# A Model based design flow for Dynamic Reconfigurable FPGAs

Imran Rafiq Quadri, Samy Meftali, Jean-Luc Dekeyser

# A Model based design flow for Dynamic Reconfigurable FPGAs

Imran Rafiq Quadri, Samy Meftali and Jean-Luc Dekeyser,
INRIA LILLE NORD EUROPE - LIFL - University of Lille - CNRS, Lille, France
{Imran.Quadri, Samy.Meftali, Jean-Luc.Dekeyser}@lifl.fr

***Abstract*— As System-on-Chip (SoC) based embedded systems have become a de-facto industry standard, their overall design complexity has increased exponentially in recent years, necessitating the introduction of new seamless methodologies and tools to handle the SoC co-design aspects. This paper presents a novel SoC co-design methodology based on Model Driven Engineering and the MARTE (Modeling and Analysis of Real-Time and Embedded Systems) standard, permitting us to raise the abstraction levels and allows to model fine grain reconfigurable architectures such as FPGAs. Extensions of this methodology have enabled us to integrate new features such as Partial Dynamic Reconfiguration supported by Modern FPGAs. The overall objective is to carry out system modeling at a high abstraction level expressed in a graphical language like UML (Unified Modeling Language) and afterwards transformation of these models, automatically generate the necessary code for FPGA synthesis.**

## I. Introduction

Since the early 2000s, System-on-Chips (SoCs) have emerged as a new methodology for designing embedded systems in order to target data parallel intensive processing (DIP) applications. While rapid evolution in SoC technology permits to increase computation power, by doubling the number of integrated transistors on chip approximately every two years, the targeted application domains: such as multimedia video codecs, software-defined radio and radar/sonar detection systems are becoming more sophisticated and resource consuming. However the gap between hardware and software evolution is rapidly increasing due to issues such as reduction of product life cycles, increase in design time and budget limitations. System reliability and verification are also the main hurdles facing the SoC industry and are directly affected by the design complexity. An important challenge is to find efficient design methodologies which raise the design abstraction levels to reduce overall complexity while effectively handling issues such as accurate expression of system parallelism.

For SoC conception, currently High Level Synthesis (HLS) approaches are utilized: the behavioral description of the system is refined into an accurate register-transfer level (RTL) design for SoC implementation. An effective HLS flow must be *adaptable* to cope with the rapid hardware/software evolution and *maintainable* by the tool designers. The underlying low level implementation details are hidden from users and their automatic generation reduces time to market and fabrication costs as compared to hand written HDL (Hardware Description Languages) based implementations. However in reality, the abstraction level of the user-side tools is usually not elevated enough to be totally independent from low level implementations. Each particular implementation of the system (application/architecture) requires a particular specification which is usually in SystemC [1] or a similar language resulting in several disadvantages. Immediate recognition of system information such as related to hierarchy, data parallelism and dependencies is not possible; differentiation between different concepts is a daunting task in a textual description and makes modifications complex and time consuming.

Model Driven Engineering [2] (MDE) is an emerging domain and can be seen as a *High Level Design Flow* for SoC and an effective solution for resolving the above mentioned issues. The advantage of MDE is that the complete system (application and architecture) is modeled at a high specification level allowing several abstraction stages, thus a system can be viewed globally or from a specific point of view of the system allowing to separate the system model into parts according to relations between system concepts defined at different abstraction stages. This *Separation of Views* (SoV) allows a designer to focus on a domain aspect related to an abstraction stage thus permitting a transition from solution space to problem space. MDE's UML (Unified Modeling Language) graphical nature increases system comprehensibility and allows users to provide high abstraction level descriptions of systems in order to easily identify the internal concepts (task/data parallelism, data dependencies and hierarchy). The graphical nature of these specifications allows for their reuse, modification, maintenance and extension.

Partial Dynamic Reconfiguration [3] (PDR) is an emerging feature supported by modern FPGAs allowing specific regions of an FPGA to be reconfigured on the fly, hence introducing the notion of *virtual hardware* with the advantage of time-sharing the available hardware resources for executing multiple tasks. PDR allows task swapping depending upon application needs, hardware limitations and Quality-of-Service (QoS) requirements (power consumption, performance, execution time etc.) Currently only Xilinx FPGAs fully support this feature.

MARTE [4] (Modeling and Analysis of Real-Time and Embedded Systems) is an industry standard proposal of the Object Management Group (OMG) for model-driven development of embedded systems. It add capabilities to UML allowing to model software, hardware and their relations, along with added extensions (for e.g. performance and scheduling analysis). This standard although while rich in concepts, unfortunately lacks tools to move to execution platforms and is insufficient for FPGA modeling.

GASPARD [5][9] is a MARTE compliant SoC co-design framework dedicated specially towards parallel hardware and software and allows to move from high level MARTE specifications to an executable platform. It exploits the parallelism included in repetitive constructions of hardware elements or regular constructions such as application loops.

The main contribution of this paper is to present part of a novel design flow using an extended version of MARTE for general modeling of FPGAs. Our methodology allows us to introduce PDR in MARTE for modeling all types of FPGAs supporting our chosen PDR flow. Finally using the MDE model transformations, the design flow can be used to bridge the gap between high level specifications and low implementation details to automatically generate the code required for the creation of bitstream(s) for FPGA implementation.

The rest of this paper is organized as follows. An overview of MDE is provided in section 2 while section 3 summarizes our MARTE compliant GASPARD framework. Section 4 describes PDR while section 5 gives a summary of related works. Section 6 illustrates our methodology related to implementing PDR supported FPGAs. This paper finishes with a case study in section 7 followed by a conclusion.

## II. Model Driven Engineering

MDE is centered around three focal concepts. *Models*, *Metamodels* and *Transformations*. A model is an abstract representation of some reality and has two core elements: *concepts* and *relations*. Concepts represent "things" and relations are the "links" between these things in reality. A model can be observed from different abstract point of views (views in MDE). A metamodel is a collection of concepts and relations for describing a model using a model description language and defines syntax of a model. This relation is analogous to a text and its language grammar. Each model is said to *conform* to its metamodel at a higher definition level.

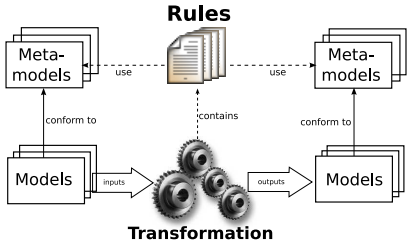


Fig. 1. *An overview of Model Transformations*

Models in MDE are not only used for communication and comprehension but using model transformations [6], produce concrete results such as a source code. A model transformation as shown in figure 1 is a compilation process that transforms a *source* model into a *target* model and allows to move from an abstract model to a more detailed model. The source and target models each conform to their respective metamodels thus respecting *exogenous* transformations. A model transformation is based on a set of *rules* (either declarative or imperative) that help to identify concepts in a source metamodel in order to create enriched concepts in the target metamodel. This separation allows to easily extend and maintain the compilation process. New rules extend the compilation process and each rule can be independently modified. Model transformations carry out refinements moving from high abstraction levels to low levels for code generation. At each intermediate level, implementation details are added to the compilation process. The advantage of this approach is that it allows to define several model transformations from the same abstraction level but targeted to different lower levels, offering opportunities to generate several implementations from a specification. The model transformations can be either unidirectional (modification of source model only: targeted model generated automatically) or bidirectional (target model is also modifiable) in nature. In the second case, this could lead to a model synchronization issue [7]. OMG has proposed the Meta-Object Facility (MOF) Query/View/Transformation (QVT) [8] standard for model query and transformations.

