KTH Industrial Engineering
and Management

A Model Management and Integration Platform for Mechatronics Product Development

Jad El-khoury

Doctoral Thesis
Stockholm, Sweden 2006

TRITA – MMK 2006:03
ISSN 1400-1179
ISRN/KTH/MMK/R-06/03-SE
ISBN 91-7178-268-0

School of Industrial Engineering
and Management
Department of Machine Design
Royal Institute of Technology
SE-100 44 Stockholm, Sweden

Academic thesis, which with the approval of Kungliga Tekniska Högskolan, will be presented for public review in fulfilment of the requirements for a Doctorate of Engineering in Machine Design. The public review is held at Kungliga Tekniska Högskolan, Salongen, Osquars backe 31, Stockholm; at 10:00 on the 3rd of March 2006.

Abstract

Mechatronics development requires the close collaboration of various specialist teams and engineering disciplines. Developers from the different disciplines use domain-specific tools to specify and analyse the system of interest. This leads to different views of the system, each targeting a specific audience, using that audience's familiar language, and concentrating on that audience's concerns. Successful system development requires that the views of all developers produced by the different tools are well integrated into a whole, reducing any risks of inconsistencies and conflicts in the design information specified.

This thesis discusses techniques of managing and integrating the views from various disciplines, taking better advantage of multidisciplinary, model-based, development. A Model Data Management (MDM) platform that generically manages models from the various domain-specific tools used in development is presented. The platform is viewed as a unification of the management functionalities typically provided by the discipline-specific PDM and SCM systems. The unification is achieved by unifying the kind of objects it manages – models. View integration is considered as an integral functionality of this platform.

In demonstrating the platform's feasibility, a generic version management functionality of models is implemented. In addition, model integration is investigated for the allocation of system functions onto the implementing hardware architecture. The proposed approach promotes the independent development of the views, allowing developers from each discipline to work concurrently, yet ensuring the completeness, correctness and analysis of any inter-view design decisions made.

The prototype MDM platform builds on existing technologies from each of the mechanical and software disciplines. The proposed MDM system is built based on a configurable PDM system, given its maturity and ability to manage model contents appropriately. At the same time, the version control functionality borrows ideas from the fine-grained version control algorithms in the software discipline.

The platform is argued to be feasible given the move towards model-based development in software engineering, bringing the discipline's needs closer to those of the hardware discipline. This leads the way for an easier and more effective integrated management platform satisfying the needs of both disciplines using a common set of mechanisms.

Table of Contents

Preface

The work presented in this thesis is partially funded by the Swedish Strategic Research Foundation through the SAVE project, the national Swedish Real-Time Systems research initiative ARTES, and the Swedish Governmental Agency for Innovation Systems (VINNOVA) through the CODEX project, a collaborative effort between Scania and KTH.

On a personal note, I would first like to show my appreciation to my supervisor *Martin Törngren* for giving me the opportunity to do my PhD studies under his supervision. Thanks also to my colleagues at the Mechatronics Lab, current and old, for providing a pleasant working environment, and for putting up with me all these years. The *Ola*s deserve special recognition for a great cooperation and interesting discussions. My senior, *Redell*! Your role as a co-supervision was vital for my survival. Thank you for your Swedish lessons, and for not giving up on my stubbornness to learn. (Jag tycker att jag börja fatta nu.) My junior, *Larses*! Your enthusiasm for Hårdrock, Capitalism and Sprit will be missed. A salute to my old colleague and friend *Jonas*, our brother in the struggle, for all the laughs and good times we've had in the room and outside.

My thoughts go to all at the Department of Machine Design for the small chats in the photocopier room, 'Good Mornings', lunches, etc. Special thanks go to the administration team. In particular, the admin girls, *Heléne Hedin* and *Jacqueline Bernsten*, for their long dedication for a good drink - I mean for their efficient personnel and economic support. Unfortunately, my work did not involve much welding. Just the same, a thank goes to *Ulf Andorff* and *Kurt Lindqvist* for supervising and blessing my early arrivals to work, during their lunch breaks. You made the dreaded fifteen minute long walk to work less painful. An appreciation goes to *Peter Reuterås*, *Tobias Grundel* and *Payam Madjidi* for their superb IT-support. Most importantly, the Fredagsöl gang leader, *Martin Grimheden*, deserves a special mention for being there late enough to initiate a Friday Beer, any time of the week. Your dedication is appreciated by many enduring PhD students.

Hugs and kisses go to my Family in Sweden: *Anja*, *Anja*, *Farnosh*, *Laurent* and *Mike*. Thank you for your moral support and sustaining my needs for a good chat over coffee breaks, lunches or the odd drink.

Elin! A superb job at reminding me about post-PhD life. Thank you for being there, listening and sometimes ignoring my whining.

I am grateful for my parents, *Marwan* and *Nawal*, for giving me the opportunities to be where I am today. Your unquestioning support is invaluable. Thanks sister *Josiane*; brothers *Walid* and *Toufic*; and dear friends *Sam* and *Toufic* for your arbitrary phone calls in the middle of the night, just because you felt like a chat. I know you care.

Last but certainly not least, for the one person without whom this work would not have been possible: *Me*. Thank you for putting up all these six years. I promise I will not put you through this again. I have said it before, and I say it again: '*Jad*! You're a Genius!'

Stockholm, January 2006

Jad El-khoury

No other humans or animals were harmed
in the making of this thesis.

Appended Publications

Paper A

El-khoury J., Redell O. and Törngren M., A Tool Integration Platform for Multi-Disciplinary Development, 31st Euromicro Conference on Software Engineering and Advanced Applications, 2005.

The platform and algorithms presented are implemented by Jad El-khoury. The paper was written in close collaboration between the authors.

Paper B

El-khoury J. and Redell O., Towards a Multi-View Modelling Environment for Mechatronics Systems, Technical report, ISRN/KTH/MMK/R-05/24-SE, TRITA-MMK 2005:24, ISSN 1400-1179, Department of Machine Design, KTH, 2005.

The work presented in the paper and the writing was made by Jad El-khoury. Ola Redell assisted in discussing and provided input on the mechanisms and implementation details.

Paper C

El-khoury J., Model Data Management – Towards a common solution for PDM/SCM systems, Twelfth International Software Configuration Management Workshop (SCM-12), 2005.

Paper D

El-khoury J., The Version Control Algorithm Implementation in the Model Data Management (MDM) Platform, Technical report, ISRN/KTH/MMK/R-05/27-SE, TRITA-MMK 2005:27, ISSN 1400-1179, Department of Machine Design, KTH, 2005.

Other Publications by the Author

Publications Discussed in this Thesis:

1. Larses O. and El-khoury J., Function Modelling to Improve Software Documentation. Technical report, ISRN/KTH/MMK/R-05/25-SE, TRITA-MMK 2005:25, ISSN 1400-1179, Department of Machine Design, KTH, 2005.
2. Redell O., El-khoury J. and Törngren M., The AIDA-toolset for design and implementation analysis of distributed real-time control systems. Microprocessors and Microsystems, volume 28, 2004.
3. El-khoury J., Chen D. and Törngren M., A survey of modelling approaches for embedded computer control systems (Version 2.0), Technical report, ISRN/KTH/MMK/R-03/11-SE, TRITA-MMK 2003:36, ISSN 1400-1179, Department of Machine Design, KTH, 2003.
4. El-Khoury J. and Törngren M., Towards a Toolset for Architectural Design of Distributed Real-Time Control Systems, Proceedings of Real-Time Systems Symposium (RTSS), 2001.

Other Publications:

1. El-khoury J., Redell O. and Törngren M., Integrating views in a multi-view modelling environment, Proceedings of the 15th International Symposium of the Systems Engineering Conference, 2005.
2. Larses O. and El-khoury J., Views on General System Theory, Technical report, ISRN/KTH/MMK/R-05/10-SE,TRITA-MMK 2005:10 ISSN 1400-1179, Department of Machine Design, KTH, 2005.
3. Larses O. and El-khoury J., Multidisciplinary Modeling and Tool Support for EE Architecture Design, 30th World Automotive Congress, FISITA, 2004.
4. Chen D. J., El-Khoury J. and Törngren M., A Modeling Framework for Automotive Embedded Control Systems. SAE World Congress, SAE Technical Paper Series 2004-01-0721, 2004.
5. Henriksson D., Redell O., El-khoury J., Törngren M. and Årzén K., Tools for Real-time Control Systems CoDesign - A Survey, Department of Automatic Control, Lund Institute of Technology, Internal report - ISRN LUTFD2/TFRT—7611—SE, 2004.
6. Törngren M., El-khoury J., Sanfridson M. and Redell O., Modelling and Simulation of Embedded Computer Control Systems: Problem Formulation, Technical report, TRITA-MMK 2001:3, ISSN 1400-1179, ISRN KTH/MMK/R--01/3--SE, 2001

1. Introduction

With the introduction of computer technology as a feature in mechanical engineering products, a change is experienced in the expected functionality of these mechatronics products, as well as the means of their development. The use of micro-controllers, software, and network systems in modern technical products has permitted functionality that would otherwise be impossible or very expensive. The contribution of this technology is indispensable, and product success is increasingly dependant on it. More resources are allocated to computer technology, in order to gain an edge over competing products. For example, in the ever increasing complexity of automotive electronics, roughly 70% of functional innovations are made possible and performed by software [1].

The advantages of introducing computer technology in modern products come at the cost of increasing the product development complexity, where designers are facing many challenges to ensure that the products meet their requirements.

One source of complexity is due to the dramatic increase in the number of software-based functions in the system. For example, in the automotive industry, X-by-wire functions are projected to boost the share of electronics in chassis production from today's 12% to approximately 40% within the next ten years [2]. While the functions themselves can vary in complexity, the sheer number of these functions forms a development challenge for the complete system. Weinberg [3] discusses the issue of system complexity as related to its size. In promoting his General Systems Thinking, he declares that 'To a first approximation, we were able to use the number of objects as a measure of complexity – the complement of simplicity'. The challenge is to handle systems of 'organised complexity' – systems that are too complex for analysis and too organised for statistics.

Complexity is further compounded by the dependencies between the system functions. Previously standalone functions are becoming more interdependent, where functions need to share common resources, as well as cooperate with each other in order to fulfil their expected behaviour. Besides these functional dependencies, other types of relationships need to be considered during system development such as the mission-criticality or the strategic make/buy relationships between functions [4].

Complexity is not an inherent property of the system itself, but lies in the relation between the system and its observer. Depending on the observer's concerns, different types of objects and relations between them are perceived. For example, given the automation facilities in a modern car, its driver does not necessarily perceive the system complexity in the same manner as its developer that needs to provide such automation support.

In discussing the complexity problems of science, Checkland explains in [5] that the world is a giant complex, and to cope with it, we are forced to reduce it into separate areas which can be examined separately. This arrangement of knowledge is inevitable given our limited ability to take in the whole. 'Our knowledge of the world is thus necessarily divided into different "subjects" or "disciplines"'.

Similarly, when dealing with system development complexity, multidisciplinarity may become a necessity. Mechatronics systems development requires the close collaboration of various specialist teams and engineering disciplines. In automotive system design, for example, developers from the many disciplines of engineering, such as control, software, mechanical and electrical engineering, need to interact to meet the demands for dependable and cost-efficient integrated systems.

The developers from the different disciplines use their own specific tools, providing their own specific views of the system to be developed. Each system view targets a specific audience, using that audience's familiar language (viewpoint), and concentrating on that audience's concerns [6]. Figure 1 illustrates some of the viewpoints and views that may be necessary during the development of a typical vehicular system.

However, multidisciplinarity may in turn become a source of complexity. Developers from the different disciplines differ in the design concerns and interests in which they are involved. These concerns and interests are not necessarily exclusive, which leads to overlap and dependencies in their development information space. Even though they attempt to develop the same system, developers from the different disciplines may then form a different perception of the system's aims, problems and solutions. This becomes a source of conflict and complexity during development.

To take full advantage of multidisciplinary development, it is essential to have good integration of the efforts of all involved disciplines, as well as good communication between them. For successful system development, the views of all developers produced by the different tools should be well integrated into a whole, reducing any risks of inconsistencies and conflicts in the design information specified in these views.

