# DEVELOPMENT PROCESS FOR MULTI-DISCIPLINARY EMBEDDED CONTROL SYSTEMS

**Sune Wolff**[1,2], **Peter Gorm Larsen**[2], **Tammy Noergaard**[1]

[1]Terma A/S, Integrated Defense Systems
Hovmarken 4, 8520 Lystrup, Denmark,
[2]Aarhus School of Engineering
Dalgas Avenue 2, 8000 Aarhus C, Denmark

*sw@terma.com(Sune Wolff)*

**Abstract**

Developing hybrid systems which combines software, electronics and mechanics is very complex and can lead to delayed projects and erroneous products. By utilizing system-level models, a more complete picture of the system can be produced very early in the development process, and thereby reducing the possibilities of errors due to misunderstandings across domains. In addition, this approach focuses on system-level optimization, instead of super-optimizing the individual sub-systems, which will lead to overall better performance. Many research projects have attempted to tackle the challenges in hybrid systems development, but few have gained acceptance in industry. This paper describes a new project called ”Development Process for Multi-Disciplinary Embedded Control Systems”, which aims at describing a model-driven development process, attempting to bridge the gap between academia and industry needs. The methodology under development will describe how continuous-time and discrete-event models are to be combined in a co-simulation of the entire system, for use in the early design phases of embedded systems. The developed methodology will be tried out on several safety-critical embedded system case studies from the industry, in order to make sure it fits into existing development processes.

**Keywords: Hybrid systems, Model-driven development, Methodology, Co-simulation.**

**Presenting Author’s Biography**

Sune Wolff holds a M.Sc. (2009) in Distributed Real-time Systems from Aarhus School of Engineering, Denmark. After working as a research assistant for at year, he initiated an industrial Ph.D. in collaboration with Terma A/S beginning of 2010. His main field of research is modeling and simulation of embedded systems using formally defined models, with special focus on co-simulation of continuous-time and discrete-event models. This paper gives a description of this industrial Ph.D. project.

# 1 Introduction

The development of hybrid systems consisting of software, electronics and mechanical components which are operating in a physical world is a very complex challenge [1, 2]. The task is made even more challenging due to the fact that these types of systems often are developed out of phase – initially the mechanical parts are designed, followed by electronics and finally software is designed. Any problems discovered late in the development process, can really only be corrected in the software without causing huge delays to the complete project due to longer iterative cycles in electronics and mechanical development. These very late changes often increase the complexity of the software and the risk of introducing new bugs. Hence, an otherwise well thought-out software design can be compromised. In order to avoid situations like this, early feedback at a systems level is invaluable.

Model driven development can address some of these challenges. Normally, the different types of engineers involved in the development of a hybrid system make use of domain specific tools in order to create models of the individual parts of the system. This enables the individual disciplines to optimize one specific part of the system. By creating abstract high-level multi-disciplinary models of the entire hybrid system and the ability to run a co-simulation of these, system engineers will be able to reason about system-level properties of the control system across multiple disciplines. This will ensure early feedback on system properties, as well as ease communication across different disciplines, since the impact of a given design decision can be made visible to the entire team.

This paper describes a new project (Development Process for Multi-Disciplinary Embedded Control Systems) which focuses on model-driven development of safety critical systems which control their physical surroundings. Using existing tools, multi-disciplinary models will be created in order to analyze the methodology used for model-driven development, as well as give to suggestions to how this methodology can fit into existing development processes used in industry.

The challenges in embedded systems development are described from an industrial point of view followed by an overview of academic research projects which aim at overcoming these challenges in Section 2. This section will make it clear that there is a long way from the industrial challenges to current solutions suggested by academia. Following this, current state of the art in model-driven development is described in Section 3, with main focus on the tools used in the different engineering disciplines. In order to fully describe current state-of-the-art model-driven development, research on model-driven methodology is summarized in Section 4. A quick overview of the expected outcome of the project as well as expected research activities are described in Section 5. Preliminary research results are described in Section 6 followed by a brief summary in Section 7.

# 2 Embedded Systems Design Challenges

In order to develop hybrid systems, engineers from different backgrounds and with diverse fields of expertise are involved, making communication much harder than in mono-disciplinary projects. It is close to impossible for each individual engineer to foresee all the cross-discipline consequences of a given design decision.

Initially, the industrial challenges on hybrid system development will be described from an embedded systems specialists point of view. Following this, several academic research projects will be listed in order to show that academia has not adequately solved all of these industrial challenges.

