Design Support and Tooling for Dependable Embedded Control Software

J. F. Broenink
University of Twente
The Netherlands
J.F.Broenink@ewi.utwente.nl

P. G. Larsen
Engineering College of Aarhus
Denmark
pgl@iha.dk

M. Verhoef
CHESS BV
The Netherlands
Marcel.Verhoef@chess.nl

C. Kleijn
Controllab Products BV
The Netherlands
Christian.Kleijn@controllab.nl

D. Jovanovic
Neopost BV
The Netherlands
D.Jovanovic@neopost.com

K. Pierce
Newcastle University
UK
K.G.Pierce@ncl.ac.uk

ABSTRACT

The efficient design of resilient embedded systems is hampered by the separation of engineering disciplines in current development approaches. We describe a new project entitled "Design Support and Tooling for Embedded Control Software" (DESTECS), which aims to develop a methodology and open tools platform for collaborative and multi-disciplinary development of dependable embedded real-time control systems. We also present some initial results from a small co-simulation case study.

The DESTECS methodology combines continuous-time and discrete-event modelling via co-simulation, allowing explicit modelling of faults and fault-tolerance mechanisms from the outset. Continuous-time models are expressed using differential equations, which we represent using the well-known bond graph notation, supported by the 20-sim tool. We model discrete-event controllers using the Vienna Development Method (VDM), supported by the Overture tools. An open, extensible tools platform will be developed, populated with plug-ins to support static analysis, co-simulation, testing and fault analysis. Trials will be conducted on industrial case studies from several domains, including document handling, inertial measurement and personal transportation.

Categories and Subject Descriptors

B.2.3 [**Reliability, Testing, and Fault-Tolerance**]: Error checking; B.8 [**Performance and Reliability**]: General; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

Keywords

Formal Methods, Resilience, Fault Tolerance, Embedded Systems, Co-simulation

SERENE 2010, London, United Kingdom

1. INTRODUCTION

The embedded systems market is a rapidly evolving one, making it imperative that developers can conceive and evaluate designs quickly and with confidence. This is made all the more challenging by two factors. First, ever more demanding and interdependent requirements, including the need for reliability, fault tolerance, performance and interoperability. Second, the increasingly distributed character of embedded systems, which introduces a wider range of architectures – and faults – for controllers. This paper describes a new project (DESTECS)[1] that addresses collaborative, multi-disciplinary design of embedded systems using methodology and tools that promote rapid construction and evaluation of system models.

One of the main impediments to the design of embedded real-time control solutions is the separation of control engineering, which typically uses tools operating on continuous-time models, and software engineering, which is founded on discrete-event models. In order to evaluate alternative designs and support early defect analysis / correction, it is essential that engineers collaborate across disciplines in short windows of opportunity. Model-based approaches provide a way of encouraging collaboration, but engineers need to perform design evaluation and analysis using models expressed in different tools. These tools should reflect the relevant aspects of the design in a natural way, but also allow consistent, rapid analysis and comparison of models. Achieving this requires advances in continuous-time modelling; formal discrete-event modelling of controllers and architectures; fault modelling and fault tolerance; and open tools frameworks. These various advances are the aim of the DESTECS project.

The rest of this paper is structured as follows. Section 2 briefly discusses our initial selection of continuous-time tool and discrete-event method. The concept of co-simulation is also introduced. Section 3 describes an early result in co-simulating a simple water tank case study. Section 4 discusses the goals of the DESTECS project and sets out some of the challenges we face. Section 5 describes how we hope to support the design of dependable embedded systems by allowing designers to explore and create fault tolerant designs. Section 6 introduces the main industrial case studies. Finally, we provide a few concluding remarks in Section 7.

[1] `http://www.destecs.org`.

2. MODEL-BASED DESIGN

Many tools and techniques have been developed to support model-based design as a way to support the collaboration of engineering teams. For the development of embedded control systems, the greatest challenge lies in bridging the gaps between the different computational models underpinning the disciplines involved [7], particularly between continuous-time (CT) and discrete-event (DE) modelling.