## III. GASPARD Co-Design Framework

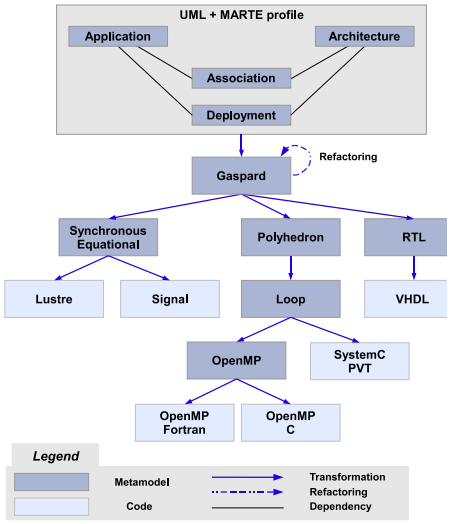


Fig. 2. *The existing GASPARD framework with deployment added at the MARTE modeling abstraction level*

GASPARD [5][9] is a MDE oriented SoC co-design framework and a subset of the MARTE standard currently supported by the SoC industry. In GASPARD as in MARTE, a clear *separation of concerns* exists between the hardware/software models as shown in figure 2. GASPARD integrates the MARTE allocation mechanism (*Alloc* package) that permits to link the independent hardware and software models (for e.g. mapping of a task or data onto a processor or a memory respectively). The concept used to specify an allocation is called an *Allocate*. An allocation can represent either a *spatial* or a *temporal* placement. Up till now GASPARD only supported spatial placement but we have also integrated the temporal placement allocation in order to implement systems supporting PDR.

GASPARD has contributed in MARTE conception with the *Repetitive Structure Modeling (RSM)* package. RSM is based on a MoC (Model of Computation) known as ArrayOL [10] which describes the potential parallelism in a system and is dedicated to intensive multidimensional signal processing

(ISP). RSM allows to describe the regularity of a system's structure (composed of repetitions of structural components interconnected in a regular connection pattern) and topology in a compact manner. GASPARD uses the RSM semantics to model large regular hardware architectures (such as multi-processor architectures) and parallel applications. GASPARD currently targets *control and data flow oriented* ISP applications (such as multimedia video codes, high performance applications, anti-collision radar detection applications). The applications targeted in GASPARD are widely encountered in SoC domain and respect ArrayOL semantics [10]. Although MARTE is suitable for modeling purposes, it lacks the means to move from modeling specifications to execution platforms. GASPARD bridges this gap and introduces additional concepts and semantics to fill this requirement for SoC co-design.

The first addition relates to the semantics of modeled applications. In MARTE, nearly all kinds of embedded applications can be specified but their behavior cannot be entirely defined. It is up to the designer/programmer to determine the precise behavior. As GASPARD deals with ISP applications based on a specific MoC, we only use the UML concept of *Component* (in order to define an application component) and MARTE *FlowPort* type (to define all port types in both the application and the architecture).

GASPARD also benefits from the notion of a *Deployment* model level [11] which is related to the specification of elementary components (basic building blocks of all other components). To transform the high abstraction level models to concrete code, detailed information must be provided. The Deployment level links every elementary component to an existing code for both the hardware and the application hence facilitating Intellectual Property (IP) reuse. Each elementary component can have several implementations: e.g. an application functionality can either be optimized for a processor (written in C/C++) or written in hardware (HDL) for implementation as an hardware accelerator. Hence this level is able to differentiate between the hardware and software functionalities independent from the compilation target. It provides IP information for model transformations to form a compilation chain to transform the high abstraction level models (application, architecture and allocation) for different domains (formal verification, simulation, high performance computing or synthesis). This concept is currently not present in MARTE and is a potential extension of the standard to allow a complete flow from model conception to automatic code generation. It should be noted that the different transformation chains (simulation, synthesis, verification etc.) are currently unidirectional in nature.

Once GASPARD models are specified in a graphical environment, MOMOTE (MOdel to MOdel Transformation Engine) tool which has been developed internally in the team and is based on EMFT QUERY [12], takes these models as input. MOMOTE is a Java framework that allows to perform model to model transformations. It is composed of an API and an engine. It takes source models as input and produces target models with each conforming to some metamodel.

MOCODE (MOdels to CODe Engine) is another GASPARD integrated tool for automatic code generation which is based on EMF JET (Java Emitter Templates) [13]. JET is a generic template engine for code generation purposes. The JET templates are specified by using a JSP (JavaServer Pages) like syntax and are used to generate Java implementation classes. Finally these classes can be invoked to generate user customized source code, such as Structured Query Language (SQL), eXtensible Markup Language (XML), Java source code or any other user specified syntax. MOCODE offers an API that reads input models, and also an engine that recursively takes elements from input models and executes a corresponding JET Java implementation class on them.

We are also in process of modifying the deployment level into a *controlled deployment* model to integrate the control aspect of PDR which is an offshoot of the works being done in the synchronous domain in the GASPARD framework [14]. This model will allow to link an elementary component with several IPs (allowing several possible final implementations) as compared to the current approach where an elementary component is only linked finally with one IP among several. This has allowed the concept of *configurations*: an elementary component can have different implementations in different configurations respecting the semantics of partial bitstreams. The control aspect in the deployment level allows to convert the semantics of the new deployment level into an control-mode automata based component approach and afterwards via model transformations, convert this control aspect into the state machine code to be implemented in the reconfigurable controller in the FPGA automatizing part of the reconfiguration management. However, this aspect is out of scope of this paper as here we only focus on the modeling approach.

## IV. Basic PDR related concepts

Currently PDR is only supported by Xilinx FPGAs. Xilinx initially proposed two methodologies (difference based and module based) [15],[16] followed by the *Early Access Partial Reconfiguration (EAPR)* [17] flow. The EAPR flow allows static nets to cross the reconfigurable region boundaries and supports 2D reconfigurable module shapes, thus resolving the drawbacks present in the earlier modular design methodology. The idea is that part(s) of the FPGA remains static, while another part(s) is dynamically reconfigurable at run-time. *Bus macros* are used to ensure proper routing between the static and dynamic parts during and after reconfiguration. The *Internal Reconfiguration Access Port (ICAP)* [18] is an integral component that permits to read/write the FPGA configuration memory at run-time. The ICAP is present in nearly all Xilinx FPGAs ranging from the low cost Spartan-3A(N) to the high performance Virtex-5 FPGAs [19]. For Virtex-II and Virtex-II Pro series, the ICAP furnishes 8-bit input/output data buses while with the Virtex-4 Series, the ICAP interface has been updated with 32-bit input/output data buses to increase its bandwidth. In combination with the ICAP, a *Reconfiguration controller* (either a PowerPC or a Microblaze) can be implemented inside the FPGA in order to build a self controlling dynamically reconfigurable system [18].

