# Discrete-Continuous Semantic Adaptations for Simulating SysML Models in VHDL-AMS

Daniel Chaves Café[1,2], Cécile Hardebolle[1], Christophe Jacquet[1], Filipe Vinci dos Santos[2], and Frédéric Boulanger[1]

[1] Supélec E3S – Computer Science Departement,
[2] Thales Chair on Advanced Analog System Design,
{daniel.cafe, cecile.hardebolle, christophe.jacquet, filipe.vinci, frederic.boulanger}@supelec.fr

**Abstract.** Our research focuses on the simulation of heterogeneous systems modeled in SysML, in particular, systems that mix different engineering domains such as mechanics, analog and digital circuits. Because of their nature, expressing multi-paradigm behavior in heterogeneous systems is a cumbersome endeavor. SysML does not provide a standard method for defining the operational semantics of individual blocks nor any intrinsic adaptation mechanism when coupling blocks of different domains. We present in this paper a way to address these obstacles. We give well-defined operational semantics to SysML blocks by using profile extensions, together with a language for the description of adaptors. We apply our approach to a test case, using a toolset for SysML to VHDL-AMS transformation, capable of automated generation of VHDL-AMS code for system verification by simulation.

## 1 Introduction

In the Electronic Design Automation (EDA) industry, the need for modeling and verification of mixed-signal systems gave rise to several system design languages supporting Analog and Mixed Signal (AMS) extensions. Some examples are VHDL-AMS [6] and SystemC-AMS [8]. These extensions support the use of different models of computation concurrently in a single design thus enabling the modeling of heterogeneous systems. As complexity increases, these textual languages are no longer suitable for proper documentation and communication among different teams. For these use cases, graphical languages are preferable, and they play well with Model Driven Engineering workflows.

SysML, the Systems Modeling Language, is an industry standard for systems specification. It provides a large set of diagrams which can be used to specify system's requirements, model their behavior or even detail the interconnections of structural blocks. Despite its flexibility, SysML does not provide clear semantics. On the one hand, this can be helpful for engineers wishing to describe systems in an early development phase, especially when some implementation details are not yet entirely defined. In this case, SysML is a helpful communication tool.

On the other hand, the lack of clear semantics can be cumbersome if one wants to run simulations from the SysML diagrams.

For the purpose of solving the lack of semantics of SysML diagrams, we have developed a technique to generate executable code from SysML models which is based on two foundations : (a) Explicitly state the semantics of modeling elements, and (b) Define the semantic adaptations between heterogeneous models. The focus of this work is the creation of an adaptor instantiation language for semantic adaptation for specifying interfaces precisely and without ambiguity.

The organization of this article is as follows : The state of the art is presented in section 2 detailing existing heterogeneous modeling techniques. A case study is then introduced in section 3 to illustrate the problem. The actual implementation of our solution is detailed in section 4. And finally we discuss the approach and the results in sections 5 and 6.

## 2 Related Work

One of the precursors of heterogeneous modeling is the well-known Ptolemy II [7] framework. Here, heterogeneity is handled by hierarchy. Components are nested in black boxes called actors for which the semantics of execution and communication are described by an entity called *Director*. It defines the model of computation (MoC) of the actor. In Ptolemy, computation and communication are defined for a large set of MoCs. These include Process Networks, Dataflow, Discrete Event, Finite State Machines, Continuous Time and others. Unfortunately, Ptolemy does not provide explicit ways to define adaptations between models that use different MoCs. For example, interactions between discrete event (DE) and synchronous dataflow (SDF) models can result in redundant events in the DE domain if a given value does not change. In the same way, an SDF model might not be regularly activated as discussed in [2].

ModHel'X [10] was developed to explore semantic adaptations in heterogeneous models. ModHel'X improves upon the execution algorithm of Ptolemy by introducing an adaptation phase. This yields an effective way to define the semantics of the interactions between different models of computation. The current implementation of ModHel'X is based on a non-standard metamodel which makes it hard to integrate with existing toolchains.

We propose to introduce ModHel'X's good practices of stating the semantics of different components and explicit modeling of the semantic adaptation between heterogeneous components, into an industry standard modeling language: SysML. In our approach SysML acts as a pivot language from which we generate executable code for widely deployed languages, such as SystemC-AMS and VHDL-AMS. We use a custom profile to extend the semantics of SysML blocks for continuous-time and discrete-event blocks. These two domains are generalized into two stereotypes $\ll$ *analog* $\gg$ and $\ll$ *digital* $\gg$. A third stereotype is dedicated to the description of $\ll$ *adaptor* $\gg$ blocks. Those provide explicit behavior on how to adapt data, time and/or control. To do so, a mini-DSL was designed to allow the instantiation of off-the-shelf types of adapters. Depending

on the target language these could either be present in standard libraries or custom designed.

