.

Executive Committee

The main task of the executive committee is to assist the programme director in planning, preparing proposals and evaluating activities. Members of the ARTES *executive committee* are:

- Bengt Asker, Chairman of the board
- Hans Hansson, Programme Director

- Jan Torin, Chalmers University of Technology
- Anders Törne, Linköping University, Graduate School Director
- Jan Wikander, KTH

Graduate School Director

The board has appointed Anders Törne, Linköping University, as *Graduate School Director*. As such he is responsible for the ARTES graduate school. This includes to

- propose curricula,
- propose and evaluate ARTES courses,
- monitor and evaluate the progress in terms of course work of RT graduate students,
- plan and prepare the Summer school together with the Programme director.

PAMP management

To assist the programme director and board in the management of the associated PAMP programme a steering committee has been appointed. Member of that committee are:

- Bertil Emmertz, chair, representative of the ARTES board.
- Hans Hansson, ARTES programme director.
- Håkan Millroth, PAMP industrial coordinator, Ericsson Telecom.
- Per Stenström, PAMP coordinator, Chalmers.

PAMP will be handled as a relatively independent set of projects managed by the PAMP coordinator. The graduate courses, mobility and infrastructure will however be handled in conjunction with the corresponding ARTES activities.

The main tasks of the PAMP steering committee are

- to be responsible for liaison between PAMP and ARTES,
- to review plans and progress of PAMP projects and activities, and
- to help in disseminating the results from the projects and give scientific advice to the project participants.

To carry out these task, the individual responsibilities of the steering committee are as follows:

The representative of the ARTES board is responsible for reviewing that the projects in PAMP fulfill the general goals of PAMP and ARTES as detailed in this programme plan (including the PAMP programme plan in Appendix D. In particular, an annual progress review will take place to see that project progress coincides with the initial project intentions according to the milestones set out in the project plans. The industrial coordinator is responsible for monitoring the projects so that the industrial goals of the project are fulfilled as specified by the individual research plans. Finally, the coordinator is responsible for monitoring that the scientific goals are fulfilled in terms of quality of the work, Ph.D. student progress etc.

Advisory Board

The initial ARTES application proposed an *Advisory Board* consisting of internationally reputable scientist and experienced (national or international) representatives from industry. The main tasks of this advisory board should be to

- review the scientific quality and industrial relevance of the research and graduate education supported by ARTES, and
- in general be instrumental in the development of ARTES, e.g. by recommending activities to be supported (or cancelled).

A permanent advisory board has not been formed. Instead ARTES has (and will) engage internationally reputable scientist temporarily when need arise (e.g. for evaluation of project proposals, projects and the programme). This far ARTES has used the expertise of Prof. Alan Burns (Univ. of York) and Prof. John Stankovic (Univ. of Virginia). It should also be noted that the members of the board of directors have substantial scientific and industrial experiences.

Administration

It has been decided to have a minimal administration. Initially (during 1997-98), the programme director has a 50% engagement, with a 5% support, mainly to handle payments and book-keeping. The balance between administrative support and programme director may change in the future, but this will not affect the total cost for the administration.

9.2 The informal ARTES network

The informal ARTES network consists of industrial and academic groups with documented interest in the real-time area. For each such group (or *node* in ARTES terminology) a contact person is identified. The contact persons receives information from ARTES, and are expected to marketing the network into their organisations, as well as being active partners in the development of the network, i.e., by providing input to the planning of ARTES and proposing activities to be supported. ARTES is flexible and open for new nodes to enter the network, thus providing a dynamic structure that can evolve and adapt to meet the demands of its environment.

A presentation of the current academic ARTES nodes are provided in Appendix B. The below table presents the current and estimated number of graduate students in the real-time systems area (including ARTES and PAMP) based on the presentations in Appendix B.

	1998		1999		2000		2001		2002	
	A	P	A	P	A	P	A	P	A	P
Stud. in area	75		90		110		130		150	
Stud. in pgm.	7	6	17	10	27	9	31	8	26	6
Lic. in area	20		30		40		50		60	
Lic. in pgm.	0	0	0	1	7	4	8	5	8	0
PhD in area	10		15		20		25		30	
PhD in pgm.	0	0	0	0	1	0	3	1	5	1

The following are the current industrial ARTES nodes.

