Distributed control for reconfigurable FPGA systems: a high-level design approach

Chiraz Trabelsi, Samy Meftali, Jean-Luc Dekeyser

Distributed control for reconfigurable FPGA systems: a high-level design approach

Chiraz Trabelsi, Samy Meftali and Jean-Luc Dekeyser
INRIA Lille Nord Europe - LIFL - Universite Lille1
Lille, FRANCE
Email:{Firstname.Lastname}@inria.fr

***Abstract*—Due to their exponential complexity, designing adaptation control for Reconfigurable Systems-on-Chip (RSoC) is becoming one of the most challenging tasks, resulting in longer design cycles and increased time-to-market. This paper addresses this issue and proposes a novel adaptation control design approach for FPGA-based reconfigurable systems aiming to increase design productivity. This approach combines control distribution and high-level modeling in order to decrease design complexity and enhance design reuse and scalability. Control distribution is based on allocating local control aspects (monitoring, decision and reconfiguration) to distributed controllers, while respecting global system constraints/objectives using a coordinator. High-level modeling makes use of Model-Driven Engineering and the MARTE (Modeling and Analysis of Real-Time and Embedded Systems) standard in order to move from high level models to automatic code generation, which significantly simplifies the control design. The proposed design approach is integrated in a model-driven RSoC design flow and allows to model adaptation aspects at different design levels: application, architecture, allocation and deployment, which allows to target a wide range of control requirements. In order to validate our approach, a video processing application was implemented on a reconfigurable system that contained four distributed hardware controllers.**

***Index Terms*—Distributed control, high-level modeling, UML MARTE, reconfiguration control, partial dynamic reconfiguration, FPGA**

I. Introduction

Being able to be reconfigured an arbitrary number of times, Reconfigurable Systems-on-Chip (RSoC) are becoming widely used thanks to their flexibility and adaptivity. These systems offer the possibility of being modified after fabrication in order to adapt to changes such as user preferences, application requirements and standard changes. Dynamic reconfiguration has the potential of offering a run-time system modification by replacing reconfiguration data (bitstream). Partial Dynamic Reconfiguration (PDR) is a very powerful mechanism supported by some modern FPGAs [1]. Supporting PDR provides a high platform flexibility allowing to reconfigure only some regions of the FPGA (loading partial bitstreams) without disturbing the operation of the rest of the system. Such a mechanism allows to reduce the total system area while meeting performance constraints. However, adaptivity and reconfigurability implies an additional task for SoC designers, which is adaptation control design whose complexity is increasing with the size of the applications that modern RSoC are able to embed. In this context, designing one controller to handle the adaptation of the whole system is becoming a complex and time-consuming task. Moreover, the rigidity of such a design, due to its independence to the implemented system, represents an obstacle to design reuse. Therefore, increasing autonomy in control design can be viewed as an effective solution to deal with the growing complexity of the reconfigurable systems design [2]. Such an autonomy allows to divide the control problem between autonomous controllers having each a local vision of the system, which allows to decrease their design complexity.

Another solution to decrease design complexity is to elevate the low-level technical details using Model-Driven Engineering (MDE) [3]. MDE is a high-level design approach that is suitable for SoC co-design, allowing to model both software and hardware parts. In order to use the MDE for a high level description of a system in a specific domain such as embedded systems, UML (Unified Modeling Language) [4] profiles are used. A UML profile is a set of stereotypes that add specific information to a UML model in order to describe a system related to a specific domain. Several UML profiles target embedded systems design such as the Modeling and Analysis of Real-Time and Embedded systems (MARTE) [5] profile, a standard profile promoted by the Object Management Group (OMG).

In a previous work [6], we proposed a distributed control model for reconfigurable FPGA-based systems. This model is based on autonomous modular controllers and formalism-oriented design in order to decrease design complexity and facilitate design reuse and scalability. In this paper, we enhance this work by a high-level modeling approach. This approach allows designers to target dynamic reconfiguration without being expert of modern FPGAs, which simplifies the control design and decreases its complexity. We also integrate this high-level control design into a model-based SoC design framework in order to generate automatically controlled reconfigurable FPGA-based systems, which allows to reduce time-to-market and to enhance design productivity. The rest of the paper is as follows. Section 2 gives a summary of the related works on high-level modeling. Section 3 gives an overview of the proposed control distribution. Section 4 presents the high-level modeling approach through a video processing application case study. Section 5 gives experimental results. The last section presents a conclusion and future works.

