HAL
open science

Modeling of Configurations for Embedded System Implementations in MARTE

Imran Rafiq Quadri, Abdoulaye Gamatié, Pierre Boulet, Jean-Luc Dekeyser

To cite this version:

Imran Rafiq Quadri, Abdoulaye Gamatié, Pierre Boulet, Jean-Luc Dekeyser. Modeling of Configurations for Embedded System Implementations in MARTE. 1st workshop on Model Based Engineering for Embedded Systems Design - Design, Automation and Test in Europe (DATE 2010), Mar 2010, Dresden, Germany. inria-00486845

HAL Id: inria-00486845

https://inria.hal.science/inria-00486845v1

Submitted on 27 May 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Modeling of Configurations for Embedded System Implementations in MARTE

Imran Rafiq Quadri, Abdoulaye Gamatié, Pierre Boulet, Jean-Luc Dekeyser
LIFL (UMR USTL/CNRS 8022), INRIA Lille-Nord Europe
40 avenue Halley, 59650 Villeneuve d'Ascq, FRANCE
Email: {Firstname.Lastname}@lifl.fr

***Abstract*—This paper deals with aspects related to modeling of system configurations, which are very useful for describing various states of an embedded system, from both structural and operational viewpoints. We discuss in detail the current proposition of the UML MARTE profile via some examples, and point out some limitations of the current proposition, mainly concerning the semantic aspects of the defined concepts. In order to draw answering elements, we report our experiences about the modeling of implementations and execution modes in Systems-on-Chip, within the Gaspard2 SoC co-design framework.**

I. Introduction

Dynamic or run-time reconfiguration of a component or platform-based system largely depends on the context required by designer or environmental conditions. This adaptation can be determined and effectively linked to different Quality-of-Service (QoS) criteria; such as energy consumption levels, performance throughput etc. Run-time reconfiguration requires the integration of an efficient *controller* component for managing different system configurations. Additionally, the semantics related to the component infrastructure must take into consideration several key issues: such as instantiation and termination of system components, deletion in case of user requirements. Components can be also homogeneous or heterogeneous in the context of a component framework, raising additional issues. Nowadays, modern embedded systems are mainly composed of heterogeneous components. Finally, re-usability of systems components is also an important issue that cannot be left ignored.

Gaspard2 [1], [2] is a Systems-on-Chip (SoC) co-design framework dedicated to parallel hardware and software and is based on the classical Y-chart [3]. One of the most important features of Gaspard2 is its ability for system co-modeling at an high abstraction level. Gaspard2 uses the Model-Driven Engineering methodology to model real-time embedded systems using the UML MARTE profile [4]; and UML graphical tools and technologies such as Papyrus [5] and Eclipse Modeling Framework [6].

In this paper, we discuss the need of configuration modeling with respect to Gaspard2 at different SoC design levels. We first present the current proposition of the MARTE specifications via some simple examples. Some limitations concerning the current semantic aspects are also pointed out. As an alternative solution, we present our contributions related to modeling of system implementations and execution modes in SoC, within the context of Gaspard2 framework.

