Model-based specification, analysis and synthesis of servo controllers for lithoscanners

Ramon R.H. Schiffelers *
ASML
De Run 6501
5504 DR Veldhoven, The Netherlands
r.r.h.schiffelers@tue.nl

Wilbert Alberts
ASML
De Run 6501
5504 DR Veldhoven, The Netherlands
wilbert.alberts@asml.com

Jeroen P.M. Voeten †
Eindhoven University of Technology
Den Dolech 2
5600 MB Eindhoven, The Netherlands
j.p.m.voeten@tue.nl

ABSTRACT

ASML is the world's leading provider of complex lithography systems for the semiconductor industry. Such systems consist of numerous servo control systems. To design such control systems, a multi-disciplinary model-based development environment has been developed. It is based on a set of domain specific languages (DSLs) describing A) the transducers and control logic, i.e. the application; B) the relevant subset of the hardware, i.e. the platform; and C) the mapping of the application on the platform. Models specified with these DSLs are used for different types of analysis, for example load prediction of computing nodes and networks between them. Furthermore, the behavioral specification present in the models is transformed into efficient C code that is executed in a hard real-time setting. Finally, the models are used during startup of a twinscanner to initialize the servo controllers and their execution platforms, and to schedule the control blocks on the computing nodes.

1. INTRODUCTION

ASML is the world's leading provider of complex lithography systems (Figure 1) for the semiconductor industry. Such systems consist of numerous servo control systems to control, for instance, the positioning of a 15 kg wafer positioning module in six degrees of freedom with nanometer accuracy at acceleration speeds exceeding 110 G. This results in tight hard real-time requirements on their implementation.

Within ASML, these servo control systems are developed according to the Control Architecture Reference Model (CARM) that consists of different layers to describe the control logic and the execution platform of a lithoscanner at different levels of abstraction.

To manage the increasing complexity of control applications, and to deal with hybrid computation platforms, analysis is required early in the design process. Furthermore, to deal with high demands on time-to-market and quality of the designs, a seamless parallel multi-disciplinary development process is used/needed, combined with deterministic, automatic synthesis techniques.

*Dr. Schiffelers also holds a position as assistant professor at the Eindhoven University of Technology

†Dr. Voeten also holds a position as Research Fellow at the Embedded Systems Institute (ESI).

Figure 1: ASML NXE Lithoscanner

The core of CARM relies on a set of domain-specific languages (DSLs) that formalize in a coherent, consistent and unambiguous way the domain concepts governed by the different CARM layers. The domain specificity of the languages allows a tight fit with the established terms, concepts and needs for each design discipline. Models specified with these DSLs are the first-class citizens (pivot) in the design process consisting of specification, analysis, and synthesis phases:

- *Specification* By means of a multi-disciplinary integrated development environment (IDE), formal models are developed that describe A) the control logic in terms of (interconnected) servo networks and transducers (sensors and actuators), typically specified by mechatronic engineers; B) the (hybrid) computation platform in terms of single/multi-core processors as well as accelerator support (e.g. FPGAs), typically specified by electrical engineers; C) the mapping how the control logic is deployed and scheduled on the computation platform, typically specified by embedded software engineers. Incorporating different levels of abstraction into the DSL framework enables the disentanglement between the various pieces of information that originate in the several disciplines, reducing the complexity of the (parallel) design process. The IDE provides the design engineers with feedback on the models early in the design process improving quality of the designs.

- *Analysis* Analysis models are used as input for key decisions for which it has to be proven that a design will work. Furthermore, verification of the designs reduces the risk on errors during integration, or eliminates the integration effort altogether. More specifically, in CARM, A) models are transformed to formal specification languages such as data flow languages to analyze their correctness (e.g. no deadlock); B) models are transformed automatically into formal, executable (simulation) models to verify upfront whether the timing requirements are met and to predict the effect of control loop changes and/or platform changes.
- *Synthesis* The same models used at design time are used A) during the build by code generators to generate software that is executed on the lithoscanners; B) during start-up of a lithoscanner, to initialize the servo controllers, to configure the computation platforms, and to schedule control blocks over computing nodes.