In a development led by control engineering, the plant to be controlled is described in a CT model using differential equations and with, initially, an assumption of fault-free behaviour. The control software will typically be described at a low level of abstraction, so that the major part of the controller model is devoted to a complex description of special cases, including faults. The lack of abstraction raises the complexity of analysis of system behaviour and increases the maintenance effort on the models. By contrast, a software engineering approach typically starts from an abstract model of the logic of the controller software. This facilitates model-driven development and the description of fault handling at a higher level of abstraction. However, the discrete-event formalisms used to describe such systems are inappropriate for describing the whole system's dynamic behaviour. Control laws are typically expressed as algorithms, making it complex to analyse the desired control properties.

There are several attempts to integrate CT and DE models. Matlab/Simulink, in combination with the Stateflow toolbox, provides a tool chain that is well-suited for fine grained controller design. The abilities of the Stateflow toolbox to support existing methods and practices of software engineering are, however, limited by the low level of abstraction in the notations. Ptolemy-II, a more radical component-based approach, supports several domains, each of which is based on a particular model of computation and may be combined with others to build a system model [1, 3]. Industrial adoption has been limited, possibly because of the extent to which it represents a departure from current design practice, leading to it being regarded as high risk for adoption. Possibly the most promising method to combine the computational models is to use the most successful tools chains of the various engineering domains and integrate them at simulation level. This is known as co-simulation.

In a co-simulation, both the CT and DE models appear to execute at the same time. In fact, each model is executed alterately with the other, simulating for a period of time (that is, performing a step). This time step is initially suggested by the DE controller, which calculates the smallest time it can simulate before it will perform an action, e.g. Δt. This is time-triggered control. The CT model can then simulate for this period, but may discover that before the end of the step, a change occurs which the controller should know about, e.g. at $\Delta t/2$. The DE model must then simulate for this shorter period instead. This is event-triggered control.

2.1 Continuous-Time Modelling

Without doubt, Matlab/Simulink [14] has the largest user base in industry as well as in the academic world. The modelling and simulation part, Simulink, is built upon the Matlab environment and provides block diagram modelling. The base library of Simulink is limited to block diagrams. External libraries with physical components can be purchased. These libraries are comparable to what is offered in Modelica and 20-sim [12], but not with the same level of sophistication. Moreover, the library models are closed source.

Scilab is an open-source scientific software package containing two toolboxes for modelling and simulation: Scicos and the OpenModelica Toolbox [2]. Scicos is the counterpart of Simulink but is limited to block diagrams only. The Open Modelica Toolbox is an attempt to create models at the physical component level. The toolbox however has not reached a sufficient level of maturity yet for use in an industrial setting.

Modelica is an open-source, object-oriented multi-domain language for modelling physical systems [19]. Next to the language, Modelica has a number of open and closed source libraries of physical components. There are several tools available for simulating Modelica based models. Some of them are open source but have limited capabilities. Probably the best known tool is the Dymola package.

20-sim is a multi-domain modelling and simulation package for modelling complex physical systems. All model libraries of 20-sim are open source, and have the same level of sophistication as Modelica. The package supports mixed mode integration techniques to allow the modelling and simulation of computer controlled physical systems that contain continuous as well as discrete time parts. The package supports the connection of external software through dll-functions, both at modelling and simulation level (discrete-time, continuous-time or hybrid). 20-sim allows export to Matlab/Simulink at all levels.

Other well-known packages are Easy5, Vissim, AMESim, Labview and ACSL. They are not described in detail here, because they all lack one or more important capabilities (integrated simulator with discrete-event support, co-simulation interface, on-board libraries for plant design etc.).

2.2 Discrete-Event Modelling

DE modelling notations used in industry are mostly based on finite state machines and have a low level of abstraction. IBM Rational Technical Developer (formerly Rational Rose Real-time) and IBM/Telelogic Rhapsody provide modelling capabilities based on the Unified Modelling Language (UML) and the System Modelling Language (SysML) and are supported by mature development processes (RUP and Harmony/ESW respectively). Both tools aim to develop executable models that are deployed on the target system as soon as possible to close the design loop, requiring that the model evolves to a low level of abstraction early in the design process: for instance, the resolution and accuracy of the timing objects are already determined at the modelling-language level by the target platform's operating system services.