II. Motivations for configurations in Gaspard2

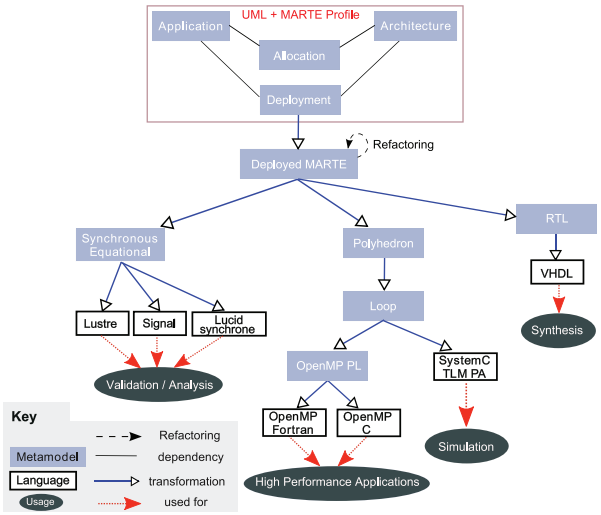

Fig. 1: A global view of the Gaspard2 framework

Figure 1 shows a global view of the Gaspard2 framework. Gaspard2 enables to model *software applications*, *hardware architectures* and their *allocations* in a concurrent manner. Once models of software applications and hardware architectures are defined, the functional parts (such as application tasks and data) can be mapped onto hardware resources (such as processors and memories) via *allocation(s)*. Gaspard also introduces a *deployment* level that allows to link hardware and software *elementary* components with intellectual properties (IPs); permitting re-utilization of IPs and enables to target different execution platforms.

For the purpose of automatic code generation from high level models, Gaspard adopts MDE model transformations (*model to model* and *model to text* transformations) [7] towards different execution platforms, such as targeted towards synchronous domain for validation and analysis purposes or FPGA synthesis, as shown in Figure 1. Model transformation chains permit moving from high abstraction levels to low enriched levels. Usually, the initial high level models contain only domain-specific concepts, while technological concepts are introduced seamlessly in the intermediate levels.

Now, we illustrate the usefulness of configuration modeling at different system design levels in Gaspard2.

A. Software application

Considering different versions of an algorithm in a system functionality is sometimes important, particularly when these versions yield different degrees of precision concerning the computed results. For instance, in the video processing domain, it is usual to encounter multiple algorithms dedicated for the same purpose, e.g. video decoding or encoding, video compression, etc. A very frequent transformation found in these algorithms is the discrete cosine transformation (DCT), which is normally a time-consuming process. Let us consider two different versions of a DCT that encode the same functionality, but are optimized differently. The choice between both versions according to a required video processing precision can be captured by considering the MARTE concepts for configuration modeling.

B. Hardware architecture

In a similar way, the configuration modeling concepts of MARTE could be useful for the hardware architecture of a system in order to change the structure of the architecture by modifying different parameters, such as the communication interconnections, bus widths, etc. The configurations can also be used to replace some hardware component by another, e.g. a processor by an hardware accelerator, for better execution performances. Similarly, characteristics of a processor can be interchanged by utilizing the characteristics of *Dynamic Voltage/Frequency Scaling* [8], [9].

C. Software/Hardware allocation

Considering different system configurations in terms of software/hardware allocations can be useful from several points of views: increasing the execution performances of the system functionality, decreasing the number of active executing computing units to reduce the overall power consumption levels, etc. Tasks of an application that are executing parallely on processing units may produce the desired computation at an optimal processing speed, but at a cost of increased power consumption levels. Modifying the allocation of the application on to the architecture can produce different combinations and different end results. A task may be switched to another processing unit that consumes less power, similarly, all tasks can be associated on to a single processing unit resulting in a temporal allocation as compared to a spatial one. This strategy reduces the power consumption levels along with decrease in the processing frequency. Thus allocation level allows incorporation of *Design Space Exploration* aspects, which in turn can be manipulated by the designers depending upon their chosen QoS criteria.

D. System deployment with intellectual properties (IPs)

The deployment level considered in our framework enables one to express how a specific IP implements an associated (elementary) component of software application or hardware architecture, among several possibilities present in an IP library. The reason is that in SoC design, a functionality can be implemented in different ways. For example, an application functionality can either be optimized for a processor, thus written in C/C++, or implemented as an hardware accelerator using *Hardware Description Languages* (HDLs). Hence the deployment level permits moving from platform-independent high level models to platform-dependent models for eventual implementation. Considering different modes for system deployment with IPs can also be seen as a way to deal with QoS criteria such as consumed resources, latency, power consumption levels, etc. This change in IPs at the deployment level may cause a global influence to the overall system design, offering different end results based on the chosen QoS criteria.

