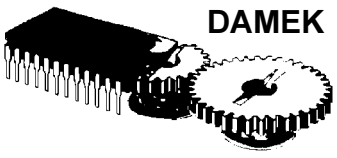
TRITA-MMK {2001:3}
ISSN 1400-1179
ISRN KTH/MMK--{01/3}--SE

Modelling and Simulation of Embedded Computer Control Systems: Problem Formulation

by

Martin Törngren, Jad Elkhoury, Martin Sanfridson and Ola Redell



Stockholm
2001

Technical Report
Mechatronics Lab
Department of Machine Design
Royal Institute of Technology, KTH
S-100 44 Stockholm Sweden

Mechatronics Lab
Department of Machine Design
Royal Institute of Technology
S-100 44 Stockholm, Sweden

TRITA-MMK {2001:3}
ISSN 1400-1179
ISRN KTH/MMK/R--{01/3}--SE

Document type
Technical report

Date
2001-01-09

Author(s)
Martin Törngren, Jad Elkhoury, Martin Sanfridson and Ola Redell

Supervisor(s)
Martin Törngren and Jan Wikander

Sponsor(s)
Include ARTES and NUTEK

Title

Modelling and Simulation of Embedded Computer Control Systems: Problem Formulation

Abstract

Modern machinery, such as automobiles, trains and aircraft, are equipped with embedded distributed computer control systems where the software implemented functionality is steadily increasing. The development of such systems, however, is still a relatively new and young discipline. There is consequently a lack of tools, methods and models to support, in particular, the early architectural design stages.

Closely associated with the design of a distributed control system are design decisions on the overall system structure, function triggering and synchronization, policies for scheduling, communication and error handling. These parameters to a large extent determine the resulting system timing and dependability, and hence, impact the control application performance and robustness.

The primary target for the research described here is the development of executable models to support architectural design. The report describes earlier efforts in this direction including case studies in the DICOSMOS and AIDA projects. Experiences from the studies are discussed including aspects on modelling to support interdisciplinary design, modeling abstraction and accuracy, and tool implementation.

A state of the art survey of related modelling efforts reveals that very little appears to have been done in terms of modelling and simulation that involves simulation of the environment with the computer control system. In addition, the treatment of distributed computer systems, redundant systems, and error scenarios in this context appear to be scarce.

The possibilities for extending the simulation models to arrive at a ready to use library to support the design of distributed control systems are discussed. This work is already under way. Remaining work includes to verify the developed simulation models, and to evaluate the models and their usage in case studies. One related topic for further work includes integrating the simulation models with the AIDA modeling framework. A longer term aim is that the models and simulation features should form part of a larger toolset supporting also earlier and later development stages, thus providing more comprehensive analysis and synthesis support.

Keywords
modelling, simulation, embedded systems, distributed systems, real-time, control systems, performance, timing, networks, scheduling, architectural design

Language
English

1. INTRODUCTION

1.1. Background: embedded control systems, trends and challenges

Machinery, such as automobiles, trains and aircraft, are increasingly being equipped with embedded control systems that are based on distributed computer systems. In these systems the functionality is steadily increasing due to the possibilities enabled by software and networks for information exchange. The computer system is typically composed of a number of networks that connect the different nodes of the system, and where the networks are interconnected through gateways. One example of such a system is given by current automotive systems, that include a high speed network (based on the controller area network) for connecting engine, transmission and brake related nodes; and one network for connecting other "body electronics functions", from instrument panels to alarms. Often, a separate network is available for diagnostics. A node typically includes sensor and actuator interfaces, a microcontroller, and a communication interface to the broadcast bus. From a control function perspective, the vehicle can be said to be controlled by a hierarchical system, where subfunctions interact with each other and through the vehicle dynamics to provide the desired control performance. The subfunctions are implemented in various nodes of the vehicle, but not always in a top-down fashion, because the development is strongly governed by aspects such as the organizational structure (Internal organisation, system integrators and subcontractors).

Enabled by the dramatic evolution of electronics and software, a mechatronics perspective provides a large number of opportunities to create improved machinery by the aid of software, electronics, sensors, actuators and control. At the same, a number of challenges face the developers of machinery in order to really be able to benefit from the technology advances:

- *The need to cope with the dramatic increase in system (software) complexity*. There are several reasons for the increasing amount of software implemented functionality in machine control applications. Taking the automotive industry as an example, complexity "drivers" include legislation (e.g. exhaust regulations) requiring advanced engine control, the possibility to implement functions for active safety such as vehicle dynamics control, the possibility to include new functions for coordination of previously stand-alone systems, and the inclusion of diagnostics functions to improve vehicle maintenance as well as for providing diagnostics of the new functions and their infrastructure. Apart from the sheer amount of new functionality, complexity also arises from different types of interference between functions partly arising from their usage of shared computer system resources.
- *Managing and exploiting multidisciplinarity*. The development of embedded control systems requires knowledge in, and cooperation between, several different scientific and engineering disciplines. There is a great challenge in exploiting the possibilities opened up by a mechatronics perspective; solution space exploration both in theory and in practice should in the best case be able to find solutions and techniques that exploit a synergetic combination of, for example, automatic control and computer system infrastructure.
- *Verification and validation of dependability requirements*. The main challenge here is that of developing mechatronic (software based) systems that are safer and more reliable than their mechanical counterparts. Albeit the advantages, software easily becomes very complex and is difficult to verify and validate. In addition, the distributed computer systems where the software is implemented add "new" failure modes.

1.2. Report aim: modelling and simulation in the context of architectural design

The development of embedded computer control systems is still a relatively new and young discipline. Consequently, there is a lack of tools, methods and models to support, in particular, the early so called architectural design stages. The primary target for the research described here is the development of models to support architectural design, building upon earlier work in the DICOSMOS (2001) and AIDA (2001) projects. In the context of designing a distributed computer control system, architectural design is here used to refer to decisions on the

- overall system structure, including both functional, software and computer structure (these issues include deciding on allocation and partitioning)
- function triggering, synchronization, and policies for scheduling all resources
- communication principles
- policies for error handling, and the mechanisms used for error detection.

Choices of these “design parameters” to a large extent determine the resulting system reliability, safety, and timing, and impact the control application performance and robustness.

The very basic idea is to develop models that can be used to analyse different architectural proposals, simulation is the analysis approach taken in this work. In a longer term perspective, the aim is that the models and simulation features should form part of a larger toolset being developed at the mechatronics lab; a toolset that supports the design of embedded control systems in order the meet the main identified challenges: complexity, multidisciplinarity and dependability.

Some of the requirements on the models are as follows (the reader is referred to Törngren (1995) and Redell (1998) for more detail and background):

- The models should allow both functionality, to be implemented in a computer system, and the computer systems to be represented. Given the challenges described in section 1.1, the functionality must encompass both time- and event-triggered algorithms, as typified by discrete-time control and finite state machines (hybrid systems). The computer system models must represent the basic mechanisms and algorithms that affect the overall system timing behavior.
- The models should allow co-simulation of functionality, as implemented in a computer system, together with the controlled continuous time processes and the behavior of the computer system.
- The models should support interdisciplinary design, thus taking into account different supporting methods, modelling views, abstractions and accuracy, as required by control, system and computer engineers.
- Preferably, the models should be useful also for a descriptive framework, visualising different aspects of the system, as well as being useful for other types of analysis such as scheduling analysis.