## 2.1 Industrial Point Of View

At the core of the tough challenges real-world development teams building embedded systems face, is balancing quality versus schedule versus features. To be able to ship a very high quality product on-time requires team members to have the wisdom to start demystifying what is being built from day one. This requires that all team members have at the very least a systems-level understanding of the system, and a solid technical foundation of the components that make up the design. One of the common mistakes made is not investigating or understanding the complete embedded systems, the components that can make up the device, as well as the impact of individual components on each-other.

For instance, hardware engineers need to demystify the software layers, and understand the software requirements. This includes everything from device drivers to operating systems to middleware to the application software components. This is because target system hardware requirements depend on the underlying requirements of the software, and must ensure that the required IO (input/output) is available, the target hardware has sufficient memory of the right type, and the processor is powerful enough for example.

On the flip-side, programmers with a purely software background that are accustomed to developing on PCs or larger computer systems, are often intimidated by the embedded hardware and tools. Thus, many teams end up using PCs themselves as both target and host in the place of available target hardware to do development and testing. In this case programmers do not recognize the importance of using some type of reference hardware and/or simulation platform that reflects the target. When using PCs in the place of target hardware, programmers then make software design decisions that would cause failures on the embedded target hardware, because of the mistake of treating embedded hardware like it is a Windows-PC desktop. Developing for an embedded hardware target is not like developing on a PC as the target. Meaning, software executing on a PC of GHz of processing power and Mbytes of memory will not behave the same (or even run at all) when trying to run that same software on an embedded board with MHz of processing power and Kbytes of memory.

When focusing on innovation, it becomes necessary to

ruthlessly address the costly competitive challenges that have arisen due to software taking on major relevance within the successful production of a product line. This software trend is consistent with what has been happening through out technology companies, across all industries. There has been a dramatic growth in projects with increased system software complexity, rising time and cost pressure, as well as higher quality demands. In fact, in many projects where software plays a significant role, the software-related life-cycle costs are often significantly underestimated.

First, there is not enough focus on software from the start. For better or worse, the reality is that core software design decisions and a majority of software-specific product life-cycle costs are defined and driven at the start. The most common triggers that result in negative software-related consequences – from system-level failures to large cost overruns – are due to the consequences of these design decisions. Second, it is very costly to bypass software development best practices in a short-term effort to save time and money. The challenges of balancing the delivery of a high quality product that meets requirements, on-time, all while maintaining a profit places these goals constantly at odds with each other. In a software-intensive, fast-paced project the way to guarantee vast amounts of money will be squandered is when developers do not have the right development process for innovation in place. This methodology is what provides the foundation for the no-nonsense, best-practice discipline necessary for developing best-in-class software, faster.

For example, throwing programmers at a project without the right software development infrastructure will only increase costs and decrease efficiency. This is because, the more people added on ad-hoc to a project, the more interruptions programmers have to manage. Another example is when programmers, in an attempt to save time, cut corners and ignore key best practice approaches in the software development process. When programmers do this, it results in code being developed more slowly – that is more expensive to develop and maintain. This is because not using a proper method and bypassing these best practices when programming results in badly written software will consume more development effort than well written source code to try to complete the project. Fixing software defects consumes a big chunk of software development costs, and of the schedule in debugging and testing. So, by using for example hardcore code inspections, it becomes cheaper than debugging, and result in a reduction of debugging time and logistical maintenance of the code after software release.

Developing and deploying innovative technology and product solutions requires a powerful approach that addresses the reality that the process of successfully engineering a product cannot be applied to all types of engineering. Meaning, to successfully and profitably create innovations to technology requires an inherently different approach, a model that supports an integration of a more dynamic business and engineering framework. This model is the foundation that actively manages the uncertainty in creative innovation of technology, providing the flexibility to efficiently and strategically adapt to constantly changing circumstances and needs of projects that require a more creative, dynamic approach to be profitable. Industry needs this missing stepping stone to insure that the necessary engineering support, data, training and/or guidance is available that maximizes profits for least amount of risk for an innovation.