III. The MARTE proposition for mode and configuration modeling

A. Overall presentation

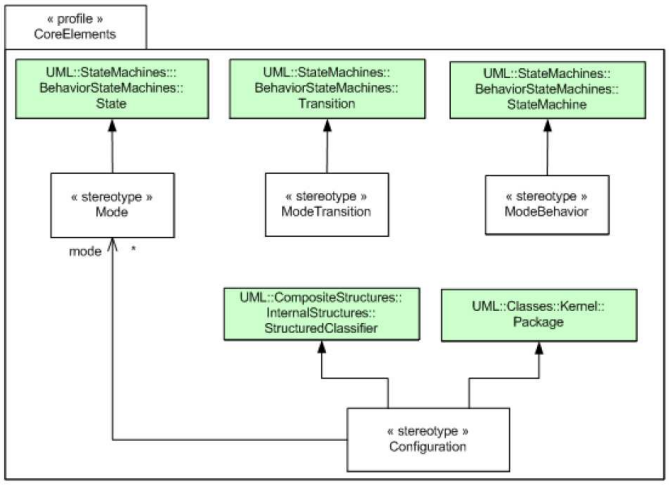

Fig. 2: Modes and Configurations in MARTE profile

The current MARTE proposition for the modeling of embedded system configurations and their associated controllers relies respectively on the use of components and finite state machines (FSMs), as illustrated in the following examples. This proposal has been inspired from AADL or *Architecture analysis and design language* [10]. Fig. 2 illustrates the MARTE profile concepts related to system modes and configurations. It is evident that there is a one to one correspondence between the MARTE concepts and pure UML state machine semantics. A mode is related to a system configuration, and mode transitions represent the transitions between the different available configurations. Equally, a mode behavior represents a state machine responsible for switching between the different available configurations.

B. Example of system configuration modeling

Fig. 3 illustrates a component, named `Mode_SystemConfiguration1`, with the stereotype `<<configuration>>` that represents a configuration for an

allocation of an application functionality on an hardware architecture. More generally, this component encapsulates a model in the form of a classifier or a package. The mode value `FullProcessorMode` indicating when this configuration is active is noted on the top right of the `Mode_SystemConfiguration1` component.

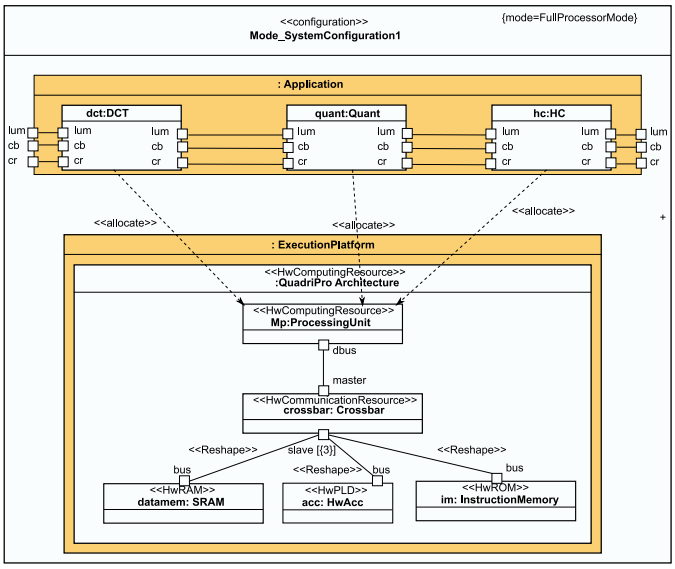

Fig. 3: Processor-based homogeneous allocation.

