Hardware In the Loop for VDM-Real Time Modeling of Embedded Systems

José Antonio Esparza Isasa, Peter Würtz Vinther Jørgensen and Peter Gorm Larsen
Department of Engineering, Aarhus University, Finlandsgade 22, Aarhus, Denmark

Keywords: Hardware In the Loop (HIL), Modeling, Embedded Systems, VDM-RT.

Abstract: This paper introduces a generic solution for gradually moving from a model of an embedded system to include embedded hardware and software components into the simulation of the model. Our technique enables combined execution (co-execution) of system components models expressed in the VDM-RT formalism with actual hardware/software realizations through the application of Hardware In the Loop (HIL) simulation. Introducing such component realizations in the simulation increases the fidelity of the simulation outcome, thus enabling improved prediction of properties for the system realization.

1 INTRODUCTION

When developing embedded systems, complexity is typically high and the possible design space with alternative hardware components to use internally is huge. In order to master complexity it is possible to introduce precise abstract models that can be analysed in different ways, one of them being simulation.

HIL simulation is a well known technique for enabling incorporation of hardware components in the simulation of some Device Under Test (DUT). This technique is particularly helpful when comparing alternatives in a simulation context and for ensuring the fidelity of the models for the purpose of the analysis required. A common application of HIL is testing. In a testing scenario typically a hardware system realization is tested against a simulated environment. The HIL simulation application proposed in this paper is different, since we aim at using HIL as a way to co-execute system models with real hardware and software implementations deployed on target. In order to develop the system models we use the Vienna Development Method Real-Time (VDM-RT) dialect (Verhoef et al., 2006), which is a formal notation enabling modeling and analysis of distributed systems. VDM-RT is supported by an Eclipse-based development platform called Overture (Larsen et al., 2010a) that includes debugging capabilities and enables incorporation of code written in Java (Nielsen et al., 2012).

Modeling of the embedded system gives the ability to focus on the important development challenges (e.g. correction of control algorithms) by applying a higher level of abstraction to things of less importance (e.g. details of hardware drivers). VDM-RT and its tool support further enables validation of timing requirements (Fitzgerald and Larsen, 2007), inspection of the execution trace, generation of the implementation in a programming language (CSK, 2007; Jørgensen and Larsen, 2013) and common consistency mechanisms associated with a formal notation (e.g. invariants and generation of proof obligations).

This paper is structured such that Section 2 introduces the technologies used in our work. Then Section 3 provides an overview of the setup for HIL simulation of embedded systems expressed in VDM-RT. This is followed by a prototype application using this setup in Section 4. Afterwards Section 5 describes the future work planned and Section 6 provides references for related work. Finally the paper concludes in Section 7.

2 TECHNOLOGIES USED

This section presents the modeling, software and hardware technologies that have been used to create and explain the HIL setup.

2.1 Overture with support for VDM-RT

VDM supports different kinds of collections (sets, sequences and mappings) and has a significant focus on

Esparza Isasa J., Jørgensen P. and Larsen P..
Hardware In the Loop for VDM-Real Time Modeling of Embedded Systems.
DOI: 10.5220/0004698902090216
In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2014), pages 209-216
ISBN: 978-989-758-007-9
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

logic expressions in its way to model. In this paper we deal with the VDM-RT notation, which is an object-oriented notation enabling modeling of distributed systems with an explicit notion of global time. In VDM-RT it is possible to model different CPUs and their connections using busses. Multiple threads can exist at each CPU but at any point in time only one thread can be executing at each CPU. In a simulation all expressions and statements that are evaluated add a default increase in global time when being performed. Special *cycles* and *duration* statements exist for overruling the default time increments either relative to the CPU speed or with a fixed amount of time, respectively.

2.2 Prototype Hardware

The DUT selected for this work is a System on Chip (SoC). This kind of platform allows the combined deployment of hardware and software components in the same piece of silicon. We have considered it appropriate for this work since it allows testing of the HIL concept for VDM-RT in a more complete manner, taking into account both hardware and software components. These components can act together or in isolation while running on the same piece of hardware.

In order to monitor the output produced by the DUT on its digital buses we have used a logic analyzer (Saleae Logic 8). This device samples a number of logical lines over a period of time. Any changes on the lines during this period of time will be logged. The logic analyzer selected for this work is connected to a PC and it is possible to control it from custom software, created using a Software Development Kit (SDK) provided by the manufacturer.

