A Model-Driven Design Framework for Massively Parallel Embedded Systems

ABDOULAYE GAMATIÉ, LIFL Lab and INRIA
SÉBASTIEN LE BEUX, École Polytechnique Montréal
ÉRIC PIEL Technical University of Delft
RABIE BEN ATITALLAH, UVHC, LAMIH
ANNE ETIEN, PHILIPPE MARQUET, and JEAN-LUC DEKEYSER, LIFL Lab and INRIA

Modern embedded systems integrate more and more complex functionalities. At the same time, the semiconductor technology advances enable to increase the amount of hardware resources on a chip for the execution. Massively parallel embedded systems specifically deal with the optimized usage of such hardware resources to efficiently execute their functionalities. The design of these systems mainly relies on the following challenging issues: first, how to deal with the parallelism in order to increase the performance; second, how to abstract their implementation details in order to manage their complexity; third, how to refine these abstract representations in order to produce efficient implementations.

This article presents the GASPARD design framework for massively parallel embedded systems as a solution to the preceding issues. GASPARD uses the repetitive Model of Computation (MoC), which offers a powerful expression of the regular parallelism available in both system functionality and architecture. Embedded systems are designed at a high abstraction level with the MARTE (Modeling and Analysis of Real-time and Embedded systems) standard profile, in which our repetitive MoC is described by the so-called Repetitive Structure Modeling (RSM) package. Based on the Model-Driven Engineering (MDE) paradigm, MARTE models are refined towards lower abstraction levels, which make possible the design space exploration. By combining all these capabilities, GASPARD allows the designers to automatically generate code for formal verification, simulation and hardware synthesis from high-level specifications of high-performance embedded systems. Its effectiveness is demonstrated with the design of an embedded system for a multimedia application.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.6 [**Software Engineering**]: Programming Environments; C.0 [**Computer Systems Organization**]: General—*Systems specification methodology*; C.1 [**Computer Systems Organization**]: Processor Architectures; C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Real-time and embedded systems; signal processing systems*

General Terms: Design

Additional Key Words and Phrases: Embedded system design, system-on-chip, model driven engineering, MARTE standard profile, parallel programming, repetitive structure modeling

ACM Reference Format:
Gamatié, A., Le Beux, S., Piel, É., Ben Atitallah, R., Etien, A., Marquet, P., and Dekeyser, J.-L. 2011. A model-driven design framework for massively parallel embedded systems. ACM Trans. Embed. Comput. Syst. 10, 4, Article 39 (November 2011), 36 pages.
DOI = 10.1145/2043662.2043663 http://doi.acm.org/10.1145/2043662.2043663

Authors' addresses: A. Gamatié, LIFL (UMR CNRS 8022) lab and INRIA Lille Nord-Europe, France; email: abdoulaye.gamatie@lifl.fr; S. Le Beux, École Polytechnique Montréal, Montréal (Québec); É. Piel, Technical University of Delft, The Netherlands; R. Ben Atitallah, UVHC, LAMIH, F-59313 Valenciennes, France; A. Etien, P. Marquet, and J.-L. Dekeyser, LIFL (UMR CNRS 8022) lab and INRIA Lille Nord-Europe, France.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from the Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2011 ACM 1539-9087/2011/11-ART39 $10.00
DOI 10.1145/2043662.2043663 http://doi.acm.org/10.1145/2043662.2043663

1. INTRODUCTION

Modern embedded systems are becoming more and more sophisticated and resource demanding. The concerned application domains are numerous: state-of-the-art multimedia applications such as video encoding/decoding; software-defined radio-detection systems such as radars, sonars; and telecommunication systems such as mobile phones, antennas, etc. The performance requirements of such systems are very important. The recent technological advances that enable the integration of an increasing number of transistors (currently up to a billion) on a single chip greatly contribute to meet these requirements. In this aim, *Systems-on-Chip* (SoCs) have been a very promising solution. Because of the physical restrictions in terms of frequency and voltage, which have a real impact on the computational performance, the expansion of the processing power of a SoC requires to put multiple and possibly heterogeneous processors or cores into a single chip. For this reason, the exploitation of the parallelism available in *Multi-Processor System-on-Chips* (MPSoCs) architectures is a very attractive solution for the execution of high-performance applications.

1.1 Design Challenges

This section discusses some critical design challenges faced by designers of high-performance embedded systems. Some of these challenges concern embedded systems in general and are largely discussed in literature [Sangiovanni-Vincentelli 2007]. From these discussions, a number of important requirements are identified as a basis to define well-suited design methodologies for these systems.

1.1.1 Need to Deal with Parallelism and Regularity. In the current industrial practice, hand-coding is still widely adopted in the development of embedded systems. Parallelism is managed at a low level by different expert teams, which have a deep knowledge of the system (both hardware and software parts) in order to meet the performance requirements. Hand-coding is clearly not suitable for an efficient development of large embedded systems because it is very tedious, error-prone, and expensive. Another way to deal with parallelism consists in using general parallel programming models such as the distributed memory programming model *Message Passing Interface* (MPI) [Message Passing Interface Forum 2009] or the shared memory model OpenMP [OpenMP Architecture Review Board 2009].

Both programming models provide developers with a set of directives and library of routines that are used to express parallel computations. An MPI program identifies a set of processes and explicitly manages low-level communications between them. This parallelism expression is on one side far from our application structure which mainly manipulates regular streams of parallel data and, on the other side, far from our target execution platform which is a hardware platform with a limited amount of operating system and runtime layers. The same objection applies for OpenMP, although it focuses on a thread-based execution of parallel iterations rather than on communicative tasks.

Moreover, in the case of SoCs, the architecture itself has to be specified in order to satisfy particular design constraints. Recent propositions of data-parallel languages such as Brook [Stanford Streaming Supercomputer Project 2009] or OpenCL [Khronos Group 2009] that allow the expression of the algorithmics in our application do not include any description of the architecture or the application mapping on the architecture. They rely on a compiler to generate the mapping on a predefined class of architectures such as GPU in the frame of the GPGPU (General-purpose computing on graphics processing units) approach, or a set of multicore processors.

For all these reasons, we believe that the design of high-performance embedded systems particularly calls for new expressive *parallel programming paradigms*. Such paradigms should provide designers with some efficient ways to separately represent, at a high level, all the potential parallelism that is inherent to both software applications and hardware architectures. Furthermore, they should be rich enough to explicitly express different mapping possibilities of the application on the architecture, taking into account embedded system parallelism.

Most of the high-performance system examples mentioned previously (multimedia systems, signal processing in radar and sonar systems, etc.) achieve a huge number of calculations in parallel and in a regular manner. Typically, in image processing algorithms, filters are often applied modularly and regularly to regions (columns or lines) of each image. The same holds in radar signal processing where filters are applied to regular samples of the transformed signals. In the literature, such regular embedded systems are referred to as "embarrassingly parallel" when the different calculations performed in parallel are independent. Beyond the algorithmic side, the regularity can be also found in the architecture topologies of high-performance embedded systems such as MPSoCs. An example is the Tile64 architecture of Tilera [Tilera Corporation 2009], which consists of a grid of 64 processing elements. Since the regularity of these systems is a central feature, we also need parallel programming paradigms that efficiently exploit this feature. In Gaspard, our models take into account both task parallelism and data parallelism, as well on the application description side as on the architecture description side. The data parallelism-level expression is factorized, thus the expression of "embarrassing parallelism".

1.1.2 Need of Abstract Models. The design of SoCs is facing today a strong pressure on reducing time-to-market while the complexity of these systems has been increasing. In addition to this dilemma, we notice that the initial cost for the physical realization of the SoC (the mask creation of the chip) is very expensive. Such an outlay strongly imposes a more careful development of prototypes for design analysis and validation. System developers have to rely on some costless means allowing them to simulate and analyze the behavior of designed systems before their realization.

Design abstraction offers a possible solution to address the preceding issues concerning the time-to-market and complexity dilemma, and the SoC development cost. More concretely, one needs models that capture the strict relevant information depending on the required abstraction level. The global complexity of a system is addressed from multiple viewpoints or abstraction levels, so that one is able to easily focus on some specific aspects. Abstract models favor an efficient design reuse, typically through incremental refinements from higher-level models to lower-level models. Here, by refinement, we mean a manual or automatic transformation that makes a given model more concrete with respect to a target representation. On the other hand, since models are often executable and verifiable, they also serve as an interesting support for both behavioral simulation and property analysis without having necessarily the actual implementation of the systems. In some cases, they are even used to automatically synthesize this implementation. Finally, abstract models enable to deal with the heterogeneity of a system since its components can be manipulated at high description levels that suitably abstract away the specific details of each component.

1.1.3 Need of Seamless Methodologies. The development of a SoC usually starts with the concurrent design, or *codesign*, of both software application and hardware architecture. Then, the application part is mapped onto the hardware part, during the association phase. Finally, simulation models from various abstraction levels are generated for the whole system. The different aspects of this development process are potentially

handled by different domain experts who must communicate safely in order to achieve the resulting design. In such a context, the design and analysis activities become very difficult due to the ever-increasing complexity of SoCs. As a consequence, the productivity of designers strongly gets penalized. Another critical aspect concerns the design space exploration, that is, how the analysis and the simulation results obtained from the different abstraction levels are exploited for an efficient redesign by modifying the high-level system models. So, an important challenge here is to find design methodologies with supporting tools that adequately address all these issues concerning large and complex embedded systems.

1.2 Our Proposition: The GASPARD Framework

Over the past five years, we have been extensively working on the definition of a design framework [Dekeyser et al. 2008] for the development of massively parallel embedded systems, which addresses the aforesaid challenges. Here, by framework, we mean an environment that provides designers with at least the following means: a formalism for the description of embedded systems at a high abstraction level, a methodology covering all system design steps, and a tool-set that supports the entire design activity. Our resulting design framework is referred to as GASPARD[1] (*Graphical Array Specification for Parallel and Distributed Computing*).

The design of SoCs in GASPARD specifically relies on the repetitive Model of Computation (MOC) [Boulet 2008], which offers a very suitable way to express and manage the potential parallelism in a system. This MOC is inspired by ARRAYOL [Demeure and Del Gallo 1998], a domain-specific language originally dedicated to intensive signal processing applications. It extends the basic notions of this language and offers an elegant and very expressive way to describe both task parallelism and data parallelism, and the combination of both.

The repetitive MOC is used in GASPARD, via the MARTE standard profile [Object Management Group 2007a] and more precisely its RSM package, to describe parallel computations in the application software part, the parallel structure of its hardware architecture part, and the association of both parts. MARTE stands for Modeling and Analysis of Real-time and Embedded systems. It is an evolution of the UML SPT profile [Object Management Group Inc. 2005] and borrows some concepts from the more general SysML profile [Object Management Group Inc. 2006]. In GASPARD, the resulting abstract models are enriched with specific information according to the target technologies. Finally, different automatic refinements from the higher abstraction level are defined, according to the Model-Driven Engineering (MDE) paradigm, towards lower levels for various purposes: simulation at different abstraction levels with SystemC [Ben Atitallah et al. 2007b; Piel et al. 2008b], hardware synthesis with VHDL [Le Beux et al. 2008], formal validation with synchronous languages [Yu et al. 2008], high-performance computing with OpenMP Fortran and C [Taillard et al. 2008a]. In this article, we mainly consider the first three facilities to deal with the design space exploration of high-performance embedded systems, implemented on globally heterogeneous architectures with regular and homogeneous multiprocessor parts.

Figure 1 gives an overview of GASPARD according to the used packages. The appearance of the packages varies with the implication of our research group in the definition of the specification concepts: empty boxes mean no implication in the definition of the corresponding packages; hatched boxes mean only a partial participation to the definition of the packages; and dark boxes mean the full definition of the whole packages. As shown in the figure, GASPARD partly relies on MARTE packages, with additional

[1] http://www.gaspard2.org

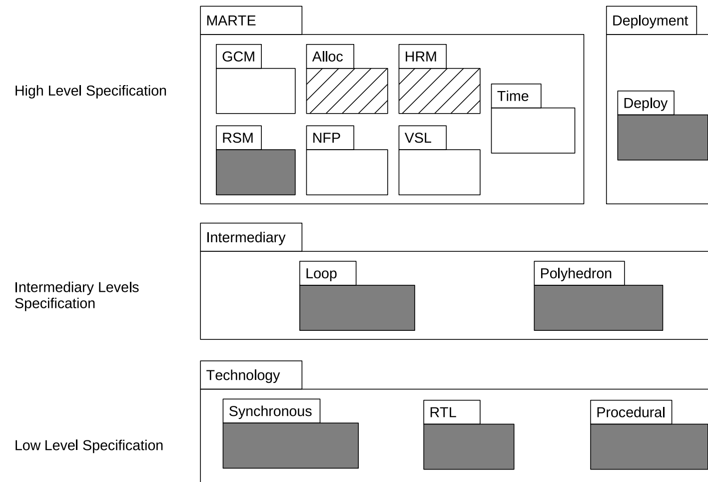

Fig. 1. An overview of GASPARD design packages: an empty box means no implication of authors in the definition of the package; a hatched box means only a partial participation of authors; and a dark box means definition of the whole package by authors.

packages specified at different abstraction levels in order to deal with concepts, required to target different technologies via model transformations. Among the MARTE packages, GASPARD uses the following.

— the GCM (Generic Component Model) package for application design. This package contains basic concepts such as data flow ports and components;
— the HRM (Hardware Resource Modeling) package for architecture representation. It specializes the concepts of GCM into hardware resource such as memory, processor, etc.
— the Alloc package for the representation of the mapping between applications and architectures;
— the RSM (Repetitive Structure Modeling) package for repetition specification in an application, an architecture and the mapping of the two;
— the NFP (NonFunctional Properties) package for the specification of nonfunctional properties such as the frequency of a processor;
— the VSL (Value Specification Language) package for the structuring and specification of values;
— the time package for the specification of temporal properties such as clock constraints (the use of this package in GASPARD is not in the scope of this article).