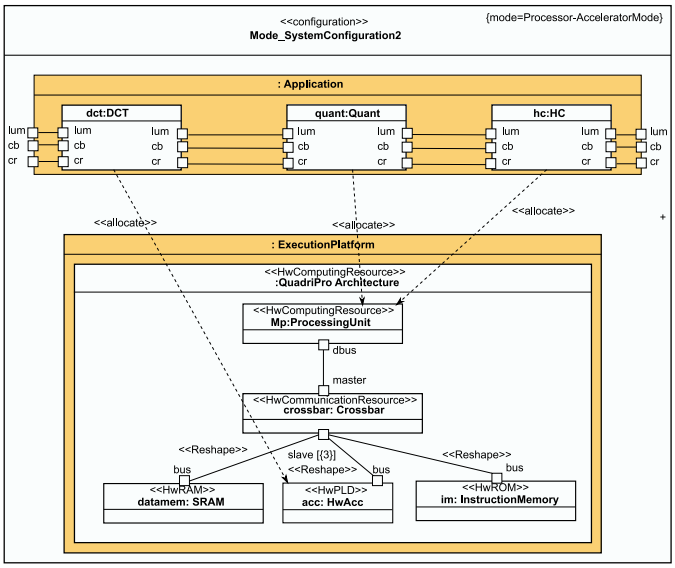

Fig. 4: Mixed processor/hardware accelerator allocation.

More concretely, Fig. 3 shows the mapping of the intra-part of the H263 encoder dedicated to video processing [11]. The software application part is composed of three main components: a DCT, a quantizer and a Huffman coding function. The application is allocated onto an hardware architecture that is composed of a processor, an hardware accelerator, and memory devices. Here, the allocation expresses that all application functional components are allocated and subsequently executed on the processor.

Now, Fig. 4 depicts another software/hardware allocation scenario according to which only the quantizer and the Huffman coding function are executed on the processor. The DCT is now executed on a dedicated hardware accelerator that offers better execution performances than the processor, due to its parallel regular execution. Indeed, it is well-known that the DCT is one of the most resource-demanding parts of video encoding algorithms, and particularly of the H.263 encoder.

The way mode values are produced for selecting configurations is specified via an FSM, as modeled in Fig. 5. The FSM contains two states corresponding to the two configurations illustrated before. Each state of the FSM is associated with some mode value specifications. An active configuration is therefore the one associated with a mode value corresponding to the current state of the controller FSM. The transition from a configuration to another is captured via the transitions of the states present in the FSM.

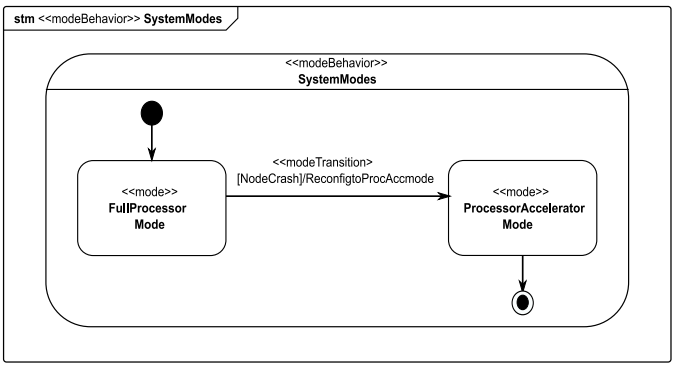

Fig. 5: Mode specification with a FSM in MARTE.

C. Open questions about model semantics

Beyond the already identified unclarities [12], [13] in the semantics of UML state machines on which the configuration controller definition relies upon, there are further concerns during the interactions between a controller and its associated configurations, as discussed below.

1) Configuration switch: An important aspect that requires further clarifications about configuration switch concerns the nature of transitions between the states of a controller. In some contexts, such transitions are not immediate. Typically, when considering a configuration switch at an hardware architecture level, the circuit needs to go through a stabilization phase before changing its status. So at least, there should be a clear distinction related to *weak* or *strong* transitions between configuration as in synchronous mode automata [14], [15]. Roughly speaking, a weak transition delays the observation of the results from a suspended configuration to the moment at which a new configuration becomes active. This delay may be interpreted as a stabilization phase before the production of the results in the suspended configuration. In contrast a strong

transition makes the results of the suspended configuration instantaneously available.

2) Multi-level configuration control: It is possible to have multi-level controllers in a complex system. These controllers can be combined at different SoC levels to describe more complex configuration switches. Typically, controllers can also be composed in parallel, as illustrated in Fig. 6.

Here, two modes are available for both the application and the architectural aspects of a given system; and a transition can be carried out depending upon the given requirements. Therefore, an expected behavior from this approach is that all controllers make their transitions in parallel, within the same global transition. These parallel controllers may synchronize through their event occurrences, the output from one controller being an input of the other.