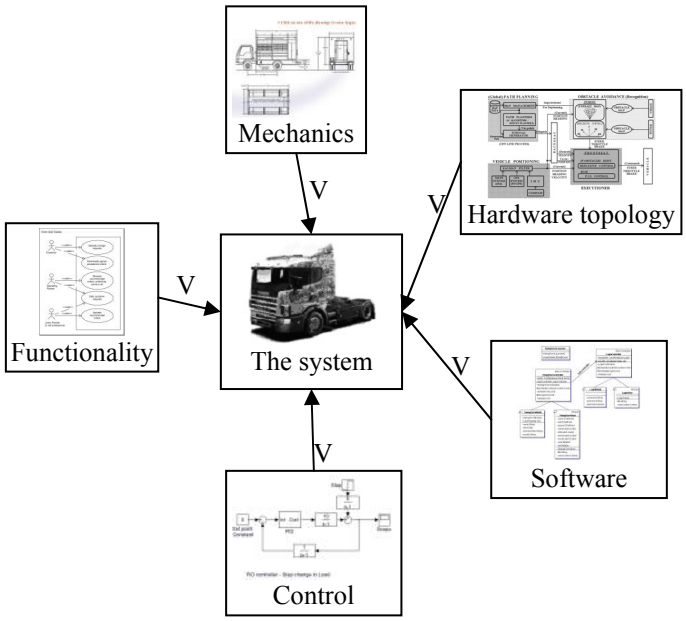

Figure 1. Some of the disciplines and views in system development.

This thesis discusses techniques of managing and integrating the views from the various disciplines, in order to minimise the complexity due to multidisciplinary development, while optimising its benefits.

Prior to presenting the contribution of this thesis, some earlier experiences within the research project in multidisciplinary tool development are discussed in the following section. These experiences justified and inspired the aim and approach advocated in this thesis, which will be detailed in sections 3 and 4. Section 5 introduces the particular thesis contributions, further detailed in the appended papers. A survey of modelling and integration approaches is then presented in section 6, followed by a summary of relevant industrial case studies in section 7. Finally, future work is discussed in section 8 before concluding in section 9.

2. Background - Earlier Attempts

This section presents earlier efforts made within this research project at developing modelling and analysis tools to support certain aspects of mechatronics system design. The aim and approach dealt with in this thesis are motivated by first hand experiences in tool and model integration, discovered by the author when developing and using these tools. A more complete description of the Aida-toolset and XILO tools can be found in [7] and [8] respectively.

2.1. The AIDA-toolset – A Real-time System Design Tool

The Aida-toolset integrates the specification and performance analysis of control systems with embedded real-time system design. Various aspects of the system can be described, from the control system specification to its implementation on a distributed network of processors.

The aim of the toolset is to help the user evaluate a number of different system designs before the actual realisation of the system. Design iterations may include changes in the software structuring, function allocations, hardware structuring, process priorities, process scheduling, communication protocols, etc. Evaluations are based on timing analyses as well as simulations of the resulting control system performance.

The AIDA-toolset is designed to support one particular work-flow, visualized in figure 2, leading to a specific precedence in the order of building the models. Initially, a pure control specification is designed and tested using Matlab/Simulink [9], within which control performance analysis can be performed by simulation. The resulting control algorithm and system dynamics provide the necessary information for the software specification. At this stage of development, important requirements such as controller jitter and delays are often overlooked, since they are dependant on implementation details and their values can only be deduced once the system is implemented. Next the control design is imported into the AIDA-toolset where the Simulink model is translated to a data-flow diagram. The resulting model is augmented with additional information such as execution times

for functions and size of data-flows. This model becomes the base for the real-time system design. In the real-time system design, the user defines the target hardware, allocates the functions to processors, maps the functions into processes and specifies communication, triggering and scheduling related characteristics. When the real-time design is complete, response time analysis techniques are used to calculate the response times and release jitter of the processes and their contained functions. Once successfully analysed, the model is exported back to Simulink for further simulation. The new Simulink diagram is a copy of the original, augmented with the implementation-induced time delays. These implementation effects are hence taken into account in the resulting control performance analysis.

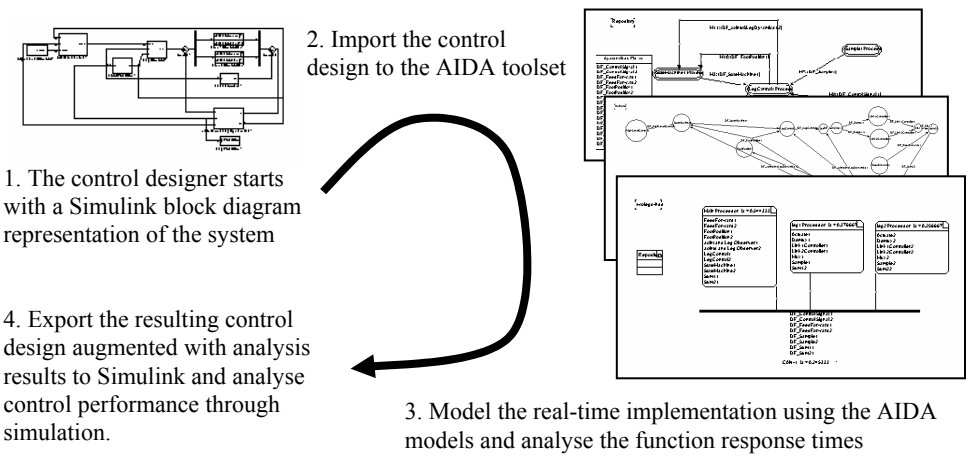

Figure 2. The work flow supported by the AIDA-toolset. Three different system views in the AIDA-toolset are represented to the right: a Process Structure Diagram, a Data Flow Diagram and a Hardware Structure Diagram.

The models used in the Aida-toolset are based on a larger modelling framework for mechatronics systems [10]. In this framework, sixteen different models are defined, of which seven are used in the toolset:

- The *data-flow diagram* (DFD) defines functions specifying the system functionality and data-flows specifying the data exchange between these functions.
- The *function timing and triggering diagram* (FTTD) defines the required time precedence relations between these functions.
- The *hardware structure diagram* (HSD) describes the structure of the target computer hardware.

- The *process timing and triggering diagram* (PTTD) defines, for each processor in the system, the timing and triggering properties of its set of processes and the mapping of functions into processes.
- The *process structure diagram* (PSD) defines the inter-process messages, based on the data-flow information from the DFD and the processes described in the PTTDs.
- The *communication link diagram* (CLD) defines, for each communication bus, the communication frames based on the messages defined in the PSD.
- The *process internal timing and triggering* diagram defines, for each process in the system, the time precedence relations between the functions allocated to the process.

The environment of the Aida-toolset is based on two separate tools: DoME [11] and Matlab/Simulink [9]. The use of the single tool, DoME, for the real-time domain modelling allows easy integration and exchange of data between models, given its provided facilities to define new domain-specific models. Matlab/Simulink was chosen for its good support of control design and simulation capabilities, which are also used to evaluate the implementation architecture developed. These capabilities could not be provided in the DoME environment. As shown in figure 3, the Aida-toolset consists of three major parts:

- Aidasign - The real-time system modelling environment.
- Aidalyze - The response time analysis tool, implemented in C++, performing timing analysis methods for distributed real-time systems [12].
- The interface with Matlab/Simulink - connects Aidasign to Matlab/Simulink, enabling import of Simulink data flow diagrams to Aidasign and later export to Simulink.

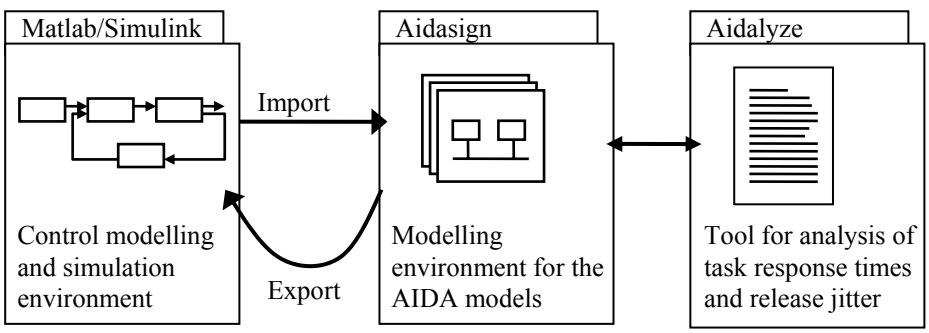

Figure 3. Architectural overview of the AIDA-toolset, highlighting its three major parts and their relations.

2.2. XILO – A Control/Scheduling Co-simulation Tool

The XILO tool supports the design of distributed real-time control systems, through the modelling and co-simulation of control functionality together with the controlled processes and the behaviour of the computer system. The co-simulation of scheduling and other implementation-related mechanisms with the control application allows the user to directly study the impact of such design decisions on the resulting system behaviour. The tool promotes interdisciplinary design by combining the views of control and computer engineering into one view.

The workflow supported by XILO is similar to that of the AIDA-toolset, visualized in figure 2, with the following differences:

- The complete set of XILO models are developed within the same environment. Hence, there is no need to perform import/export of the models between tools.
- In XILO, the analysis is only performed through the co-simulation of the application software behaviour, together with the system software and hardware behaviour.

In order to achieve the goal of a multidisciplinary modelling environment, modelling aspects were borrowed from a number of sources:

- The AIDA modelling framework [10] provided insights into the control implementation requirements needed, the component models and their parameters.
- The CODARTS method [13], as a software engineering design methodology and model, highlighted the aspects of software that need to be included.
- Data flow diagrams from the control engineering approach were used for the modelling of the application functionality.

XILO allows the modelling and simulation of the following views:

- *Application software* encompassing different functionalities in a wide variety of styles (e.g. discrete-time, even-triggered, data-flow, state machines etc.).
- *System software* including the behaviour of the operating system scheduling and inter-thread communication protocols.
- *Distributed computer systems* including communication networks and computer nodes.
- *Mechanical systems* including sensors, actuators and mechanical system dynamics.

The various views are modelled within a single hierarchy. At the top level, the hardware topology of the whole system is modelled. This hardware structure consists of three types of components: (1) The *environment* modelling the mechanical dynamics of the system including sensors and actuators; (2) *Communication Links* defining the communication protocols between computer nodes; and (3) *Computer Nodes* in which the application and system software is modelled.

Within each computer node, the software structure is defined through: (1) *Tasks* defining the application software; (2) A *task scheduler* modelling a wide range of schedulers such as event/time triggered, static/dynamic, and off-line/on-line schedulers; (3) *Operating system services* such as inter-task communication, task synchronisation and semaphores and (4) *Hardware drivers* such as communication controllers, timers, ADCs and DACs.

Finally, within each software task, the application functionality is defined as a sequence of sub-functions.

The XILO tool is based on a set of library components for the modelling of standard functionalities such as schedulers, communication mechanisms and basic operating system services. This approach allows the developers to evaluate a number of different system designs, by the simple exchange and reconfiguration of components.

The environment used to build and execute the models is Matlab/Simulink. This environment is biased towards the control engineer environment, allowing the control engineer to specify, validate and interact with the computer engineer in a familiar environment.

2.3. Integration Experiences

2.3.1. Tool Integration

In the Aida-toolset, the relationships between the various models are outlined in figure 4, where solid arrows correspond to subdiagram relationships while dashed arrows indicate import relationships between tools.

From a usability perspective, it is desired to transparently integrate the tools. Since Matlab/Simulink and DoME tools have no common mechanisms that enable direct communication between them, integration of the models is performed through import/export mechanisms. The import mechanism of the Aida-toolset allows the translation of a Simulink model into a DFD model, through a one-to-one mapping from Simulink blocks to DFD functions. Once a Simulink model has been

imported into the AIDA-toolset, additional information such as function execution times and data-flow sizes can be specified. However, to enable future export to Simulink, the model may not be otherwise modified, since the export mechanism assumes the structure of original imported Simulink model. This restriction undesirably creates a precedence relation between the models from the different tools, preventing their parallel and independent development.

In comparison, the XILO tool handles all models within a single tool and hence avoids the problem of tool integration. The adopted tool is however not necessarily optimal for software and hardware development.

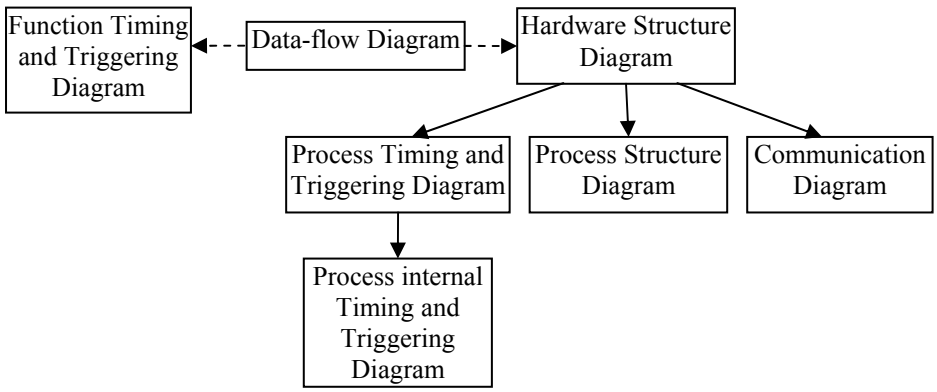

Figure 4. The structure of the models in the AIDA-toolset, where solid arrows denote subdiagram relationships while dashed arrows denote import relationships.

