A Modelling Framework to Support the Design and Analysis of Distributed Real-Time Control Systems

Martin Törngren and Ola Redell

Mechatronics Lab, Department of Machine Design, Royal Institute of Technology
SE-100 44 Stockholm, Sweden, email: {martin, ola}@damek.kth.se

ABSTRACT: Within the AIDA project a modelling framework has been developed to support design issues related to the implementation of control applications in embedded distributed computer systems. At a relatively high level of abstraction the models describe the structure and timing behaviour of a control application (in terms of functions and operational modes) and its implementation (hardware, operating system threads and resources). The resource description allows the timing behaviour of the implementation to be analysed and fed back into the application models. The models form the basis for a decentralization tool-set, where a first prototype is under development. Examples of the models are given and the framework is compared to related modelling approaches.

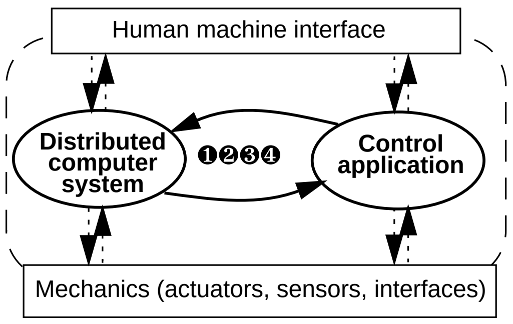
1. INTRODUCTION

Due to the dramatic evolution of electronics, machine industry faces a shift in machine design where more and more functionality is included in the form of software on embedded distributed computer systems. In the first phase of this transition, e.g. in the automotive industry, the new software based solutions were in principle stand-alone systems, [23], in terms of both the computer system and the used set of sensors and actuators. To really benefit from the potential of software based systems, the next phase in this development is that different, previously stand-alone functions, will be interacting and cooperating so as to optimize the overall performance of a system such as in the next generation of cars. Essential, in both an engineering and research context, is the resulting need for interactions between control, mechanical, and software/computer/electronics engineers, [33]. This, in addition, necessitates that new approaches and facilities are provided for system modelling, analysis and tool support.

It often pays off to introduce an embedded distributed computer control system in new machinery primarily because of the reduction in cabling, hydraulic hoses, mechanical linkage etc., but also due to the flexibility inherent in digital solutions allowing e.g. diagnostics and improved control algorithms, [28]. In a machine embedded distributed control system, electronic hardware (microprocessors, ASICs, etc.) and control software is distributed in the machine and connected to sensors and actuators. The topology of the distributed computer system is usually to a large extent given by the structure of the machine with its sensors and actuators.

In the development of such systems, however, designers are faced with several 'new' problems. For example, the design of the mechanics can no longer be isolated from the design of electronic control nodes because the nodes need to be embedded into the machine together with their information and energy cabling.

The design issues depicted in Figure 1 form the primary background for this work.


Human machine interface
Distributed computer system
❶❷❸❹
Control application
Mechanics (actuators, sensors, interfaces)

❶ *Structuring, partitioning* and *allocation* deals with the definition of components, their connections and to the mappings between different structures (e.g. functions/threads/processors).

❷ *Execution and communication policies.* The definition of policies and mechanisms for triggering, scheduling, synchronisation and communication.

❸ *Performance requirements.* The definition of timing requirements and related trade-offs.

❹ *Error detection and handling policies.* In particular additional failure modes of distributed real-time systems must be identified and handled.

Figure 1. Essential design issues for distributed real-time control systems

The horizontal arrows in Figure 1 indicate different design cases and disciplinary interactions. For example the performance requirements of the control application could be used to dictate the choice of the execution policy in the computer system. On the other hand, these policies may be given, thus constraining the solution space and potentially requiring appropriate counter measures to be taken in control design (e.g. including compensation for time-varying delays) or even the requirements to be reformulated.

The design issues of Figure 1 have the following overall characteristics

- The determination of a suitable degree of decentralization of the control system on the distributed computer system depends on a number of sometimes conflicting factors including control performance, reliability, flexibility and cost, forming a multi-objective design problem.
- The issues should be addressed in early design stages since they affect essential architectural properties of the integrated computer control system. For example, the degree of decentralization of the control system is highly related to the dimensioning of the computational and communicational resources of the distributed computer system. Neglecting early performance analysis may lead to costly clashes in system integration. The adoption of early performance analysis, on the other hand, enables the design of cost-effective solutions.

1.1. The AIDA project and the structure of the paper

Considering newer applications with increasing complexity, and with high reliability and safety requirements, there is an increasing need to provide models and tools to support the designers of machine embedded control systems. Although several useful modelling approaches and computer aided tools exist for the design of control, software and mechanical systems, such models and tools provide very little or no support when the applications are implemented in embedded distributed real-time computer systems.

The project *Automatic control in distributed applications* (AIDA) was therefore started with the aim of developing a supporting modelling framework and a tool-set targeting analysis and synthesis related to the design issues of Figure 1, see [2] for more background information on the project. The current status of the tool-set development is that a modelling framework has been developed on which the tool-set will be based. The modelling framework has been used in a few case studies, [28][22]. Based on the existing framework the implementation of a tool prototype has just began.

The AIDA approach is far from a complete development method with supporting models and tools. Instead, the whole concept is devoted towards solving a few concrete design problems that occur when a control application is to be implemented in a distributed computer system. Thus, the AIDA models and related design guidelines, [27][28], can be used to complement existing methodologies when appropriate. Issues such as timing analysis, the provision of graphical views of system structure and timing behaviour should be useful not only in early conceptual design stages but also in work related to the augmentation of already existing systems. The envisioned tool-set could be seen as a real-time design complement to existing control engineering tools.

The interactions between the different design choices of Figure 1 are considerable, and have a great impact on the final system performance. Therefore the AIDA tool-set is intended to support iterative engineering, answering a number of “what if” questions put by the designer. Typical analysis results that the AIDA tool-set should be able to provide are estimates of the timing behaviour of the control system given a particular implementation (allocation, processors, scheduling policies etc.). Example results include estimates of feedback delays, delay variations, jitter in sampling periods, and the utilization of processor and communication links.

Such results can be used to answer a number of different design questions. For example; With fixed hardware and application structures, what performance can be achieved by (i) changing the hardware components, (ii) changing policies for scheduling, clock synchronisation etc. Apart from analysis, synthesis functionality such as (semi) automatic allocation and control system restructuring are also planned to be included.

