# MODELING AND SYNTHESIS OF A DYNAMIC AND PARTIAL RECONFIGURATION CONTROLLER

*S. Guillet*, F. de Lamotte*, N. Le Griguer*, É. Rutten**, J.-P. Diguet*, G. Gogniat**

*Lab-STICC, Université de Bretagne Sud, France,
{sebastien.guillet,florent.lamotte, nicolas.le-griguer,jean-philippe.diguet,guy.gogniat}@univ-ubs.fr
**LIG / INRIA Rhône-Alpes, France, eric.rutten@inria.fr

## ABSTRACT

This paper presents a framework to integrate the formal synthesis of a reconfiguration controller into a Model Driven Engineering methodology used for reliable design of reconfigurable architectures. This methodology is based on an extension of UML/MARTE, GASPARD, and the aforementioned controller is obtained as a C code through a formal technique named Discrete Controller Synthesis. Taking advantage of using both modeling and synthesis techniques, the approach demonstrates an effective reduction of complexity in the specification of such reconfigurable systems. An application model of an image processing application is presented as a case study.

## 1. INTRODUCTION

Designing adaptive Systems-on-Chip (SoC) has raised in complexity and sophistication over the last few years, and designing them to be safe is becoming even more complex. Even if adaptive systems became a reality with the introduction of Dynamic and Partial Reconfiguration (DPR) technologies by Xilinx, autonomous reconfiguration has not much been addressed. Designing correctness is an important issue in critical SoCs, as a single error can lead to a financial, material and/or human disaster. However, adding reconfiguration concepts in such SoCs increases the design complexity as adaptive systems need to be both secure and optimized to address (often) contradictory constraints (i.e. quality of service, energy consumption, etc.). Many studies, including this one, are based on the hypothesis that the complexity will continue to increase, and thus the need in terms of modeling and formal methodologies will become significant. This is leading to a progressive usage of methodologies to apply abstraction and modeling techniques such as Model Driven Engineering (MDE) [1] and formal techniques to prove execution properties in these systems. MDE allows to specify a system both vertically (compilation flow) and horizontally (models for analysis, verification, performance...). The proposed approach relies on such a methodology, starting from different models of the SoC, concerning its application, its architecture and association between software and hardware components. Properties about execution states and reconfiguration possibilities are then extracted from these specifications to be used by a formal technique, which synthesizes the code of a controller. The role of the controller is to enforce these properties at runtime. Concerning the methodology, this paper will focus especially on how to specify a Gaspard model in order to be ready for controller synthesis. Discrete Controller Synthesis (DCS) is the formal technique chosen in this study to tackle the problem of complexity in the reconfiguration specification. After presenting the related work and tools, the control concepts are further detailed and finally, an example of a Gaspard model containing control information is shown.

This work is supported by the ANR FAMOUS project (ANR-09-SEGI-003)

## 2. RELATED WORK

### 2.1. Closed-loop control in reconfigurable architectures

Many research works contribute to the domain of reconfigurable embedded systems [2] [3], some of them use continuous control techniques. This study is especially interested in those which care about the closed-loop management of reconfiguration. In [4], an FPGA-based PID motion control system that dynamically adapts the behavior of a robot is presented. Several designs can be swapped, and tradeoff between them are evaluated in terms of area, speed or power consumption. It has to be noted that functional correctness of all the designs is verified by experiment, and not by a formal method. To assure the correctness of the execution of embedded systems, analysis verification and control methods are needed. These methods are often based on model checking, for example in [5] authors use such technique for migration (reconfiguration) of algorithms from hardware to software. Another approach is based on theorem proving, such as [6] that presents a framework for description and verification of parametrized hardware libraries with lay-

978-1-4673-2256-0/12/$31.00 

out information (explicit symbolic coordinates, neighboring placement). The correctness of the generated layout is established by proof in higher order logic using Isabelle theorem prover. What these studies have in common is that each time, the control system for reconfiguration must be entirely specified by the designer, so that it can be verified or proven afterwards. But a technique from discrete control theory, discussed in the next section, allows for the control design only by giving its constraints.

A closer approach to the current proposition is [7] where a formal control technique, based on Discrete Controller Synthesis, is used to control the communications between system-on-chip components by filtering their inputs. But the technique is only applicable in a class of applications in hardware design where input filtering can be safely achieved.

### 2.2. Synchronous approach and Discrete Controller Synthesis

The considered reconfigurable SoCs are a specialization of autonomic computing systems [8], which adapt and reconfigure themselves through the presence of a feedback loop. This loop takes inputs from the environment (e.g. sensors), updates a representation (e.g. Petri nets, automata) of the system under control, and decides to reconfigure the system if necessary.

