

Using the UML Profile for MARTE to MPSoC Co-Design

Jean-Luc Dekeyser
jean-luc.dekeyser@lifl.fr

Abdoulaye Gamatié
abdoulaye.gamatie@lifl.fr

Anne Etien
anne.etien@lifl.fr

Rabie Ben Atitallah
rabie.ben-atitallah@lifl.fr

Pierre Boulet
pierre.boulet@lifl.fr

LIFL
UMR 8022 CNRS – USTL
59655 Villeneuve d’Ascq

INRIA Lille – Nord Europe
40 Avenue Halley
59650 Villeneuve d’Ascq

Abstract

The increasing amount of hardware resources in next generation MultiProcessor Systems-on-Chip (MPSoC) calls for efficient design methodologies and tools to reduce their development complexity. This paper presents a candidate MPSoC design environment Gaspard2, which uses the MARTE (Modeling and Analysis of Real-Time and Embedded systems) standard profile for high-level system specification. Gaspard2 adopts a methodology based on Model-Driven Engineering. It promotes separation of concerns, reusability and automatic model refinement from higher abstraction levels to executable descriptions.

Keywords: MPSoC, MARTE, Gaspard2, MDE, co-design

1. Introduction

Context and motivation. Modern embedded applications are becoming more and more sophisticated and resource demanding. The concerned domains are numerous: state-of-the-art multimedia applications such as high-definition digital television, biometric data processing, etc. On the other hand, the current technological advances have permitted to increase the number of integrated transistors on a single chip, then to improve the computational power of chips. The resulting architectural paradigms consider multiple processors or cores on the same chip. This favors solutions to the highly critical problem of resource need in modern embedded systems. In such a context, however, the design and verification activities become very difficult due to the complexity of these systems. As a consequence, the productivity of designers strongly gets penalized. So, an important challenge is to find adequate design methodologies that efficiently address the issues about large and complex system-on-chip.

Model-driven engineering (MDE) has been introduced as possible answer to deal with the above design challenges.

It advocates the use of *models* at different abstraction levels so as to provide designers with different viewpoints of a system under design. Complex systems can be easily understood thanks to such abstract and simplified representations. The information exchanges between various domain experts are made easier. A model basically highlights the intention of a system without describing the implementation details. It is progressively enriched with technological information via *transformations* according to the ultimate purpose of the system design: behavioral simulation, verification, executable code generation, communication, etc. MDE inherits from methodologies developed in seventies to manipulate models. However in MDE, the resulting models must be comprehensive and interpretable by computers. MDE thus also covers the code generation which puts a model in a concrete form. In this way, it stands apart from the other methodologies based on models.

An interesting model specification language is UML [10], which proposes general concepts allowing one to express both behavioral and structural aspects of a system. The generality of these concepts can be semantically constrained in such a way that a specialization of the language becomes possible for a specific domain, e.g., embedded systems. The resulting modeling sub-language is generally termed a *profile*. The primary aim of the MARTE (*Modeling and Analysis of Real-Time and Embedded systems*) [15] standard profile proposed by the Object Management Group (OMG) is to add capabilities to UML for model-driven development of real-time and embedded systems. The new concepts and notions defined by this profile significantly improve the usual way of modeling software, hardware and relations between them in UML. Further extensions are provided to facilitate performance and scheduling analysis and to take into account platform services (e.g. services of an operating system).

Related work. The UML-based design of systems is a very young discipline (less than a decade). The first attempts (e.g. the UML profile for SystemC based on UML

1.4 [4]) faced the problem of limited supporting tools and capabilities to target executable platforms. To overcome these obstacles, significant efforts have been made, which led to the MDA initiative of the OMG. MDA identifies a set of model transformation guidelines from platform-independent points of view towards platform-specific ones.

As an improvement of the first attempts, further propositions have been suggested [12] [11]. SysML [12], on which some concepts of MARTE are based, is one of them. However, it has the disadvantage of being too much generic. In the opposite, the standardization proposal by Fujitsu [11] about a UML profile for SoC tended to be very close to the implementation level, providing concepts very tied to SystemC. None of these works investigated model transformation issues, and specifically how to reach the implementations from high-level specifications.

