# From system-level models to heterogeneous embedded systems

Jean-Christophe Le Lann, Joël Champeau, Papa Issa Diallo, Pierre-Laurent Lagalaye

# From system-level models to heterogeneous embedded systems

Jean-Christophe Le Lann
Joel Champeau
Papa-Issa Diallo
Lab-STICC UMR CNRS 6285
ENSTA-Bretagne
29806 BREST cedex 9
{jean-christophe.le_lann,
philippe.dhaussy}
@ensta-bretagne.fr

Pierre-Laurent Lagalaye
Modaë Technologies
35000 Rennes
pierre-laurent.lagalaye
@modae-tech.com

## ABSTRACT

The need for higher level models during system design has resulted in many different Electronic System Level (ESL) formalisms that seldom succeeded in the past in the quest for efficient top-down design methodologies. In this paper, we propose a new working toolchain starting from UML specifications coupled to an industrial-level system-level synthesis (SLS) tool. In order to isolate a maximum of concepts, we resorted to state-of-the-art model-driven software engineering and strong models-of-computation (MoC). This tooling allows for rapid prototyping of mixed hardware-software systems in line with the high degree of heterogeneity encountered in today embedded platforms.

## Keywords

Models of computation, MDD, Model-based software engineering

## 1. INTRODUCTION

Due to the sustained and exponential growth in complexity in embedded systems, the need for abstraction is overwhelming. To be concrete, such a standard as H.264 in video compression is considered eight times as complex as previous MPEG2 standard to decode, but still no efficient methodologies have succeeded to handle the entire design flow of associated integrated circuits. In 2012, new video compression algorithms are inevitably showing new promises in term of compression ratio, but methodologies have not shown significant advances in the meanwhile. As a consequence, designers seem to reach their limits in term of productivity, without any convincing help from tool chains. In the automotive industry, the rise of embedded systems have "the potential to raise the bar for vehicle safety, as well as luxury and convenience, but only if they work" [1]. In this field, the safety critical aspects add a supplemental dramatical dimension in the problem. Some practitioners have proposed to tackle this problem, using a software model-based approach. In this view, UML remains a candidate of choice as a system level visual language to support various tasks in the V-model : analysis, specification, design, verification and validation. Moreover, UML can be extended in various ways for customization: tag definitions, constraints, and stereotypes, which are applied to elements of the model. This yields the notion of profiles, that help tune the language for dedicated purposes.

This trend has grown rapidly, to the extend that not only software but even hardware code generation is now studied. In this work, we propose an new toolchain that allows refining UML specification down to hardware-software implementations. We used one of the more adopted UML 2.0 modeler, namely IBM Rhapsody, as front-end and Modaë system-level synthesis (SLS) Compiler as backend. In order to transform UML descriptions into suitable Modaë formalism, we resorted to software model-driven technology, that guided our work in the sense of elicitation of fundamental communication concepts between UML components.

The rest of the paper is organized as follows. Next section recalls the state of the art concerning the use of UML for embedded system-level modeling, dedicated profiles and frameworks, ending by a short survey on model-based high-level synthesis. Section 3 presents our design flow, while section 4 depicts a concrete experiment using this design flow. We conclude with a discussion on our future work.

## 2. STATE OF THE ART

### 2.1 ESL scopes

Electronic System Level (ESL) is technically rooted in EDA (Electronic Design Automation), with the ambition to raise the level of abstraction above RTL descriptions (register transfer level) that characterize VHDL and Verilog. During the last decade, many new formalisms were proposed to address the complexity of embedded systems. Their names clearly reflect their orientation towards system-level descriptions : SystemC, SpecC, HandelC, SystemVerilog,.... ESL can be segmented into three distincts types of tools :

- High-level synthesis (HLS), that automatically translates sequential code into RTL, typical of top-down strategies. SLS is an extension of HLS in that it can operate on a network of such sequential algorithms, and target both hardware and software.
- IP-based design, that focuses on reuse of intellectual property blocks (IP), in a bottom-up strategy.