3 HIL SETUP DESIGN

This section presents an overall description of the system and its hardware and software architecture.

3.1 Overall System Description

The main objective of this system is to allow a step-wise transformation of models of functionality into components that implement that functionality. In order to benefit from the models one can combine the implemented components with the models of those that still have not been implemented. In this way it is possible to execute the components in a single model implementation co-execution. This is the key idea behind the HIL system presented in this paper, which has not been applied previously for VDM-RT. Figure 1 illustrates this concept with an example based on the design of an embedded system that enters three stages: Acquire Data, Process and Provide Output. The design of such a system starts with the modeling of the complete system functionality. This is illustrated in the upper row of Figure 1 (Full Modeling). As it can be seen all the components are modeled (represented on the VDM side) and no components have been implemented on the target (the HW side). After the system has been modeled the component providing the Process functionality is implemented. It is the intention to substitute the modeled Process component with its correspondent implementation and still use the rest of the model for system simulation. This approach is shown in the lower row of Figure 1. At this point it is possible to start the system simulation on the VDM-RT side in the Acquire Data stage and then the HIL system will hand-over the execution of the Process stage to the implementation deployed on target. Once Process has completed, execution will return from the target embedded system to the VDM-RT model of this functionality and continue with the Provide Output stage.

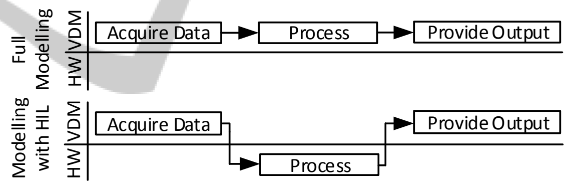

Figure 1: Device modeling and combined device modeling with HIL.

By applying the approach proposed in this paper the design engineer is able to benefit from: 1) the expressiveness of VDM-RT to represent real-time constraints and system properties 2) the simulation capabilities of Overture and 3) the insight gained through prototyping using a combined-modeling prototyping approach.

3.2 Hardware

In order to create a system that supports the functionality presented above we have used a number of external hardware components. This hardware architecture is presented in Figure 2 using a System Modeling Language[1] (SysML) Block Definition Dia-

[1]SysML (Sandford Friedenthal, 2008) extends the Unified Modelling Language (UML) and allows modelling of both hardware and software in the same system description. In our work we use it for describing the structural and dynamical aspects of the HIL setup.

gram (BDD). This diagram shows the components that comprise the system in a hierarchical manner without showing their connections. These components are:

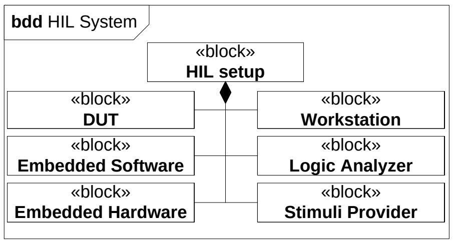

Figure 2: Main components of the HIL setup shown in a SysML BDD.

HIL Setup: is the top level representation of the HIL system. This block represents the system as a whole and not with a concrete piece of hardware.

Workstation: is a computer running Overture and the required software infrastructure to communicate with the rest of the hardware components connected to it.

DUT: is the external device that contains a partial implementation of the system. This partial implementation contains a certain functionality that will be triggered from the model execution. The DUT is composed of Embedded Software and Embedded Hardware, i.e. the functionality under test can be implemented in hardware and/or software components.

Stimuli Provider: is an electronic device able to provide digital signals to the DUT. These external inputs can represent any device with a logical interface, e.g. a sensor measuring physical parameters or a different digital hardware block.

Logic Analyzer: is a Digital Acquisition (DAQ) board enabling capturing of the evolution of a number of digital signals over time. The main purpose of this block is to monitor the digital outputs provided by the DUT.

In order to show the connection between the most relevant hardware blocks we have used a SysML Internal Block Diagram (IBD), presented in Figure 3. We have used standard ports to represent discrete logical communication and flow ports to represent continuous signals. The Workstation uses USB and serial busses to communicate with the rest of the blocks presented in the IBD. The communication between the Stimuli Provider and DUT is made through a logical bus that maps the Stimuli Provider outputs to the DUT inputs. The same kind of bus is used between the DUT and the Logical Analyzer to monitor the DUT output values. Additionally two dedicated inputs in the Logic Analyzer are used to analyze a DUT Pulse Width Modulated (PWM) output and the function execution time (Duration Pin).