1.3. Why model based architectural design

By model based design we primarily refer to models that are sufficiently formal to allow analysis to be carried out. Early architectural analysis allows the solution space to be explored, and at the same time provides a better opportunity for the early detection of erroneous requirements and design bugs. Clearly, such mistakes are very costly if detected late. Compared to traditional testing, the approach allows different failure modes to be approached and analysed one at a time, progressing the verification through the development in an iterative and incremental fashion.

1.4. Layout of the report:

Chapter two summarizes accomplishments so far in this research area in projects where researchers from the mechatronics lab have been involved. Chapter three gives an overview of related efforts in the research community. Chapter four discusses some basic issues and experiences that are important for modelling and simulation. Chapter five discusses different approaches for extending the work carried out so far. Finally, chapter six concludes by proposing avenues for further work.

2. ACCOMPLISHMENTS IN MODELLING AND SIMULATION AT THE MECHATRONICS LAB

2.1. Early work on modelling and simulation in the DICOSMOS project

Traditionally, when approaching the real-time implementation of a controller, fairly little is done to analyse of the effects of the computer system. It is common to add a computational delay in the control system analysis; often the delay is set to equal the period of the controller. It is also common to investigate the maximum delay in the feedback loop that can be tolerated from the point of view of the control system performance/stability. This delay then has to include all the delays added by a real system, including process delays, delays in sensors and filters, delays in actuators, and the delays introduced by the computer system.

With the increase of more complex computer control systems, one should not expect the implementation of a feedback control system to behave exactly as the pure functional design. This is because a distributed computer system introduces delays, jitter, potential data-loss in buffers, and various other potential “failures” such as transient loss of communication packets. An early investigation of these “timing problems” was carried out in the DICOSMOS project, see Wittenmark et al. (1995), Törngren (1995), Nilsson (1996). Here the effects of time-varying

feedback delays, sampling period jitter and data loss in feedback control systems were investigated. A number of aspects influence the actual performance degradation, such as where in the feedback loop the “timing problem” is introduced (i.e. depending on the allocation and partitioning of the control system) and the actual control design (e.g. the bandwidth of the closed loop system, whether compensation is included for a constant delay or not, etc.).

Inspired by earlier work along the lines of Ray (1988), cosimulation was one approach taken in DICOSMOS towards analysing the timing problems. Special Simulink (www.mathworks.com) blocks were developed to model time-varying delays, sampling instant jitter, vacant sampling and sample rejection (data-loss through over-writing of data), Törngren (1995). A feedback control system, as modelled in Simulink, could then be instrumented with these blocks to come one step towards modelling a distributed computer implementation. The approach turned out to be useful for illustrating the effects of the timing problems and for analysing them, e.g. for comparing the effects of sampling period jitter vs. a time varying delay, or for analysing the sensitivity of a particular control design.

2.2. The AIDA modelling framework

The project *Automatic control in distributed applications* (AIDA) was started with the aim of developing a modelling framework and a tool-set targeting analysis and synthesis for control applications to be implemented in distributed computer systems. The tool-set was envisioned as a real-time design complement to existing control and software engineering tools. This report is written as part of a continued research effort that involves the AIDA2 and DICOSMOS2 research projects; and here these initial ideas of the AIDA project are being pursued. Related work also included developing design guidelines for control system decentralization including some of the architectural design issues discussed in section 1.2, Törngren and Wikander (1996).

When starting the AIDA project it relatively soon became clear that there was a lack of models that allowed a system designer (architect) to reason about the functionality, the computer hardware structure, the software structure, and the behaviours of interest (from requirements over designed behaviour to actual behaviour). Therefore, the work from 1996 to 1998 focused on developing the AIDA modelling framework.

The basis for developing the modelling framework was to determine the information needed in the context of the early stages of design of distributed control systems. The models should be able to describe motion control applications implying the need to be able to model time- and event-triggered, multirate, control systems with different modes of operation. These control systems are to be implemented on distributed heterogeneous hardware with both serial and parallel communication links interconnecting the processing elements. There is a need for modelling the various system specific overheads, scheduling policies, error handling etc.

The derived requirements and the AIDA modelling framework are thoroughly described by Redell (1998), Törngren and Redell (2000); a brief description is given here.

The models part of the AIDA modelling framework are divided into three separate parts, *application*, *computer system* and *mechanics.* The application models describe the functional structure, data and control flows as well as timing requirements of the motion control system. Furthermore, the functions and the inter-function communications that constitute the smallest building blocks of the application models are described in detail in terms of resource requirements. In the computer system models, the tasks and their interconnections resulting from partitioning, are described together with the computer hardware. Operating system behaviour, scheduling policies etc. are also defined. The mechanical model describes the sensors, actuators, drive units and other elements that form the interface between the computer system and the controlled process. One attribute of the mechanical models is the physical locations of such interfaces.

2.3. Modelling and simulation of the SMART satellite fault-tolerant computer system

During very early design stages of the development of the distributed computer control system of the SMART satellite, to be launched in 2002 by the European Space Agency, the Swedish Space Corporation needed to analyse and verify the use of the Controller Area Network (CAN) in the on-board satellite computer control system. In particular the error handling of CAN in conjunction with the chosen design for a fault tolerant network was of concern. This work was partly carried out by the authors of this report, and included assessing the design by means of modelling and simulation. A brief description of the SMART system is given in the following. Some information on the simulation models is also discussed in section 4. For more information on the simulation models the reader is referred to Törngren and Fredriksson (1999).

The development of a satellite computer control system has its own very special characteristics. The cosmic

radiation is one prominent factor, requiring special electronics to be used with additional error detection and correction circuitry. This harsh environment has the effect that transient and permanent errors in the electronics, e.g. CPU malfunction, are considered as very normal scenarios that must be accounted for. Another aspect of the satellite design through the European Space Agency is that it is a political effort, determining where (country and companies) the design tasks are allocated. This in essence means that the control system modularization and fixed allocation come with the design task.

The SMART satellite on its way to the moon carries a number of scientific experiments, the main one being the electric propulsion engine. The on-board computer system is divided into two parts; the satellite control system including engine control, energy supply through solar panels, the attitude and orbit control, ground communication, and the payload system dealing with the other experiments.

Basically, all major components of the satellite correspond to nodes of the distributed computer system (engine, telemetry and telecommand, power control, sensor and actuator subsystems including solar panels, star tracker, gyros and reaction wheels). All in all, the SMART system consists of tens of node units that communicate with each other via CAN. In addition there is a main controller node, the space craft controller, which coordinates all the nodes and also the bus (master/slave access). In addition, the main controller node links the payload system with the satellite controls. The main part of the attitude and orbit control function is allocated to the space craft controller; the feedback loop is distributed since the sensors and actuators have their own nodes. In the distributed computer system basically all components have a redundant spare, passive redundancy is employed. Also, two CAN networks exist in the system, the main and a redundant. Each node unit decides autonomously to switch between the two network buses when it does not hear a certain message sent by the master on the network. The rules and timing of when a node is reset or switched to redundant mode by the master, or when a node moves to another bus are defined by the application. The main controller node has also the capabilities to reset any of the other nodes in the system, or switch between the main and redundant node.