Substantial work has been carried out to apply SysML/UML to the design of electronic (analog and digital) systems. Many researchers focused on the generation of VHDL-AMS code from SysML diagrams. D. Guihal [9] and J. Verriers [14] extended the VHDL metamodel proposed in [1] and [13] to use AMS constructions in their code generators. J.-M. Gautier et al. [3] used model transformations to generate VHDL-AMS code from SysML Block Definition Diagrams and Internal Block Diagrams. They have used block constraints to define physical equations in VHDL-AMS modules.

Although these previous works have shown methods to generate VHDL-AMS code from SysML diagrams, they have not dealt with the semantic inconsistencies that heterogeneity introduces. Our previous work [4] presents a technique to deal with this problem. It targets SystemC-AMS simulation language. The present work is a follow-up that introduces a new adaptor instantiation language and MoC definition mechanisms that are better suited for model driven engineering. This is also an opportunity to show that previously developed techniques apply to other target languages as well, namely VHDL-AMS.

## 3 A case study of a MEMS Accelerometer

Micro Electro Mechanical Systems (MEMS) motion-sensing devices are a good example of heterogeneous systems that mix mechanical, analog and digital components in the same system. They can be used to measure a variety of physical quantities such as acceleration, pressure, force, or chemical concentrations. To make such measurements, MEMS sensors can take advantage of several transduction mechanisms, for example, piezoresistive or capacitive sensing. Here we build a simple model of a capacitive sensing accelerometer to illustrate our proposal.

### 3.1 Description of the system

Our case study is a capacitive sensing accelerometer composed of two electrodes and an intermediary membrane free to move only in the vertical axis as illustrated in figure 1. This structure forms two capacitors between the middle membrane and both the top and bottom walls. The vertical movement of the membrane implies the variation of both capacitances since $C \propto 1/(g_0 \pm x)$, where $g_0$ is the gap distance at rest and $x$ is the displacement of the membrane from rest. One can either connect the membrane to ground hence fixing the middle voltage $V_{middle}$ to zero or one can leave it disconnected thus fixing the current to zero. In the first case, the change in stored charge caused by the displacement of the membrane leads to a current flow. In the second case, since the middle electrode is disconnected, there is no current flow, and by charge conservation the voltage across the membrane must change with the displacement.

Using the second method, we obtain a linear relation between the membrane's voltage and its displacement provided that we apply a symmetric voltage on both top and bottom electrodes (i.e. $V_{top} = -V_{bottom} = V_0$) as explained in [5]:

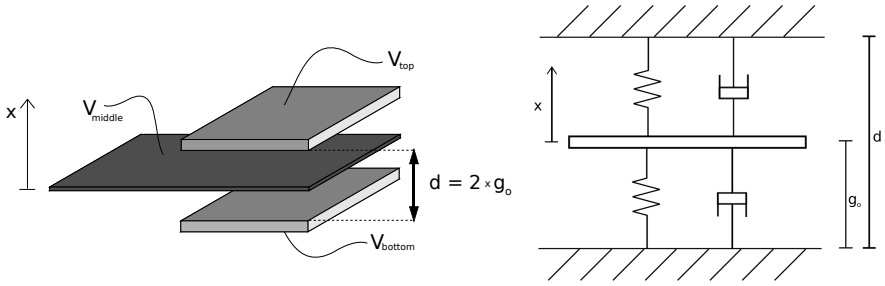


**Fig. 1.** Electrical vs Mechanical Model

$$V_{middle} = V_0 \frac{x}{g_0} \tag{1}$$

The membrane's displacement depends on several forces. In our example we consider only inertial, spring and friction ones. These are assumed to act exclusively at the center of the membrane. The spring force is proportional to displacement and the damping (friction) to velocity. We are here interested in studying the behavior of this system when an external force is applied to the membrane, typically gravity, but it could be any external force. Applying Newton's law, we end up with:

$$F_{external} = -kx - c\dot{x} + m\ddot{x} \tag{2}$$

Several precautions must be taken to accurately extract $V_m$. Our model uses the most simple read-out circuit, an operational amplifier configured as a buffer. The output of the buffer is fed to a voltage comparator, giving a one-bit output that undergoes further processing in the digital domain. The details of the rest of the system fall outside of the scope of the discussion.