### 2.2 Academic Research Projects

In the academic world, hybrid systems have been considered for more than a decade starting from a fuzzy logic perspective [3] to an automata perspective [4]. Since then, many academic research projects have attempted to tackle the challenges for developing such multi-disciplinary systems in predictable fashions. These include:

- AVACS investigates automatic verification and analysis of complex systems in particular hybrid systems using model checking.
- COCONUT addresses the design and verification of modern embedded platforms by focusing on the formal specification of software and compilation, formal refinement and formal proof.
- CREDO focuses on the development and application of an integrated suite of tools for compositional modeling, testing, and validation of software for evolving networks of dynamically reconfigurable components.
- DEPLOY is a major European FP7/IP addressing the industry deployment of formal methods for fault tolerant (discrete event) computing systems.
- INTERESTED focuses on creating a reference and open interoperable embedded systems tool-chain. The project's main focus is on discrete event tools for graphical overview and code generation, and not on co-simulation.
- MEDEIA aims to bridge the gap between engineering disciplines in the discrete engineering domain, by using containers that contain design models from various disciplines which can be seamlessly interconnected. Like the INTERESTED, MEDEIA aims to connect tools in the discrete event domain.
- MOGENTES aims to significantly enhance testing and verification of dependable embedded systems by means of automated generation of efficient test cases.
- PREDATOR aims to advance the state of the art in the development of safety-critical embedded systems focusing on timing aspects.
- PRODI aims to develop and test new and improved fault detection and isolation techniques, tailored to the complex area of power systems.

- QUASIMODO aims to develop techniques and tools for model-driven design, analysis, testing and code-generation for embedded systems where ensuring quantitative bounds on resource consumption is a central problem. It focuses on formal notations for timed, probabilistic and hybrid systems that can be subjected to exhaustive state space analysis techniques such as model checking.
- SPEEDS focus on tool-supported collaborative modeling for embedded systems design.
- TOPCASED is developing an Eclipse-based IDE for critical embedded systems with an open source approach.
- VERTIGO aims to develop a systematic methodology to enhance modeling, integration and verification of architectures targeting embedded systems. It will use a co-simulation strategy that allows the correctness of the interaction between HW and SW to be assessed by simulation and assertion checking.

Many of these projects have a very academic approach to system modeling, and hence can be hard to apply in the industry. In order to successfully solve the challenges seen in industry, it is clear that closer cooperation between academia and the industry is needed.

In 2007 the major stakeholders for embedded systems in Europe have jointly started the organization ARTEMIS (Advanced Research & Technology in Embedded Intelligence and System). They have defined a Strategic Research Agenda (SRA) to focus on the main challenges for the future. In this SRA it is explicitly stated that there is a need for multidisciplinary, multi-objective, systems design techniques that will yield appropriate price/performance for acceptable power and with manageable temporal behavior.

One of the main concerns of the industrial Ph.D. project described in this paper is to bridge the gab between industry and academia by ensuring that the project outcome solves actual issues seen in industry. In addition, it is of great importance that the final methodology described fits into existing development methods used in industry, and hence can be applied directly.

# 3 Model-Driven Development

A major challenge in model-driven development of hybrid systems, lies in the combination of the models used by the different disciplines involved in the development of the system. As long as the models lies within the same domain, this challenge is of limited complexity. When combining models from different domains like continuous-time and discrete-event models, the challenge is significant, and as will be argued this has not yet been done in a satisfying manner.

## 3.1 Discrete-Event Modeling

Software engineers can make use of discrete-event models when specifying the logic of the controller of an embedded system. Formal methods can be used to describe these models, where a mathematically based notation enables the engineer to rigorously describe and analyze the properties of the controller. The motivation of using these models is to precisely describe the desired properties of the systems, while applying a higher level of abstraction to less important parts of the system. For example, device drivers and hardware components can be excluded from the model, enabling the software engineer to focus on the algorithmic challenges of the controller.

Several different discrete-event notations are used in industry and academia. Without a doubt, one of the most commonly used technique in industry is the Unified Modeling Language (UML) created by the Object Management Group. UML enables engineers to create several different views of the software under design. For example, requirement specification using use-case diagrams, static architecture using class diagrams and dynamic behavior using sequence diagrams. Industrial strength tools like IBM Rational Rhapsody enables the software engineer to code generate full C/C++ and Java code from a complete suite of UML diagrams, in order to uncover defects earlier in the product life cycle.

In object-oriented formal languages, like the Vienna Development Method (VDM++) [5], the desired functionality of the controller can be described at the desired level of abstraction. This is a major advantage, since the model can describe the current problem at hand, and disregard all other parts of the system. VDM supports abstract data types, pre- and post-conditions as well as invariants which adds to the expressiveness of the language, enabling the user to define expected behavior of the system. A real-time extension of VDM exists called VDM-RT [6], where real-time systems can be modeled as well as distribution and asynchronous communication. VDM is supported by the industrial tool VDM-Tools [7] as well as the open source initiative Overture [8]. Both tools enable the user to make transformations between VDM and UML class diagrams.