Virtex devices also support the feature of *glitchless dynamic reconfiguration*: If a configuration bit holds the same value

before and after reconfiguration, the resource controlled by that bit does not experience any discontinuity in operation, with the exception of LUTRAMs and SRL16 primitives [3]. This limitation was removed in the Virtex-4 family. With the introduction of EAPR flow tools, this problem has also been resolved for Virtex-II/Pro FPGAs.

## V. Related Works

ROSES [20] is an environment for Multiprocessor SoC (MPSoC) design and specification however it does not conform to MDE concepts and as compared to our framework, starts from a low level description equivalent to our deployment level. [21] provides a simulink based graphical HW/SW co-design approach for MPSoC but the MDE concepts are absent. In contrast, [22] uses the MDE approach for the design of a Software-Defined Radio (SDR), but they do not utilize the MARTE standard as proposed by OMG and utilize only pure UML specifications. While works such as [23] and [24] are focused on generating VHDL from UML state machines, they fail to integrate the MDE concepts for HW/SW co-design and are not capable of managing complex ISP applications. MILAN [25] is another project for SoC co-design benefiting from the MDE concepts but is not compliant with MARTE. Only the approach defined in [26] and [27] comes close to our intended methodology by using the MDE concepts and the MARTE standard for SoC co-design. Yet the disadvantage is that in reality it only generates the ISP application part to be implemented as a hardware accelerator in an FPGA. Hence there is no hardware description of FPGA at the high design level. MOPCOM [28] uses MDE and MARTE but is not oriented towards PDR. In [29], the authors present a design flow to manage partially reconfigurable regions of an FPGA automatically using SynDEx. A complete system (application/architecture) can be modeled and implemented, however the MDE concepts are strikingly absent. Similarly [30] present an HLS flow for PDR, yet it still starts from a lower abstraction level as compared to MDE.

In the domain of runtime reconfiguration, Xilinx initially proposed two design flows in [15] and [16] termed as the *Modular based* and *Difference based* approaches. The difference based approach is suitable for small changes in a bitstream but is inappropriate for a large dynamically reconfigurable module necessitating the use of the modular approach. However, both approaches were not very effective leading to new alternatives.

Sedcole et al [31] presented a modular approach that was more effective than the initial Xilinx methodologies and were able to carry out 2D reconfiguration by placing hardware cores above each other. The layout (size and placement) of these cores was predetermined. They made use of reserved static routing in the reconfigurable modules which allowed the signals from the base region to pass through the reconfigurable modules allowing communication between modules by using the principle of glitchless dynamic reconfiguration.

Huebner et al [32] implemented 1D modular reconfiguration using a horizontal slice based bus macro. All the reconfigurable modules that stretched vertically to the height of the device were connected with the bus macro for communication. They followed by providing 2D placement of modules of any rectangular size by using routing primitives that stretch vertically throughout the device [33]. A module could be attached to the primitive at any location, hence providing arbitrary placement of modules. The routing primitives are LUT based and need to be reconfigured at the region where they connect to the modules. A drawback of this approach is that the number of signals passing through the primitives are limited due to the utilization of LUTs. This approach has been further refined in [34].

In March of 2006, Xilinx introduced the *Early Access Partial Reconfiguration (EAPR)* [17] design flow along with the introduction of CLB based bus macros which are pre-routed IP cores. The concepts introduced in [31] and [32] were integrated in this flow. The restriction of full column modular PDR was removed allowing reconfigurable modules of any arbitrary rectangular size to be created. The EAPR flow also allows signals from the static region(s) to cross through the partially reconfigurable region(s) without the use of bus macros. Using the principle of glitchless reconfiguration, no glitches will occur in signal routes as long as they are implemented identically in every reconfigurable module for a region. The only limitation of this approach is that all the partial bitstreams for a module to be executed on a reconfigurable region must be predetermined hence making it *semi-partial dynamic* in nature.

Works such as [19] and [35] focus on implementing softcore internal configuration ports on Xilinx FPGAs such as the pure Spartan-3 which do not have the hardware ICAP core rendering dynamic reconfiguration impossible via traditional means. In [35] a soft ICAP known as JCAP (based on the serial JTAG interface) is introduced for realizing PDR while [19] introduces the notion of a PCAP (based on the parallel SelectMAP interface) providing improved reconfiguration rates as compared to the JTAG approach. However this approach is only suitable to reconfigure very small regions of FPGA and since the design is not an embedded one, it is impossible to retrieve bitstreams from an external memory. This issue has been addressed in [36], where a complete reconfigurable embedded design on a Spartan-3 board has been implemented using a reconfigurable coprocessor. The user application can map to a number of potential coprocessors and the reconfiguration controller can order the self reconfiguration of the system for the reconfigurable coprocessor resulting in loading of the partial bitstream related to a potential coprocessor. The results show that this achieves a compromise between the works presented in [19] and [35].

In [37], a new framework is introduced for implementing PDR by the utilization of a PLB ICAP. The ICAP is connected to the PLB bus as a master peripheral with direct memory access (DMA) to a connected BRAM (as compared to the traditional OPB based approach). This provides an increased throughput of about 20 percent by lowering the process load. [38] provides another flavor of a PDR architecture by attaching a Reconfigurable Hardware accelerator to a Microblaze Reconfiguration controller via a Fast Simplex Link (FSL) [39]. Works such as [40] use ICAP to connect with Network on chip (NoC) to allow distributed access to speed up reconfiguration

time. However the Read-modify-write (RMW) [18] mechanism is not supported which is an important factor to speed up reconfiguration times. This limitation has been resolved in [41] where an ICAP communicates with a NoC using a light weight RMW method in order to reduce reconfiguration time.

For our implementation purposes, we have focused mainly on the Xilinx EAPR flow methodology [3] as it is openly available and can be adapted to other PDR architecture implementations. Our contribution does not relates to creating a new PDR architecture methodology per se at the RTL level, but is based on how the methodology can be raised to a higher abstraction level for a) reducing design complexity and b) to create a generic PDR approach for implementing all ISP applications supporting our MoC. This approach can then be taken as an input for the designers who contribute to the PDR domain at the RTL level. While there are lots of related tools, works and projects; we have only detailed some and have not given an exhaustive summary. To the best of our knowledge, only our methodology takes into account the following domain spaces: SoC HW/SW co-design, ISP applications, MDE, MARTE standard and PDR which is the novelty of our design flow.

