Des Autom Embed Syst (2012) 16:161–187
DOI 10.1007/s10617-012-9101-2

System level modeling methodology of NoC design from UML-MARTE to VHDL

Majdi Elhaji · Pierre Boulet · Abdelkrim Zitouni · Samy Meftali · Jean-Luc Dekeyser · Rached Tourki

Received: 22 October 2011 / Accepted: 14 November 2012 / Published online: 7 December 2012

Abstract The evolution of the semiconductor technology caters for the increase in the System-on-Chip (SoC) complexity. In particular, this complexity appears in the communication infrastructures like the Network-on-Chips (NoCs). However many complex SoCs are becoming increasingly hard to manage. In fact, the design space, which represents all the concepts that need to be explored during the SoC design, is becoming dramatically large and difficult to explore. In addition, the manipulation of SoCs at low levels, like the Register Transfer Level (RTL), is based on manual approaches. This has resulted in the increase of both time-to-market and the development costs. Thus, there is a need for developing some automated high level modeling environments for computer aided design in order to handle the design complexity and meet tight time-to-market requirements. The extension of the UML language called UML profile for MARTE (Modeling and Analysis of Real-Time and Embedded systems) allows the modeling of repetitive structures such as the NoC topologies which are based on specific concepts. This paper presents a new methodology for modeling concepts of NoC-based architectures, especially the modeling of topology of the interconnections with the help of the repetitive structure modeling (RSM) package of MARTE profile. This work deals with the ways of improving the effectiveness of the MARTE standard by clarifying and extending some notations in order to model complex NoC topologies. Our contribution includes a description of how these concepts may be mapped into VHDL. The generated code has been successfully evaluated and validated for several NoC topologies.

Keywords SoC · NoC · RTL · RSM · UML/MARTE

1 Introduction

Systems-on-chips (SoCs) provide integrated solutions to meet the challenging design problems in the areas of embedded systems and consumer electronic. Much of the progress in

M. Elhaji (✉)
Electronics and Microelectronics Laboratory, Physics Department, Faculty of Sciences of Monastir, Monastir, Tunisia
e-mail: majdi.elhaji@leme.tn

these fields allows the designers to conceive complex electronic engines and reduce time-to-market. Modern SoCs are composed of tens to hundreds of IP cores. Actually, they have emerged as a new solution for designing embedded systems in order to increase performance and reduce power consumption. The on-chip communication paradigm is introduced as a Network-on-Chip (NoC) by Benini and DeMicheli [1], Dally [2] and Sgroi [3]. Many propositions of NoC architectures for SoC design have been presented in literature, such as Spin [4], Hermes [5], Xpipes [6], Octagon [7], Mesh [8], Honeycomb [9].

The designer is empowered with a level of abstraction that focuses more on the system functionality rather than on the low level design details. There is also a demand for developing high level modeling environments for computer-aided design. The modeling of highly repetitive structures such as the network on chip topologies in graphical forms poses a particular challenge. The Model Driven Engineering is a software development methodology in which the complete system is modeled at a high abstraction level allowing several abstraction stages. UML is now the most widespread modeling language used by both industry and research communities.

The field of real-time embedded software systems is a domain for which extensions to UML are required to provide a more precise expression of domain-specific phenomena. Consequently, the OMG (Object Management Group) has adopted a new UML profile for real-time and embedded systems, called MARTE [10]. MARTE [11] was required to support modeling and analysis of component-based architectures. The main objective of the development of these software approaches and tools is the automatic generation of accelerators in a HDL (Hardware Description Language) at RTL. However, the design of hardware accelerators is generally error-prone as it is directly dependent on the designer's expertise. Due to the necessity of functional validation of each component in the hardware architectures, the required development time increases exponentially.

For these reasons, deducing and automating the VHDL code generation from UML/-MARTE, precisely from the repetitive structure modeling (RSM) concepts of MARTE, seems to be of crucial importance for developing time reduction and code optimization.

The RSM package of the MARTE profile is based on the Array-OL language, which was developed at Thales Underwater Systems [12] to fulfill the needs for the specification of multidimensional signal processing applications. Array-OL has been extended with delays [13] and these concepts have been generalized in RSM to model repetitive hardware architectures and regular mappings of parallel applications to repetitive hardware. In this paper, we present a methodology for modeling NoCs using the MARTE profile and the Gaspard2 tool to generate VHDL codes for many NoC topologies.

Gaspard2 [14] is a CAD tool that aims at providing a new method for SoC design and is dedicated to the visual co-design for embedded systems based on MARTE. It is based on the extensive use of the RSM package of MARTE at the specification level. It allows to model multidimensional data accesses without compromising the usability of the specification and provides ways to statically schedule these applications on parallel hardware platforms. All the regularity in the application, hardware architecture and mapping, can be modeled in a compact way. By adopting the Gaspard2 strategy, the massive reuse of a few existing components is guaranteed. For example, to generate a VHDL code of an $M \times N$ grid we just need to change the value of M or N and reuse the same model. This approach can significantly reduce the time-to-market, facilitates redesign when a new SoC needs to be developed and presents a way for fast development. We evaluate our approach using the quality of solution in terms of mapping and space complexity.

The structure of the paper is as follows. After a discussion of related works, we recall some background on NoCs, Gaspard2 and MARTE RSM. We then present our methodology and the VHDL code generation. We conclude our paper with a discussion of our results.

2 Related work

We note that in literature there are a few works that deal with NoC modeling via the MARTE profile, defining methodology or providing the effectiveness of notation to model all NoC concepts like topology. In [15], authors identified two main characteristics of NoCs that are useful to design, analyze and understand the structure and behavior of NoCs: the topology of the network and the routing algorithm.

Furthermore, in the RTL level, the method of describing hardware architectures is performed via VHDL or VERILOG language. Thus, to bridge the gap between RTL and system level many works try to propose new approaches for making a link between both levels.

For NoCs, at RTL level, there are many works that describe new architectures and topologies. In [1], authors describe NoC as a new paradigm for the SoC communication. Other works are based on manual methods for modeling new architectures and topologies of NoC-based HDL [4–6, 16].

Some efforts are presented in the literature to generate an RTL code from a NoC specification. For instance, the Xpipes compiler tool that generates the entire design is intended to xpipe NoC. This automates some of the most critical and time intensive NoC design steps such as topology synthesis and core mapping. Also the SUNMAP tool is used to design only a standard topologies like a mesh and torus [17].

Other design flows have been proposed in the research community like: Arteris [18] iNoCs [19], Silistix [20] and Spider [21]. They provide some design automation tools for NoCs. For instance, in the iNoCs the tool flow starts from the constraints of both the architecture and the application to go from input specifications all the way to the RTL description of the topology.

In order to handle the design complexity and meet the tight time-to-market constraints, it is not sufficient to automate most of these NoC design phases. For example, mapping routers to a topology is time consuming and can require a large number of developers. In our work, one can model, in a compact way, the topology using the MARTE RSM package and use the Garpard tool to generate the corresponding VHDL code.

However, the design of hardware accelerators at RTL level is error-prone and time-consuming. Therefore, tools that are based on automated methodologies have been proposed, which aids in the design of hardware accelerators. In [22], the generation of hardware accelerators at RTL level is proposed by ALPHA0 and ALPAHRD. Other techniques based on Synchronous Data Flow (SDF) are presented in [23], where the VHDL code generation is carried out via SDF. Authors in [24] presented a Simulink-based HDL for VHDL code generation. In [25], the authors used the Array-OL language, which is the basis of the RSM package of the MARTE profile, in order to map the structure into VHDL. However, the generated VHDL code is not completely correct. Authors of [26] propose a methodology for generating a VHDL code via Array-OL and using an MDE approach. Several recent works propose the use of high level abstraction mechanisms based on UML for the VHDL code generation. In [27], a metamodel, which is strongly related to VHDL syntax, has been proposed for code generation. The design of embedded systems based on a high level modeling approach, has become more and more attractive. Designers can achieve higher levels of abstraction by focusing on a particular area, enjoying a form of visual expression that enhances the comprehensibility of systems. According to [28], any system can be viewed as a model. A model is an abstraction of a system made from a specific point of view.

In [29], authors describe an approach for high-level synthesis that can be used for VHDL code generation from UML behavioral models. It is based on the transformation of UML models into a textual code by using a formal language called SMDL. The SDML code is

then translated into automata and finally this finite state machine is described in VHDL. Thus, this approach goes through several intermediate manual steps that could result in a code with some errors. This leads to a long production delay, without a real possibility of reuse.

In [30], an approach to automatically generate VHDL source codes from UML is presented. The GenERTiCA tool is used in this work to generate VHDL codes. The presented approach follows the Aspect-Oriented Model-Driven Engineering for Real Time systems (AmoDER-RT) flow. UML models are transformed to other formal models based on DERCS (Distributed Embedded Compact Specification). In the UML modeling step, the MARTE profile is used to annotate models with a suitable stereotype to specify real-time characteristics of some elements. In this work several model elements that represent the same element can be mapped in a single DERCS element. In the code generation step, the UML concepts are mapped to VHDL syntax. In [31], an approach to map UML to VHDL is presented. It is based on a simple translation of UML concepts into VHDL ones. Therefore, it could be a source of errors and not offer a real possibility of reuse.

These cited approaches are strongly related to an abstraction of the system using high-level models and without a real possibility of reuse. However, the drawback is that the translations from an abstraction level to a lower one are very tedious and error-prone. Different to these approaches, in our work we focus on modeling the NoC topologies by using the MARTE RSM package and generating the VHDL code of the interconnection between the ports of the routers of the NoC. The main originality is the compactness of the way we deal with the regularity in the NoC, taking into account the reuse and the genericity of the models.

3 Background

Our design framework is based on MDE techniques which allow to reduce the efforts of development and maintainability of design tools. At each abstraction level, the concepts and the relations between these concepts are specifically defined. This leads to managing the complexity of the tools by dividing the various compilation steps and by formally defining the concepts associated with each step.