- Platform-based design, that is based on the configuration and extension of generic hardware plateforms.

## 2.2 UML profiles for ESL modeling

However, the classical ESL languages essentially still exhibits concepts specific to the electronic industry, and remains only attractive for the specialists of this industry. The notion of system should encompass many other aspects (software, mechanical parts, requirement engineering, etc). In this regard, UML stand as more natural candidate [21] to fulfill the need for "universal" formalism, understoodable by the different actors of the industry and help improve cross-disciplinary communication. It offers suitable means to describe concepts specific to dedicated domains : the notion of profiles are especially invented as a generic extension mechanism for customizing UML models, while maintaining its core notions. However, as explained by Schattkowsky [21], its cannot be instantly applied to HW/SW codesign.

As soon as 2002, CATS, Rational, and Fujitsu proposed a dedicated profile for System-On-Chip (SoC) [17]. OMG released it in 2006. It formalized many concepts of SystemC and especially the transaction-level modeling that render SystemC attractive for simulation speed [20]. Using such a typical profile, the aforementioned pitfalls of dedicated system-level languages are somehow hidden by UML. More recently (2009), MARTE was proposed by the OMG as the standard for modelling real-time and embedded applications with UML2. It covers much more aspects of those applications [12],[24],[11], spanning from modeling of applications to plateforms architectures (processors, DMA,...) known as "Generic Resource Modeling", offering various descriptions of platform configurations [19]. Non-Functional Properties can also be described, as well as rich timing constraints, inheriting much work from synchronous languages [15]. In the same vein, Sysml offers modeling concepts [6], many of them borrowed from UML 2.0 (seven out of thirteen UML 2 diagrams are present in SysML). SysML is however more adequate for system engineering, with requirements capture capabilities, parametric diagrams and a more intuitive block-diagram definition from classes.

## 2.3 Model-based toolchains

Mopcom is a french cooperative research project, lead by Thales inc., that aims to leverage model-driven architecture mapping concepts to SOPC development flows. Mopcom methodology has been applied to different co-design experiment like [14] and [26]. Three levels of abstraction are proned here : abstract modeling level, execution modeling level and detailed modeling level (AML, EML and DML, respectively). AML level allows a designer to express its purely functional application as concurrent entities. EML level represents this application mapped onto non-detailed architectural elements (communication topology and abstract processors). Finally DML is supposed to represent the same platform at the RTL level. Schattkowsky describes a very similar approach. Mopcom is similar to our present approach in that is relies on almost the same tooling, with the exception of HLS that has not been used in Mopcom. Instead direct RTL VHDL generation was ambitioned by their flow, but remains unpractical.

The SATURN [18] design flow is also an important step in the possible adoption of model-based toolchains : Saturn introduces UML profiles for the co-modeling of SystemC and C, with code generation support relying on Artisan Studio. The later provides complete support for OMG UML and SysML in a single, integrated toolset. The synergetic association of models, transformations and ESL objectives are closed to our paper.

Coyle [4] also present MODCO, that handles XML statecharts and generates VHDL. The approach is similar in that Rhapsody is also used, with a MDD-like approach. However, the author do not resort to any meta-modeling strategies. Wood et al [27] can also be cited here, with the same limits.

## 2.4 Connections to high-level synthesis

Very few papers have made a convincing bridge between system-level models and HLS/SLS technologies. The works we already cited have instead tried to proceed using a single step : from UML directly to RTL VHDL. The focus of these papers if clearly on the translation of only statecharts to RTL, with a very limited algorithmic scope. In our view this seem contradictory with the ambition of high-level models. Model-based HLS seems a much more natural approach. Our work can be best compared to commercial Axilica FalconML tool : for instance, [25] describe the synthesis of H.264/MPEG-4 AVC video decoder from UML.