2.3.2. View Integration

Within the Aida-toolset models, a challenge in having the many different views is to keep the models consistent, whereby changes of information in one model are propagated to other related models that share the information. The use of a central database to manage all data shared by the models in the toolset was identified as a need to avoid the problem of inconsistency. This was not possible due to DoME limitations. Instead, the approach taken was to, for each piece of data, designate one model that is the data owner, while the other dependent models operate on data copies. Data is then automatically updated, when manually triggered by the user, and in this way regaining consistency in the model set. The major drawback of this approach is that model changes are not reflected in the whole system immediately, leading to inconsistent models in the intervals between model updates.

In the XILO tool, the mapping from the control-based functional model to the real-time implementation model is not managed, and no attempt is made to maintain the models synchronised. In addition, the XILO tool avoids the consistency problem by assuming a single model structure to fit the many implementation views of the system. This approach however conflicted with the need for different viewpoints for different disciplines, allowing developers to concentrate on specific aspects.

2.4. Integrating the Aida-toolset and XILO Tools

During their development, it was realised that the Aida-toolset and XILO tools had many properties in common, leading to the intention of integrating them. This goal was deemed feasible given that the tools are inspired by the same modelling framework [10]. The main differences between the tools are presented in table 1. The tools essentially contain the same modelling content, while they mainly focus on different analysis techniques, namely timing analysis and co-simulation. It would hence be desired to provide the two complementary approaches for system analysis based on the same modelling framework, and without the need to manually duplicate the models.

Table 1. The main differences between the AIDA-toolset and the XILO tool.

	XILO	Aida-toolset
Analysis	Co-simulation	▪ Timing analysis ▪ simulation
Tools	One tool for all disciplines	Two domain-specific tools
View modelling	Views modelled within one hierarchy	Separate models for each view.
Analysis results	Control performance	▪ Timing behaviour in terms of worst/best case response times and jitter. ▪ Control performance

However, each analysis technique requires a specific environment to work within: the Simulink simulation environment for XILO and Dome for the Aida-toolset. The challenge is to manage the modelling content in a tool-independent manner, not favouring one tool over the other, nor creating dependencies between them. This desire directed the research interest towards model content management and tool integration.

3. Goal

This thesis aims to develop a model integration and management platform that supports the multidisciplinary, model-based development of mechatronics systems. The platform should allow for the management and sharing of the product information produced by tools and disciplines throughout the development life cycle. Consequently, various analyses can be performed based on the same information set. The platform should also facilitate the communication of information between the different stakeholders, allowing any inconsistencies and conflicts to be identified and dealt with.

Two assumptions or limitations are implicit in the above inter-disciplinary integration aim: (1) A product domain focus and (2) a model-based development approach. These are further developed in the following subsections.

3.1. The Product Domain Focus

In studying the complexity of product development, Eppinger and Salminen introduce three domains of analysis: Process, product and organisation [14]. Decomposition is used within each of these domains in order to manage the development complexity. The full development process is decomposed into phases; an organisation is decomposed into teams; and a product is decomposed into sub-systems. With the separation of development into product, process and organisation domains, the interactions between these domains can be better analysed, giving a better understanding of the complexity of product development. The interactions within and between the three domains are illustrated in figure 5.

This model of product development does not explicitly take into consideration the multidisciplinary nature of certain products. It is assumed that a single product decomposition exists within the product domain. This assumption simplifies the patterns of interaction between the product structure and the remaining domains.

However, the development of multidisciplinary products adds another dimension of development complexity, whereby within each domain, the interactions between the disciplines play an important role and need to be additionally analysed.

For example, no single product structure can be assumed in a mechatronics product. Developers from the different disciplines have their own specific viewpoints of the system to be developed. That is, different description languages and analytical methods are adopted to deal with the specific concerns of the different disciplines [6]. The need to consider the product from different viewpoints leads to different product structures – or views – of the system.

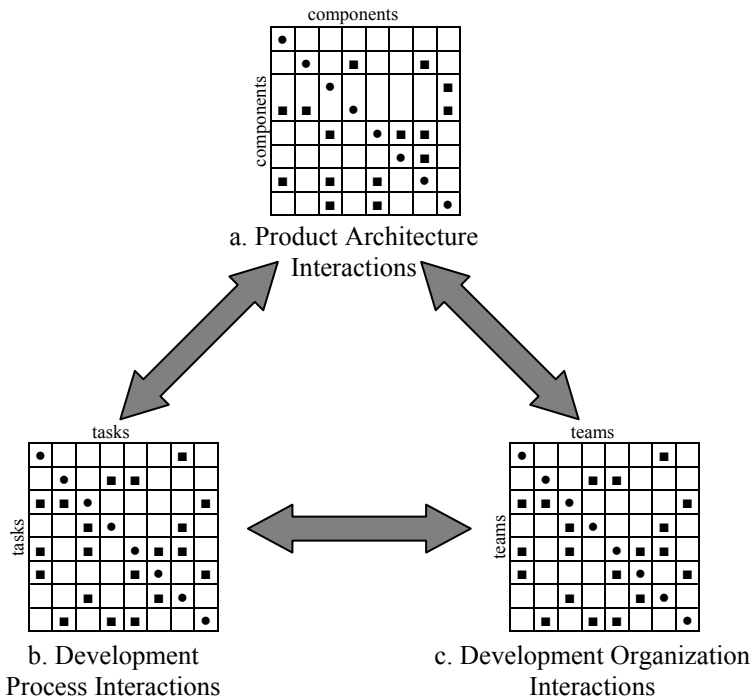

Figure 5. The patterns of interaction within each of the three domains of product development, as well as across them (Reproduced from [14]).

Within the product domain, the interactions between the various structures need to be analysed, in order to avoid inconsistencies between them. Similarly, the different disciplines may need to follow different development processes, leading to different process structures for each discipline [15]. In multidisciplinary development, this leads to multiple process structures. From the organisational perspective, the teams can no longer be viewed homogenously, as various members (or entire teams) may belong to specific disciplines, creating multiple organisation structures. As a result, the interactions between the domains can no longer be treated as suggested in [14], since the mapping is no longer between single structures within the domains.

Note that the source of different viewpoints (and hence the different structures) stems not only from the different needs of the disciplines. Within each discipline, different viewpoints may also be needed. The predominant system structure used in traditional mechanical development reflects the physical decomposition of the product into its designed components. On the other hand, software development employs many structures, which also need to be integrated. In UML [16], for example, many structures are adopted such as Class, Statechart, Use Case and Deployment models. In this general sense, a discipline can be viewed as a broader grouping of many views.

With this complex model in mind, the contribution of this thesis focuses on the interactions between the various disciplines within the product domain. We aim to integrate the various views produced by the different disciplines, ensuring the consistency of the information assumed from their various viewpoints, and providing a common basis for information flow between them.

It is acknowledged that the remaining domains cannot be simply ignored, and handling the complexity within one domain does influence the complexity in the remaining aspects. After all, the integration's final aim is to support the engineers in their development process. Nevertheless, it cannot be claimed that this thesis' contribution directly integrates the development processes assumed by the different disciplines, nor the integration of people within an organisation.

By formalising the interactions between the various product structures within the product domain, this thesis can form a step to understand the more complex interactions between the above three domains, assuming a multidisciplinary product and development.

3.2. Model-based Development

A precondition to be able to integrate and handle the interactions between the various product views is the availability of an explicit representation of these views. That is, models describing the product structures – and hence the product – are available.

Moreover, it does not suffice that the product models are simply provided. Instead, for successful development, tying the product, process and organisation domains together, the product models should be the basis of the development process within the organisation. Product models form the basis for the interactions and communication between the teams of the organisation; as well as the information flow between the development phases. Such a basis for development is here termed as model-based development.

Model-based development refers to a development approach whose activities emphasise the use of models, tools and analysis techniques for the documentation, communication and analysis of decisions taken at each stage of the development lifecycle. Models can take many forms such as physical prototypes, graphical and textual models. It is essential however that the models contain sufficient and consistent information about the system, allowing reproducible and reliable analysis of specific properties to be performed. In model-based development, analysis plays the critical role of ensuring that the models being built - hence the design decisions being taken – are consistent and satisfy the system requirements.

Within a given discipline, model-based development is commonly used, such as the use of CAD tools in mechanical engineering. In the maturing software engineering domain, model-based development is gaining acceptance. The popularity of modelling languages such as UML is an indication of this trend.

In multidisciplinary model-based development, several viewpoints of the system are formed by the different disciplines. This leads to several models, representing the different product structures produced. In the integration of these models, the discipline-specific description languages and analysis methods used to model these structures should be preserved. Proper model integration may become a strong basis of communication between engineers of different disciplines.

This thesis suggests an approach in which the integration of models from the various design domains is also model-based, ensuring the explicit documentation of the interactions between the product views. The state of practice of social integration [17], where informal communication between engineers tries to ensure consistency, is not desired.

Given the recent establishment of the model-based development in certain disciplines such as software engineering, the sensibility of this assumption can be questioned. According to Encyclopædia Britannica [18], ‘engineering’ is defined as the ‘professional art of applying science to the optimum conversion of the resources of nature to the uses of humankind’. Given this definition, one can reverse the question and wonder how the application of the sciences can be validly performed during engineering activities without access to explicit and reproducible information. Product information and design decisions need to be explicitly and unambiguously documented for their communication between engineers, and to become a basis onto which scientific analysis can be performed. Engineering is a combination of craftsmanship and scientific exploration; and model-based development is a basic requirement for the latter to be possible. In other words, in order for software development to change from an art to becoming an engineering discipline, it ought to become model-based.

4. Approach

The aim of the integration platform is to integrate the different models used to represent the structures or views from the various development disciplines. In the development of large and complex products, an organisation normally adopts some kind of product management tools in order to manage the large amount of documents storing these models. For example, the development of software-intensive products relies on Software Configuration Management (SCM) systems, while mechanical system development uses Product Data Management (PDM) systems. The need to obtain consistent access to the documents storing the models leads to the necessity to coordinate the intended integration platform with these management tools.

In multi-disciplinary product development, a number of these management environments come into simultaneous use. This is necessary since developers from each discipline require the specific support provided by its corresponding management system. Integrating these environments becomes essential for the successful integration of the efforts of all disciplines involved, considering the central role they take in controlling the development process as well as facilitating the communication between developers.

In summary, a model integration platform integrating different development tools needs to be itself integrated with the management tools, which in turn need to be integrated with each other. The various integration needs are illustrated in figure 6.

Another approach to the problem is to step back and treat the view integration problem as part of the management problem already covered by PDM/SCM systems. Model integration is treated as another functionality that can be augmented to the conventionally expected functionalities of management tools. This approach is illustrated in figure 7.

In one sense, incorporating the management tools expands the integration problem. However, expanding the problem domain provides a better fit of the view integration problem. Much can be borrowed from the PDM/SCM integration efforts such as the work suggested in [15] and [19]. In addition, by absorbing the

management tools into the platform, a smaller number of tools need to be integrated.

Problem simplification can also be claimed given the assumption of model-based development. As argued in section 5.3 (Paper-C), the integration of PDM/SCM is considered more feasible with this assumption, suggesting a unified platform that generically handles models from all disciplines. Based on this platform, the integration of the models from the different disciplines is made more feasible.

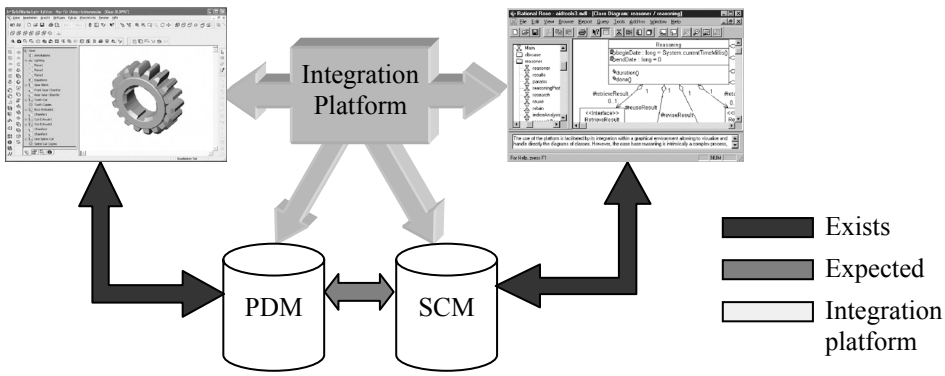

Figure 6. The integration needs of the various development and management tools for mechatronics systems.