We first present the Gaspard2 co-modeling environment, then the particular problem of NoC modeling and the RSM package that we will use to model the regularity in the NoC topologies.

3.1 Gaspard2 Co-modeling environment

The Gaspard2 tool set is provided as an Eclipse plug-in allowing the designer to define SoC and explore the design space based on synthesis, simulation and verification of automatically generated codes [34]. The entry point which corresponds to the high-level abstraction is a MARTE-compliant model. This kind of model is identified by the user with UML modeling tools such as Papyrus. The design of SoCs in Gaspard2 is particularly related to the repetitive Model of Computation (MOC) [35], which is an appropriate way to express the potential parallelism in a system. This MOC is inspired by a domain-specific language, ArrayOL [36], originally dedicated to intensive signal processing applications. It includes the basic notions of this language and is an attractive and expressive model to describe the task and the data parallelism, as well as a combination of both.

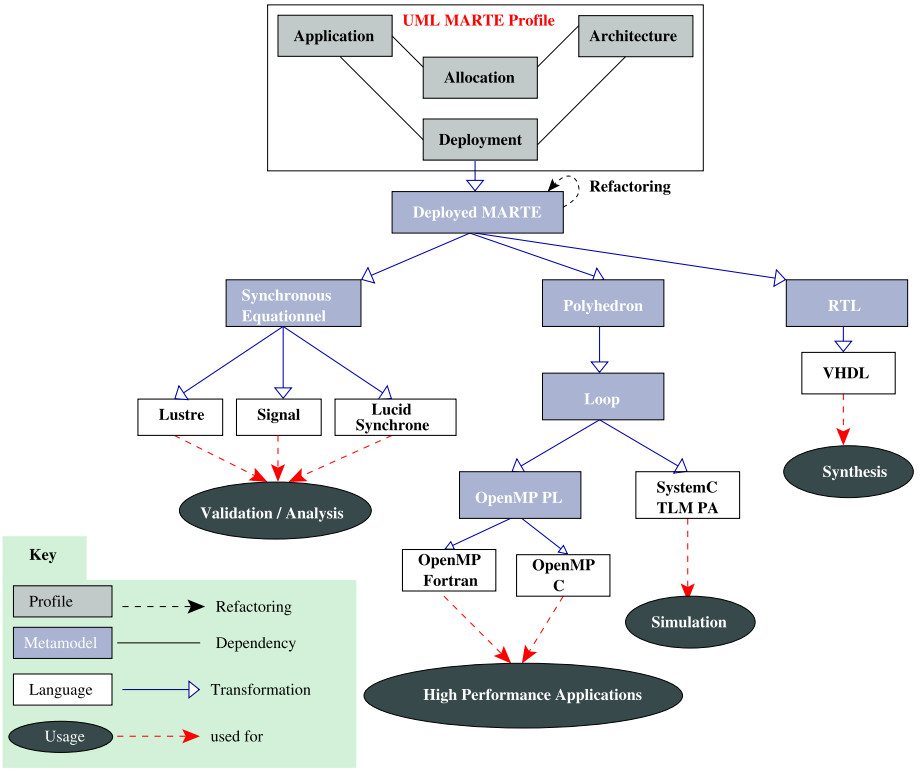


Fig. 1 Gaspard2 transformation chains

In Gaspard2, the use of the repetitive MOC involves the MARTE standard profile and more precisely its RSM package, to describe parallel computations in the application, parallel structure of the hardware architecture part and the association of both parts. It is a consequence of the UML SPT profile evolution [37] whose concepts are deduced from SysML profile [38]. Moreover, the resulting abstract models in Gaspard2 are improved with specific information related to the target technologies. Finally, according to the MDE paradigm, different automatic refinements from the highest abstraction levels to the lowest ones are defined. Various aims are targeted: hardware synthesis with VHDL [26], formal validation with synchronous languages [39], high-performance computing with OpenMP Fortran, C [40] and simulation at different abstraction levels with SystemC [41]. Gaspard2 partly relies on MARTE packages, with additional MDE corresponding to two concepts: metamodel and transformation. A metamodel is a collection of concepts and relations for describing a model. A model transformation explains how to produce a model conforming to a target metamodel from another model similar to a source metamodel. The target metamodel is usually more specific than the original one and manipulates concepts similar to the code. The transformation chain results from a successive application of several model transformations and allows code generation from high-level models. These chains have already been published in some articles [26, 39, 40].

Figure 1 shows an overview of these chains in Gaspard2. The top of this figure exhibits the high-level modeling concepts allowing the modeling of application, architecture, asso-

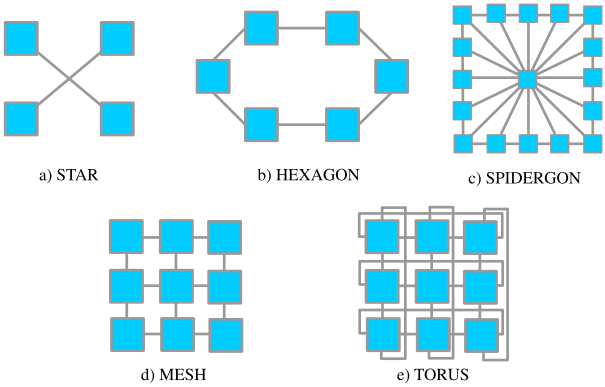


Fig. 2 Examples of standard topologies

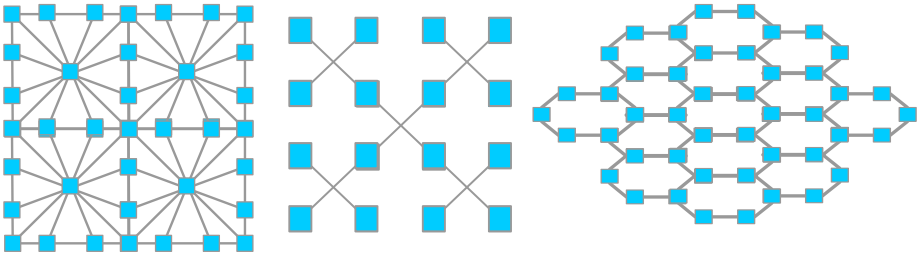
ciation, and deployment. Then, the bottom side of Fig. 1 defines different transformational chains, which transform the high-level models towards specific technologies. From a practical point of view, the generated code in synchronous language, e.g., SystemC, OpenMP Fortran, and VHDL, corresponds to the same high-level system model. This is ensured by the fact that all functional dependencies specified at the highest modeling level in Gaspard2, are entirely preserved by each transformational chain.

3.2 Network-on-chip characteristics

The NoC is based on the repetition of connected routers. Its concept provides an excellent groundwork for such standard interfaces. From a predictability perspective, the regularity of a NoC layout provides well characterized electrical and physical properties. The topology refers to the physical structure of the network graph, i.e., how networks are physically connected. It defines the connectivity or the routing possibility between nodes, thus having a fundamental impact on the network performance as well as the switch structure (e.g. the number of ports and port width). The tradeoff between generality and customization is an important issue to determine a network topology. The NoC topology can be described by a direct graph $NT(U, F)$ where each vertex $u_i \in U$ represents a node in the topology and the directed edge (u_i, u_j), denoted as $f_{ij} \in F$, represents a direct communication between the vertices u_i and u_j. Each topology can be characterized by a few properties.

The degree of a topology is the number of links connecting a router to its neighbor vertices. A topology is considered as regular when all routers have the same degree, if not it is considered as irregular. For example, Spidergon from STMicroelectronics, GEXspidergon NoC [16] and Honeycomb [9] are considered as irregular. Spidergon [33] is one of the NoCs developed by STMicroelectronics and is being proposed as an evolution of the STNoC. It is inspired from the Octagon topology. The 2D-mesh and the torus are the most used topologies due to their simplicity. The mesh consists of horizontal and vertical lines with nodes placed at their intersections. This specific structure is often used because the inter-node delay can be predicted at a high level. Figure 2 shows some of these topologies.

In this paper, the topologies are divided into two groups: standard topologies (e.g. Spidergon, hexagon, star, mesh and torus...) and hierarchical topologies shown respectively in Figs. 2 and 3. Hierarchical topologies (e.g. hierarchical star, honeycomb and GEXspidergon) consist of local and global network topologies where a local topology can be of any type of standard topologies.



a) GEXspidergon b) Hierarchical star c) Honeycomb

Fig. 3 Hierarchical topologies

3.3 MARTE repetitive structure modeling package

The repetitive structure modeling (RSM) is based on a Model of Computation (MoC) known as Array-OL with delays [13, 32] which aims at describing the potential parallelism in a system. RSM allows the description of the regularity of a system structure and topology in a compact manner. Formally, with MARTE RSM, an architecture is described as a set of tasks connected through ports, representing multidimensional arrays characterized by their shape (the number of elements) and their direction. The tasks are equivalent to mathematical functions reading data on their input ports and writing data on their output ports. The tasks are of three kinds: elementary, compound and repetitive. An elementary task is atomic (a black box), it can come from a library for example. A compound is a dependence graph whose nodes are tasks connected via their ports and it allows expressing task parallelism.

A repetition is a task expressing how a single sub-task is repeated; each instance of the repeated task operates with sub-arrays of the inputs and outputs of the repetition. Making the repetitions independent and therefore parallel by construction, it allows expressing data parallelism. For a given input or output, all the sub-array instances have the same shape, are composed of regularly spaced elements and are regularly placed in the array.

The RSM constructions can be used to compactly model repetitive architectures. It is especially useful for architectures with a large number of identical components, like a repetition of routers, where a compact representation both simplifies the modeling stage and concretely identifies the components with identical properties.