HLS is an equivalent of traditional compiler technology, targeted to hardware code generation (Register-transfer level, expressed in hardware description language such as VHDL) instead of assembly. The input code (such as ANSI-C or derivatives like SystemC) is first transformed into a control/data flow graph (CDFG) by a front-end pass. This CDFG acts as internal representation (IR) on which every pass are applied. This early stage can benefits from classical compiler-like optimizations (dead-code elimination, constant propagation, common sub-expression elimination) and –more importantly for hardware generation purposes– such as loop unrolling, that exhibits the potential parallelism of the application. Then specific transformations are applied on thir CDFG IR : scheduling, allocation, binding that orient the process towards specific computing and routing ressources utilization. Allocation and binding assign operations onto functional units, and variables and data structures onto registers, wires, or memory locations. The HLS process may depend on technological exact parameters (e.g a silicon foundry library) or not (generic ressources modeling). Many commercial HLS tools are now widespread [3] : CatapultC by Mentor Graphics, Cynthesizer by Forte Design Systems, Impulse CoDeveloper by Impulse Accelerated Technologies, Synfony HLS by Synopsys, the C-to-silicon by Cadence, the C to Verilog Compiler by C-to-Verilog, the AutoPilot by AutoESL, the PICO by Synfora, and the CyberWorkBench by NEC System Technologies Ltd.

# 3. DESIGN FLOW

## 3.1 Overview

The overall toolchain presented here is depicted on figure 1 : the link between model-based capture and HLS in ensured by MDD techniques.

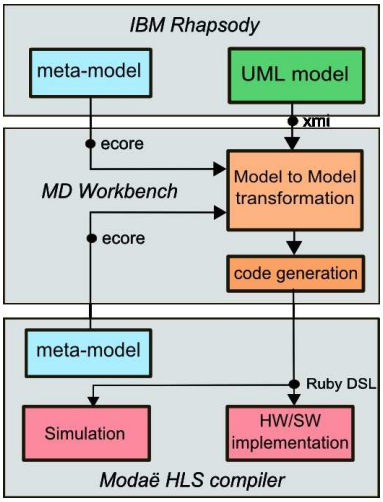


**Figure 1: Overview of our MDD toolchain : UML-Rhapsody ensures capture of application, while MD-Workbench is used for transformations of metamodels to Modaë HW/SW HLS. Xmi is transformed into Ruby DSL thanks to ecore metamodels knowledge.**

## 3.2 Rhapsody

Rational Rhapsody is a modeling environment based on UML. Since 1996, Rhapsody acts as visual development environment for systems engineers and software developers. It is mostly used to creating real-time or embedded systems and software. It is used now to abstract complexity visually using industry standard languages (UML, SysML, AUTOSAR, DoDAF, MODAF, UPDM). Its technological roots can be found in David Harel [8] [9] early works on the notion of statecharts at Weizmann Institute of Science. Statecharts extend FSMs with many capabilities : events, conditions; hierarchy (state nesting); concurrence, and history states. In addition, as explained below, statecharts have built-in capabilities for describing their interaction with multiple objects in their environment. However, the Rhapsody tool has also full UML as support for the description and specification of systems : different types of diagrams can be used for the different design phases. For the functional specification phase, the problems addressed are related to the description of the structure, behavior (functionality) that makes the system, and the types of data exchanged between entities. Rhapsody incorporates mainly two types of diagrams to achieve these aspects : in this context, class diagrams and composite structure diagrams. As shown in figure (see fig), class diagrams are used to describe functional entities meaning: an atomic functionality of the system. The composite structure diagrams describe hierarchical entities that can contain instances (parts) of the functional classes. Statecharts describe behaviors similar to those of Finite State Machine; this behavior can be complemented by the definition of functions or actions implemented from the action language. A class can be defined as an active entity (concurrent), sequential (passive), or reactive. Communication between entities is done by direct reference or through the use of communication port. The connection between entities is assumed by the use of connector (Link). A composite structure can also optionally have a global behavior as statecharts for the control of sub-modules (units) therein.