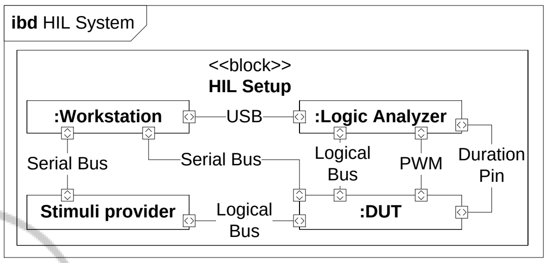

Figure 3: SysML Internal Block Diagram showing the hardware connections to the DUT.

The software components running on the Workstation are presented in the Section 3.3. The Embedded Software and Hardware blocks are not discussed since they are dependent on the system under study. An example of those will be given in section 4.

3.3 Software and Modeling Components

The workstation software and modeling components of the HIL setup are shown in relation to each other in the class diagram in Figure 4. Each of the components are treated individually in the description below. To supplement the description of the software and modeling structure in this subsection, the execution flow will be covered in Section 4.2.

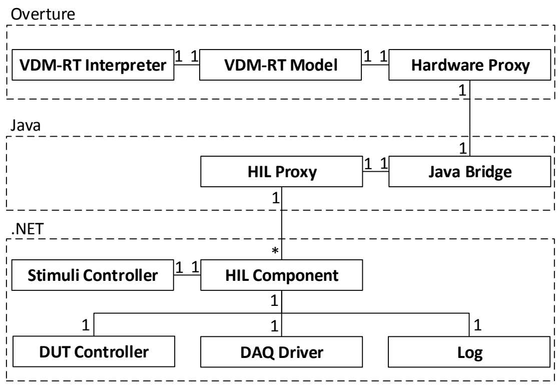

Figure 4: The software and modeling components of the HIL setup shown in a class diagram.

VDM-RT Interpreter: executes the VDM-RT Model representing the embedded system under consideration.

Hardware Proxy: specifies the interface of the hardware being controlled from the VDM-RT execution. All invocations are initiated in VDM-RT and communicated through the Java Bridge before they reach the physical hardware.

Java Bridge: enables delegation of VDM-RT functionality execution to Java. In this way one can invoke a VDM-RT operation or function and have the body specified and executed as a Java method. The Java Bridge does automatic conversion and transferring of input and output values between Java and VDM-RT.

HIL Proxy: is a thin software component that accepts inputs from the Java Bridge inside Overture and relays the invocations originating from the VDM-RT execution to a HIL component responsible for communicating with the hardware used in the HIL setup. The HIL proxy is interoperability glue that enables communication between Overture and the hardware.

HIL Component: is responsible for communicating directly with the hardware used in the HIL setup through serial connections. Although we only connect to a single instance of the HIL Component it would be possible to extend this to multiple instances as indicated in Figure 4 and described in Section 5.5. Finally, the HIL Component uses the Log for writing the data acquired from the DAQ to the file system.

Stimuli Controller: instructed by the HIL Component the Stimuli Controller provides the DUT with input signals prior to executing the functionality under test.

DUT Controller: instructs the external DUT to execute some function in software or hardware.

DAQ Driver: provides the software interface enabling the HIL component to launch the DAQ so the data needed can be acquired from the DUT execution.

4 HIL SETUP PROTOTYPE

This section presents the initial setup for the HIL system by describing the prototype as well as the co-execution of the model and the partial implementation.

4.1 Prototype Description

The current prototype for this case implements a control algorithm for a mechanical device. This algorithm is composed of several stages: 1) Initialization, 2) Regulation Loop and 3) Results Logging. We have modeled the algorithm in VDM-RT and continued by producing a preliminary implementation of the Regulation Loop that can be deployed to target. The aim of this first approach is to combine in a single execution trace the models of Initialization and Results Logging with the implementation of Regulation Loop that is deployed on the DUT. Additionally, we are interested in measuring the execution time of the regulation cycle on real hardware in order to verify if real-time constraints are met. This real-time information will also be incorporated to the VDM-RT models afterwards. The following sections elaborate on the structure and the functionality provided by the Embedded Hardware and Software blocks which the DUT is composed of.

4.1.1 Embedded Hardware

The system has been implemented in a PSoC 5 with an ARM Cortex M3 processor and the additional hardware components presented in the SysML BDD presented in Figure 5. These components support the communication with external logical interfaces (Actuator Logical Interface and Sensor Logical Interface), the PWM control of an actuator, the communication with the workstation running the VDM-RT model (UART) and finally the pin toggling for time measurements (Duration Pin).