The purpose of this paper is to give an overview of the results so far, primarily dealing with the modelling framework. Section 1.2. gives a brief overview of related research projects with respect to tool support. Section 2 describes the developed modelling framework including examples of the usage of the models. Section 3 discusses other existing modelling approaches and their relation to the AIDA models. Finally section 4 provides conclusions and discusses further work in the project.

1.2. Related research efforts on tools

State of the art computer aided control and software engineering (CACE and CASE respectively) tools do support rapid prototyping for specific distributed computer systems. The tool-sets provided by the companies Mathworks and dSPACE enable control system design, analysis and simulation, code generation from the graphical block diagrams of Simulink, “transparent” implementation of generated code on DSPs, and real-time operator interaction and logging, [14]. Transparency here refers to the fact that the user manually specifies allocation of Simulink blocks to processors, the physical configuration of which is also described in Simulink. The required communication code is then included transparently. The resulting code is managed by a real-time executive.

While such tools are extremely useful for control designers, there is still a large amount of functionality missing with respect to implementation analysis. This holds for both related control and software engineering tools which do not allow exploration of implementation alternatives, for example with respect to resource structures and execution strategies. Nor do they provide any guarantee that an implementation will work with respect to real-time behaviour.

Out of a number of related research tools we have found the three following to be highly interesting:

- The Development Framework, [4], incorporates some of the functions and a tool structure similar to that we envision. The control design is specified in Simulink and then transferred to a commercial CASE tool (Software through Pictures) which models can be analysed and transformed by other tools (clustering, replication). The combination of CACE and CASE technology facilitates multi disciplinary project development and makes use of the strong points of the respective modelling and analysis features. Hardware modelling, execution and communication strategy considerations appear to be weak points in the Development Framework.
- The GRAPE tool-set, [18], is developed for digital signal processing systems and has an interesting tool structure, including tools for allocation, scheduling and restructuring of the computer hardware. Only point to point architectures are considered.
- An interesting effort is the Parallel Scalable Design Tool-set (PSDT) from Honeywell research. PSDT is a CASE tool specifically addressing implementation in distributed and parallel real-time systems, [5].

In the real-time research community, a number of prototypical tools have been developed for schedule simulation, timing analysis and schedule generation, [10][31][15][3]. The amount of theoretical research work is vast including work on general purpose parallel processing, parallel and distributed real-time systems, and related work in control engineering. One important shortcoming of the supporting models is that they, with a few rare exceptions, do not directly address the timing constraints of sampled data systems, [22][28].

2. THE MODELLING FRAMEWORK

2.1. Requirements on the modelling framework

The overall purpose of the models is to support the design of distributed real-time control systems by providing means for describing system structure and timing behaviour, and by including information to support analysis and synthesis. The ambition has been, and is still, to fill in the gaps in terms of missing models and tools. Commercial tools and research provide good support for analysis and simulation of continuous-time and discrete-event dynamic systems. Lacking support for behaviour modelling with respect to timing behaviour (e.g. precedence, parallelism, triggers, durations etc.) and high-level modelling of resource management is therefore the primary focus.

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. A major challenge in developing the framework concerns the provision of adequate and different abstraction levels taking into account the analysis needs of industrial designers and the modelling precision needed. Consider for example the implementation of a data-flow over two processors and a serial network. A high level of abstraction is to simply model this network as a time-delay with potential data-loss, thus providing an abstraction suitable for control engineers. Increasing detail would take into account essential resource management (e.g. processor and communication scheduling) such that the timing behaviour of system functions can be analysed sufficiently well, providing a suitable high level of abstraction for software and computer engineers.

The following requirements on the models were initially formulated:

- The models must support the multidisciplinary characteristics of the design problems where at least both computer (software and hardware) and control engineering is involved. To do this the modelling framework must provide suitable abstractions, views and links between these.

- The model abstractions should be on a relatively high level to support ‘architectural decisions’ that directly affect the system structure and timing behaviour, and thereby a number of essential system properties such as control system performance (as a function of the system timing), extensibility, testability etc.
- The aim is to give support for the description of realistic control systems. This implies 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. Furthermore the processing elements can be placed on different locations in the mechanical structure, hence the sites of sensors and actuators should be modelled as well as the assignment of processing elements to such sites. There is also a need for modelling the various system specific overheads, scheduling policies, error handling etc.
- Since design could be based also on existing systems or components, the models must allow description of earlier implementations with many already taken and hence fixed design decisions.

2.2. The structure of the modelling framework

The AIDA modelling framework is outlined in Figure 2. The models 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 threads (no distinction is made here between threads and processes) 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 important attribute of the mechanical models is the physical locations of such interfaces. Currently there is no timing behaviour description implemented for the mechanical parts, instead time-constants and delays are given as attributes at the component level. No continuous and discrete-time behaviour is described for the mechanical system since this would include modelling of mechanical dynamics; for this there already exists good models and tools. The AIDA modelling framework is more thoroughly described in [22].

	Application models	**Computer system models: Software and Hardware**	**Mechanical models**
Overall behaviour	*Modes of Operation*		
Structure	Functional block diagrams	Computer hardware structure Process structure diagram	Location model Interface components placement
Timing Behaviour Specification & Implementation	Timing and Triggering Diagram (TTD) Implementation TTD	Inter Process Communications IPC model, Communication resources, Communication resource chains Processes: Process TTD and Process internal TTD Computer system: Operating system, Scheduling policy, Synchronisation	*See comment in section 2.2.*
Component Descriptions	Detailed function description Detailed data flow description	HW characteristics: Processing elements, Communication links, Clocks SW characteristics: Implemented Processes	Interface component model Environment interfaces model

Figure 2. Overview of the structure of the AIDA modelling framework

2.3. A case study used to describe a representative subset of the AIDA models

A sub-set of the models will be briefly described in the coming sections with simple examples to give a flavour of, and an introduction to the modelling framework. The emphasis in the description is on parts of the application and computer system models. The description is based on a case study in which models of the control system of a four legged vehicle were derived. In the case study, fixed vehicle mechanics is assumed as is the design of the control application. The vehicle is currently being built at the Mechatronics lab.

In the complete case study, described in [22], two choices of hardware architectures are investigated. Based on the developed application and computer models a coarse analysis of the timing behaviour of the implementation is performed.