## 3.3 MDD : model-driven development

The Rhapsody-Modaë toolchain has been designed using an intermediate meta-model (figure 2) that render both the structure of Modaë DSL and its channels semantics explicit. It may be usefull to recall that a meta-model (model of a model) is simply the expression of a set of concepts – represented as class diagram– exhibiting their possible relationships (inheritance,..) and hence allow the specification of languages (DSL) as object-oriented structure models. A model always conforms to a unique metamodel. Metamodeling is central to Model Driven Software Development (MDSD). This approach is proned by OMG and technically supported by many tools : in our case we used EMF (Eclipse modeling framework, [22]) for the definition of the metamodel and MDWorkbench for the transformations. MDWorkbench has been incorporated within Rhapsody (and renamed RulesComposer), in order to allow direct access and manipulation of the internal Rhapsody UML metamodel, which helps get complete control of the code generated in Rhapsody.

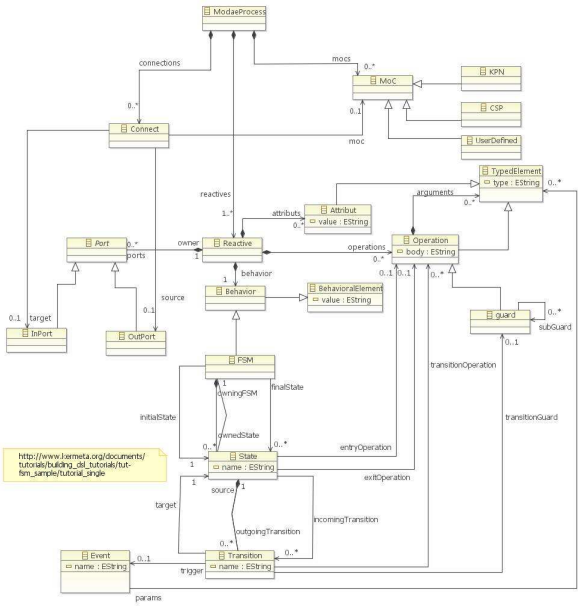


**Figure 2: Modaë metamodel enabling model-based transformation engineering between Rhapsody and HLS**

## 3.4 Modaë DSL and SLS

Modaë SLS is an innovative system-level behavioral synthesizer that takes DSL expressed in Ruby and Python interpretated languages as input, contrary to classical approaches that start with C language or its derivatives (like SystemC). The DSL itself is an internal DSL that take avantage of the host langage (e.g Ruby) for the expression of domain-specific notions. In our case, these notions are limited to ports, channels, graph of blocks (known as process networks), absent natively from the host language. A skeleton of such internal block expressed in Ruby DSL is presented in figure 3. The

assembly and synthesis flow of such blocks is explained in in figures 4 and 5.

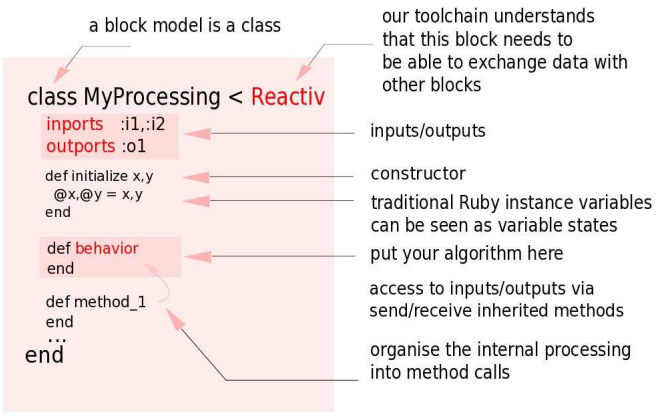


**Figure 3: Example of blocks expressed in Ruby DSL**

Our DSL also allows the expression of exact information on the types of variables, size of arrays, etc : they are used in source-to-source transformations by our compiler.