## VI. Modeling of Partially Dynamically Reconfigurable FPGAs

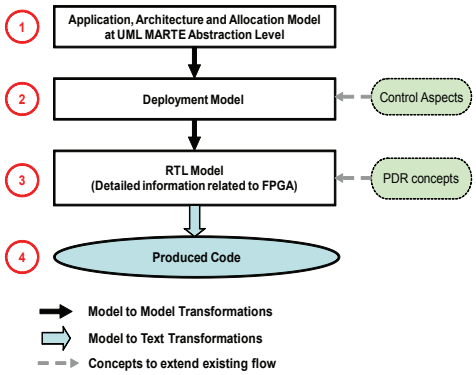


Fig. 3. *The complete design flow*

We first present our design flow to model and implement PDR supported fine grain reconfigurable architectures (FPGAs) as shown in figure 3 which is an extension of the design flow present in [27]. As described before, this paper only focuses on the first layer of our design flow (application, architecture and allocation modeling) which is the most abstract in nature. The 2nd layer deals with the Deployment layer with integrated control aspects for determining the configuration aspects for static/partial bitstreams. This layer serves as an input to the PDR-RTL layer where detailed transformation rules related to targeted application and FPGA in general (clock/reset signals, interface creation, constraint file among others) are present. This layer uses the control aspects in layer 2 for generating part of the reconfiguration controller and is responsible for partial FPGA layout for accelerator placement. Each part of these model levels/layers correspond to its respective metamodel. Finally using MOCODE, it is possible to convert the models to source code. Once the source code for the application (implemented as a hardware accelerator) and the reconfigurable controller is obtained, usual synthesis flow can be invoked using commercial tools such as Xilinx ISE [42] for final implementation. Our aim is not to replace the commercial tools but to aid them in the conception of a system. While tools like PlanAhead [43] are capable of estimating the FPGA resources required for a reconfigurable module, it is finally up to the user to decide the best placement depending on QoS requirements. Also as our work deals with dynamic partially reconfigurable FPGAs and currently only Xilinx FPGAs support this feature, our modeling methodology revolves around the Xilinx reconfiguration flow as it is openly available and flexible enough to be modified. While this does make the architectural aspects of our design flow restricted to Xilinx based technologies, it is an implementation choice as currently no other FPGA vendor supports this feature. It should be noted that our methodology can be used as a building block to support other non standard PDR implementations based on Xilinx FPGAs (use of Soft ICAP cores for example).

### A. MARTE Hardware concepts overview

The hardware concepts in MARTE are grouped in the *Hardware Resource Model (HRM)* package. HRM consists of several views, a functional view (*HwLogical* sub-package), a physical view (*HwPhysical* sub-package) or a merge of the two. The two sub-packages derive certain concepts from the *HwGeneral* root package in which *HwResource* is a core concept that defines a generic hardware entity. A HwResource can be composed of other HwResource(s) (for example a processor containing an ALU). This concept is then further expanded according to the functional or physical specifications. The functional view of HRM defines hardware resources as either *computing*, *storage*, *communication*, *timing* or *device* resources. The physical view represents hardware resources as physical components with details about their shape, size and power consumption among other attributes. GASPARD currently only supports the functional view, but we have also integrated the physical and merged views for modeling PDR featured architectures. The HRM also exploits the Non-Functional Properties (NFP) MARTE package that introduces a value specification language (VSL) which supports complex expressions for specifying non-functional properties and quantitative annotations with measurement units. The NFP package provides a rich library of basic types like *Data size*, *Data Transmission Rate* and *Duration*.

### B. MARTE modifications for PDR concepts

In order to model PDR supported FPGAs, the HRM package was examined and we found it to be lacking in certain aspects. The *HwComputing* sub-package in the HRM functional view defines a set of active processing resources pivotal for an execution platform. A *HwComputingResource* symbolizes an active processing resource that can be specialized as either a processor (*HwProcessor*), an ASIC (*HwASIC*) or a PLD (*HwPLD*). An FPGA is represented by the HwPLD stereotype, it can contain a RAM memory (*HwRAM*) (as well as other

HwResources) and is characterized by a technology (SRAM, Antifuse etc.). The cell organization of the FPGA is characterized by the number of rows and columns, but also by the type of architecture (Symmetrical array, row based etc.). These concepts are partly sufficient enough for high level abstract FPGA description but do not integrate all aspects (such as interfaces for IP cores, processor implementation type etc.) and need a detailed modeling for representing a complete real heterogeneous FPGA. Also the concepts related to representing a processor are not sufficient for a complex SoC on FPGA design in which a processor can either be implemented as a softcore IP or integrated as a hardcore IP. We thus add the attribute **imtype** (Implementation_Type) that is flexible enough to define a processor implementation as either **Hardcore** or **Softcore** and adaptable using the **Other** and **Undefined** types. The last two types have been added for extension purposes. The **Other** type is denoted for other existing technologies which are not actually specified at the time of modeling (in the case of processor implementation, this type is set to false) and **Undefined** for future evolution in hardware and to allow easy modification of existing models. They can be viewed as having equivalent purposes but are created to avoid ambiguity. Figure 4 shows only the simplified modeling description of the modified HwComputing sub-package related to a processor implementation.

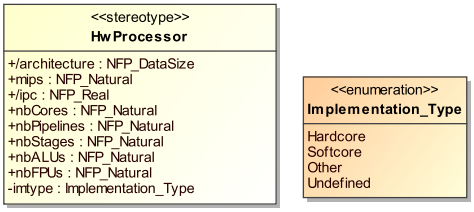


Fig. 4. *Modified version of the HwProcessor concept*

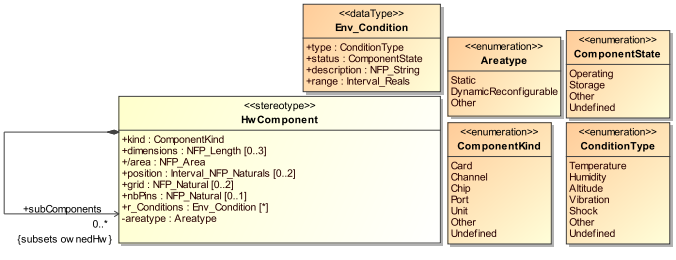


Fig. 5. *Modified version of the HwComponent concept*

The second modification relates to the physical *HwLayout* sub-package as shown in figure 5. The core concept of this package is *HwComponent* which is an abstraction of any real hardware entity based on its physical attributes. HwComponent can be specialized as either *HwChip* (e.g. a processor), *HwChannel* (e.g a bus), *HwPort* (e.g. an interface), *HwCard* (e.g. a motherboard) or a *HwUnit* (a hardware resource that does not fall into the preceding four categories). As a PDR featured architecture consists of either static or dynamically reconfigurable region(s), we have introduced the attribute **areatype** (Areatype) which can be either **Static**, **DynamicReconfigurable** or typed as **Other** for extension purposes. This concept has been introduced in the MARTE physical concepts as the area properties for a hardware component are usually expressed in the physical sub-package of the HRM. Figure 5 thus shows only the simplified overview of our modified HwComponent concept.

These are the 2 added extensions of the MARTE standard. These concepts are specifically added to the high level in order to generally benefit other frameworks and system descriptions and they could be easily extended. While these modifications seem trivial in nature, they make a definite impact in the corresponding model to model transformations for the final implementation. We now present the specific concepts related to FPGA and PDR in our methodology.