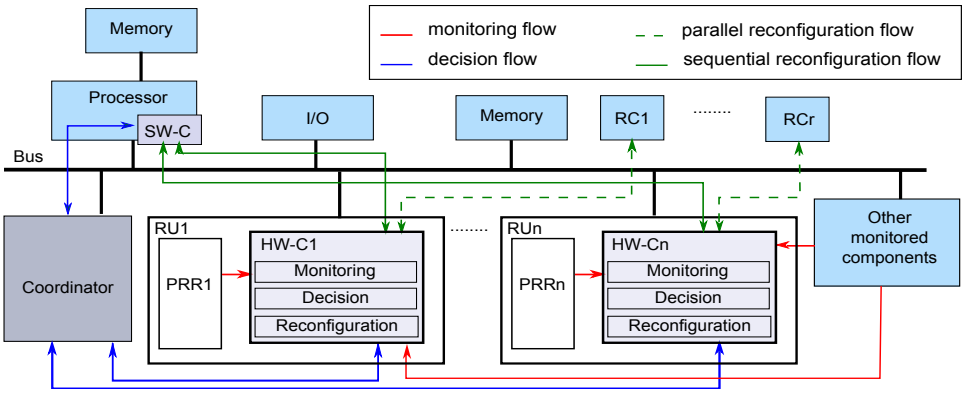

Fig. 1: Considered reconfigurable system

II. Related work

High-level modeling for reconfigurable systems was proposed in several works. In [7], the authors present a co-design methodology for reconfigurable MPSoC (Multiprocessor SoC), in order to generate automatically reconfigurable embedded systems using the MARTE UML profile. The proposed solution for control modeling is extremely simplistic, based on a UML state machine handling a repeated sequence of configurations for one reconfigurable co-processor. This solution is not suitable for more complex reconfiguration requirements. Besides, reconfiguration aspects were only handled at the allocation level. In [8], the authors propose a MARTE-oriented design approach of dynamic reconfigurable FPGA (Field Programmable Gate Arrays) -based RSoC using mode automata control for the Gaspard2 SoC design framework [9]. However, the proposed approach is based on a centralized reconfiguration controller for the whole FPGA, which makes it not suitable for large complex systems. In [10], the authors present a design approach for dynamic reconfigurable FPGA-based RSoC. This approach is based on the MARTE concepts for reconfiguration control such as mode and configuration concepts. However, it has the limit of remodeling the whole system for each possible configuration. There are also several other model-driven approaches for RSoC control that are not MARTE-oriented [11] [12]. Compared to the mentioned works, our approach allows to handle adaptation control design at different design levels (application, architecture, allocation and deployment), which offers a high design flexibility and allows to cover a wide range of control requirements.

III. The control distribution approach

The considered reconfigurable systems in this paper are those supporting partial reconfiguration. Figure 1 shows an overview of the targeted architecture. It is mainly composed of a processor, a number of reconfigurable hardware units (RUs), memories, and I/Os. Application tasks are thus composed of software tasks executed by the processor, and hardware tasks executed by the RUs. In order to decrease the control design complexity of such systems, we propose to divide control concerns between distributed controllers in order to decrease their design complexity and facilitates their verification and reuse, resulting in a higher design productivity. Adaptation control is distributed as follows: 1) a software controller (SW-C) handling the software tasks adaptation, 2) a number of hardware controllers (HW-Cs) handling RUs adaptations. Each RU is composed thus of a partial reconfigurable region (PRR), handling a hardware task, and a HW-C controlling it. Software and hardware controllers are coordinated using a coordinator, in order to guarantee that the system configuration respects global constraints/objectives.

As shows Figure 1, each HW-C contains three modules handling monitoring, decision and reconfiguration. Monitored data (monitoring flow in Figure 1) can be related to the internal status of PRRs or detected from external sources (other monitored components in Figure 1), such as battery sensors, user inputs, etc. Monitoring modules detect, from monitored data, information and events that will be used by decision modules for decision-making. Each decision module makes local decisions about whether or not a reconfiguration of the controlled region is required. Due to the local vision of each HW-C, launching a reconfiguration of its controlled region without checking whether it can coexist with the current configurations of the other regions might result in problems such as safety problems or might not respect the control global constraints such as those related to performance, temperature, energy consumption, etc. Therefore, before launching a reconfiguration that it estimates required according to the monitoring data, the Hw-C has to send a reconfiguration request to the coordinator (decision flow).

The role of the coordinator is to check whether the global system constraints allow to launch the requested reconfiguration without adapting other regions. If so, the coordinator authorizes the reconfiguration. Otherwise, it sends reconfiguration suggestions to the HW-Cs of the involved regions. According to the responses of the HW-Cs (acceptance or refusal), the coordinator gives its decision about the requested reconfiguration. This decision can be either an authorization or a refusal of the reconfiguration. More details about the used coordination mechanism can be found in [6]. In case of authorization, the coordinator notifies the HW-Cs. Their