Another interesting scenario for composing controllers is an hierarchic approach, where a state of a controller may itself consist of another controller. In simpler terms, that means that a state may itself contain an embedded FSM. This scenario is represented in Fig. 7. Here, `FullProcessorMode` itself contains two modes or states that correspond to different behaviors related to the processor present in the system. Thus the processor can either operate in a low energy consumption mode, at the comprise of performance throughput or it can operate in an alternative manner.

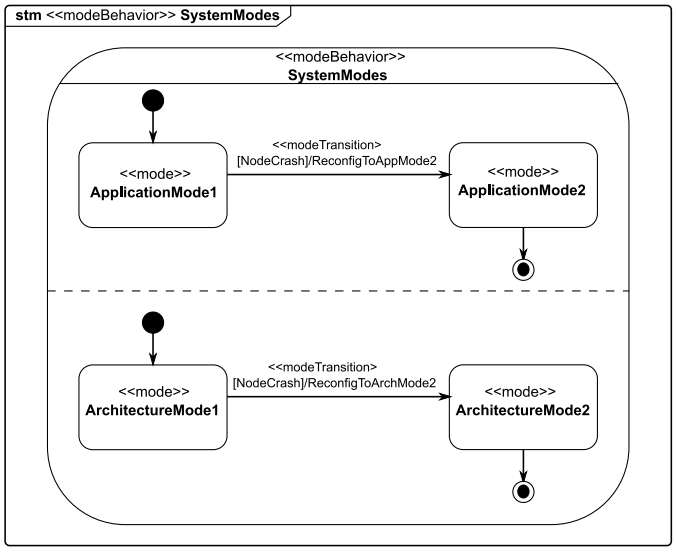

Fig. 6: A scenario with parallel composition of modes.

In the current proposition of MARTE, the semantics of such multi-level compositions of configuration controllers is not clearly discussed. For instance, in an hierarchical controller, how does one synchronize the states at a given level with states at the sub-levels? Should a state that itself contains embedded states be stereotyped as either a `Mode` or a `ModeBehavior`? While in MARTE, the proposal specifies that the owned states of a state machine must be stereotyped only as modes, what happens in the case of hierarchy? Equally, when a configuration gets suspended from a controller state at a sub-level, how does one manages to resume this configuration? These questions can be answered only if the semantics aspects are precised. In the literature, there are already several proposal candidates for defining such a semantics, as illustrated in [16], [17], [18], among other popular references.

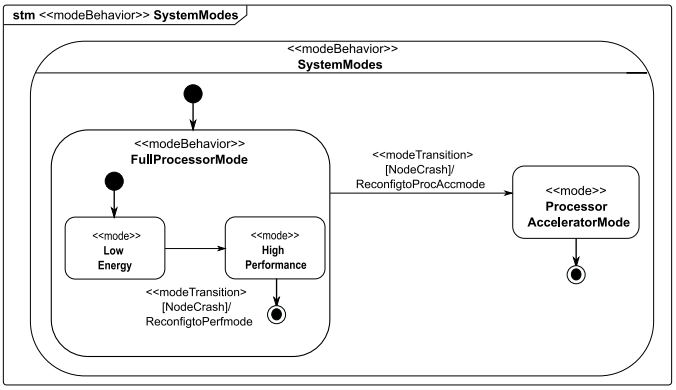

Fig. 7: Hierarchic mode composition for a given system.

D. Need of guidelines for concept usage

An arguable position about the absence of a precise semantics for the proposed configuration modeling concepts could be justified by the need of generality, as for the UML language, by leaving the semantics only partially defined. The advantage is that the concepts are usable in different contexts according to suitable semantics.

However, one has to notice that the above concepts are defined in a profile, which aims to add all necessary ingredients to the more general UML language in order to be adequately exploited by embedded system designers. So, in our opinion, in absence of a precise semantics, there should be (at least) some accompanying guidelines explaining possible relevant ways of combining such concepts in practice. Typically, according to a given system design level, as mentioned in section II, how does one can use these concepts in the same way? We believe that such guidelines could be a very worthy alternative to a precise semantics.