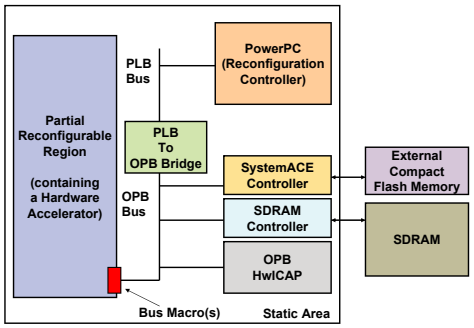


Fig. 6. *Block Diagram of the architecture of our reconfigurable system*

In figure 6 we present an example of a PDR supported Xilinx FPGA that we have implemented in reality. We have used the Virtex-II Pro XC2VP30 on a XUP Board [44] as a reference as it seems to be a popular choice for implementing PDR. We have implemented a Reconfiguration Controller (a PowerPC in this case) connected to the high speed 64-bit PLB bus and links with the slower slave peripherals (connected to the 32-bit OPB bus) via a PLB to OPB Bridge. The buses and the bridge are a part of the IBM Coreconnect technology [45]. The OPB bus is attached to some peripherals such as: A SystemACE controller (for accessing the partial bitstreams placed in an external onboard Compact Flash (CF) card). A SDRAM controller for a DDR SDRAM present onboard (permits the partial bitstreams to be preloaded from the CF during initialization for decreasing the reconfiguration time). An ICAP is present in the form of an OPB peripheral (OPBHwICAP) and carries out partial reconfiguration using the read-modify-write (RMW) mechanism. The static (base) portion of the FPGA is connected to a Reconfigurable Hardware Accelerator (RHA) via bus macros. Although the RHA can be placed with the fast PLB bus, it is an implementation choice to connect it with the OPB bus to make the system more diverse at the cost of reconfiguration time. The concepts such as PowerPC, PLB and OPB buses, PLB to OPB Bridge, CF and SDRAM memories can defined using the current MARTE HRM concepts. However the peripherals, bus macros, ICAP and RHA require an extended and more detailed conception. An internal memory can also be used to store the partial bitstreams depending upon the application size. Since our targeted applications cannot be placed inside the internal memory, we have used an external memory.

The *HwCommunication* sub-package in the HRM functional view represents the concepts for all hardware communications. *HwMedia* is the central concept that defines a communication resource capable of data transfer with a theoretical bandwidth. It can be controlled by *HwArbiter*(s) and connected to other HwMedia(s) by means of a *HwBridge*. A *HwEndpoint* defines a connection point of a HwResource and can be defined as an interface (e.g. pin or port). *HwBus* illustrates a specific wired channel with particular functional attributes. These concepts are sufficient and abstract enough to define all kind of communication resources. Some of the other common HRM concepts that we utilize for PDR are *HwComputingResource* (to describe a general computing resource) from the *HwComputing* package, *HwRAM* and *HwROM* from the *HwMemory* package (for RAM and ROM concepts), *HwStorageManager* from the *HwStorageManager* package (for a memory controller), *HwClock* from the *HwTiming* package (to specify a clock) and *HwIO* from the *HwIO* package (for an I/O resource).

Xilinx provides the notion of an Intellectual Property Interface (IPIF) module which acts as a hardware bus wrapper specially designed to ease IP core interfacing with the IBM Coreconnect buses using IPIC connections. It can also be used for other purposes such as connecting the OPB bus to a DCR bus [45] (another bus of the Coreconnect technology). As all peripherals in our architecture consist of the IPIF module and an IP core, this is a vital modeling concept and has permitted us to model all peripherals which are themselves hierarchically composed. The abstract IPIF module has two basic attributes: a **mode** which can be either **Master**, **Slave** or **Master/Slave**, and **type** that determines the protocol of IPIF adapted for a particular bus. It can be either **PLB**, **OPB** or extensible using **Other** or **Undefined** types. We avoided adding detailed information related to the options and protocols offered by IPIF (software registers, FIFOs etc) to simplify its definition at the high abstraction level. The IPIF is typed as *HwEndpoint* to illustrate that it is a hardware wrapper module providing an interface to the actual IP core. This approach can be adapted to model customized wrappers for customized user IPs. Figure 7 shows the IPIF design.

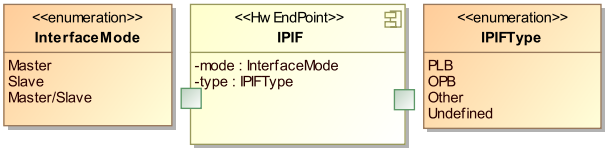


Fig. 7. *Modeling of the IPIF hardware wrapper*

The second modeling concept is of Bus macros (BMs). Although the EAPR flow allows static nets in the base design to pass through the reconfigurable region(s) without the use of bus macros, they are still essential in order to ensure the correct communication routing between the static and dynamic regions. Being CLB based in nature, they provide a unidirectional 8-bit data transfer. Bus macros have been modeled having four attributes. The **sigdir** attribute determines the communication direction that can be **Left2Right** or **Right2Left** (for Virtex-II and Virtex-II Pro devices), as well as **Top2Bottom**, **Bottom2Top** or **Other** for Virtex-IV and other future PDR supported devices. The **width** attribute determines the CLB width of the bus macro (**2CLBs** or **4CLBs** width making it either a narrow or wide bus macro or use of **Other** for a user specified width). The **Synchronous** attribute determines if the bus macro is a synchronous one or not. We have assigned a default value of **true** to this attribute (as recommended by Xilinx). The final attribute **device** determines the targeted FPGA device family (either **Virtex-II**, **Virtex-II Pro**, **Virtex-4** or a newer device such as Virtex-5 using the **Other** type). The Bus macro (*Busmacro*) (as shown in figure 8) is typed as *HwEndpoint* to illustrate that it is a communication medium between the static and dynamically reconfigurable modules of the FPGA.

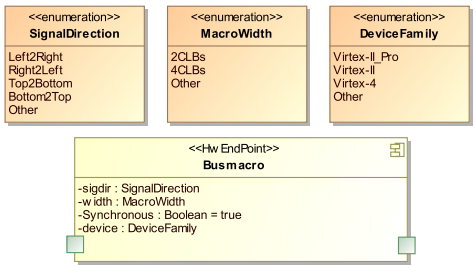


Fig. 8. *Modeling of a Bus macro*

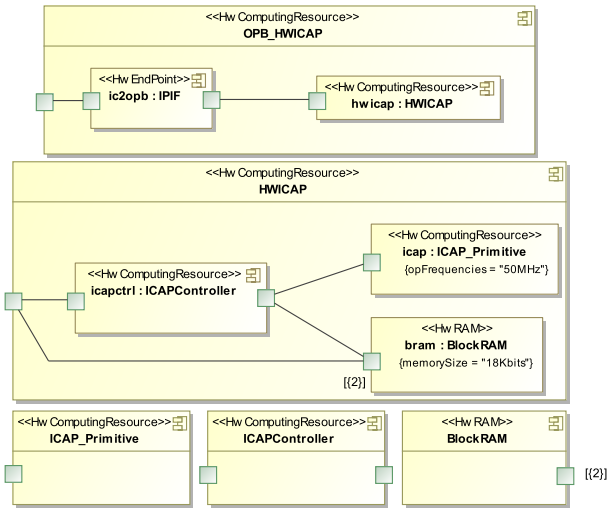


Fig. 9. *Modeling of the OPB HWICAP Peripheral*

Modeling of the OPB_HWICAP peripheral is then carried out as shown in figure 9. It consists of an IPIF (**ic2opb**) connected to the HWICAP core (**hwicap**) (typed as *HwComputingResource*) and is itself defined as a *HwComputingResource*. The HWICAP core is itself composed of three sub components: an ICAP controller (**icapctrl**) and ICAP Primitive (**icap**) both typed as *HwComputingResource*(s) and a BlockRAM (**bram**) defined as *HwRAM* for storing a configuration frame of FPGA memory. The BlockRAM contains a port having a multiplicity of 2 indicating that it is repeated two times (dual port RAM). We have used the notion of a *Reshape* connector [10] (as defined in the MARTE RSM package and in our MoC) in order to link the sub components of the HWICAP. The Reshape allows to represent complex link topologies

in a simplified manner. In figure 9, the Reshape connectors permit to specify accurately which port (either the port of the ICAPController or the single port of the HWICAP itself) is connected to which repetition of the port of the BlockRAM. The sub components of HWICAP also have specific attributes (such as BlockRAM having a 18Kbit memory) related to actual architectural details of the targeted FPGA. We refer the reader to [18] for a detailed description related to HWICAP.