2.4. Application models

The application is the control design as it is originally modelled in for example Simulink. In the AIDA application models, the control design is further specified with three basic types of diagrams: The macro mode transition diagram, the functional block diagrams and timing and triggering diagrams.

The Macro Mode Transition Diagram (MMTD) describes the different high-level control modes of the system. An example is the take-off, cruising and landing of an aircraft. Each mode can be expected to use substantially different controllers that are best described separately.

The Functional Block Diagrams describe the data-flows between the different control functions within each macro mode. Note that the emphasis is on describing the functional and implementation independent structure of the control application with its functional blocks and data-flows. No execution semantics (e.g. execute a block when data is available) is associated with a block diagram. Instead, precise timing and triggering specifications are given in separate diagrams.

Figure 3 shows a part of a functional block diagram for the control system of the vehicle mentioned above. The torque control loop of one joint of the vehicle's four legs is shown. The diagram shows how functions *S*, *TC* and *A* are connected to perform a sample (*S*), compute (*TC*) and actuate (*A*) loop. The position samples include measures of two joint angles (on both sides of the flexible joint), used by *TC* to compute the joint torque.

Data-flows to functions not shown in Figure 3 are indicated by dashed lines. Those data-flows correspond to the position sample, *pos*, sent to an observer function higher in the hierarchy and the reception of a torque reference signal, *torq*, from a higher level controller.

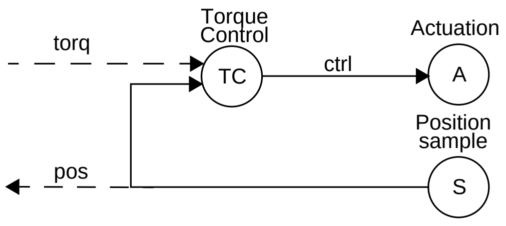


Figure 3. Functional block diagram for the torque control loop.

The behaviour of the application in terms of timing, triggering and precedence is modelled in TTDs (Timing and Triggering Diagrams). The same functions are used as building blocks as in the functional block diagrams, but the TTDs separate functions that are executed with different periods into activities. A TTD describes the control flow of the activities within each macro mode. Precedence relations, time and event triggers are shown and tolerances can be given for the control delays, jitter and synchronisation. This is useful to specify the requirements that the control design poses on the implementation.

With the example functions in Figure 3, a TTD can be specified. It is assumed that all functions in the torque control loop are specified to be executed with the same frequency (period) but that the actuate function should be triggered 0.5 *ms* after the sample function in order to achieve a constant control delay. Hence, the functions are collected into one activity, described by one single TTD. It is further assumed that there are requirements on the admissible jitter of the sample function, as well as on the variation in control delay. These requirements are shown in the TTD as specifications on the time trigger. A time trigger is defined as follows: `<Type: Source, Period:[Delay], Tolerance>`. The *Type* describes if it is a time, event or loop trigger (TT, ET or LOOP). The *source* indicates what the source of the trigger is (a clock for time triggers and an event for event triggers). *Period* gives the period of a time trigger and minimum period of an event trigger. *Delay* is an optional specification of the delay (phase) of the trigger compared to other triggers having the same source. Finally, the *tolerance* defines a tolerance of what variation is admissible from the specified triggering time.

Figure 4 shows the TTD for the torque control loop in Figure 3. The precedence relations between the functions are shown as well as the explicit triggers that both the sampler (*S*) and actuate (*A*) functions are given. It is specified that the actuation should be done 0.5 *ms* after the sampling. Both the sample and actuate functions are allowed to deviate with at most 100 µ*s* from their specified start times

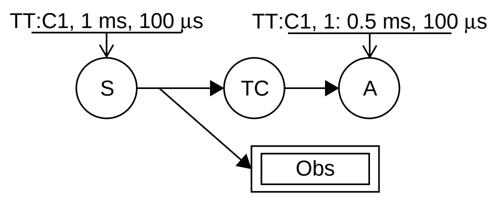


Figure 4. Timing and triggering diagram for the torque control activity.

The double rectangle defines a *rate interfacing function* which is used to describe asynchronous (non blocking) communication between different activities. The meaning is that the sample function sends data (*pos* in Figure 3) to an observer function, *Obs*, that belongs to the parts of the control system that are not modelled here. The observer function should however be part of some other TTD in a complete system modelling. In a similar manner, a rate interfacing function (tagged *TC*) is used in some other TTD (which is not given here), to describe the communication associated with the data-flow, *torq*, in Figure 3. In that case, *TC* is the receiving function.

A timing and triggering diagram can describe parallel paths as well as alternative paths, selected at run-time. A path that is not necessarily executed every time the activity is triggered is called a micro mode. Two complementary micro modes may e.g. be the use of a fast linear controller versus using a better but slower non-linear one. It should be noted that the arbitration between different micro modes is not described at a low level with e.g. if-then-else constructs. Micro modes are included only to indicate possible execution paths in an activity, not to show how the micro modes are selected.

One additional timing and triggering diagram is used to describe the timing that results from an implementation. The Implementation TTD (I-TTD) is expected to be the result of some analysis tool within the AIDA tool-set. An I-TTD is similar to its corresponding original TTD but with the difference that extra delay functions have been included to show the effects of implementation. Furthermore, the triggers are specified with their achieved values on jitter, delays and period. The added delay functions include delays originating from e.g. communications and scheduling effects such as the sequential execution of concurrent activities.

Figure 5 shows an I-TTD for the torque control activity in Figure 4. The delay functions D_1 and D_2 have been included and the values specifying the triggers have been exchanged for some estimated values obtained by analysis.

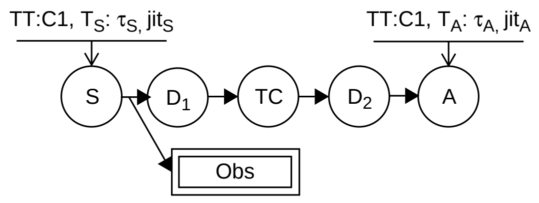

TT:C1, T_S: τ_S, jit_S
TT:C1, T_A: τ_A, jit_A
S
D_1
TC
D_2
A
Obs

Figure 5. Implementation timing and triggering diagram for the torque control activity.