## 3.2 Continuous-Time Modeling

In classical physics, aspects like movement, acceleration and force are described using differential equations. Naturally, models of the environment in which an embedded system is operating is best described using differential equations as well. Whereas continuous-time models excel at describing physical movement, hydraulics and pneumatics. These types of models, however, generally lack the possibility to apply the correct level of abstraction to the discrete-event controller.

In industry Matlab/Simulink [9] created by MathWorks is without a doubt one of the most widely used tool for creating continuous-time models. Matlab is a high-level language and interactive environment which perform computationally intensive tasks very fast compared to traditional programming languages. Simulink is an environment for multi-domain simulation and Model-Based Design for dynamic and embedded systems. It provides an interactive graphical environment

and a customizable set of block libraries which lets the user design, simulate, implement, and test a variety of time-varying systems, including communications, controls, signal processing, video processing, and image processing.

20-sim [10] is a modeling and simulation tool for dynamic systems, such as electrical, mechanical and hydraulic systems or any combination of these. This tool fully supports graphical modeling, allowing design and analysis of dynamic systems in an intuitive and user friendly way. 20-sim supports the use of components, enabling the engineer to choose predefined components from a library and connecting these. In addition, the underlying technology, Bond Graphs, can be used directly in order to model dynamic behavior. It is also possible to export 20-sim models to Matlab/Simulink.

### 3.3 Combining Discrete-Event and Continuous-Time Models

Whereas the individual technologies and tools described above excel at creating models in their specific domain, few (if any) are suitable for modeling systems operating on both discrete-events and in continuous-time. The Stateflow application from MathWorks extends Simulink by allowing the use of language elements to describe complex logic. It is tightly integrated with Matlab and Simulink, and provides an environment for designing embedded hybrid systems. Matlab is a very efficient tool for algorithm development and large calculations, but lacks the possibility of using object-oriented architecture. The tool chain is also limited in the inability to apply a higher level of abstraction to parts of the system not currently in focus.

Another approach to combine discrete events and continuous time is to use discrete abstractions of the continuous elements. This approach was originally proposed by Alur et al. [11] who uses the first derivative of a continuous signal as well as state changes as discrete events which can be simulated with the remaining discrete parts of the system. An example of this can be seen in Fig. 1. Every time the signal crosses a threshold, and at every local maximum and minimum a discrete event is generated. This has the unfortunate side effect that all notions of time is lost in the continuous time model. If this is acceptable for the problem at hand, this method can be used to great success. But especially if real-time aspects of the system is of interest, valuable information is lost.

The project ”Ptolemy2” [12] from University of California, Berkeley allows modeling, simulation and design of concurrent real-time embedded systems within multiple domains such as discrete-time, continuous-time, dynamic data flow, state machines and more [13, 14]. A special build-in tool HyVisual is especially suited for modeling hybrid systems, where continuous-time components are modeled using block diagrams and discrete-event controllers are modeled using state machines. Unfortunately Ptolemy2 has not gained ground in the industry, perhaps because engineers would like to continue using the same tools they

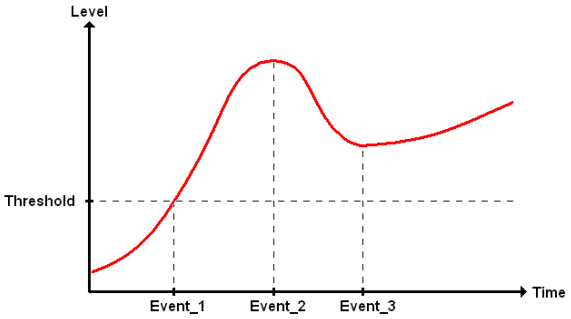


Fig. 1 Discrete abstraction of continuous-time

are accustomed to. In addition, it is not possible to formally verify Ptolemy2 models due to lack of formal definition of the semantics.

The project ”Design Support and Tooling for Embedded Control Software” (DESTECS) [15] aims at creating a methodology and open tools platform for the collaborative and multidisciplinary development of dependable embedded real-time control systems. The methodology combines continuous-time and discrete-event modeling via co-simulation, allowing explicit modeling of faults and fault-tolerance mechanisms from the outset. Continuous-time models are expressed using bond graphs supported by the 20-sim tool while discrete-event controllers are modeled using VDM supported by the Overture tools. This project holds great promise both regarding tools and methodology, and will be followed closely throughout this project.