Figure 10 represents the modeling of the Reconfigurable Hardware Accelerator (RHA). The PRR (Partial reconfigurable region) consists of a RHA (**HwAcc**) defined as *HwPLD* having ports **AccessIn** and **AccessOut** and an IPIF module (**Acc2opb**). The PRR is typed as the generic *HwResource* type in order to illustrate that the partially reconfigurable region can be either generic or have a specific functionality. The RHA is typed as *HwPLD* as it is reconfigurable, as compared to a typical hardware accelerator in a large scale SoC design which can be seen as a *HwASIC* (after fabrication) depending upon the designer's point of view.

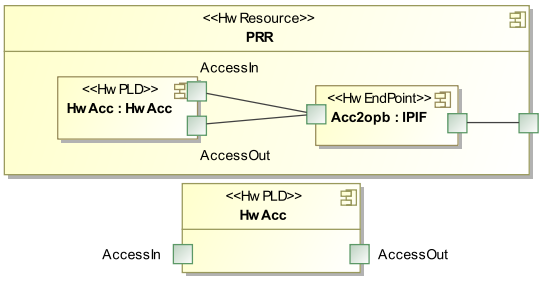


Fig. 10. *A Reconfigurable Hardware Accelerator*

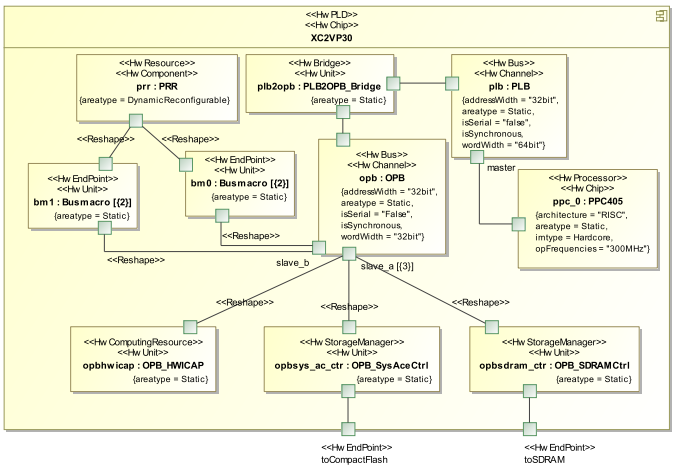


Fig. 11. *Modeling of our PDR Architecture*

Figure 11 finally shows our reconfigurable architecture (An XC2VP30 Virtex-II Pro chip) using our proposed concepts in a merged functional/physical view to express all the necessary attributes related to the corresponding physical/logical stereotypes. Every hardware component has two type definitions (the first being the functional and the second representing the physical one). The XC2VP30 chip consists of a PowerPC PPC405 (**ppc_0**) connected to the slave peripherals: the OPB_SysAceCtrl (**opbsys_ac_ctr**), the OPB_HWICAP (**opbhwicap**), the OPB_SDRAMCtrl (**opbsdram_ctr**) and the PRR (**prr**) via the PLB (**plb**) and OPB (**opb**) buses. The PLB2OPB_Bridge (**plb2opb**) connects the two buses, while Bus macro(s) (**bm0** and **bm1** having types Left2Right and Right2Left respectively) connect the OPB bus to the PRR. Each of the bus macros is instantiated two times (multiplicity of 2 on both **bm0** and **bm1** respectively). The OPB bus has a **slave_a** port with a multiplicity of 3 to allow the bus to connect to the peripherals (opbhwicap, opbsys_ac_ctr and opbsdram_ctr). Reshape connectors are used to determine which peripheral is connected to which repetition of the slave port. Similarly Reshape connectors are used to determine the accurate connections between the bus macros and the ports of OPB and PRR. Although a single slave port can be used on OPB with an appropriate multiplicity to include the topology of bus macros, this is avoided to reduce the design complexity. Finally, the XC2VP30 contains two HwEndPoint(s) interfaces, **toCompactFlash** and **toSDRAM** to connect **opbsys_ac_ctr** and **opbsdram_ctr** to the external Compact Flash and SDRAM memories respectively. The OPB arbiter is not modeled as it is considered to be a part of the OPB Bus. It should be noted that this is a top level view only and nearly each component is itself hierarchically composed. Note that the new attributes and those by default in the HRM package of MARTE allow the designer to specify general attributes of each component at the highest abstraction level (e.g. ppc_0 having a frequency of 300 MHz).

## VII. Case Study: A GASPARD application mapped on our PDR architecture

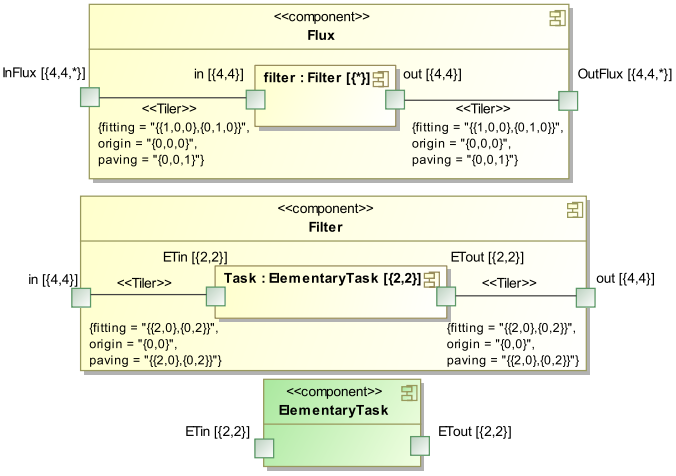


Fig. 12. *Model of an Image Filter task*

A case study of a complete SoC model is presented here to illustrate our modeling methodology. The modeled application *MainApplication* is an academic grayscale 4x4 pixel image filter application (producing 8-bit images) respecting our MoC. It consists of three tasks (application components) : An image sensor PictureGen (**pg**), the main image filter task Flux (**tasks**) (figure 12) and an output PictureRead (**pr**). The Flux component is comprised of a Filter component (**filter**) (repeating infinitely as shown by the multiplicity of *). The

Filter component itself contains an elementary application component ElementaryTask (**Task**) being repeated four times (having a multiplicity of 2,2). The Tiler connectors are used to describe the tiling of produced and consumed arrays by a pattern mechanism [10]. The elementary component can have several implementations and the controlled deployment layer can create different configurations for the reconfigurable hardware accelerator and this information is thus passed to the PDR-RTL layer.