IV. Mode-oriented design in Gaspard2

As an alternative to the solution presented in the MARTE profile, here we present an interesting substitute, in order to model multimode scenarios in Gaspard2. Our presented approach shares some common concepts as presented in the MARTE profile. We now introduce some of the main concepts related to our contribution, as well as the adopted semantics.

A. Mode switch component

A *mode* plays a role similar to a configuration illustrated earlier in the paper. A `Mode Switch Component` contains at least one mode; and offers a switch functionality that selects one mode (or configuration) to be executed among several available modes [19]. For instance, in Fig. 8, the `Mode Switch Component` has an input mode value port `Mode`. The switch

between the different modes is carried out according to the value received through the `Mode` port. The different modes of the `Mode Switch Component` are describe by using UML collaboration diagrams associated with the component. These instance level collaborations specify roles of components, via usage of connectors and parts in composite structures. A collaboration specifies the relation between some collaborating components (or roles). Each of these roles provides a specific function, and executes some required functionality in a collective way. Only the concerned aspects of a role are included in a collaboration while others are omitted.

The name of the collaborations correspond to the mode values and thus these collaborations define the activity of a mode switch component upon receiving a particular mode value. For example, the collaboration `FullProcMode` shows the relationship between the mode switch component and the mode/configuration `FullProcessorConfiguration`; indicating that mode value `FullProcMode` switches the current executing mode to `FullProcessorConfiguration`. As in this mode only `FullProcessorConfiguration` is to be executed, the second mode `ProcessorAccConfiguration` is omitted along with the mode port of the mode switch component, due to the semantics of UML collaborations. The collaboration is finally linked to the mode switch component.

Additionally, each mode component can be hierarchical or elementary in nature. Given a hierarchical level, all modes have the same interface [20]. Finally, for a received mode value, the mode run exclusively at any instant.

B. State graph

A state graph in Gaspard2 is similar to a statechart. We term these state graphs as `Gaspard state graphs`. Each state is associated with some mode value specifications that provide mode values for the state. The mode values allow to activate different exclusive modes in the associated mode switch components. Similarly to the mode switch component, a Gaspard2 state graph component has an interfaces including event inputs from the environment, source state inputs, target state outputs and mode value outputs. Event inputs are used to trigger transitions. The source state inputs determine the states from which the transitions take place, while target state outputs determine the current destination states of the fired transitions. The mode outputs are associated with a mode switch component in order to select the right operating mode.

C. Combining modes switch and state graphs

A `Macro Component` is used to compose mode switch components and state graph components together. A UML representation of the macro component in Fig .8 illustrates one possible composition; and represents a complete Gaspard2 control structure. In this structure, the state graph component produces a mode value (or a set of mode values) and sends it (them) to the mode switch component. The latter switches the modes accordingly. The illustrated figure is used as a basic composition, however, other compositions are also possible, for instance, a state graph component can control several mode switch components [21].

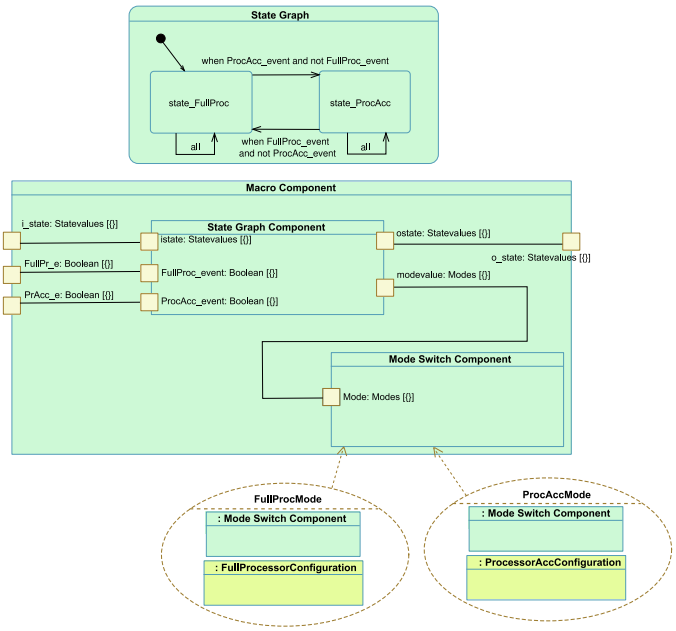

Fig. 8: A UML example illustrating a macro component.