The purpose and meaning of the remaining packages, which are specific to GASPARD, are presented further in the article.

More generally in this article, the main contribution is the presentation of a design framework that supports a high-level model with automatic transformations. The supported model provides an efficient high-level representation of massively parallel embedded systems. This representation hides the various cumbersome low-level technological specificities (e.g., hardware accelerators, processors) that developers usually have to cope with. The automatic transformations of the model target various abstraction levels and technologies as a support for a fast design space exploration. Our framewok aims at strongly contributing to increase the productivity of designers.

1.3 Outline of the Article

The overall organization of the article can be decomposed into four main parts, themselves composed of sections, as follows.

— *Preliminary: related works and introduction of a case study.* This part starts with Section 2 describing some important related works about the design of high-performance embedded systems. This section points out the major limitations of these works, which are addressed in the GASPARD framework. To understand how this is achieved, a typical massively parallel embedded system, corresponding to a Digital Signal Processing (DSP) system from the multimedia domain, is introduced in Section 3. It is considered along the article to illustrate our contribution.
— *Design concepts and refinement in* GASPARD. Here, Section 4 introduces the repetitive MoC on which rely the GASPARD modeling concepts presented in Section 5. More precisely, these two sections show how the different parts of a system are designed with GASPARD: functionalities and hardware architectures, their association, and their deployment with existing platform-specific components. The refinement of the high-level models obtained before is addressed in Section 6. Different transformation chains are implemented towards various analysis and simulation target technologies in GASPARD.
— *A design methodology.* Based on the previous design concepts and refinement facilities of GASPARD, we propose a multilevel design space exploration methodology, explained in Section 7. This methodology is illustrated on our running example (i.e., the case study) in Section 8 so as to explore the design space.
— *Concluding remarks.* In order to assess the contribution of the article, Section 9 discusses our proposition by highlighting its advantages and weakness for the design of massively parallel embedded systems. Finally, the conclusions are given in Section 10.

2. RELATED WORKS

We first present some high-performance programming models that deal with massively parallel systems in general. Then, we discuss methodological aspects through some outstanding paradigms: hardware-software codesign approaches for embedded systems and model-driven engineering. In both paradigms, abstract models and model refinements are central.

2.1 High-Performance Programming Models

The programming of high-performance systems has been extensively investigated through the last decades. Among the proposed solutions, we have already mentioned the parallel programming models offered in MPI and OpenMP.

More recent languages are StreamIt [Thies et al. 2002], Brook [Stanford Streaming Supercomputer Project 2009], and OpenCL [Khronos Group 2009]. They promote new parallel programming models and introduce the notion of stream, a flow of data, and the notion of kernel, a function that is applied on a stream in parallel.

Their main objective is to provide a model of computation that abstracts the new model of architecture provided by GPU (Graphics Processing Unit) in the frame of the GPGPU (General-Purpose computing On GPU) approach, or by multicore processors. Despite their indisputable advantage with respect to the previous low-level programming of these architectures, these languages are by nature limited to the targeted architecture they implicitly defined in their models. No definition of the architecture and no definition of the mapping of the computation (for example, the kernel or the stream) on an architecture can be expressed by the programmer. An implicit mapping

is defined and the compiler generates the code based on this mapping. Thus, similarly to MPI and OpenMP, they also have the inconvenience to be not well adapted for SoC design in which one needs to program specialized architectures.

An interesting high-level language is the data-parallel formalism ALPHA [Wilde 1994], which is very close to ARRAYOL. It manipulates polyhedra instead of arrays. This leads to different specification styles. ALPHA particularly suits for the specification of systolic architectures which are a specific case of massively parallel architectures. As a result, it does not offer a satisfactory solution to the design of other types of architecture models as it is needed here for SoCs.

2.2 Hardware/Software Codesign Approaches for Embedded Systems

There are several codesign methodologies for embedded systems that start from high-level designs from which the system implementation is produced after some automatic or manual refinements.

The first methodologies for hardware-software codesign of systems were based on system synthesis methods. They consist in transforming, through successive refinements, the original sequential specification into a concurrent one defining all the implementation details. This relies on the use of high-level languages. At the beginning, a system is represented only as a network of processes according to a MoC such as KPN (Kahn Process Network) [Kahn 1974] or SDF (Synchronous DataFlow) [Lee and Messerschmitt 1987]. It is then rewritten in languages such as C (for the software part) and in languages such as VHDL (for the hardware part). COSMOS [Ismail et al. 1994], Chinook [Chou et al. 1995], and Specsyn [Gajski et al. 1998] are among the first projects that proposed automated transformations from a high abstraction level to a lower one. However, one of their main drawbacks is that the translations from an abstraction level to a lower one are not complete and are very tedious to be manually achieved. These proposals were more about methodologies to guide the designer through the refinement process, but none of them addressed the usage of multiprocessor.

Further methodologies can also be mentioned such as SynDEx [Grandpierre and Sorel 2003], the Space Codesign framework [Chevalier et al. 2006], Ptolemy [Lee 2001], PeaCE [Ha et al. 2007], or HOPES [Ha 2007]. The SynDEx codesign methodology is referred to as *algorithm architecture adequation*. It covers the design steps from the specification of functionalities to the deployment on target multiprocessor architectures including specific integrated circuits. The system implementation code can be automatically generated. A main limitation of SynDEx is that it does not efficiently handle massively parallel architecture topologies or computation structures. The space codesign framework allows to model hardware-software systems at high abstraction level and to refine them until FPGA implementation. However, this framework does not manage factorized expression of parallelism. Thus, it does not scale for massively parallel systems. Unlike the previous two approaches, Ptolemy is rather devoted to the modeling, simulation, and design of embedded systems by integrating different models of computation (e.g., synchronous/reactive, continuous time, etc.) in order to deal with concurrency and time in a heterogeneous system description. The PeaCE approach is built upon Ptolemy to address the hardware-software codesign for multimedia embedded systems. Its design flow includes an interactive design space exploration framework based on a hardware-software cosimulation at both TLM and RTL levels, and an automatic code generation for hardware and software. The HOPES approach extends the PeaCE codesign methodology by mainly introducing a new abstraction layer, called Common Intermediate Code (CIC), within the code generation process. The CIC representation is independent from any communication architecture

and operation system. It offers enough flexibility to target code generation for different MPSoC architectures. Compared to the PeaCE and HOPES frameworks, GASPARD enables a more powerful modeling of parallelism in data-intensive computation by using MARTE RSM concepts instead of SDF variants. In addition, the other MARTE packages used in our framework offer a richer set of concepts for architecture description. The automatic model transformations and code generation achieved in GASPARD also include intermediate representations that are retargetable. They can be seen as a generalization of the idea of using the CIC representation in HOPES.

The approaches adopted in Koski [Kangas et al. 2006], SystemCoDesigner (SCD) [Keinert et al. 2009], and System-on-Chip Environment (SCE) [Dömer et al. 2008] are also close to our work in that they propose frameworks for SoC design. They provide supports for automated design space exploration and synthesis. The GASPARD framework also aims at an easy design space exploration by making it possible to automatically produce input code for existing analysis, simulation, and performance evaluation tools. Koski uses a UML-based modeling (more precisely the TUT Profile, see next section) of system specification expressed as KPNs. It supports automatic back annotations and modification of system models based on simulation and performance evaluation results. In Gaspard, modifications of system models after exploration are currently only manually possible. The SCD framework considers as input the SystemC behavioral model of a system, expressed in an annotated actor-oriented style. It automatically extracts the useful information from this model and performs multi-objective design space exploration. The exploration process is entirely automated within the synthesis of SoC implementations in SCD. In the SCE environment (successor of Specsyn), a comprehensive and automated refinement approach addresses the design of MPSoCs from abstract specifications to implementation. At each design step, specific decisions can be manually entered in order to automatically generate refined system models. These models are described in SpecC and the decisions are taken based on design exploration via simulation and performance evaluation. As a global remark, we can observe here that neither Koski nor SCD nor SCE considers a high-level and expressive modeling formalism, such as the MARTE profile, to adequately deal with massive parallelism.

The Platform-Based Design (PBD) approach [Sangiovanni-Vincentelli and Martin 2001] is another typical example of methodology. Its main idea is to facilitate the design task by enabling successive refinements of high-level specifications of system functionality and architecture with reusable components so as to rapidly meet the implementation requirements of the system. The principles of PBD are found in frameworks such as Metropolis [Balarin et al. 2003] of Berkeley, VCC [Martin and Salefski 1998] of Cadence, the Artemis workbench [Pimentel 2008], and CoFluent studio [Calvez 1993]. Except the former, which considers several MoCs, each of these frameworks adopts a particular MoC and provides the designers with a library of domain-specific components for platform instantiation. While PBD helps to shorten the time-to-market and to reuse verification, it may potentially reduce the flexibility of design, since the space of choices is limited to the available system components. Furthermore, in the case of SoCs, which are composed of heterogeneous components that are often developed with dedicated tools (e.g., an ARM processor with the ARM environment, a hardware accelerator with a high-level synthesis tool), it is highly desirable to have a single design model that covers the whole system description. But, most of PBD approaches combine different design models that capture various aspects of a system. For instance, the Artemis workbench that is devoted to multimedia domain considers Simulink representations for functionality description and KPN MOC for architecture description.

2.3 Model-Driven Engineering for Embedded Systems

Another interesting design paradigm that is very close to PBD is the already mentioned MDE approach. It has been increasingly adopted for the design of embedded systems in general [Schmidt 2006]. The basic modeling formalism is the general-purpose language UML, which offers attractive graphical specification concepts. Because of its generality, UML is refined by the notion of *profile* to address domain-specific problems.

There are currently several profiles for the design of embedded systems such as SysML [Object Management Group Inc. 2006], UML SPT [Object Management Group Inc. 2005], UML-RT [Selic 1998], TUT Profile [Kangas et al. 2006], ACCOR/UML [Lanusse et al. 1998], and Embedded UML [Martin et al. 2001]. Among these profiles, only SysML and UML SPT have been standardized by the Object Management Group (OMG). SysML is a general-purpose modeling language for system design, while UML SPT is dedicated to the modeling of time, schedulability, and performance-related aspects of real-time systems. The UML-RT and ACCOR/UML profiles are less rich in terms of concepts than UML SPT. The embedded UML profile has been defined within VCC as an experimental proposal that goes beyond the real-time field. It also includes aspects from the hardware-software codesign field. This last field is mainly taken into account in the TUT profile, which defines concepts allowing one to model applications, platforms, and their mapping. Among the few profiles that specifically focus on SoC modeling, we mention UML4SystemC [Riccobene et al. 2005] and the OMG UML4SoC profile [Hasegawa 2004]. They offer an abstraction of the RTL level. This abstraction accelerates the simulation of the software and hardware parts of a system. Because all these profiles may potentially overlap, significant standardization efforts have been recently realized by the OMG, resulting in the single unified and effective MARTE standard profile on which GASPARD relies.

While these profiles allow to specify a system with high-level models, refinements from such models towards low-level models have to be achieved. Typically, from the specification of an embedded system with a profile, one would like to generate executable implementations of the system. For instance, the UML4SoC and UML4SystemC profiles manipulate concepts that are very close to the implementation level, and are thus mappable to the targeted languages, for example, SystemC or SystemVerilog. This reduces the flexibility when considering other targets because the overall mapping has to be redefined. Some alternative propositions use specific notations instead of using profiles, defining an entirely executable model semantics [Alanen et al. 2006; Nguyen et al. 2004; Riccobene et al. 2006]. Such expressive notations allow one to define models with sufficient information so that the specified system can be completely generated. However, here also, the code is directly generated from the specifications, without any intermediary representation. The same is observed in the VHDL code generation from UML [Björklund and Lilius 2002; Coyle and Thornton 2005], where the code is obtained directly by mapping the UML concepts to the VHDL syntax. While these approaches rely on an abstraction of the system by using high-level models, they only exploit a little of its benefits by directly being dependent on target languages or abstraction levels.

From the previously presented profiles, only a subset has been proposed as de facto industrial standards normalized by OMG. They are implemented in commercialized modeling tools such as Rhapsody and TAU of IBM/Telelogic, MagicDraw of No Magic, and RSA of IBM. Among these specific profiles, SysML and MARTE appear as the most popular. UML4SoC and UML SPT are also intended for industrial usage but they have less success than the previous two. All the other profiles are rather academic experimental prototypes with dedicated tools, with a very limited impact in practice.

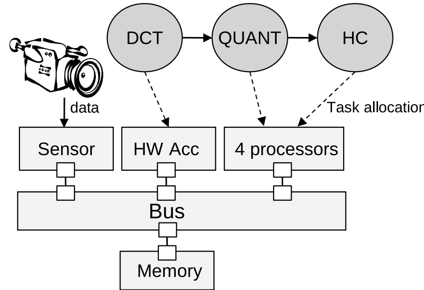

Fig. 2. Informal representation of the intra part of the H.263 encoder.

3. A TYPICAL DSP EMBEDDED SYSTEM

A typical embedded system dedicated to signal processing application is presented in this section. Our objective is to illustrate the design challenges faced by developers in terms of architecture topology, application task mapping, and hardware and software component reuse. We consider the H.263 video codec standard [Cote et al. 1998; International Telecommunication Union (ITU) 2005]. We focus on the *intra part* of the encoder, which performs regular and massively parallel computations. Most of the examples used in this article are extracted from this application.

The intra part of the H.263 encoder application sketched in Figure 2 is composed of three tasks: the Discrete Cosine Transform (DCT), the quantization (QUANT), and the Huffman Coding (HC).

The encoding algorithm deals with the input stream of QCIF frames by dividing each frame into macroblocks. Since the processing of each macroblock can be done independently from the others, this data structure confers to the H.263 algorithm a potential parallelism in terms of data computation. This is an embarrassingly parallel algorithm that is executable in a regular way.