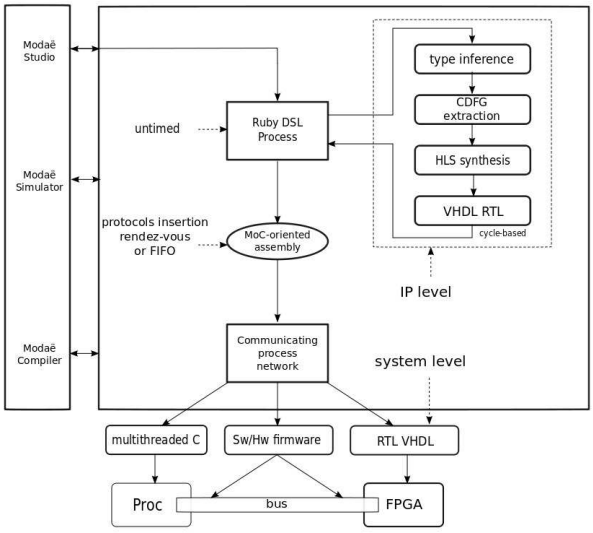


**Figure 4: Overall Modaë HLS flow.**

The main reason why interpretated languages make sense in the field of ESL is that algorithmicians tend to privilegiate these languages for early explorations, due to their simplicity and ease of use. Moreover they come with a growing set of excellent dedicated scientific libraries, and vizualisation capabilities. In this view, Ruby and Python can be compared to Matlab, with the supplemental advantages of being fully open-source, free of charge, multi-paradigm, multi-domain and multi-platform. With respect to HLS, they however raise some supplemental technical problems : they resort to many dynamic aspects, that are hard to deal with. To solve this issue, Modaë HLS embedds two typing mecanisms : static inference and probing. The former is performed within the compiler, that propagates known types in its IR (internal representation). The block-diagram oriented design methodology associated with Modaë studio facilitates this propagation : at the higher level of the block-diagram capture, the system has no inputs nor outputs. Dedicated blocks need to be present to feed or retreive data, and, as such, are also part of the system description. This typing strategy works for class-level typing : variables types can be found (Integer, Float, Arrays,...). For bit-level dimensioning, we resort to dynamic probing during simulation in order to detect variables ranges. Again, this is a rupture with previous approaches, that separate the design phase from simulation phase. We believe the huge effort made in simulation (code coverage etc) can be avantageously used for other purpose, like typing. HDL engineers are thus relieved from the burden of bit-exact design.

## 3.5 Underlying models of computations

Models of computations (MoC), as studied for instance in Ptolemy [5], describe the way concurrent processes (or "components") exchange with each other, via explicit access to ports, and basically how their synchronisation system works. Stated differently, it is the "set of rules that allows to compute the behavior resulting from the composition of the individual behaviors of the components" [7]. This simple definition also implies that it is possible to isolate the synchronization from the core algorithms of the processes : as stated by Maraninchi [16], MoC allows engineers to "forget as much as possible, as soon as possible". Many recent works take advantage of this separation of concerns, by providing system-level specifications at abstract (untimed) levels, with predefined MoCs (among others [23], [10] are applied to automotive). In Modaë block-diagram capture, several models of communication between block ports can be choosen by the user, through simple annotations 5.

A channel can behave as synchronous memory-less, functioning as a rendez-vous, or asynchronous (implemented as bounded FIFOs). Channels can be connected either one-to-one or one-to-many (synchronous broadcast). Other mecanims are available, but not used in this paper. During synthesis, small distributed finite-state machines (FSM) associated to each port and channel ensure the execution of the intended semantics. For instance, a a one-to-many synchronous channel needs as many FSMs as ports, plus a supplemental FSM for the rendez-vous.