An interesting aspect of the CAN design is that a commercial VHDL component is used and implemented in radiation tolerant FPGA's, which are also provided to each node supplier. The VHDL circuit, in addition to CAN, includes logic for error detection and redundancy logic that enables each controller to switch between the redundant networks.

The main aim of the simulation was to verify that the given rules for redundancy are not conflicting, and that the system can recover from a single permanent fault. It was necessary to ensure that the nodes will end up finding each other and recover from the error. It was also important to ensure that the starting up of the system and of a certain node is successful. Together with the Swedish Space Corporation, a list of errors that should be analysed was decided on. These for example included permanent errors on the network such as a broken bus.

The simulation models consequently focused on the computer system and the activities (applications) related to CAN communication and fault-tolerance. This included both activities implemented in software and hardware. To be able to assess and analyse the system behaviour for the considered error scenarios, the simulation models were instrumented to enable the insertion of faults and errors of interest.

The complete system model consisted of some 20 application activities (belonging to the different nodes of the system), the CAN controllers, and the CAN bus. Two CAN bus entities exist, the active and the redundant one. The communication between the activities, the CAN controllers and the CAN bus, is carried out through message buffers. The complete simulation was carried out using time-triggered entities. The utilization of the system is relatively low. The time constants, periods and timeouts, of the application activities are all in the order of seconds. This is quite in contrast to the behavior of the CAN system, where the sending of a frame takes some 100 microseconds and the time for a CAN circuit to change state (from error active to error passive and bus-off when a failure has occurred) is from hundreds of microseconds up to a few milliseconds. The CAN models are further described in section 4.1.

During this work it was necessary to produce the simulation in a very tight time budget. However, due to the lack of ready to use simulation toolboxes and model libraries, it was necessary to develop the models from scratch. A major part of the work included developing and implementing the models, and creating a suitable simulation setup within the chosen tool (Matlab/Simulink).

All in all, the modelling work was rewarding and successful. Three potentially severe failures were detected (Törngren and Fredriksson, 1999), one of which actually referred to the basic operation of the CAN error handling in response to a permanent error. If the receive input of a CAN controller becomes stuck to a logical one during an

ongoing transmission, this faulty CAN controller will perceive an error and start to emit error frames. Unfortunately, the error scenario is in the worst case recurring. This will be the case if a CAN controller is implemented to automatically recover from the bus-off state; the consequence will then be that approximately 28% of the available CAN bandwidth is spoiled!

2.4. Cosimulation within the DICOSMOS2 project

Sanfridson (1999) describes a cosimulation of a truck and semi-trailer using a CAN bus for the on-board distributed control system. The function under investigation is Vehicle Dynamics Control (VDC) which among other things incorporates yaw control. VDC is intended to help the driver in difficult situations and is only active for a short time interval. The force vectors under the tires of the truck should have a magnitude and direction which prevents skidding and jack-knifing in a situation where the driver brakes hard to avoid an obstacle. The force vectors are controlled by individual braking on each wheel. In principle, this can also be achieved by steering the front and rear wheels, but this is not done in this case. The dynamic model of the truck and semi-trailer is approximated with a linearized inverted pendulum. The tire model is assumed to be linear. This greatly reduces the level of detail of the dynamic model.

The distributed control system has seven nodes connected to a CAN bus. The CAN bus, with 22 periodic messages, is simulated on a frame level, which is rather detailed. To create high jitter, the bit rate is 250 kbit/s, the control period is 80 ms and the utilization is very high. The time triggered sensors and the controller are hosted in one node and the event triggered actuators in other nodes. The control signal is "sent" on the bus and becomes delayed, thus causing jitter in the feedback delay. The delay jitter can also result in vacant sampling.

The simulation shows that the delay jitter on the bus has a negative impact on the control of the vehicle. The control performance also gets worse if the constant part of the transmission delay increases.

Sanfridson (2000) used co-simulation to investigate the performance of control applications in order to adjust control periods and message priorities on-line. Two different adjustment schemes are compared. The adjustment is based on the control performance with respect to the timely behaviour of the distributed system. This calls for a detailed model of the communication, including e.g. the priority mechanism. Bit stuffing in the protocol is simulated by adding a random number of bits to a frame. In the simulation, a sender outputs its priority to request a transmission. When the bus becomes idle, the receiver with the corresponding priority is told to deliver the message after a simulated transmission delay, during which no other transmission requests can be serviced. A time triggered sensor and an event triggered controller are hosted in the same node, but the event triggered actuator resides in another node. The control output is sent over the CAN bus and exhibits blocking and interference by other messages, which is an important part of the simulation. Three applications, each consisting of a sensor, controller and actuator share a CAN bus, together with an extra node used to insert disturbing messages with higher priorities. The dynamics of the three applications are described with linear transfer functions.

The simulations have been implemented in Matlab/Simulink, which is a simulation package suitable for hybrid systems. The model of the CAN bus is fairly detailed which gives a realistic timely behaviour of the network communication. That was the purpose of the cosimulation. The distribution of delay periods is similar to that of a real CAN bus. The major drawback is the amount of processor time required to carry out the simulation at this detailed level. The simulated CAN controllers are event triggered and when the bus is idle they listen on the bus. This has however not been implemented in an event triggered fashion in the simulation tool; thus the bus must be polled every time step. A typical figure of the complexity of the implementation in Matlab/Simulink is that it takes around 1 millisecond to simulate a time duration of 1 microsecond on a Sun Sparc Ultra 1.

The notion of time scale is important in relation to the aim of the simulation and the detail of the communication protocol. The time scale for the CAN bus is given by the highest baud rate; at one Mbit/s it takes one microsecond to transmit one bit. The longest time it takes to send a message with the maximum length of eight bytes is 135 microseconds. This is to be compared with the dynamics of the application. If the application can be modelled as a linear system with a sampling time which is about 1 to 10 milliseconds a detailed model can be worthwhile. If the dynamics of the application under control allows a longer sampling time, a detailed level becomes less necessary.

Another aspect is the load of the bus. If the bus is lightly loaded the collisions are few and the response time jitter becomes small. It can thus be reasonable to approximate the communication with a constant transmission delay.

3. A BRIEF SURVEY OF STATE OF THE ART

Various simulation tools have been developed to tackle some of the issues discussed in this report. The following is a small survey covering a range of these tools, an earlier complementing survey is given by Redell (1998). In this survey we aim to summarise these tools in order to extract their essence and compare them from different aspects. What is common between them is the fact that they are developed with the aim of simulating real-time computer systems. However, each tool is focused on particular aspects of such systems. After a short description of each, we will try to classify/categorise these tools and highlight differences and similarities between them. This should help identify the important criteria needed for a better simulation environment.