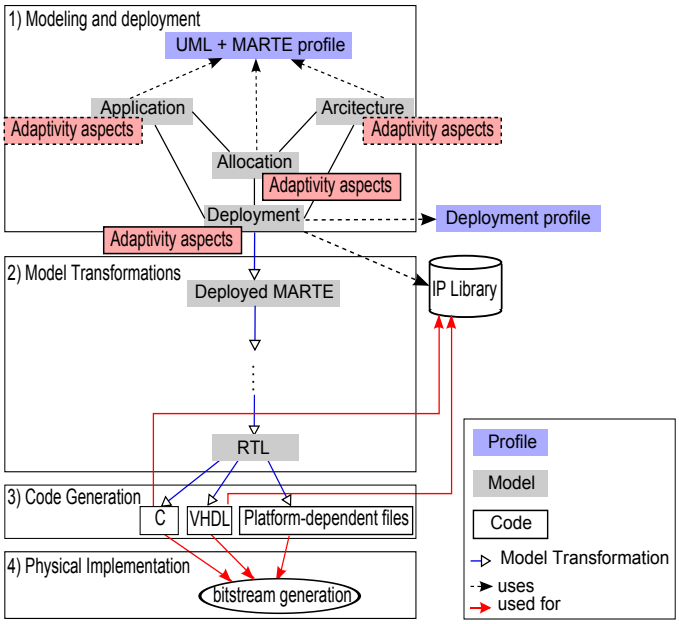

Fig. 2: Adaptation control integration to the Gaspard2 framework

reconfiguration modules launch then reconfiguration through reconfiguration controllers (RCs) in order to load the required partial bitstreams in a parallel way (parallel reconfiguration flow), which reduces reconfiguration time. For a Xilinx platform, these RCs correspond to ICAP controllers. The coordinator notifies also the SW-C so that adapts the application software tasks to the new system configuration.

In order to adapt to technical limitations of current FPGAs, which mostly contain one configuration port (ICAP) and do not allow thus parallel reconfiguration, our control design approach proposes a second reconfiguration flow. In case of a reconfiguration authorization from the coordinator, each reconfiguration module updates a dedicated register indicating which mode is to be loaded in the controlled region. These registers are then read by the SW-C, which communicates with the ICAP port in order to load the required bitstreams in the reconfigurable regions. Then, the SW-C notifies the coordinator and the HW-Cs so that they update their "current configuration" registers. Taking into account the new system configuration, the SW-C updates application software tasks before sending data to reconfigurable regions. This control flow is represented by the sequential reconfiguration flow in Figure 1.

IV. The High-Level Modeling Approach

Gaspard2 [13] is an MDE oriented SoC co-design framework based on the Modeling and Analysis for Real Time and Embedded systems (MARTE) standard. Our contribution in this framework is to integrate reconfiguration aspects in the design flow, by integrating adaptivity aspects at different design levels: application, architecture, allocation and deployment, as shows Figure 2. This is an enhancement to previous control modeling approaches in the Gaspard2 framework [8] [14], which targeted centralized control and limit control modeling at one design level. The design flow in Gaspard2 follows several steps: 1)system modeling and deployment, 2)model transformations, 3)code generation, and 4)physical implementation, as shows Figure 2. More details about these steps can be found in [13]. Gaspard2 allows code generation targeting different languages such as Fortran, SystemC, OpenCL, C, VHDL, etc. In Figure 2, only the used languages in this paper are mentioned (C and VHDL). The generation tool generates also platform (FPGA)-dependent files. The generated code can then be integrated into FPGA tools for synthesis, PAR and bitstream generation. It can also be integrated into the IP library in order to be reused in other system designs at the deployment level.

The case study concerned in this paper deals with video scaling, which is considerably important for previews or for streaming for small form factor devices, such as mobile phones. The application is a classical downscaler, which transforms a video signal, which is expressed in Common Intermediate Format (CIF:352x288 pixels), into a smaller size video (132x128 pixels). This application is composed of two main tasks: a horizontal filter and a vertical filter. By applying the horizontal filter on the input frame, its size is reduced to 132x288 pixels. Then, by applying the vertical filter, we obtain the target size (132x128 pixels). Each filter is composed of a repetition of an elementary task. At each iteration of this task, a frame block is treated. In order to guarantee a high performance of the application, both filters are implemented in hardware using hardware accelerators. Each accelerator performs an elementary task of one of the filters. Using different parallelism degrees, each filter can be implemented by more than one accelerator. The input block sizes of the accelerators can also be varied in order to obtain different performance and power results. Indeed, using a bigger input block size for an accelerator reduces the execution time thanks to a higher hardware parallelism, at the cost of a higher resource overhead and thus a higher power consumption.

In our case study, the objective of the control is to adapt the downscaler application to changes in performance and power requirements. Each accelerator is implemented in a reconfigurable region using three different versions, varying the size of the input blocks. In this case study we consider the following block sizes: 35, 19 and 11 pixels for the horizontal filter, and 41, 23 and 14 pixels for the vertical filter. Here, for both filters, the first version corresponds to the highest performance but at the same time the highest power consumption. In order to take performance and consumption requirements into account at runtime, each HW-C monitors two elements. The first element is the required performance level, which is a user input. The second is the battery level sent by a battery sensor.