# 4. EXPERIMENT

In order to illustrate the use of the design flow, we choose a simple abstract example. Our system is simply made of three concurrent untimed entities named 'forProc', 'whileProc', 'untilProc'. Each of them perform a possibly data-dependent computation and exchange data through ports : forProc" computes the square of the first ten naturals 0..9, the second one computes back the square root of these samples, using an iterative algorithm, and finally the last one acts as a pure sink (pumping the values from the second process). The system has been modeled in Rhapsody as three instances of classes (figure 6). Our MDD toolchain transforms the UML 2 model, transform it into a model conforms to our

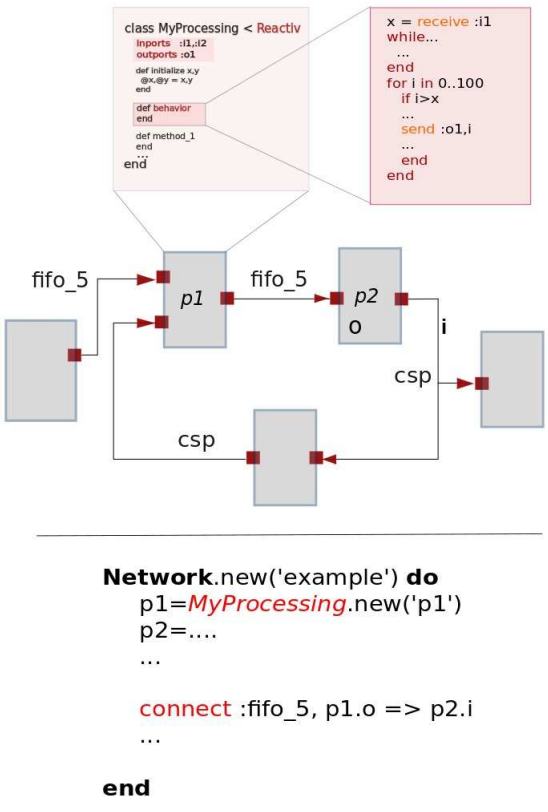


**Figure 5: Network of communicating processes assembled in Ruby DSL. Channels are tagged with a dedicated MoC.**

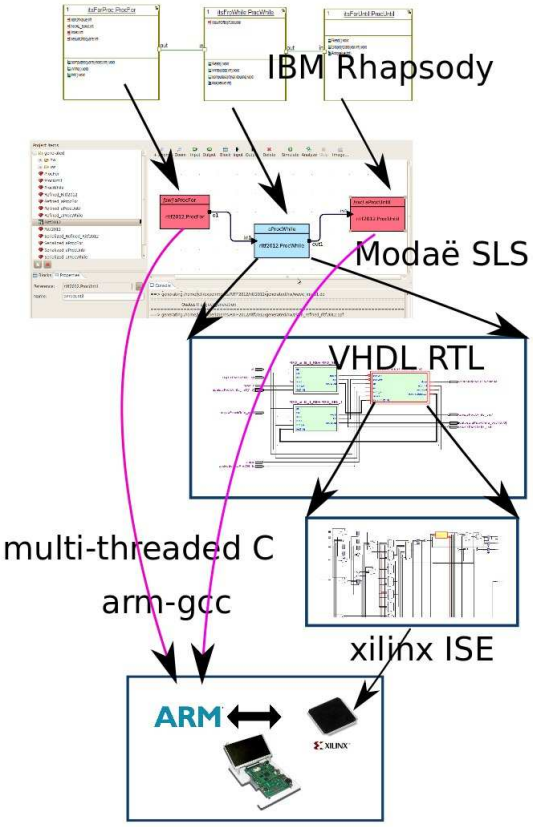


**Figure 6: HW/SW codesign using Modaë Studio, starting from UML 2 captured in IBM Rhapsody. The hardware plaform embedds a ARM Cortex A8 and a Xilinx Spartan 6 FPGA**

intermediate Modaë-compliant metamodel, and finally outputs a Ruby code, amenable to system-level synthesis. Once read back by Modaë studio the Ruby code can also be simulated. Moreover, a recent work [13] showed a convincing connection between Modaë models and formal verification by model-checking, which is a supplemental path to verify design correctness (verification of properties). In Modaë studio, a HW/SW partitionning has been precised, as well as a target : Armadeus APF 51 embedding a ARM Cortex A8 and a Xilinx Spartan 6 FPGA.