The following is a list of aspects that will help in evaluating the simulation tools:

1. Aim	To test system, performance, scheduling algorithm, etc …
2. Target application	Control systems, general IT, Hard real-time, event/time triggered, etc …
3. Components simulated	Nodes, networks, times of tasks, semaphores, queues, etc ...
4. Scope	Error scenarios, environment, users, etc ...
5. Level of abstraction & accuracy	Simple network delays, time delay to represent tasks, task behaviour included, etc ...
6. Assumptions & Limitations	Scheduler assumed not to consume processor time. Only supports point to point networks, only static scheduling, etc...
7. Representation/language	Graphical vs. Textual models, unique simulation language, etc...
8. Openness/Expandability	Can the user expand the simulator to change or add a scheduler of his own?
9. Designer interface	Can easily build the simulator, need to create simulations from scratch, need to learn a language, etc...
10. User interface	Graphical representation of results, user interaction with system, logging, etc...
11. Interface to other tools	Open design, API, etc...
12. Analysis and design facilities	Such as scheduling analysis, automatic partitioning and allocation, etc...
13. Design process	Simulation part of a design process, suggestion of when/how to perform simulation, Need to translate from current models to the suggested models, etc...

3.1. Brief summary of existing simulation systems

RT/CS Co-design: A Matlab Toolbox for Real-time and Control Systems Co-design (Eker, 1999). (See Table 1 for an overview) This toolbox is developed in the Matlab/Simulink environment for the simulation of real-time control systems. It's main aim is to simulate the real-time kernel of the computer together with the continuous-time environment, allowing the user to explore the timely behaviour of control algorithms, and experiment with more flexible approaches to real-time control systems such as feedback scheduling. Also, various scheduling policies can be evaluated to study their effect on the controller performance.

According to the paper, control systems are developed by two separate teams of engineers, namely control and systems engineers. Good interaction between the two is necessary to yield better results, remove any misunderstandings, as well as open for a more integrated approach.

Typically, the control engineers model the physical plants as well as their control laws using Matlab/Simulink. However, the actual execution of the controllers on the real hardware has been ignored. On the other hand, most task simulators ignore the simulation of the environment around the computer system. This simulator is designed for the simultaneous simulation of plant dynamics, tasks and network, in order to study the effects of the task interaction on the control performance and behaviour.

The tool simply models the relevant timely aspects of the code that are relevant to other tasks and the environment such as input/output actions and resource sharing. The kernel model handles the scheduling of tasks as well as the proper interface to the environment.

In this simulator, the kernel is tick driven. The scheduling policy is implemented in a function, which the user can modify to implement another scheduler. The task model is similar to the live task model in DRTSS. Each task has a set of attributes such as period, deadline and a list of code segments. Some attributes are constant while others are dynamic. The code for a task is divided into several code segments whose execution time is dynamic. Actual

code execution can only occur during two points: At the very beginning or very end of a segment. The simulator supports the use of kernel primitives such as mutual exclusions, events and network communication blocks. However, users have to write their own network protocol. The tool only provides the mechanism for data passing.

From the user point of view, the controllers are implemented as periodic tasks on the computer using M-functions. The user may probe various aspects of the system such as the execution time of a task by using the Simulink output blocks.

	RT/CS Co-design
1	Simulate the computer system, application and the environment. Evaluate various scheduling policies. Co-design the control and computer systems in an integrated tool. Study effect of implementation on the controllers.
2	Real-time control applications.
3	Tasks, semaphores, and possible extension to include network delays
4	The continuous environment is included in the simulation.
5	Tasks divided into finer execution blocks with variable execution delays. Task behaviour is also simulated.
6	Scheduler consumes no processing time. Only periodic tasks are possible. Scheduler is tick-based and hence cannot handle interrupts. Controllers have to be implemented in an M-function.
7	Simulation within the Matlab/Simulink environment with all its capabilities.
8	Source code available. User can develop own scheduling function. Very open through Matlab/Simulink
9	Blocks can be built from the standard Simulink library and C-code can be attached.
10	A drag/drop concept is used to build the whole system. Computer nodes are simply configured for the application. Output can be chosen by user by placing probes in the appropriate places.
11	In principle the same as the interface for Matlab/Simulink
12	-
13	-

Table 1. Overview of the RT/CS co-design tool

DRTSS: A Simulation Framework for Complex Real-time Systems (Storch, 1996). (See Table 2 for an overview) This tool is a simulation framework for discrete-event distributed real-time systems, with support for active resources, networks, sensors, storage units and graphical displays. It is a member of the PERTS family (Liu J. W. ,1993) of prototyping and verification tools. It is a useful tool for exploring the interaction between schedulers at different levels of the hierarchy.

The tool consists of three major components:

- The search controller that handles the parameters for a particular simulation run. The search controller directs the execution engine through the search parameter space, with the aim to determine which combinations of parameters can meet certain criterion. The parameters can be any numeric parameter of tasks, processors or resources.

- The execution engine is the main component of the tool. The engine is based on the Live Task model in which each task represents a unit of work and contains executable code that defines its behaviour. The simulator provides a virtual-time execution environment for the tasks.
- The interface tool, SETI, analyses the results from the engine, and provides a graphical user interface.

In this tool, the user can enter a system design with some timing parameters and perform experiments to study if it meets certain requirements. The user can choose from a provided suite of scheduling algorithms and resource access control algorithms, as well as develop their own.

The primary representative model is the PERTS task/resource graphs, which allow for the representation of hierarchical schedulers in the system. The resource graph is used to describe the hardware and system software, while the task graphs describe the application. Using the Live Task model as an execution model, the timing aspects of tasks, as well as their behaviour are modelled. This facilitates the implementation of schedulers as real tasks, and hence accounts for the overhead of the scheduling algorithm.

	DRTSS
1	Simulation of real-time distributed systems Explore the interaction between schedulers at different levels of the hierarchy.
2	Discrete-event, distributed real-time systems
3	Resources, sensors, networks, storage units.
4	No user nor environment interaction.
5	Can model task behaviour Detailed network design Scheduler overhead modelled
6	Restricted to the PERTS models for representing the system.
7	Based on the PERTS models for representation. Uses the Live Task model for the execution model.
8	Member of the PERTS development tools.
9	Can implement own scheduling algorithm using C code
10	Graphical representation of output results. Design using the PERTS models.
11	-
12	Search Controller finds the combination of parameters to meet certain requirements.
13	Fits well with the PERTS environment.

Table 2. Overview of the DRTSS simulation framework

STRESS - A Simulator for Hard Real-time Systems (Audsley, 1991). (See Table 3 for an overview) The STRESS environment is a collection of tools for the analysis and simulation of hard real-time safety-critical applications. The main aim of the tool is to evaluate various scheduling and resource management algorithms, but can also be used to study the application itself.

STRESS permits the description of hardware architecture, application software and kernel software using a single textual language. The STRESS language can also be used to define one's own scheduler and communication protocols, allowing the evaluation of the kernel itself. The scheduler is assumed to take no execution time.

The tools used are:

- The front-end tool providing a graphical interface to the tools.
- A feasibility analysis tool that also provides information to the simulator.