At the functional level of the application models, the functions and the data-flows are described with worst case execution times, communication requirements etc. Hence, the functions are not described in detail, only their resource needs are specified.

2.5. Computer System

The computer system models include specification of the computer hardware, the processing elements and their inter connecting communication links. Also, the operating system is described in terms of scheduling policies, dispatching time, clock tick period etc.

When the application has been partitioned into processes, the connections (communications) between the processes are shown in a process structure diagram.

Inter process communications (IPCs) are modelled with communication resources (message queues and transmitters) that specify e.g. blocking times of message queues and scheduling policies of the communication links.

The communication models can later be used for analysis of communication latencies.

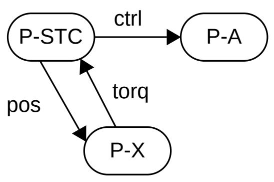


Figure 6. Process structure diagram for the three processes in the example.

Assume that the example activity shown in Figure 4 is partitioned into two processes, *P-STC* and *P-A*. The partitioning may correspond to e.g. a distributed implementation with the sample and computation functions at one processor, and the actuation at another. *P-STC* contains the sample (*S*) and computation functions (*TC*), and *P-A* contains the actuation function (*A*). Also assume that another process, *P-X*, corresponding to all other activities in the system, is defined. The corresponding process structure diagram is shown in Figure 6. Note that three IPCs are defined: *pos*, *torq* and *ctrl*. Each of the IPCs are data-flows in Figure 3 that "cross process borders". For simplicity, the IPCs are here given the same tags as their corresponding data-flows in the functional block diagram.

The timing of the processes is specified in a Process timing and triggering diagram (P-TTD) that looks like a regular TTD only that it specifies the timing and triggering of the processes and not of the functions. In Figure 7, a P-TTD is shown that defines the requirements of the process timing.

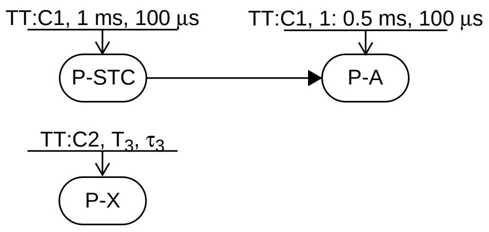


Figure 7. Process timing and triggering diagram for the three processes in the example.

The same types of triggers are used as in the regular TTD but no rate interfacing functions are included. Instead, the correspondence to rate interfacing functions, the inter process communications, are visible in the Process internal timing and triggering diagrams (Pi-TTD). A Pi-TTD defines the control flow internal to a process with the functions from the original TTD as building blocks. Also the IPCs are visible as blocks defining the reception and sending of a message. Furthermore, any functions that are allowed to execute in parallel according to the original TTDs, are shown in sequence. The blocks corresponding to the sending and the reception of a message are associated with communication resources belonging to some communication model. The communication resource blocks are shown as rectangles in the Pi-TTD.

In Figure 8 the Pi-TTDs for the processes *P-STC* and *P-A* are shown. The first of the diagrams shows the *P-STC* process with visible blocks for interfacing the sender and receiver communication resources. Sending of a position message (*pos*) is shown as the block *S_pos*, reception of torque reference is tagged *R_torq* etc.

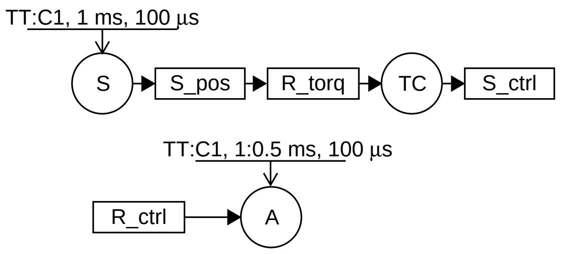


Figure 8. Process internal timing and triggering diagrams for process *P-STC* and *P-A*.

Note that all the diagrams shown in this example are specification diagrams used to specify a required behaviour rather than showing the results of an implementation. Just like in the TTD and I-TTD case however, the diagrams can be used with achieved values of the time triggers to show a result rather than a specification. In that case, triggers are assigned to the starting block of each process. Compare Figure 8 where the requirement specification is given for the timing of the actuate function.

To describe the behaviour of IPCs the AIDA models use communication resources (CRs) and chains of such, called communication resource chains (CR-chains). These provide the ability to describe various implementations ranging from an RTOS message queue to a complex communication system including software and networks. A communication resource is an entity used for communication. It can be either a message queue with an associated data storage and arbitration policy, or it may be a transmitter which does not store data. Typically, a register in a communication device is modelled as a one place buffer (in AIDA terms, a message queue of length one) and the communication media is described by a transmitter resource. A communication resource may be used in several CR-chains associated to different IPCs. This makes it possible to model priority arbitration between different messages using the same resource(s). The CR-chains also make it possible to describe priority arbitration in several levels (one in each resource). A CR-chain starts and ends with message queues to which the sending process writes and from which the receiving process reads the data (called sender and receiver resource respectively).

Figure 9 shows the CR-chain for the *ctrl* IPC from Figure 6 when the IPC is implemented over a Controller Area Network (CAN). *Mailbox 1* is a message queue resource describing the buffer in the CAN communication

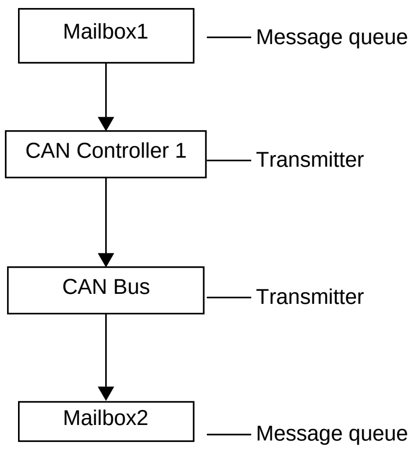


Figure 9. Visual description of a communication resource chain for an IPC using a CAN network.

chip of the sending node. The *P-STC* process ‘writes’ data to *Mailbox 1* when it wants to send data via the corresponding *ctrl* IPC. The *CAN controller 1* resource is a transmitter that performs arbitration on the available mailboxes that have data to send. The *CAN bus* is likewise a transmitter resource that, in priority order, selects one of the messages served to it by its connected CAN controllers. The last communication resource, *Mailbox 2* in the described chain, is the buffer in the CAN chip at the receiving node. *Mailbox 2* is modelled as a message queue from which the receiving process (*P-A*) reads the data.