D. Mode automata-based semantics

For the semantics of Gaspard2 models including modes, we have adopted the semantics of mode automata [18]. Mode automata are mainly composed of modes and transitions. In an automaton, each mode has the same interface. Transitions can be associated with conditions, which act as triggers. Finally, only weak transitions are considered. Mode automata can be composed together in a parallel way, which is called *parallel composition*. The composition result is the Cartesian product of the sets of modes of the automata to be composed. The composition also makes it possible that the automata communicate with each other in a way that one output of one automaton can be taken as an input of another automaton. Hierarchical composition is based on the refinement of certain modes in the automata. At each level, the variables in the states are considered *global*, as the state at this level sees these variables. However, a variable cannot be multiply defined at different hierarchical levels.

The proposed semantics can be implemented at different levels of a SoC co-design framework, as illustrated in section II. During the research works related to modes, we have observed the integration at two levels, mainly the application and the IP deployment level, however, future works hope to focus on other design levels such as the architecture and allocation level.

E. Integrating modes at the application level

The Gaspard control model has been implemented with UML state machines and collaborations. A model transformation chain from high level MARTE models to synchronous

languages can bridge the gap between these models and targeted synchronous language code. The model transformation chain can be divided into several successive parts in order to ease integration of new concepts in the chain [20]. By considering the code generated from an application model, validation techniques such as model checking can be applied. The same code can also be used for controller synthesis to enforce relevant properties with respect to functional and non-functional requirements. All these aspects have been addressed in a case study for the design of a Gaspard data-parallel multimedia application [20].

F. Mode integration at deployment level for FPGA synthesis

Equally, the mode automata proposal has been introduced at the IP deployment level in Gaspard2 for the purpose of implementing run-time reconfiguration in modern FPGAs. This type of reconfiguration separates the FPGA into static/dynamic parts and permits to partly change the current executing FPGA configuration with another, depending upon designer requirements and QoS criteria [22]. Thus designers can initially implement, and afterwards, reconfigure a complete SoC on FPGA for the required customized solution. A reconfigurable region can have several implementations, with each having the same interface, and can be viewed as a mode switch component with different modes.

For dynamic reconfiguration, an embedded controller (usually in the form of a micro-processor) inside the FPGA is responsible for the switch between the different configurations. The controller can be either intelligent to take the switch decision itself, or it can depend upon external stimuli such as inputs or events specified by a user.

The control at the deployment level is utilized to generate the switch mechanism present inside the embedded controller situated in the FPGA static part, automatically via model transformations. Equally, an high level application model is transformed into a run-time reconfigurable hardware accelerator having several implementations. These works were validated within a case study related to the development of a run-time reconfigurable correlation module, which is a part of a larger anti-collision radar detection system. Details related to these works can be found in [23].

V. Summary

In this paper, we have provided the motivations behind the need of configuration modeling related to modern embedded systems, via different point of views. We have elaborated the current concepts present in the MARTE specifications for this purpose. Equally, by means of some interesting examples, we have illustrated how such concepts can be used in a SoC co-design framework, namely Gaspard2. Some limitations related to the proposal have been highlighted in the paper, which should be addressed in future revisions of the specifications, for enable more effective model expressions from a semantic viewpoint. We also also presented our experiences related to mode-oriented design in Gaspard2, and advocated it as a possible inspiration for the MARTE proposition. Finally, integration of this mode based approach has been explained briefly at two design levels for different purposes and execution platforms.

References

[1] DaRT team, "GASPARD SoC Framework," 2010, www.gaspard2.org/.