In the rest of this section, we detail our approach to integrate adaptation control modeling in the Gaspard2 framework, the application, architecture, allocation and deployment levels.

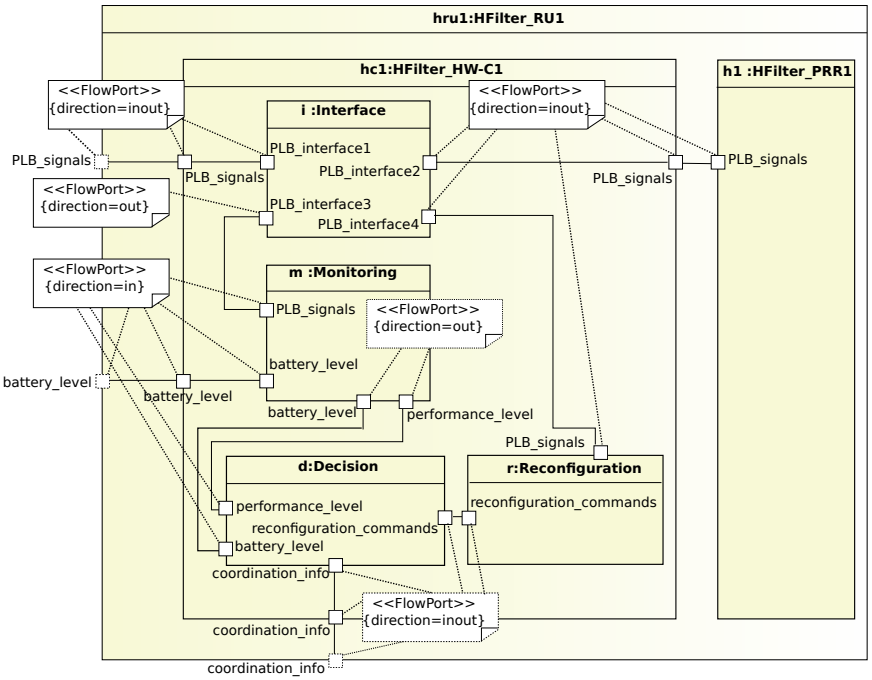

Fig. 3: The RU modeling

A. Application modeling

An application is composed of hardware and software tasks. Each task is modeled by a component. Inputs and outputs of a task are represented by ports having the *FlowPort* stereotype of MARTE. More details about application modeling in Gaspard2 can be found in [13]. In order to integrate adaptation aspects, each reconfigurable task is modeled by a number of components, representing each a version of the related task. In our case study, a hardware task corresponds to an elementary filter task executed by a PRR. Three versions ($HwHFilter_i/HwVFilter_i, i \in [1..3]$) are possible for both filters, as we said previously. These versions have different input and output block sizes. Software tasks are divided to a number of static tasks and a reconfigurable task. Static tasks perform frame reading and building before and after the downscaler execution, respectively. The reconfigurable task is the task that sends input frame blocks to RUs and retrieves output blocks afterwards. The configuration of this task depends thus on the current configurations of the RUs whose inputs/output block sizes is different from a configuration to another. As we said previously, the coordinator guarantee that the system configuration respects global system constraints. In this case study, we assume that these constraints require that all the RUs implement the same version number. In this case, the reconfigurable software task, has three versions $Downscaler_i, i \in [1..3]$, where each $Downscaler_i$ communicates with RUs implementing $HwHFilter_i$ and $HwVFilter_i$.

B. Architecture modeling

The architecture model contains the hardware system implemented on FPGA, on which the application will be running. More details about architecture models in Gaspard2 can be found in [13]. Here, we focus on the adaptation control integration at the architecture level. For this, we integrate RUs modeling at this level. For our case study, we choose to implement the application with two RUs for horizontal filter and two for the vertical filter. The architecture follows the concepts presented in Figure 1. The reconfigurable part of the system corresponds to 4 PRRs. Each PRR is controlled by a HW-C forming thus 4 RUs. HW-Cs are connected to a coordinator. The inputs of the HW-Cs are the battery level sent by the battery sensor, the bus signals, and the coordination information. In order to offer the possibility for the user to introduce the required performance level, the system contains push buttons and a interruption controller to handle the user commands as interruptions. The system contains a SysACE controller for accessing the partial bitstreams placed in a Compact Flash, an ICAP controller to carry out partial reconfiguration using the read-modify-write mechanism, and the DDR3 to store the input and downscaled frames. The compact Flash will also be used to store the images to be downscaled. This can be replaced by a camera connected to the FPGA in order to downscale its input video. The system contains also a TFT (Thin Film Transistor) controller in order to display images before and after downscaling, and a UART allowing the user to follow the application and the control progress.