More recently, there have been works concerned by obtaining a realization out from the modeling of SoCs, proposing *model transformations*. They highlighted the need for model notation to have an entirely *executable* semantics [8]. Such an expressive notation must allow one to define models with sufficient information so that each part of the system can be completely realized. So far, the propositions [8, 1, 14] have been limited as direct transformations from UML profiles to SystemC simulations, taking little out of the powerful MDE approach.

The vision adopted in Gaspard2 [6] environment consists of an efficient usage of MDE by abstracting more the SoC specification level and by leveraging the model transformations to target the multiple types of outputs that are needed during the SoC design.

2. Design concepts for MPSoC

The modeling of SoC in Gaspard2 almost relies on the MARTE profile (and of course on UML). One of the main advantages of MARTE is that it clearly distinguishes concepts for the description of application functionality, hardware architecture, and the mapping of the former on the latter. For SoC design, this separation is of prime importance as it is usual to create those two parts of the system simultaneously, by different teams. Moreover, it provides a flexible way to independently change the functionality part or the hardware part. For instance, one can analyze the same functionality on different kinds of hardware architecture, or conversely the same architecture can be reused for different applications.

2.1. Repetitive structure modeling of MARTE

The *repetitive structure modeling* (RSM) of MARTE [9] proposes concepts that enable to describe in a compact way the regularity of a system's structure or topology. The considered structures are composed of repetitions of structural elements interconnected via a regular connection pat-

tern. It provides the designer with a way to express models efficiently and explicitly with a high number of identical components. RSM is originally inspired by the domain-specific language Array-OL dedicated to intensive multidimensional signal processing [2]. Fig. 1 depicts the basic stereotypes associated with the RSM package.

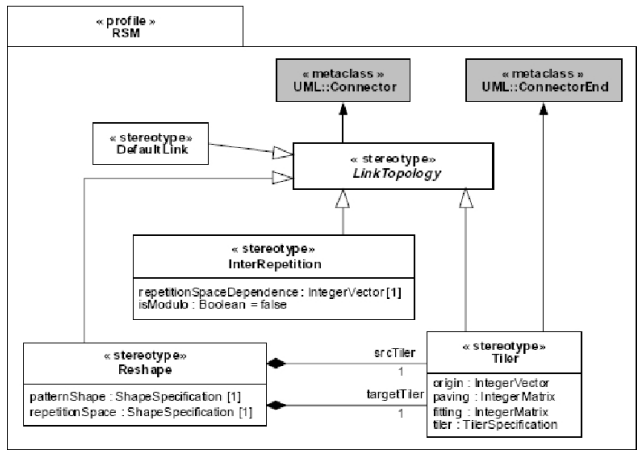


Figure 1. RSM package of MARTE.

According to the Model of Computation (MoC) of Array-OL, a data-parallel task T is repeated or replicated into several task instances $\{T_i, i \in 1..k\}$ that take as inputs subsets of data extracted from the inputs of T , which are multidimensional arrays. These subsets of array elements are referred to as *patterns* or *tiles*. For a given task repetition, the number k of task instances T_i is given by the *repetition space* associated with the task T .

The *Tiler* stereotype, described in Fig. 1, expresses how multidimensional arrays are tiled by patterns. When applied to a delegation connector, a *Tiler* connects an external port with a port of an internal part. The shape of the external ports defines the shape of input/output arrays of a task. The port shape of the internal part defines the pattern shape and the shape of the part itself defines the repetition space.

A *Tiler* uses three main information (see the attributes of the corresponding stereotype) to define the tiling operation:

- **origin** vector, which specifies the origin of the reference tile in the array;
- **fitting** matrix, which specifies how the patterns are filled with array elements;
- **paving** matrix, which specifies how an array is covered by pattern elements.