## 4 Model-Driven Methodology

Just as important as the individual technologies and tools, is a thoroughly described methodology of how to exploit the possibilities provided. For the monodisciplinary tools several methods have been described. Prior work [16] suggests how a VDM-RT model should be developed through a sequence of steps, each adding more details to the model. One aspect of this method which is still lacking is a description of how to determine the requirements of a system. This challenge is the main focus of Hayes, Jackson and Jones [17, 18], who argue that the specifications of a system should be derived formally from a description of required phenomena in the physical world in which the system will operate. Combining these two methods, will result in a complete description of how requirements of the system is determined, how these requirements are incorporated in a VDM model, which is refined through a series of steps to finally end up with a VDM-RT model defining timing aspects and distributed nature of the system. This method is focused on development of discrete controllers, and as such does not try to counter the challenges of combining models from different domains.

Agile methods are used increasingly in industry, since many companies have realized the importance of a close cooperation with the customer or end user. Even for safety critical systems, agile methods can be used

– usually a hybrid approach is used though, where all requirements are defined in a traditional fashion, followed by a more agile approach to the day-to-day development of the system. This ensures that the customer gets several running prototypes based on which feedback can be provided, ensuring that errors or misunderstandings are caught early on. The combination of formal and agile methods have been researched in [19], which indicate that there is no reason not to combine these two approaches in order to get the best of both worlds.

Only little research in methods for multi-disciplinary model-driven development has been carried out. The project "Beyond the Ordinary: Design of Embedded Real-Time Control" (BODERC) from University of Eindhoven in The Netherlands suggests a model-driven design process. They demonstrate how to reason about system properties by making abstract high-level and multi-disciplinary models. The BODERC design methodology consists of a system-level reasoning framework and plug-ins that are used during system design to get more in-depth insight. For this system-level reasoning BODERC suggests using the *CAFCR* method [20]. This method uses a system architecture description which is divided into 5 views, which together describe what the customer wishes, what the desired functionality of the system is and how the system should be implemented.

# 5 Project Objectives

Inspired by the previous research introduced in Section 4, the project "Development Process for Multi-Disciplinary Embedded Control Systems" aims at creating a methodology describing how the model-driven development ideology fits into existing and well proven classic development methods. The methodology will describe when co-simulation is the right approach and how the complete system should be divided into the continuous-time and discrete-event domains. It is the goal to create a *golden reference* to be used when utilizing co-simulation of system-level models during the development of embedded systems. It is the hypothesis that shorter time-to-market can be achieved by using co-simulation at the system-level. The time invested in analyzing models and co-simulation is gained later in development because of fewer errors. A more optimal design at the system level is achieved, and thereby improving the price/performance of the final product.

As part of achieving the overall goal of the project, several objectives must be met:

1. Evaluate existing tools and methods for creating continuous-time models, discrete-event models and multi-disciplinary models. Compile a list of pros and cons for each tool and method, as well as estimate the prevalence in industry.

2. Expand prior work to include description of how system requirements should be derived as described in Section 4.

3. Describe a framework for model-driven development process. Special focus must be put on development methods currently used in industry, since it is imperative that the model-driven development fits into existing methods.

4. Demonstrate viability of model-driven development of safety critical hybrid systems in an industrial setting. Software, mechanics and systems engineers must all be involved in this test, in order to get thorough feedback on the advantages and disadvantages of the model-driven development methodology described.

# 6 Research Activities

As described above, the project is focused on the methodology of model-driven development. Even though the project is still in its initial phases, good progress has been made with points one and three listed in Section 5. The following subsections describe these research activities in more detail.

## 6.1 Comparison of Continuous-Time Modeling Tools

Different tools excel at modeling different types of systems, or excel at different aspects of the modeling process – but no single tool does it all. Hence, it is important to have an overview of the different tools available to know the pros and cons of the individual tools in order to make a qualified choice for the problem at hand.

There exists a lot of tools for modeling physical systems – too many to mention here, and of course too many to include all of them in a comparative test. For the survey carried out, four different tools have been chosen. Matlab/Simulink was chosen since it is widely used in the industry – MathWorks claim that more than 1 million engineers worldwide use the tool. As a free alternative, the open-source tool Scilab/Xcos [21] was chosen. This tool is known by many as the "Open-source version of Matlab". The tool 20-sim has its roots in academia but is now maintained and further developed by the company ControlLabs which is an off-spring company from the Control Engineering Group at Twente University. It was chosen as an example of a tool which has successfully moved into industry. Finally, Ptolemy2/HyVisual was chosen as a purely academic tool developed and maintained by the Center for Hybrid and Embedded Software Systems (CHESS) at University of California at Berkeley.