The main contribution of this paper is that we show that DSLs, including their graphical and textual concrete syntaxis, and text-to-model (T2M), model-to-model (M2M), and model-to-text (M2T) transformations allow to define an integrated design environment to design, analyse and synthesize control systems. The industrial relevance of this work is underlined by its usage within ASML to develop hard real-time control systems for lithoscanners.

The outline of this paper is as follows. The domain of interest and CARM are described in Section 2. Section 3 describes the developed DSLs, their relations, and the modeling environment. Several types of analysis, based on these models, are described in Section 4. In Section 5, it is outlined how the models are used during the several synthesis phases. Related work is discussed in Section 6. Finally, we conclude with some experiences and directions for future work.

2. SERVO CONTROL DOMAIN AND CARM

2.1 Servo control domain

Control theory is defined as an interdisciplinary branch of engineering and mathematics that deals with the behavior of dynamic systems. The external input of a system is called the reference. When one or more output variables of a system need to follow a certain reference over time, a controller manipulates the inputs to a system to obtain the desired effect on the output of the system.

A typical application of control theory is a cruise control system. Such system provides the ability to set and maintain the speed of car. The desired speed, as set by the driver, is the system's reference. The cruise control system actuates the throttle position in such way that the actual speed, as measured by the speed sensor, matches the reference as good as possible. This form of control is also called *closed loop* control, or servo control.

2.2 CARM

CARM divides the controller designs in 4 layers: *application*, *process control*, *transducers* and *connectivity*.

The application layer is responsible for providing the reference and the actions of the driver, like setting the desired speed or turning the cruise control off. The transducer layer implements the behavior of sensors and actuators. In case the cruise control system is implemented in a digital way, this layer also needs to take care of measuring the speed of the car and converting it in a number (in SI units). The process control layer receives (the converted) sensor information and compares it to the reference. Based on the difference, new settings for the actuators are calculated. As the controller and the transducers tend to be deployed on different (computing) nodes, a connectivity layer is used to transport signals from the controller to the (physical) transducers and vice versa. In the cruise control system, the connectivity layer should ensure that the output of the speed sensor is correctly provided to the control system. In more complex systems, this might require deployment of physically separated nodes that communicate with each other via a network.

3. SPECIFICATION

The core of CARM relies on a set of DSLs. In Section 3.1, we describe the design principles used to design the DSLs. Given the size of the languages, their contents can only be discussed very briefly in this paper. The developed multi-disciplinary IDE is described in Section 3.2.

3.1 Domain Specific Languages

A domain specific language is defined by means of a domain model (metamodel) and accompanying constraints. The domain model represents domain concepts (entities) and their relationships. The domain models are described using (Ecore) class diagrams ([1]), where *classifiers* represent concepts, and *associations* represent relationships. Another advantage of such explicit domain model is its formality. Furthermore, during the design of a domain model, it is not necessary to deal with concrete syntax. After the definition of the domain model, one or more concrete textual and/or a graphical syntax(es) for the language can be defined.

For a language, many different metamodels can be defined. For example, a metamodel representing the abstract syntax tree (AST), which resembles a BNF (Backus-Naur Form) language specification, can be defined using containment relations only. An instance of such a metamodel is a tree of language elements, and additional scope (reference) resolving is required to map identifiers to objects. A metamodel can also be developed using (non-containment) relations/references, resulting in a graph, for which reference resolving is not needed.

It is possible to include static type information for variables and omit type information for the operators and value nodes of expression trees. In this case, additional type computing is required in order to determine whether expressions are type-correct. Alternatively, type information on all nodes of expression trees can be included.

In the CARM DSL framework, it was chosen to design the metamodels of languages such that instances of these metamodels are fully statically typed graphs. In this way, A) transformations do not have to deal with issues related to type computing and scope resolving; B) transformations

only have to deal with mappings between concepts; and C) the metamodel abstracts away from parser-related elements that are often contained in language definitions.