The execution of the intra part of the H.263 encoder in an embedded system implies several constraints. For instance, depending on the frame rate (e.g., 10 or 30 frames per second) to achieve, the encoder is executed more or less efficiently. Its execution architecture thus has to exploit the potential parallelism of the algorithm in order to reach satisfactory performances. Figure 2 represents a possible architecture including four processors, a hardware accelerator, a memory, and a bus. The workload of QUANT and HC tasks is equitably shared between the four processors. The DCT task is mapped onto the hardware accelerator, which is an electronic circuit allowing a maximal parallelization of the computation needed to execute an application.

The architecture and the mapping sketched in this figure are only one possible implementation solution. For instance, an architecture only composed of processors as computing resources could be an interesting alternative. A high number of processors may provide a sufficient computing power enabling one to avoid the usage of a hardware accelerator. The choice among these various solutions mostly relies on the knowledge of embedded system designers. However, the current design methodologies based on designer's know-how are no more adapted to address the exploration of wide design spaces.

To solve this bottleneck, we believe that it is useful to manage the three challenges mentioned in Section 1. It is necessary to exploit the potential parallelism available in applications when designing the architecture and the mapping of the former on the latter. High-abstraction-level representations of embedded systems enable one to deal with their complexity. A methodology in which implementations are generated from such representations enables to rapidly explore wide design spaces without the

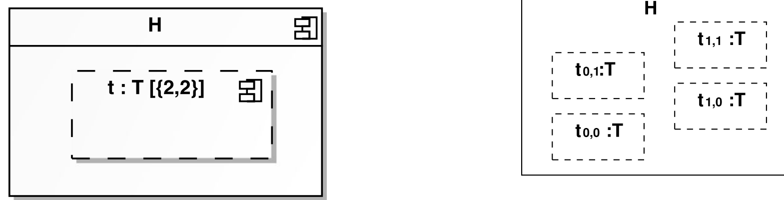

Fig. 3. Factorized (left) and unrolled (right) regular structure.

drawbacks of manual implementations. These aspects are key points for the design of tomorrow's high-performance embedded systems.

4. THE REPETITIVE MODEL OF COMPUTATION

Modeling high-performance embedded systems requires concepts that allow us to describe the regularity of a system structure in a factorized way. Such concepts provide the designer with a way to efficiently and explicitly express models with a high number of identical components. The repetitive structures are the basis of the repetitive MoC originally inspired by ArrayOL, which is dedicated to multidimensional signal processing [Demeure and Del Gallo 1998].

ArrayOL is a domain-specific language enabling the specification of intensive signal processing applications manipulating large amounts of data, in the form of multidimensional (and possibly infinite) arrays. The potential parallelism available in an application is captured via data dependencies. The specification of an application consists of two views: *global* and *local* models. A global model is a directed acyclic graph where nodes represent tasks and edges represent data dependencies labeled by multidimensional arrays. Therefore, it permits to represent task parallelism. The local model rather describes data parallelism in an application by using the notion of *repetition* explained in the sequel.

Since a few years, we have been significantly extending ArrayOL so as to enable the design of high-performance embedded systems based on the model-driven engineering approach [Cuccuru et al. 2005]. This extension led to the repetitive MoC [Boulet 2008; Glitia et al. 2009], which enables the description of regular system structures or topologies beyond the application level (as it is the case with ArrayOL): hardware architectural topologies, hardware-software association, data allocation in memory. A benefit of such a description is that it is specified in the same factorized way and at a high abstraction level.

The following description of the repetitive MoC is independent from application and architecture considerations. Among the presented notions, the *shape*, *reshape* and *interrepetition* concepts are part of our extension of ArrayOL.

4.1 Repetition and Shape

A structural element T is replicated into several structural elements. The *repetition space* is defined by a vector. The number of iterations is determined by multiplying the coordinates of this vector. The left-hand side of Figure 3 represents a repetitive structure. H is a hierarchical structure, the dashed box t:T is a *repeated structural element*. {2,2} corresponds to the repetition space of this structural element. The right-hand side corresponds to the unrolled (i.e., explicitly enumerated) equivalent representation. Each dashed box $t_{i,j}$:T, referred to as a *repetition*, corresponds to a structural element associated with a given iteration in the repetition space.

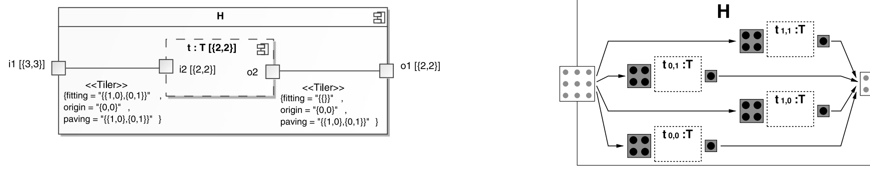

Fig. 4. Tilers express multiple link topologies in a factorized way.

In other words, a repeated structural element t is a set of repetitions $\{t_k\}$, each corresponding to the same functionality and structure. The number of repetitions is the cardinal of the repetition space associated with t.

Structural elements handle information in the form of arrays materialized by ports. The shape of a port defines the shape of the corresponding array and specifies how the information are structured. Dependencies are defined between the arrays handled by different structural elements. A repeated structural element manipulates subsets of such arrays, which are referred to as *patterns*. The dependencies between arrays and patterns are expressed with the three repetitive link topologies, Tiler, Reshape, and InterRepetition, detailed next.

4.2 Data Dependencies and Communications

4.2.1 Tiler. A Tiler connector expresses how a multidimensional array is tiled by patterns. For this purpose, it connects an array to the patterns of a repeated structural element, as illustrated in the left-hand side of Figure 4. In this example, i1 and o1 are the ports of the H structural element. They represent the multidimensional arrays of H. The ports i2 and o2 represent the patterns of the t:T repeated structural element.

Boulet [2008] gives a formal description of tilers. In this article, we adopt an intuitive approach to explain how tilers work. The right-hand part of Figure 4 is an equivalent but not factorized expression of the Tilers illustrated on the left-hand side. All the repetitions (*e.g.* $t_{0,0}$:T, $t_{0,1}$:T) of the repeated structural element are made explicit. Each one manipulates its own patterns: the pattern corresponding to i2 is a bidimensional array whereas the one corresponding to o2 is a scalar. The patterns are built according to the arrays of the H structural element, as indicated via arrows. The 2×2-patterns i2 are constructed from elements contained in the 3×3-array i1. Symmetrically, the o2 scalars are used to build the 2×2-array o1.

The way patterns are built from an array depends on a *tiling operation* which requires the origin, paving, and fitting attributes. The origin vector specifies the origin of the reference pattern in the array. The paving and fitting matrices, respectively, specify how an array is covered by patterns and how the patterns are constructed with array elements. In the remainder, for shortcut, we denote the origin vector, the paving, and fitting matrices by $\mathbf{o}$, P, and F, respectively.

Figure 5 precisely describes the dependencies between patterns and array elements. The vector r denotes iteration steps in the repetition space: each $r = \binom{i}{j}$ in Figure 5 identifies the corresponding $t_{i,j}$:T in Figure 4. For each iteration, the elements highlighted in the 3×3-array i1 are used to build the corresponding 2×2-pattern i2. For example, the four elements on the bottom left-hand side of the array are used to build the pattern of the $t_{0,0}$:T structural element. Thus, the scalar element in the middle of the array is used to build each of the four patterns. In the same way, the right-hand side of Figure 5 illustrates the dependencies between o2 and o1.

The paving and fitting operations considered in a repetition and illustrated in Figure 5 are formally defined as follows.

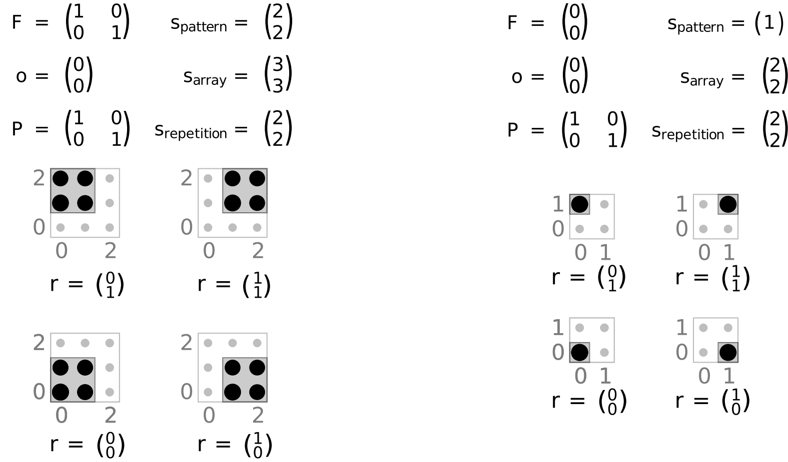

Fig. 5. Dependencies between patterns and array elements: the left-hand side and the right-hand side show how the input and output patterns are respectively built.

For each repetition instance, one needs to specify the reference element $\mathrm{ref}_{\mathbf{r}}$ of its associated pattern. The reference pattern (of the initial repetition instance) is determined by considering the origin vector of a tiler. The patterns associated with the other repetition instances are built relatively to the reference pattern. As previously, their coordinates are built as a linear combination of the vectors of the paving matrix as follows. We have

$$\forall\, \mathbf{r}, \mathbf{0} \le \mathbf{r} < \mathbf{s}_{\mathrm{repetition}},\ \mathrm{ref}_{\mathbf{r}} = \mathbf{o} + P \times \mathbf{r} \bmod \mathbf{s}_{\mathrm{array}}, \tag{1}$$

where $\mathbf{s}_{\mathrm{repetition}}$ is the shape of the repetition space, $\mathbf{o}$ is the origin vector, P the paving matrix, and $\mathbf{s}_{\mathrm{array}}$ the shape of the array.

Now, given a pattern, let its reference element, ref, denote the initial point from which all its other elements are determined. The coordinates of these elements, represented by $\mathbf{e_i}$, are built as the sum of the coordinates of the reference element and a linear combination of the vectors in the fitting matrix, the whole modulo the size of the array (since arrays may be toroidal) as follows. We have

$$\forall\, \mathbf{i}, \mathbf{0} \le \mathbf{i} < \mathbf{s}_{\mathrm{pattern}},\ \mathbf{e_i} = \mathrm{ref} + F \times \mathbf{i} \bmod \mathbf{s}_{\mathrm{array}}, \tag{2}$$

where $\mathbf{s}_{\mathrm{pattern}}$ is the shape of patterns, $\mathbf{s}_{\mathrm{array}}$ is the shape of the array, and F is the fitting matrix.

According to its attributes (origin vector, paving, and fitting matrices), a Tiler expresses dependencies between a M-dimensional array and N-dimensional patterns. These dependencies are not limited to compact and parallel to axis patterns (e.g., rectangular shaped patterns). Dependencies are usually expressed via indexes in languages. It is the case in ALPHA [Wilde 1994]. The explicit manipulation of indexes in specifications is tedious and error-prone: the implied complexity dramatically increases with the number of dimensions and the shape of patterns. Tilers avoid these drawbacks.

4.2.2 Reshape and InterRepetition. A Reshape enables us to express complex link topologies in which the elements of a multidimensional array are redistributed in another multidimensional array. For this purpose, a Reshape connector links two ports of structural elements included in the same hierarchical structural element (as opposed to the Tiler which links a port of a repeated element to those of a hierarchical one). In fact, a

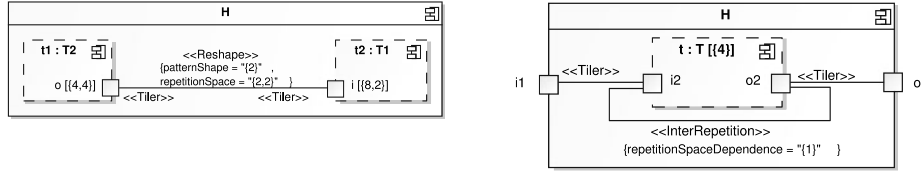

Fig. 6. Representation of the Reshape (left) and the InterRepetition (right) dependencies.

Reshape is a combination of two Tilers. It enables the factorization of complex dependencies between two arrays. The left-hand side of Figure 6 represents a Reshape connector that expresses dependencies between the o and i ports.

The InterRepetition connector links a pattern of a repeated structural element with another pattern of the same repeated structural element, as illustrated in the right-hand part of Figure 6. It therefore induces a partial order on the execution of these repetitions. The usefulness of the InterRepetition link topology is illustrated later in the article.

The concepts manipulated in our repetitive MoC have been first defined in a profile [Cuccuru et al. 2005] for the GASPARD framework. Today, they are fully adopted in the MARTE standard profile via its RSM package. A complete description of all these concepts can be found at Object Management Group [2007a].

In the following section, we demonstrate the suitability of MARTE for the modeling of high-performance embedded systems in GASPARD.

5. MODELING CONCEPTS

This section presents the concepts provided by MARTE for the description of intensive signal processing applications, hardware architecture, and the mapping of the former on the latter. Additional information corresponding to implementation details are provided through a deployment mechanism. All these concepts are illustrated in the modeling of an embedded system dedicated to the execution of the H.263 encoder.

5.1 Application Functionality

5.1.1 Components. The targeted application functionality generally consists of data-intensive computation algorithms as in the H.263 encoder application. There are basically three kinds of task components that are used to specify application functionality. An *elementary* component defines a task as an atomic function. A *repetitive* component expresses a data-parallel task in an algorithm. Finally, a *hierarchical* component enables to specify complex functionalities in a modular way; in particular, task parallelism is described using such a component. All these components are modeled by combining the concepts provided in the MARTE GCM, VSL, and RSM packages used in GASPARD (see Figure 1).

We introduce next the H.263 encoder application designed with the MARTE profile.

5.1.2 Modeling the H.263 Encoder. In Figure 7, the H263Encoder component corresponds to the task that reads a QCIF frame in order to produce a compressed frame. A QCIF frame is decomposed into the three arrays materialized by the ports lumin, cbin, and crin (one port for the luminance and two ports for the chrominance). The compressed frame produced is decomposed into arrays materialized by the ports mbout and size, denoting respectively the compressed data and the compression level.