The *Reshape* stereotype extends the UML 2 Connector metaclass. It enables to represent complex link topologies in which the elements of a multidimensional array are redistributed in another array. The major difference between the *Tiler* and *Reshape* stereotypes is that the former is used for delegation connector (i.e. between the port of a component and the ports of its parts) and the latter is used for connector that links two parts. The *InterRepetition* stereotype is

used to specify a data dependency between repetitive task instances. For a full description of all these stereotypes, the reader may refer to the MARTE specification document [9].

2.2. Basic modeling features

Application functionality. The application functionality generally consists of data intensive computation algorithms. The above concepts that define the Repetitive Structure Modeling are used to specify these algorithms. They offer a well-suited way to describe both data parallelism and task parallelism in the application. The resulting application model is represented concisely in a declarative way.

There are basically three kinds of task components that are used to specify application functionality. An *elementary* component defines a function as an atomic block. A *repetitive* component expresses the data-parallelism in an algorithm according to RSM concepts (for an illustration, see the application model defined in Section 3). Finally, a *hierarchical* component enables to specify complex functionalities in a modular way; in particular, task parallelism can be described using such a component.

Hardware architecture. Compared to UML, the MARTE profile enables one to describe hardware properties in a more precised way. The architecture of the hardware is described in a structural way. The notion of component permits to reuse a set of basic components in various places and to define the architecture hierarchically. In Gaspard2, only the HW_Logical sub-package is used. It allows to describe information about the kind of available components (HwRAM, HwProcessor, HwASIC, HwBus...), their non functional characteristics, and how they are connected to each other. In Gaspard2 no use is done of the HW_Physical sub-package because when the system is entirely on one chip, synthesis tools can compute automatically an optimized layout.

Similarly to application components, the structure of a hardware architecture can also be described using the repetitive Model of Computation based on Array-OL [2]. This is very useful when describing, e.g. a grid of processing elements such as the Tile64 architecture of Tileria [16].

Mapping of application on architecture. Since hardware architecture and application functionality are modeled independently, the concepts that express the mapping of the latter on the former are obviously required. Gaspard2 leverages from the MARTE profile the Alloc package, which offers the needed concepts to describe different kinds of mapping. In MARTE, the mechanism to specify this mapping is called *allocation*. The set of all the allocations of the model defines the mapping. The concept used to specify an allocation is **Allocate**. It is used for associating elements from a logical context, application model elements, to named elements described in a more physical context, execution platform model elements.

An allocation can represent either a spatial placement or a temporal placement. Spatial placement is the allocation

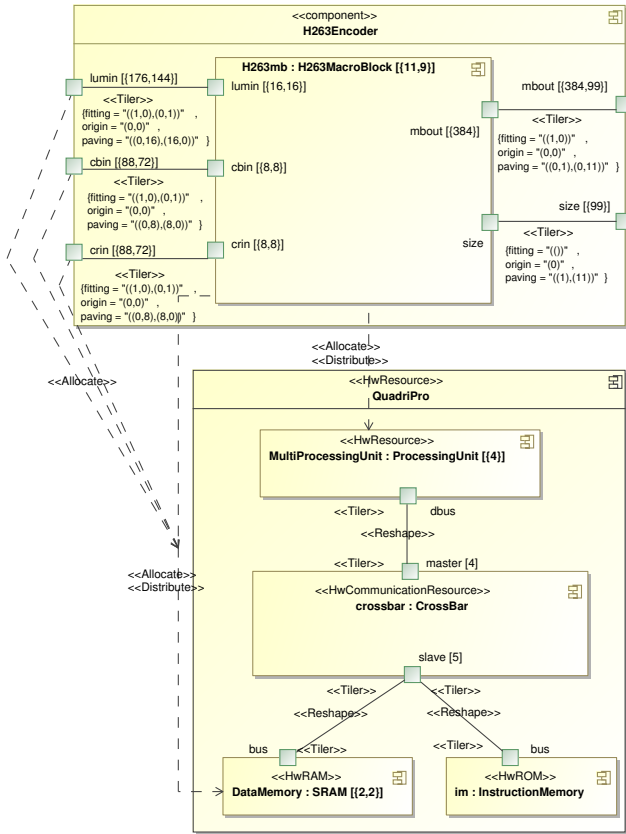
of computations to processing resources, of data to memories at specific ranges of addresses, and of dependencies between application components to communication resources. Temporal placement is the scheduling of a set of elements spatially allocated to the same platform resource. For instance, several tasks can be scheduled on a processor, or a range of memory addresses can be used at different times to hold different data. Only spatial placement is supported in Gaspard2. Temporal placement is part of future work.