Industrial Node	City	Contact person
ABB Corporate Research	Västerås	Bo Johansson
ABB Industrial Systems	Västerås	Bertil Emmertz
Alfa Laval Automation	Malmö	Mikael Meyer
Arcticus Systems AB	Järfälla	Kurt-Lennart Lundbäck
Berifors AB	Bromma	Lena Sundsvik
Carlstedt Research & Technology	Göteborg	Jesper Vasell
Combitech Software	Stockholm etc.	Johan Hellqvist
DDA Consulting	Malmö	Ola Dahl
Diagnostics, Pharmacia AB	Uppsala	Kjell Rosengren
Enea OSE Systems AB	Täby	Lars Osterberg
Enator Mälardalen AB	Västerås	Mikael Gustafsson
Ericsson (UAB etc.)	Stockholm etc.	Anders Nyman
Exallon Systems	Malmö	Göran Lindh
IAR Systems	Uppsala	Olle Landström
Industrilogik	Stockholm	Göran Anger
Kockums Submarine Systems	Malmö	Kerstin Paulsson
Lawson Förlag & Konsult AB	Lidingö	Harold Lawson
Mecel AB	Göteborg	Mikael Strömberg
Saab	Linköping	Dag Folkesson
Saab Dynamics	Linköping	Lars-Åke Classon
Volvo Construction Equip. Components	Eskilstuna	Nils Erik Bänkestad
Volvo TU	Göteborg	Peter Lidén

10 Financial Plan and Budget

The plan and budget assumes that SSF decides to increase the level of funding to MSEK 10 in 1999 and to MSEK 15 in 2000.

Costs

The costs in the budgets have been calculated as follows:

Projects Projects are supported in graduate student units, each unit intended to cover the costs for one full-time graduate student, 20% supervisor, travel, basic equipment and overheads. The current amount for one unit is kSEK 500/yr.

In addition to the direct support from ARTES, we assume industrial involvement and indirect support from universities. Industry is assumed to cover its own project costs and salary to senior academics on industrial sabbaticals, as well as some support to thesis projects. Our contacts with industry in the currently planned and decided ARTES and PAMP projects indicate that the value of the industrial support is at least 60% of the project costs in the below table, in many cases substantially more. To get an estimate of the indirect support provided by the universities we assume that the total annual cost for a graduate student (including all overheads, but excluding supervision) is kSEK 625, and for a supervisor kSEK 900. This gives a total annual cost for the above graduate student unit of kSEK 805, hence the indirect support can be estimated to kSEK 305/yr, i.e. 61% of the project costs.

In calculating project costs we assume that an average admitted students will be funded by the programme from April 1st (however for 1998 we assume that all PAMP students

are funded from July 1st). Also, we assume that the average funding level is 80% (with 20% departmental duties), and that 20% of the Licentiates will start industrial careers (not continuing towards PhDs).

Graduate school The annual cost for the summer school is assumed to be kSEK 350 for ARTES. This covers costs for inviting international experts, accomodation and facilities for real-time students and senior researchers, etc. External and industrial participants are expected to cover their own costs.

The remaining graduate school costs are related to graduate courses. This far, support of kSEK 50 for nationally offering courses that otherwise would be given as local courses has been provided. In coming years, support will additionally be given to course development and to the invitation of internationally leading scientists to give courses.

Mobility We assume support in units of kSEK 25 to cover additional costs for mobility between nodes, and support in units of kSEK 100 for invitations of guest researchers. (E.g. for 2000 the budget assumes 4 guest researchers and 16 units of node-mobility support.)

Infrastructure We assume for ARTES that the initial annual cost of kSEK 120 for information and advertisements will increase to kSEK 200 in 2000. Support to workshops are in units of kSEK 50, and feasibility studies in units of kSEK 50. (E.g. for 2000 the budget assumes 6 workshops and 5 feasibility studies.)

Administration The administrative costs include a programme director with a 50% activity level, 5% administrative support, costs for boards, and a limited support to the graduate school director. (E.g. for 2000 the budget assumes kSEK 600 for programme director and administration, kSEK 300 for board and scientific advisors, kSEK 100 for PAMP management, and kSEK 80 for graduate school director.)

It should be noted that the reason for the relative high administrative cost for 1998 is that costs from the actual start of the programme in July 1997 are included.