The model of the operational amplifier consists of a single piecewise equation considering the gain and saturation. The latter assures that the output does not exceeds the supply voltages of $V_{DD} = +15V$ and $V_{SS} = -15V$. The piecewise equation is as follows:

$$V_{out}(V_{in}) = \begin{cases} V_{SS} & : V_{in} < V_{SS}/gain \\ V_{DD} & : V_{in} > V_{DD}/gain \\ V_{in} \times gain & : elsewhere \end{cases} \tag{3}$$

### 3.2 SysML Model

In SysML, we have divided the system into five major blocks as illustrated in figure 2: the **accelerometer** models the electromechanical dynamics, the **opamp** models the operational amplifier, the **sampler** adapts analog data to the digital world by periodic sampling, the **comparator** checks for a threshold

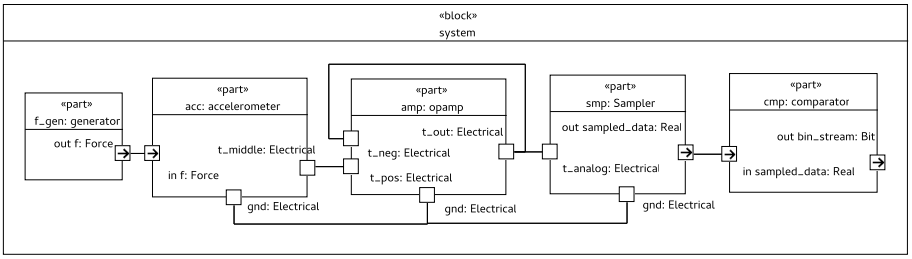


**Fig. 2.** SysML Model [IBD]

crossing generating a bit stream from the output of the sampler, and finally a **source** sine wave force generator stimulates the model.

While analog blocks use differential equations defined in continuous time, digital circuitry is best modeled in the discrete domain. These two formalisms handle different types of data and react differently to inputs. If we wish to model and simulate a system with both of them together, we must specify not only the operational semantics (i.e. the Model of Computation) of a particular block but also the semantic adaptation between both domains.

To solve the semantic ambiguity issue, we use custom stereotypes defined in a separate profile. Stereotypes are element modifiers that allow us to give precise meaning to base elements of SysML. In our case, we have chosen to apply specific stereotypes to SysML blocks in order to specify the use of a given model of computation. Since we are dealing with continuous-digital integration, we have added the notion of $\ll analog \gg$ and $\ll digital \gg$ blocks to SysML as seen in figure 3. We have also added one stereotype $\ll adaptor \gg$ to specify blocks that are in the frontier of two different MoCs.

For an analog block, we use SysML/UML constraints to describe the physical relations shown previously. The equations defined in SysML constraints are considered to be continuous. The interconnections in an analog block impose other equations that can be inferred from the topology of the system. In the case of electrical circuits, these are the Kirchoff laws. Digital blocks on the other hand are connected by signals that transmit events. Even though in the real world, digital circuits have analog behavior, this formalism abstracts these electrical phenomena making digital circuits design simpler.

One particular case that is worth noting here is the definition of piecewise equations. We have used a particular syntax in SysML constraints to describe these kind of relations. For instance, equation 3 describes the simplified behavior of an operational amplifier and is represented by one SysML constraint preceded by the keywords *PIECEWISE FUNCTION* in our mini-DSL (see figure 4-left). In VHDL-AMS, this is translated to *USE* conditions as we see in figure 4-right.

We have also defined the quantities that exists between terminals, such as voltage or current using SysML properties (see figure 3). These are translated to VHDL-AMS quantities directly.