2.3. A useful extension: deployment

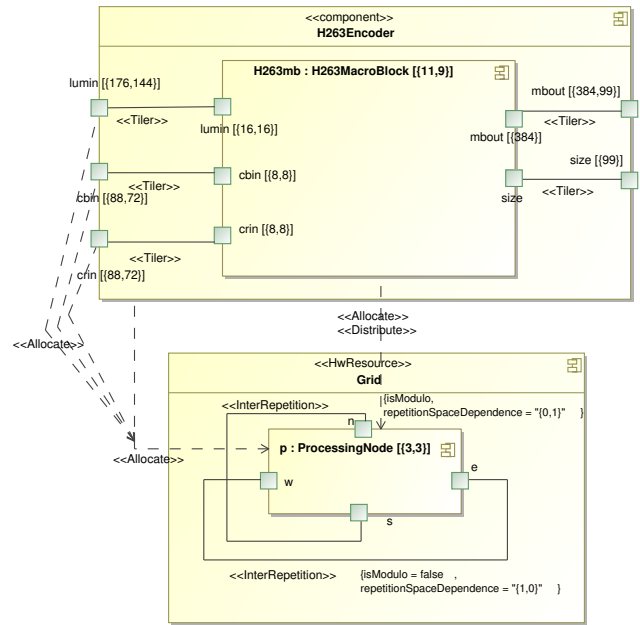
MARTE strives to be generic, for its best, so that it can accommodate a large set of needs. As a result, some of the information required for the implementation is not expressible only with the provided concepts. Thus, Gaspard2 introduces additional concepts and semantics to fill up this need in the particular domain of SoC modeling. This is done via a specific UML profile. This is a very good demonstration of the advantage of the UML profiling mechanism: use of largely understood standard concepts, and introduction of additional and very precise concepts for the specific domain of interest. Below are presented the major two extensions introduced for the need of MPSoC automatic generation.

The first addition to MARTE concerns the semantics of the application modeling. In MARTE, as in UML, mostly any kind of application can be specified but the behaviour of the application cannot be entirely defined, it is up to the programmers to code the precise behaviour of each component. In Gaspard2 we have restricted the domain of application that can be modeled to data-flow systematic applications, which is one of the major domains of application encountered in the field of MPSoC. Beyond UML concept **Component** (for the application component) and the MARTE stereotypes **FlowPort** (for all the ports), the semantics considers the repetitive MoC [2].

The second extension of Gaspard2 over the MARTE profile concerns the description of the elementary components: the components which are the basic blocks of all the other components. To transform the high abstraction level models into simulation code, very detailed deployment information must be provided. In particular, every elementary component must be linked to an existing code, for both the hardware and the application. A specificity of the SoC design is that one functionality can be implemented in different ways. This is necessary for testing the system with different tools, or at different levels of abstraction, along the design phase. Moreover, this is necessary to facilitate application component reuse on different hardware components: e.g. an application component can be optimized for a specific processor, or written as a hardware accelerator. Hence, we have introduced the concept of **AbstractImplementation** to express a hardware or software functionality, independently from the compilation target. It contains one or several **Implementations**, each one being used to define a specific implementation at a given simulation level and programming language.



(a) Mapping onto a shared memory architecture



(b) Mapping onto a distributed memory architecture

Figure 2. Two mapping choices for the same application functionality.

The Implementation which fits best the target will be used to reify the component during the compilation phase. This automatic selection allows to generate the exact same SoC model at different simulation levels. Additionally, the concept of CodeFile is used to specify the code and compilation options required.