The class diagram of a language contains static semantic constraints such as *every Servogroup contains at least one control block*. Static semantic constraints such as *within a servogroup, the names of the control blocks should be unique* cannot be expressed using class diagrams. In order to define such static semantic constraints, the *Object Constraint Language (OCL)* ([2]) is used.

Figure 2 shows the developed DSLs organized according to the Y-chart paradigm [3].

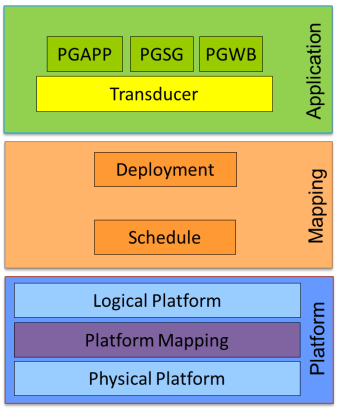

Figure 2: CARM Domain Specific Languages

The *Application level* describes the (control) logic; the *Platform level* provides the means to implement the logic; and the *Mapping level* maps each element from the Application level to an element in the Platform level. Within these Y-chart levels, different levels of abstraction present in the DSL framework reduce the complexity (entanglement) of the multi-disciplinary parallel design processes.

3.1.1 Application level

The application level contains the description of the control application. It consists of the control logic described by means of the PGAPP, PGSG, and PGWB languages, and the description of the transducers in the transducer language. The *PGAPP* defines a network of servogroups and transducer groups. Both transducer and servogroups contain (data) ports. Information exchange between them can be established by means of connecting these dataports. Servogroups consist of interconnected control blocks. Control blocks contain input and output (data) ports, as well as the behavior of the block in terms of input/output relations, specified by imperative statements and expressions over (internal state) variables, dataports, and values. As mentioned before, the languages are strongly typed. Networks of servo and transducer groups are defined in the PGAPP language, servogroups in the PGSG language, control blocks in the PGWB language and transducers in the Transducer language. By means of the *transducer language*, electrical and mechanical transducers can be defined. Complex transducers can be composed of multiple (connected) transducers, resulting in transducer groups.

3.1.2 Platform level

In the platform level, the execution platform of the lithoscanners is described. It consists of 3 domain-specific languages.

The *physical platform language* contains a description of (a subset of) the hardware and their physical connections as present in the lithoscanners. Typical concepts are (single and multi-core) HPPCs (High Performance Process Controllers), IOBoards (input-output boards), electrical and mechanical transducers, network switches and connectivity, and racks containing these components. A model in the physical platform language represents a set of possible platforms, since it does not contain the configuration data of the hardware. The configuration of a particular platform from this set is described by means of configuration data in the platform mapping language. Physical limitations of the hardware, such as for instance the maximum frequency at which a IOBoard can acquire sensor data are also contained in this language. The *logical platform language* abstracts from the physical (properties) of the hardware (location, IOboard types, HPPC processor types, network connections etc.). Concepts contained in this language are Worker (entity that can perform computations, abstracting from the real computing hardware), ProcessingUnit (abstracting from processor or core), IOWorker (abstracting from IOBoard type, and the location of Electrical and Mechanical TRansducers (ETRs, MTRs)), and Channel (using for data communication between nodes, abstracting from network type and topology). The *platform mapping language* contains the configuration data of the physical platform at hand, as well as the mapping from logical platform elements to physical platform elements by defining directed associations between them. Example of some mappings are: Worker(s) to HPPC, Board to IOBoard(s), and Channel to Network elements. An example of configuration data for the physical platform is the information at which rate the IOBoards are triggered to send (sensor) data. This configuration data depends on the application that has to be executed as well as the physical limitations of the hardware.

3.1.3 Mapping level

The mapping level describes the mapping of elements from the control application language to elements from the logical platform language.