In the application part of a MARTE model, a repetition space on a task expresses data parallelism. On Figure 7, the multiplicity {11,9} associated with the H263mb

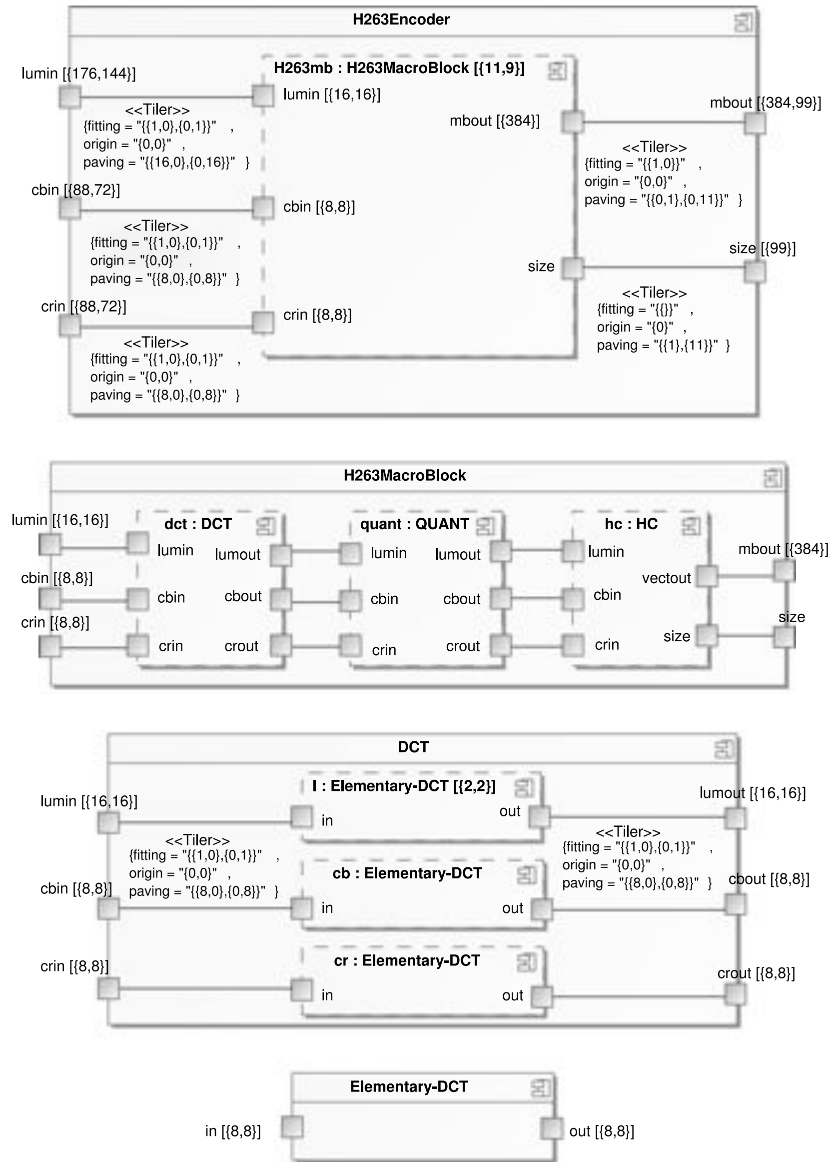

Fig. 7. The main application components of the H.263 encoder.

instance of the H263MacroBlock component (noted as H263mb:H263MacroBlock) denotes such a data-parallel task. Each repetition of the task H263mb consumes three input patterns (lumin, crin, and cbin) and produces two output patterns (mbout and size). A pattern corresponds to a subset of the overall frame. Each iteration step within the task repetition space consumes and produces patterns. Their construction relies on the data dependencies expressed via the Tiler connectors.

The hierarchical component H263MacroBlock is formed of three tasks, which correspond to the three steps of the H.263 encoder algorithm: DCT, QUANT, and HC. The DCT component contains both data and task parallelisms. The aim of this component is

to apply the DCT transformation algorithm to the luminance and chrominance macroblocks. For this purpose, the Elementary-DCT task is instantiated. It consumes a 8×8 array of pixels in order to produce a 8×8 array of spatial frequency coefficients. Since this task is elementary, its behavior is provided according to additional information included in the deployment described later on.

At the top level of the application model, the received video streams are encoded according to a global repetition around the H263Encoder component. This repetition infinitely applies the encoding algorithm to the successive video frames.

5.2 Hardware Architecture

5.2.1 Architecture Modeling. Compared to UML, the MARTE profile allows one to describe hardware characteristics in a more precise way [Taha et al. 2007]. The hardware architecture is described in a structural way. In GASPARD, only the HW_Logical sub-part of the HRM package in MARTE is used. This subpackage describes typical hardware components (e.g., HwRAM,HwProcessor,HwASIC,HwBus), their nonfunctional properties (e.g., operating frequency, power consumption). More generally, it allows one to represent any architecture topology.

Similarly to the application modeling, the hardware architecture is also modeled using the repetitive concepts of the MARTE RSM package. This is very useful when describing a grid of processing elements, such as the Tile64 architecture of Tilera [Tilera Corporation 2009]: the model is kept factorized.

In MARTE, some hardware architecture modeling concepts [Taha et al. 2007] are inspired by the initial UML-based propositions in GASPARD for system architecture modeling.

5.2.2 Architecture Models. Figure 8 shows a model of an MPSoC architecture. The main component of this architecture, called QuadriPro, is composed of four processing units specified using the repetition concept, four RAM modules specified as a 2×2-grid, a ROM, and a crossbar that interconnects these components together. The Reshape connector (whose attributes are not displayed in this figure for the sake of simplicity) between the dbus port of MultiProcessingUnit and the master port of the crossbar specifies how processors are connected to the communication network. Similarly, the Reshape connectors relating the slave port of the crossbar to the bus ports of memories specify how the repetition of the ports are linked to the ports of the memories. Repeated memory components correspond to several memory banks accessible from the same address space, with a linearly increasing base address for each bank.

An important remark is that the MARTE concepts inherited by GASPARD for hardware architecture description enable to model any topology (e.g., graph topology). Among these topologies, regular ones are those which fully take advantage of the efficient expression of the parallelism offered by the repetitive concepts.

5.3 Allocation

5.3.1 Mapping Concepts. Since hardware architecture and application functionality are independently modeled, the *allocation* (also referred to as *association*) is in charge of expressing the mapping of the application onto the architecture. More precisely, the Allocate stereotype provided in the MARTE Alloc package allows one to specify the mapping of the computations on processing resources and the mapping of data on memory units. In order to define allocation on models containing repetitive structures, a notion of *distribution* is also provided in MARTE. The Distribute stereotype proposes a way to express regular distributions from an array of task (respectively, data) to an array of processors (respectively, memory units). It expresses, in a compact way, for

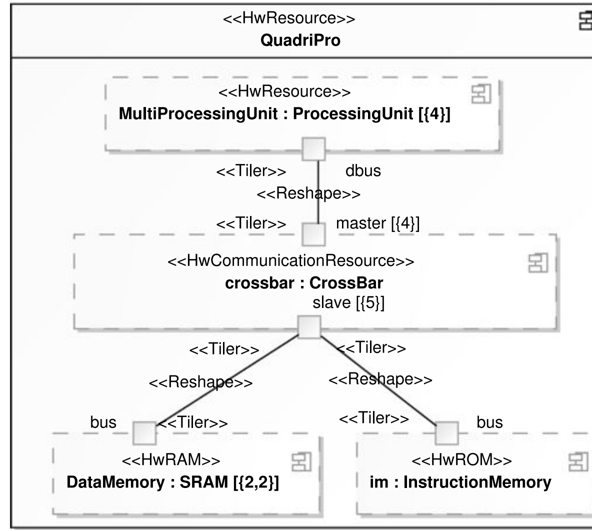

Fig. 8. Architecture of a shared memory MPSoC.

each repetition of the task which processor executes it [Boulet et al. 2007]. Similarly, it defines for each element of a data array which memory bank contains it.

5.3.2 Application Mapping on Architecture. The components on the top and bottom of Figure 9 correspond to two different hierarchical levels of the H.263 encoder application model. In the middle of Figure 9, the main component of a hardware architecture, as seen previously in Figure 8, is shown. The whole H263mb task is mapped on the processors of the QuadriPro hardware architecture via the Allocate links. The Distribute stereotype (whose attributes are not shown in the figure) specifies precisely the distribution of the 99 repetitions of H263mb onto the four processors. Informally explained, the distribution linearizes the 11×9 repetitions and maps them on the array of the processors via a modulo function. Consequently, 25 repetitions out of the 99 ones are assigned to each processor, except the fourth one which receives only 24 repetitions.

Similarly, all the data handled in the H263Encoder application are mapped onto the DataMemory. Since the modeled architecture is shared memory, the distribution does not require to closely fit the distribution of the tasks. A simple one, allocating a quarter of each data array to each memory unit, is selected. Moreover, the data array allocation can be more precise. For example, the luminance and the chrominance parts of the overall frames can be distributed independently by directly allocating the lumin, cbin, and crin ports of the H263mb task onto memories.

5.4 Deployment Using IPs

5.4.1 Deployment Concepts. In order to generate an entire system from a high-level specification, all implementation details of every elementary component have to be specified. IPs are used to ease component reuse. They correspond to specific implementations of a given functionality, either hardware or software. For instance, different IPs can be provided for a given application component and may correspond to either an optimized version for a specific processor or a version compliant with a given language.

Although the notion of deployment is present in UML, the SoC design has special needs not fulfilled by this notion. Hence, GASPARD extends the MARTE profile with

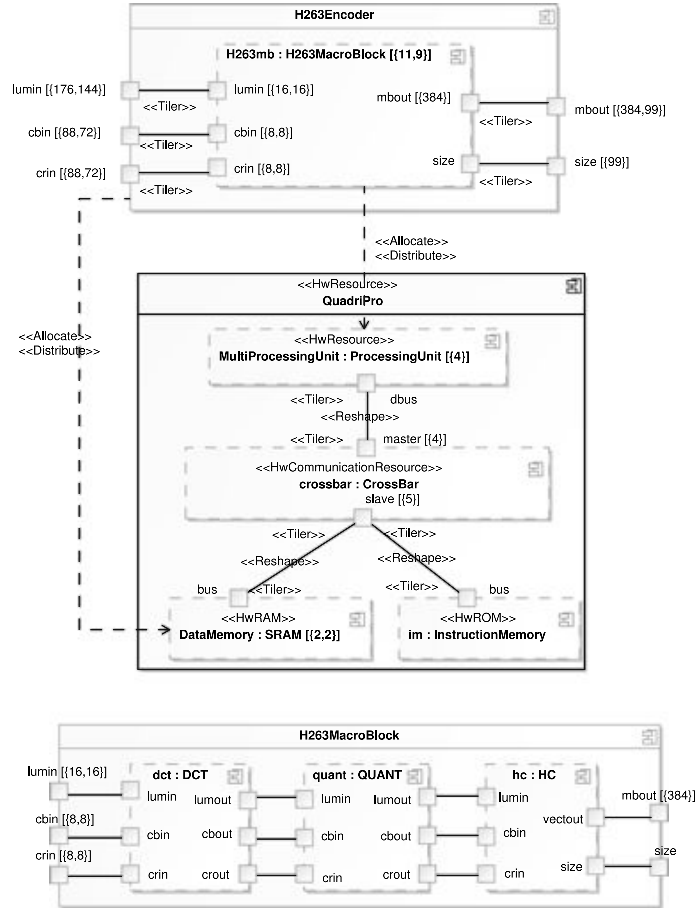

Fig. 9. Mapping onto a processor-based architecture.

the Deploypackage (see Figure 1) to allow deploying elementary components with IPs. For this purpose, we have introduced the concept of VirtualIP as a generic wrapper that captures various implementations of a given elementary component (either software or hardware), independently from the usage context. A VirtualIP is implemented by one or several IPs, each one used to define a specific implementation at a given abstraction level and in a given language. Finally, the concept of CodeFile is used to specify, for a given IP, the file corresponding to the source-code and its required compilation options. The used IP is selected by the SoC designer by linking it to the elementary component through the Implements dependency. Some IPs provided by the SoC industry can be parametrized. These parameters are specified using the Characteristic concept.

5.4.2 Deployment Example. Figure 10 represents the deployment of the Elementary-DCT elementary component onto the VirtualDCT VirtualIP. The VirtualDCT may be implemented by two different SoftwareIP. The DCT-VHDL IP corresponds to VHDL code for an implementation at the RTL level. The DCT-C IP corresponds to C code that is executed on a processor. A CodeFile is associated with each of these two IPs. Thus, designers can choose among the two implementation options.

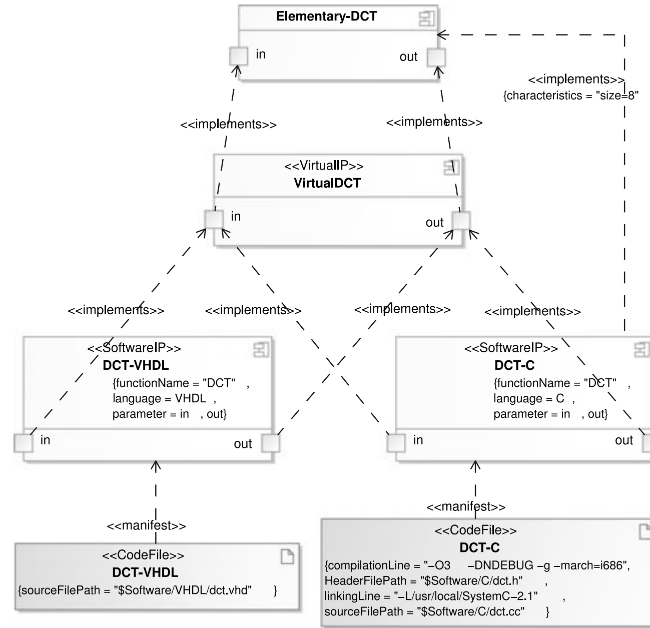

Fig. 10. Deployment of the Elementary-DCT task.