3. Design examples

In this section, we present how the concepts introduced previously are used to design an image processing application and its mapping on different architectures in Gaspard2. The deployment of a sub-part of the resulting system is also illustrated. The considered application is an H.263 [5] video coding algorithm. It aims at compressing a video stream originally represented as a sequence of QCIF pictures (176×144 pixels in YCbCr format). Due to space restriction, we do not show the entire model of the application. The reader may refer to [13] for further details.

3.1. Application, architecture and mapping

First scenario. On the top of Figure 2(a) is shown the high hierarchical level of the application model. The H263Encoder component reads one picture decomposed

in three parts, corresponding to the three ports lumin, cbin, crin. It links via Tilers the processing of block of pixels to the H263MacroBlock component within which a specific algorithm, called *Huffman coding function*, is applied to each block (it will not be detailed here).

The result of the processing is then reassembled via other Tilers and provided as output.

At the bottom of Figure 2(a) is shown a model of the main component of a hardware architecture, called QuadriPro. It is composed of four processing units (specified using the repetition concept), a RAM, a ROM, and a crossbar to interconnect those components together. The MultiProcessingUnit component is not stereotyped Hw-Processor because this component is composed of both a processor and a cache. The Reshape connectors permit to specify which component is connected to which repetition of the slave port of the crossbar.

The mapping of the whole H263Encoder application on the QuadriPro hardware architecture is expressed via the Allocate links specified in Figure 2(a). On the one hand, the ports lumin, cbin and crin of the application are mapped on the RAM DataMemory in the architecture. On the other hand, the computation part of the application, represented by H263mb component instances, is mapped on the processing units of the hardware architecture. Here, note that

the memory can be accessed by all processing units, i.e. the modeled architecture is *shared memory*.

The `Distribute` stereotype associated with the association links proposes a way to express regular (e.g. block, cyclic, k-cyclic) distributions from an array of elements to another array of elements. It is an extension of the allocation mechanism described above designed to handle the repetitive structures. A detailed definition of its semantics can be found in [3].

Second scenario. Now, let us consider the same application model mapped on a new architecture description as shown in Figure 2(b). This architecture differs from the previous one in that it assumes a *distributed memory*. It consists of a toroidal 3×3 -grid of processing nodes where each node holds its own memory.

This second model defines a new implementation of the system that is straightforwardly obtained by reusing the existing application model on the new hardware architecture model. This is an important advantage of using models for system design.

3.2. Deployment

The result of the above mappings is an abstract description in which the choice of the target implementation platform remains to decide. The deployment phase enables one to precise a specific implementation for each elementary concept among a set of possibilities.

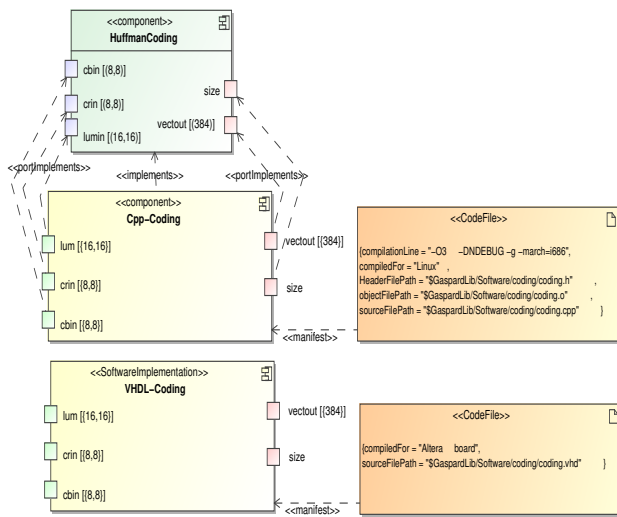


Figure 3. Deployment of a component.