The following is a non-exhaustive list of comparison criteria which are used for the continuous-time modeling tools survey:

**Accessibility and usability:** How easy are the tools to use with little or no training? It should be noted that the authors had only little experience with the four tools prior to the comparison.

**Extend model:** How easy is it to extend the model with additional details to gain higher fidelity? It is normal procedure to start out with a very abstract model, and

later on add details to describe the system more precisely – the tools should support this approach as best as possible.

**Reuse model:** How easy is it to reuse parts of or the entire model for other systems? Reuse is important in order to cut down on design time and to avoid errors. The tools should support the creation of sub-models or user defined libraries, in order to ease the reuse of parts of the model.

**Visualization:** How well do the tools support visualization of data and output of the model? It is common for modeling tools to visualize output in 2D graphs, but exporting to data-files or built-in 3D animation tools would also be an advantage.

A central part of the tools comparison, is the creation of a model of an embedded system use case using each of the different tools. The system chosen is a water tank, with a constant drain and a controllable water source. An overview of the case study is presented in Fig. 2.

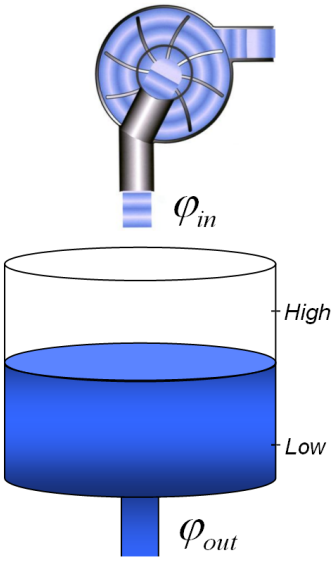


Fig. 2 Water tank case study

The example is very simple, but still contains a lot of characteristics from an embedded control system. Most importantly, a model of the case should include both a continuous-time model describing the change of drain flow due to pressure from the changing water level, and a discrete model describing the controller. This will test the ability of the continuous-time modeling tools to describe discrete-event controllers. The water tank is filled by a controllable water source $\varphi_{in}$ and through a drain the output flow $\varphi_{out}$ is continuously running. The controller must make sure that the water level is held between an upper and a lower threshold.

From a common mathematical model, the water tank case study was modeled in each of the four tools. Fig. 3 shows the output from the 20-sim model, which clearly shows how the change of water pressure affects the drain of water.

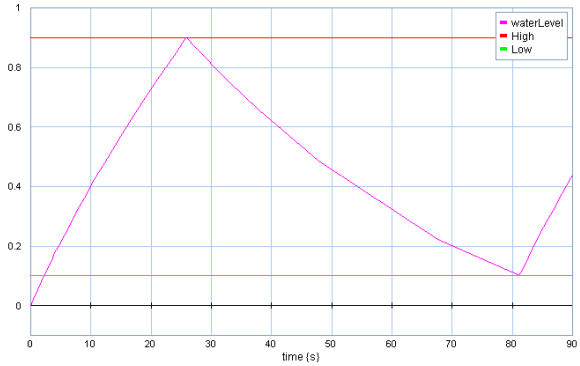


Fig. 3 Water tank model 2D graph output

It is still too early to give a clear conclusion of the tool comparison, but the following gives a quick overview of some of the findings:

**Accessibility and usability:** All four tools support the use of block diagrams, so once the differential equations of the mathematical model have been derived, it is quite easy to create a model of the system. 20-sim also supports the use of iconic diagrams, which are building blocks of real components e.g. mass, spring and forces. This makes it easier for new users to start using the tool, without spending time deriving the underlying differential equations first.

**Reuse model:** All four tools support sub-models, which help to modularize the system model, and thereby making it easier to reuse some of the components. In addition, 20-sim enables the user to create their own components and to insert these into a commonly used library for use in other models. In Matlab, m-functions can easily be reused as well.

**Visualization:** All tools support data visualization through 2D graphs. In addition, both Ptolemy2/HyVisual and 20-sim have built-in 3D animation tools, where data can be tied to simple objects like cubes and cylinders, to dynamically change the position or the size of these objects. The Simulink 3D Animation toolbox provides similar functionality, but a user would have to pay extra for this feature.