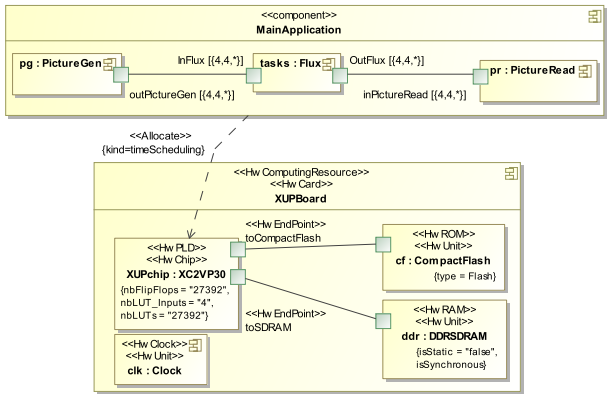


Fig. 13. *Allocation: Level 1*

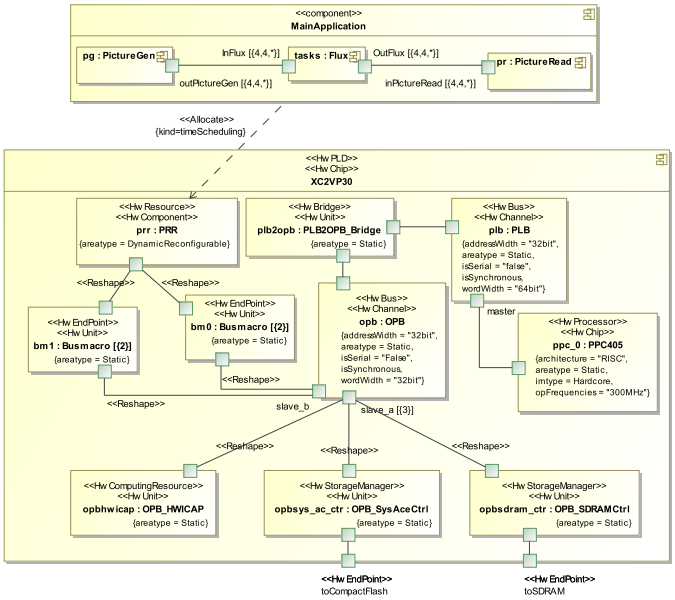


Fig. 14. *Allocation: Level 2*

We then illustrate the different levels of allocation of the application onto the architecture. In figure 13, the model of the whole application is shown allocated to the XC2VP30 chip (**XUPchip**) on an XUPBoard using the *Allocate* type allocation. Currently GASPARD only supports spacial placement (static scheduling at compilation time due to the nature of targeted applications), however due to the nature of PDR and related applications; we have integrate the temporal placement: *timeScheduling* (dynamic scheduling of a set of tasks spatially allocated to the same platform resource) notion of allocation as defined in MARTE standard. Figure 14 presents a detailed view of the allocation illustrating the mapping of the application onto the PRR reconfigurable portion. Due to space limitations we have not presented the last level of allocation in which the application is finally placed on the hardware accelerator **HwAcc** for execution. The XUPBoard also contains a global Clock (**clk**) and the CompactFlash (**cf**) and DDR SDRAM (**ddr**) memories. The concepts introduced in our approach can be modified and extended to manipulate other PDR supported architectures such as introduced in [37], [38] and can be adapted to serve new emerging technologies such as explained in [19] and [35].

This point is validated as we present another PDR architecture as shown in figure 15. The figure shows the merged functional/physical modeling of a PLB ICAP based PDR architecture as defined in [37]. We have omitted some of the high level attribute specifications and type definitions in the figure in order to respect the space limitations. However, the modeling clearly illustrates that the PDR modeling methodology that we have proposed can be used as a building block. The model to model transformation rules can be extended by addition of new rules, hence it is possible to implement other existing and future PDR architectures.

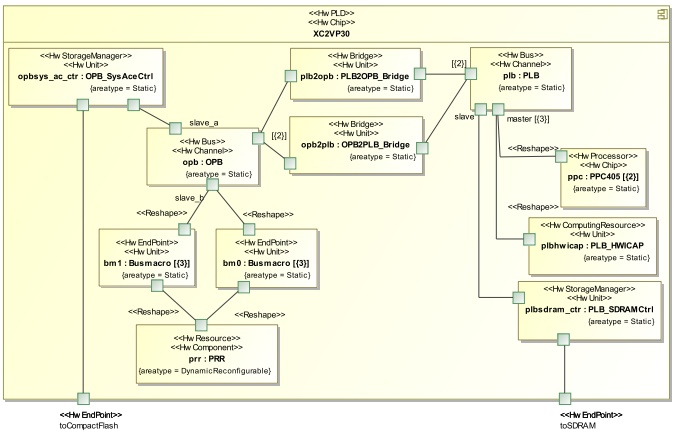


Fig. 15. *Modeling of a PLB ICAP based Reconfigurable Architecture*

Our modeling methodology can also be extended by integrating the MARTE *HwPhysical* arrangement notation which provides rectangular grid based placement mechanisms in order to bridge the gap between UML diagrams and actual physical layout and topology of the targeted architecture. Unfortunately, due to the current functional limitations of the modeling tools (Papyrus[1], MagicDraw[2]), it is not possible to express this view. However, this view could be a potential additional aid to commercial PDR tools such as PlanAhead [43]. Designers can specify the FPGA layout at the MARTE specification level. At the simulation level, designers can accurately estimate if the layout is feasible and determine the number of consumed FPGA resources. Finally using these simulation results, the high level models can be modified resulting in an effective Design Space Exploration Strategy (DSE) for PDR based FPGA implementation.

[1]http://www.papyrusuml.org/
[2]http://www.magicdraw.com/

## VIII. Conclusion

This paper presents a novel methodology to implement FPGAs based on a MDE approach using the MARTE standard. For this purpose, modifications have been made to the MARTE specifications to resolve the current limitations for FPGA modeling. This paper introduces notions in the MARTE standard such as those of peripherals and hardware wrappers, which can be adapted to new versions of the standard. These modifications make a direct impact to the corresponding model transformations in order to move from model level specifications to an executable FPGA platform. Further more, they allow us to model a complete SoC on an FPGA. Afterwards we integrate the aspects of Partial Dynamic Reconfiguration using the modified version of the standard. Currently we adhere to the Xilinx based PDR design flow due to its availability and extendable nature. However our PDR based methodology can be used as a template in order to model and implement other existing or future PDR based fine grain reconfigurable architectures. Coarse grain reconfigurable architectures can also be addressed using the GASPARD framework and our design flow. By modeling a complete system (application and architecture) we have defined the first stage of our design flow. In future works, we will detail the controlled deployment level which will allow to link an elementary component with several unique IPs thus creating the concept of configurations, and hence creating part of the reconfigurable controller responsible for managing the self reconfiguration. Finally the enriched RTL level (the level which details the abstract FPGA concepts modeled above) will be able to take the upper model levels as inputs and generate the necessary code required for PDR implementation. The code can then be used as input for commercial tools for final FPGA synthesis.