Figure 3 expresses two alternatives for the deployment of the HuffmanCoding function that applies to each macro block obtained from an input image of the H263Encoder task. The first alternative suggests a C++ implementation of the function while the other proposes a VHDL implementation. Here, the `implements` and `portImplements` links express the fact that the former alternative is chosen because the expected final implementation is generated in SystemC.

Although the deployment is a specific notion introduced in Gaspard2, the IP models that are used to deploy abstract elementary components are described using MARTE concepts. These IPs are usable for both application functionalities and hardware architecture.

4. Model refinement in Gaspard2

The design entry point within the Gaspard2 environment are MARTE models, which are the only specifications users have to provide. These models describe the aspects illustrated in Section 3. They are afterwards refined into further models that feature specific implementations. This refinement process is achieved via automatic transformations provided by the Gaspard2 environment. Several studies [13] [7] have shown that the generated code with Gaspard2 is reasonably efficient compared to a manually written code. However, a very interesting point is that the automatic generation enables to save an important amount of coding time. This is particularly valuable during design exploration since only minor modifications of high-level models are necessary to automatically re-generate new implementations. Such modifications may potentially concern the application functionalities, the hardware architecture, the mapping of both or the IP deployment.

On the other hand, MDE reduces here the difficulties to develop and maintain high performance computing co-design tools. At each abstraction level the concepts and the relations between these concepts are precisely defined. The refinement from high abstraction levels to low abstraction levels are taken over by compilation/transformation steps.

Overview of the Gaspard2 transformations. More concretely, the model transformations developed in the Gaspard2 environment in order to implement the MPSoC compilation flow, are illustrated in Figure 4. The top level of the schema corresponds to the high-level modeling concepts described by different metamodels: Application, Architecture, Association, IP and Deployed. Note that at this level, the refactoring of system models is possible whenever an adaptation of the mapping between application functionality and hardware architecture is necessary for better implementation performances.

Different transformation chains are defined, which refine high-level models towards specific technologies: synchronous languages for formal validation, SystemC for simulation, OpenMP Fortran for execution and VHDL for circuitry synthesis. The output model of a transformation is used as the input of another. These models, respecting predefined metamodels, provide strongly documented "synchronization points" in the compilation flow. Each transformation is independent from the others. This facilitates the reuse of transformations while compiling towards different abstraction levels. Moreover the compilation flow adapts easily to the changes of the fast evolving SoC domain.

Declarative languages are well-suited for the description

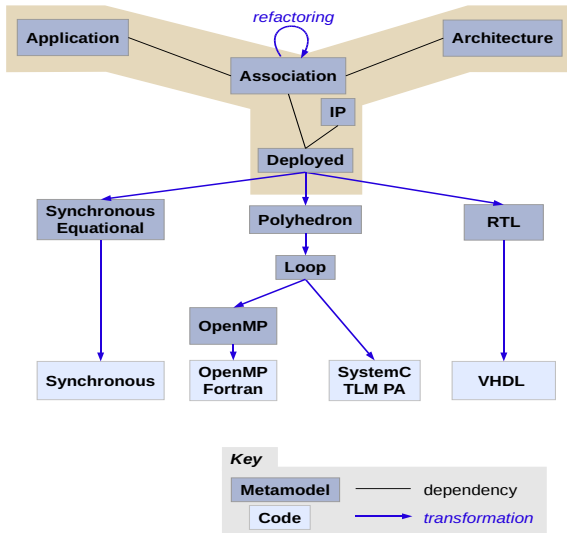


Figure 4. Gaspard2 transformation chains.

of the model transformations in that modifications are simplified and their maintainability is increased. The declarative approach specifies that each transformation is separated into a set of rules. Each rule is associated with an explicit set of input and output patterns. In Gaspard2, the engine we used to execute the transformations is a framework which permits to define declarative structures in Java.