3. QUALITATIVE COMPARISON WITH RELATED MODELLING APPROACHES

The purpose of this section is to provide another perspective to, and evaluation of, the AIDA models. Three interesting modelling approaches have been chosen here. These are representative in that they represent efforts from different ‘communities’ and methods including ‘object-oriented analysis and design’, ‘structured analysis and design’, and ‘real-time scheduling research’. Some further comments on these choices and their relations to other modelling approaches are given below. The selection criteria are discussed in section 3.2.

3.1. Modelling approaches selected for comparison

- **RT-UML** - Real-time extensions to the Unified Modelling Language. UML is the result of a recent effort in the object oriented community to develop a unified (object-oriented) modelling language with a graphical notation, [29][30]. Consequently, UML incorporates most of the models and diagrams found in previously developed OO methods. The large UML repertoire includes diagrams for describing use cases, class/object structures and relations, statecharts (state transition diagrams) and activity diagrams, sequences, collaboration, components and hardware deployment. UML also incorporates three extensibility mechanisms to extend or tailor the modelling capabilities. These mechanisms include constraints (rules applied to elements of a model), tagged values and stereotypes (with which new model elements can be created and associated with user-defined icons). UML is claimed to have very broad applicability also encompassing at least real-time event-triggered systems, [30][26]. One major question for research still appears to be to what extent object orientation and UML are sufficient for modelling hybrid real-time control systems. It is relatively well accepted that object orientation in general, see e.g. [31][6][25], and UML in its current form, see e.g. [20], are not sufficient for real-time systems. Limitations include the specification of timing constraints, the handling of periodic time-triggered systems, and the definition of specific computer system configurations. This is also being manifested by the ongoing work within the Object Management Group to define real-time extensions for UML. In this comparison the basis is the use of UML for real-time systems as described in [9] and the real-time extensions proposed in [20] and other related articles, [1].

- **RTT** - Real Time Talk**.** RTT is a design framework for developing distributed real-time applications, [10]. RTT is based on object orientation with the premise of reuse and modularity advantages. To support real-time systems a number of notions are incorporated into an extended object oriented framework. These notions include explicit descriptions of synchronization (ordering the execution of objects), timing requirements of individual and interacting objects, and communication between objects. RTT encompasses programming in the large and in the small, and includes considerations on how to combine soft and hard real-time software. In terms of timing models and analysis, RTT is closely linked to the real-time scheduling theory community. DEDOS, [31], and HRT-HOOD, [6] are two approaches that are related since they also extend traditional object oriented concepts with data- and control-flow specifications. As in RTT, timing analysis based on real-time scheduling theory is incorporated in the design guidelines.
- **DARTS/DA -** the design approach for real-time systems extended for distributed system, [12][13]. DARTS/DA builds upon the structured analysis and design (SA/SD) methods but extends these for concurrent distributed systems. In traditional SA/SD, the analysis and design steps are accompanied by a set of diagrams including hierarchical data-flow diagrams (DFD) which on the highest level describe the system context. The structure of a distributed computer system and of operating system processes are also described by DFD's. The DFD's may include control-flow (divided into triggers, enable and disable events) and control transformations. Additional models include state transition diagrams, data dictionaries, mini-specifications of primitive transformations, and structure charts showing the system decomposition into modules. DARTS/DA, and refined versions of it [13], elaborate 'task architecture diagrams' (i.e. the process structure and the inter-process communication principles) and also modularization into objects (information hiding modules), thus attempting to bring the structured and object oriented approaches together.

3.2. Selection of comparison criteria

To support the design issues targeted in this paper a modelling framework must enable multidisciplinary and concurrent development. It must enable high-level design trade-offs to be made and evaluated from both control and computer system perspectives. There is then a need to incorporate suitable viewpoints, relations and links between these views, see e.g. [32]. In addition, it is essential that the specification languages and models (in textual or graphical form) capture the designer's conceptual view of the system with minimum effort, [11][21]. I.e., the models should be natural to use and preferably directly and explicitly support the features that the designer wishes to model.

Several definitions and classifications of views exists. Some classifications focus on the users and/or subsystem viewpoints. This is for example the approach taken in the COntrolled REquirements expression requirements analysis method. This is related to the AIDA modelling parts, where domain views, corresponding to different disciplines are provided. A closely related classification of views is given by the "4+1 view model of architecture", [16]. The views include the *logical* view (describing system functionality and structure in terms of objects and classes), the *process* view (describing concurrency and communication), the *development* view (describing the organization of the actual software modules), the *physical* view (describing the mapping of the software on the hardware), and the *scenario* view, which is used to support architectural design and to indicate essential requirements.

Another classification is given in [21] where a view is said to describe a system with respect to some set of attributes or concerns. The following views are defined: *system purpose* (i.e. requirements), *form* (i.e. structure), *behavioural/functional*, *performance*, *data*, and *managerial* (process related).

Which are the most essential views? This probably to a large extent depends on the type of system, compare with a database vs. a real-time control system. According to [21] the *system purpose* and *form* views are in general the most important ones. Given certain critical system requirements, models to analyse their satisfaction are also essential. For distributed real-time systems and with the focus of this paper it follows that the requirements and structure criteria (and views) need to be complemented by performance models that enable analysis and prediction of the timing behaviour of a particular system.

Based on the preceding discussion the following criteria for qualitative comparison have been chosen:

- *Timing behaviour - can the specified timing as well as the implemented timing behaviour be described with relation to system functions?* Timing behaviour is one of the key attributes and requirements addressed in this evaluation. There is a need to express relevant timing requirements for the targeted time- and event-triggered control systems, such as response and feedback delays, sampling periods, allowable jitter, synchronization and precedence. There is also a need to describe the timing behaviour of an implemented system at an application abstraction level, i.e. in terms of the actual or predicted delays and timing variations that are introduced due to computations, communications, blocking, non synchronous activities, and interference. Due to the different focuses of the evaluated models the distinction between time- and event-triggered systems is included. Note that time-triggered here primarily refers to the need to describe multirate systems where functions (objects or transformation) executing at different periods communicate. Behavioural descriptions of processes (threads), primarily in terms of the communication primitives (asynchronous vs. synchronous), are not included in the comparison. Such descriptions are available in mixed structural and behavioural descriptions in UML and DARTS/DA in terms of concurrency and data-flow diagrams.
- *Structure:* Given the targeted design issues it is clear that the structures corresponding to the application functions, threading in the software implementation, the hardware and environment interfaces need to be described. Links (mappings or allocations) between these structural representations must be supported. Hierarchies are an important part of structural descriptions and offer well known advantages in dealing with system complexity thereby allowing a designer to focus on different abstractions and subsystems at a time, [19], [11]. Different types of hierarchies exist; the most common is that of decomposition or refinement of objects (components, functions etc.) to describe the details of the objects in terms of lower level objects within the same domain. Compare for example with a computer hardware structural description starting at the network level, where each network is composed of nodes and communication links, and where each node is composed processors and peripherals, [8]. Nancy Leveson, [19], in the context of system specifications, discusses another type of hierarchy termed "*means-ends hierarchies*". According to [19] these have been shown to play an important role in the understanding of complex systems. "*Ends*" refers to "upper levels" and describes goals, whereas "*Means*" refers to lower level implementation descriptions. Such a hierarchy is applicable both to the structure (compare logical vs. process views) and to the timing behaviour.
- *Resource management - evaluating the performance models with respect to timing analysis:* Under the heading 'resource management' we group policies and mechanisms, in software and hardware, used to implement scheduling, process communication, interprocessor communication and synchronization. Related information includes descriptions of the provided processing and communication performance (e.g. overhead and effective bits/s in communication). Such descriptions are essential to be able to predict the timing behaviour of an application. Other important pieces of information for timing analysis, such as requirements on processing (e.g. the number of and type of operations required) and communication (e.g. messages/s) should be contained within the application description.
- *Graphical notation: evaluating the model diagrams for the engineering of distributed real-time systems*. Models in an appropriate graphical format are essential for describing complex systems (cmp. with [7]). This point should be seen as a general assessment of each of the models regarding the 'maturity' of the available graphical notations. This criteria is emphasized by the importance of models for communication, for example to provide an understanding of a system under construction, [21].
- *Guidelines: do guidelines exist that indicate how the models can be used in the design of distributed real-time systems?* Given a modelling approach there is a need for guidelines on how to use the models in system

design. A modelling approach may be closely associated with a particular design philosophy or more standalone. The emphasis here is to judge whether available guidelines do provide support for the design issues of Figure 1. A guideline should describe how the models can be used to represent the structure of the system, the requirements on the system, the timing behaviour of the implemented system and how the model contents can be used for analysis and other design activities. As for the timing behaviour criteria a distinction between time- and event-triggered systems is included here.

3.3. Qualitative comparison and discussion

For the related modelling approaches, the comparison should be seen as an attempt to answer the following question; How well does the modelling approach support the design issues of Figure 1? Although the comparison is qualitative it is of interest for the AIDA project also for other reasons. So far the AIDA project has focused on the multidisciplinary aspects of the models and their content rather than on the graphical notation and the language for describing the models. I.e., a further question is if the AIDA models can benefit from the other approaches in this respect.

Despite the fact that the modelling approaches chosen for comparison in some sense target distributed real-time systems, they are still rather different. The AIDA modelling framework is targeted to support certain specific design issues for real-time control systems. UML on the other hand is a “general purpose” modelling language with software origin that has been extended to better handle distributed real-time systems. Also DARTS/DA is more general in nature compared to AIDA. There is a closer similarity between AIDA and RTT since they are both targeted towards distributed real-time control systems, although AIDA is more focused on the modelling framework and RTT on software design.

The comparison is summarized in Figure 10 where, for each approach, full, partial or little/no support for the comparison criteria is indicated through black, gray and white bullets respectively.

	Timing Behavior: Time T. / Event T.		Structure	Resource Management	Graphical notation	Guidelines: Time T. / Event T.	
AIDA	●	○	◐	●	○	●	○
RT-UML	○	●	◐	○	●	○	◐
DARTS/DA	◐	◐	●	○	●	◐	◐
RTT	◐	○	○	◐	○	●	○

Figure 10. Qualitative comparison of modelling approaches for distributed real-time systems

Timing Behaviour: Time-triggered (TT) vs. Event-triggered (ET) systems

- The **AIDA** timing and triggering diagrams describe the required and implemented timing behaviour, cmp. with section 2. The models are developed primarily with TT multirate systems in mind. Limited support is provided for event-triggered parts where overall timing requirements on event-sequences can be specified. No details such as conditions on state transitions can be specified, thus the behaviour of micro-modes and modes can not be described in detail.
- **UML** can describe timing behaviour through sequence, activity and state transitions diagrams. In the real-time extensions of UML, the concept of modes and mode transitions, and timing information in sequence diagrams have been included, [1][20]. With the extended sequence diagrams it is principally possible to describe both the required and implemented timing behaviour. The basic modelling concepts, however, are developed for event-triggered object oriented systems. It remains to be shown how useful these diagrams are for time-triggered multirate systems. Compare with the pinpointing of this problem by ObjectTime, [25].

- **DARTS/DA** expresses requirements on timing textually. The behavioural model is based on data-flow diagrams extended with a discrete-event formalism. The data-flow diagrams thus combine pure data-flow with control flow, where the control transformations are linked to finite state machine descriptions. The data-flow diagrams have associated triggering semantics, transformations are triggered to execute upon the arrival of data. These descriptions are used for requirements and may encompass time- and event-triggered systems although they do not seem to have been applied to multirate systems. There are no explicit means or methods to describe the implemented timing behaviour.
- The **RTT** programming model is hierarchical and in descending order composed of modes, use-cases, tasks and objects. Tasks are used to encapsulate objects, and provide the thread of control and communication for the object. Timing behaviour is described through extended scheduling theory models; precedence graphs define the order in which tasks execute, the timing attributes of individual tasks (execution time, release time relative period and deadline relative period), constraints among tasks such as mutual exclusion, and the timing attributes of the precedence graphs (period time). Although multirate systems can be specified this is done in terms of task oriented timing constraints rather than application level timing requirements, [24]. On the other hand, the model is formal and an off-line scheduler tool has been developed and successfully commercialized. The description is focused on time-triggered systems for which off-line scheduling is used in the implementation. There is currently no way to fed back results from timing analysis to describe the actual behaviour of implemented tasks.