- The simulator.
- A display tool showing the execution of each task on a graph.

The STRESS simulator can handle periodic, sporadic, and aperiodic tasks interacting through the use of semaphores and mailboxes. Feasibility analysis can also be performed to determine whether a set of tasks will meet their deadlines at run-time. The scheduling approach can be off-line scheduling to run on a cyclic executive, static priority scheduling, or dynamic priority scheduling.

The model is represented using the STRESS language. A system is a collection of nodes connected via network. Nodes contain processors that communicate via mailboxes. Processors contain tasks. A task is represented by [n,m] which is the range of execution units it consumes and the semaphores it uses. Only tasks consume processor time. The system only supports point to point networks with messages passing between them. Message delay is represented by [n, m] to indicate lower and upper bounds and the actual delay is chosen randomly from that range.

	STRESS
1	Evaluate scheduling and resource management algorithms Can be used to study application itself
2	Hard real-time safety critical systems
3	Any type of tasks, semaphores, mailboxes, various schedulers, nodes, point to point network
4	Can't simulate the environment. Can't simulate the system behaviour.
5	Task simply represented by a range of execution times. Network messages represented by a simple time delay.
6	The language used is textual and user must define all the system in that language. Only tasks consume execution time.
7	Single generic language for hardware & software description.
8	-
9	Designer can define own scheduler and communication protocols.
10	Graphical representation of simulation results.
11	-
12	Feasibility analysis tool.
13	-

Table 3. Overview of the STRESS co-design tool

HaRTS: Design and Simulation of Hard Real-time Applications (Zhu, 1994). (See Table 4 for an overview) HaRTS is a graphical environment for the design and simulation of hard real-time applications. It includes tools for design, scheduling, and animated simulation. The simulation tool supports a hierarchical design diagram with both control and data flows, in which a design is broken down into self-contained sub-designs. The scheduling tool also supports a graphical animation for simulating the dynamic behaviour of the application. The design and scheduling tools are not yet integrated.

In the HaRTS design tool, a system is viewed as a set of state machines, called tasks that can be periodic or sporadic. Each state machine is composed of a subset of the system states and an algorithm for the state transformation. During run-time, the tasks are either active or inactive. Only periodic tasks are considered, since its has been shown how sporadic tasks can be transformed into periodic tasks.

As shown in *Figure 1*, the application can be represented as a set of boxes, arrows, operators and associated text

which together define its control flow, data flow and timing constraints. Each box represents a system state transformation function. The Cnt-in triggers the function to execute, and when the execution finishes, the Cnt-out stimulus is generated and triggers other functions. The control Stimuli is generated by either the Control Driving Sources (timers and external events) or by completing the execution of boxes. The data-in and data-out arrows carry the data in and out of the box at the start and end of execution respectively.

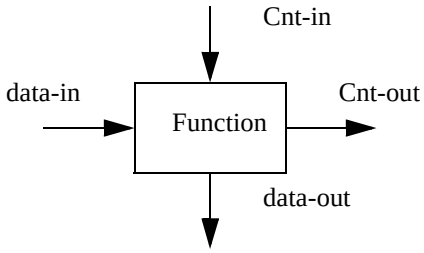


Figure 1. The basic component of an application, representing a function.

The scheduling tool takes in a task graph, schedules it, and then simulates the task execution. Various scheduling algorithms are implemented. These algorithms try to allocate tasks to certain processors in some way and then schedule them. The network messaging is kept simple, and it is assumed that it takes a fixed amount of time to send a network message, and that once a message is ready to be sent, it has the network available to it.

Based on the derived schedule, the task execution is simulated by animating the task graph. Also, the behaviour of the processors and network is simulated in a second interface.

	HaRTS
1	Design and task allocations in a system
2	Hard, real-time systems
3	Tasks, network
4	No user nor environment interaction.
5	Simple network simulation No task behaviour simulated
6	No interaction between tasks. Only periodic tasks are assumed. Very simple presentation of the network
7	System consists of a set of state machines - tasks. Hierarchical design diagram combining control and data flows. A function is represented as a box with input and output data flows. Input and output control flows allow boxes to interact and trigger each other.
8	-
9	User restricted to the provided functionality. Cannot select own scheduler.
10	Interface dynamically shows which task is running at any one time.
11	The simulator and scheduler tools are not yet integrated.
12	Scheduling tools tries to allocate and schedule tasks to processes to meet criterion.
13	-

Table 4. Overview of the HARTS environment

4. DISCUSSION: SOME ESSENTIAL ISSUES IN MODELLING AND SIMULATION

4.1. Interdisciplinary design: modeling purpose, abstraction and accuracy

A model constitutes a simplified representation of a system that is developed with sufficient accuracy for a particular purpose; to be cost-effective too much detail should be avoided. One well known challenge in modelling is to be able to identify the accuracy required for the given purpose. Consider for example the implementation of a data-flow over two processors and a serial network. A huge span of modeling detail is possible, ranging from a simple delay, over discrete-event resource management models (e.g. processor and communication scheduling) to low level behavioral models.

The interdisciplinarity of the design also typically means that engineers of different categories/disciplines will be working with the different aspects of the system. One traditional approach is to have control engineers developing control functions, handing them to software engineers for implementation, in turn cooperating with hardware engineers. Good cooperation and interfaces between the involved engineers is essential to maintain consistency between specifications, design and implementation.

When developing a motion control system for a machine, somehow the functions and elements thereof need to be allocated to the nodes. This principally means that an implementation independent functional design needs to be enhanced with new “system” functions that for example

- perform communication between parts of the control system, now residing on different nodes
- perform scheduling of the computer system processors and networks
- perform additional error detection and handling to cater for new failure modes (e.g. broken network, temporary node failure, etc.)

This mapping will change the timing behavior of the functions due to effects such as delays and jitter. Some reflections related to this are as follows:

- The introduction of “application level effects”, such as delays, jitter and data loss, into a control design, was found to be an appropriate abstraction for control engineering purposes. (as discussed in section 2.1.) Although this is true, the “mapping” to the actual computer system may be non-trivial. For example, delays and jitter can be caused by various combinations of execution, communication, interference and blocking. More accurate computer system models will be required to compare alternative designs (architectures), and to provide estimates of the system behavior. In addition, modelling is a sort of prototyping, and as such important in the design process.
- It is interesting that the underlying model of the computer system could be more or less detailed, given that they provide the right abstraction. For example, if a fairly detailed CAN model has been developed it could still be used in the context of control system simulation given that it is sufficiently efficient to simulate and that its complexity is masked away.
- To be useful for control engineering it is obvious that the models of the computer system need to reflect the real system. In early stages, an architecture only exists on the “drawing board”. As the design proceeds, more and more details will be available; consequently, the models used for analysis must be updated.

An experience with the accuracy of CAN models: In the CAN models mentioned in section 2.3., several simplifications were introduced. For example, the arbitration was not modeled in detail, the precise ordering of frames was not of interest for that analysis since the time delays due to communication scheduling had been investigated separately. The error detection and handling of CAN is quite complex. We early thought of two modelling alternatives:

(1) developing a detailed model including internal behaviors of CAN controllers such as detecting and sending error frames, and managing receive and transmit error counters.