In model-oriented formal methods such as VDM [5] we can describe the desired functionality at a higher level of abstraction. VDM is supported at industrial-strength level by VDMTools [4, 6] and it is already coupled to UML. Recently, VDM and VDMTools have been extended to better support the description and analysis of real-time embedded and distributed systems. These include primitives for modelling deployment to a distributed hardware architecture and support for asynchronous communication. The VDM technology has been extended with a capability to generate traces derived from simulations [17, 16]. An initial proof of concept of integration between VDM and 20-sim (for continuous-time simulation) has already been carried out [18, 15]. Overture

[13] includes the same support directly on top of the Eclipse platform. This tool will form the basis for the DE simulation since a simulator for the executable subset of VDM is already a part of the Overture tool suite.

Is it possible to support both control engineering and software engineering using a single unified method or tool? Several attempts have been made to unify both worlds. For example, Hooman et al. have co-simulated Rose Real-time software models with control laws specified in Matlab/Simulink by providing a platform neutral notion of time instead [9]. This is a step forward, but also suggests that Rose Real-time lacks a suitable notion of simulation time and does not allow interrupts due to events in the plant. IBM/Telelogic Rhapsody is able to integrate with Simulink models running in discrete time.

3. CO-SIMULATION EXAMPLE

As an initial case study, we use a small water tank example. Here, the water level in the tankforms a continuous-time system, described by differential equations. This is the plant. In this simple example, the tank is subject to an arbitrary input flow, however it could be extended with other "disturbances", such as evaporation of the water. The term "disturbance" is used for phenomenon that are not caused by the controller but nevertheless influences the system. The controller can observe properties of this plant (the water level) and can change the state of the plant by performing a control action (opening a valve to allow water to flow out), according to some control law. This control law keeps the system as a whole in some desired state.

In our case study, the aim of the controller is to keep the water level between the low and high watermark. The controller can observe the water level through three sensors: a pressure sensor at the bottom of the tank, which measures the current water level continuously; and two discrete sensors, which measure the water level within the tank. The upper sensor informs the controller when the water level exceeds the high water mark and the lower sensor fires if the water level drops below the low water mark. The controller can influence the water level by opening or closing a valve at the bottom of the tank.

The reason why co-simulation is needed for an example such as this is that the speed by which the water pours out of the water tank depends upon the volume of water in the tank so the response time of DE depends on the state of CT. The case study concerns a water tank that is filled by a constant input flow f_{I} and can be emptied by opening a valve resulting in an output flow f_{O}. The volume change is described by equations (1) and (2), where A is the surface area of the tank bottom, V is the volume, g is the gravitation constant, ρ is the density of liquid and R is the resistance of the valve exit.

$$\frac{d\mathrm{V}}{dt} = f_{\mathrm{I}} - f_{\mathrm{O}} \tag{1}$$

$$f_{\mathrm{O}} = \begin{cases} \frac{\rho \cdot g}{\mathrm{A} \cdot \mathrm{R}} \cdot \mathrm{V} & \text{if valve = open} \\ 0 & \text{if valve = closed} \end{cases} \tag{2}$$

Modelling physic laws such as these can conveniently be expressed in a continuous-time tool such as 20-sim. In order to cope with both event-based as well as time-triggered we need to identify events of interests. The event REE is the so-called rising edge zero crossing and FEE is the falling edge zero crossing. For our case study, we define two edge triggered events: REE (*level*, 3.0) and FEE (*level*, 2.0), whereby *level* is a shared continuous time variable that represents the height of the water level in the tank. In case such events happend before the time limit for the co-simulation is due a smaller time step will be taken such that the controller will be able to react at the right point of time.