Several works [9] [10] chose to describe these loops in terms of Discrete Controller Synthesis (DCS) problems. It consists in considering on the one hand, the set of possible behaviors of a discrete event system [11], where variables are partitioned into uncontrollable and controllable ones. The uncontrollable variables typically come from the system's environment (i.e. "inputs"), while the values of the controllable variables are given by the synthesized controller itself. On the other hand, it requires a specification of a control objective: a property typically concerning reachability or invariance of a state space subset. Such programming makes use of reconfiguration policy by logical contract. Namely, specifications with contracts amount to specify declaratively the control objective, and to have an automaton describing possible behaviors, rather than writing down the complete correct control solution. The basic case is that of contracts on logical properties i.e., involving only Boolean conditions on states and events.

Within the synchronous approach [12], DCS has been defined and implemented as a tool integrated with the synchronous languages: SIGALI [13]. It handles transition systems with the multi-event labels typical of the synchronous approach, and features weight functions mechanisms to introduce some quantitative information and perform optimal DCS. It has been applied to the generation of correct task handlers, adaptive resource management [10], reconfigurable component-based systems [14], and integrated in a synchronous language, named BZR [15].

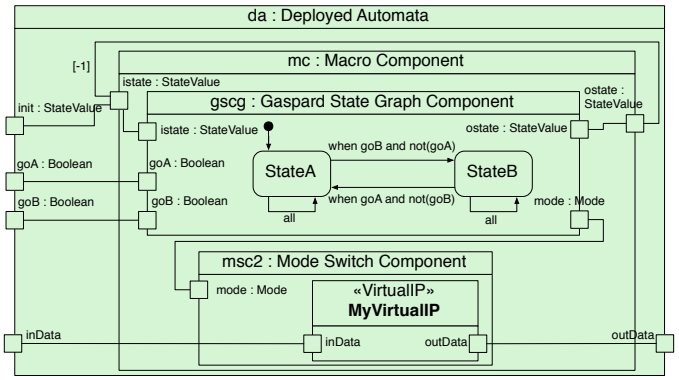


**Fig. 1**. Modeling of a deployed mode automata

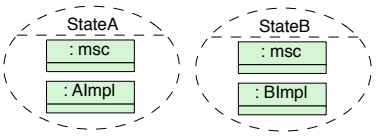


**Fig. 2**. Associated collaborations of a MSC

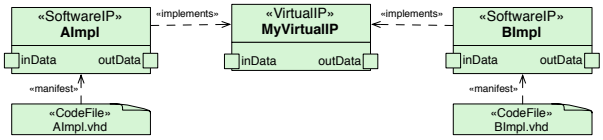


**Fig. 3**. Implementations of a VirtuaIP

BZR extends Heptagon, which is itself a synchronous programming language, and includes a DCS usage from SIGALI within its compilation. BZR is used in this work, and its compilation yields a correct-by-construction controller (here in C language), which is then integrated into the reconfigurable platform.

Actual transformations into BZR goes beyond the scope of this paper, and the interested reader should refer to [18] to understand the formal concepts of these transformations.

## 3. GASPARD AND MARTE CONCEPTS USED IN THE CURRENT METHODOLOGY

For the sake of simplicity, only the GASPARD and MARTE elements concerned by control aspects are presented here. More information about how to model and generate a SoC (especially the architecture) from GASPARD can be found in [17]. We use this MARTE based methodology which adopts a component based approach to express dynamic aspects in a SoC. It makes use of UML collaborations and variants of UML state machines, referred as Gaspard State Graphs (GSG), embedded in Gaspard State Graph Components (GSGC), which control Mode Switch Components (MSC).

A GSGC reacts to external events, and is always considered as a controlling component, meaning that it outputs at least one value, termed as a mode value, reflecting the implementation of a reconfigurable component. The interface of the GSGC is represented by ports, stereotyped accordingly as MARTE FlowPorts. The input ports of a GSGC can be either event or state ports. Event ports serve in triggering a transition in the associated GSG and are typed Boolean or Integer. Values associated to state ports are termed as state values and identify the different states in the associated state graph. The input state port for a GSGC indicates the initial state upon entering the GSGC. A GSGC also supports two kinds of output ports: mode ports and state ports. The GSG associated to the GSGC carries out transition functions on the states and each state is associated to one mode. The output mode port thus carries mode values to a Mode Switch Component (MSC); which are determined by the transitions of the state graphs. The output state ports are similar to the input state ports and provide the next state of the GSGC (the next state after a transition). Figure 1 illustrates an association of a GSG with a GSCG into a Macro Component (MC). This MC is also encapsulated in a Deployed Automata component, showing the generic repetition between the input and output states which is employed as a memorization of the previous state. As for the GSC, it shows an example with two states, one of them (StateA) being connected to the initial pseudo state, and transition between them, triggered by Boolean expression defined on events. A MSC, associated to collaborations, is employed to switch between different implementations of a virtual IP defined by an interface. It takes a mode value as input and executes the corresponding mode. Only one exclusive mode per MSC can be selected at a time. Collaborations associated to MSCs specify the relation between some collaborating components, or roles, each role providing a specific function. Figure 2 depicts the behavior of a MSC. For example, the collaboration StateA shows the relationship between the MSC and the mode/configuration AImpl. It indicates that the mode value StateA switches from the current executing mode to AImpl, which is an implementation of the virtual IP embedded in the MSC. The second mode, StateB, is then omitted along with the mode port of the MSC, due to the semantics of UML collaborations. An IP component in MARTE represents an intellectual property or implementation, which can be either software or hardware based in nature, and thus inherited by a concrete SoftwareIP or HardwareIP component in MARTE. The actual technical implementation of an IP is defined as a CodeFile reference which links it to the source files of the IP, cf Figure 3. A VirtualIP component defines a generic interface, implemented by one or more IPs, i.e. these IPs have the same number of port, which have the same semantics and type. The actual runtime implementation of a VirtualIP component is selected by an associated MSC.