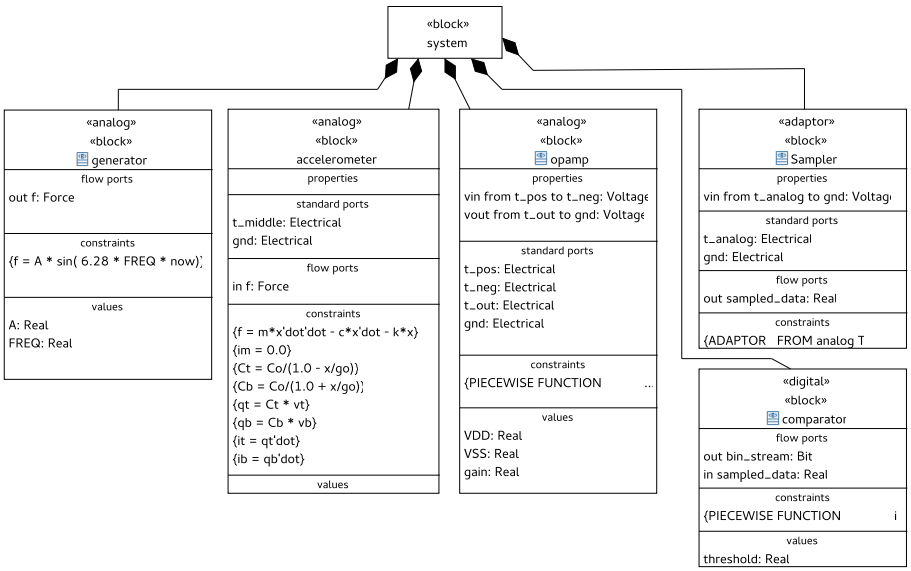


**Fig. 3.** SysML Model [BDD]

```
1 PIECEWISE FUNCTION
2   V_in < VSS/gain:
3      V_out = VSS,
4   V_in > VDD/gain:
5      V_out = VDD,
6   elsewhere:
7      V_out = gain * V_in
```

```
1 IF V_in'ABOVE(VDD/gain) USE
2   V_out == VDD;
3 ELSIF NOT V_in'ABOVE(VSS/gain) USE
4   V_out == VSS;
5 ELSE
6   V_out == gain * V_in;
7 END USE;
```

**Fig. 4.** Definition of piecewise equations (SysML vs VHDL-AMS)

### 3.3 Adaptation Mechanisms

The ≪ *adaptor* ≫ stereotype defines a block whose main purpose is to adapt data, time and/or control from one domain to another. This special block defines the adaptation semantics of heterogeneous interfaces. If they are well defined, then generating executable code from that model should produce the same result regardless of the target language (VHDL-AMS in this case, but it could be any other AMS-capable language, such as SystemC-AMS).

In our example, the block **sampler** is an adaptor from analog to digital domain. It samples data periodically. We do not specify the behavior of the adaptor using our language, rather we instantiate and parameterize a pre-defined adaptor. This is achieved using a SysML constraint starting with the *ADAPTOR* keyword. Figure 5 shows our mini-DSL being used to instantiate the sampler.

Analog data is generated at a dynamic timestep in VHDL-AMS simulators. The adaptor specification guarantees that output data will be sampled at a fixed *timestep* of 2 $\mu$s. This case of adaptor is interesting because we are not only

```
1 ADAPTOR
2   FROM analog TO digital
3   IS sampler
4   PARAMS
5     input     : vin,
6     output    : sampled_data,
7     timestep  : 2us
```

**Fig. 5.** Adaptor specification in SysML constraints

adapting the time base but also the data format. In the analog domain, ports are considered to be terminals connected to nodes of a circuit. Kirchhoff equations can be then deduced from the topology of the circuit. The adaptor must extract the voltage between the input terminal and a reference and propagate it to a discrete event domain as data tokens. The parameters *input* and *output* of figure 5 indicate the analog voltage to read and the binary stream to write.

Certain adaptors that we make available to system designers have off-the-shelf counterparts in the target language; others do not. In the former case, our transformation chain chooses adaptors from a standard library; in the latter case it defines new adaptors in the target language.

In this case study, the sampler isn't present in the VHDL-AMS library so we generate the module responsible for this specific adaptation. The generated code can be separated into two different VHDL processes. One for setting the time step and a second one, triggered by the first, to model the adaptor semantics. In this case, the semantics are fairly simple. It consists on copying the analog voltage from input terminal and its reference to the discrete event output at a scheduled moment in time, i.e. every $2\,\mu$s.

The output of the sampler is connected to a comparator which will generate a bit stream from its input. When the input analog voltage crosses a value given by the *threshold* parameter, the digital output switches to a logical value of '1', or '0' otherwise.

## 4 Model Transformation

Our approach, illustrated in figure 6 consists in two separated phases. Starting from a SysML model, we first perform a model-to-model (M2M) transformation $T_1$ in order to obtain a VHDL-AMS model. We then generate VHDL-AMS code through a model-to-text (M2T) transformation $T_2$.

The model-to-model step $T_1$ translates every SysML element into its equivalent VHDL-AMS element. This step provides the model with semantics on how to interpret SysML elements. For instance, UML ports are converted to terminals while SysML flow ports are translated to quantity ports. In the same way, a SysML constraint will be transformed into an equation with its variables translated into VHDL-AMS quantities.