The available modeling mechanisms in RSM are directed towards two aspects. The first aspect is that RSM helps specify the shape of a repetition using a multidimensional form and allows the representation of the potential instances as a multidimensional array. This repetition can be specified for an instance or a port of a component. RSM also provides a mechanism for expressing the potentially complex topologies of the links between these arrays of ports or instances. Complex, regular and repetitive structures such as cubes and grids can be modeled easily in a compact manner via the RSM package. We will show later how to model the complex NoC topologies presented in the previous section. Figure 4 shows the basic core concepts of the RSM package.

A shape can be specified for an instance of a repeated component. For example a shape of $2, 4$ indicates that the task or component is repeated 2×4 times. This concept is also called the repetition space of this task or component. The shape concept is inspired from the bound notion of the parallel nested loops which are present in high-performance computing languages.

The tiler concept represents the mathematical expression of the elements of the patterns as tiles of the array, and is composed of: a fitting matrix F, whose column vectors represent

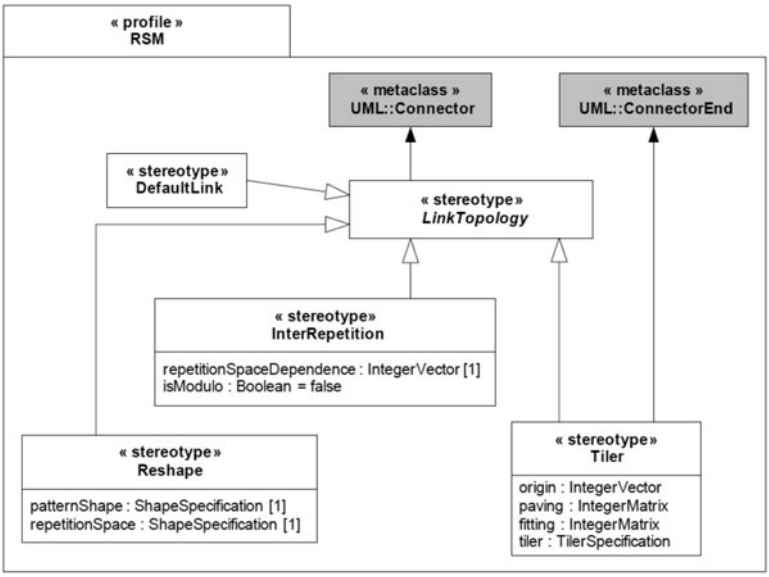


Fig. 4 RSM package of the MARTE profile

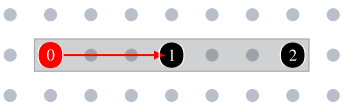
the regular spacing between the elements of a pattern in the array; **o**, the origin of the reference element (for the reference repetition) and a paving matrix P, whose column vectors represent the regular spacing between the patterns.

From a reference element (**ref**) in the array, one can extract a pattern by enumerating its other elements relatively to this reference element. The fitting matrix is used to compute the other elements. The coordinates of the elements of the pattern ($\mathbf{e}_i$) are built as the sum of the coordinates of the reference element and a linear combination of the fitting vectors as follows:

$$\forall \mathbf{i},\ \mathbf{0} \leq \mathbf{i} < \mathbf{s}_{\text{pattern}}, \quad \mathbf{e_i} = \mathbf{ref} + F.\mathbf{i} \bmod \mathbf{s}_{\text{array}} \tag{1}$$

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

In the following example of fitting matrix and tile, the tile is drawn from a reference element in a 2D array. The array elements are labeled by their index in the pattern, **i**, illustrating (1). The fitting vectors constituting the basis of the tile are drawn from the reference point.

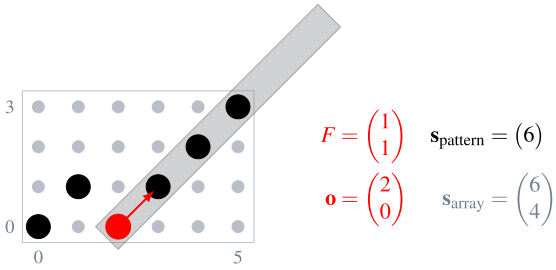


$$F = \begin{pmatrix} 3 \\ 0 \end{pmatrix} \quad \mathbf{s}_{\text{pattern}} = (3)$$

There are here 3 elements in this tile because the shape of the pattern is 3. The indices of these elements are thus (0), (1) and (2). Their positions in the tile relatively to the reference point are thus $F.(0) = (0, 0)$, $F.(1) = (3, 0)$, $F.(2) = (6, 0)$.

A key element one has to remember when using Array-OL is that all the dimensions of the arrays are toroidal. This means that all the coordinates of the tile elements are computed modulo the size of the array dimensions. The following more complex examples of tiles are

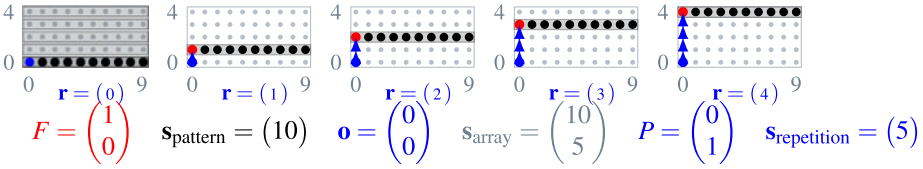
drawn from a fixed reference element (**o** as origin) in fixed size arrays, as illustrated once again in 1.



For each repetition, one needs to design the reference elements of the input and output patterns. A similar scheme as the one used to enumerate the elements of a pattern is used for that purpose. The reference elements of the reference repetition are given by the origin vector, **o**, of each tiler. The reference elements of the other repetitions are built relatively to this one. As above, their coordinates are built as a linear combination of the vectors of the paving matrix as follows:

$$\forall \mathbf{r},\ \mathbf{0} \leq \mathbf{r} < \mathbf{s}_{\text{repetition}}, \quad \mathbf{ref}_{\mathbf{r}} = \mathbf{o} + P.\mathbf{r} \bmod \mathbf{s}_{\text{array}}, \tag{2}$$

where $\mathbf{s}_{\text{repetition}}$ is the shape of the repetition space, P the paving matrix and $\mathbf{s}_{\text{array}}$ the shape of the array. Here is an example.



This figure represents the tiles for all the repetitions in the repetition space, indexed by r. The paving vectors drawn from the origin o indicate how the coordinates of the reference element $\mathbf{ref}_{\mathbf{r}}$ of the current tile are computed. Here the array is tiled row by row.

We can summarize all these explanations with one formula. For a given repetition index $\mathbf{r}$, $\mathbf{0} \leq \mathbf{r} < \mathbf{s}_{\text{repetition}}$ and a given index $\mathbf{i}$, $\mathbf{0} \leq \mathbf{i} < \mathbf{s}_{\text{pattern}}$ in the pattern, the corresponding element in the array has the coordinates $\mathbf{o} + (P\ F).\begin{pmatrix} \mathbf{r} \\ \mathbf{i} \end{pmatrix} \bmod \mathbf{s}_{\text{array}}$, where $\mathbf{s}_{\text{array}}$ is the shape of the array, $\mathbf{s}_{\text{pattern}}$ is the shape of the pattern, $\mathbf{s}_{\text{repetition}}$ is the shape of the repetition space, and **o** are the coordinates of the reference element of the reference pattern, also called the origin.

Inter-repetition dependence (IRD): The inter-repetition dependence connects an output port (pout) of a repeated component with one of its input ports (pin). The shapes of the connected ports must be identical. The inter-repetition dependence connector is tagged with a dependence vector **d** that defines the dependence distance between the dependent repetitions.

For example, the tiles are single points. The uniform dependence vector $\mathbf{d} = (1)$ tells that each repetition **r** depends on repetition $\mathbf{r}_{\text{dep}} = \mathbf{r} - \mathbf{d} = \mathbf{r} - 1$. In this example the inter-repetition dependence is used to express that the output value of a repetition is used as input by the following repetition. Each repetition will take, as input, two values on its two ports, an input value from the tile and the result of the previous repetition. These values are added

and provided on the output port which will serve as an input for the next repetition, and so on. To start the process, a default value is provided via a default link.

Another simple example for the applicability of such a construction is a two dimensional repetition (grid, Mesh topology) space for which repetitions can have different default values in function of the direction in which we exit the repetition space (north, south, east or west).

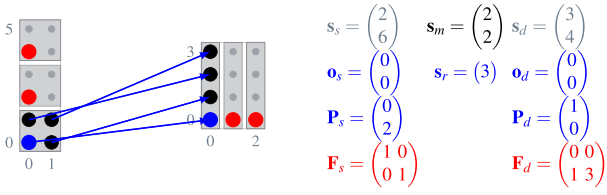
Reshape: This stereotype extends the UML connector metaclass and is equivalent to two tilers acting on the same pattern, one indicating how to fill the tiles from the source array and the other one showing how to map these tiles in the destination array. Its purpose is to allow the representation of more complex topologies in which the elements of a multidimensional array are redistributed or reorganized in another array.