The *Deployment language* contains associations to the control application and the logical platform language. Some examples of mappings are servoGroup to Worker, controlBlock to ProccessingUnit, and DataConnection to Channel. After scheduling, all atomic application elements (controlBlocks) should be mapped to atomic platform elements (processingUnits). Furthermore, the control blocks should be statically ordered respecting the data dependencies present in the servogroup model. This information is captured in the *Schedule language*. More information regarding the deployment and scheduling is given in Section 4.

3.2 Multi-disciplinary IDE

The design of control systems requires parallel development of mechatronic engineers, electronic engineers, and software engineers. Traditionally, this design process suffers from a complex environment that spans these different disciplines and dedicated tools that are loosely coupled with each other

based on conventions instead of formal specifications and relations between them. This leads to errors resulting from disobeying these conventions or even not knowing about them until the moment the subsystems are integrated. The developed multi-disciplinary IDE provides means to analyze and validate designs early in the design process, before integrating them and even before constructing them physically. In addition, it facilitates efficient construction of the associated software components that are executed on the lithoscanners. The IDE is based on OMG [2] standards, such as EMOF, and OCL for the definition of the domain specific languages, QVTo for model transformations, and M2T for model-to-text transformations. Regarding implementation technology, the Eclipse modeling framework (EMF) [1], QVTo [4], and Acceleo [5] are used, respectively. Depending on the DSL, textual and/or graphical concrete syntax(es) and associated editors are developed. For textual editors, EMFText and Xtext are used. Graphical editors are developed using Eclipse GMF [6]. Depending on the models, different persistence formats can be choosen: XML for easy post-processing with non-EMF tools, binary for fast processing of large models, and encrypted XMI for confidential models. The IDE supports the development process by providing means to specify the behavior of individual process control blocks up to the complete specification of a full CARM model stack. A textual editor is provided that is used to specify and generate new process control blocks. A graphical editor is provided that enables specification of complete ServoGroups. A similar editor can be used to group a selection of ServoGroups into a CARM application. In addition, editors exist that provide means to specify the execution platform at various levels of abstraction. This enables mapping the application on a conventional HPPC single core based computation platform but the application can also be mapped on a multicore or, in the future, even an FPGA-based platform. The IDE provides means to validate the application against timing and processor load constraints which avoids late detection of such problems during integration. By using this, and only this, framework, the following advantages are obtained: 1) consistency within the framework which increases usability; 2) optimized development effort for creation of the IDE; 3) optimized data exchange by using a single persistency mechanism. This results in a development environment that is efficient for the users as well for the developers.

4. ANALYSIS

A wafer scanner contains a large number of servo control systems. These systems are specified with the DSLs depicted in Figure 2. Servo control systems consist of control applications that are mapped onto an execution platform, as depicted in Figure 3. Control applications read values from sensors, compute mathematical functions specified in control blocks, and send results to actuators. In Figure 3, only a single application with a few functions, sensors and actuators is shown, but wafer scanners have hundreds of applications, each with hundreds of control blocks, sensors and actuators and this complexity grows with each new generation of machines. To obtain the required machine performance, applications have to run at high rates and have to satisfy stringent latency requirements between sensors and actuators. For this purpose a lot of computational power is needed, which is offered by multi-processor platforms consisting of

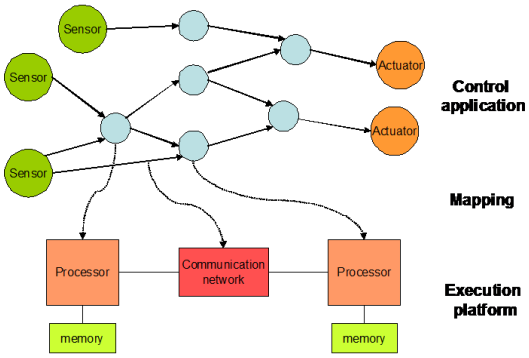

Figure 3: Servo control application mapped on execution platform.