On the controller side we model the intended behaviour using VDM. The shared continuous sensor and actuator variables *level* and *valve* are declared on Line 4 and 5 below. Whenever *level* is read, it contains the actual value of the corresponding continuous time.

```
01 class Controller
02
03 instance variables
04 static public level : real;
05 static public valve : bool := false
06
07 operations
08 static public async open: () ==> ()
09 open () == valve := true;
10
11 static public async close: () ==> ()
12 close () == valve := false;
13
14 loop: () ==> ()
15 loop () ==
16   if level >= 3 then open()
17   elseif level <= 2 then close();
18
19 threads
20  periodic(1000,0,0,0)(loop)
21
22 sync
23  mutex(open, close, loop)
24
25 end Controller
```

For illustration purposes the *loop* operation which does the control is periodically invoked every second and it will open or close the value whenever necessary. However, it may be actived more freqently in case the events mentioned above are activated.

This case study was presented as a co-simulation between 20-sim and VDM in [15]. The initial result from DESTECS is to port this example to the Overture platform. This serves as both a technical exercise in the mechanics of co-simulation between 20-sim and Overture and as a starting point for exploring the methodological issues of collaborative design and co-simulation. This co-simulation example also includes a 3D animation of the water tank and its associated level, which reflects the state of the model over time, as illustrated in Figure 1.

Further case studies for the DESTECS project are described in Section 6.

4. DESTECS GOALS AND CHALLENGES

The goal of DESTECS is to improve the productivity of innovative embedded system design by providing and evaluating new methods and tool support that can be used to design fault-tolerant, embedded systems using a multidisciplinary, collaborative model-based approach (see Figure 2). Achieving this goal entails the following objectives:

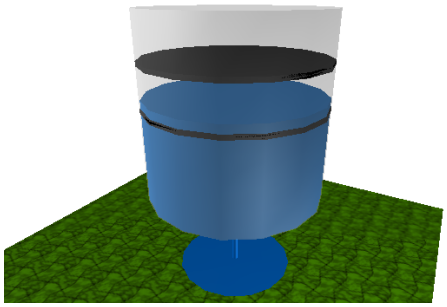

Figure 1: Screen dump from a 3D animation

1. To reduce the effort spent in design iterations compared to current best practice for fault-tolerant embedded control systems by means of multidisciplinary collaborative modelling.
2. To demonstrate the viability of industry-strength tool support for collaborative modelling and co-simulation.
3. To evaluate, in an industrial setting, the effectiveness of collaborative modelling methods and tools for rapid design exploration and tool support.
4. Development of a user and research community in collaborative modelling and co-simulation for embedded systems development.

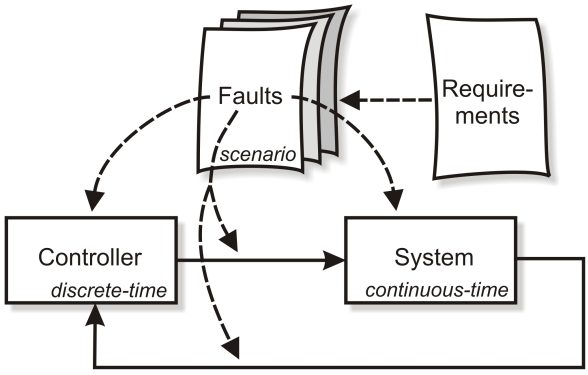

Figure 2: DESTECS: co-simulated controller and system, tested using fault scenarios

The main outputs of the DESTECS project will be an Integrated Development Environment (IDE) and an associated design methodology that supports the collaborative modelling and analysis by co-simulation of candidate designs for dependable embedded control systems.

The IDE will combine 20-sim and Overture, allowing the user to connect models and perform both static checks across the boundaries of the notations as well as co-simulation between them. The co-simulation will enable both a time-triggered approach [8] as well as an event-triggered approach. The methodology will be described in a set of guidelines, intended to be a manual for designers and users of our approach.

There are a number of challenges that we face in achieving the above goals. The IDE needs to handle multiple versions of each model (for design space exploration) and mutliple fault scenarios, all in various combinations. This will require static checks to be performed before co-simulation can occur. In addition, the tools should allow automatic combination and regression testing over various models and fault scenarios.

The IDE should also allow the user to interact with the co-simulation (e.g. pause the simulation) and to inspect the state of the co-simulation. Although the state of two models within a co-simulation may well be extremely complex, this information must to be accessible to the user, hence another challenge lies in finding a practical and natural way of presenting this information.