Structure:

- **AIDA** includes provisions for the required structures (application functions, threading, the hardware and environment interfaces). The different hierarchical relations however need to be further developed.
- **UML** with its real-time extensions, has the models needed to describe an object-oriented structure, threading in the software implementation, the hardware and environment interfaces. Again, however, the basic modelling concepts are developed for event-triggered object oriented systems where functionality and transformational views traditionally have been de-emphasised. One slight problem with the object and class views are that they really constitute integrated views combining a number of hierarchies; aggregation, inheritance and associations. Aggregation is a decomposition of functionality (or composition of objects) whereas inheritance defines an additional class hierarchy. Associations, on the other hand, can be used to model for example inter object communication, i.e. data-flow. Although all these views are valuable it is difficult to see what a control engineer would gain from having them all integrated. Nevertheless, a tool could of course in principle solve this problem by only showing associations corresponding to data-flow, thus in principle converting the class diagram to a functional block diagram. Alternatively, a collaboration diagram could be used to show functional structure, [1]. Further evaluation is required to show that UML is useful for time-triggered control systems and that the models can satisfy the “conceptual views” of designers.
- **DARTS/DA** provides models for describing a so called physical model, i.e. hardware and interfaces, and a threading structure. The functional structure is given through the combined behavioural data-flow and control-flow diagrams. Links between these structures are supported. This provides a relatively complete view of the required structures.
- **RTT** describes the object structure and object communications and the task structure (for time-triggered threads). No hardware structure is given.

Resource management:

- **AIDA** focuses strongly on this issue, compare with Figure 2. None of **RT-UML, DARTS/DA** and **RTT** come close to these types of models. **RTT** provides partial information here; for the hard-real time model there is ample of information regarding timing requirements and the off-line scheduling. However, other aspects are very limited.

Graphical notation:

- In **AIDA** specific work on a graphical notation has mainly focused on the timing and triggering diagrams. These were developed to complement functional block diagrams and to separate timing behaviour from structural specifications. Both **UML** and **DARTS/DA** lean on diagramming concepts that have been used for quite some time although in different notational versions. For **UML** there is still work to do on the real-time extensions. As for AIDA, **RTT** provides a graphical notation only for parts of the models.

Guidelines: Time-triggered (TT) vs. Event-triggered (ET) systems

- For **AIDA** this is a strong point with respect to *TT* systems although there is as yet no single condensed guideline. The research in [22][24][27][28] provides guidelines on the derivation of timing requirements and constraints, control system structuring, allocation, choice of execution and communication strategies, and exploits scheduling theory for timing analysis. This is also a strong point for **RTT**, partly carried out in cooperation with the AIDA-team. For both, there is much less emphasis on *ET* systems.
- For **UML** the numerous existing OO-methodologies provide good support in the design of event-triggered non real-time systems. Guidelines which attempt to cover distributed real-time systems include [16][26][9]. These however provide relatively little support for these issues. For example, in [9], partitioning, the use of deployment diagrams, and links to scheduling are briefly discussed. The guidelines are similar in nature for **RT-UML**, [1] which however place more emphasis on modelling concurrency, timing information in sequence diagrams, the hardware representation and the system context. Regarding the event-triggered parts, research developments in timed automata, formal semantics, state machine analysis and verification techniques (see e.g. [17]) still remain to be exploited. All in all there is room for more work on guidelines having both time- and event-triggered systems in mind for object-oriented systems.
- **DARTS/DA** provides guidelines in terms of heuristics for allocation and partitioning. Some timing analysis considerations for ET and TT systems are also incorporated based on fixed priority scheduling [13].

3.4. Concluding discussion on the AIDA models: views and model relations

The “4+1 view model of architecture”, [16], corresponds relatively well to AIDA. The *logical* view matches the AIDA application models, and the *process* and *physical* views match the structural parts of the AIDA computer system model. There is no direct correspondance in AIDA to the *development* view describing the software module organization since this topic is not within the scope of AIDA. The computer system resource models come closest to the development view in that it provides a layered view of the implementation software and hardware. The scenario view can be seen as a correspondance to the timing requirements part of the AIDA application models. A difference in AIDA is that it in addition incorporates these requirements and implementation models for both the application and computer system timing behaviour models. The AIDA resource models, cmp. Figure 9, do not seem to have a correspondance in the “4+1 view model” which are more focused on describing system structure.

Compared to the classification given in [21], the AIDA models provide a set of requirements focused on timing, structural, timing behaviour and performance models. There is no data view or model in AIDA. There is no explicit *managerial* (process related) view but the models are related to an associated design method.

The multidisciplinary views provided in AIDA could be interpreted as *means-ends* hierarchies (the term introduced in [19], recall section 3.2.) for both structure and timing behaviour. The latter hierarchy is established by the functions in the application model that describe the required timing behaviour and the computer system model which describes the implementation in terms of schedulable processes and the resources and policies used in the implementation.

Reindeer is a recent research project where an environment to support design of distributed systems is being developed, [8]. Integrated within Reindeer is a diagramming methodology where sets of diagrams are provided for each hierarchical level of design (requirements, network, node, processor, task). The utilized diagrams can be seen as an extension to DARTS/DA incorporating also for example sequence diagrams and use cases. Relations between these diagrams through the hierarchies are also provided. The proposed diagrams to a large extent correspond to the AIDA models wherefore this project is of interest to study further.

4. CONCLUSIONS AND FUTURE WORK

The paper has described the modelling framework that has been developed within the AIDA project. The AIDA modelling framework is by no means complete, there still remains work and research on some of the submodels. Future work on the AIDA modelling framework will include refinements and more exact semantic definitions of the models and their hierarchies. Since the emphasis in the design of the modelling framework has been the modelling of time-triggered control systems, the modelling of event-triggered systems is only partially supported. The direction the models should take here is open at this stage. From Figure 10. and with the extensibility mechanisms of UML in mind it is straightforward to say that the combination, or integration, of the AIDA modelling framework with RT-UML could be very fruitful. Further evaluation of RT-UML in tool implementations will therefore be carried out.

Desired analysis and synthesis capabilities of the prototype tool-set will be investigated. The primary idea is to exploit advances in real-time scheduling theory that can be incorporated into the toolset. The execution time analysis that is supported by the models is coarse. Functions for more accurate execution time analysis are not planned to be a part of the AIDA tool-set even though it may be of interest to include this in the future. The toolset will provide for further evaluation possibilities of the models and their graphical notations.