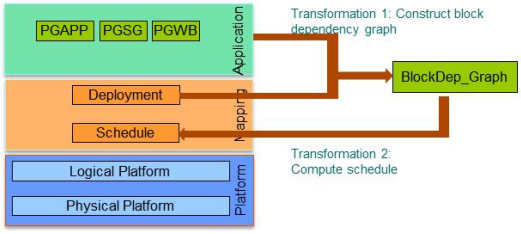

Figure 4: Model transformations for schedulings.

tens of general purpose (multi-core) processors communicating through low-latency packet-switched communication networks.

The mapping layer in our DSL framework specifies for each function on what processor it is deployed. In general, a single processor executes more than one control block and hence the order in which processors execute these functions, i.e. the schedule, is also specified in the mapping layer. Currently developers have to specify the deployment information by hand. Consequently a scheduling algorithm tries to compute a schedule for each processor in such a way that the latency requirements of the application are met. Whether the scheduler is able to discover a feasible schedule, heavily depends on the deployment choices of developer. To support the developer making appropriate choices, a detailed timing analysis can be performed to explore alternative solutions. In the next subsections scheduling and detailed timing analysis are explained in more detail.

4.1 Scheduling

The computation of a schedule is based on the information in the models conforming to our DSL framework. As a first step, essential scheduling information is extracted from the application and mapping DSLs. This is done by a model-to-model transformation that constructs a block dependency graph, see Figure 4. The dependency graph specifies control blocks and their dependencies. Latency requirements of the application are transformed into corresponding deadlines of blocks. In addition, a block is aware of the processor it is deployed on as well as its execution time.

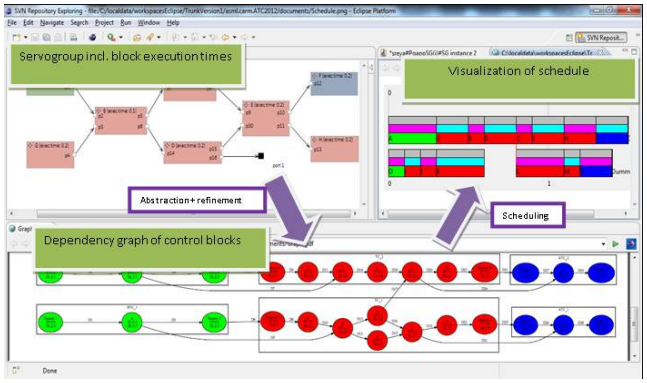

Figure 5: Scheduling process visualized in the IDE

The essential information in the dependency graph is used in a second step to compute schedules for each processor resource in the platform. These schedules complete the required information in the DSL mapping layer to carry out a detailed timing analysis. This process is supported by the IDE, see Figure 5.

4.2 Detailed timing analysis

Before a schedule can be computed, a lot of configuration information is detailed in the DSLs. This information includes the platform configuration (the number of processors, the topology of the switched interconnects, the settings of synchronization timers) and the deployment. To support developers to optimally configure the system, the timing impact of alternative configuration has to be analyzed. Where the scheduling DSL abstracts from detailed platform information (such as communication contention and jitter and timer settings), this information is fully taken into account when performing the detailed timing analysis. For this purpose, the DSLs are transformed into an executable model formalized in the POOSL language [7]. The resulting executable model follows the modularization as dictated by the Y-chart and formalizes the total system in terms of (stochastically) timed communicating parallel processes. During simulation, application-level timing requirements are automatically checked. Other properties of interest, such as processor loads, the loads on communication switches, timing averages and jitter are deduced as well. Next to a simulation-based analysis, an exhaustive dataflow analysis allows the detection of deadlocks.