In this figure, the implements dependency from DCT-C to Elementary-DCT specifies that DCT-C is selected as the implementation of Elementary-DCT. This dependency also allows one to parametrize the IP by setting the size characteristic to 8. Here, 8 corresponds to the array size needed to execute the DCT functionality. The other implements dependencies indicate how the ports of VirtualDCT (respectively, DCT-VHDL and DCT-C) are linked to the ports of Elementary-DCT (respectively, VirtualDCT).

Such a deployment mechanism allows one to easily modify the selected IP or its corresponding characteristics, without any modification of the application or the architecture.

6. MODEL REFINEMENT

Model-driven engineering relies on two concepts: metamodel and transformation. A *metamodel* precisely expresses the semantics of the concepts and the relations between them in order to state the characteristics of a valid model: a model conforms to a metamodel. A *model transformation* defines how to produce a model conforming to a target metamodel from another model conforming to a source metamodel. The target metamodel is usually more specialized than the source one and manipulates concepts closer to the code. A successive application of several model transformations yields a *transformation chain* and allows code generation from high-level models.

In this section, we present the refinement of the previous high-level models, via the transformation chains implemented in GASPARD. These chains have already been separately described and published in some articles [Le Beux et al. 2007; Piel et al. 2008b; Taillard et al. 2008a; Yu et al. 2008]. In this article, we describe Gaspard as a multipurpose framework underlying the unity and the complementarity between the chains. Figure 11 presents an overview of these chains in GASPARD. The top of this figure corresponds to the high-level modeling concepts described previously with the MARTE profile: Application, Architecture, Association, and Deployment. Then, different

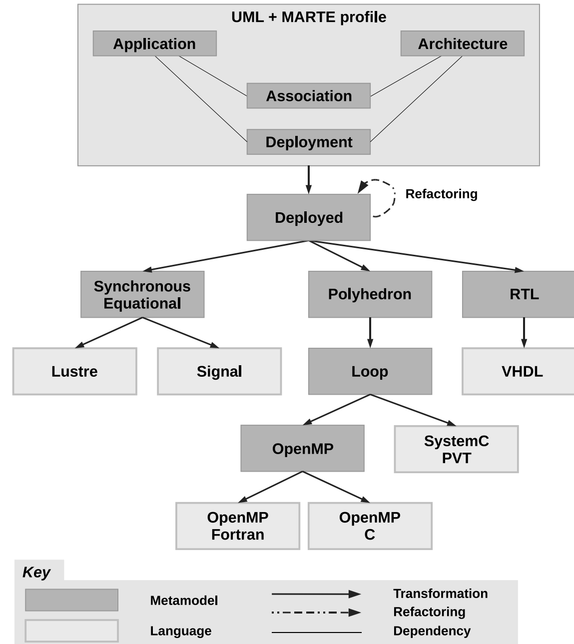

Fig. 11. GASPARD transformation chains.

transformation chains are defined, which transform high-level models towards specific technologies, identified in the bottom side of Figure 11.

From the functional viewpoint, the pieces of code generated in synchronous language, SystemC, OpenMP Fortran/C, and VHDL, corresponding to the same high-level system model have strictly the same behavior. In other words, the same inputs always lead to the same outputs. This is ensured by the fact that all functional dependencies specified at the highest modeling level in GASPARD are entirely preserved by each transformation chain. For instance, for the functional correctness of the code generation in synchronous languages, the reader may refer to Gamatié et al. [2008] for details. It is the case for all the other chains. From the nonfunctional viewpoint, this does not hold. Typically, the execution/simulation duration of the different pieces of code may differ from one target technology to another.

6.1 GASPARD Transformation Chains

The first step is the same in each transformation chain of GASPARD. It consists in targeting the Deployed metamodel. This metamodel corresponds to an intrinsic definition of the concepts, identified in the MARTE profile extended with the deployment concepts without taking into account the UML concepts on which they rely. Models that conform to this metamodel can be modified using refactoring functions [Glitia and Boulet 2008]. These functions consist of classical loop transformations (i.e., fusion, collapse, etc.). They modify, create, or delete the hierarchy according to the regular parallelism. They can thus be used to adapt the design provided by the user while keeping the functionality unchanged, for example in order to optimize the execution. The other intermediate metamodels are described in the corresponding transformation chains.

6.1.1 Transformation Chain towards Synchronous Languages. The goal of this transformation chain is to provide GASPARD designers with the possibility to address functional

correctness issues. Indeed, MARTE does not provide the semantics necessary to formally validate the designed system. Synchronous languages [Benveniste et al. 2003] are well-known for their formal aspects and their richness in terms of tools for validation and verification at the functional level. Safe array assignment and causality in data dependencies [Gamatié et al. 2008] are examples of functional properties that are checked in an application designed with MARTE.

The *synchronous equational* metamodel, proposed in the Synchronous package of GASPARD (see Figure 1), gathers synchronous programming concepts and is generic enough to be transformed in different synchronous dataflow languages [Yu et al. 2008]: Lustre, Lucid Synchrone, and Signal. More precisely, it enables to describe systems of equations over variables called signals. Such equations specify relations between the values and the logical instants at which the signals occur. The generated code mainly serves to check functional properties of the applications described with MARTE.

6.1.2 Transformation Chain towards SystemC. This chain leads to the generation of a SystemC-based simulated embedded system [Piel et al. 2008b]. The targeted abstraction level is PVT (Programmer View Timed). Furthermore, in intensive signal processing applications, instruction and synchronization transfers via the interconnection network are largely smaller than the data transfers [Ben Atitallah et al. 2007a]. For this reason, in our implementation, only data transfers are simulated. Consequently to these two points, at the PVT level, instead of using an instruction set simulator for the software execution, the internal architecture of the processors is replaced by application parts. Thus, only data accesses (i.e., the low-level functionalities such as read/write), or the idle mode are really implemented in the processors. The GASPARD implementation of a system at the PVT level is useful for a fast functional analysis, an execution time estimation, the bus contention monitoring, and the identification of the most consuming application parts.

The *polyhedron metamodel*, proposed in the Polyhedron package of GASPARD (see Figure 1), presents the association mechanism from the architecture viewpoint. Instead of having concepts to indicate the placement of tasks and data arrays as in the deployed metamodel (i.e., allocation and distribution), in the Polyhedron metamodel, the data arrays are contained into memories (in which they are assigned specific address-ranges) and the tasks are contained in the processors. In order to faithfully represent the repetitions of these distributed elements, the *polyhedron* mathematical concept is used. A polyhedron is a set of linear equations and inequalities from which an efficient execution of loop iterations on processors is made possible by using the CLooG tool [Bastoul 2004].

The *loop metamodel* is the second intermediate metamodel, defined in the Loop package of GASPARD (see Figure 1). It is very close to the Polyhedron metamodel. The unique difference is the representation of the task repetition. Instead of using a polyhedron, the repetition is represented by a *LoopStatement*. It corresponds to the pseudocode structure that, for a given processor index, goes all over the repetition index of the associated tasks [Piel et al. 2008a]. This metamodel is the last one in the transformation chain towards SystemC. The SystemC code is directly generated from this metamodel.

The execution of the application follows an execution model that is common to all simulation levels, so that the same behavior is exactly simulated and implemented at each level. In particular, this execution model defines how tasks exchange data. Each connection between tasks is implemented as a FIFO supporting multiple readers and multiple writers. This also enables us to handle the synchronization between the tasks. The FIFOs are implemented via channels interconnected to modules that simulate buses and crossbars, so that the communications can be satisfied. A significant

amount of work can be involved concerning the routing of data at hardware level, particularly when considering distributed memory, which requires the use of specific protocols for data transmission via intermediate nodes. But, in the current status of our framework implementation, only models with shared memory are supported.

An annotated timed model is defined and integrated in our PVT level. It facilitates accurate performance estimation for design space exploration. For this purpose, several timing concepts are introduced: a local timer associated with each processor, and a time parameter passed to the communication methods. The proposed timed model requires to preliminarily identify for each hardware component the relevant time-consuming activities, for example, data hit/miss for the caches or read/write access for the shared memory. The estimation of the execution time requires assigning a delay value to each activity. In our approach, execution delays are either measured from a physical characterization of a hardware component or from an analytical model at a low abstraction level. This activity is performed only once for each hardware component. The results are then reported in the deployment model.

6.1.3 Transformation Chain towards OpenMP Languages. This chain shares several metamodels with the previous one. However, while the SystemC chain generates code directly from the Loop metamodel, an additional intermediate metamodel is used to generate procedural languages. The OpenMP metamodel of the Procedural package in GASPARD (see Figure 1), is an abstraction of such languages. In this chain, the Single Program Multiple Data (SPMD) execution model is considered: each processor has the same code fragment, parametrized with the processor number. This chain has been used to implement a conjugate gradient solving Maxwell equation and numerical computations on large matrices [Taillard et al. 2008a].

The *OpenMP metamodel* is inspired by the ANSI C and Fortran grammars and extended by OpenMP statements. The aim of this metamodel is to use the same model to represent Fortran and C code. Thus, from an OpenMP model, it is possible to generate OpenMP/Fortran or OpenMP/C. The generated code includes parallelism directives and control loops to distribute task repetitions over processors [Taillard et al. 2008b].

6.1.4 Transformation Chain towards VHDL. This transformation chain enables the design of hardware accelerators. An accelerator is generally dedicated to the execution of a specific application. Each accelerator must be customized separately without real possible reuse. This leads to long production delay and high design cost. Thanks to this chain, it is possible to automatically produce hardware accelerators that solve these two issues and reduce human intervention, which is often error-prone.

The generation of a hardware accelerator relies on the hardware-software partitioning specified in high-level MARTE models. More precisely, it depends on the hierarchical task (application model) that is allocated onto the hardware accelerator (architecture model). This hierarchical task and the lower ones are executed by the hardware accelerator, while the loops contained in the upper hierarchical levels are managed by a processor. The processor iterates on its corresponding repetition space: at each iteration, the processor launches an execution on the hardware accelerator. In this context, the processor is the master and the hardware accelerator is a slave.

In order to generate the hardware design for the tasks allocated on the accelerator, the VHDL transformation chain handles two types of hardware executions for the data parallelism. The hardware-sequential execution consumes few resources but provides a relatively long execution time, whereas the hardware-parallel execution consumes more resources but increases the performance. The hardware-software and the parallel-sequential hardware partitionings give the opportunity to customize the performance and/or the area cost of an accelerator.

The *RTL metamodel*, defined in the RTL package of GASPARD (see Figure 1), gathers the necessary concepts to describe hardware accelerators at the RTL (Register Transfer Level) level, which allows the hardware execution of applications. The RTL metamodel is independent from any Hardware Description Languages (HDL) such as VHDL [IEEE 1994] or Verilog [Thomas and Moorby 1998]. However, it is precise enough to enable the generation of synthesizable HDL code. Furthermore, we developed a process to estimate the consumed hardware resources (i.e., the area cost) and the execution time in terms of clock cycles of the hardware accelerators modeled in RTL models [Le Beux et al. 2008]. This estimation requires a preliminary step: each IP is individually synthesized with the usual synthesis tool in order to get its area cost and timing information. This step is achieved only once, and the results are reported to the considered IP models. The estimation of the other elementary elements, for example, register and multiplexers, is computed by using simple mathematical expressions during the transformation. The overall accelerator is then characterized following a bottom-up approach.

As a technical remark, we can note that when the Gaspard framework has been built, no available tool fully was compliant with QVT (i.e., Query/View/Transformation) [Object Management Group Inc. 2007b], the OMG standard for model transformation languages) and no transformation tool easily supported black-box call. As a result, most model transformations in the Gaspard framework are written using Java with EMF query. This allows us to specify the transformations while benefiting from the Java expressiveness. Since new QVT-compliant tools are now available, our transformations are being rewritten using QVTO [Borland 2007]. The code generation is written using Java templates because no standard is implemented.

6.2 GASPARD Toolset

The GASPARD tool-set is provided as an Eclipse plugin that allows users to define embedded systems and to explore the design space based on simulation, synthesis, and verification of automatically generated code. The entry point corresponding to the high-level specifications is a MARTE-compliant model. Such a model is specified by a user with UML modeling tools such as MagicDraw [No Magic 2007] or Papyrus [2007].

The user selects the transformation chain to be executed among the preceding chains. For instance, the screen-shot of Figure 12 corresponds to an execution of the transformation chain towards SystemC. The upper panel corresponds to the UML model. The lowest panel is the generated SystemC code. The other panels correspond to the intermediate models.

7. MULTILEVEL DESIGN SPACE EXPLORATION

A major goal of GASPARD is to rapidly design an embedded system that meets its requirements, in particular those related to performance and correctness. This goal is achieved by considering high-level models and the different technologies that are reached via the refinement chains provided in our framework. For that purpose, some precise ordered design steps should be respected. Based on these steps, we define a methodology dedicated to the multilevel design space exploration for high-performance embedded systems in GASPARD. This methodology relies on the refinement chains presented in Section 6 except the chain that targets OpenMP languages, which is specifically dedicated to high-performance computing for scientific applications.

Given a system to be designed, two main steps are identified: high-level model specification and low-level analyses. There are possible feedback loops from the low-level analyses to the high-level specification in order to enhance the design.

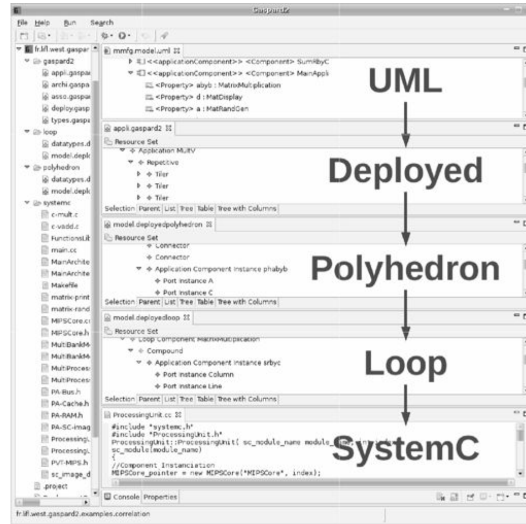

Fig. 12. Execution of the transformation chain towards SystemC.