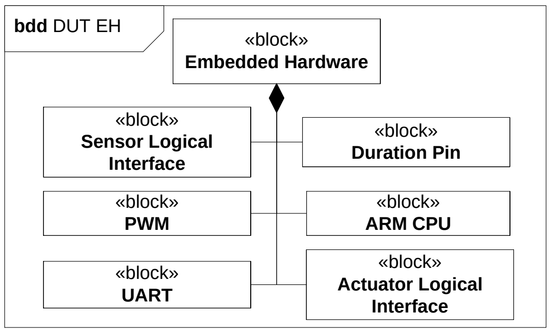
bdd DUT EH
«block» Embedded Hardware
«block» Sensor Logical Interface
«block» Duration Pin
«block» PWM
«block» ARM CPU
«block» UART
«block» Actuator Logical Interface

Figure 5: The Embedded Hardware (EH) of the DUT shown in a SysML BDD.

Additional details on these components can be found in the documentation provided by the manufacturer (Cypress, 2013).

4.1.2 Embedded Software

The embedded software executing on the DUT is deployed on the ARM processor. The software is composed of three main blocks: Hardware Drivers, Functionality Under Test (FUT) and HIL Support as shown in Figure 6. The Hardware Drivers provide the necessary software support to interface the hardware blocks from software. HIL Support contains the logic to start the execution of the FUT upon request from the DUT Controller running on the workstation. This functionality is structured in three phases:

1. **Wait for Command:** The system waits for a possibly parametrized command over UART describing the functionality to execute.

2. **Serve HIL Request:** The system initiates the execution of the FUT. This can be surrounded by pin toggling operations to measure the execution time of the function. Optionally additional pin toggling can be performed inside the function to measure the time it takes to reach different points during the execution.

3. **Return to DUT Controller:** Once the system has completed the execution of the FUT it returns to the DUT controller so it can stop signal sampling and notify the model execution environment to resume model execution. Additionally it can return result values produced by the execution of the FUT.

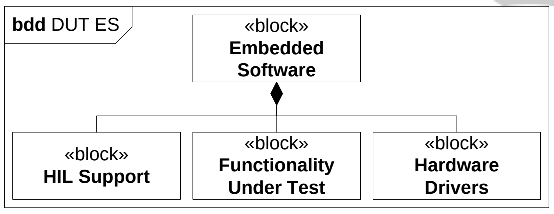

Figure 6: The Embedded Software (ES) of the DUT shown in a SysML BDD.

The FUT in this case is a control sequence implemented in C and outlined in code Listing 1. This presents a regulation cycle composed of three phases sensor values retrieval (`readSensors`), calculation of the regulation parameters (`regulate`), and finally command of actuators according to those (`produceLogicalOutput` and `modifyPWM`). Note that these functions have been surrounded by pin toggling instructions (`toggleDurationPin`) to enable the calculation of execution time.

Listing 1: Control sequence implementation.

```
toggleDurationPin();
readSensors();
regulate();
produceLogicalOutput();
modifyPWM();
toggleDurationPin();
```

4.2 Model Execution

Hardware control is initiated through the Hardware Proxy that specifies the functionality of the physical hardware accessible to the VDM-RT model (e.g. providing the DUT with input signals or performing hardware or software computations). Using the duration or cycles statements the modeller can specify the number of cycles (for the VDM-RT CPU) or absolute time it takes to execute a hardware or software function. The global time of VDM-RT will then progress accordingly. Furthermore, the Java Bridge executes atomically, i.e. no thread will be swapped in until execution returns to the model.

Invocations to the Hardware Proxy are initiated in the VDM-RT model, and then relayed by the HIL Proxy to the HIL Component as shown in Figure 7.

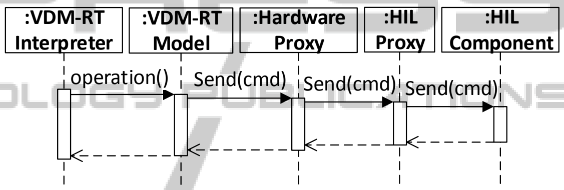

Figure 7: Sequence diagram showing how invocations originating from the VDM-RT interpreter reaches the HIL Component.