**General observations:** The commercial tools Matlab/Simulink and 20-sim seem a bit more stable than the two other tools, which crash from time to time during usage.

### 6.2 Co-simulation Methodology

The initial work on a methodology for using co-simulation as a tool to develop hybrid systems, is partly inspired by the work of Hayes, Jackson and Jones briefly mentioned in Section 4. They make a clear distinction of the assumptions of the physical parts of the system or the surroundings in which it is operating, and the actual requirements of the controller. In short, the method consists of two steps:

1. Describe the required phenomena in the physical

world; and

2. Derive a specification of the control system.

By making a clear separation of environment assumptions and controller requirements, dividing the system into the continuous-time discrete-event domains is made easier. The interface between the two domains is also being specified in the process, which will be exemplified using the water tank case study:

Tab. 1 List of requirements

| Id | Description |
|---|---|
| R1 | If water level reaches “High level”, the water source must be turned off |
| R2 | If water level reaches “Low level”, the water source must be turned on |

The water tank only has two simple requirements – to keep the water level between the “High level” and “Low level” thresholds. It would be possible to come up with additional requirements, such as safety requirements stating what should happen if the water level should ever exceed the “High level” mark, but this is not considered in this simple example.

Tab. 2 List of assumptions

| Id | Description |
|---|---|
| A1 | A ”High level” sensor is installed |
| A2 | A ”Low level” sensor is installed |
| A3 | Controller can read ”High level” sensor value |
| A4 | Controller can read ”Low level” sensor value |
| A5 | Controller can turn the water source on |
| A6 | Controller can turn the water source off |
| A7 | If water source is on, water level is raised |
| A8 | If water source is off, water level is lowered |

It is important to be very thorough when specifying all the assumptions about installed equipment and behavior of the physical world in which the system is operating. The more detailed the assumptions, the better the interface between the continuous-time domain and discrete-event controller will be specified.

When all requirements and assumptions have been specified, these must be placed in a chart, where nouns are either in the continuous-time domain, discrete-event domains or in the interface between the two. Additional requirements and assumptions describing the relation between these nouns are drawn as arrows on the chart – a result of this analysis can be seen in Fig. 4.

The method described above only covers the dissection of the system into continuous-time and discrete-event domains, so as a methodology it is highly incomplete. In spite of this, two pieces of very valuable information can be gained from the analysis:

- The system is divided into the two domains, which clearly describes the purpose of the different models, as well as clearly defining the interface between the models. The resulting system breakdown chart can be seen as a contract between the two domains involved. By having a clearly defined purpose of the model and interface to other models, it will be much less challenging to make the models and ensuring integration problems are avoided.

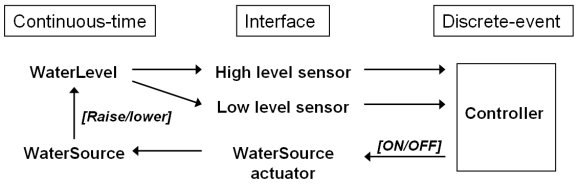


Fig. 4 System breakdown chart

- The results of the analysis can help in the decision process of whether to model the entire system in one domain, or if co-simulation between multiple domains should be utilized. If, for example, most of the system ends up in the discrete-event domain, and only very simple parts of the system end up in the continuous-time domain, there is no motivation to do a multi-domain co-simulation of the system. If, on the other hand, the different components of the system are distributed evenly across the two domains, co-simulation will be enabled.

## 7 Summary

Designing hybrid systems consisting of software, electronics and mechanical components is a highly complex affair, and existing mono-disciplinary development methods cannot be applied directly to these types of systems. Even though a lot of academic projects has been carried out in this area of research, few - if any - has gained acceptance in industry.

The aim of the project ”Development Process for Multi-Disciplinary Embedded Control Systems” is to describe a model-driven development methodology, which can be used as a golden reference when creating system-level models in multi-disciplinary projects. It is of great importance that the method fits into existing classic development processes, as well as support dialog across disciplines. The methodology will be tested on case studies of safety-critical hybrid systems. It is imperative that the developed method can be widely accepted by engineers with different fields of expertise such as electronics, mechanics and software.

A comparative survey of four continuous-time modeling tools has been initiated, in order to get an overview of their capabilities. This comparison will help determine which tools are better suited for which modeling task, and will in turn help determine which parts of the complete system should be modeled in which tool. It will also help in the process of determining whether a single tool can be used to model the entire system, or if a co-simulation of several domain specific models

should be used. Future work includes expanding this survey to include discrete-event tools as well.