An interesting issue for further model developments is the consideration of including other architectural aspects into to the framework. A prime candidate here is dependability requirements and their implementation. Today, the AIDA models can be used to describe certain requirements and behavioural characteristics of redundant/fault-tolerant systems; for example, a particular redundancy approach usually comes along with fixed and specified strategies for synchronization, scheduling, etc. Explicit specifications of fault-tolerance requirements and the mechanisms for error detection and handling are currently not part of the models.

It appears that there are still several research challenges in modelling, in particular regarding “hybrid” systems composed of both time- and event-triggered subsystems, the relations between views, and how all this can be incorporated into a design tool provided with appropriate graphical representations to be beneficial for users. It is also clear that tool and model integration will be increasingly called for in the future to combine the strength of different modelling formalisms and analysis tools.

These interesting problems and probably many more are ahead of us as we embark on the prototype tool implementation.

5. ACKNOWLEDGEMENTS

This work was supported in part by the Mechatronics trade-association, the Swedish National Board for Industrial and Technical Development (NUTEK) and the Volvo Research Foundation. The authors want to acknowledge the AIDA reference group for providing us with useful input in the development of the modelling framework. In addition De Jiu Chen and Marcelo Masera are acknowledged for valuable comments.

REFERENCES

[1] ARTISAN Software. http://www.artisansw.com/articles/index.html

[2] AIDA (1998). http://www.damek.kth.se/~martin/aida.html

[3] Ancilotti P. et al. Design and Programming Tools for Time Critical Applications. *J. of Real-Time Systems*, Vol. 14, No. 3, 1998.

[4] Bass J. M. et al. (1994), *Automating the Development of Distributed Control Software*. IEEE Parallel & Distributed Tech., Winter 1994.

[5] Bhatt D. (1996), *A methodology and toolset for the design of parallel embedded systems.* In Proc. of the School on Embedded Systems, Organised by the European Educational Forum, Veldhoven, NL, Nov. 1996

[6] Burns A., Wellings A.J., (1994). HRT-HOOD: A Structured Design Method for Hard Real-Time Systems. *J. of Real-Time Systems*, Vol. 6, No. 1, 1994.

[7] Cooling J.E. Real-time software systems: an introduction to structured and object-oriented design. Int. Thomson Computer Press, 1997.

[8] Cooling J.E. et al. The design of software for real-time systems. *European Embedded Systems Conf.* Sept. 1998.

[9] Douglass B.P. (1998). *Real-time UML*: Developing efficient object for embedded systems. Addison Wesley Longman, Inc.

[10] Eriksson C., et al. (1996). An Overview of RealTimeTalk, a Design Framework for Real-time Systems. J. of Parallel and Distributed Computing, No. 36, p 66-80, Academic Press Inc.

[11] Gajski. D.D. et al. Specification and design of embedded systems. Prentice Hall 1994.

[12] Gomaa H. A software design method for distributed real-time applications. J. of Systems and Software, 9, 81-94. Elsevier Science Publishing.

[13] Gomaa H. *Software design methods for concurrent and real-time systems*. Addison-Wesley publishing company, 1993.

[14] Hanselmann H. (1995), *DSP in Control: The Total Development Environment.* Int. Conference on signal processing applications and technology. Oct. 24-26, 1995, Boston, MA, USA.

[15] Hansson H. et al. (1998). BASEMENT: A distributed real-time architecture for vehicle applications. *J. of real-time systems*, 11, p. 223-244. Kluwer Academic Publishers.

[16] Kruchten P. The 4+1 view model of architecture. *IEEE Software* Nov. 1995, 12 (6), pp. 42-50.

[17] Larsen et al. (1997). UPPAAL in a Nutshell. *Springer Int. Journal of Software Tools for Technology Transfer*, 1(1/2).

[18] Lauwereins R. et al. (1995), *Grape-II: A system-level prototyping environment for DSP applications.* IEEE Computer, Feb. 1995, pp. 35-43.

[19] Leveson N.G. Intent Specifications: An approach to building human-centered specifications. 1998. *IEEE*

[20] McLaughlin M.J. and More A. (1998). Real-time extensions to UML: Timing, concurrency and hardware interfaces. *Dr. Dobb's Journal* December 1998.

[21] Rechtin E. and Maier M.W. (1997). *The Art of System Architecting*, ISBN 0-8439-7836-2, CRC Press, 1997.

[22] Redell O. *Modelling of Distributed Real-Time Control Systems, An Approach for Design and Early Analysis*, Licentiate Thesis, Dep. of Machine Design, KTH, 1998, TRITA-MMK 1998:9, ISSN 1400-1179, ISRN KTH/MMK--98/9--SE

[23] Roppenecher G. and Wallentowitz H. (1993). Integration of chassis and traction control systems: What is possible- What makes sense - What is under development. J. of Vehicle System Dynamics, 22 (1993), pp. 283-298.

[24] Sandström K. Eriksson C. Törngren M. (1999). Modeling and scheduling of control systems. Research Report Dep. of Machine Design, KTH, TRITA-MMK 1999:4, ISSN 1400-1179, ISRN KTH/MMK--99/4--SE

[25] Selic B. Periodic Tasks in Room. Position paper submitted to the *Workshop on OO Real-Time Systems, ACM OOPSLA '95 Conference*, Austin Texas, Oct. 15-19, 1995.

[26] Selic B. and J. Rumbaugh. Using UML for Modelling Complex Real-Time Systems. White Paper. www.rational.com/uml/resources/whitepapers

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

[28] Törngren M. (1995), *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.

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

[30] UML Summary. Version 1.1. Sept. 1997. Object Management Group, doc. no. ad/97-08-03.

[31] Verhoosel J. (1995), *A Model for Scheduling of Object-Based, Distributed Real-Time Systems*, J. of Real-Time Systems, Vol. 8(1), January 1995

[32] Vestal S. Integrating Control and Software Views in a CACE/CASE toolset. *IEEE/IFAC Joint Symposium on Computer Aided Control System Design for Control Systems*, pp. 353-358, Tucson, AZ, 1994.

[33] Wikander J. and Törngren M. Mechatronics as an engineering science. Proc. of the *6th UK Mechatronics Forum Int. Conf.* Skövde Sweden, 9-11 September 1998.