From here the HIL Component either invokes the Stimuli Controller or the DUT Controller. This process is represented with a UML sequence diagram in Figure 8. In this diagram the HIL Component first signals the Stimuli Controller to supply the DUT with input signals. Next it launches the DAQ using the corresponding driver and finally the DUT is instructed to perform some computation via the DUT Controller. While the DUT is executing the DAQ is continuously sampling the state of the DUT, which is logged to a file at the end of execution. Invocations from the VDM-RT interpreter, Hardware Proxy, HIL Proxy and HIL Component (including data logging) have been omitted from Figure 8.

4.3 HIL Execution Results

The co-execution of the model and the implementation made it possible to validate the real-time behaviour of the FUT component as well as the correct operation of its control logic. We have been able to validate that the time slot allocated to the regulation cycle is not overrun and obtained a precise time measurement that can be incorporated to the models. Additionally it has been possible to exercise the implemented control logic together with the modeled components on a number of scenarios. Finally, this HIL setup has facilitated the stepwise implementation from a system level design approach.

Our approach is further supported by tools for automatic generation of the system implementation. For software components it is possible to produce Java and C++ from VDM using code generators (CSK, 2007; Jørgensen and Larsen, 2013). However, for hardware components there is currently no support for generating the implementation in a hardware description language. Even though automatic hardware generation is not available in VDM-RT this language can be used to study component-level time constraints and to some extent hardware/software partitioning problems (Isasa et al., 2012).

5 FUTURE WORK

The work presented in this paper is a proof of concept showing that HIL can be implemented for VDM-RT. We are planning to extend this initial prototype so more complex analysis can be performed. Such analysis would enable study of further real-time aspects of the system, system response in the analog domain and the application of this setup in a HIL network.

5.1 Profiling Function Execution

As described above it is possible with the current HIL setup to measure the execution time of the implemented functionality by manual analysis of the waveforms captured. With the current HIL setup one would have to annotate the VDM-RT model with the execution time manually, i.e. using the duration statement parametrized with a number indicating the execution time in nanoseconds. This future work item suggests improving this process so that the VDM-RT model execution is automatically annotated with the execution time as measured from the DUT.

In this scenario, invocations originating from the Hardware Proxy are all intercepted by the HIL Proxy which has access to the core functionality of the Overture platform such as the AST and the interpreter executing the VDM-RT model. Thus it would be possible to extend on the HIL Proxy and inject the duration statement programmatically using the Java functionality of the Overture platform. In this way it is possible to make the global time of VDM-RT progress by the time measured from the execution of the DUT function transparently when this is being executed by the VDM-RT interpreter.

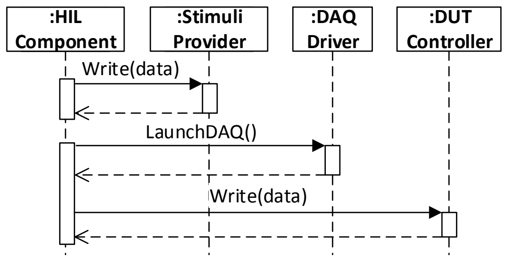

Figure 8: Sequence diagram showing the Workstation communicating with the DAQ Driver and the DUT Controller.

5.2 Specification and Validation of RT Deadlines

The work of (Fitzgerald and Larsen, 2007) presents an extension to VDM-RT, enabling the validation of system-level timing properties. *Validation conjectures* are descriptions of temporal relations between system level events, which can be evaluated over the VDM-RT execution trace. Later this work has been extended in (Ribeiro et al., 2011) to include runtime validation. Predefined standard forms of validation conjectures have been defined, directly supporting the validation of a deadline or separation between two events, called the *trigger* and the *response*. The trigger event could be the press of a button, and the corresponding response may be the update of a display. From the standard forms, more specific validation conjectures can be constructed. In this way it is possible to specify system level timing properties and then automatically validate these against the execution time as measured from the execution of the DUT. In case validation conjectures are violated during the execution of a VDM-RT model this can be analysed graphically using the RealTime Log Viewer available in the Overture tool (Larsen et al., 2010b).

5.3 Analog Data Acquisition

The current HIL setup only supports the acquisition of digital signals. We would like to extend this to the acquisition of analog signals so it will be possible to monitor other kind of hardware blocks such as Digital to Analog Converters or analog sensors among others.

We have done a preliminary integration of an analog acquisition board with the HIL setup proposed in this paper using a preliminary SDK. The current results show that it is possible to use this device in a similar way to the logic analyzer. However we still need to explore this approach further.