5. SYNTHESIS

Currently, models are used for synthesis for two different purposes. Firstly, models are transformed into implementation code during development. In the CARM framework, specifications of control blocks are transformed into implementations expressed in the C programming language. Secondly, during startup, models are processed to generate the schedule for the control blocks (Section 4.1). Both cases are handled by applying text to model (T2M), model to model (M2M) and model to text (M2T) transformations. Based on previous experiences during the development of the Compositional Interchange Format [8], [9], transformations consist of many small transformations, each with their own concern, which can be composed into complete tool chains. In the next subsections, code generation and scheduling are explained in more detail.

5.1 Code generation

The block code generator is responsible for translating a block specification, made by mechatronics developers, into a set of C source files that can be compiled and added to the software release used during installation on lithoscanners. This code is executed in a hard-real-time environment, at rates of 10 kHz. An execution platform has been realized that provides a number of facilities like system startup, inter CPU communication, timing, diagnostics etc. The C code generated from a block specification is plugged into this platform. The architecture of the block generator adheres to the pipeline design pattern. Starting from the initial specification, information is added and modified until sufficient detail is achieved that generation of the implementation code has become a trivial task. Using an automated process of generating block implementations has a number of advantages: In hand crafted implementations, the way certain problems are implemented, vary as a result of knowledge and preference of the individual developers. The use of the generator results in uniform solutions to these problems. Various implementations (of the same problem) now can easily be compared with each other resulting in selection of the most efficient one. Sometimes, features are added to the execution platform. In case of cross cutting features, all block implementations need to be changed. Utilization of the block generator reduces the required effort for such refactorizations. Instead of having to change all block implementations manually, only the model to model transformations and the code generation templates need to be changed. After that, refactoring can be concluded by regenerating all implementations. The effort needed to create a new block implementation is also reduced by utilization of the code generator. For a very simple block that sums up the values on its inputs and puts it on its output, the following metrics have been obtained. Specification of this block using PGWB consists of 72 lines. The generated code boils down to 597 lines of C code. As a result, the time needed to create the implementation has been reduced from multiple days into a couple of minutes. Note that this excludes the time needed to create the block specification itself. Creation of the block specification is always needed, regardless the fact whether the implementation is hand crafted or produced by a generator.

5.2 Scheduling on the lithoscanners

There is a strong correlation between models used during analysis and models used for synthesis. This correlation origins from the fact that the system, described in the analysis phase, will be constructed if the results of the analysis are satisfactory. Note that analysis models can be more abstract, in case the required analysis accuracy is low, but can also be more detailed when for instance processor cache effects need to be investigated. A similar correlation is identified for the determination of the computation order of the blocks in the control network. During analysis, a scheduling algorithm is used to determine an optimal calculation order that satisfies the timing requirements. During startup, the scheduling algorithm is run again and the results are directly applied in the running system. In principle, no need should exist to execute this algorithm again as the results should not be different. However during integration, partially as-

sembled systems are initialized whose configuration has not been considered during analysis time. Also for these partial systems, a suitable calculation order is required, thus scheduling algorithm is run during initialization time.

6. RELATED WORK

We have strived for a development environment that: 1) allows the domain to be expressed concisely in the way ASML developers perceive it; 2) offers the necessary analysis and synthesis support that scale to the complexity of our systems; and 3) is extendible and open to interact with other modules in our system (e.g. such as supervisory control modules). In our vision, to simultaneously address these goals the syntactic (and static semantic) facilities to specify the domain should be separated from the underlying semantic models. The reason is that multiple of such semantic models are required, dependent on the specific analysis or synthesis task. E.g. the semantic model for scalable scheduling and proving absence of deadlock in our application is Synchronous Data Flow (SDF) [10]. For detailed stochastic simulation, on the other hand, the basis is the Segala model of probabilistic timed transition systems [11]. Our environment is designed in such a way that it can be coupled to the necessary semantic models and can be extended with additional ones if needed. At the same time it offers the proper syntactic support that allows developers to express concisely their domain, and nothing but their domain. While studying literature, we found many integrated modeling environments, but none of them sufficiently fitted our purpose. For instance, the standard environment Simulink [12] to model and analyze control systems, has no support to express our execution platforms. The TrueTime suite [13] does have such facilities, but requires one to encode platform information in low-level primitives. In addition TrueTime has no support to schedule applications on the platform. Scheduling is supported in an encompassing tool chain [14] but the underlying semantic model of this chain is TTA, while our platforms are based on a data-flow paradigm.