Our proposed solution is to create a co-simulation *tool connector*. The tool connector would be responsible for co-simulation, controlling the global flow of time, advancing each model as required. The tool connector would also be responsible for allowing the user to control and inspect the co-simulation in a meaningful way. It would also be responsible for combination testing of model variations and fault scenarios, as well as regression testing. The construction of such a tool connector is non-trivial.

On the methodology side, the challenges lie in presenting *useful* guidelines that target co-simulation, but which do not constrain the user unnecessarily. For example, devleopments may begin with informal, natural language requirements, but equally a user may approach DESTECS having previously performed some formal analysis of the requirement. The methodology should support both starting points. We would also wish to provide support for different design patterns, which reflect the user's needs. For example: a simple, single-layered host controller pattern may suit a certain project, whereas a multi-layered controller based on a three-tiered architecture may be necessary for another.

There are also sociological issues involved in collaborating in multidisciplinary domains. For example, the need to become familiar with new paradigms and ways of thinking, such as CT versus DE modelling and fault tolerance techniques. It is hoped that the DESTECS tools and methodology will mitigate the need to study new paradigms in depth, for end users at least.

5. DEPENDABILITY/FAULT TOLERANCE

The DESTECS project aims to support the design of dependable embedded systems by allowing designers to explore, model and reason about the use of fault tolerance techniques. The benefit of combining CT and DE modelling in a single methodology and tools platform is that it allows faults to be considered in parts of the system model in a consistent way. In addition, the effect of faults between the boundary of the models —at the interface level— can be explored. Co-simulation is a key component to achieving this combined approach.

One aim is to allow designers to model faulty components and explore: how individual faults affect the system; how combinations of faults affect the system; and how faults propagate through the system. These are the "scenarios" in Figure 2. 20-sim already allows for components to be realised by different implementations, including implementations with non-ideal behaviour. This could be extended to provide faulty components and perhaps include stochastic metadata to allow for the modelling of intermittent faults.

The methodology component of the DESTECS method

should aid designers in applying fault tolerance methods to deal with these faults. The ultimate aim is to provide sophisticated fault tolerance patterns that designers can appeal to in both their exploration of possible designs and directly in the design of fault-tolerant controllers. Support for these patterns should also be incorporated in the tools platform. For example, in the case of the water tank example from Section 3, the designer might explore how the controller behaves if a sensor incorrectly reports the water level. They could then apply a replication pattern, which introduces multiple sensors to deal with this single point of failure.

Another goal is to allow the designers to describe degraded behaviours within the system and controller (for circumstances where faults may make fully correct behaviour impossible) and fail-safe behaviours, for when the system cannot continue after faults. For example, the personal transporter (see Section 6) is a good example of where we might wish to explore these behaviours, where personal safety is at stake. The use of a formal method such as VDM to model the controller offers the potential to reason about these degraded behaviours and fail-safes, in order to increase confidence in the dependability of the design.

Fault tolerance techniques which we hope to incorporate include error detection, compensation and recovery; redundancy in hardware, software or time; and both backward (e.g. roll-back) and forward (e.g. exception handling) error handling. For embedded systems, there is only limited research on the analysis of the tight time bounds required for these mechanisms. There is a large body of work on identifying and isolating failing nodes, however the integration of these aspects to develop fault tolerant systems cost-effectively remains a challenge [11].

Design support for fault tolerance exists for hard real-time fault tolerance at the low level (custom hardware or specialised facilities such as a global time base); at the scheduling level; at software level (e.g. using SWIFT code transformation); at the architectural level (nested recovery units) and at the formal specification/refinement level. DESTECS is complementary to all these in aiming to make progress towards the selection of fault tolerance strategies at the very early modelling and simulation stage. An important area is the incorporation of stochastic metadata into models; although there is initial work on this aspect in the discrete-event side, there is little investigation of recording stochastic information to support co-simulation of the system as a whole.