5.4 Power and Energy Consumption Profiling

Analog DAQ enables measuring of energy consumed by the DUT using a shunt resistor for monitoring the evolution of voltage over time in the resistor. Voltage variations could be logged for later processing in order to calculate consumed energy and power of different components

This allows the engineer to profile the energy consumption of different components in the system at different levels of abstraction. One of the possibilities is to profile the energy consumption of the hardware blocks used during system operation. A second possibility is to profile the energy consumption of code sections by annotating the different software routines in the real-time model with figures describing the energy consumed. This can be extended to profiling of energy consumption for communication with external systems, by monitoring the energy consumed of the communication interfaces. These measurements could be used for annotating the VDM-RT model and associated with the communication abstractions available in VDM-RT.

5.5 Multiple HIL Nodes

The current HIL setup only considers the situation where the HIL Proxy connects to a single HIL component. However, for situations where the system is composed of multiple hardware platforms communicating over a network while being physically far apart, it may be better to have a HIL Component for each of the hardware platforms composing the system. In this way the HIL Proxy can be connected to several HIL Components each being responsible for performing DAQ and data logging on a hardware platform. This case was briefly mentioned in Section 3.3 and indicated by the multiplicity of the association between the HIL Proxy and HIL Component in the class diagram in Figure 4.

5.6 Waveform Analysis

To ease the analysis of logical signals it is crucial to use a tool that enables their graphical representation over time and automates at least partial waveform analysis. Current software for this purpose allows the definition of expressions to evaluate signal properties automatically and the definition of time markers to measure time intervals among other facilities. We would like to benefit from these features by using an ad-hoc signal analysis tool such as IMPULSE (Haber, 2013) or GTKWave (GTKWave, 2013). In order to use these tools the HIL setup proposed in this paper must implement the generation of time logs in Value Change Dump format (IEEE, 1996). Using this format will allow the generation of smaller files that are easier to handle by the tools.

6 RELATED WORK

HIL simulation has been used extensively in safety-critical sectors such as avionics (Peleska, 2002) and automotive (Schuette and Waeltermann, 2005; Schulte et al., 2012). It has primarily been used in those domains for testing, which would otherwise be very expensive using traditional approaches such as physical prototyping. However, the industrial take-up of HIL testing is spreading rapidly to less critical domains (Fathy et al., 2006), also due to the speed of modern processors as well as different commercial solutions arising (NI, 2012; MathWorks, 2011).

Currently there are several approaches to HIL. This technique can be used to provide a certain DUT with a simulated environment in order to validate its control logic. Chudy et al. use a HIL component running a flight simulation composed of a number of physical models to test avionic systems (Chudy and Rzucidlo, 2012). A similar approach is used in (Schlegel et al., 2002) to test an automatic gearbox.

The approach presented in this paper differs from the ones described above since the HIL component is used so partial system implementations can be co-executed with a VDM-RT model, thus enabling step-wise system development. Additionally VDM-RT facilitates the development of quality systems using the consistency mechanisms previously described. This means that no model of any kind (neither physical, nor logical) is executing in the HIL component.

Orth et al. propose the application of Petri nets to develop Programmable Control Logic software and describe its combination with Software/Hardware/System in the loop simulation (Orth et al., 2005). However the setup used in their case study is specific for industrial automation and not as generic as the one proposed in this paper. Previous work also exists concerning transformation of Petri nets models into asynchronous hardware components (Carmona et al., 2004). By combining our approach with the ideas proposed by Orth et al. and the transformation approach suggested by Carmona et al. one could develop a similar setup to support the co-execution of Petri net models with hardware/software system realizations.

7 CONCLUDING REMARKS

In this paper we have presented the initial work we have carried out to enable HIL simulation using models written in VDM-RT. This proof-of-concept demonstrates the potential of our approach but naturally there is significant future work remaining to make this a useful feature for industrial use.

REFERENCES

Carmona, J., Cortadella, J., Khomenko, V., and Yakovlev, A. (2004). Synthesis of asynchronous hardware from petri nets. In *Advances in Petri Nets, LNCS 3098*, pages 345–401. Springer.

Chudy, P. and Rzucidlo, P. (2012). Hil simulation of a light aircraft flight control system. In *Digital Avionics Systems Conference (DASC), 2012 IEEE/AIAA 31st*, pages 6D1–1–6D1–13.