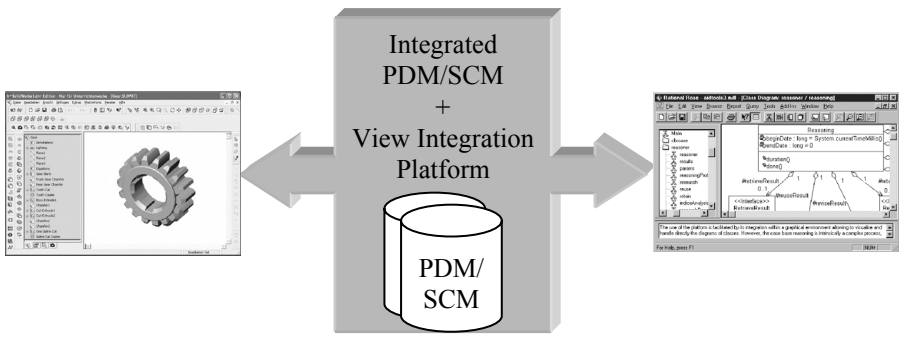

Figure 7. An integration approach treating view integration as part of the management systems.

The integration problem is reduced to that of integrating PDM and SCM systems, plus providing integration functionality based on the integrated solution. Within the context of figure 5, the approach not only contributes to the integration of the disciplines within the product domain by integrating their views, but by also contributing to the integration of the management facilities such as process

control, workflow control, user management, etc. These facilities are used in the process and organisational domains, leading to a better alignment of the three domains.

4.1. Model and Tool Integration

Model integration is made a lot easier if one assumes a single tool that fully supports the development of all involved views. Model management and integration can thus be provided within the tool implementation itself. While this may be desired, experience shows that no such silver bullet can be provided. Our conviction is that no matter how large and encompassing modelling tools get, one will never reach the point when a single tool will meet all the needs of a multidisciplinary development process in any organisation. As a consequence, the need to integrate model information between the tools that act on this information will always exist.

No tool in the tool-set should take a predominant role, to which all other tools integrate. Such an approach creates a dependency on that tool, and peripheral tools can only be integrated indirectly. Instead, a central platform is suggested to which tools are connected. It is through this platform that communication between tools, and the integration of their models, occurs. Naturally, dependencies are created to the integration platform, which is however expected to be more stable, as suggested in section 4.3.

4.2. Platform Requirements

In summary, the integration platform should support the following needs:

- **Support for discipline-specific tools** – It should be possible to integrate different kinds of tools from the various disciplines, recognising that different organisations will assume a different toolset.
- **Data sharing and view integration** – A tool integration mechanism should manage the duplication of information between tools, synchronizing and maintaining its consistency. In addition, having chosen a specific set of tools, certain design information ends up in between tools. This information specifies a relationship between the different views (inter-view information). Good integration mechanisms should permit the specifications of such cross-view information, reflecting points of interaction at which the respective stakeholders need to communicate.
- **Model management** – includes functionalities such as the storage of models, handling of versions and variants of models, change request management,

process/workflow management as well as support for geographically distributed development. Support for discipline-specific functionality should also be provided such as build management for software development. An integration platform ought to provide these functionalities centrally for all tools that it integrates.

4.3. Integration Cases

Caution should be taken when adopting a given integration solution, given the central role such a platform assumes in an organisation, and the dependencies it creates between developers. In addition, an integration platform is expected to outlive the many tools it integrates. While metrics such as the Return on Investment (ROI) are developed to justify investments in central systems like PDM and SCM [20], no such metrics are necessary in adopting tools such as compilers or editors, which may be used locally within an organisation and are replaced relatively more easily over time.

For these reasons, a stable, long-lasting and universal integration solution, which can anticipate future changes in tools, is to be expected.

This stability is threatened by factors such as the fast growth in modelling languages and tools, specifically for the maturing software engineering discipline. On the other hand, partial standard efforts such as the MOF modelling standard [21], formatting standards such as XML [22], and basic communication mechanisms such as CORBA [23] and COM [24], provide a valuable foundation. The appearance of the STEP [25] standard within the mechanical engineering discipline is historical evidence that such efforts are possible.

In this thesis, it is recognised that achieving the stability expected of an integration platform is very much a standardisation effort. For this reason, focus is instead placed on two cases of integration techniques to cover each of the main needs specified above: view integration and model management.

Concerning view integration, the integration of the system functional view to the hardware architecture view, through the allocation of functions to hardware components, is investigated. With each view related to a different discipline, this example highlights the multidisciplinary problem. Further details are discussed in section 5.2 and Paper-B.

Concerning model management, a generic version management functionality of models is investigated. While version control is needed in both the mechanical and software disciplines, the functionality differs between SCM and PDM systems. This allows us to investigate how far such mechanisms can be aligned between the disciplines. Version control is also critical since it will put to the test the other

crucial management functionalities of any common management system such as the possibility of having a common product structure and data representation. Further details are discussed in section 5.4 and Paper-D.

Finally, to satisfy the need to support discipline-specific tools, these cases need to be dealt with assuming different modelling tools.

5. Summary of Appended Papers

This section provides a summary of the appended papers of this thesis. The combination of these papers provides a good description of the tool integration platform.

The reader is advised to read these papers before proceeding with the remaining chapters of the thesis.

5.1. Paper A - A Tool Integration Platform for Multi-Disciplinary Development

This paper presents the architecture for the Model Data Management (MDM) platform that aims to satisfy the needs for tool and model integration presented in section 4.2. MDM generically manages and integrates models from the various tools used in the development of mechatronics products.

The platform aims to provide generic model management functionalities including supporting the storage of models, handling of versions and variants of models, access control, change request management, process/workflow management as well as support for geographically distributed development. This is viewed as a unification of the management functionalities typically provided by the discipline-specific PDM and SCM systems traditionally used in the hardware and software disciplines respectively. The model-based approach to data management unifies the software and hardware disciplines by unifying the kinds of objects it manages – models. The model-based management functionalities and the need to interrelate the internal model contents require that the platform manages the fine-grained details of each model from the integrated tools.

The architecture supports the decoupling of the modelling tools from the MDM platform, permitting an open architecture where various tools can be integrated as desired. This is made possible through the adaption layer that maps the tool-specific format and meta-model, used internally by the tool to manage its model data, to the generic format and meta-model of the platform.

The proposed architecture explores the idea of building on existing technologies from the more mature discipline of mechanical engineering, as well as borrowing advanced functionalities from the software domain. MDM is built based on a configurable PDM system. PDM is adopted due to its maturity and ability to define information models, with a high level query language to access and modify the model data in the repository. In addition, it is envisaged that the development of the remaining MDM functionalities is made easier given the already developed functionalities of PDM such as the support for distributed development, change management, workflow control, etc. At the same time, the version control functionality borrows ideas from the fine-grained version control algorithms in the software discipline.

Model management functionalities are illustrated through the implementation of the version control algorithm of Paper-D. In addition, model integration techniques are provided, where model content can be shared across different tools. This is illustrated in the partial implementation of the view integration mechanisms proposed in Paper-B.

5.2. Paper B - Towards a Multi-View Modelling Environment for Mechatronics Systems

The paper presents an approach to multi-view modelling and integration which systematically integrates the two generally accepted complexity reduction techniques of multi-view and hierarchical decomposition. The approach defines how inter-view relationships can be used to tightly interweave the views' hierarchies.

Through the use of a case study, model integration is investigated for the allocation of system functions onto the implementing hardware architecture. The resulting approach maintains the principle of hierarchical design within, as well as between the views, where allocation can be performed at arbitrary levels across the hardware and function hierarchies. The proposed approach promotes the independent development of the views, allowing developers from each discipline to work concurrently, yet providing support for a holistic view.

Mechanisms are defined to ensure the completeness and correctness of any inter-view design decisions made, as well as, to perform cross-view keyfigure analyses. The principle that a part of the complete system is a system of its own, with its own set of views is reinforced, with the possibilities to perform cross-view analysis on the complete system as well as its individual parts.

The feasibility of the inter-view mechanisms is investigated through the implementation of a prototype tool, in which views, as well as, inter-view design

information and analysis, could be performed. In addition, a partial implementation of the approach is developed based on the MDM platform of Paper-A. Through a generic inter-view association mechanism, the model data from different tools can be interrelated. This acknowledges the need for the different views to be modelled using domain-specific tools. The integration platform takes a centralisation role in which the inter-tool information is managed and stored.

The paper also presents the meta-meta-model of the MDM platform. A simple meta-meta-model is adopted, allowing focus to be placed on the view integration mechanisms and the management functionalities of interest.

5.3. Paper C - Model Data Management – Towards a common solution for PDM/SCM systems

This paper investigates the effect of adopting model-based development in software engineering in bringing the discipline closer to the hardware engineering discipline and permitting a tighter integration of their management systems. The investigation considers the three crucial factors for a successful integration: tools and technologies, processes, and people [26].

It is argued that, as software development becomes increasingly model-based, its needs become closer to those of hardware development. In particular, the process management and information modelling functionalities expected of SCM systems come closer to those provided by PDM systems for hardware development. This leads the way for a more effective integrated management platform satisfying the needs of both disciplines using a common set of mechanisms. The model-based approach to data management unifies the disciplines by unifying the kind of objects it manages – models. Management functionalities deal with models and their internal contents as central entities, transparent of the file structure used to store them.

The MDM platform, presented in Paper-A, provides a basis for the desired common management functionalities, by generically handling different kinds of models produced from a set of different tools and disciplines. To illustrate the suggested common management solution, a model-based version management functionality is implemented, as presented in Paper-D.

5.4. Paper D - The Version Control Algorithm Implementation in the Model Data Management (MDM) Platform

In this paper, a simple model version control functionality (MVC) was implemented, in order to exemplify the PDM/SCM integration approach suggested in Paper-C, and test its feasibility using the MDM platform of Paper-A.

While version control is needed in both the mechanical and software disciplines, the functionality differs between SCM and PDM systems. This allows us to investigate how far such mechanisms can be aligned between the disciplines. Version control is most fundamental and best validates the MDM approach since it will put to the test the other crucial PDM/SCM integration factors such as the possibility of having a common product structure and data representation. Naturally, a full validation of the approach needs to investigate the feasibility of the remaining management functionalities using the model-based approach.

MVC provides mechanisms that allow a user to save and extract any part of the system model through check-in and check-out operations respectively. This permits stakeholders to perform design activities in terms of models, where they can organise, share and modify their models, transparent to the underlying file structure.

The algorithm generically supports the fine-grained versioning of any model that can be mapped to the meta-meta-model assumed in the platform, and presented in Paper-B. In the current implementation, Data Flow Diagram (DFD) [27] models from the Matlab/Simulink tool and Hardware Structure Diagram models [7] in the Dome tool are handled.

6. A Survey of Modelling and Integration Approaches

A survey of current approaches for the modelling of embedded computer control systems was performed as part of this research project [28]. A short summary of this study is presented in this section, together with a complementary survey of representative tool integration approaches. The study was initiated to appreciate the various flavours of modelling approaches available, and understand the differences between them. The common patterns found between the approaches formed a good basis for the definition of the meta-meta-model suggested in the MDM platform (Paper-B). The tool integration solutions suggested by these approaches, and their limitations, also became a good motivation for further research on model and tool integration.

The survey aimed to study 'what' each approach models, with less focus on the details of 'how' this is performed. For this purpose, a framework for characterizing, comprehending and comparing the different approaches was developed, focusing on the modelling content. As illustrated in Figure 8, the framework combines generic modelling concepts with multiple iterations from the evaluation of twelve modelling approaches covering different levels of design and disciplines. This evolved and stabilised the framework, consolidating more precisely the defined factors.

A modelling approach refers to any support technique or solution provided for the design of embedded computer control systems, such as computer tools, languages and standards. The choice of approaches covers different application domains, disciplines and levels of design, ensuring that a broad collection of modelling features are covered.

Twelve approaches have been evaluated based on published materials from the respective developers. ACME [29], Wright [30], UniCon [31] and Rapide [32] are software Architecture Description Languages (ADL). Lustre [33] and MAST [34] have a computer science origin with formal methods and scheduling theory background respectively. VCC [35] is an approach from the automotive industry. Orccad [36], Giotto [37] and MetaH [38] are domain-specific approaches that aim

at control applications to be implemented on computer systems. Finally, both Ptolemy [39] and SDL [40] focus on the high-level specification of the system, and less on implementation details.