(2) developing a simplified simulation model based on predesigned error scenarios.

The choice was to select the latter alternative, partly because the error scenarios needed to be analysed in detail anyway, and partly because of a tight project deadline. The resulting error scenarios are quite complex, and depend on among other things the CAN protocol, its implementation in CAN controllers, what type of fault that is inserted, and what the node/CAN controller struck by the fault is doing at the time the fault is inserted.

An example of the behavior of a CAN network in the presence of a local permanent error is as follows. Assume

that a CAN controller is transmitting a frame when its transmitter pin gets stuck to a logical zero. This means that all CAN controllers after a minimum of six consecutive bits will detect an erroneous state on the bus. All controllers will then be increasing their receive and transmit error counters according to the CAN protocol. This will continue until the counters have reached the limits that define when a CAN controller will switch mode. Eventually, some nodes will turn in to error passive mode, and after an additional time into bus-off mode. This behavior is deterministic and can be precomputed although one needs to account for a large number of cases. This principle was utilized to analyse all error scenarios following insertion of the predefined faults. Whereas the analyses were relatively complex, the corresponding error scenarios as implemented in the simulation are relatively simple. They involve detecting the error and then signalling the relevant CAN controllers after the precomputed time constants (depending on the type of error and when it is introduced) to switch state.

4.2. Issues related to system development and tool implementation

Our brief survey on modeling and simulation indicates that very few tools provide cosimulation of the environment - the controlled process - with the computer control system. In addition, the treatment of distributed computer systems, redundant systems, and error scenarios in this context appear to be scarce.

While developing distributed control systems, it would be advantageous to have a simulation toolbox or library, in which the user can build the system based on pre-built modules to define things such as the network protocols and the scheduling algorithms. With such a tool, the user can focus on the application details instead. Such a tool is possible since components like different types of schedulers and CAN network are well defined and standardised across applications. In the same way a programmer works on a certain level of abstraction, at which the hardware and operating system details are hidden by the compiler, the simulation tool should give the user a high level of abstraction to develop the application. Such a tool will enforce a boundary between the application and the rest of the system. This will speed up the development process, and gives the developers extra flexibility in developing the application.

To be useful and cost-effective for system development, the usage of the models and the simulation facilities need to be integrated into the development process. For example, the failure mode effects analysis carried out through simulation, described in section 2.3., was based on error and failure models derived from hazard analysis. Support for model reuse is essential to justify the effort spent on model development.

In the SMART project, the developers needed to analyse and verify their application at the architectural level in order to proceed to the next stage. What was interesting was the fact that the system functionality was not yet fully defined at this stage. In addition, several implementation related decisions were not taken at the time of modelling. One example of this is how data, transferred between the applications and the CAN controllers, was to be buffered. Modelling was carried out based on the preliminary design documents. Thus, the details of the simulation models had to be filled in through discussions with the Swedish Space Corporation - discussing alternatives and relevance. From this exercise it became clear that model building for the purpose of simulation, acts as a design review spurred by the need to arrive at executable models.

Simulation time resolution, system time constants and real-time. From the experiences with CAN models and simulation (sections 2.3. and 2.4.) it became clear that a wide range of involved time constants can cause problems in the simulation implementation, in particular with respect to simulation efficiency. For example, in the SMART case, the application time constants of the redundancy logic are in the range of seconds, the time constants of the CAN error scenarios are in the range of ms, while the low-level operation of CAN has a resolution of microseconds, or even less if the bitwise arbitration process should be modelled accurately, considering potentially simultaneous bits. As discussed in sections 2.3. and 2.4., the Controller Area Network is thus a good example where the purpose of the analysis has to be evaluated against the model accuracy/complexity and the needed simulation time. Other important aspects with respect to the simulation time include to what extent the simulation is time- or event-triggered, and the particular way the model is implemented.

Model implementation and simulation correctness in cosimulation. It is clear that the implementation of the types of hybrid systems we are aiming to model, requires a thorough knowledge of the simulation tool. Cosimulation of hybrid systems requires the simulation engine to handle both time-driven and event triggered parts. The former includes sampled subsystems as well as continuous time subsystems, handled by a numerical integration algorithm that can be based on a fixed or varying step size. The latter may involve state machines and other forms

of event-triggered logic.

In the examples given in chapter two, Matlab/Simulink was the tool used for simulation. Primarily so because it is widely used for embedded control systems and supports hybrid systems. The models of the computer system can be implemented in different ways in the tool, for example, sections 2.3. and 2.4. involved time-triggered implementations based on S-functions, and statecharts (finite state machines) with additional C-functions within Simulink.

Some examples of aspects that need consideration for tool implementation are as follows:

- If events are used in the computer system model these must be detected by the simulation engine. How is the event detection mechanism implemented in the simulation tool?
- At which simulation steps are the actions of a stateflow system carried out?
- How can actions be defined to be atomic (carried out during one simulation step)?
- How can pre-emption of simulation be implemented including temporary blocking to model the effects of computer system scheduling?

Another aspect of correctness is that of verifying the models/simulation against a real system. This is a topic for further work.

5. POSSIBILITIES FOR EXTENDING THE CURRENT SIMULATION MODELS TOWARDS DISTRIBUTED REAL-TIME CONTROL SYSTEMS

5.1. Global vs. local clocks/synchronization

Both synchronous and asynchronous systems exist; industrially asynchronous systems still predominate but this may change with the introduction of newer safety critical applications such as steer-by-wire in cars, because of the advantages inherent in distributed systems based on a global clock (Törngren, 1995).

On a microscale, the communication circuits typically are hard synchronized which is required to be able to receive bits and arbitrate properly. In a system with low level synchronization, and/or synchronized clocks, the synchronization could fail in different ways. For asynchronous systems, the clock drift could be of interest to incorporate. Given different clocks with different speeds, this will affect all durations within each node. A conventional way of expressing a duration is simply by a time value; in this case the values could possibly be scaled during the simulation setup. All the above mentioned behaviours could be of interest to model and simulate.

5.2. Node level tasking and functional models.

The most essential characteristic of a distributed computer system is undoubtedly its communication. In early design stages, the distributed and communication aspects are often targeted first; but then node scheduling also becomes interesting and is therefore of high relevance. It is very common that many activities coexist on nodes. Also, these activities typically have different timing requirements and may also be safety critical to different degrees. The scheduling on the nodes affect the distributed system by causing local delays that can influence the behaviour of the overall system.

A node is composed of application activities, system software including a real-time kernel, low level I/O drivers, and hardware functions including the communication interface.

The node tasking model needs to include:

- a definition of tasks, their triggering, and execution times for “execution units”.
- a definition of the interactions between tasks in terms of scheduling, inter-task communication and resource sharing.
- a definition of the real-time kernel and other system software with respect to execution time, blocking, etc.

Some issues in the further development include what type of inter task communication and synchronization that should be supported (for example, signals, mailboxes, semaphores, ...) and whether, and to what extent, there is a need to consider hierarchical and hybrid scheduling. (for example including both processor/interrupt and real-time kernel scheduling levels).