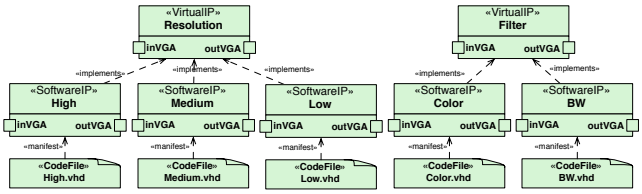


**Fig. 4**. Resolution and Color VirtualIPs implementations

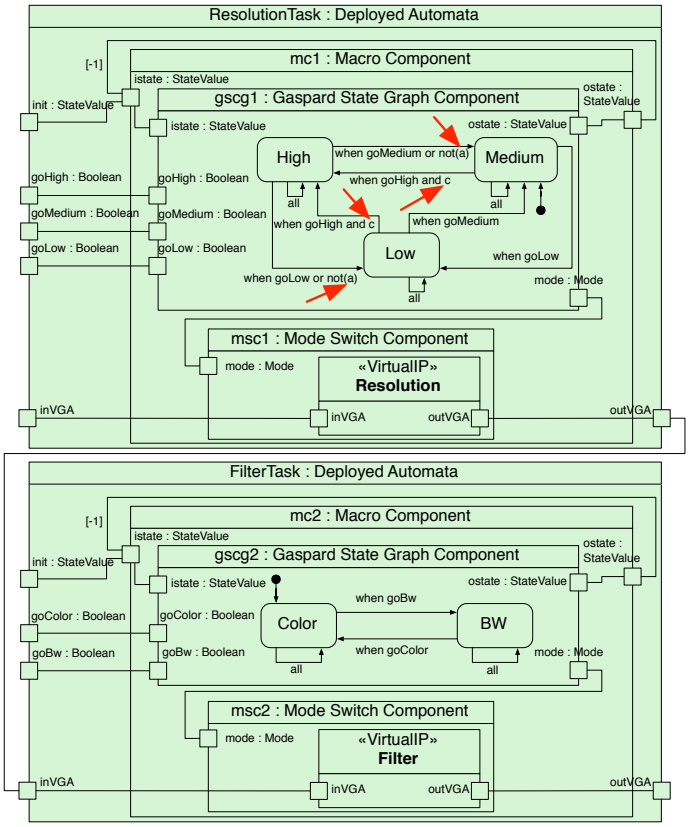


**Fig. 5**. Application task graph

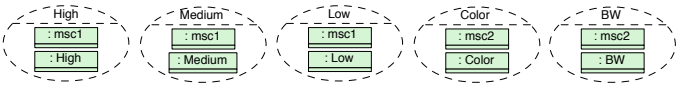


**Fig. 6**. Collaborations between the modes of msc1 and msc2 and their target implementations

### 3.1. GASPARD model

An example of a reconfigurable image processing application is modeled in GASPARD. The interesting parts of this model, containing control aspects, are presented here. The considered system is composed of two reconfigurable tasks, ResolutionTask and FilterTask, respectively resizing and filtering images from a video stream. Each task has several implementations (IPs), respectively High, Medium, Low and

Color, Bw. Figure 4 shows the implementation relations between these IPs and their generic interface (VirtualIP). When a control step is triggered (e.g. by pushing a button), the system receives five events from its environment (eg. from dip switches): goHigh, goMedium, goLow, goColor and goBw, which are used to take transitions from an implementation to another in both tasks. For example, upon the reception of goMedium, the system is supposed to go from whatever implementation of ResolutionTask to the Medium implementation. Figure 5 shows the reconfiguration behavior of the two tasks upon the reception of these events, by using the semantics of Deployed Automata, Gaspard State Graphs and Mode Switch Components. Figure 6 shows the collaborations between the MSC and the actual implementation of their VirtualIP, defined by the mode they receive.