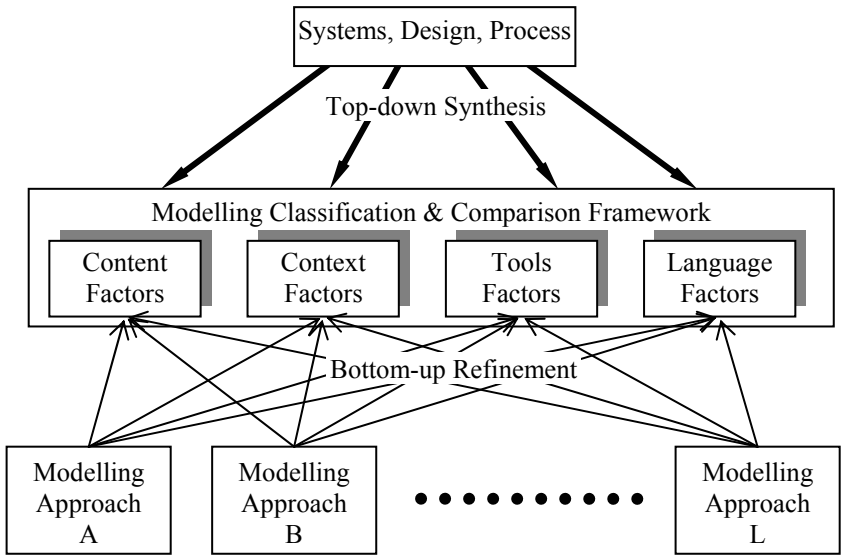

Figure 8. Technique for defining the framework – Top-down synthesis and bottom-up refinement

6.1. Comparison Framework

To compare different modelling approaches, both the model contents, as well as the design and analysis context within which the models are used, need to be taken into consideration. In the comparison framework, this is formulated using three groups of comparison factors: *modelling content*, *design context* and *analysis context*. These factors are summarized in figure 9.

The content factors aim to identify the various system aspects that can be modelled by a particular modelling approach. In this framework, a model is seen as consisting of a set of abstractions that represent real system entities. The abstractions may be classified into a set of common types. Furthermore, there exist different types of relationships between the different abstractions, such as communication between abstractions and decomposition of one abstraction into a set of other types of abstractions. Following this view on models, the set of *abstraction types*, the *properties* that define them, and the *inter-abstraction relation types* that may exist in any modelling approach are identified.

To facilitate the comparison, abstraction types, their properties and relation types most relevant for embedded control systems are predefined in the framework, as listed in figure 9. The content classification forms a common basis upon which it is possible to organise and compare the content support provided by each modelling approach.

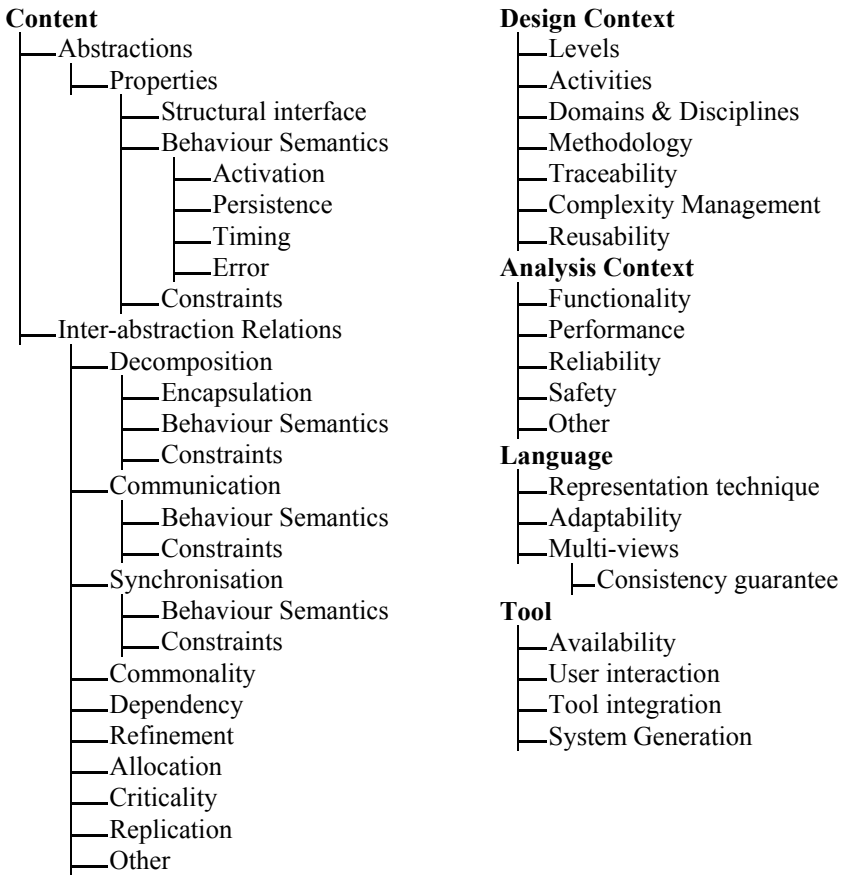

Figure 9. Comparison framework structure and factors

Within the design context, the *level of design* at which the content is used by the approach is of most interest. For comparison, four general design steps are defined, ranging from implementation-independent specifications, towards the final solution description: *functional design*, *architectural design*, *medium-level design*, and *detailed design*.

Within the analysis context, it is interesting to study the types of analysis that can be performed given the modelling content provided by the approach. For embedded computer control systems, relevant analysis types include: *functionality*, *performance*, *reliability* and *safety* analysis.

Two other groups of factors are also handled in the framework: *language* and *tool*. The former deals with the techniques and rules adopted by a modelling approach for representing its content. Even though two approaches have the same content, they may differ in the way this content is handled, used and represented in the models. Finally, the tool factors attempt to identify the computer-aided techniques and facilities available for manipulating, managing and verifying the models.

6.2. Comparison

The major part of this work was in the surveying and analysis of the modelling content of the approaches. A detailed discussion and comparison of the content can be found in the original study [28]. The procedure used to acquire the comparison framework highlights the common features between the studied approaches. Abstractions such as *communication* and *software* types; properties such as *timing*; and inter-abstraction relations such as *decomposition*, *communication*, *refinement* and *allocation* are most common between the studied approaches.

Furthermore, in structuring the modelling content, common techniques are found between the modelling approaches in order to absorb the complexity of the system being modelled. The major identifiable mechanisms for complexity management are: The widely adopted hierarchical decomposition, the use of domain-specific terminology and concepts, the repeated use of a few central concepts, good language and tool support, the division of content into multiple views, and commonality mechanisms such as typing and specialisation/generalisation.

Through the analysis of the modelling content, the design levels addressed by each modelling approach are determined, as illustrated in figure 10. In addition, table 2 presents a summary of the *available* and *possible* analysis techniques provided by each approach. Available analysis techniques are those explicitly identified and supported by an approach. Possible techniques are those that can be potentially performed, given the content supported by an approach.

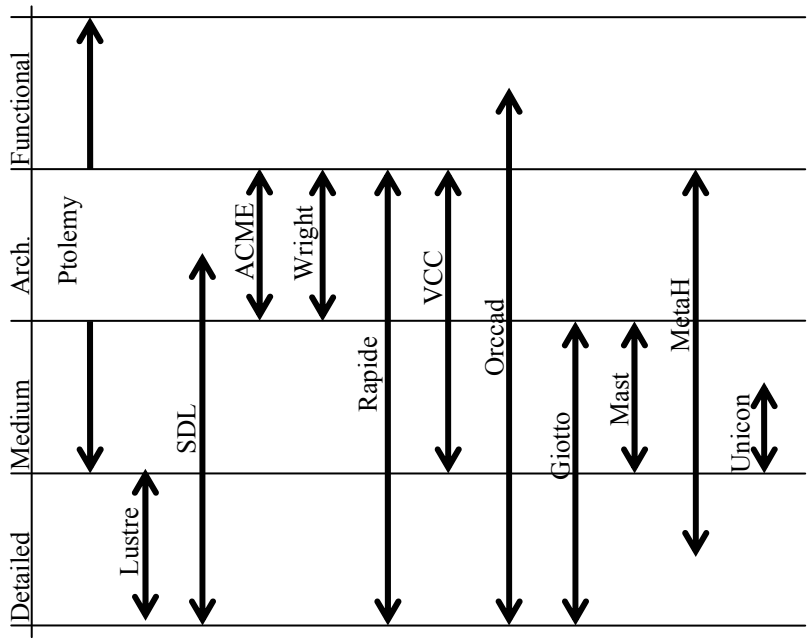

Figure 10. Design levels focused on by each modelling approach.

Table 2. Summary of available (√) and possible (+) analysis techniques

	Functionality		Performance			Reliability	Safety
	Simulation	Model Checking	Simulation	Model Checking	Timing		
Ptolemy	√		√				
Lustre	+	√	+	+			
SDL	+		+				
Acme							
Wright	+	√					
Rapide	√	√	+	√	+		
VCC	√		√		+		
Orccad	√	√	√	+	√		
Giotto	√	√	√		√		
MAST			√		√		
MetaH	+		+		√	√	
Unicon	+		+		√		

Concerning tool integration capabilities, the modelling approaches tend to integrate other tools in order to cover certain aspects that are weak or not covered

in the original approach. Compared to integration platforms (section 6.3), such integration efforts tend to be ad-hoc, implemented to meet the current needs of the approach. For example, MetaH is integrated with ControlH for the functional description of its subprograms, and Giotto uses Simulink for graphical representations. Certain approaches become quite dependent on this integration to be usable. For example, Wright needs to have a CSP checker to perform any kind of analysis. On the other hand, MetaH can still be operable without the use of ControlH.

Much overlap exists between the content covered by the approaches. This is specifically the case for approaches that attempt to cover similar activities and analysis techniques, at the same level of design. The similarities between the ADL languages, where focus is mainly placed on software modelling at the architectural level, is a typical example. In these approaches, the main abstractions covered are components, connectors and configurations used to model system software. It can be argued that content overlap between approaches is an indication of integration potential between them. The challenge remains to coordinate the remaining content that does not entirely overlap.

Approaches covering the same activities at the same level of design can be used interchangeably. Integrating such approaches might be of interest when the different approaches provide complementary functionalities or analysis techniques. For example, the ACME ADL might be desired to use for its possibilities for generic specifications, while Wright provides analysis possibilities through simulation and model checking.

In addition, approaches covering different activities, or different design levels would be of interest to integrate to cover a wider range of design levels and activities. For example, it may be of interest to integrate an ADL such as Rapide with Ptolemy. While the latter provides higher level functional descriptions, the former can be suitable for the architectural level of design. The model of computation provided in Rapide (timed-posets) can also be complemented by the variety of models of computations provided by Ptolemy.

An abundance of modelling languages and approaches that target various aspects of system development exists. The union of these approaches may cover all that can be desired. The challenge remains however in providing such a union. A necessary component of any such integration effort is the integration of their modelling content. Ad-hoc integration, as experienced in the studied approaches, creates undesirable dependencies to the modelling tool. Instead, as discussed in section 4.1, a platform addressing the integration of tools should be used. The next subsection surveys a number of such platforms.

6.3. Tool Integration Approaches

This survey is based on the study of seven tool integration approaches: Cheops [41], Eclipse [42], Fujaba [43], GeneralStore [44], IDM [45], IMPROVE [46] and Toolnet [47].

Tool integration can be divided into two general categories: *data integration* and *control integration*. The former focuses on relating the model data produced by the different tools. On the other hand, control integration deals with tool activities such as integrating the services or functionalities provided by the tools, providing a common look and feel across the tools, controlling the workflow between the tools, managing tool interactions, etc. A typical example of control integration is the Eclipse platform for software development. Eclipse provides a plug-in based framework to create, integrate and utilize software tools. The plug-in mechanism is used to realise the services of the integrated tools, and through which tools can interact and request services from each other. However, any files and data items produced are managed internally by the integrated tools and are beyond the scope of the platform. Naturally, certain tools such as Fujaba take into consideration both aspects of integration. This section focuses mostly on data integration, given its relevance for the issues discussed in this research.

Two different needs for data integration can be identified: the integration of models covering different components of the complete system - *component model integration*; and the integration of models covering different views of the same system – *view integration*. These needs lead to different integration solutions.

The challenge in component model integration comes when the different components are modelled using different models of computation, such as the time-continuous or time-discrete models of computation. In this case, the heterogeneous models need to be appropriately coupled at their interfaces to form a complete model. From the surveyed approaches, GeneralStore and Cheops focus on component model integration of software systems and mathematical models of chemical plants respectively. Both perform component model integration through the transformation of the heterogeneous models to a common internal representation, based on a single meta-model. However, the common meta-model in GeneralStore is only used to store the models, while the integrated system model consists of the original models, together with wrapper elements generated based on the specified interface definitions. Cheops, on the other hand, integrates the transformed models into a complete system model, on which a common numerical analysis method can be used. With both approaches, the resulting complete model can be used for the co-simulation of the integrated components.