Finally, the RTL VHDL code and multi-threaded has been generated by Modaë synthesizer and compiled using classical tools (gcc cross-compiler and Xilinx ISE logic synthesis).

## 5. DISCUSSION AND FUTURE WORK

Among the surprising outcome of this experimental toolchain, we found it difficult to encode our algorithm into UML statecharts, compared to the agile capabilities of Modaë Ruby DSL : it may be easier to encode data-crunchy algorithms easier in this DSL instead of statecharts. However, Rhapsody remains a solid and widespread tool for high-level capture. A further experiment will be conducted to benefit from both tool during the first phase of the design, instead of considering Ruby DSL as just a backend format. Another interesting problem (not discussed here) relies on the correct encoding of events and exchange mecanism present in Rhapsody : the transposition in a different tool is delicate. The Rhapsody event-driven simulation mecanism (MoC) needs to be better understood to ensure transformation correctness. We notice that this effort in understanding an existing tool is also cited for Simulink [2] : that probably reveals that the underlying semantics of many tools make them uncompatible with the idea of semantically-sound toolchains. That problem is of utmost importance for society.

## 6. CONCLUSION

In this paper, we have proposed a system-level toolchain starting from UML 2.0 formalism captured in IBM Rhapsody, coupled with Modäe high-level synthesizer. The coulping itself has been designed using mainstream software model-driven tooling, based on metamodeling, which differ from previous works in the area. This innovative toolchain allows us to generate multithreaded embedded software and RTL hardware for mixte processor-FPGA plateforms, as well as communication links between both, which is a second differentiation. The experiment has been applied to Armadeus platforms showing the effectivness and usability of the approach. Our future work will focus on two aspects : new experiments in particular engineering fields and optimizations of communcations.

# 7. REFERENCES


[1] Embedded systems overhaul. it's time to tune up for the future of the automotive industry. Technical report, IBM-Institute for Business Value executive, 2004.

[2] O. Bouissou and A. Chapoutot. An operational semantics for simulink's simulation engine. In R. Wilhelm, H. Falk, and W. Yi, editors, *LCTES*, pages 129–138. ACM, 2012.

[3] P. Coussy and A. Morawiec. *High-Level Synthesis: from Algorithm to Digital Circuit*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[4] F. P. Coyle and et al. From uml to hdl: a model driven architectural approach to hardware-software co-design, information systems: New generations conference (isng), 2005.

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

[6] S. Friedenthal, A. Moore, and R. Steiner. *A Practical Guide to SysML: Systems Modeling Language*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[7] C. Hardebolle, F. Boulanger, D. Marcadet, and G. Vidal-Naquet. A generic execution framework for models of computation. In J. M. Fernandes, R. J. Machado, R. Khédri, and S. Clarke, editors, *MOMPES*, pages 45–54. IEEE Computer Society, 2007.

[8] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, June 1987.

[9] D. Harel and E. Gery. Executable object modeling with statecharts. *IEEE Computer*, 30:31–42, 1997.

[10] J. Keinert, M. Streubühr, T. Schlichter, J. Falk, J. Gladigau, C. Haubelt, J. Teich, and M. Meredith. SYSTEMCODESIGNER - An Automatic ESL Synthesis Approach by Design Space Exploration and Behavioral Synthesis for Streaming Applications. *ACM Transactions on Design Automation of Electronic Systems*, 14(1):1–23, 2009.