The functional model used in conventional control design should be reusable within the combined function/computer models. This implies that it should be possible to somehow adapt or refine the functional models to in-

corporate a node level tasking model.

5.3. Communication models

The types of communication protocols of interest are confined by the area considered. Nevertheless, a number of different communication protocols are currently being developed in view of future embedded control systems. CAN is currently a de facto standard, but there is also an interest of including the following:

- Time triggered CAN (TTCAN, 2001) referring to CAN systems designed to incorporate clock synchronization suitable for distributed control applications. This rests on the potential of the recent ISO revision of CAN to easier implement clock synchronization in CAN and where the retransmissions can be turned off.
- Properties reflecting state of the art fault-tolerant protocols such as the Time-Triggered Protocol (Kopetz and Grundsteidl, 1993) and FlexRay(FlexRay, 2001). Fault-tolerance mechanisms of these protocols, such as membership management and atomic broadcast, then need to be appropriately modelled.

It is often the case that parts of the protocols are realised in software, for example dealing with message fragmentation, certain error detection and potential retransmissions. Both the execution of the protocol and its scheduling then need to be modelled. The semantics of the communication, and in particular of buffers is another important aspect, compare for example with overwriting and non-consuming semantics vs. different types of buffering. Whether the communication is blocking or not from the point of view of the sender and receiver is also related to the communication semantics.

Another issue is which “low-level” features of communication controllers that need to be taken into account. Compare for example with the associative filtering capability of CAN controllers, and their internal sorting of message buffers scheduled for transmission (relates back to hierarchical scheduling).

5.4. Computer architecture, other aspects

- The topology of nodes. Broadcast networks are of primary interest. Whether networks connected through gateways should be supported is an open issue.
- Fault tolerance. Which principles should be supported? Clearly, for safety critical systems redundancy will be employed (in some cases only for error detection purposes, in some cases also to provide fault-tolerance). These systems could use either passive redundancy, or which perhaps is more common in real-time systems, active redundancy. Managing the redundancy, switches between components, and reintegration then becomes interesting. An open issue is what the basic modeling entities and concepts are to be used for flexibly modeling variants of such systems.
- Single vs. multiprocessor nodes. Single processor nodes are the primary target, since these are most common. Nevertheless, there are of course real-time control systems that are based on multiprocessor systems. Low priority is currently placed on this since it would entail additional modeling of e.g. multiprocessor communication and synchronization. Another facet of this is nodes based on, or including ASIC nodes. One instance here of interest is “intelligent” sensor nodes, that provide the sensor with electronics to communicate. These types of nodes are of interest and were included in the work outlined in the SMART satellite modeling.

5.5. Fault models

The use of fault models is essential for the design of dependable systems. The specific models of interest are very application specific. For inclusion in models and simulation, it is of interest to investigate generic fault models and their ease of implementation. There are a number of available studies on fault models dealing with transient and permanent hardware faults, and to some extent also categorizing design faults, see for example Thane (2000). As always, there is also the issue of insertion in the form of a *fault*, an *error* or a *failure*. Consequently this is a prioritized topic for further work.

6. CONCLUDING DISCUSSION ON RESEARCH TOPICS

A strong advantage of model based design used in the context of early analysis, is the ability to evaluate alternative architectures with very few restrictions. Compared to traditional integration testing, or hardware in-the-loop simulation where the complete environment of a node is simulated in real-time, model based design and analysis

provides additional advantages including the ability to easily and with low cost change the design, and the possibility to instrument and analyse the internals of the distributed computer system. This is not to say that model based design can replace the others, but the tasks of for example the system integrator can be made much easier given that some aspects of the system, such as its timing behavior, have been analysed thoroughly earlier. A strong point of the modelling activity is that it is a very useful process that can reveal a number of aspects, such as missing requirements and design bugs.

The investigation of computer system models, at different levels of abstraction with different accuracy/complexity, is an educative process that we hope can provide additional understanding to support interdisciplinary design. The model and tool prototypes are also very important as part of the research because they enable different forms of feedback.

In the following, some directions for further work and research providing a broader perspective than just modeling and simulation are discussed.

Extending the modeling and simulation prototype, and linking it with design. This is a relatively straightforward continuation based on the work presented in this report and already under way. The developed models are being extended to encompass distributed control systems including local node scheduling. The models and simulation techniques will be verified by comparing with analysis, and also with real systems. Case studies will be carried out to evaluate the chosen modeling approach and how it fits the design process. The types of analysis that can be carried out and its support for different design issues should be investigated. The ambition of the modelling and simulation prototype is illustrated in Table 5.

	Ultimate Simulator
1: Aim	Evaluate system behaviour and performance. Study various implementation alternatives
2: Target	Distributed real-time control applications
3: Components	Network, nodes, schedulers/real-time kernels, inter and intra process communication, ...
4: Scope	Study error scenarios. Simulate system and environment
5: Abstraction	High level of abstraction for the user. Possibility to provide the accuracy required. Lower level abstraction available for system designer.
6: Limitations	
7: Language	Tailored interdisciplinary language. Based on well established languages from the various disciplines.
8: Openness	Framework developed for user to contribute modules in a reusable fashion. Main generic mechanisms developed to build the rest of the system on.
9: Designer interface	Designer provided with basic mechanism to build from. Designer can use language of choice.
10: User interface	User works within their own environment and hence acquire the environment's interface.
11: Interfaces	Tool should have an API to interact with other tools
12: Analysis	Models are extractable and thus reusable for other tools. API should facilitates the interaction with other analysis tools.
13: Design	Simulation part of a design process, suggestion of when/how to perform simulation

Table 5. Features of the “ultimate simulator” being developed at the Mechatronics Lab.

Investigating and extending the AIDA modeling framework. Some further ideas for developing the models are as follows:

- The developed simulation models should be integrated with the AIDA modelling framework. Ideally, the same basic models should be useful for static analysis and for simulation.
- The simulation models do describe both system structure and behavior but they are not always very descriptive since they sometimes lack graphical views. Some proposals in this direction have been made in the AIDA models (Redell, 1998).
- The semantics of pure simulation models (such as the ones in Simulink) inherently differ from the timing behaviour of real-time execution, (Törngren et al., 1997). How can this semantic gap be bridged?
- The simulation models and the AIDA framework need to be extended both upwards, to models used in earlier design stages, and downwards, towards the implementation. Basically, the motivation is given by the need for a holistic design framework that enables models (and work accomplished) to be reused, and used efficiently. Thus it would be desirable to be able to refine the models such that they can be used not just in early design. In addition, work carried out in configuring a computer node for example, should be reusable in later prototyping and implementation stages.
- The use of the Unified Modelling Language (UML, 1997) and CODARTS models (Gomaa, 1993, a development of structured analysis models) in conjunction with the modeling framework will be investigated. Key aspects here include assessing when and why to use object models vs. functional (structured analysis) models, and how they map to other entities such as tasks.