In dealing with *view integration*, the models generally need to be integrated at a finer level of detail, associating specific content within the models to each other.

In this survey, four integration approaches deal with view integration: Fujaba, IDM, IMPROVE and Toolnet. Different types of relations can be setup either manually or automatically between the models. As identified in Toolnet, two general categories of relations can be defined: *general dependencies* and *data duplication*. Once the relations are setup, the most common analysis support provided as part of the integration platforms is that of consistency checking of model data between the tools, as provided in Fujaba, IMPROVE and Toolnet. The approaches also provide mechanisms to repair any inconsistencies found during the analysis. In certain cases, the integrated models deal with the same or close aspects of the system being modelled. In other words, much duplicated or similar data is found in the heterogeneous models. In such cases, a transformation between the different model types can also be performed. Transformation facilities are provided by Fujaba, IDM and IMPROVE.

Very few platforms consider the issue of data management. In Eclipse, such support is gained through the integration of the CVS [48] versioning tool. Considering that Eclipse does not perform data integration, CVS is simply treated identically to any other development tool. Such integration is similar to that illustrated in figure 6. The management tool manages the documents at the coarse file level, without dealing directly with the fine-grained model data. From the studied platforms, GeneralStore is the only platform to provide management functionalities such as user authentication, transaction management and fine-grained object versioning. This approach is closer to that illustrated in figure 7, but not entirely satisfactory, since the need to integrate the platform with existing PDM/SCM systems remains.

The general trend in the implementation of the platforms focusing on data integration is to assume a centralised data storage system, to which tools are integrated through a wrapper or a plug-in. The wrapper provides the necessary abstraction from the tool-specific implementation and formats, and in this way providing a uniform interface to the platform. The storage system can be a database management system such as for GeneralStore, or a simple file as in IMPROVE.

With the exception of GeneralStore, the repository is not generally used to manage the complete set of model data from the tools. Instead, the platforms only handle reference objects to the model data and additional integration information such as relations between the references objects and relevant metadata. Model data is expected to be managed and stored by its producing integrated tool. The strongest motivation for not storing modelling data is to avoid the duplication of information in the modelling tools as well as platform. Such an approach however limits the possibility to provide the necessary management functionalities, as advocated in this thesis.

7. Industrial Case Studies

This section presents a summary of two industrial case studies carried out at Scania, as part of this research project. As briefly discussed in section 7.3, the case studies were used as a source of inspiration, as well as to evaluate some of the ideas presented in this thesis. The first case study aimed at a quantitative analysis of architecture designs based on a set of keyfigures that reflect important quality attributes. Given exposure to the challenges faced during this case study, a second case study was initiated to deal with an analysis of the function modelling capabilities at the organisation, together with a recommendation for future improvements. A more complete description of these case studies can be found in [49] and [50] respectively.

7.1. Keyfigure Analysis Case Study

During the early architectural design of a truck, architects face the challenge of choosing the Electrical/Electronics (EE) architecture, onto which the system functionality is to be implemented. It is desired to quantitatively analyse and compare different architecture designs, taking into consideration and optimising important design keyfigures such as the resulting system weight and costs. The evaluation needs to perform trade-offs between a set of keyfigures, taking into consideration a range of product variants.

For this end, a keyfigure tool supporting the architecture design of allocating functions to control units, as well as the quantitative calculation and weighting of selected keyfigures, was developed. The architecture of the developed keyfigure tool, together with its different data sources is shown in figure 11.

A central database was used to collect information about the functional specifications, communication signals and components set of the product variants. Data was collected from a range of dispersed sources in the organisation. A core source of information was the Function and Component Databases that needed to be manually manipulated to suite the needs of the study. Another important source of implementation data was the Communication Database used to deduce the communication needs between functions, the decomposition of functions into

subfunctions, and their allocation to electronic units. In addition, specific product variants were imported from proprietary product identification files, in which variants were defined as a selection of a set of user functions.

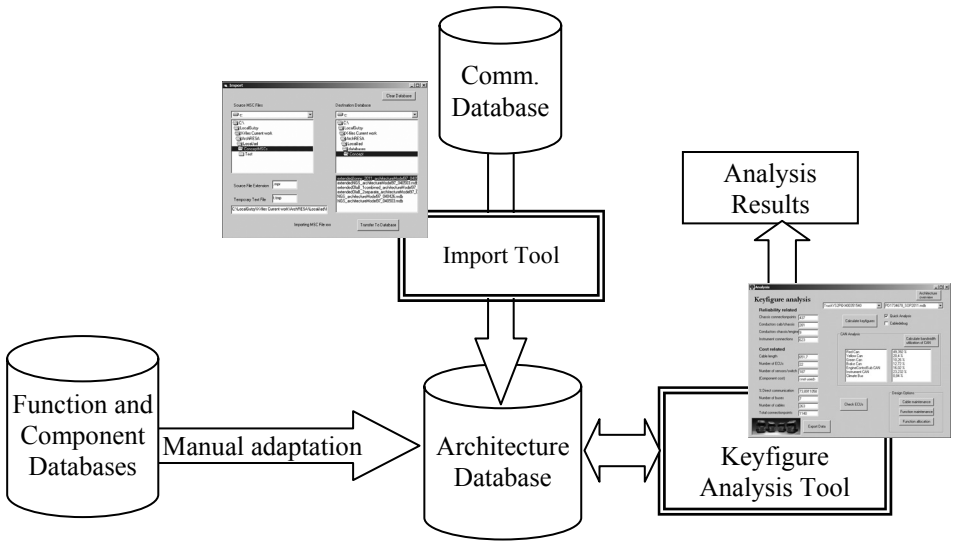

Figure 11. Tool architecture for the keyfigure calculation tool

A wide range of keyfigures (See table 3) was selected based on four important product aspects: Dependability, cost-efficiency, modularity and performance. An example keyfigure is the number of cable connection points. This keyfigure relates to the dependability aspect, since connections are an important source of faults and failures in embedded automotive control systems. The aim is to reduce the number of connection points in difficult environments, through the appropriate positioning of control units. The length of cables and number of components are other easily analyzable keyfigures relating to cost-efficiency.

In the study, the specification of the functionality and the hardware architecture were separated, creating two views of the system. The separation facilitated the possibility to perform multiple allocation strategies without needing to re-model the system functionality. The functionality was modelled as function blocks linked by communication links. The implementation was modelled as electronic units linked by cables. The electronic units include sensors, actuators and electronic control units (ECU). The different views are then interrelated once the functional allocation onto the hardware is defined, where function blocks are associated to electronic units and communication links are associated to one or more cables.

Table 3. The keyfigures considered in the quantitative architectural design analysis.

▪ Number of connection points ▪ Cable length ▪ Connections in bad environment ▪ Number of cables in difficult passages ▪ Number of ECUs ▪ Number of sensors ▪ Weight ▪ Number of units developed in-house	▪ Number of suppliers/sensors ▪ Modularity ▪ Number of messages through gateway ▪ Number of Mission critical units ▪ Processor utilization ▪ Gateway utilization ▪ Number of suppliers/ECU	▪ Number of mission critical connections ▪ Number of part numbers ▪ Number of distributed functions ▪ Number of widely distributed functions ▪ Number of pins/ECU ▪ Component cost ▪ Bandwidth utilization

Once the functional allocation is performed, an analysis tool allowed the keyfigure calculations for a specific product variant and system architecture. A screenshot of the main analysis window, highlighting some of the measured keyfigures, is provided in figure 12. Using this tool, it was possible to quickly compare alternative architectures and find the weaknesses and strengths of the alternatives as indicated by quantitative keyfigures.

7.2. Function Modelling To Improve Software Documentation

Among the many distributed sources of information within the Scania organisation, the current functional documentation of the EE (Electrical/Electronics) system is mainly based on three core documents:

- User Function Specification (UFS) - specifies a *User Function*, which is a specific functionality to be implemented in a vehicle, implemented over more than one system.
- System Description (SD) - specifies a *System*, describing the physical entities onto which User Functions are implemented such as sensors, actuators and ECU-hardware units.
- Message sequence charts (MSC) - Specifies a *Scenario* describing a specific sequence of events for a given User Function. Multiple scenarios are specified for each User Function and these are grouped into *Use Cases*.

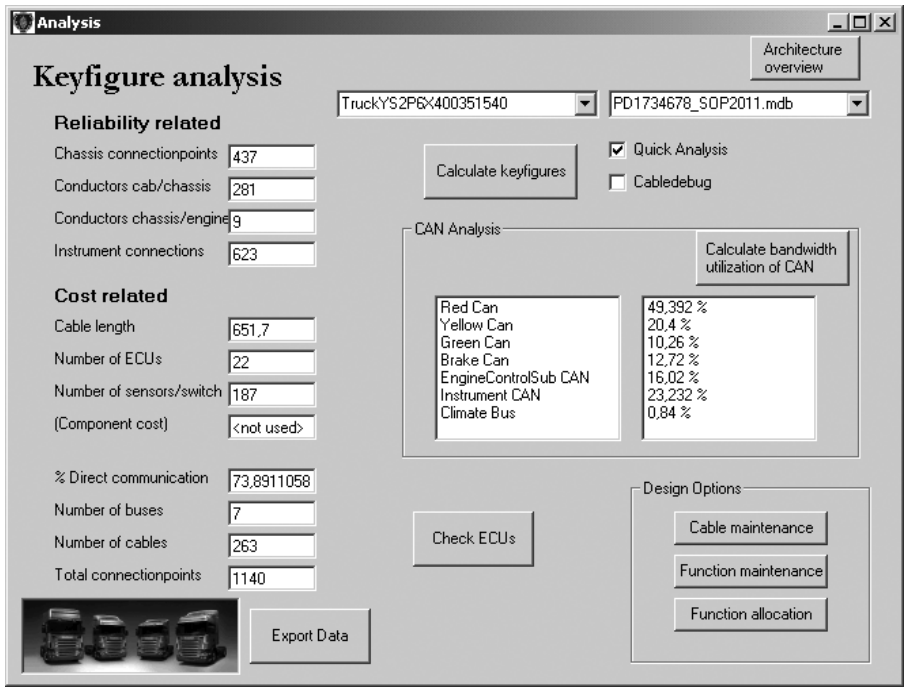

Figure 12. Screenshot of architecture scorecard tool

In a preliminary internal study, a range of problems were identified with the current functional documentation, namely:

- **Document inconsistencies** - Text editors are used for the documentation, where references to other documents are hard-coded, with no mechanisms to update these links upon changes.
- **Incomplete information** – A scenario-based behaviour description of the functions is used, leading to an incomplete specification. In addition, functionality to be completely implemented within one hardware unit is not necessarily documented.
- **No user function overview** – No documentation currently provides a general overview of functions, focusing on the end-user aspects.
- **Unclear dependencies** - For a particular user function, the distribution of function parts onto systems is implicit.
- **Function and Implementation mixed-up** - The current User Function Specification document contains information about both function and implementation, limiting the possibility of function reuse given

implementation changes, as well as blurring the boundaries between the roles of the system owner and the function owner.

A brief investigation to deal with these problems was performed. The study resulted in an information model and a documentation approach to function specifications. The proposed techniques were evaluated through the specification of three functions of varying complexity.

7.2.1. Information Modelling

The proposed information model to handle the document contents is illustrated in Figure 13. The information model is broken down into different views that group entities together targeting particular aspects of the system. Roles were also identified to control access to the information model entities.

The three main views of the system are the *Functional view*, *Software view* and *Hardware view*. A common pattern exists between each of these views, specifically: (1) The hierarchical decomposition used within each view, for managing the size and complexity of the system description. This highlights that there exists no single dominating product structure, and each view describes the system from a specific perspective. (2) The definition of entity interface through which the entity interacts with its external environment.

- Function View - The main object in this view is the *Function*, with two sub-types: *PartFunction* and *Variable*. A PartFunction object designates certain functionality that given a certain input, produces a certain output. A Variable object designates a transportation link that manages certain data internally and provides access to this data to connected PartFunctions. A Function can be decomposed into a set of (sub-)Functions, forming a hierarchical product structure. The interface definition of a Function is defined by a set of *ports*, where a port acts as a placeholder for a subset of its object's externally accessible properties.
- Software View - Similar to the Function view, the main object in this view is the *SoftwarePart*, with two sub-types: *SoftwareComponent* and *Data*. A SoftwareComponent object designates a sourcecode module that given a certain input, produces a certain output. A Data object designates a data storage facility that manages certain data internally and provides access to this data to connected SoftwareComponents. A SoftwarePart can also be decomposed into a set of (sub-)SoftwareParts, forming a hierarchical product structure. The interface definition of a SoftwarePart is defined by a set of *SoftwarePorts*, where a SoftwarePort designates a certain internal data item that is externally accessible to other SoftwareParts.