5. Conclusions

In this paper, we have shown how the UML MARTE profile provides interesting concepts for the co-design of MPSoC in the Gaspard2 environment. The resulting high-level descriptions are refined towards specific technologies for various purposes: formal verification, execution, simulation or circuitry synthesis. This is achieved via automatic transformation chains implemented according to MDE principles. From an overall point of view, we can note the following important aspects provided by Gaspard2 for MPSoC design: *separation of concerns* by allowing one to describe separately application functionality, hardware architecture, mapping and deployment on target platform; *reusability* by providing the possibility to reuse already defined concepts during design and transformation; *efficient and compact representation* of models by offering the notion of repetition that suits for the definition of MPSoC topologies; and finally, *refinement* by proposing several model transformation chains that connect different abstraction levels from which various design aspects can be addressed.

6. Acknowledgment

We would like to gratefully thank all the members of the DaRT team of LIFL/INRIA, who actively participate to the work presented in this paper.

References

- [1] M. Alanen, J. Lilius, I. Porres, D. Truscan, I. Oliver, and K. Sandstrom. Design method support for domain specific soc design. In *Proceedings of MBD-MOMPES'06 Workshop*, pages 25–32, Washington, DC, USA, 2006. IEEE Computer Society.
- [2] P. Boulet. Array-OL revisited, multidimensional intensive signal processing specification. Research Report RR-6113, INRIA, Feb. 2007.
- [3] P. Boulet, P. Marquet, E. Piel, and J. Taillard. Repetitive Allocation Modeling with MARTE. In *Forum on specification and design languages (FDL'07)*, Barcelona, Spain, Sept. 2007. Invited Paper.
- [4] F. Bruschi and D. Sciuto. SystemC based design flow starting from uml model. In *Proceedings of ESCUG*, 2002.
- [5] G. Cote, B. Erol, M. Gallant, and F. Kossentini. H.263+: video coding at low bit rates. *IEEE Trans. On Circuits And Systems For Video Technology*, Nov. 1998.
- [6] DaRT Team LIFL/INRIA, Lille, France. Graphical array specification for parallel and distributed computing (GAS-PARD2). <https://gforge.inria.fr/projects/gaspard2/>, 2008.
- [7] S. Lebeux. *Un flot de conception pour applications de traitement du signal systématique implémentées sur FPGA à base d'Ingénierie Dirigée par les Modèles*. Thèse de doctorat (PhD Thesis), Université des sciences et technologies de Lille, France, Dec. 2007.
- [8] K. D. Nguyen, Z. Sun, P. S. Thiagarajan, and W.-F. Wong. Model-driven SoC design via executable UML to SystemC. In *RTSS '04: Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*, pages 459–468, Washington, DC, USA, 2004. IEEE Computer Society.
- [9] Object Management Group. A uml profile for marte, 2007. <http://www.omg-marte.org>.
- [10] Object Management Group, Inc., editor. *UML 2 Superstructure (Available Specification)*. <http://www.omg.org/cgi-bin/doc?ptc/2004-10-02>, Oct. 2004.
- [11] Object Management Group, Inc., editor. *UML Extension Profile for SoC RFC*. <http://www.omg.org/cgi-bin/doc?realtime/2005-03-01>, Mar. 2005.
- [12] Object Management Group, Inc., editor. *Final Adopted OMG SysML Specification*. <http://www.omg.org/cgi-bin/doc?ptc/06-0504>, May 2006.
- [13] E. Piel. *Ordonnancement de systèmes parallèles temps-réel, De la modélisation à la mise en œuvre par l'ingénierie dirigée par les modèles*. Thèse de doctorat (PhD Thesis), Université des sciences et technologies de Lille, France, Dec. 2007.
- [14] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A model-driven design environment for embedded systems. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 915–918, New York, NY, USA, 2006. ACM.
- [15] L. Rioux, T. Saunier, S. Gerard, A. Radermacher, R. de Simone, T. Gautier, Y. Sorel, J. Forget, J.-L. Dekeyser, A. Cucuru, C. Dumoulin, and C. Andre. MARTE: A new profile RFP for the modeling and analysis of real-time embedded systems. In *UML-SoC'05, DAC 2005 Workshop UML for SoC Design*, Anaheim, CA, June 2005.
- [16] TILE64 Processor Family. <http://www.tilera.com/products/processors.php>.