CSK (2007). VDMTools homepage. *http://www.vdmtools.jp/en/*.

Cypress (2013). http://www.cypress.com/. Cypress official website.

Fathy, H. K., Filipi, Z. S., Hagena, J., and Stein, J. L. (2006). Review of Hardware-in-the-Loop Simulation and its Prospects in the Automotive Area. In *Proc. SPIE 6228, Modeling and Simulation for Military Applications*. http://dx.doi.org/10.1117/12.667794.

Fitzgerald, J. S. and Larsen, P. G. (2007). Triumphs and Challenges for the Industrial Application of Model-Oriented Formal Methods. In Margaria, T., Philippou, A., and Steffen, B., editors, *Proc. 2nd Intl. Symp. on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2007)*. Also Technical Report CS-TR-999, School of Computing Science, Newcastle University.

GTKWave (2013). http://gtkwave.sourceforge.net/. GTKWave official website.

Haber, T. (2013). http://toem.de/index.php/impulse. IMPULSE official website.

IEEE (1996). IEEE Standard Hardware Description Language Based on the Verilog(R) Hardware Description Language. *IEEE Std 1364-1995*.

Isasa, J. A. E., Larsen, P. G., and Bjerge, K. (2012). Supporting the Partitioning Process in Hardware/Software Co-design with VDM-RT. In *Proceedings of the 10th Overture Workshop 2012*, School of Computing Science, Newcastle University.

Jørgensen, P. W. and Larsen, P. G. (2013). Towards an Overture Code Generator. In *Submitted to the Overture 2013 workshop*.

Larsen, P. G., Battle, N., Ferreira, M., Fitzgerald, J., Lausdahl, K., and Verhoef, M. (2010a). The Overture Initiative – Integrating Tools for VDM. *SIGSOFT Softw. Eng. Notes*, 35(1):1–6.

Larsen, P. G., Lausdahl, K., Ribeiro, A., Wolff, S., and Battle, N. (2010b). Overture VDM-10 Tool Support: User Guide. Technical Report TR-2010-02, The Overture Initiative, www.overturetool.org.

MathWorks (2011). http://www.mathworks.com/. Simulink official website.

NI (2012). Hardware-in-the-Loop (HIL) Testing. http://www.ni.com/hil/.

Nielsen, C. B., Lausdahl, K., and Larsen, P. G. (2012). Combining VDM with Executable Code. In Derrick, J., Fitzgerald, J., Gnesi, S., Khurshid, S., Leuschel, M., Reeves, S., and Riccobene, E., editors, *Abstract State Machines, Alloy, B, VDM, and Z*, volume 7316 of *Lecture Notes in Computer Science*, pages 266–279, Berlin, Heidelberg. Springer-Verlag. ISBN 978-3-642-30884-0.

Orth, P., Bollig, A., and Abel, D. (2005). Rapid Prototyping of Sequential Controllers With Petri Nets. In *Proceedings of the 16th IFAC World Congress*.

Peleska, J. (2002). Hardware/Software Integration Testing for the new Airbus Aircraft Families. In Schieferdecker, I., König, H., and Wolisz, A., editors, *Testing of Communicating Systems XIV. Application to Internet Technologies and Services*, pages 335–351. Kluwer Academic Publishers.

Ribeiro, A., Lausdahl, K., and Larsen, P. G. (2011). Run-Time Validation of Timing Constraints for VDM-RT Models. In Wolff, S. and Fitzgerald, J., editors, *Proceedings of the 9th Overture Workshop*, number ECE-TT-2 in Technical Report Series, pages 4–16.

Sandford Friedenthal, Alan Moore, R. S. (2008). *A Practical Guide to SysML*. Morgan Kaufman OMG Press, Friendenthal, Sanford, First edition. ISBN 978-0-12-374379-4.

Schlegel, C., Bross, M., and Beater, P. (2002). Hil-simulation of the hydraulics and mechanics of an automatic gearbox. In *Proceedings of the second International Modelica Conference*, pages 67–75.

Schuette, H. and Waeltermann, P. (2005). Hardware-in-the-Loop Testing of Vehicle Dynamics Controllers – A Technical Survey. In *Proc. of SAE05*, Detroit. SAE. SAE Technical Paper 2005-01-1660.

Schulte, T., Kiffe, A., and Puschmann, F. (2012). Hil simulation of power electronics and electric drives for automotive applications. In *Electronics*, volume 12, pages 130–135.

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