7. CONCLUSIONS

In this paper, we described the CARM framework consisting of DSLs and a multi-disciplinary development environment to design, analyse and construct control systems for ASML lithoscanners. We have shown that DSLs, T2M, M2M, and M2T transformations allow to define an integrated design environment for analysis and construction which is applied in a hard real-time industrial environment.

During the development of this framework, several improvements to the current state-of-pratice of model-based engineering have been identified. For instance, to enable a more modular design of DSLs, such that DSLs can be defined in terms of several other (smaller) DSLs, explicit interface definitions, supported by tools, on metamodels are required to specify which part of a metamodel can be used by other metamodels. Currently, one has to explicitly model this modularity in the DSLs itself. The development of domain specific languages is a difficult process. It deals with the formalization of implicit conventions present within a particular domain, choosing the correct level of abstraction which is a trade-off between too domain specific, and too general, in combination with trying to foresee and incorporate future extensions in the domain. The resulting domain languages differ from software class diagrams. The domain expert should play an important role in the development of the domain model in order to ensure that the domain model 'explains itself' to its users later on. If the DSL at hand is strongly typed, there is no out-of-the-box tooling support to define the type computing. Structural semantics can be well captured within the class diagrams in combination with OCL constraints. However, there is no out-of-the-box support for the definition of (formal) behavioral semantics, in terms of, for instance SOS rules [15].

8. REFERENCES

[1] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: Eclipse Modeling Framework. Addison-Wesley (2009)

[2] Object Management Group: Object Constraint Language, version 2.0; MOF Model to Text Transformation, Version 1.0; Query/View/Transformation (QVT), version 1.0. http://www.omg.org/ (2006-2012)

[3] Kienhuis, B.: Design Space Exploration of Stream-based Dataflow Architectures: Methods and Tools. PhD thesis, Delft University of Technology (1999)

[4] Eclipse: QVT Operational. http://www.eclipse.org/m2m/ (2012)

[5] Obeo: Acceleo, version 3.0. http://www.eclipse.org/acceleo/ (2012)

[6] Gronback, R.C.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley (2009)

[7] Putten, P.H.A.v.d., Voeten, J.P.M.: Specification of reactive hardware/software systems - The method software/hardware engineering. PhD thesis, Eindhoven University of Technology (1997)

[8] Systems Engineering Group TU/e: CIF toolset. http://se.wtb.tue.nl/sewiki/cif (2012)

[9] Hendriks, D., Schiffelers, R., Hüfner, M., Sonntag, C.: A Transformation Framework for the Compositional Interchange Format for Hybrid Systems. In: Proc. 18th IFAC World Congress. (2011)

[10] Lee, E.A., Messerschmitt, D.G.: Synchronous Data Flow. In: Proc. of the IEEE, vol. 75. (1987) No.9 1235–1245

[11] Segala, R.: Modeling and verification of randomized distributed real-time systems. PhD thesis, Massachusetts Institute of Technology, Massachusetts (1995)

[12] The MathWorks, Inc: Using Simulink, version 6. http://www.mathworks.com (2005)

[13] Eker, J., Cervin, A.: A Matlab toolbox for real-time and control systems co-design. In: Proc. 6th Int. Conference on Real-Time Computing Systems and Applications. (1999) 320–327

[14] Hemingway, G., Porter, J., Kottenstette, N., vanBuskirk, C., Karsai, G., Sztipanovits, J.: Automated synthesis of time-triggered architecture-based truetime models for platform effects simulation and analysis. In: Rapid System Prototyping, IEEE (2010) 1–7

[15] Plotkin, G.D.: A structural approach to operational semantics. J. of Logic and Alg. Progr. **60-61** (2004)