A Reshape connector represents a relationship between the points of two multidimensional arrays. It is composed of four elements: a repetition space $\mathbf{r}$, the shape of the pattern $\mathbf{m}$, and two tilers. Each of the tilers is composed of a vector $\mathbf{o}$, called origin, and two matrices F and P for fitting and paving. The reshape connector, as so defined, describes a mathematical relation $\Re$, between the points of the source (A_s) and destination (A_d) arrays. Consequently, two points $p_s \in A_s$ and $p_d \in A_d$ are in relation $p_s \Re p_d$ if and only if there exists a repetition index, $\mathbf{r} \in \mathbb{N}^{d_r}$, $\mathbf{r} < \mathbf{s_r}$, and an index in the pattern, $\mathbf{i} \in \mathbb{N}^{d_m}$, $\mathbf{i} < \mathbf{s_m}$, which designates them via their respective tilers:

$$p_s \Re p_d \quad \Leftrightarrow \quad \exists \mathbf{i} \in \mathbb{N}^{d_m},\ \mathbf{i} < \mathbf{s_m},\ \exists \mathbf{r} \in \mathbb{N}^{d_r},\ \mathbf{r} < \mathbf{s_r},$$
$$\begin{cases} p_s = \mathbf{o_s} + (P_s \quad F_s) \begin{pmatrix} \mathbf{r} \\ \mathbf{i} \end{pmatrix} \bmod \mathbf{s_s} \\ \text{and} \\ p_d = \mathbf{o_d} + (P_d \quad F_d) \begin{pmatrix} \mathbf{r} \\ \mathbf{i} \end{pmatrix} \bmod \mathbf{s_d} \end{cases} \tag{3}$$

Where A_s (A_d) represents a multidimensional array, which is characterized by its dimension $d_s(d_d) \in \mathbb{N}^{d_m}$, and its shape $\mathbf{s_s}(\mathbf{s_d}) \in (\mathbb{N} \cup \infty)^{d_s(d_d)}$.

Here is a simple example of such a redistribution of all the elements of a 2×6 array into a 3×4 array.



Note that, in principle, such a reshape relation can relate only a part of the elements of the source array with part of the elements of the destination array and that the source array can be the same as the destination array. We will use this possibility a lot in the following.

4 NoC modeling methodology

Designing an efficient NoC architecture, while satisfying the application performance constraints, is a complex process. The design issues require several abstraction levels, ranging from high level modeling to physical layout level. Among the important phases in the NoC design is the choice of topology. This paper focuses mainly on this concept.

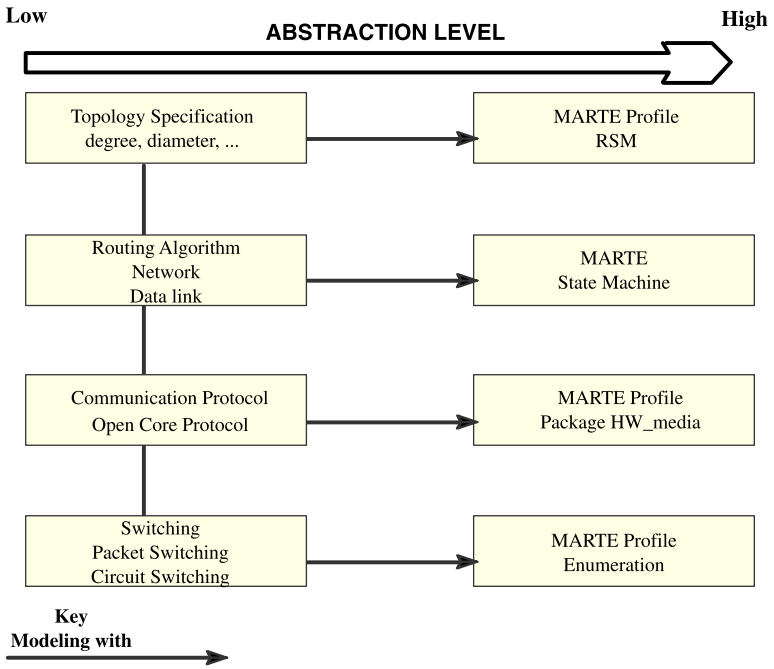


Fig. 5 Design flow for NoC modeling

4.1 Methodology flow

In this paper, we focus on the modeling of some complex topologies for evaluating the MARTE profile, validate the methodology, clarify some notations and generate the VHDL code of the topology. Topologies can be broadly classified into two main categories: standard and hierarchical topologies, as presented in Sect. 3. The construction of a hierarchical topology from a basic one, which is purported regular, can create some problems with the regularity of the constructed architecture. Therefore we define a new concept, which is a globally irregular locally regular (GILR) topology such as GEXspidergon and honeycomb depicted in Fig. 3. In order to model complex NoC systems, we propose the design flow shown in Fig. 5. This flow will be able to model all the NoC concepts. This work focuses only on the topology.

Our aim is to explain the phase of designing NoC at a high level of abstraction and help designers use tools in the design of a system. This methodology can be expressed in two phases. In the first phase, we begin by detailing concepts relating to NoCs, such as mathematical studies of topology, routing algorithm, communication protocol and switching technique. In the second phase, we specify the package we will use and make the relation between hardware architecture concepts and the notations that allow the modeling via MARTE profile.

Several topologies have been modeled in MARTE profile such as hypercube, star and ring [42]. In this paper we will divide the work into two parts. The first part is the modeling of topologies (e.g. Mesh, Torus, GEXspidergon and Honeycomb), whose objective is to explain some notations for modeling topologies with the help of the RSM package of the MARTE profile. This can be achieved with a never published before use of the reshape stereotype.

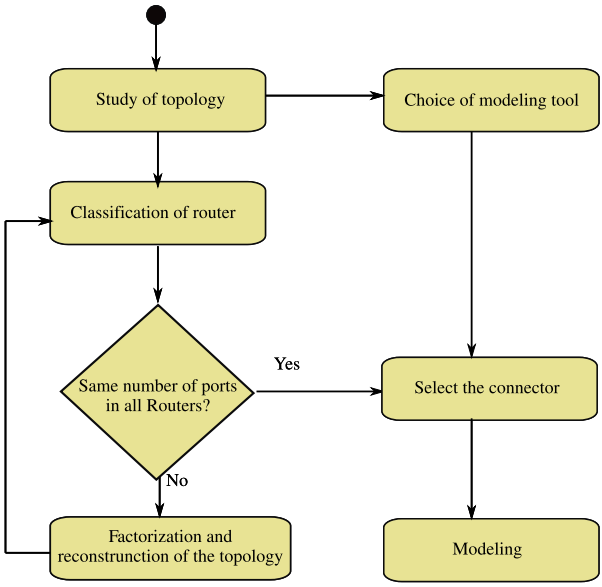


Fig. 6 Methodology for modeling NoC topologies

The complexity of modeling in MARTE appears when the topology is not perfectly regular or hierarchical. So it is difficult to extract regularity for modeling this topology in a compact manner. The modeling method is described as follows:

1. Choose the modeling tool which is Gaspard2 in this work.
2. Identify the topology by extracting some information of the graph such as the degree, the valence, the number of links, and the number of input/output ports.
3. Classify routers by their number of ports, factorize routers if it is possible and provide a representative for each type of routers.
4. Specify the type of the connector that exists between routers and which can be a Reshape, an inter-repetition dependence or a Tiler.
5. Modeling, which can constructed in a compact manner after selecting the connector.

The activity diagram in Fig. 6 explains our methodology for modeling all topologies; regular or globally irregular locally regular.

After modeling in this high level of abstraction, a second part of the work proposes a view for the construction of the notation used in the modeling, which appears important for the automation of the VHDL code generation. This will be explained in the next section.

4.2 Case study

In this section we present some modeling examples of NoCs that can be described via the MARTE profile. In particular, we take advantage of the RSM package since several topologies can be modeled in the MARTE profile with the help of this package. We validate the proposed methodology with some examples of NoCs, from the simple mesh and torus topologies to more complex topologies, namely GEXspidergon and Honeycomb.

4.2.1 Modeling of the mesh and torus topologies

A mesh topology consists of m columns and n rows. The addresses of routers and resources can be easily defined as $x-y$ coordinates in a mesh or torus topology. These two types of

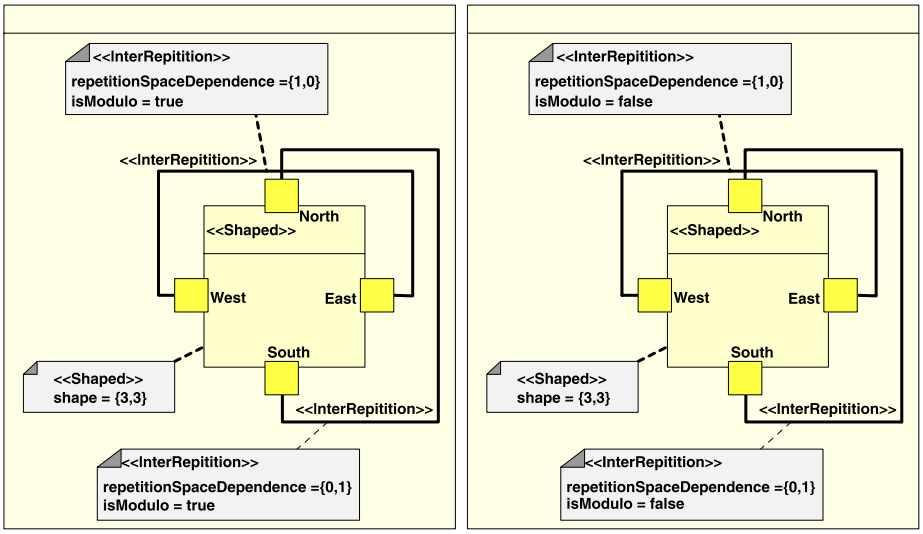


Fig. 7 Compact modeling of Torus (*left*) and Mesh (*right*) topologies: one component and inter repetition dependences

topologies are often proposed in the literature in order to represent topologies for on chip networks. For modeling mesh and torus topologies, we use the IRD concepts allowing the construction of complex dependences like NoC topologies. The connector can be connected to a port of the component containing the inter-repetition dependence and then we can establish a connection between dependences and connections between ports with different shapes. The mesh and torus topologies are good examples to demonstrate the applicability of these concepts. The construction is based on two-dimensional repetition space in which repetitions can have different values.

Therefore, the RSM dependencies allow the modeling of the mesh or torus in a compact manner. The shape of $(3, 3)$ on the router component illustrates that this component is repeated 9 times. The value of $(1, 0)$ is related to the "repetitionSpaceDependence" vector to determine the dependency between the repetitions. In the torus topology all kinds of ports (East, West, North, and South) establish a connection with all other ports of other routers, unlike that of the mesh topology. Thus, to express a full connection of the topology, "modulo" information must be true. Figure 7 shows the modeling of mesh and torus topologies.