Figure 3 shows the structure of one of the RUs used for the horizontal filter. The rest of the RUs follow the same concept. Each HW-C is composed of four components: monitoring, decision-making, reconfiguration and interface. The interface component allows the communication of the processor with

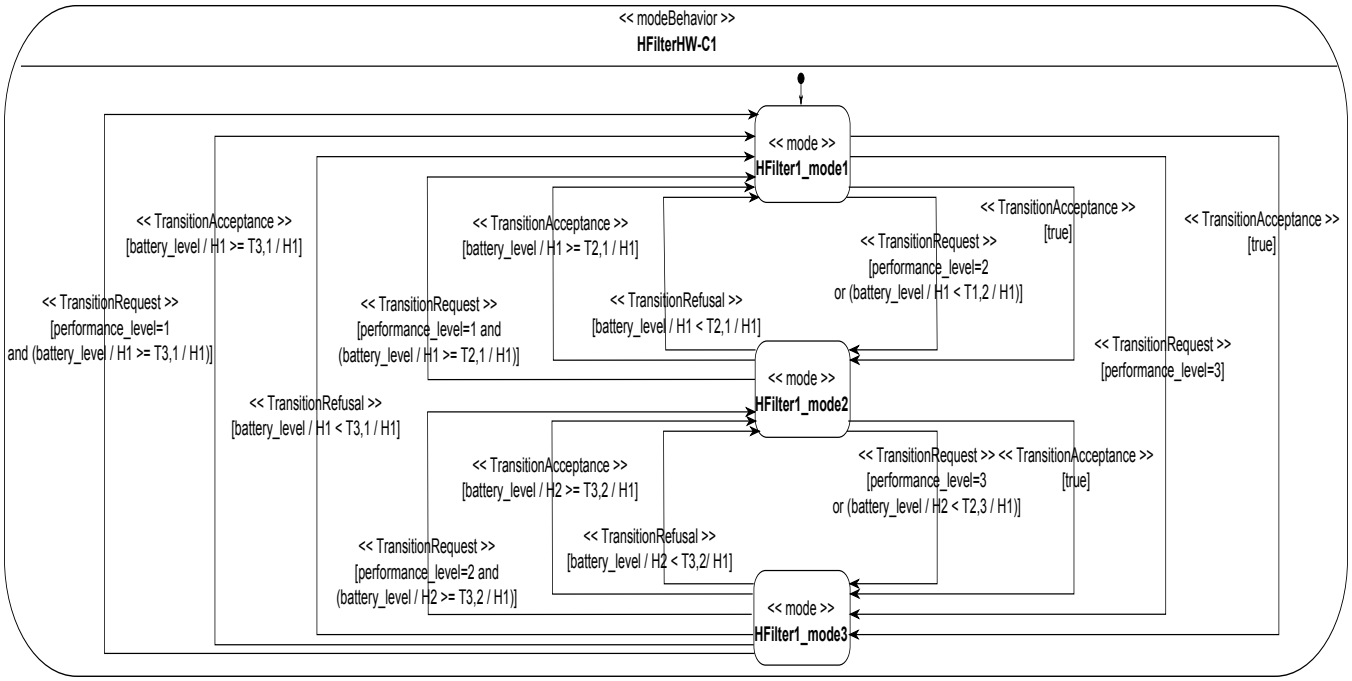

Fig. 4: The HW-C's mode-automaton

the PRR and the reconfiguration component. It allows also the monitoring module to have the necessary information for the monitoring through the *PLB_interface3* port. The monitoring component inputs are the PLB signals (allowing to extract the user performance level since they are sent by the processor), and the battery level sent by the battery sensor. The decision component inputs are the monitoring outputs modeled by the $performance\_level$ and $battery\_level$ ports, and the coordination information through the $configuration\_info$ port. After a reconfiguration is authorized by the coordinator, the decision component notifies the reconfiguration component through the $reconfiguration\_commands$ port. The reconfiguration component has two ports. The first one allows it receive reconfiguration commands from the decision component and to notify this latter when the required reconfiguration has finished. The second one allows it to communicate with the processor as explained previously.