DESTECS will also be supporting trade-off analysis such that alternative candidate system solutions can be compared against each other. The achievement of dependability targets of a given system should be incorporated explicitly into the full system life cycle [10]. Relatively little research has been conducted for dependability data to support decision making during design. Work on modelling multi-layered approaches to dependability and alternative fault tolerance strategies in early stages appear promising. Here again, the ability to model and simulate the effects of such an approach in an embedded system context is still lacking.

6. PILOT INDUSTRY STUDIES

The DESTECS project includes three industry partners who have provided case studies that the project can work with. These case studies will provide impetus for the research and challenge problems for the resulting DESTECS methodology and tools.

In addition to the industrial partners, an Industry Follow Group (IFG) has been established. The members of the IFG will be updated on progress through briefings and workshops. They are invited to contribute challenges to the DESTECS project, which the methodology and tools should try to address. Through the IFG, it is anticipated that the DESTECS technology will be exploited in more domain areas. At the start of the project, the IFG consists of 17 members and additional members will be able to join the IFG during the project.

The case studies have been selected to provide a range of embedded systems applications with different forms of complexity, involving engineering heterogeneity (so that collaborative approaches are of interest) and all having the need to provide a predictable level of fault tolerance. They are each chosen to represent a state-of-the-art innovative design problem but they also intended to be recognisable and acceptable for the industry at large, in order to ensure impact. They are each small enough to facilitate an iterative development approach with yearly cycles. The diversity of the case studies will facilitate the generalisation of the research results.

6.1 Document Handling

The Neopost document handling system folds documents, inserts them into envelopes and seals these envelopes. The core operation of the document handling system is the *paper path*. Empty envelopes and prepared documents travel along this paper path. While in transit, the documents are aligned and folded, before being inserted into the envelopes.

The design of the document handling system involves tightly integrated mechatronic disciplines, including mechanics, electronics, and software design. In order to release new generations of the system, it is essential to develop these components concurrently. For concurrent engineering of the paper path's electromechanical components and operational logic, the concept of Hardware-In-the-Loop is deployed. A state-machine model of the electromechanical interface to the paper path allows the embedded controller to be tested before the electromechanical components of the real paper path are integrated.

In DESTECS, we want to lift this concept to a higher abstraction level ("Model-In-the-Loop"). Instead of an artificial state-machine model of the electromechanical interface, we want to directly model the electromechanics in 20-sim. Instead of testing the operational logic at a low level of abstraction, we wish to assess the functionality and dependability at a higher level through VDM.

6.2 Inertial Measurement

Verhaert's Itrack platform is an inertial measurement unit, that is able to measure movement in real-time in 6 degrees of freedom with output rates up to 100 Hz. It is used in applications where high positioning accuracy and high-speed acquisition is required. The core of this product is a complex sensor fusion algorithm which consists of a Kalman filter which processes several sensor signals in parallel, for example accelerometers, gyroscopes and magnetometers. This application will form the second case study inside the DESTECS project.

6.3 Personal Transportation

Chess has created a demonstration mobility platform called the "ChessWay", inspired by the famous Segway personal transporter. This is basically an inverted pendulum with two powerful electric motors to provide active stability. The person standing on the platform can move in a forward direction by moving their centre of gravity forward or decelerate by moving their centre of gravity backward. It is conceptually very simple, but an intrinsically unstable system. Therefore, the control algorithms need to be carefully designed, which is quite challenging and we expect that it can be done elegantly using the DESTECS model-driven approach. In particular, fault detection, isolation and repair strategies will be the major challenge in this case study. These strategies must be carefully selected; efficiently modelled and analysed; and implemented effectively.

7. CONCLUSIONS

The DESTECS project aims to support the rapid development of dependable embedded control systems through co-simulation of 20-sim (continuous time) and VDM (discrete event) models. The use of co-simulation allows designers to model and test both the environment *and* controller early within the delveopment process, reducing time-to-market and increasing confidence in correctness.

We aim produce a tools platform that supports co-simulation, including model versioning, combinatorial tesing, regression testing; and fault injection. We face challenges in producing a tools that can manage these complex, interacting goals in a practical way. We also face challenges in developing methodological guidelines which complement the DESTECS tools and *support* rather than *constrain* the users of our approach.