7.1 High-Level Model Specification

The designer should exploit the expressive factorization mechanism provided by the RSM package. So, he or she specifies the potential parallelism available in the different system parts: functionality, hardware, and the corresponding associations. We distinguish three substeps that are combined in different ways.

Description of Application Functionality and Architecture. The user defines the application functionality in terms of MARTE concepts where the elementary components will be instantiated with the adequate IPs (e.g., see the H.263 encoder example in Section 5.4) on the one hand, and the hardware architecture (e.g., a quadriprocessor) on the other hand.

Allocation of Application Functionality onto Architecture. The user decides a strategy for task and data distribution on the defined hardware architecture. Data are allocated to memory units while tasks are allocated to processing units. Furthermore, a hardware-software partitioning may be specified.

IP Deployment. At this stage, the elementary components used in the system functionality and the architecture are deployed on IPs so as to target a given technology.

While it is obvious that the first substep is the starting point of the model specification, the other two substeps are applied according to two possible scenarios: either the deployment decisions are partial or complete. The former case enables to postpone a part of these decisions after the allocation of the system functionality on a given architecture. The latter case happens when the user does not necessarily need to wait for allocation to start some design verification or simulation.

7.2 Low-Level Analyses

From the high-level system models resulting from the previous step, we target three technologies for system analysis at different abstraction levels: functional level with synchronous languages, PVT level with SystemC, and RTL level with VHDL. These technologies are used in a complementary way to define a top-down multilevel

exploration approach. During each exploration substep, the results obtained from the analyses enable to redesign the considered high-level models for a better suitability with respect to the system requirements.

Formal Verification of Functional Properties with Synchronous Languages. For a system under design, the synchronous technology is first used to formally check the system correctness regarding the functional properties, that is, given any inputs, the system always computes the expected outputs of the implemented function. The architecture-related properties are not addressed here but only in the next exploration steps (see the following). Typically, possible errors in the specification of the algorithm defining the system functionality can be detected. In that case, the user has to go back to the high-level specification step in order to correct the embedded system model. Nonfunctional aspects related to architectural details are not addressed in this substep. The next two substeps are suitable for that purpose.

Simulation-Based Exploration at PVT Level with SystemC. Here, the design analysis takes into account both system functionality and architecture. The PVT abstraction level considered in SystemC is sufficiently high to keep the simulation fast enough, and to enable the test of several configurations within a limited time-frame. Such a test includes, for example, the variation of the number and the type of processors and memory, the modification of the communication network topology, etc. An illustration is given in Section 8 for the implementation of the H.263 encoder application.

Hardware-Accelerator-Based Exploration at RTL Level with VHDL. This substep is a suitable complement of the previous one in that it enables to explore further implementation alternatives by taking into account hardware accelerators in the system design. Typically, when the performance requirements of a system are not satisfied during the SystemC-based simulations, one can decide to synthesize some parts of the system functionality as hardware. This significantly increases the execution performance (see Section 8). Among the useful information that are obtained in order to select the best hardware implementation, we mention the execution time and the surface occupied by the accelerator.

The aforesaid three target technologies offer a very interesting basis for design space exploration. Based on the feedback information they provide, the high-level models of a system are modified so as to better meet the design requirements. Their corresponding target pieces of code can be then regenerated very rapidly and easily. This contributes to converge efficiently towards embedded systems that meet both correctness and performance requirements.

7.3 Exploration of Next-Generation Embedded Systems

The preceding methodology relies on the current implementation of the GASPARD framework, and particularly on its transformation chains. It will be very interesting to enhance the complementarity of the transformation chains targeting the PVT and the RTL levels. Indeed, supporting hardware accelerators at the PVT level and processor-based architectures at the RTL level would make possible the simulation of the whole system at each of these levels.

The PVT level enables fast simulations but yields approximate results. In the opposite, at the RTL level, the results are very precise while the simulation time is very long. The PVT level is thus more adapted to explore large design spaces whereas the RTL level should be used to explore a limited solution set. According to our methodology, the subset of the overall design space determined at the PVT level should be

explored at the RTL level. However, this set is still quite large. Thus, its effective exploration at the RTL level is very time consuming due to slow simulations. Henceforth, it is relevant to reduce the design space to be explored at the RTL level. For that, an intermediate subset of the design space resulting from the PVT level has to be introduced. Such a subset may be defined according to the simulations performed at an intermediate abstraction level. Indeed, at the Cycle-Accurate Byte-Accurate (CABA) abstraction level, simulations are performed faster than at the RTL level. The results obtained from these simulations are more precise than those obtained at the PVT level. Hence, the CABA level suitably complements the PVT and the RTL levels. The CABA level can be easily reached in GASPARD by introducing a new transformation chain, which is under development from the Loop metamodel.

By providing a methodology and a tool-set allowing the exploration of a very large design space via successive reductions until reaching a satisfactory solution, GASPARD would successfully face the main challenge of future embedded systems generations that have to perform even more complex applications onto even more potentially powerful architectures. None of existing MDE-based approaches for design exploration [Bakshi et al. 2001; do Nascimento et al. 2007] offers similar capabilities as GASPARD for high-performance embedded systems.

In the next section, we illustrate the exploration of different low-level implementations on our running example based on SystemC and VHDL programs generated from high-level specifications defined with the GASPARD modeling concepts.

8. IMPLEMENTATION RESULTS IN THE CASE STUDY

This section presents an experiment that evaluates the effectiveness of our design methodology and of the overall GASPARD framework. We focus on the design of an embedded system that suits for encoding a 30 frames per second video sequence. The corresponding H.263 video encoder application has been introduced in Section 3 and modeled in Section 5. We first check the correctness of its functional part using the synchronous technology. Then, we address the architecture exploration issues. We design a homogeneous processor-based architecture and explore the design space covered by such an architecture. The number of processors is modified in order to increase performance of the embedded system. The performances of different configurations are evaluated according to fast SystemC simulations performed at the PVT level. These evaluations give the performance limits of a processor-based architecture. We thus keep on exploring the design space by introducing a hardware accelerator. We allocate the most time-consuming part of the application on this accelerator. Different versions of this accelerator are automatically generated at the RTL level by using the VHDL transformation chain. Finally, the resulting embedded system is heterogeneous, since it uses both processors and a hardware accelerator.

8.1 High-Level Specification

According to our design space exploration methodology, the starting point is the high-level specification of the parts necessary to build the embedded system. The H.263 application model illustrated in Figure 7 corresponds to the system functionality. The targeted architecture is composed of four processors and a shared memory, as illustrated in Figure 8. The mapping of the application onto this architecture is the one illustrated in Figure 9. Finally, all elementary component models in the application functionality as well as the architecture component models (e.g., processors, buses) are linked to IPs thanks to the deployment, partially illustrated in Figure 10.

8.2 Formal Verification with Synchronous Languages

The H.263 encoder application must be formally verified before starting the design space exploration. For this purpose, Signal or Lustre code is generated by executing the transformation chain towards synchronous languages. With this chain, the data dependencies specified in the high-level application model are transformed into equations in the generated code [Gamatié et al. 2008]. Usual analysis tools (compilers and model-checkers) dedicated to synchronous languages check whether or not the specification of the H.263 encoder application satisfies some functional properties.

Among these properties, the single assignment and the absence of causality cycle are verified here. The single assignment property ensures that each data of the output frames is computed only once and is not overwritten by another data. The absence of causality cycle ensures that the H.263 encoder is free of any deadlock during its execution. Notice that the applicability of synchronous techniques for the formal verification quite depends on the size of considered models. In particular, because of the explicit enumeration of repetitions adopted during the synchronous code generation, the resulting programs can be huge. Consequently, either the code generation tools, for example, Eclipse, or the verification tools, for example, a compiler [Amagbegnon et al. 1995], may not scale due to memory limitation. A few experiments showed that from 4.10^4 repetitions in a model, the size of its corresponding synchronous program potentially becomes critical. In order to overcome such a scalability limitation, a possible solution consists of modular code generation, which allows the different subparts of a program to be addressed easier.

Here, the H.263 encoder is not concerned by any scalability problem. Since these properties are verified by the system, the design space is now explored at the PVT and the RTL abstraction levels successively, as advocated in our methodology.

8.3 Simulation and Evaluation of MPSoC at PVT Level

In this part, we evaluate how the transformation chain towards SystemC simulation of MPSoC at PVT level helps for the early design space exploration. Indeed, this chain allows a fast evaluation of the embedded system, for example, its execution time, its occupied physical surface, the possible communication contention.

The SystemC simulation code resulting from the transformation is executed so as to obtain the execution time. In order to increase the performance (i.e., to reduce the execution time), the design space is explored by increasing the number of processors. This consists in modifying two numbers in the high-level model specification: the shape of the MultiProcessingUnit processor and the shape associated to the master port of the crossbar communication resource. Then, the simulation source-code is automatically regenerated, and the simulation of the new configuration is executed afterwards. The algorithm complexity and the size of the obtained SystemC code is independent of the number of repeated components in a model (both for the application and for the hardware): only the value of the loop bounds representing the number of repetitions are modified. For instance, the transformation chain executed with 4-processors and 1024-processors architectures requires 3 seconds in both cases.

In order to compare the simulation at the PVT level, we have manually written a simulation code of the same system at the CABA abstraction level (Cycle-Accurate Byte-Accurate). The input data correspond to the frames of a standard file used for testing QCIF encoders (`akiyo_qcif`), which has a resolution of 176×144 pixels. The simulation results obtained from this code are supposed to be close to the results that could be observed at the RTL level. Our PVT simulation provides 15% to 30% underestimated results than this CABA hand-coding, but is faster to execute: in this experiment it was approximately 20 times faster. In spite of the underestimation

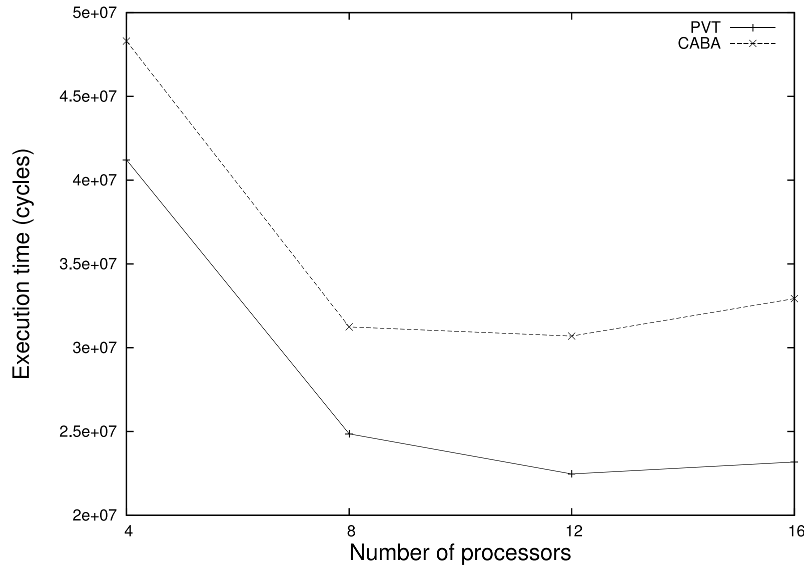

Fig. 13. Number of simulated cycles depending on the amount of processors, at the PVT and CABA abstraction levels.

noted on the PVT simulations, the relative order between the curves resulting from both simulations is coherent. This is the most interesting information needed at this high level of abstraction. The precise timing information of the best configuration are obtained at lower abstraction levels. Besides the CABA slowness, the difficulty to modify the simulation code at this low abstraction level considerably reduces the number of testable configurations in a given time. In addition to accelerate the design space exploration, the PVT code has the advantage to be automatically generated from high-level description UML models. This demonstrates the ability to generate fairly relevant simulations of the MPSoC from UML.

Figure 13 presents the number of simulated cycles used for the encoding of one QCIF frame when the architecture has 4, 8, 12, and 16 processors, at the PVT and CABA abstraction levels. In spite of the underestimation noted on the PVT simulations, the relative order between the curves resulting from both simulations is coherent. In particular, both simulations show that a 16-processors architecture is slower than a 12-processors one. This is explained by the parallel overhead, which includes data transfers and synchronizations between processors. The PVT simulation therefore enables us to easily find the optimal configuration (here, 12 processors) for a processor-based architecture. According to its results, 22.10^6 cycles are necessary to compute a frame on the 12-processors configuration. When this embedded system runs at 100 MHz frequency, 220 milliseconds are necessary to achieve this computation while a maximum of 33 milliseconds is tolerated for a 30 frames per second video sequence. This simulation thus demonstrates that even the most powerful processor-based architecture does not meet the performance requirements.

Figure 14 illustrates the parallel overhead and the processor usage in percentage per frame for the main tasks of the case study: DCT, Quant, and HC tasks. The profiling of the application is done by considering the timing information provided in the associated model. At the considered level (see Section 6.1.2), an application is expressed as blocks of code containing synchronizations, nested loops, and calls to the code of elementary components. The timing information allows to estimate the

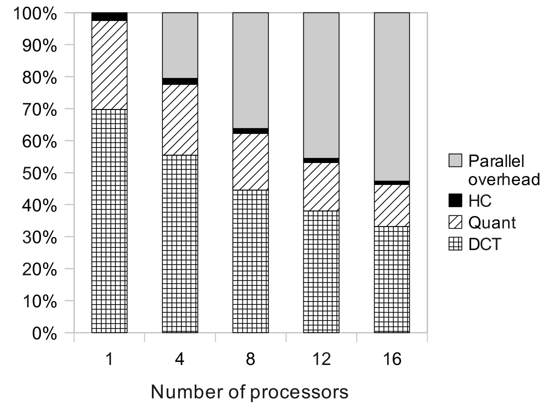

Fig. 14. Processor usage per frame for the intra part of the H.263 encoder.

duration of each of these parts according to the mapping of the application on multiple processors. The accurate timing comes from the fact that every operation external to the processors (e.g., read, write, idle) is synchronized with the rest of the simulation. Then, GASPARD reports for each component the time spent to execute it on processors, and the time spent in synchronizations (i.e., waiting for data, synchronization barriers) between all processors. By aggregating those numbers to the wanted granularity, the user obtains a result as illustrated in Figure 14. As shown in the depicted diagram, the most time-consuming task is the DCT task. From one to sixteen processors in the considered architecture, its processor usage varies from 70% to 33%. Whatever the number of processors, the DCT task consumes much more time than the HC and Quant tasks. This observation is compatible with other simulation results found in the literature [Ben Atitallah et al. 2006; Jang et al. 2000; Nguyen et al. 2000] about the H.263 encoder.