In order to model the behavior of the HW-C, we associate to its decision component a mode-automaton, where each mode corresponds to a given configuration/mode of the controlled RU. Each mode represents thus a version of the hardware tasks mentioned previously. The mode-automaton is modeled using the MARTE stereotypes. Inspired from reactive mode automata formalism [15], a mode in MARTE is an extension of the UML state, and mode behavior is an extension of the UML state machine. The mode-automaton has the *modeBahavior* stereotype, and each mode has the *mode* stereotype. Transitions have the *modeTransition* stereotype, and give the events triggering the transition and the activity that has to be carried out to move from a mode to another. Provided that all reconfigurable regions have three configuration possibilities, each Hw-C uses a decision module modeled by a three-mode automaton. In order to simplify the modeling of the Hw-C decisions, we added extensions to the MARTE profile. These extensions allow to facilitate the modeling the decision module activities (requests, suggestion acceptance and suggestion refusal), by making them implicit in transitions. Figure 4 shows the mode-automaton of the HW-C modeled in Figure 3. Three stereotypes are added and are applied to transitions as extensions of the *ModeTransition* stereotype. The *TransitionRequest* stereotype indicates that the decision module sends a request to the coordinator asking for a reconfiguration to the target mode if the transition condition is valid. Likewise, *TransitionAcceptance* and *TransitionRefusal* stereotypes indicate that the decision module accepts or refuses the received reconfiguration suggestion to the target mode. Although these stereotypes are applied to transitions, at the implementation level (the decision mode-automaton to be generated), they are not actual transitions to the target mode since, in the targeted control implementation, the actual transition is only done after the loading of the corresponding configuration (partial bitstream). However, using these extensions allows to hide many implementation details (which are handled automatically by the model transformation and code generation tools) for designers. This simplifies significantly the control design. The designer has only to mention the transition conditions, since the transition activities are implicit (request, acceptance and refusal). In our case study, the decision making process of each decision module is mainly based on the following rules, which are represented by the transition conditions in Figure 4:

- Being at $H/VFilter\_mode_{j1}$, a controller decides that a reconfiguration to a less consuming mode

$H/VFilter\_mode_{j2}$ is required, only if the user requires a lower performance level, or the consumption constraints do not allow to stay at $H/VFilter\_mode_{j1}$, which is the case when the following condition is valid

$$battery\_level/H_{j1} < T_{j1,j2}/H_1 \quad (1)$$

where H_j (V_j for the vertical filter) is the energy consumption per cycle of the controlled region's mode $H/VFilter\_mode_j$. This constraint allows to check whether the available energy is under a threshold (determined by $T_{j1,j2}$) that allows to stay at $H/VFilter\_mode_{j1}$, taking as a reference the highest consumption (H_1).

- In order to move from a $H/VFilter\_mode_{j2}$ to a $H/VFilter\_mode_{j1}$ that consumes more, it is necessary that the user requires a performance level that is higher than the previous one and that the consumption constraints allow to move to the target mode, which is the case when

$$battery\_level/H_{j1} >= T_{j2,j1}/H_1 \quad (2)$$

- If the controller receives a reconfiguration suggestion from the coordinator it treats it as follows. If the suggestion requires to move to a less consuming mode, the controller accepts directly. Otherwise, the controller checks the consumption constraints in (2) in order to accept or refuse.

As for the coordinator modeling, it can be done by defining global system constraints using a contract-based language as presented in [14]. However, contracts cannot be modeled using MARTE. Therefore, MARTE has to be extended in order to handle further control needs. This extension is out of the scope of this paper. In the current implementation, the coordinator is written directly in VHDL and stored in the IP library in order to be attached later to the design at the deployment level.

In order to model the SW-C, we associate a mode-automaton to the processor. Each mode of the automaton corresponds to one configuration of the system. Here, we speak about a system configuration because in our case, in addition to software adaptation, the processor has to have a vision of the PRR configurations in order to launch the PRRs reconfiguration correctly. This global vision will be modeled later at the allocation model. As we said previously, we assume that global system constraints require that all the RUs implement the same mode number. Therefore, there are three possible system configurations, which correspond to a three-mode automaton. Transitions from a mode to another is handled by the SW-C that the processor will implement. The determination of the required system configuration will be handled by the code generation tool. This tool will generate the necessary code allowing the processor to communicate with the HW-Cs in order to extract the required system configuration and launch partial reconfiguration accordingly. As for the required partial bitstreams, they will be indicated later at the deployment level, so that the code generation tool inserts them in the SW-C generated code.