- Hardware View - Similar to the Function view, the main object in this view is the *HardwarePart*, with two sub-types: *HardwareComponent* and *Cable*. A HardwareComponent object designates a physical block having geometrical dimensions and a position. A Cable object designates a single cable with a certain geometrical path. A HardwarePart can also be decomposed into a set of (sub-)HardwareParts, forming a hierarchical product structure. The interface definition of a HardwarePart is defined by a set of *pins*, where a Pin designates a spatial location at which the HardwarePart can be connected to other HardwareParts.

In addition, the *User Function* view is a special view targeting the product user, and hence focuses on structuring the product functionality from the user perspective. A complete system is described using a network of hierarchically decomposed Functions. However, from the user perspective, certain sets of Functions form a clear and valuable contribution that the user can relate to. Such a set is managed in the information model using the *UserFunction* object. Ignoring Function variants for the moment, a UserFunction is a grouping of Function objects, forming a fully defined specific functionality (just like the hierarchical composition of functions into PartFunctions). It is important to note that a Function object does not exclusively belong to a single UserFunction. Certain functionality, such a 'speed sensing', provides services that can be shared by many UserFunctions. Such functions are a good indication of the interaction and dependencies between user functionalities.

Finally, given the importance of product configurations, each of the above views is further described using a specific variant view: *FunctionVariant*, *SoftwareVariant* and *HardwareVariant* views, describing variants of functionalities, software realizations of functionality and the hardware platform in which the software realizations are allocated respectively. Again, a pattern can be found in representing these three variant needs, and in their relation to other objects in the information model.

- The FunctionVariant is used to represent variations for a particular user functionality. A UserFunction is a grouping of FunctionVariants that provide similar or competing functionality from which the user can choose. A FunctionVariant object is in turn a grouping of Function objects, forming a fully defined specific functionality. It is important to note that a Function object does not exclusively belong to a single FunctionVariant, since certain functionality can be a common part among the various variants of a given UserFunction.

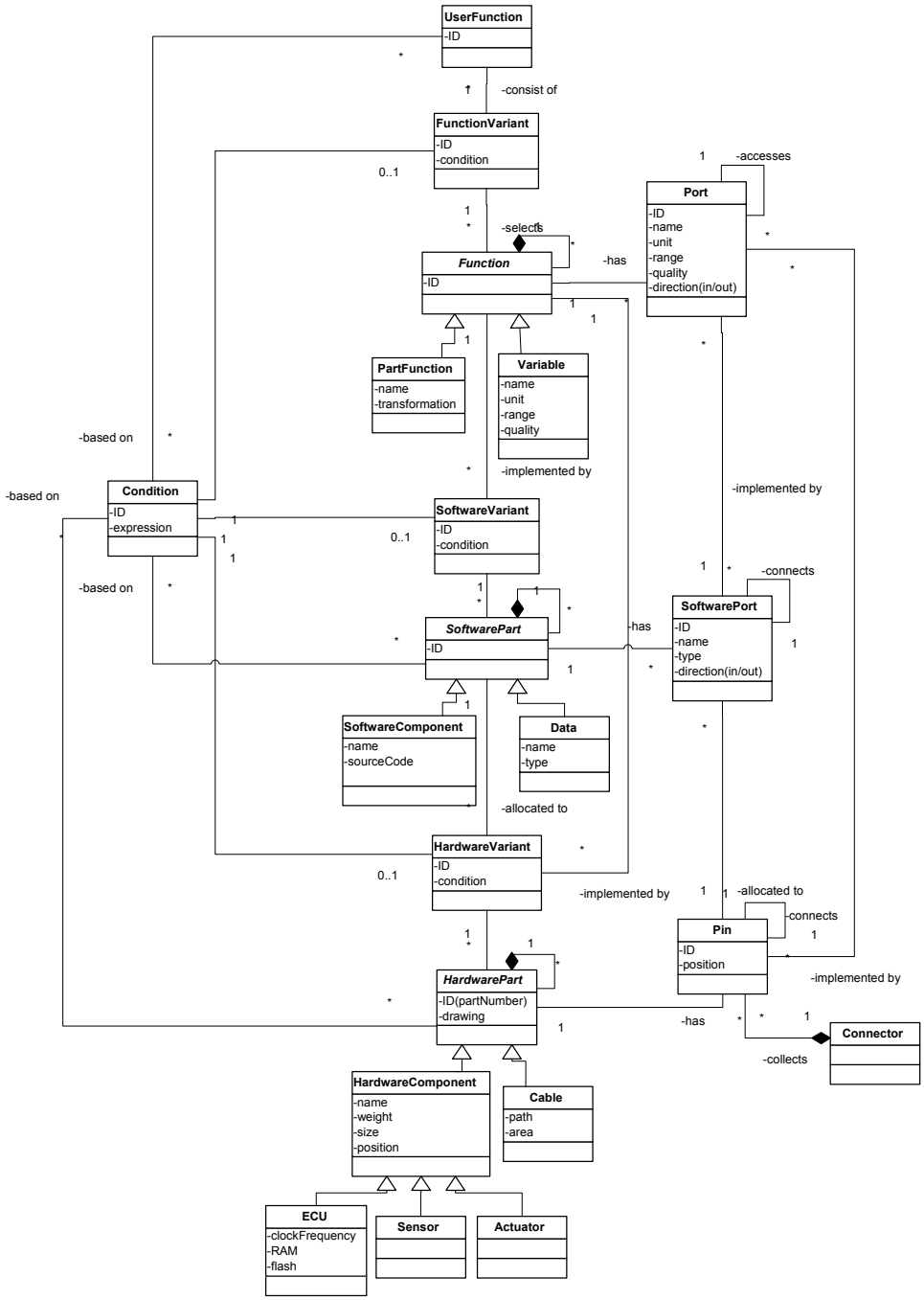

Figure 13. The proposed information model

- The SoftwareVariant is used to represent the different variants in how a particular Function is implemented in software. A SoftwareVariant is a grouping of SoftwarePart objects that together realise a given Function.
- The HardwareVariant is used to represent the different variants in how a particular SoftwarePart is allocated to hardware. A HardwareVariant is a grouping of HardwarePart objects that together implement a given SoftwarePart.

In the above views, objects do not exclusively belong to one view. For example, the SoftwarePart object belongs to both a Software view describing the software implementation, as well as a HardwareVariant view describing the allocation of software to hardware. Such objects help identify the dependencies that exists between views, calling for special attention for their management, in order to reduce duplication and inconsistencies in the product description.

7.2.2. Roles

As illustrated in table 4, certain roles responsible for the development of the views were identified. In most cases, the responsibility of defining the objects within a given view lies with the same role, and the table is hence presented relating views to roles. However, given that objects may not be exclusively defined within one view, it was necessary to relate the role responsibilities at a finer-grained level, relating roles to specific information objects. For brevity, the fine-grained responsibility sharing is not discussed here. In addition, besides the *Owner* roles, there exist several other roles that only need to access the product information, such as the system user, tester, safety analyst and maintenance/repair.

7.2.3. Proposed Documentation

The information model must be captured in some kind of descriptions, textual or graphical, collected in documents. Given the shortcomings of the original documentation, a new documentation solution is proposed replacing the original UFS and MSC documents. Two new documents are suggested instead: A User Function Description (UFD) document and a Function Architecture Description (FAD) document, specifying the implementation-independent functionality and their software/hardware implementation respectively. In the proposal, the SD document is also redefined to focus on the hardware aspects of the system it describes. The content of the new documents is simply a restructuring of the previous documentation, and major changes have been avoided where possible in order to permit a smoother shift to the new documentation structure. Since an analysis of potential tools and models were beyond the scope of the study, and

recognising the effort needed in introducing new tools, documents are still defined using text editors. The use of UML 2.0 activity diagrams for describing functions is however proposed, given the present experience in its usage by some members of the organisation.

Table 4. The roles responsible for the development of the information model views.

View	Owner role	Role Description
Function	Function owner	Responsible for the specification, development and validation of a user function.
Software	System owner	Responsible for the development of a selected set of software/hardware components for the implementation of a selection of partFunctions/softwareParts.
Hardware	System owner	
Function variant	Configuration coordinator (functions)	Manages and ensures compatibility between the combinations of hardware and software for a given configuration. A configuration is a selection of systems with defined hardware and software versions. The configuration coordinator manages the conditions pointing out different variants.
Software variant	Configuration coordinator (software)	
Hardware variant	Configuration coordinator (hardware)	
User function	Function coordinator	Manages the interaction of user functions by coordinating the definition and development of partFunctions and their interactions.
F-SW allocation	Function coordinator	
F-HW allocation	Function coordinator	
SW-HW allocation	Communication coordinator	Manages the allocation of communication between software components both within and between processing units. The communication coordinator is responsible for reliable communication and non-congested channels.

7.3. Conclusion

The keyfigure analysis case study borrowed many ideas from the tool and mechanisms discussed in Paper-B. The multi-view principles presented in Paper-B were adopted in the restructuring and division of the available dataset into different views, thereby facilitating the desired analysis as well as the possibility to perform multiple allocation strategies without needing to remodel the system functionality. In addition, the database structure used in this case study is based on the meta-meta-model suggested in the paper.

Preliminary studies and keyfigure analysis of the case study were first performed using the prototype tool presented in Paper-B. However, a new keyfigure tool implementation was ultimately used to facilitate the process of importing information from the various sources at the organisation. In the final tool, the use of hierarchy within each view, and hence the cross-hierarchy allocation mechanisms, was not adopted. Nevertheless, the prototype tool later took advantage of the case study material for experimentation and testing purposes.

During the import of information from the various data sources, many inconsistencies in the documents were discovered due to duplication of information in the different documents and the lack of mechanisms to propagate changes between them. The needs for an integrated data management system as advocated in this thesis were confirmed from experiences in the case study.

The discovery of inconsistencies also triggered the documentation case study of section 7.2. The scope of the study did not encompass the implementation of tools for the automated management of the suggested documentation. For this reason, it was not possible, nor expected, to directly apply any of the solutions presented in this thesis. However, many ideas were borrowed such as the division of the information model into multiple views, as well as the particular meta-model within each view. Given the lack of automated support, integration was achieved through the restructuring of the documents to minimise the duplication of information and to highlight any relationships between their contents.

8. Future Work

As mentioned in section 4.3, this thesis focused on two cases of integration to cover each of the identified needs of view integration and model management. The potential for future developments is hence great.

The view integration mechanisms presented in Paper-B need to be expanded to cover other types of relationships. While specific to the allocation of system functions to hardware, it is believed that these mechanisms can be applied to other types of relationships such as that of mapping software components to hardware. However, no claim can be made that these mechanisms are general enough to handle all types of relationships. In particular, future work should address the management of duplicated information between tools, synchronizing and maintaining its consistency. A systematic approach when implementing these relationships should allow a reuse of many of the concepts already explored. In addition, the ability to perform inter-view associations over a larger number of views is a challenge to handle in future developments. Finally, a complete MDM-based implementation of the inter-view allocation approach remains to be developed.

A full validation of the PDM/SCM unification approach needs to investigate the feasibility of the remaining management functionalities. The functionalities of the union of typical SCM and PDM tools would include: Version management, product structure management, build management, change management, release management, workflow and process management, document management, concurrent development, configuration management and workspace management [15]. A unified approach should support the common needs of hardware and software development, as well as the discipline-specific needs such as build management for software development.

Relating to implementation issues, the current platform implementation investigates the potential of implementing the MDM platform using the technology offered by a commercial PDM system. This reference implementation can be used to highlight the shortcomings of conventional PDM, as well as the specific needs of MDM. The experience gained can then be used in the development of dedicated MDM systems.

The implementation of the current functionalities has not considered the performance issue yet, focusing instead on the feasibility of the approach in the large. It remains however to see if the expected performance can be provided by a conventional PDM, given that such a system is not normally designed to deal with a large number of fine-grained data items. Such an evaluation will provide valuable feedback on to the expected performance of new MDM solutions.

Finally, some process related and usability issues have been touched upon in this thesis, and are relevant for future work.

The inter-view mechanisms defined in Paper-B support a process-independent allocation practice. By placing certain restrictions, the allocation practices can be constrained. For example, disallowing the possibilities for association extensions through the sub-systems provides a top-down approach, where sub-system design can only refine design decisions specified at the higher level. The open approach however allows for the possibility to feedback information up the hierarchy. Exploring these process issues can be of interest for future extensions.