For this first step, we have used the Atlas Transformation Language (ATL) [11]. ATL is a language for defining model transformations by a set of rules. Being a

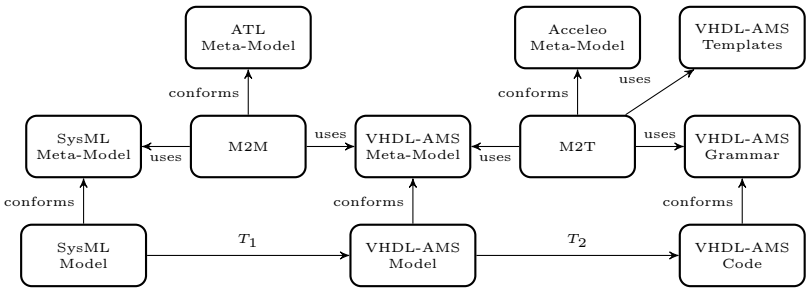


**Fig. 6.** Our approach

model itself, the transformation has its own meta-model. ATL is based on pattern recognition of input elements and conditions which trigger the creation of output elements in the resulting model.

The VHDL-AMS metamodel is an improvement from previous works [1, 9, 14]. It includes the notion of parameterizable adaptors and some slight modifications to the general structure of how libraries are used inside a model. This has proven to be very practical in our implementation but it introduces elements to the metamodel that are not totally part of VHDL-AMS. Instead, a two-step approach separating pure syntax from semantics could also be considered.

Finally, transformation $T_2$ is responsible for generating the actual code that will be used for running the simulation. For this we use the ACCELEO [12] model-to-text engine from Obeo. In ACCELEO we write templates that specify the code to generate for the various model elements. The adaptors, for instance were instantiated depending on their type. This is specified in our mini-DSL by the keyword *IS* and is parameterized by the list of parameters listed after *PARAMS*. The generated code is a template with two VHDL processes (as explained in section 5).

## 5 Simulation Results

Applying both transformations ($T_1$ and $T_2$) to the SysML model presented in figure 3, we obtain several VHDL-AMS files (one per block) which we use to run simulations. In figure 7 we show the output of the Hamster VHDL-AMS simulation tool for a sinusoidal input force.

Note that, despite the non-linear variation of both top and bottom capacitances, the output voltage is linear and follows the input stimulus, which is conform to equation 1. The left side of figure 7 allows us to conclude that the threshold detection mechanism described by the block **comparator** works correctly as the binary stream output follows the sign of the **opamp**'s output.

A closer look allows us to confirm that digital data is sampled at a fixed timestep even tough the analog data is not. The signal *clk* generates events every $2\,\mu$s, both on the rising and falling edge. The detail of the right part of figure 7 shows that the output voltage was already negative several simulation

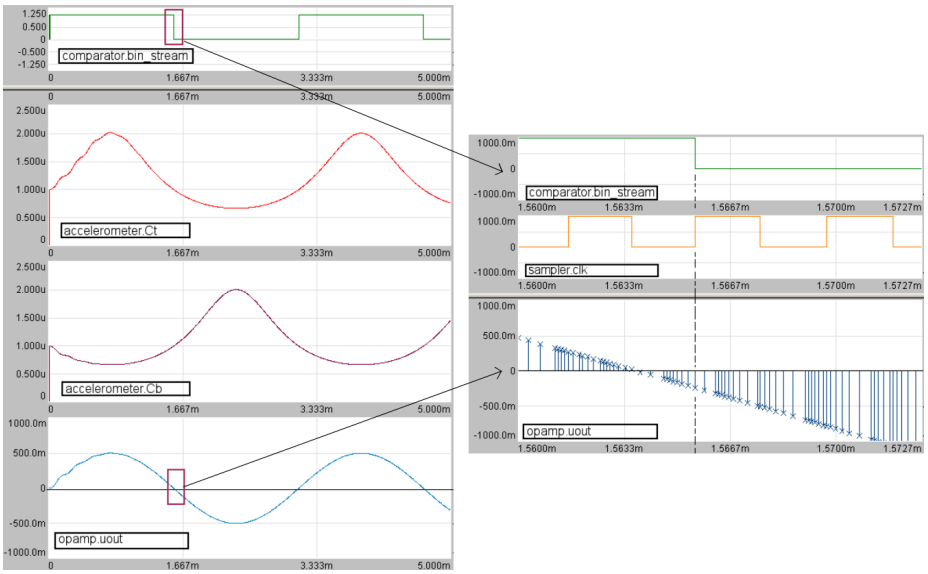


**Fig. 7.** Simulation Results

cycles before the threshold detection. This translates into a delay between the effective crossing of the threshold (bottom left of figure 7) and its detection. This is the expected behavior since the specification of the adaptor constraint of figure 5 specifies a 2 $\mu$s sampling period thus there can be a delay of up to 2 $\mu$s.