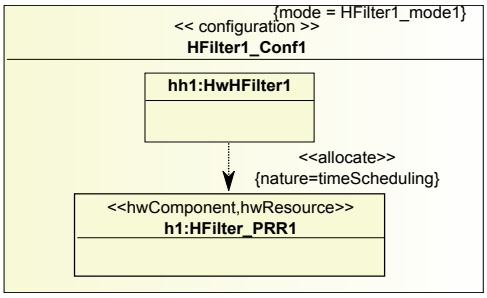

Fig. 5: Allocation of a hardware task on a PRR

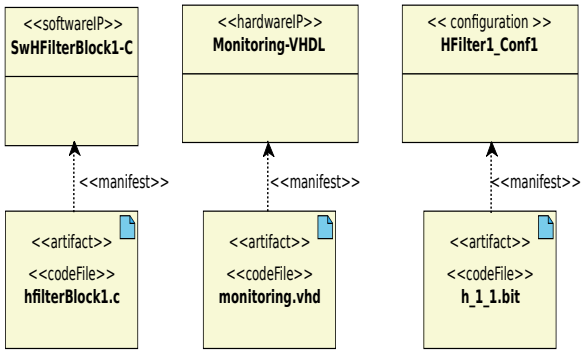

Fig. 7: Components deployment

C. Allocation modeling

The allocation model allows to allocate application tasks to architectural components (the processor and the PRRs). For this, Gaspard2 uses the MARTE *allocate* stereotype, which is an extension of the UML abstraction. In order to integrate adaptation aspects at this level, we use the *configuration* MARTE stereotype, in order to model the different configurations of the PRRs as well as the reconfigurable software tasks. Figure 5 shows a part of the allocation model. It gives, as example, the configuration corresponding to the first mode of a PRR implementing the horizontal filter, following the mode-automaton in modeled in Figure 4). This *configuration* indicates the allocation of a version of a hardware task ($HwHFilter1$) to the PRR ($HFilter\_PRR1$). For this, the property *nature* of the *allocate* stereotype is set to *timeScheduling* in order to indicate to the generation code tool that the PRRs will be reconfigured in order to implement different hardware tasks. In order to enhance design reuse, our approach reuses the PRR configurations when modeling system configurations, as shows Figure 6. System modes are also modeled by configurations. Each configuration combines the allocation of the software tasks to the processor, and the PRRs configurations corresponding to the considered system mode. Reconfigurable software tasks are allocated using *timeScheduling* for the nature property of the *allocate* stereotype, as shows Figure 6.

D. Deployment modeling

The deployment level allows to link existing IPs to the modeled components in order to add details allowing to get closer

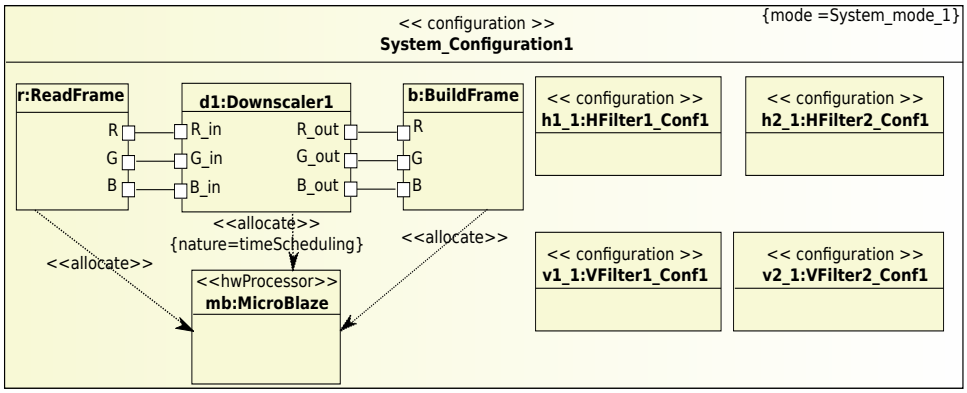

Fig. 6: Modeling of a system configuration

to the targeted platform. At this level, we used the Gaspard2 deployment profile. Stereotypes *softwareIP* and *hardwareIP* are used to indicate several platform-dependent attributes such as the implementation language and other parameters. The *softwareIP* stereotype is used for the application software elementary components. The *hardwareIP* stereotype is used for the application and architecture hardware elementary components. More details about Gaspard2 deployment can be found in [13]. All the elementary components of our design are deployed using the *softwareIP* and *hardwareIP* stereotypes, except for the PRRs since they have different implementations. Existing code files are attached to elementary components through the *codeFile* stereotype, which is an extension of the UML artifact. We associate VHDL *codeFiles* to hardware IPs, and C *codeFiles* to software IPs. For the PRR deployment, we associate *codeFiles* to the different PRR *configurations*. These *codeFiles* indicate the name of the corresponding bitstreams as shows Figure 7, which gives a part of the deployment model handling the deployment of a software task, a hardware component (the monitoring component), and a PRR configuration. As for the HW-Cs, we associate code files to the interface, monitoring and configuration components, whereas decision modules will be generated automatically from the modeled mode-automata.