4.2.2 Modeling of the honeycomb topology

The honeycomb topology, based on a hexagonal plan tessellation, is considered as a hierarchical interconnection network. It is based on a combination of hexagonal routers. This topology is used in literature for various applications such as cellular phones. The honeycomb is obtained by joining the center of each triangle in the hexagonal mesh with the center of the neighboring triangle. The hexagon is a honeycomb of size 1. A honeycomb of size 2 is obtained by adding six hexagons to the boundary edge of the size 1 honeycomb. Therefore, a honeycomb of size m is obtained from size $m - 1$ honeycomb. The concerned topology is formed by a repetition of a single router element. Each potential instance of this element is connected to the other potential instances of the same element. In our case, each instance

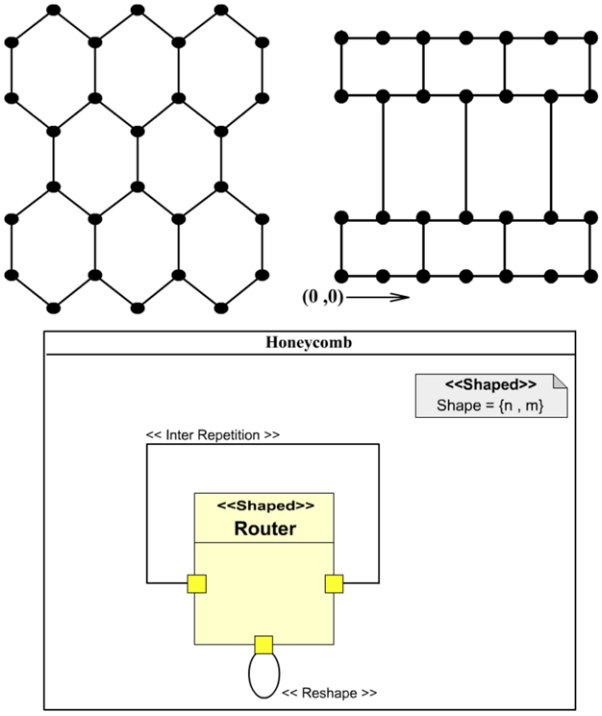


Fig. 8 Modeling of a honeycomb topology using a single component

is connected to the neighbors that are located at north, south, east and west sides. The inter repetition topology enables to specify the position of every neighbor of every potential instance of a model element with a multidimensional shape. In this architecture, it is worth mentioning that it is the first time in literature that Reshape is used on a connector connecting two ports of the same component (part in the UML terminology). It should be noted that this topology link specifies the set of the runtime links connecting the multidimensional array. It defines the tiling of the arrays by an identical pattern. Figure 8 shows the modeling of a honeycomb topology.

The proposed mathematical models for reshape and inter repetition are described below:

- The InterRepetition describes the horizontal connections with a repetitionSpaceDependence $= (1, 0)$
- The Reshape describes the vertical connections. Its attributes are
  - patternShape $= ()$, denoting a point
  - repetitionSpace $= \binom{n/2}{m-1}$
  - sourceTiler $= \{\text{origin} = \binom{0}{0}, \text{fitting} = (), \text{paving} = \binom{2\ 1}{0\ 1}\}$
  - destinationTiler $= \{\text{origin} = \binom{0}{1}, \text{fitting} = (), \text{paving} = \binom{2\ 1}{0\ 1}\}$

The reshape has been built using a tile of a single point, hence the empty (of dimension 0) patternShape and fitting matrices. The origins of the tilers express the connection between the router of index $\binom{0}{0}$ and the router of index $\binom{0}{1}$, i.e. the lower left vertical connector. All the other connectors can be deduced from this one by a translation of vector $\binom{2}{0}$ for the first row and by a translation of vector $\binom{1}{1}$ to reach all the rows with an horizontal shift of 1. These two vectors are the column vectors of both paving matrices and the number

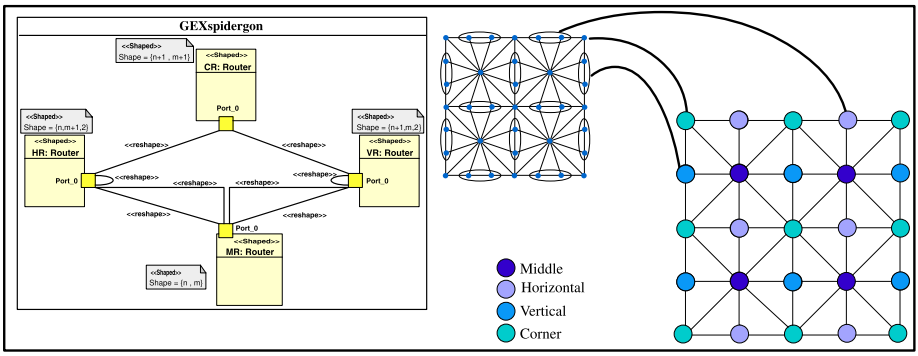


Fig. 9 Compact modeling of the GEXspidergon topology: one component by router degree and reshape connectors

of repetitions of the translations in the repetitionSpace, $\binom{n/2}{m-1}$, where n is the number of routers in a row and m the number of rows.

Through this topology, we have shown first how to use the reshape connector to connect two ports of the same part, and secondly that the RSM package is used for modeling this topology in a compact manner, which enhances its applicability.

4.2.3 Modeling of the GEXspidergon topology

The GEXspidergon can be seen as a hierarchical topology. It is an academic topology for NoC. This study presents a generic NoC architecture based on a configurable and generic router. This topology is constructed using an elementary polygon network, which is a combination of the star and the ring architectures. This elementary network is composed of $4R + 1 (R = 1, 2, \ldots)$ routers including a central router, which is connected with the $4R$ peripherals routers via point to point links. The peripheral routers are connected to each other in the form of a ring. The elementary network is characterized by its valence ($m = 4R$) that represents the number of the peripheral routers. These routers need $2m$ links to be connected to the central router. Each peripheral router is connected to 4 input/output ports and the central router is connected to $m + 1$ input/output ports.

In our case the GEXspidergon is irregular. Thus, the routers of this topology are categorized into four groups in this analysis:

- Corner Routers.
- Middle Routers.
- Horizontal Routers, package of two.
- Vertical Routers, package of two.

The modeling of the GEXspidergon topology is provided in Fig. 9. It has been carried out in a compact manner with the help of the reshape stereotype between arrays of different shapes.

When expressing the link between vertical/horizontal routers and middle routers, we have had to use two reshapes since the considered tiles are not formed by regularly spaced points. Indeed, for a given reshape, the points of the tile must be regularly spaced because they are built from the reference point of the tile by a linear combination of the column vectors of the fitting matrix. We have found that it's possible to use two reshapes in order to express the link topology and hence we can extract the regularity. The description stays compact because it uses a constant number of modeling elements to model networks of any size.

For space reason, we will only detail one of the reshapes of this model: the reshape from the middle routers to the vertical ones on their right (the other reshape between those two kinds of routers models the connections from the middle routers to the vertical routers on their left).

The shape of the source array is $\binom{n}{m}$ for $n \times m$ cells in the network and one middle router per cell. The shape of the destination array is $\binom{n+1}{m}{}_{2}$ for $(n+1) \times n$ sets of 2 routers on the vertical sides of the cells.

The attributes of the reshape are

- patternShape = (2)
- repetitionSpace = $\binom{n}{m}$, for one set of links per cell
- sourceTiler = {origin = $\binom{0}{0}$, fitting = $\binom{0}{0}$, paving = $\left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)$}
- destinationTiler = {origin = $\begin{pmatrix}1\\0\\0\end{pmatrix}$, fitting = $\begin{pmatrix}0\\0\\1\end{pmatrix}$, paving = $\left(\begin{smallmatrix}1 & 0\\ 0 & 1\\ 0 & 0\end{smallmatrix}\right)$}

The reshape has been built using a tile of two points denoting the two routers on the vertical side to the right of a cell. The source fitting matrix expresses that these two vertical routers are both connected to the same middle router. The destination router is iterated on the third dimension of the destination array, namely the one that corresponds to the routers on the same vertical side of a cell. Thus, depending on the connection end of the reshape, the tile represents the middle router of a cell or the two vertical routers on the right side of this same cell.

The origins of the tilers express the connection between the middle router of the cell $\binom{0}{0}$ and the vertical routers of index $\begin{pmatrix}1\\0\\0\end{pmatrix}$ and $\begin{pmatrix}1\\0\\1\end{pmatrix}$, i.e. the right vertical routers of the cell $\binom{0}{0}$. All the other connectors can be deduced from this one by a translation of the vector $\binom{1}{0}$ for the first row and by a translation of the vector $\binom{0}{1}$ to reach all the rows. These two vectors are the column vectors of the source paving matrix and they are completed by a 0 in the third dimension of the destination router. Finally the number of repetitions of the translations in the repetitionSpace is $\binom{n}{m}$, where n is the number of routers in a row and m the number of rows to enumerate all the cells.

Through all these case studies we have shown that we can model, in a compact manner, irregular topologies (with routers of different degrees) with RSM by extracting the spacial regularity that exists in the topology.

5 Model to VHDL code generation

5.1 Transformation chain from MARTE to VHDL

The first step in each transformation chain of Gaspard2 is always the same and it consists in a transformation from the UML + MARTE profile metamodel to the deployed metamodel. Indeed this metamodel corresponds to an intrinsic definition of the concepts identified in the MARTE profile. A chain of transformations then targets an RTL metamodel from which the VHDL code can be generated (see Fig. 10).