(MSEK)	1998		1999		2000		2001		2002		Total	
Funding	A	P	A	P	A	P	A	P	A	P	A	P
SSF	5.0	2.0	10.0	4.0	15.0	5.0	15.0	5.0	15.0	5.0	60.0	21.0

Costs	A	P	A	P	A	P	A	P	A	P	A	P
Projects	2.1	1.2	5.8	3.6	9.8	3.7	11.6	3.4	10.4	2.6	39.7	14.5
Grad. School	0.8		1.1		1.4		1.9		1.9		7.1	
Mobility	0.4		0.7		0.8		0.8		0.8		3.5	
Infrastructure	0.4		0.6		0.7		0.8		0.8		3.3	
Adm.	1.5		1.1		1.1		1.1		1.1		5.9	
VAT (8%)	0.6		1.1		1.6		1.6		1.6		6.5	
Total	7.0		14.0		19.1		21.2		19.2		80.5	
Funding-Costs	0.0		0.0		0.9		-1.2		0.8		0.5	

11 Collaborations

The ARTES network includes almost all Swedish academic groups active in the real-time systems area. Hence, ARTES will in itself be the main organisation for national research cooperation. It will also be natural for ARTES to cooperate with related programmes and organisations, since many of the leading individuals in these efforts are also involved in ARTES. In particular ,

ARTES will have strong ties with the SSF programmes Autonomous Systems, VISIT, and HMI programmes, as well as with SICS and the NUTEK Real-Time/Complex Systems Programme.

Close contacts with industry will be ensured by the participating industrial nodes, the industrially dominated board, and the many existing industrial contacts at academic nodes.

The participating nodes are involved in many international projects, have extensive international contacts and collaborations. Through the ARTES mobility programme and infrastructure support these contacts will be further developed and made available to the entire network.

12 Other Plans and Policies

12.1 Continuation of successful programme

The ARTES network is creating an infrastructure for cooperation between academia and industry. The support from SSF is needed to establish the network and demonstrate its value. The graduate education and establishment of a strong national real-time systems research community will most likely lead to continued efforts in the area. Thus, if ARTES is successful, it should after the initial 5-10 years funding period be possible to finance many of its activities via fees and support from other funding agencies.

12.2 Future careers of young researchers

The ARTES graduate education is motivated by industrial needs (in particular a need of qualified personnel), and the majority of thesis projects will be performed in close industrial cooperation. Thus, the licentiates and PhDs will play important roles in advanced industrial research and development projects. The close industrial ties will be maintained, not only via industrially motivated projects, but also via periods of practical industrial training. Traditional academic careers will of course also be open for ARTES students. But the ambition is clear: at least 80% of the students should start an industrial career.

12.3 Recruitment of female students

This is an important but difficult issue, since the vast majority of undergraduate students in the area are males. Special efforts are needed to increase the number of female graduate students in the area.

ARTES will form a network of female graduate students, industrial engineers and researchers with the purpose of giving visibility to positive role models, facilitating recruitment, and providing support and encouragement to admitted female graduate students. Possible activities for this network include:

- A mentor-programme, where female engineers/scientists mentor female undergraduate students and try to get them interested in real-time systems and graduate education.
- Workshop and meetings to exchange views and ideas.

12.4 Intellectual property and exploitation of results

Due to the high ambitions and applied nature of many ARTES projects it is very likely that the research will result in commercially interesting methods and prototypes. Hence, it is important that the ARTES programme has clear rules for intellectual property rights. Furthermore, since

ARTES projects typically are performed in conjunction with related projects funded from other sources, it is important that these rules are compatible with corresponding rules for related programmes.

In general the intellectual property rights will follow those of the participating universities. For specific industrial collaboration a case by case agreement will be reached with the involved companies. It is in such cases important that these contracts do not violate any general rule at the participating universities.

Furthermore, since the handling of property rights is both a difficult matter and very important for the motivation and possibility to start spin-off companies, ARTES encourages SSF to make a serious effort to develop general guidelines and/or rules for how to handle this matter. Especially since we see it as advantageous, even mandatory, that all programmes have the same IPR rules.

Finally, concerning exploitation of results, ARTES has the ambition to be instrumental in the establishment of new industries. For instance, based on the yearly review of ongoing research, ARTES will identify research results which are commercially promising, and then assist in establishing contacts between the researchers and the appropriate financiers, industrialists and entrepreneurs to facilitate the creation of new ventures in the real-time systems area.

12.5 Conditions for discontinuation of the programme

ARTES should be discontinued if it after an initial five year funding period has not had a fundamental impact on industry-academia cooperation and graduate education. It is the responsibility of the ARTES board of directors to monitor the development of the programme and to report to SSF, as well as recommend any necessary action.

If ARTES is discontinued, negotiations with participating organisations are needed to find solutions that make it possible for successful parts of the programme to continue, and such that the involved students can finish their studies.