[11] A. Koudri, D. Aulagnier, D. Vojtisek, P. Soulard, C. Moy, J. Champeau, J. Vidal, and J.-C. Le Lann. Using MARTE in a Co-Design Methodology. In *MARTE UML profile workshop co-located with DATE'08*, page 6 pages, Munich, Germany, Mar. 2008.

[12] A. Koudri, A. Cuccuru, S. Gerard, and F. Terrier. Designing heterogeneous component based systems: Evaluation of marte standard and enhancement proposal. In J. Whittle, T. Clark, and T. KÃijhne, editors, *MoDELS*, volume 6981 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 2011.

[13] J.-C. Le Lann, P. Dhaussy, and P.-L. Lagalaye. Modélisation algorithmique et synthèse d'architectures assistées par model-checking. In *CAL 2012*, page xx, Montpellier, France, May 2012.

[14] S. Lecomte, S. Guillouard, C. Moy, P. Leray, and P. Soulard. A co-design methodology based on model driven architecture for real time embedded systems. *Math. Comput. Model.*, 53(3-4):471–484, 2011.

[15] F. Mallet, C. André, and R. de Simone. CCSL: specifying clock constraints with UML/Marte. *Innovations in Systems and Software Engineering*, 4(3):309–314, 2008.

[16] F. Maraninchi and T. Bouhadiba. 42: programmable models of computation for a component-based approach to heterogeneous embedded systems. In C. Consel and J. L. Lawall, editors, *GPCE*, pages 53–62. ACM, 2007.

[17] G. Martin and W. Müller. *UML for SOC Design*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[18] W. Mueller, D. He, F. Mischkalla, A. Wegele, P. Whiston, P. Penil, E. Villar, N. Mitas, D. Kritharidis, F. Azcarate, and M. Carballeda. The saturn approach to sysml-based hw/sw codesign. In *Proceedings of the 2010 IEEE Annual Symposium on VLSI*, ISVLSI '10, pages 506–511, Washington, DC, USA, 2010. IEEE Computer Society.

[19] I. R. Quadri, A. Gamatié, P. Boulet, S. Meftali, and J.-L. Dekeyser. Expressing embedded systems configurations at high abstraction levels with uml marte profile: Advantages, limitations and alternatives. *Journal of Systems Architecture - Embedded Systems Design*, 58(5):178–194, 2012.

[20] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A uml 2.0 profile for systemc: toward high-level soc design. In *EMSOFT*, pages 138–141, 2005.

[21] T. Schattkowsky. Uml 2.0 - overview and perspectives in soc design. *CoRR*, abs/0710.4641, 2007.

[22] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley, Boston, MA, 2. edition, 2009.

[23] M. Streubühr, M. Jäntsch, C. Haubelt, and J. Teich. From Model-based Design to Virtual Prototypes for Automotive Applications. In *Proceedings of the Embedded World Conference*, pages 1–10, Nuremberg, Germany, Mar. 2009.

[24] S. Taha, A. Radermacher, S. Gérard, and J.-L. Dekeyser. Marte: Uml-based hardware design from modelling to simulation. In *FDL*, pages 274–279. ECSI, 2007.

[25] R. Thomson, S. Moyers, D. Mulvaney, and V. Chouliaras. The uml-based design of a hardware h.264/mpeg-4 avc video decompression core falcon ml design flow. pages 1–6, 2008. ACM Order No.: 477081.

[26] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet. A co-design approach for embedded system modeling and code generation with uml and marte. In *DATE*, pages 226–231. IEEE, 2009.

[27] S. K. Wood, D. H. Akehurst, O. Uzenkov, W. Howells, and K. D. McDonald-Maier. A model-driven development approach to mapping uml state diagrams to synthesizable vhdl. *IEEE Transactions on Computers*, 57:1357–1371, 2008.