[2] A. Gamatié and S. Le Beux and E. Piel and A. Etien and R. B. Atitallah and P. Marquet and J.-L Dekeyser, "A model driven design framework for high performance embedded systems," INRIA, Research Report RR-6614, 2008, http://hal.inria.fr/inria-00311115/en.

[3] D. D. Gajski and R. Kuhn., "Guest editor introduction : New VLSI-tools," *IEEE Computer*, vol. 16, no. 12, pp. 11–14, Dec. 1983.

[4] OMG, "Modeling and Analysis of Real-time and Embedded systems (MARTE), Version 1.0," http://www.omg.org/spec/MARTE/1.0/, 2009.

[5] Papyrus, "Open source tool for graphical UML2 modeling," www.papyrusuml.org/.

[6] Eclipse, "Eclipse Modeling Framework," http://www.eclipse.org/emf.

[7] 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.

[8] L. Yung-Hsiang, L. Benini, and G. D. Micheli, "Dynamic frequency scaling with buffer insertion for mixed workloads," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 11, pp. 1284–1305, 2002.

[9] C. Im, H. Kim, and S. Ha, "Dynamic voltage scheduling technique for low-power multimedia applications using buffers," pp. 34–39, 2001.

[10] AADL, "The architecture analysis & design language (aadl): An introduction," http://www.sei.cmu.edu/publications/documents/06.reports/06tn011.html, 2006.

[11] International Telecommunication Union (ITU), "Recommendation h.263. video coding for low bit rate communication," January 2005.

[12] Software Technology Laboratory, "Stl: Uml semantics project. school of computing, queen's university," http://www.cs.queensu.ca/~stl/internal/uml2/bibtex/ref_umlstatemachines.htm, 2009.

[13] H. Fecher, J. Schönborn, M. Kyas, and W. P. de Roever, "29 new unclarities in the semantics of uml 2.0 state machines," in *7th International Conf. on Formal Engineering Methods (ICFEM'05)*, 2005, pp. 52–65.

[14] J.-L. Colaço, G. Hamon, and M. Pouzet, "Mixing Signals and Modes in Synchronous Data-flow Systems," in *ACM International Conference on Embedded Software (EMSOFT'06)*, Seoul, South Korea, October 2006.

[15] J.-P. Talpin, C. Brunette, T. Gautier, and A. Gamatié, "Polychronous mode automata," in *EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software*. New York, NY, USA: ACM, 2006, pp. 83–92.

[16] M. von der Beeck, "A comparison of statecharts variants," in *ProCoS: 3rd International Symposium Organized Jointly with the Working Group Provably Correct Systems on Formal Techniques in Real-Time and Fault-Tolerant Systems*. London, UK: Springer-Verlag, 1994, pp. 128–148.

[17] C. André, "Synccharts: a visual representation of reactive behaviors," I3S, Sophia-Antipolis, France, Tech. Rep. RR 95–52, rev. RR (96–56), Rev. April 1996.

[18] F. Maraninchi and Y. Rémond, "Mode-automata: About modes and states for reactive systems," in *European Symposium On Programming*. Lisbon (Portugal): Springer verlag, Mar. 1998.

[19] O. Labbani and J.-L Dekeyser and Pierre Boulet and É. Rutten, "Introducing control in the gaspard2 data-parallel metamodel: Synchronous approach," in *Proceedings of the International Workshop MARTES: Modeling and Analysis of Real-Time and Embedded Systems*, 2005.

[20] H. Yu, "A MARTE based reactive model for data-parallel intensive processing: Transformation toward the synchronous model," Ph.D. dissertation, USTL, 2008.

[21] I. R. Quadri, S. Meftali, and J.-L. Dekeyser, "Integrating mode automata control models in soc co-design for dynamically reconfigurable fpgas," in *International Conference on Design and Architectures for Signal and Image Processing (DASIP 09)*, 2009.

[22] P. Lysaght and B. Blodget and J. Mason, "Invited Paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs," in *FPL'06*, 2006.

[23] I. R. Quadri, A. Muller, S. Meftali, and J.-L. Dekeyser, "Marte based design flow for partially reconfigurable systems-on-chips," in *17th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC 09)*, 2009.