Doubt remains whether the inter-view mechanisms actually facilitate the developer's work. It is believed that the approach, while based on simple concepts, does require a new mind-set. From the limited gained experiences, the ability to focus on specific parts of the system design, as well as inheriting and extending other decisions made elsewhere in the system, is rewarding. This however does depend on good feedback and support by the integration tool. In the worst case, the approach advocated here can be seen as an experiment, or an initial step, towards other possibilities of view integration.

More advanced fine-grained version control algorithms need to be implemented in the platform. Future algorithms need to support concurrent development, by allowing parallel access to modelling elements, as well as providing branch/merge mechanisms. In addition, in supporting multiple product structures, support for the parallel development of these structures need to be provided, while ensuring the consistency of information across these structures. For usability reasons, the graphical visualisation of the differences between two model versions needs to be developed.

It would also be interesting to develop a number of version control algorithms based on the same MDM platform. The system can then be configured so that different strategies can be applied for different kinds of models. Different development needs can thus be satisfied using variants of the same basic mechanisms in a unified management system. For example, software development might require the complex version control mechanisms and concurrent development normally provided by SCM systems, while hardware development is

satisfied with sequential revision control. The different solutions ought to be based on the same basic mechanisms, user interface and terminology.

9. Conclusion

Weinberg [3] states that ‘A system is a way of looking at the world… The system is a point of view – natural for a poet, yet terrifying for a scientist!’ System structuring is not an inherent property of the system. Instead, it is a way of looking at a system to better understand it.

In the shift from mechanical to multi-disciplinary mechatronics products, the need for multiple viewpoints becomes more evident. The need for multiple disciplines during development means that there will exist multiple viewpoints – multiple product structures. This is specifically amplified with software development within which the presence of many structures is more apparent.

For the successful integration of the efforts from each of these disciplines, the views need to be appropriately integrated, preventing any inconsistencies and divergences from creeping into the system design. Each view structure is equally important and the challenge is to integrate them appropriately.

An acceptable environment to perform view integration, should also deal with the various models used to represent these views. This leads to the need for model management functionalities and hence the challenge of integrating the management systems used by the specific disciplines, namely PDM and SCM systems. It is here argued that model integration ought to be one of the many functionalities supported by such an integrated, model-based, management system.

Recognising that such an environment ought to be a result of standardisation effort, this thesis focused on two cases of integration techniques to investigate each of the view integration and model management issues.

An approach to multi-view modelling and integration which tightly integrates the view hierarchies is presented. Specifically, model integration is investigated for the allocation of system functions onto the implementing hardware architecture. The proposed approach promotes the independent development of the views, allowing developers from each discipline to work concurrently, yet ensuring the completeness, correctness and analysis of any inter-view design decisions made.

A Model Data Management (MDM) platform that generically manages models from the various tools used in development is also presented. View integration is considered as an integral functionality of this platform. The platform is viewed as a unification of the management functionalities typically provided by the discipline-specific PDM and SCM systems. The unification is achieved by unifying the kind of objects it manages – models. The advantage of MDM over conventional PDM/SCM systems is the inclusion of the internal content of its supported models, allowing for a tighter integration of the design information between different models. In demonstrating the platform feasibility, a generic version management functionality of models is implemented.

The platform is argued to be feasible given the move towards model-based development in software engineering, bringing the discipline's needs closer to those of the hardware discipline. This leads the way for an easier and more effective integrated management platform satisfying the needs of both disciplines using a common set of mechanisms. The needs of the disciplines will always differ due to the nature of the products themselves. For example, the development process of software and hardware products differ [15]. However, in a unified management approach, the development needs of both disciplines can be satisfied, using variants of the same basic mechanisms, by providing different strategies for different kinds of models. It is essential however to base the strategies on the same basic mechanisms and user interface, allowing the reuse of basic components and preventing confusion in terminologies. While most critical for multi-disciplinary development, the platform is equally appropriate for the development of purely mechanical or software products.

The major aim of the current platform implementation was to experiment and illustrate the concepts discussed in this thesis. The architecture builds on existing technologies from each of the mechanical and software disciplines. The proposed MDM system is built based on a configurable PDM system, given its maturity, ability to manage model contents and the presence of already developed management functionalities such as the support for distributed development, change management, workflow control, etc. At the same time, the version control functionality borrows ideas from the fine-grained version control algorithms in the software discipline. The adoption of a PDM system is not indispensable and one can envisage building an independent MDM that supports both disciplines. It is our ideal vision that with the acceptance of model-based development, one no longer needs to discuss the integration of PDM and SCM systems. Instead, a truly unified approach to model data management can be used by both disciplines.

10. References

[1] Stuecka R., Bridging the Gap is not Enough – Life-cycle Management for Automotive Electronics and Software, Global Automotive Manufacturing and Technology, 2003.

[2] McKinsey & Company, Knowledge-based changes in the automotive value chain, HAWK-2015, 2003.

[3] Weinberg G. M., An Introduction to General Systems Thinking, Dorset House Publishing; Silver anniversary edition, ISBN 0932633498, 2001.

[4] Larses O., Factors influencing dependable modular architectures for automotive applications. Technical Report TRITA-MMK 2005:09 ISSN 1400-1179. Royal Institute of Technology, KTH, Stockholm, 2005.

[5] Checkland P., Systems thinking, Systems practice: Includes a 30-Year Retrospective. John Wiley & Sons, 1999.

[6] IEEE, ANSI/IEEE Standard 1471-2000, Recommended practice for architectural description of software-intensive systems, September 2000.

[7] Redell O., El-khoury J. and Törngren M., The AIDA toolset for design and implementation analysis of distributed real-time control systems, Microprocessors and Microsystems, Volume 28, Issue 4, 2004.

[8] El-Khoury J. and Törngren M., Towards a Toolset for Architectural Design of Distributed Real-Time Control Systems, 21st Real-Time Systems Symposium, 2001.

[9] Mathworks, Simulink, http://www.mathworks.com/products/simulink/, accessed January 2006.

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

[11] Dome, Dome guide, Version 5.2.2, http://www.htc.honeywell.com/dome/index.htm, 1999, accessed January 2006.

[12] Redell O., Response Time Analysis for Implementation of Distributed Control Systems, Doctoral thesis, Dep. of Machine Design, KTH, TRITA-MMK 2003:17, ISSN 1400-1179, ISRN KTH/MMK--03/17--SE, 2003.

[13] Gomaa H., Software design methods for concurrent and realtime systems, Addison-Wesley publishing company, ISBN 0-201-52577-1, 1993.
[14] Eppinger S. and Salminen V., Patterns of Product Development Interactions, International Conference on Engineering Design, 2001.
[15] Crnkovic I., Asklund U. and Persson Dahlqvist A., Implementing and integrating product data management and software configuration management, Artech House Publishers, 2003.
[16] UML, OMG Unified Modelling Language Specification, V1.5, 2003.
[17] Larses O. and Adamsson N., Drivers for Model Based Development, Proceedings of the 8th International Design Conference on Design, 2004.
[18] Encyclopædia Britannica Premium Service, 'engineering', http://www.britannica.com/ebc/article-9363722, accessed January 2006.
[19] Westfechtel B. and Conradi R., Software Configuration Management and Engineering Data Management: Differences and Similarities, Proceedings 8th International Workshop on System Configuration Management, Springer-Verlag, pages 95-106, 1998.
[20] Bendix L. and Borracci L., Towards a Suite of Software Configuration Metrics, Twelfth International Software Configuration Management Workshop (SCM-12), 2005.
[21] MOF, "Meta Object Facility (MOF) specification", V1.4, April 2002.
[22] World Wide Web Consortium, Extensible Markup Language (XML) http://www.w3.org/XML, accessed January 2006.
[23] Object Management Group, Common Object Request Broker Architecture (CORBA), http://www.omg.org/technology/documents/formal/corba_2.htm, accessed January 2006.
[24] Microsoft, Component Object Model Technologies (COM), http://www.microsoft.com/com/default.mspx, accessed January 2006.
[25] Kemmerer S. J. (editor), STEP, the grand experience, National Institute of Standards and Technology, special publication 939, 1999.
[26] Dahlqvist, A.P., Crnkovic, I. and Asklund, U., Quality Improvements by Integrating Development Processes, 11th Asia-Pacific Software Engineering Conference, 2004.
[27] Cooling J., Software engineering for real-time systems. Pearson Education Limited, ISBN 0201596202, 2003.
[28] El-khoury J., Chen D. and Törngren M., A survey of modelling approaches for embedded computer control systems (Version 2.0), Technical report, ISRN/KTH/MMK/R-03/36-SE, TRITA-MMK 2003:36, ISSN 1400-1179, Department of Machine Design, KTH, 2003.
[29] Garlan D., Monroe R. and Wile D., ACME: An Architecture Description Interchange Language, Proceedings of the Centre for Advanced Studies on Collaborative Research (CASCON) Conference, 1997.

[30] Allen R. J., A Formal Approach to Software Architecture, Ph.D. Thesis, Carnegie Mellon University, Technical Report Number: CMU-CS-97-144, 1997.

[31] Shaw M., DeLine R., Klein D. V., Ross T. L., Young D. M. and Zelesnik G., Abstractions for Software Architecture and Tools to Support Them, IEEE Transactions on Software Engineering, pages 314-335, 1995.

[32] Luckham D. C. and Vera J., An Event-Based Architecture Definition Language, IEEE Transactions on Software Engineering, 1995.

[33] Halbwachs N., Synchronous programming of reactive systems: A tutorial and commented bibliography, Proceedings of the International Conference on Computer-Aided Verification (CAV), 1998.

[34] Harbour M. G., Gutiérrez J. J., Palencia J. C. and Moyano J. M. D., MAST: Modeling and Analysis Suit for Real-Time Applications, Proceedings of the Euromicro Conference on Real-Time Systems, 2001.

[35] Demmeler T., O'Rourke B. and Giusto P., Enabling Rapid Design Exploration through Virtual Integration and Simulation of Fault Tolerant Automotive Application, Society of automotive engineers, Document Number: 2002-01-0563, 2002.

[36] Simon D., Pissard-Gibollet R., Kapellos K. and Espiau B., Synchronous composition of discretized control actions: design, verification and implementation with Orccad, 6th International Conference on Real-Time Control Systems and Application, 1999.

[37] Henzinger T. A., Horowitz B. and Kirsch C.M., Giotto: A time-triggered language for embedded programming, In Proceedings of the First International Workshop on Embedded Software, 2001.

[38] Krueger J. W., Vestal S. and Lewis B., Fitting the pieces together: system/software analysis and code integration using MetaH, Proceedings of the 17th Digital Avionics Systems Conference, 1998.

[39] Liu X., Liu J., Eker J. and Lee E. A., Heterogeneous Modeling and Design of Control Systems, Software-Enabled Control: Information Technology for Dynamical Systems, 2003.

[40] Bræk R., SDL Basics, Computer Networks and ISDN Systems, volume 28, issue 12, special Issue: SDL and MSC, 1996.

[41] G. Schopfer, A. Yang and W.Marquardt, Tool-Integration in Chemical Process Modelling, 9th European software Engineering Conference and 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2003.

[42] Eclipse, The Eclipse Project, http://www.eclipse.org/, accessed January 2006.

[43] Burmester S., Giese H. (et al.), Tool integration at the meta-model level: the Fujaba approach, International Journal on Software Tools for Technology Transfer, volume 6, no. 3, 2004.

[44] Reichmann C., Kuhl M., Graf P. and Muller-Glaser K. D., GeneralStore - A CASE-Tool Integration Platform Enabling Model Level Coupling of Heterogeneous Designs for Embedded Electronic Systems, 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2004.

[45] Karsai G., Lang A. and Neema S., Design patterns for open tool integration, Software and Systems Modelling, Volume 4, Issue 2, 2004.

[46] Becker S. M., Haase T. and Westfechtel B., Model-based a-posteriori integration of engineering tools for incremental development process, Software and Systems Modelling, Volume 4, Issue 2, 2004.

[47] Freude R. and Königs A., Tool integration with consistency relations and their visualization, 9th European software Engineering Conference and 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2003.

[48] Ximbiot, CVS, http://ximbiot.com/cvs/, accessed January 2006.

[49] Larses O., Applying quantitative methods for architecture design of embedded automotive systems, Proceedings of INCOSE International Symposium, 2005.

[50] Larses O. and El-khoury J., Function Modelling to Improve Software Documentation. Technical report, ISRN/KTH/MMK/R-05/25-SE, TRITA-MMK 2005:25, ISSN 1400-1179, Department of Machine Design, KTH, 2005.