Developing a toolset for architectural analysis. This topic builds upon the ideas of the AIDA project. The idea is to develop a toolset that provides comprehensive analysis and modelling support for embedded control systems.

The following are some ideas for the implementation of a prototype toolset for research purposes:

- The envisioned toolset concept is based on a number of well established engineering tools that are complemented and extended with a number of functionalities. These functionalities include additional models to enable important analysis and synthesis to be carried out. They will also include necessary interfaces between tools. There are two strong reasons for this structure; given limited research efforts, energy should not be spent on reinventing the wheel. In addition, using available tools will make it easier to demonstrate and apply the toolset concept in a realistic setting. It will also facilitate the pinpointing of the opportunities provided by the complementing tools. As an example, there are tools around that can deal with modelling and analysis of reliability, safety, control system design, finite state machines, timing analysis, etc., but these tools are not integrated; and even if they were, can not provide the functionality required for appropriately supporting the design of distributed real-time control systems.
- The functionalities considered include support for overall system structuring in early design stages (where reliability, safety, resource load and costs are evaluated), hazard and safety analysis integrated with control system design, and control design complemented with support for distributed real-time system implementation where timing analysis is one important part.
- Model reuse must be enabled by the envisioned toolset. Reuse can come in different forms. One aspect of this is to be able to use a developed model throughout the life cycle of a product in order to support product maintenance including upgrades. As discussed earlier, this requires models to be refined during the development. Having developed a simulatable model, it would be valuable to reuse this information, for example the computer system configuration, in further prototyping and development work.

Many interesting issues exist and will arise in the development of this type of toolset.This includes how to manage the different types of models and how to use the toolset facilities in system development. The toolset should be complemented by an appropriate design methodology. One important piece of this methodology should be a verification framework that promotes the development of dependable embedded systems.

Extending the functionality of the toolset also requires the models to be extended (as partly discussed in the previous section). For example, the AIDA models currently do not include explicit fault models and attributes such as criticality.

7. ACKNOWLEDGEMENTS

This research is support in part by NUTEK (Complex Technical System Program/System Architecture, the DICOSMOS2 project) and ARTES (the AIDA2 project).

8. REFERENCES

AIDA (2001). http://www.md.kth.se/~ola/aida/index.html

Audsley (1991). Neil Audsley, *STRESS: A Simulator for Hard Real-Time Systems*, RTRG 106, Department of Computer Science, University of York (1991).

DICOSMOS (2001). http://www.damek.kth.se/~mis/dicosmos/index.html

Eker (1999). Johan Eker, A Matlab toolbox for real-time and control systems co-design, *Proc. of 6th International Real-Time Computing Systems and Applications Conference,* 1999 , pp 320-327

FlexRay (2001). http://www.flexray-group.com

Gomaa (1993). Hassan Gomaa. *Software design methods for concurrent and real-time systems*. Addison-Wesley publishing company, 1993.

Kopetz and Grundsteidl (1993). Hermann Kopetz and Gunter Grundsteidl. TTP - a Time-Triggered protocol for fault-tolerant real-time systems. *23rd IEEE International Symposium on Fault-Tolerant Computing*, FTCS-23, 1993.

Liu J. W. (1993). J. W. S. Liu, J. L. Redono, Z. Deng, T. S. Tia, R. Bettati, A. Silberman, M. Storch, R. Ha, and W. K. Shih. PERTS: A prototyping environment for real-time systems. Proceedings of the 14th IEEE Real-Time Systems Symposium, pages 184-188, Raleigh-Durham, North carolina, Dec. 1993.

Nilsson (1996). Johan Nilsson. *Real-Time Control Systems with Delays*. PhD thesis. ISRN LUTFD2/TFRT--1049--SE, Lund Institute of Technology, Sweden.

Ray and Halevi (1988). Asok Ray and Y. Halevi. Integrated Communication and Control Systems: Part II - Design Considerations. *ASME Journal of Dynamic Systems, Measurements and Control,* Vol 110, Dec. 1998, pp 374-381.

Redell (1998). Ola Redell. *Modelling of Distributed Real-Time Control Systems, An Approach for Design and Early Analysis*, Licentiate Thesis, Department of Machine Design, KTH, 1998, TRITA-MMK 1998:9, ISSN 1400-1179, ISRN KTH/MMK--98/9--SE, Stockholm, Sweden.

Sanfridson (1999). Martin Sanfridson. QoS in Distributed Control of Safety-Critical Motion Systems. Work in progress paper at the *20th Real-time Systems Symposium*, Phoenix, December 1999.

Sanfridson (2000). Martin Sanfridson. *Timing problems in distributed control.* Licentiate Thesis, TRITA-MMK 2000:14, ISSN 1400-1179, ISRN KTH/MMK--00/14--SE, May 2000.

Storch (1996). Storch, M.F., DRTSS: a simulation framework for complex real-time systems, *Proc. of Real-Time Technology and Applications Symposium,* 1996 , pp 160-169

Thane (2000). Henrik Thane. *Monitoring, Testing and Debugging of Distributed Real-Time Systems*. Doctoral thesis, Department of Machine Design, KTH, TRITA-MMK 2000:16, ISSN1400-1179, ISRN KTH/MMK/R--00/16--SE, MRTC Report 00/15.

TTCAN (2001). http://www.can.bosch.com/content/TT_CAN.html

Törngren and Wikander (1996). Martin Törngren and Jan Wikander. A Decentralization Methodology for Real-Time Control Applications. *Journal of Control Engineering Practice*, Vol. 4, No 2, pp. 219-228, 1996. Elsevier Science.

Törngren (1995). Martin Törngren. *Modelling and design of distributed real-time control applications*. Doctoral thesis, Department of Machine Design, KTH, TRITA-MMK 1995:7, ISSN1400-1179, ISRN KTH/MMK--95/7--SE.

Törngren and Fredriksson (1999). Martin Törngren and Peter Fredriksson. *SMART-1. CAN and Redundancy logic simulation of the SMART SU*. Swedish Space Corporation, Report S80-1-SRAPP-1.

Törngren and Redell (2000). Martin Törngren and Ola Redell. A Modelling Framework to support the design and analysis of distributed real-time control systems. *Journal of Microprocessors and Microsystems* 24 (2000) 81-93, Elsevier, special issue based on selected papers from the Mechatronics 98 proceedings.

Törngren et al. (1997). Martin Törngren, Christer Eriksson, Kristian Sandström (1997). Real-time issues in the design and implementation of multirate sampled data systems. In Preprints of *SNART 97 -Swedish National Association on Real-Time Systems Conference*, Lund, 21-22 August 1997.

UML (1997). UML notation guide. Version 1.1. Sept. 1997. Object Management Group, doc. no. ad/97-08-05. http://www.rational.com/uml

Wittenmark et al. (1995). Björn Wittenmark, Johan Nilsson, and Martin Törngren. Timing Problems in Real-time Control Systems. P*roceedings of the 1995 American Control Conference*, Seattle, WA, USA.

Zhu (1994). Jiang Zhu, Design and simulation of hard real-time applications, *Proc. of 27th Annual Simulation Symposium,* 1994 , pp 217 -225