## References


[1] Open SystemC Initiative, "SystemC," 2007. [Online]. Available: http://www.systemc.org/

[2] Planet MDE, "Portal of the Model Driven Engineering Community," 2007, http://www.planetmde.org.

[3] P. Lysaght et al, "Invited Paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL'06)*, 2006.

[4] Object Management Group, "OMG MARTE Standard," 2007, http://www.omgmarte.org.

[5] The DaRT team, "GASPARD Design Environment," 2008, https://gforge.inria.fr/projects/gaspard2.

[6] S. Sendall and W. Kozaczynski, "Model transformation: The heart and soul of model-driven software development," *IEEE Software*, vol. 20, no. 5, pp. 42–45, 2003.

[7] P. Stevens, "A landscape of bidirectional model transformations," in *Generative and Transformational Techniques in Software Engineering 2007 (GTTSE'07)*, 2007.

[8] Object Management Group Inc., "MOF Query / Views / Transformations," http://www.omg.org/cgi-bin/doc?ptc/2005-11-01, Nov. 2005.

[9] A. Gamatié et al, "A model driven design framework for high performance embedded systems, Tech. Rep. 6614, Aug. 2008. [Online]. Available: http://hal.inria.fr/inria-00311115/en

[10] P. Boulet, "Array-OL Revisited, Multidimensional Intensive Signal Processing Specification," INRIA, Tech. Rep., 2007, "http://hal.inria.fr/inria-00128840/en/".

[11] R.B. Atitallah et al, "Multilevel MPSoC simulation using an MDE approach," in *SoCC 2007*, 2007.

[12] Eclipse, "Eclipse Modeling Framework Technology," http://www.eclipse.org/emft.

[13] ——, "EMFT JET," http://www.eclipse.org/emft/projects/jet.

[14] H. Yu et al, "Safe design of high-performance embedded systems in an mde framework," *Innovations in Systems and Software Engineering (ISSE)*, vol. 4, no. 3, 2008. [Online]. Available: http://tinyurl.com/3qr8jz

[15] "Two Flows for Partial Reconfiguration: Module Based or Difference Based," in *Xilinx Application Note XAPP290, Version 1.1*, 2003.

[16] "Two Flows for Partial Reconfiguration: Module Based or Difference Based," in *Xilinx Application Note XAPP290, Version 1.2*, 2004.

[17] Xilinx, "Early Access Partial Reconfigurable Flow," 2006, http://www.xilinx.com/support/prealounge/protected/index.htm.

[18] B. Blodget et al, "A lightweight approach for embedded reconfiguration of FPGAs," in *Design, Automation & Test in Europe (DATE'03)*, 2003.

[19] S. Bayar and A. Yurdakul, "Dynamic Partial Self-Reconfiguration on Spartan-III FPGAs via a Parallel Configuration Access Port (PCAP)," in *2nd HiPEAC workshop on Reconfigurable Computing (HiPEAC 08)*, 2008.

[20] W. Cesario et al, "Component-Based Design Approach for Multicore SoCs," *Design Automatic Conference (DAC'02)*, vol. 00, p. 789, 2002.

[21] Y.Atat and N. Zergainoh, "Simulink-based MPSoC Design: New Approach to Bridge the Gap between Algorithm and Architecture Design," in *ISVLSI'07*, 2007, pp. 9–14.

[22] G. Gailliard et al, "Transaction level modelling of SCA compliant software defined radio waveforms and platforms PIM/PSM," in *Design, Automation & Test in Europe (DATE'07)*, 2007.

[23] R. Damasevicius and V. Stuikys, "Application of UML for hardware design based on design process model," in *ASP-DAC'04*, 2004.

[24] W.E. McUmber et al, "UML-based analysis of embedded systems using a mapping to VHDL," in *IEEE International Symposium on High Assurance Software Engineering (HASE'99)*, 1999, pp. 56–63.

[25] S. Mohanty et al, "Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation," in *LCTES/Scopes 2002*, 2002.

[26] S. Le Beux et al, "A Model Driven Engineering Design Flow to generate VHDL," in *International ModEasy'07 Workshop*, 2007.

[27] S. Le Beux, "Un flot de conception pour applications de traitement du signal systématique implémentées sur FPGA à base d'Ingénierie Dirigée par les Modeles," Ph.D. dissertation, LIFL / USTL, France, 2007.

[28] A. Koudri et al, "Using MARTE in the MOPCOM SoC/SoPC Co-Methodology," in *MARTE Workshop at DATE'08*, 2008.

[29] F. Berthelot and F. Nouvel and D. Houzet, "A Flexible system level design methodology targeting run-time reconfigurable FPGAs," *EURASIP Journal of Embedded Systems*, vol. 8, no. 3, pp. 1–18, 2008.

[30] M. Boden et al, "GePARD - a High-Level Generation Flow for Partially Reconfigurable Designs," in *ISVLSI 2008*, 2008.

[31] P. Sedcole et al, "Modular Partial Reconfiguration in Virtex FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL'05)*, 2005, pp. 211–216.

[32] J. Becker and M. Huebner and M. Ullmann, "Real-Time Dynamically Run-Time Reconfigurations for Power/Cost-optimized Virtex FPGA Realizations," in *VLSI'03*, 2003.

[33] M. Huebner et al, "New 2-Dimensional Partial Dynamic Reconfiguration Techniques for Real-Time Adaptive Microelectronic Circuits," in *ISVLSI'06*, 2006, p. 97.

[34] C. Schuck et al, "A framework for dynamic 2D placement on FPGAs," in *IPDPS 2008*, 2008.

[35] K. Paulsson et al, "Implementation of a Virtual Internal Configuration Access Port (JCAP) for Enabling Partial Self-Reconfiguration on Xilinx Spartan III FPGAs," *International Conference on Field Programmable Logic and Applications (FPL 2007)*, pp. 351–356, 2007.

[36] E. Canto et al, "Self Reconfiguration of Embedded Systems Mapped on Spartan-3," *ReCoSoC'08*, 2008.

[37] C. Claus et al, "A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration," *IPDPS 2007*, pp. 1–7, 2007.

[38] A. Tumeo et al, "A Self-Reconfigurable Implementation of the JPEG Encoder," *ASAP 2007*, pp. 24–29, 2007.

[39] Xilinx, "Fast Simplex Link Channel (FSL)," 2004.

[40] R. Koch et al, "An adaptive system-on-chip for network applications," in *IPDPS 2006*, 2006.

[41] C. Schuck et al, "An Interfance for a Decentralized 2D-Reconfiguration on Xilinx Virtex-FPGAs for Organic Computing," in *ReCoSoC'08*, 2008.

[42] Xilinx, "ISE Foundation Software," 2008.

[43] N. Dorairaj et al, "PlanAhead Software as a Platform for Partial Reconfiguration," *Xcell Journal*, vol. 55, pp. 68–71, 2005.

[44] The Xilinx XUP-V2Pro Board, http://www.xilinx.com/univ/xupv2p.html.

[45] IBM Corporation, "The Coreconnect Bus Architecture, white paper," in *International Business Machines Corporation*, 2004.