Usually, each accelerator must be customized separately without a possible reuse. This leads to long production delays and high design costs. Using this transformation chain, it is possible to automatically produce hardware architectures that solve these two issues and reduce the errors resulting from human intervention. The generation of a hardware accelerator relies on the hardware-software partitioning specified in high-level MARTE models. The

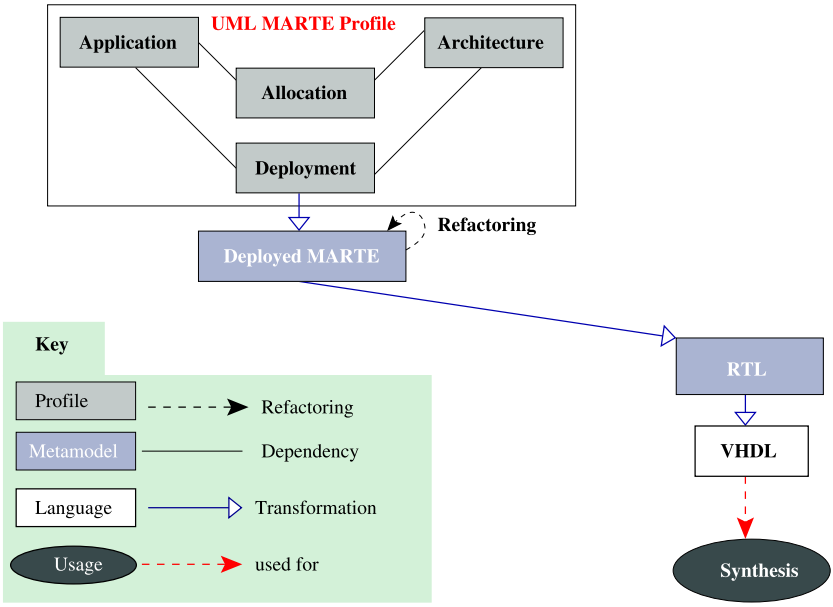


Fig. 10 The MARTE2RTL model transformation in the Gaspard2 environment

RTL metamodel, defined in the RTL package of Gaspard2, collects the necessary concepts to describe hardware architecture at the RTL (Register Transfer Level) level, which allows the hardware execution of functionality. The RTL metamodel is not related to any Hardware Description Languages (HDL) such as VHDL or Verilog. However, it is precise enough to enable the generation of a synthesizable VHDL code.

As shown in the previous section, the RSM package includes some information such as the shape of the pattern and the number of repetitions of the pattern or the tile. In this section, we present a matching semantic of the concepts of the RSM package in VHDL. In model driven engineering, a code generation is usually done from a metamodel that is independent of any concrete syntax. In our tool, the RTL metamodel is indeed independent of any HDL syntax.

The MARTE2RTL transformation converts this model into two output models: an RTL model and an RTL port type model. The basic information at the RTL model are the concepts related to the hardware accelerator. A repetitive task is transformed into a hardware repetitive component. In order to convert the deployment information presented in the MARTE model, the RTL model also contains equivalent concepts which help the code generation. Actually the code generation is carried out using model-to-text transformation. Based on the MDE concepts, the code generation is viewed in textual form. Therefore, it can be viewed as one-to-one transformation, in which each concept in the model generates a certain part of the overall text. The RTL model is not a direct input of synthesis or simulation tools contrary to the output generated code. Thus, by taking advantage of our design flow, the code generation can be directly performed from the RTL model. The abstraction level of the RTL model is sufficient enough for the generation of HDL that can be VHDL or Verilog. One of the objectives of our design flow is the generation of a correct and synthesizable VHDL code that can be implemented on ASIC or FPGA technologies. In hardware design, a component represents a hierarchy level and describes a specific functionality. In our case, the router component is instantiated in order to allow the use of such a functionality in a NoC

component. The communications between components are established with the help of the interfaces which are composed of ports. A port can be a receiver or a sender, an input or an output.

Figure 11 illustrates a set of concepts in the Hw_metamodel that can be used to model the component at RTL level. The component concept contains a clock and a reset as input ports, and four ports as in/out ports. Hardware router contains an arbitrary number of ports. Hw_port contains information on the organization of the data and their types. The Hw_shape defines the dimension of the port by means of the 'dim' link. The generated code from the Hw-metamodel is executed within a template in the objective to find concepts it is associated with. The template contains a VHDL syntax related to the Hw_component.

In our case study, the operation performed by IRD can be considered as being equivalent to a "port map" instruction used in VHDL architecture description. This allows the development of a VHDL implementation of the repetitive structure based upon its RSM description.

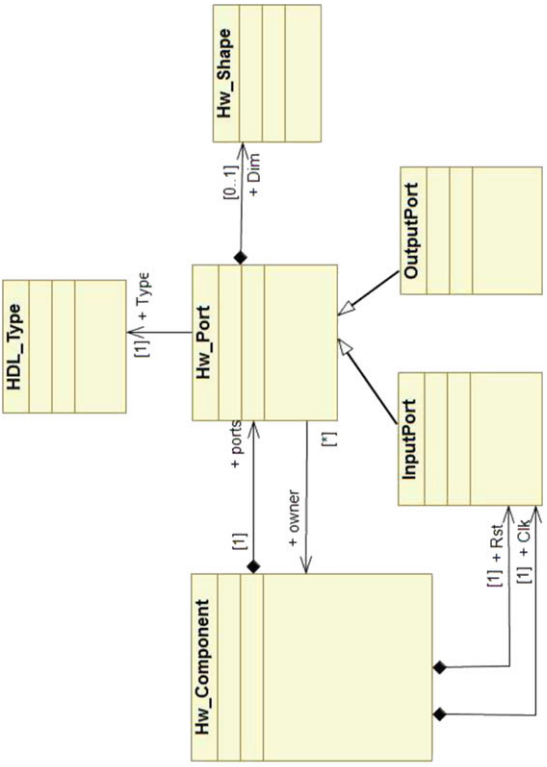
To generate the required VHDL code of the top level architecture of the topology, we first require a template. It contains information used in the modeling. This will provide a library of templates for Gaspard2.

The following code shows the generic VHDL implementation of the top level of the repetitive router referred to (x, y) direction. The architecture structure is completely generic. The parameters of the repetition are referenced from the declaration in the entity.

```
--Generic VHDL implementation of the repetitive structure
entity Repetition is
Generic (ArraysizeinXdirection: positive:3;
ArraysizeinYdirection: Positive:3;
repetition: positive:9
sizeofport: positive:32
);
port(Clk,Rst, in std_logic; s: out std_logic_vector(0 to
    repetition-1));
end Repetition;

architecture Behavioral of Repetition is
type tab_3_3 is array (0 to ArraysizeinXdirection-1)
of std_logic_vector(0 to ArraysizeinYdirection-1);
signal tab_h, tab_v: tab_3_3;
signal clk,rst:std_logic;
component router port(Clk,Rst,West,East, North,South:
inout std_logic);
end component;
begin
generate1: for i in 0 to repetition-1 generate
generate2: for j in 0 to reprition-1 generate
dependancecyclique: router port map (Clk,Rst,tab_h(i)(j),
    tab_h(i)((j+1)mod 3),
tab_v((i+1)mod 3))(j),tab_v(i)(j));

end generate generate1;
end generate generate2;
end Behavioral;
```



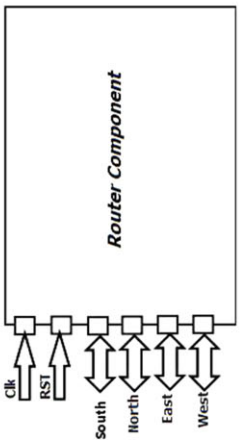


Fig. 11 Entity of the hardware router and related metamodel concept

The repetition is carried out by an iterative port map instruction of the elementary component. The number of iterations is derived from the number of required repetitions. The architecture of the component, which is a router in our case study, is described in its VHDL file. The structure of the code will now be explained in stages.

Entity The VHDL code is a description of a hardware component. The first part of this code describes the appearance of the component. The entity allows the declaration of the in/out ports, their size and generic parameters. The description of the ports and the generic parameters will be derived from numeric values defined in the modeling.

Architecture The architecture describes the construction of the top level and its operation. It defines the use of the elementary component within its structure with the necessary signals required to establish the interconnection between each component. A component is declared as a reference to an existing entity by the same parameters, i.e. the same name and its ports in exactly the same way. For example to use an elementary component like router, in our case, a component called router is defined as:

```
Component Router port( Clk,Rst,West,East,North,South:
inout std_logic_vector);
end component;
```

The input/output ports are of the same name, type and size as these declared in its entity. After the declaration of the component and the required signals, the mapping of the repetitive component can be achieved. Thus, the architecture describes how the repetitive structure is built by mapping ports of the repeated elementary component. The following code explains this process.

```
generate1: for i in 0 to repetition-1 generate
generate2: for j in 0 to repetition-1 generate
dependancecyclique: router port map(Clk, Rst, tab_h(i)(j),
tab_h(i)((j+1) mod 3), tab_v((i+1) mod 3)(j),tab_v(i)(j));
end generate generate1;
end generate generate2;
```

Each increment of i,j instantiates the next elementary component, in both directions. The generate instruction allows the repetition and provides the reference of the ports that are to be mapped to that repetition. The input/output ports are mapped using the intermediate signals which represent the links. This includes the description of the hardware architecture by the description of the VHDL code of the entire structure.