The DCT task is known to be efficiently executed with hardware accelerators. Indeed, using such a single but high-performance computing resource, the number of memory accesses decreases, and the number of contentions is also reduced. In the next part, we address the design exploration by taking into account an heterogeneous architecture that combines processors with a hardware accelerator.

8.4 Increasing Performance Using a Hardware Accelerator

In order to generate a hardware accelerator for the DCT part of the H.263 encoder, the previous UML model is modified. The application part of this UML model remains unchanged. The dct:DCT task is now allocated on a hardware accelerator, and the Elementary-DCT is deployed on the DCT-VHDL IP. For this latter, the implements dependency that links the DCT-C IP to the Elementary-DCT task in Figure 10, is simply moved from the DCT-C to the DCT-VHDL IP. Using the transformation chain towards VHDL, the code corresponding to the hardware accelerator is automatically generated.

8.4.1 Exploring the Design Space. The DCT part of the H.263 application is highly regular and has large repetition spaces in its multiple hierarchical levels. Such large repetition spaces allow us to fully exploit the existing partitioning in VHDL (i.e., hardware-software and parallel-sequential hardware). This results in several feasible implementations whose performances vary with the chosen partitioning. In order to accelerate the design space exploration, a quick but useful evaluation of the hardware accelerators' effectiveness is performed. For example the amount of consumed hardware resources (i.e., the area cost) and the execution time (in clock cycles) can be

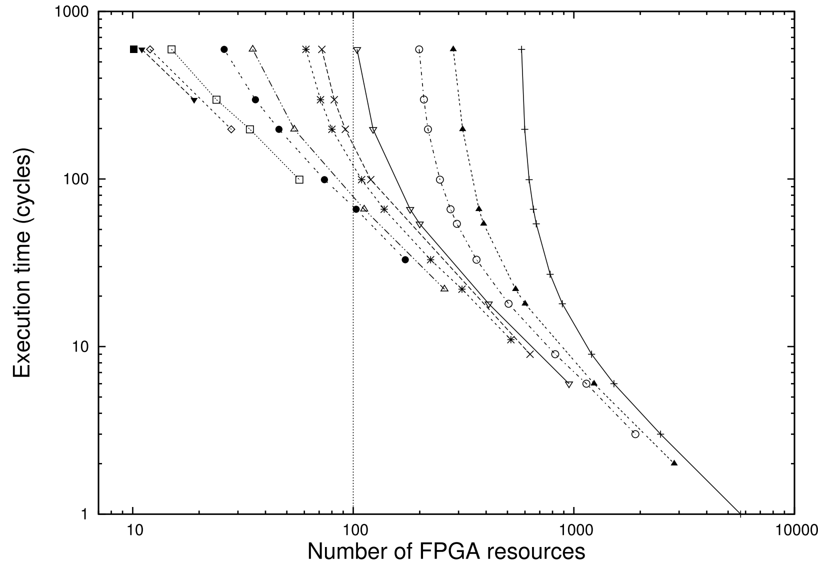

Fig. 15. The characteristics of different hardware accelerators automatically generated by GASPARD.

estimated from information contained in the RTL models. Each generated hardware accelerator is thus characterized by an execution time and an area on the material, necessary for its implementation. For the H.263 encoder DCT part, this leads to the curves shown in Figure 15 where the area is expressed in terms of FPGA resources. While these curves were obtained in a few seconds, the generation of the same result with usual synthesis tools from a corresponding VHDL code would require several hours. Hence, performing instantaneous estimation on the RTL models makes it possible to accelerate the design space exploration.

Each curve corresponds to a given hardware-software partitioning, and each point in a curve corresponds to a given sequential-parallel partitioning in the hardware execution. While the hardware-software partitioning is specified by the user in the UML model, the sequential-parallel hardware partitioning is automatically produced. Each point in the figure thus corresponds to a given implementation of the hardware accelerator but takes into account only the execution time of the hardware part. The execution time of the software part is proper to each curve.

The left-most curve is associated with the mapping of the lowest hierarchical level on the hardware accelerator: all the data parallelism is executed in software. The design space is then reduced to a single solution and the resulting embedded system does not benefit from hardware accelerator performances.

In the opposite, the right-most curve is associated with the mapping of the top hierarchical level of the DCT onto the hardware accelerator. The overall DCT is thus executed in hardware (the software part only deals with the management of this accelerator). The performances of the different implementations of the hardware accelerator increase with the number of used resources. However, this number is not realistic compared to the resources provided by nowadays FPGAs, even for the most time-consuming solution. Indeed, mapping the whole DCT in the hardware accelerator implies that the overall frame is stored on it in order to be managed on one shot. Such a configuration is thus very resource consuming. The design space is explored between these two extreme hardware-software partitionings. This leads to the intermediate curves.

8.4.2 Selection of the Appropriate Implementation. The selection of the appropriate implementation is driven by the constraints expressed with a maximum execution time and/or a maximum quantity of FPGA resources. When targeting a maximum of 100 resources (illustrated with the vertical line), all the solutions on the left-hand side of this vertical line satisfy this constraint. Among these solutions, the point corresponding to the lowest execution time denotes the most powerful implementation. The VHDL code generated from the right implementation and only this code is synthesized with usual synthesis tools.

8.5 Final Embedded System

GASPARD enables us to design an embedded system for an efficient execution of the H.263 encoder application. It includes both general-purpose processors and dedicated hardware accelerators. Such a heterogeneous architecture benefits from the complementarity of processors and hardware accelerator: the flexibility of software execution and the effectiveness of hardware execution. Following our methodology, we explored the design space at a high abstraction level using information resulting from the code generated by the transformation chains.

During the exploration, the number of processors has been first modified, then a hardware accelerator has been introduced and, finally, the most effective hardware-software partitioning has been selected. These modifications are easily done in UML, without the inconvenience of low-level implementations. Further configurations could be explored, for instance by modifying the number of hardware accelerators, the task allocation, the connexion topology in the hardware architecture, or the deployed IPs. The ease of modifications at high abstraction level coupled with the fast evaluations lead to a very powerful design space exploration framework.

9. DISCUSSIONS

In the current industrial practice, design space exploration for embedded systems is very expensive to achieve. For instance, the combination of MPSoC and hardware accelerator in the architecture would require two different expert teams. Our design framework aims at carrying out uniformly all the specifications on the same system model using MARTE. Therefore, the design space explorers do not have to deal with the interoperability of several tools, each one being dedicated to a given purpose specification of the system.

Considering the effectiveness of GASPARD for modeling massively parallel embedded systems, studies [Le Beux et al. 2008] have shown that the generated code with GASPARD is reasonably efficient compared to a manually written code: GASPARD provides the same performances as the manual approach but with a 10% additional cost in terms of resources. However, a very interesting point is that the automatic generation reduces an important amount of coding time. This is particularly valuable during design space exploration since new implementations are automatically regenerated from simple modifications in the high-level models [Ben Atitallah et al. 2007b]. Such modifications potentially concern the application functionalities, the hardware architecture, the mapping of both, or the IP deployment.

On the other hand, the repetitive MoC of GASPARD particularly offers an efficient and factorized high-level expression of parallelism that is inherent to massively parallel embedded systems performing regular computations or comprising various architectural topologies. This is an important benefit in comparison with closely related languages such as ALPHA [Wilde 1994]. Another benefit is that the design specification does not suffer from any scalability problem, which may happen when addressing massively parallel systems.

The used modeling concepts promote a separation of concerns during system design. Different aspects are distinguished: system functionality, hardware architecture, mapping, and deployment on target platforms. Contrarily to most platform-based design approaches [Sangiovanni-Vincentelli and Martin 2001], there is a clear separation between the platform-independent level of design and the deployment level at which the designer chooses a specific implementation platform. This favors more flexibility regarding the selection of the suitable implementations. Through the modeling of the H.263 encoder system, we have also shown how already defined concepts are reused thanks to the component-oriented design adopted in our approach. This is very important in order to meet the stringent time-to-market constraints imposed on designers.

The clear separation of concerns between the system functionality, the hardware architecture, and their mapping is also an important benefit of GASPARD-based design over the MPI and OpenMP programming. In the latter case, when switching from one system architecture to another, the programmer always needs to modify the functionality part in addition to the mapping, since the allocation directives are explicitly annotated in the functionality part. Thus, the code of the functionality part is not independent from the considered hardware architectures and this significantly reduces the reuse flexibility.

Our design framework relies on MDE techniques which reduce the development and maintainability efforts of high-performance computing design tools. At each abstraction level, the concepts and the relations between these concepts are precisely defined. This helps to manage the complexity of the tools by dividing the various compilation steps and by formally defining the concepts associated with each step. It also facilitates the development and the extensions of the tools. The transformation chains can benefit from two categories of extensions: fine-grain and coarse-grain. A fine-grain extension aims to integrate new concepts in metamodels, which better suit for a powerful design of other kinds of embedded systems in GASPARD. This is successfully realized by also introducing new rules in model transformations. A coarse-grain extension consists of a modification of the design flow itself for new purposes. For instance, one can decide to create a model transformation in order to generate Verilog code from the RTL metamodel or to create a RTL model from another metamodel (e.g., a metamodel used in another tool). Such flexibilities validate our point of view that advocates the development of tools using MDE: efforts done to develop a tool are capitalized. Therefore, thanks to MDE, GASPARD adapts easily to the changes of the fast-evolving SoC domain. Contrarily to UML4SoC [Hasegawa 2004] or UML4SystemC [Riccobene et al. 2005], GASPARD efficiently benefits from the preceding advantages of MDE. It abstracts more embedded system specifications and leverages the model transformations to target multiple technologies.

10. CONCLUSIONS

In this article, we have presented the GASPARD framework for the design of massively parallel embedded systems. It relies on our repetitive Model of Computation (MoC), which offers a powerful and factorized representation of parallelism in both system functionality and architecture. In GASPARD, high-level specifications of a system are defined with the MARTE standard profile. The resulting models are automatically refined into low-level implementations that are addressed with various technologies for design space exploration: formal verification, simulation, and hardware synthesis. These refinements are achieved by using Model-Driven Engineering, which enables to clearly define intermediate abstraction levels and to reach them via transformations.

As an answer to the design challenges mentioned in the Introduction section, GASPARD offers several advantages: an efficient high-level representation of parallelism, a separation of concerns, a reusability, and a unique expressive modeling formalism. All

these features strongly contribute to increase the productivity of designers. We believe that GASPARD adequately responds to the needs of next-generation high-performance embedded systems.

The whole framework is supported by user-friendly UML editors for system modeling and the Eclipse environment for model refinements. Thanks to the complementary of the different transformation chains that work at different abstraction levels, a user is able to design embedded systems that meet performance requirements as illustrated on the case study.

Currently, the transformation chain towards SystemC only manages software execution on a processor-based architecture at the PVT abstraction level, contrarily to the VHDL one which handles hardware execution on a hardware accelerator at the RTL level. As future work, we plan to enhance the complementarity of both PVT and RTL abstraction levels in GASPARD by partially merging the corresponding transformation chains. Thus, we should be able to simulate and analyze the overall embedded system at each of both abstraction levels. This enhancement provides additional information on the behavior and the performance. Depending on the design requirements, a designer could focus on the most relevant information for which the exploration could be partially automated using, for instance, some heuristics.

These evolutions will be integrated in GASPARD using MDE so as to keep on taking advantage of its features. For this purpose, some useful notions to be considered are kinds of traceability in models transformation and composition in metamodels extensions.

ACKNOWLEDGMENTS

We would like to gratefully thank all members of the DaRT research team at LIFL/INRIA who actively participated in the development of GASPARD and contributed to the present work. We are also grateful to the anonymous reviewers whose comments and suggestions significantly improved the previous version of this article.

REFERENCES

ALANEN, M., LILIUS, J., PORRES, I., TRUSCAN, D., OLIVER, I., AND SANDSTROM, K. 2006. Design method support for domain specific SoC design. In *Proceedings of the 4th Workshop on Model-Based Development of Computer-Based Systems and 3rd International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MBD-MOMPES'06)*. IEEE Computer Society, Los Alamitos, CA, 25–32.

AMAGBEGNON, P., BESNARD, L., AND GUERNIC, P. L. 1995. Implementation of the data-flow synchronous language signal. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'95)*. ACM Press, New York, 163–173.

BAKSHI, A., PRASANNA, V. K., AND LEDECZI, A. 2001. Milan: A model based integrated simulation framework for design of embedded systems. *SIGPLAN Not. 36,* 8, 82–93.

BALARIN, F., WATANABE, Y., HSIEH, H., LAVAGNO, L., PASSERONE, C., AND SANGIOVANNI-VINCENTELLI, A. L. 2003. Metropolis: An integrated electronic system design environment. *IEEE Comput. 36,* 4, 45–52.

BASTOUL, C. 2004. Code generation in the polyhedral model is easier than you think. In *Proceedings of the 13th IEEE International Conference on Parallel Architecture and Compilation Techniques (PACT'04)*. 7–16.

BEN ATITALLAH, A., KADIONIK, P., GHOZZI, F., NOUEL, P., MASMOUDI, N., AND LEVI, H. 2006. Hw/sw codesign of the h. 263 video coder. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE'06)*. 783–787.