## 6 Conclusions

In this paper we introduce an approach for simulating continuous-digital interaction in SysML models. We validate the behavior through simulation and we generate executable VHDL-AMS code using automatic model transformations. We address the ambiguity of SysML diagrams by assigning them concrete semantics (MoCs) using a simple profile. In order to solve the semantic adaptation problem, we explicitly design adaptation mechanisms using a dedicated language based on SysML constraints. These are translated into specific VHDL-AMS constructs that enforce the specified behavior.

The case study presents a typical case where integration issues occur. We have specified not only the model of computation of individual SysML blocks using stereotypes but also the semantic adaptations between continuous and discrete domains using the notion of adaptors.

In future work, we wish to separate the semantic parts of our transformation from purely syntactical ones. This would allow us to focus on a more generic approach so as to deal with heterogeneous interactions independently from the language used to run simulations. In order to do so, we have considered using a generic intermediary metamodel to facilitate transformations to other languages.

## References


1. V. Albert. Traduction d'un modèle de système hybride basé sur réseau de Petri en VHDL-AMS. *Master de conception en architecture de machines et systèmes informatiques, Université Paul Sabatier, LAAS-CNRS*, 2005.
2. Frédéric Boulanger and Cécile Hardebolle. Execution of models with heterogeneous semantics. In *Tutorial on critical systems simulation at CSDM'12*, December 2012.
3. Fabrice Bouquet, Jean-Marie Gauthier, Ahmed Hammad, and Fabien Peureux. Transformation of SysML structure diagrams to VHDL-AMS. In *Design, Control and Software Implementation for Distributed MEMS (dMEMS), 2012 Second Workshop on*, pages 74–81. IEEE, 2012.
4. Daniel Chaves Cafe, Filipe Vinci dos Santos, Cecile Hardebolle, Christophe Jacquet, and Frederic Boulanger. Multi-paradigm semantics for simulating SysML models using SystemC-AMS. In *Specification & Design Languages (FDL), 2013 Forum on*, pages 1–8. IEEE, 2013.
5. Franck Chollet and Haobing Liu. A (not so) short introduction to Micro Electro Mechanical Systems. http://memscyclopedia.org/introMEMS.html, pages 149-152. Nov 2013.
6. Ernst Christen and Kenneth Bakalar. VHDL-AMS - a hardware description language for analog and mixed-signal applications. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 46(10):1263–1272, 1999.
7. Johan Eker, Joern W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, and Yuhong Xiong. Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.
8. Christoph Grimm, Martin Barnasconi, Alain Vachoux, and Karsten Einwich. An introduction to modeling embedded analog/mixed-signal systems using SystemC AMS extensions. In *DAC2008 International Conference*, 2008.
9. Guihal, D and Andrieux, L and Esteve, D and Cazarre, A. VHDL-AMS model creation. In *Mixed Design of Integrated Circuits and System, 2006. MIXDES 2006. Proceedings of the International Conference*, pages 549–554. IEEE, 2006.
10. Cécile Hardebolle and Frédéric Boulanger. ModHel'X: A component-oriented approach to multi-formalism modeling. In *Models in Software Engineering*, pages 247–258. Springer, 2008.
11. Frédéric Jouault and Ivan Kurtev. Transforming models with ATL. *Satellite Events at the MoDELS 2005 Conference*, pages 128–138, 2006.
12. J. Musset, E. Juliot, S. Lacrampe, W. Piers, C. Brun, L. Goubet, Y. Lussaud, and F. Allilaire. Acceleo user guide, 2006.
13. Guillaume Savaton, Jérôme Delatour, and Karl Courtel. Roll your own hardware description language. In *OOPSLA & GPCE Workshop Best Practices for Model Driven Software Development*, 2004.
14. Jean Verries. *Approche pour la conception de systèmes aéronautiques innovants en vue d'optimiser l'architecture. Application au système portes passager*. PhD thesis, Université Paul Sabatier-Toulouse III, 2010.