5.2 Torus and mesh examples

As an example, we demonstrate the relationship between the IRD description and synthesizable VHDL code for torus and mesh topologies.

```
--Behavioral Torus
architecture Behavioral_Torus of Torus is
--Array that contains links
type tab_3to3 is array (0 to 2) of std_logic_vector (0 to 2)
signal tab_h, tab_v:tab_3to3;
```

```
--component declaration
component router is
port(Clk,Rst: in std_logic; North, South, West, East:
inout std_logic_vector(31 downto 0));
end component;
Begin
--instantiation in both direction
GenerateX: for i in 0 to 2 generate
GenerateY: for i on 0 to 2 generate
dependancecycle: router port map(Clk, Rst,tab_h(i)(j),
tab_h(i)((j+1)mod 3),
tab_v((i+1)mod 3)(j),tab_v(i)(j));
end generate GenerateX;
end generate GenerateY;
end Behavioral_Torus;
```

The torus topology is obtained by only one type of router. The difference between the two topologies modeling (Mesh and Torus) appears in the use of the "modulo" concept. The tagged value of this concept is "false" in mesh topology, and hence ports of the routers are not all connected. That's why we must describe each router as a specific component. Furthermore, to obtain a synthesizable code of NoC-based UML/MARTE concepts we must extract all the information that describes the topology. As is mentioned in the previous section, for modeling torus or mesh topology we have instantiated routers in both directions (x, y) with the help of two generate instructions respecting the VHDL semantics. The communication between ports is achieved by making a link and using a port map instruction.

To obtain the VHDL code of the mapped NoC topology of the mesh, we can apply the methodology detailed in Sect. 4. In fact, this methodology facilitates the generation and the mapping, thus we can classify the routers into three kinds for a (N, N) mesh topology:

1. Four routers that have 2 ports;
2. $N.(N-2)$ routers that have 3 ports;
3. $(N-2).(N-2)$ routers that have 4 ports.

The verification of the VHDL code generated in this study has been achieved by simulation and it is validated by synthesis results. The following section explains this.

5.3 Case of complex topologies

As an example of complex topologies that use a Reshape and IRD connectors, we demonstrate the generated VHDL code of the Honeycomb topology.

The Honeycomb topology is obtained by only one type of router. We notice that in such case it is possible to integrate the information which are extracted from the mathematical model of the reshape, in the generate statement. We showed in the generated VHDL code of the Honeycomb that the reshape gives the index of connected or non-connected routers, the number of routers in a row and the number of rows. In this example, this information is combined with the generalized VHDL code of the "Inter-Repetition" that is presented in previous section. The generate statement describes the horizontal connections with a repetitionSpaceDependence $= (1, 0)$ and the reshape describes the conditional vertical connections. This is transformed by the help of a conditional generate statement. The following code shows an extract of the generated VHDL code of the Honeycomb topology.

```
component router is
port
(P1 :inout std_logic; P2 :inout std_logic; p3:inout std_logic );
end component ;
begin
G1: for i in 0 to 6 generate
-- instantiation of connected routers
-- index of connected routers
gen1: if ((i=0) or (i=2) or (i=4) or (i=6)) generate
--- connection between routers of row 1
row1: router port map(links(i)  ,links(i+1), sig(i));
--  connection between routers of row 2
row2: router port map(links2(i) ,links2(i+1), sig(i));
--  connection between routers of row 3
row3: router port map(links3(i) ,links3(i+1), sig2(i));
--  connection between routers of row 4
row4: router port map(links4(i) ,links4(i+1), sig2(i));
end generate gen1;
gen2: if ((i/=0) and (i/=2) and (i/=4) and (i/=6)) generate
row1: router port map(links(i)  ,links(i+1) , open) ;
-- index of connected routers in y direction
row2: router port map(links2(i) ,links2(i+1), sigrow2_3(i));
row3: router port map(links3(i) ,links3(i+1), sigrow2_3(i));
row4: router port map(links4(i) ,links4(i+1), open);
end generate gen2;

end generate G1;
```

5.4 Synthesis results of torus and mesh examples

5.4.1 Design metrics

One of the objectives of the design flow is the generation of a correct and synthesizable VHDL code. This code, which derives from the modeling, allows the verification of the connections of the topology with the help of the RTL schematic. When generating this RTL schematic from the synthesis tool, we can verify if the connections between routers are rightly established or not. In addition, it can be verified by simulation and validated by the design metrics. If connections between routers are not well established you must revised the UML MARTE model of the topology.

The generated code can be implemented with different commercial tools targeting ASIC or FPGA technologies. In the previous section, we have detailed the mapping of the repetitive architectures and the way in which we describe the dependencies in the VHDL description generated from our design flow. The obtained code has been successfully validated by the Modelsim tool and was synthesized with the use of the Xilinx ISE9.1. Figure 12 illustrates the synthesis results for the mapping of the torus and mesh topologies. This RTL schematic, which derives from the generated code and the MARTE model of these topologies, verifies that the model is correct.

The simulation is achieved by a simple scenario, in which we tested the transfer of parquets in the NoC.

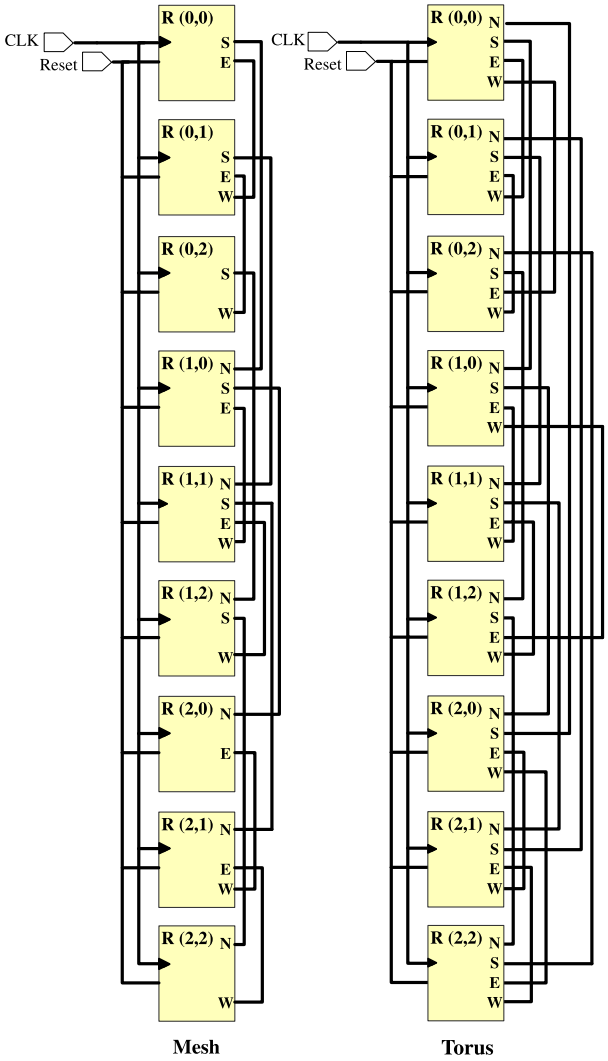


Fig. 12 RTL schematic of the synthesized codes

To evaluate the generated code in terms of design metrics, the used router is inspired from [43]. The NoC performances have been evaluated in terms of speed and area. Tables 1 and 2 show these results for a mapped torus and mesh topologies using our design flow. The generated code has been prototyped using the xc5vfx30t-2ff665 device of virtex5 FPGA technology. Based on a manual method to map the $N \times M$ torus topology we use $N \times M$ code lines. However, in our approach we use usually a two nested generate instruction to map any $N \times M$ mesh topology.

Taking into account the efficiency of GASPARD2 for modeling repetitive structure, this study has revealed that the generated code is more efficient in comparison to a manually written code. In fact, with a reduced code lines number the generated code provides the same performance as the manual approach in terms of hardware resources independently of the used topologies type.

Table 1 Synthesis results of the torus topology

Generated code	Manual code	
FPGA/Performances	Virtex5	Virtex5
Slice	9 %	9 %
Flip flop	35 %	35 %
Lut	23 %	23 %
Frequency (MHz)	100	100

Table 2 Synthesis results of the mesh topology

Generated code	Manual code	
FPGA/Performances	Virtex5	Virtex5
Slice	7 %	7 %
Flip flop	30 %	30 %
Lut	18 %	18 %
Frequency (MHz)	120	120

Results in Tables 1 and 2 are exactly the same for the manual and generated codes, because the generate instruction allows to express a set of port map instruction. So, for the synthesis tools, this is expressed by using the same hardware resources.

5.4.2 Design effort

Design effort is considered as the time required, in designer-hours, to design and implement a given system. Design effort is equivalent to design time when the project has a single developer. It is possible to minimize the required time by increasing the number of developers. However, increasing the number of designers may lead to decreases in the overall productivity per person. Hence, the solution could be the use of an environment to automate the code generation.

To measure the effort of general design, estimates of the effort for each repetitive component must be obtained and then added into required time to design system. However, components may be instantiated several times through any given design, which greatly impacts the time required for the design. Some components may also be parameterized, and different sized instances could be generated. To address these cases, used Gaspard2 environment and MARTE profile can provide the following two advantages.

Account for a single model of each component When a design reuses a model of component, we only count the design effort of one instance of it. In accordance with the principles of modular design, the model of component is designed and verified, it can be re-used elsewhere with negligible effort and time.

Parameterize a model To estimate the design effort we encourage the reuse of model by using a parameterized component. It facilitates access to the model and subsequently generating a new product with negligible effort time. As the values of the parameters may change, such as, the value of the component instance, it is not difficult to generate code for large topologies, rather than writing by hand. Moreover, with the help of the Gaspard2 flow we can easily change the generic parameters of our NoC, such as the number of nodes, or completely change the router architecture without modifying the UML/MARTE model.

Also, there are several metrics that may be related to design effort. Examples include the number of logic gates or the number of code lines in the design description. Consequently, in the design, we measure these metrics. In our case study, we showed that to map torus topology or mesh of size $N \times M$, the number of code lines is always reduced compared to code written by hand.

The reason is that, to map this topology an $N \times M$ lines of code are required. By exploiting the concepts of modeling, referred to the RSM package of MARTE profile, the code generated (illustrates in the paper) by GASPARED2 is reduced to four line of code, case of torus, for establish connection. When compared to a hand code we must write $N \times M$ lines of code to port map the topology (e.g. 100 lines for 10×10 Torus). In the case of complex topologies, usually the code generated is reduced because in our approach we use the generate instruction, in which we can integrate the concepts of MARTE model (Origin, Paving, Fitting) to express repetitions and connections between routers.