For the sake of having a control objective example, let's say that High resolution cannot be selected in concurrence with the black and white mode, and let's make the transitions coming to High and going out of High controllable. In Figure 5, two controllable variables, $c$ and $a$, are used to respectively inhibit transitions to High (if $c$ is set to False) and force transitions going out of High (if $a$ is set to False). These variables are identified as "controllable" because they don't come from any port, so their values (True by default) can only be set by the controller itself. So in this specification, if nothing has to be controlled to fulfill the control objective, $c$ and $a$ remain True and no transition is forced or inhibited. The equation of the control objective is the only part which is not yet present in the MARTE specification. Transformations and execution of an equivalent representation (automata based) is shown in [18].

## 4. CONCLUSION AND FUTURE WORK

This paper presented a way to specify a reconfiguration controller for a DPR SoC using a MARTE-based profile, GASPARD in order to use the Discrete Controller Synthesis formal technique. Controllers obtained through this methodology are guaranteed to always provide a correct configuration to the system, with respect to constraints specified by the designer, for every possible execution, thus freeing the designer to test the system on this critical aspect. This methodology is integrated in a conception flow from where a complete platform can be designed and transformed into an executable one. Both integration and execution of a controller into a reconfigurable platform is presented in [18], using an equivalent model of the one shown in this paper. [18] formalizes the transformations and also demonstrates how a decision system can be connected to the controller, to make optimizations when several valuations of the controllable variables exist.

## 5. REFERENCES


[1] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet, "A co-design approach for embedded system modeling and code generation with uml and marte," *DATE*, 2009.

[2] J.-P. Diguet, Y. Eustache, and G. Gogniat, "Closed-loop based self-adaptative hw/sw embedded systems: design methodology and smart cam case study," *ACM Transactions on Embedded Computing Systems*, vol. 10, no. 3, 2011.

[3] D. Gohringer, M. Hubner, V.Schatz, and J. Becker, "Runtime adaptive multi-processor system-on-chip: Rampsoc," *IPDPS*, pp. 1–7, April 2008.

[4] W. Zaho and B. H. Kim, "Fpga implementation of closed-loop control system for small-scale robot," *ICAR*, 2005.

[5] M. Borgatti, A. Fedeli, U. Rossi, J.-L. Lambert, I. Moussa, F. Fummi, C. Marconcini, and G. Pravadelli, "A verification methodology for reconfigurable systems," *Int. Workshop on Microprocessor Test and Verification*, 2004.

[6] O. Pell, "Verification of fpga layout generators in higher-order logic," *Journal of automated reasoning*, 2006.

[7] E. Dumitrescu, M. Ren, L. Pietrac, and E. Niel, "A supervisor implementation approach in discrete controller synthesis," *ETFA*, 2008.

[8] J. O. Kephart and D. M. Chess, "The vision of autonomic computing." *IEEE Computer*, 2003.

[9] F. Boyer, G. Delaval, N. de Palma, O. Gruber, and E. Rutten, "Discrete supervisory control application to computing systems administration," in *INCOM*, 2012.

[10] G. Delaval and É. Rutten, "Reactive model-based control of reconfiguration in the fractal component-based model," *CBSE*, 2010.

[11] C. Cassandras and S. Lafortune, "Introduction to discrete event systems," *Kluwer Acad. Publ.*, 1999.

[12] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. L. Guernic, and R. D. Simone, "The synchronous languages 12 years later," 2003.

[13] H. Marchand, P. Bournai, M. Borgne, and P. Guernic, "Synthesis of discrete-event controllers based on the signal environment," *Discrete Event Dynamic Systems*, 2000.

[14] T. Bouhadiba, Q. Sabah, G. Delaval, and E. Rutten, "Synchronous control of reconfiguration in fractal component-based systems – a case study," *EMSOFT*, 2011.

[15] G. Delaval and H. M. E. Rutten, "Contracts for modular discrete controller synthesis," in *LCTES*, 2010.

[16] S. Guillet, F. D. Lamotte, É. Rutten, G. Gogniat, and J.-P. Diguet, "Modeling and formal control of partial dynamic reconfiguration," *ReConFig*, 2010.

[17] I. R. Quadri, A. Gamatié, P. Boulet, S. Meftali, and J.-L. Dekeyser, "Expressing embedded systems configurations at high abstraction levels with uml marte profile: Advantages, limitations and alternatives," *Journal of Systems Architecture*, 2012.

[18] S. Guillet, F. D. Lamotte, N. L. Griguer, É. Rutten, G. Gogniat, and J.-P. Diguet, "Designing formal reconfiguration control using uml/marte," *ReCoSoC*, 2012.