Finally, work on the model-driven methodology has started, initially focusing on dissecting the complete system in order to designate which parts should be modeled in which domain. The early results of this method holds great promise, and will act as a solid base upon which to further develop a complete method for developing hybrid systems.

## Acknowledgment

The authors would like to thank "The Danish Agency for Science, Technology and Innovation" for providing parts of the funding for the project. In addition, Terma A/S deserves thanks for believing in the project in these trying times. Finally, the authors would like to thank the anonymous reviewers for their valuable feedback.

## 8 References

[1] Thomas A. Henzinger and Joseph Sifakis. The embedded systems design challenge. In *FM 2006: Formal Methods, 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006, Proceedings*, pages 1–15, 2006.

[2] Tom Henzinger and Joseph Sifakis. The discipline of embedded systems design. *IEEE Computer Society*, 40(10):32–40, October 2007.

[3] Suran Goonatilake and Sukhdev Khebbal, editors. *Intelligent Hybrid Systems*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[4] Thomas Henzinger. The theory of hybrid automata. In R.P. Kurshan M.K. Inan, editor, *Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS)*, pages 278–292. IEEE Computer Society Press, 1996.

[5] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, and Marcel Verhoef. *Validated Designs for Object–oriented Systems*. Springer, New York, 2005.

[6] Paul Mukherjee, Fabien Bousquet, Jérôme Delabre, Stephen Paynter, and Peter Gorm Larsen. Exploring Timing Properties Using VDM++ on an Industrial Application. In J.C. Bicarregui and J.S. Fitzgerald, editors, *Proceedings of the Second VDM Workshop*, September 2000. Available at www.vdmportal.org.

[7] John Fitzgerald, Peter Gorm Larsen, and Shin Sahara. VDMTools: Advances in Support for Formal Modeling in VDM. *Sigplan Notices*, 43(2):3–11, February 2008.

[8] Peter Gorm Larsen, Nick Battle, Miguel Ferreira, John Fitzgerald, Kenneth Lausdahl, and Marcel Verhoef. The Overture Initiative – Integrating Tools for VDM. *ACM Software Engineering Notes*, 35(1), January 2010.

[9] MathWorks. http://www.mathworks.com, 2010.

[10] Jan F. Broenink. Modelling, Simulation and Analysis with 20-Sim. *Journal A Special Issue CACSD*, 38(3):22–25, 1997.

[11] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22:181–201, 1996.

[12] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A Framework for Simulating and Prototyping Heterogeneous System. In *Int. Journal of Computer Simulation*, 1994.

[13] Christopher Brooks, Chihhong Cheng, Thomas Huining Feng, Edward A. Lee, and Reinhard von Hanxleden. Model engineering using multimodeling. In *1st International Workshop on Model Co-Evolution and Consistency Management (MCCM '08)*, September 2008.

[14] J. Eker, J.W. Janneck, E.A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong. Taming heterogeneity – the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, January 2003.

[15] J. F. Broenink, P. G. Larsen, M. Verhoef, C. Kleijn, D. Jovanovic, K. Pierce, and Wouters F. Design support and tooling for dependable embedded control software. In *Proceedings of Serene 2010 International Workshop on Software Engineering for Resilient Systems*. ACM, April 2010.

[16] Peter Gorm Larsen, John Fitzgerald, and Sune Wolff. Methods for the Developing Distributed Real-Time Systems using VDM. *International Journal of Software and Informatics*, 3(2-3), October 2009.

[17] Ian J. Hayes, Michael A. Jackson, and Cliff B: Jones. Determining the specification of a control system from that of its environment. In Keijiro Araki, Stefani Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, volume 2805 of *Lecture Notes in Computer Science*, pages 154–169. Springer-Verlag, September 2003.

[18] C. B. Jones, I. J. Hayes, and M. Jackson. Deriving Specifications for Systems That Are Connected to the Physical World. In *Formal Methods and Hybrid Real-Time Systems*, volume 4700 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

[19] S. Black, P.P. Boca, J.P. Bowen, J. Gorman, and M. Hinchey. Formal versus agile: Survival of the fittest. *IEEE Computer*, 42(9):37–45, September 2009.

[20] Gerrit Muller. CAFCR: A Multi-view Method for Embedded Systems Architecture; Balancing Genericity and Specificity, 2004.

[21] Bin Wu and Andre Bogaerts. Scilab - a simulation environment for the scalable coherent interface. In *MASCOTS 95: Proceedings of the Third International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 242–247, January 1995.