Acknowledgments

The DESTECS project have partially been funded by the European Commission. We would like to thank Nick Battle and the anonymous reviewers for providing valuable input on the contents of this paper.

8. ADDITIONAL AUTHORS

Additional authors: F. Wouters, Verhaert NV, Belgium, email: `Frederik.Wouters@verhaert.com`.

9. REFERENCES

[1] C. Brooks, C. Cheng, T. H. Feng, E. A. Lee, and R. von Hanxleden. Model engineering using multimodeling. In *1st International Workshop on Model Co-Evolution and Consistency Management (MCCM '08)*, September 2008.

[2] S. L. Campbell, J.-P. Chancelier, and R. Nikoukhah. *Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4*. Springer, 2006. ISBN: 978-0-387-27802-5.

[3] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity – the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, January 2003.

[4] R. Elmstrøm, P. G. Larsen, and P. B. Lassen. The IFAD VDM-SL Toolbox: A Practical Approach to Formal Specifications. *ACM Sigplan Notices*, 29(9):77–80, September 1994.

[5] J. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat, and M. Verhoef. *Validated Designs for Object–oriented Systems*. Springer, New York, 2005.

[6] J. Fitzgerald, P. G. Larsen, and S. Sahara. VDMTools: Advances in Support for Formal Modeling in VDM. *Sigplan Notices*, 43(2):3–11, February 2008.

[7] T. A. Henzinger and J. Sifakis. The discipline of embedded systems design. *Computer*, 40(10):32–40, 2007.

[8] Hermann Kopetz and Günther Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 91(1), January 2003.

[9] J. Hooman, N. Mulyar, and L. Posta. Coupling Simulink and UML Models. In B. Schnieder and G. Tarnai, editors, *Proceedings of Symposium FORMS/FORMATS 2004, Formal Methods for Automation and Safety in Railway and Automotive Systems*, pages 304 – 311, 2004.

[10] M. Kaaniche, J. C. Laprie, and J. P. Blanquart. A Framework for Dependability Engineering of Critical Computing Systems. *Safety Science*, 40(9):731–752, 2002.

[11] K. Kim. Fault-tolerant distributed computing: Evolution and issues. *IEEE Distributed System Online*, 3(7), July 2002.

[12] C. Kleijn. Modelling and Simulation of Fluid Power Systems with 20-sim. *International Journal of Fluid Power*, 7(3), November 2006.

[13] P. G. Larsen, N. Battle, M. Ferreira, J. Fitzgerald, K. Lausdahl, and M. Verhoef. The Overture Initiative – Integrating Tools for VDM. *ACM Software Engineering Notes*, 35(1), January 2010.

[14] Simulink - Simulation and Model-Based Design. http://www.mathworks.com/products/simulink/, 2009.

[15] M. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. PhD thesis, Radboud University Nijmegen, 2009. ISBN 978-90-9023705-3. Available on-line at `www.marcelverhoef.nl/uploads/Main/thesis.pdf`.

[16] M. Verhoef and P. G. Larsen. Interpreting Distributed System Architectures Using VDM++ – A Case Study. In B. Sauser and G. Muller, editors, *5th Annual Conference on Systems Engineering Research*, March 2007. Available at http://www.stevens.edu/engineering/cser/.

[17] M. Verhoef, P. G. Larsen, and J. Hooman. Modeling and Validating Distributed Embedded Real-Time Systems with VDM++. In J. Misra, T. Nipkow, and E. Sekerinski, editors, *FM 2006: Formal Methods*, pages 147–162. Lecture Notes in Computer Science 4085, 2006.

[18] M. Verhoef, P. Visser, J. Hooman, and J. Broenink. Co-simulation of Real-time Embedded Control Systems. In J. Davies and J. Gibbons, editors, *Integrated Formal Methods: Proc. 6th. Intl. Conference*, Lecture Notes in Computer Science 4591, pages 639–658. Springer-Verlag, July 2007.

[19] M. Wetter. Modelica-based Modelling and Simulation to support Research and Development in Building Energy and Control Systems. *Journal of Building Performance Simulation*, 2(2):143–161, June 2009.