BEN ATITALLAH, R., NIAR, S., MEFTALI, S., AND DEKEYSER, J.-L. 2007a. An MPSoC performance estimation framework using transaction level modeling. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. 525–533.

Ben Atitallah, R., Piel, E., Niar, S., Marquet, P., and Dekeyser, J.-L. 2007b. Multilevel MP-SoC simulation using an MDE approach. In *Proceedings of the IEEE International SoC Conference (SoCC'07)*. 197–200.

Benveniste, A., Caspi, P., Edwards, S., Halbwachs, N., Guernic, P. L., and de Simone, R. 2003. Synchronous languages twelve years later. *Proc. IEEE 91*, 64–83.

Björklund, D. and Lilius, J. 2002. From UML behavioral descriptions to efficient synthesizable VHDL. In *Proceedings of the 20th IEEE Norchip Conference*.

Borland. 2007. Operational QVT language. http://www.eclipse.org/m2m/qvto/doc.

Boulet, P. 2008. Formal semantics of Array-OL, a domain specific language for intensive multidimensional signal processing. INRIA Res. rep. RR-6467, INRIA. March.

Boulet, P., Marquet, P., Piel, E., and Taillard, J. 2007. Repetitive allocation modeling with MARTE. In *Proceedings of the Forum on Specification and Design Languages (FDL'07)*. 280–285.

Calvez, J.-P. 1993. *Embedded Real-Time Systems. A Specification and Design Methodology*. Willey, New York.

Chevalier, J., de Nanclas, M., Filion, L., Benny, O., Rondonneau, M., Bois, G., and Aboulhamid, E. 2006. A systemc refinement methodology for embedded software. *Des. Test Comput. IEEE 23,* 2, 148–158.

Chou, P., Ortega, R., and Borriello, G. 1995. The chinook hardware/software co-synthesis system. In *Proceedings of the IEEE International Symposium on System Synthesis*. Vol. 0. IEEE Computer Society, Los Alamitos, CA, 22–27.

Cote, G., Erol, B., Gallant, M., and Kossentini, F. 1998. H.263+: video coding at low bit rates. *IEEE Trans. Circ. Syst. Video Technol. 8,* 7, 849–866.

Coyle, F. P. and Thornton, M. A. 2005. From UML to HDL: A model driven architectural approach to hardware-software co-design. In *Proceedings of the Information Systems: New Generations Conference (ISNG)*. 88–93.

Cuccuru, A., Dekeyser, J.-L., Marquet, P., and Boulet, P. 2005. Towards UML 2 extensions for compact modeling of regular complex topologies. In *Proceedings of the ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML'05)*. 445–459.

Dekeyser, J.-L., Gamatié, A., Etien, A., Atitallah, R. B., and Boulet, P. 2008. Using the UML profile for MARTE to MPSoC co-design. In *Proceedings of the 1st International Conference on Embedded Systems & Critical Applications (ICESCA'08)*.

Demeure, A. and Del Gallo, Y. 1998. An array approach for signal processing design. In *Proceedings of Sophia-Antipolis Conference on Micro-Electronics (SAME'98)*. System-on-Chip Session.

do Nascimento, F. A. M., Oliveira, M. F. S., and Wagner, F. R. 2007. Modes: Embedded systems design methodology and tools based on mde. In *Proceedings of the 4th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES'07)*. IEEE Computer Society, Los Alamitos, CA, 67–76.

Dömer, R., Gerstlauer, A., Peng, J., Shin, D., Cai, L., Yu, H., Abdi, S., and Gajski, D. D. 2008. System-on-Chip environment: A SpecC-based framework for heterogeneous MPSoC design. *EURASIP J. Embed. Syst. 2008*, 1–13.

Gajski, D., Vahid, F., Narayan, S., and Gong, J. 1998. Specsyn: An environment supporting the specify-explorerefine paradigm for hardware/software system design. *IEEE Trans. VLSI Syst. 6,* 1, 84–100.

Gamatié, A., Rutten, E., Yu, H., Boulet, P., and Dekeyser, J.-L. 2008. Synchronous modeling and analysis of data intensive applications. *EURASIP J. Embed. Syst.* Article ID 561863, 22 pages.

Glitia, C. and Boulet, P. 2008. High level loop transformations for systematic signal processing embedded applications. In *Proceedings of the International Symposium on Systems, Architectures, MOdeling, and Simulation (SAMOSVIII)*. 187–196.

Glitia, C., Dumont, P., and Boulet, P. 2009. Array-OL with delays, a domain specific specification language for multidimensional intensive signal processing. *Multidimen. Syst. Signal Proces. J*.

Grandpierre, T. and Sorel, Y. 2003. From algorithm and architecture specifications to automatic generation of distributed real-time executives: A seamless flow of graphs transformations. In *Proceedings of the Formal Methods and Models for Codesign Conference (MEMOCODE'03)*. IEEE Computer Society, Los Alamitos, CA, 123–.

Ha, S. 2007. Model-Based programming environment of embedded software for MPSoC. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'07)*. IEEE Computer Society, Los Alamitos, CA, 330–335.

Ha, S., Kim, S., Lee, C., Yi, Y., Kwon, S., and Joo, Y.-P. 2007. PeaCE: A hardware-software codesign environment for multimedia embedded systems. *ACM Trans. Des. Autom. Electron. Syst. 12,* 3, 1–25.

HASEGAWA, T. 2004. An introduction to the UML for SoC forum in Japan. In *Proceedings of the 1st UML for SoC Workshop at DAC (UML-SoC'04)*.

IEEE. 1994. *Std 1076-1993 IEEE Standard VHDL Language, Reference Manual - Description*. http://standards.ieee.org/reading/ieee/std_public/description/dasc/1076 -1993_desc.html.

INTERNATIONAL TELECOMMUNICATION UNION (ITU). 2005. Recommendation H.263, video coding for low bit rate communication. http://www.itu.int/rec/T-REC-H.263-200501-I/en.

ISMAIL, T. B., ABID, M., AND JERRAYA, A. 1994. COSMOS: A codesign approach for communicating systems. In *Proceedings of the 3rd International Workshop on Hardware/Software Co-Design (CODES'94)*. IEEE Computer Society Press, Los Alamitos, CA, 17–24.

JANG, S., KIM, S., LEE, J., CHOI, G., AND RA, J. 2000. Hardware-Software co-implementation of a H.263 video codec. *IEEE Trans. Consum. Electron. 46,* 1, 191–200.

KAHN, G. 1974. The semantics of a simple language for parallel programming. In *Proceedings of the IFIP Congress on Information Processing*. J. L. Rosenfeld Ed., IFIP, 471–475.

KANGAS, T., KUKKALA, P., ORSILA, H., SALMINEN, E., HÄNNIKÄINEN, M., HÄMÄLÄINEN, T. D., RIIHIMÄKI, J., AND KUUSILINNA, K. 2006. UML-Based multiprocessor SoC design framework. *ACM Trans. Embed. Comput. Syst. 5,* 2, 281–320.

KEINERT, J., STREUBÜHR, M., SCHLICHTER, T., FALK, J., GLADIGAU, J., HAUBELT, C., TEICH, J., AND MEREDITH, M. 2009. SystemCoDesigner—An automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications. *ACM Trans. Des. Autom. Electron. Syst. 14,* 1, 1–23.

KHRONOS GROUP. 2009. OpenCL – The open standard for parallel programming of heterogeneous systems. http://www.khronos.org/opencl.

LANUSSE, P., GÉRARD, S., AND TERRIER, F. 1998. Real-Time modeling with UML : The ACCORD approach. In *Proceedings of the International Workshop on UML: Beyond the Notation*. 319–335.

LE BEUX, S., MARQUET, P., AND DEKEYSER, J.-L. 2007. A design flow to map parallel applications onto FPGAs. In *Proceedings of the 17th IEEE International Conference on Field Programmable Logic and Applications (FPL'07)*. 605–608.

LE BEUX, S., MARQUET, P., AND DEKEYSER, J.-L. 2008. Model driven engineering benefits for high level synthesis. INRIA Res. rep. 6615, inria.

LEE, E. A. 2001. *Overview of the Ptolemy Project*. University of California, Berkeley.

LEE, E. A. AND MESSERSCHMITT, D. G. 1987. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput. 36*, 24–35.

MARTIN, G., LAVAGNO, L., AND LOUIS-GUERIN, J. 2001. Embedded UML: A merger of real-time UML and co-design. In *Proceedings of the 9th International Symposium on Hardware/Software Codesign (CODES'01)*. 23–28.

MARTIN, G. AND SALEFSKI, B. 1998. Methodology and technology for design of communications and multimedia products via system-level ip integration. In *Proceedings of DATE'98 Designers' Forum*. IEEE Computer Society, Los Alamitos, CA, 11–18.

MESSAGE PASSING INTERFACE FORUM. 2009. MPI documents. http://www.mpi-forum.org/docs/docs.html.

NGUYEN, K. D., SUN, Z., THIAGARAJAN, P. S., AND WONG, W.-F. 2004. Model-Driven SoC design via executable UML to SystemC. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*. IEEE Computer Society, Los Alamitos, CA, 459–468.

NGUYEN, T., ZAKHOR, A., AND YELICK, K. 2000. Performance analysis of an H.263 video encoder for VIRAM. In *Proceedings of the International Conference on Image Processing*. 98–101.

NO MAGIC. 2007. MagicDraw. http://www.magicdraw.com/.

OBJECT MANAGEMENT GROUP INC. 2005. (UML) Profile for schedulability, performance, and time version 1.1. http://www.omg.org/technology/documents/formal/schedulability.htm.

OBJECT MANAGEMENT GROUP INC. 2006. Final adopted OMG SysML specification. http://www.omg.org/cgi-bin/doc?ptc/06-0504.

OBJECT MANAGEMENT GROUP. 2007a. A UML profile for MARTE. http://www.omgmarte.org.

OBJECT MANAGEMENT GROUP INC. 2007b. MOF Query/Views/Transformations. http://www.omg.org/docs/ptc/07-07-07.pdf. OMG paper.

OPENMP ARCHITECTURE REVIEW BOARD. 2009. The OpenMP API specification for parallel programming. http://openmp.org.

PAPYRUS. 2007. Papyrus UML web site. http://www.papyrusuml.org/.

PIEL, E., BEN ATITALLAH, R., MARQUET, P., MEFTALI, S., NIAR, S., ETIEN, A., DEKEYSER, J.-L., AND BOULET, P. 2008a. Gaspard2: From MARTE to SystemC simulation. In *Proceedings of the Workshop on Modeling and Analyzis of Real-Time and Embedded Systems* (with the MARTE UML profile at DATE'08). 23–28.

PIEL, E., MARQUET, P., AND DEKEYSER, J.-L. 2008b. *Generative and Transformational Techniques in Software Engineering II: Revised Papers*. Springer-Verlag, Chapter Model transformations for the compilation of multi-processor systems-on-chip, 459–473.

PIMENTEL, A. D. 2008. The Artemis workbench for system-level performance evaluation of embedded systems. *Int. J. Embed. Syst. 3,* 3, 181–196.

RICCOBENE, E., SCANDURRA, P., ROSTI, A., AND BOCCHIO, S. 2005. A UML 2.0 profile for SystemC: Toward high-level SoC design. In *Proceedings of the 5th ACM International Conference on Embedded Software (EMSOFT'05)*. ACM, New York, 138–141.

RICCOBENE, E., SCANDURRA, P., ROSTI, A., AND BOCCHIO, S. 2006. A model-driven design environment for embedded systems. In *Proceedings of the 43rd Annual Conference on Design Automation (DAC'06)*. ACM, New York, 915–918.

SANGIOVANNI-VINCENTELLI, A. 2007. Quo vadis, SLD? Reasoning about the trends and challenges of system level design. *Proc. IEEE 95,* 3, 467–506.

SANGIOVANNI-VINCENTELLI, A. AND MARTIN, G. 2001. Platform-Based design and software design methodology for embedded systems. *IEEE Des. Test Comput. 18,* 6, 23–33.

SCHMIDT, D. C. 2006. Model-Driven engineering. *IEEE Comput. 39,* 2, 41–47.

SELIC, B. 1998. Using uml for modeling complex real-time systems. In *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES'98)*. Springer, 250–260.

STANFORD STREAMING SUPERCOMPUTER PROJECT. 2009. Merrimac - Brook page. http://merrimac.stanford.edu/brook.

TAHA, S., RADERMACHER, A., GERARD, S., AND DEKEYSER, J.-L. 2007. An open framework for detailed hardware modeling. In *Proceedings of the 2nd IEEE International Symposium on Industrial Embedded Systems*. 118–125.

TAILLARD, J., GUYOMARC'H, F., AND DEKEYSER, J.-L. 2008a. A graphical framework for high performance computing using an MDE approach. In *Proceedings of the 16th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*. 165–173.

TAILLARD, J., GUYOMARC'H, F., AND DEKEYSER, J.-L. 2008b. OpenMP code generation based on an model driven engineering approach. In *Proceedings of the High Performance Computing & Simulation Conference (HPCS'08)*. 165–173.

THIES, W., KARCZMAREK, M., AND AMARASINGHE, S. 2002. StreamIt: A language for streaming applications. In *Proceedings of the 11th International Conference on Compiler Construction (CC'02)* (at ETAPS'02). Lecture Notes in Computer Science, vol. 2304/2002. Springer, 49–84.

THOMAS, D. AND MOORBY, P. 1998. *The Verilog Hardware Description Language* 4th Ed. Kluwer Academic Publishers.

TILERA CORPORATION. 2009. TILE64 processor family. http://www.tilera.com/products/processors.php.

WILDE, D. K. 1994. The ALPHA language. Tech. rep. 827, IRISA, France.

YU, H., GAMATIÉ, A., RUTTEN, E., AND DEKEYSER, J.-L. 2008. *Embedded Systems Specification and Design Languages, Selected Papers from FDL 2007*. Springer, Chapter Model transformations from a data parallel formalism towards synchronous languages, 183–198.

Received December 2008; revised September 2009; accepted November 2009