This gives the MARTE profile model an advantage of reducing the development costs, while at the same time minimizing the time-to-market.

6 Conclusion

System design based on a high level oriented approach for modeling becomes more and more attractive and important for the rapid development of SoCs. That's why designers need to increase the level of abstraction in order to address the problem of high cost of the layout and to decrease the time-to-market of SoCs. In this paper two contributions have been proposed: the first deals with the modeling via the MARTE profile and the second is the generation of the VHDL code of the corresponding hardware architectures.

We have started by presenting the modeling of NoC topologies; the repetitive structure modeling with MARTE has been successfully used to model some kinds of architectures. Several topologies have been proposed in the literature and we have shown that the RSM package of MARTE is powerful enough to model the simple regular topologies as well the more complex ones. To achieve this, we have proposed the first use of the reshape connector to connect some ports of the same part, enabling the description of regular but non uniform connections between repeated parts. The proposed modeling methodology enables to take full advantage of the regularity in the topology to factorize the model, and hence reduce the complexity of the modeling process. Such a factorization could then be used to generate a synthesis code and therefore tackle large networks with several hundreds or thousands of routers as easily as small ones.

We have also given a flow based on the Gaspard2 tool for automating the VHDL code generation from UML/MARTE models. This code has been successfully evaluated and validated for the design and the mapping of NoC topologies like mesh and torus.

The use of modeling concepts promotes a separation of concerns during the system design. Different aspects are distinguished: the functionality system, the hardware architecture, the mapping, and the deployment on target platforms. Through the modeling of NoCs topologies, we have also demonstrated how the already defined concepts are reused due to the component-oriented design adopted in our approach. This is very important in order to reduce time to market. The separation between the functionality system, the hardware architecture, and their mapping is also an important benefit of Gaspard2-based design. Besides, when switching from an architecture to another, the programmer needs only to modify the functionality part (functionality of the router in our case study). Then the designer can reuse the existing model for generating new code and this significantly reduces time to market.

References

1. Benini L, De Micheli G (2002) Networks on chips: a new SoC paradigm. IEEE Comput Soc 35(1):70–78
2. Dally WJ, Towles B (2001) Route packets not wires on-chip interconnection networks. In: Proceedings of the design automation conference (DAC01), pp 684–689
3. Sgroi M, Sheets M, Keutzer K, Malik S, Rabaey J, Vencentelli A (2001) Addressing the system-on-a-chip interconnect woes through communication-based design. In: Proceedings of the design automation conference (DAC01), pp 70–78
4. Ogras UY, Marculescu R (2005) Application-specific Network-on-chip architecture customization via long-range link insertion. In: Proceedings of the 2005 IEEE/ACM international conference on computer-aided design (ICCAD05), pp 246–253
5. Moraes F (2004) HERMES an infrastructure for low area overhead packet-switching networks on chip. Integration 38(1):69–93
6. Dall'Osso M (2003) Xpipes: a latency insensitive parameterized network on chip architecture for multi-processors SoCs. In: Proceedings of international conference on computer design, pp 536–539
7. Karim F, Nguyen A, Dey S, Ramesh R (2001) On-chip communication architecture for OC-768 network processors. In: Proceedings of design automation conference (DAC01), pp 678–683
8. Holsmark R, Kumar S (2005) Design issues and performance evaluation of mesh NoC with regions. In: Proceedings of NORCHIP conference (NORCHIP05), pp 40–43
9. Hemani A, Jantsch A, Kumar S, Postula A, Berg J, Millberg M, Lindquist D (2000) Network on a chip: an architecture for billion transistor era. In: Proceedings of NORCHIP conference (NORCHIP00), pp 40–43
10. OMG (2009) A UML profile for MARTE: modeling and analysis of real-time embedded systems, OMG adopted specification
11. OMG (2009) Modeling and analysis of real-time and embedded systems. Object Management Group. http://www.omg.org/spec/MARTE/
12. Demeure A, Lafage A, Boutillon E, Rozonelli D, Dufourd JC, Marro JL (1995) Array-OL: proposition d'un formalisme tableau pour le tratement de signal multi-dimensionnel. http://hdl.handle.net/2042/12353
13. Glita C, Dumont Ph, Boulet P (2010) ARRAY-OL with delays, a domain specific specification language for multidimensional intensive. Multidimens Syst Signal Process 21(2):105–131
14. DaRT Team LIFL/INRIA (2008) Graphical array specification for parallel and distributed computing (Gaspard). https://gforge.inria.fr/projects/gaspard2/
15. Elhaji M, Boulet P, Zitouni A, Tourki R, Dekeyser JL, Meftaly S (2011) Modeling networks-on-chip at system level with the MARTE UML profile. In: Proceedings of the model based engineering for embedded systems design (M-BED2011)
16. Zid M, Zitouni A, Baganne A, Tourki R (2009) Nouvelles architectures generiques de NoC. J Tsinghua Univ (Sci Technol) 28(1):101–133
17. Murali S (2009) Designing reliable and efficient networks on chips. Lecture notes in electrical engineering, vol 34. Springer, Berlin
18. Arteris. http://www.arteris.com/
19. INoC. http://www.inocs.com/
20. Silistix. http://www.silistix.com/
21. Dafali R, Diguet JP, Evain S, Eustache Y, Juin E (2007) Spider CAD tool: case study of NoC IP generation for FPGA. In: Proceedings of design and architectures for signal and image processing (DASIP07), pp 457–460
22. Moenner P, Perraudeau L, Quinton P, Rajopadhye S, Risset T (1996) Generating regular arithmetic circuits with ALPHARD. In: IRISA
23. Williamson MC (1996) Synthesis of parallel hardware implementations from synchronous dataflow graph specifications. In: Proceedings of the thirtieth Asilomar on signals, systems and computers, pp 1340–1343
24. MathWorks (2009) Simulink Hdl coder. www.mathworks.com/products/slhdlcoder
25. Wood SK, Akehurst DH, Howells WGJ, McDonald-Maier KD (2008) Array OL descriptions of repetitive structures in VHDL. In: Proceedings of the 4th European conference on model driven architecture, pp 137–152
26. LE Beux S, Marquet P, Dekeyser JL (2008) Model driven engineering benefits for high level synthesis. INRIA res rep 6615, INRIA 2008
27. Damasevicius R, Stuikys V (2004) Application of UML for hardware design based on design process model. In: Proceedings of the Asia and South pacific design automation conference (ASP-DAC04), pp 244–249

28. Favre M (2005) Foundations of model driven reverse engineering: models episode. I. Stories of the Fidus papyrus and of the Solarus, language engineering for model-driven software development. //drops.dagstuhl.de/opus/volltexte/2005/13
29. Bjorklund D, Lilius J (2002) From UML behavioral descriptions to efficient synthesizable VHDL. In: Proceedings of the IEEE NORCHIP conference
30. Moreiral TG, Wehrmeister MA, Pereira CE, Ptin JF, Levrat E (2010) Generating VHDL source code from UML models of embedded systems. In: Proceedings of IFIP advances in information and communication technology, pp 125–136
31. Rieder M, Steiner R, Berthouzoz C, Corthay F, Sterren T (2006) Synthesized UML, a practical approach to map UML to VHDL. In: Rapid integration of software engineering techniques. Lecture notes in computer science, vol 39, pp 203–217
32. Boulet P (2007) Array-OL revisited, multidimensional intensive signal processing specification. INRIA
33. Coppola M, Locatelli R, Maruccia G, Pieralisi L, Scandurra A (2004) Spidergon: a novel on-chip communication network. In: Proceedings of the international symposium on system-on-chip, p 15
34. Gamatié A, Le Beux S, Piel E, Ben Atitallah R, Etien A, Marquet P, Dekeyser JL (2011) A model driven design framework for massively parallel embedded systems. ACM Trans Embed Comput Syst 10(4):2–36
35. Boulet P (2008) Formal semantics of Array-OL, a domain specific language for intensive multidimensional signal processing. INRIA res rep RR-6467
36. Demeur A, Del Gallo Y (1998) An array approach for signal processing design. In: Proceedings of Sophia-Antipolis conference on micro-electronics (SAME98)
37. Object Management Group (2005) UML Profile for schedulability, performance, and time version 1.1. http://www.omg.org/technology/documents/formal/schedulability
38. Object Management Group (2006) Final adopted OMG SysML specification. http://www.omg.org/cgi-bin/docptc/06-0504
39. Yu H, Gamatié A, Rutten E, Dekeyser JL (2008) Model transformations from a data parallel formalism towards synchronous language. In: Embedded systems specification and design languages. Lecture notes in electrical engineering, vol 10, pp 183–198
40. Taillard J, Guyomarch F, Dekeyser JL (2008) A graphical framework for high performance computing using an MDE approach. In: Proceedings of Euromicro international conference on parallel, distributed and network-based processing (PDP08), pp 165–173
41. BEN Atitalah R, Piel E, Niar S, Marquet P, Dekeyser JL (2007) Multilevel MPSoC simulation using an MDE approach. In: Proceedings of the IEEE international SoC conference (SoCC07), pp 197–200
42. Quadri I, Elhaji M, Meftali S, Dekeyser J-L (2010) From MARTE to reconfigurable NoCs: a model driven design methodology, dynamic reconfigurable network-on-chip design: innovations for computational processing and communication. IGI Global, Hershey
43. Elhaji M, Attia B, Zitouni A, Meftali S, Dekeyser JL, Tourki R (2011) FERONOC: flexible and extensible router implementation for diagonal mesh topology. In: Proceedings of the design and architectures for signal and image processing (DASIP11), pp 269–276