V. Experimental results

After deployment, automatic model transformations are carried out to add some details to the input model and get closer to the targeted technologies. In order to take control aspects into account in model transformation and code generation, we integrated the proposed control aspects in the Gaspard2 metamodels as well as model transformation rules, in order to get at the end of the transformation chain an RTL model that conforms to the extended RTL metamodel, as shows Figure 2. This model contains technical details allowing the code generation related to the targeted FPGA technology. The code generation tool was extended as well in order to take control aspects into account. This tool allows to generate the code to be integrated into the Xilinx tools for physical implementation. The output of the code generation tool are C code, VHDL code and other platform dependent files as shows Figure 2. Using deployment information and code generation rules, the code generation tool generates the following elements: 1) a software project for the processor. This project contains the software application tasks to be executed by the processor as well as the code of the SW-C, handling software adaptation and the communication with distributed reconfiguration modules and the ICAP, 2) the Xilinx MHS (Microprocessor Hardware Specification) file, describing the architecture to be implemented on FPGA, 3) the UCF (User Constraints File) file, which allows to allocate the hardware component external ports to the FPGA pins, 4) a VHDL project for each RU.

Given to the Xilinx EDK tool, the generated files allowed to generate the system netlists. After system synthesis, the Xilinx PlanAhead tool is used to place the PRRs. This tool takes as input the netlist file (an ngc file) generated by the EDK tool, as well as the UCF file. To each PRR, we associate a number of netlist files corresponding the synthesis result of the different hardware tasks to be implemented by it. Bitstreams can then be generated automatically.

In our case study, three partial bitstreams were generated for each PRR, as well as a full bitstream for the initial system configuration. The horizontal and vertical filters partial bitstreams have as size 193Ko and 365Ko respectively. This difference is due to the fact that the horizontal filter treats larger input and output blocks. Reconfiguration times of horizontal and vertical filter PRRS are 4,3ms and 8,1ms respectively. Table I shows the performance and power consumption results for the system configurations. In our case study, the battery sensor was emulated by a hardware module that decrements at each cycle an energy counter depending to the current configuration. This counter was initialized with 1000J. The last column of Table I shows the number of frames that can be treated if the system keeps the same configuration until the battery is flat. Experimental results show that partial reconfiguration allows to make a trade-off between performance and power. Indeed, without partial reconfiguration, if all the hardware accelerators implement the first mode, the battery (1000J) allows to downscale only 2361 frames. Using partial reconfiguration allows

System mode	Execution time for a video frame (cycles)	Power consumption (mW)	Number of treated frames
System_mode_1	69082595	613	2361
System_mode_2	76604837	564	2444
System_mode_3	92092852	530	2585

TABLE I: Performance and power results for the case-study system

to handle user performance requirements and to save, at the same time, power and to treat thus higher numbers of frames (up to 9,4% of increase) for acceptable reconfiguration times. This trade-off can be further refined by exploring different implementations of the PRRs, but in this paper, we focus on the validation of the distributed decision-making. This validation was done by testing different scenarios by varying the values of the user performance level and the moments it is modified, which allowed to test all transition conditions of Figure 4 for all controllers. Due to space limitation, these scenarios are not detailed in this paper. More details about the distributed decision-making mechanism can be found in [6].

In order to evaluate the efficiency of our control model compared to the centralized one in terms of design reusability and scalability, we varied the application parallelism in order to implement higher numbers of PRRs for each filter. Experiments were carried out for a number of PRRs up to $n = 10$ regions, where $n/2$ regions implement the horizontal filter and the rest the vertical filter. The HW-Cs generated previously were reused in order to target larger control systems. For the coordination scalability, we only modified the number of coordinated controllers given as parameter to the coordinator, as well as the global constraints handled by it. The same systems were designed using a centralized controller. We noticed that adapting the centralized controller to different numbers of regions was more complicated. Indeed, for the centralized controller the decision-making had to be rewritten each time to adapt to the system implementation, which led to longer design phases.

VI. CONCLUSION

This paper presents a novel design approach for RSoC adaptation control. This approach combines distributed control together with high-level design in order to decrease design complexity, enhance design flexibility and reuse, and automate code generation. It is based on splitting the control concerns between a number of distributed controllers. A coordinator is used to guarantee that the local decisions made by distributed controllers respect global system constraints. Using Model-Driven Engineering and the MARTE standard, adaptation aspects were integrated in a whole RSoC design flow at different design levels (application, architecture, allocation and deployment). Generated code was then integrated into FPGA tools for physical implementation. FPGA implementations allowed to validate the distributed control and to show that it is more flexible, reusable and scalable than the centralized one. As future works, we plan to explore different control approaches such as those based on optimization problems in order to target a wider range of control objectives and reconfigurable systems. The MARTE profile can also be extended to cope with further control needs allowing, for instance, to model control in form of contracts and generate the coordinator automatically. The MARTE standard is being reworked is the ANR project FAMOUS [14], which is more dedicated to reconfigurable systems compared to Gaspard2, and in which we plan to integrate our work.

VII. ACKNOWLEDGMENT

The authors would like to thank the ANR project FAMOUS for the financial support of this work.

REFERENCES

[1] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford, "Invited paper: Enhanced architectures, design methodologies and cad tools for dynamic reconfiguration of xilinx fpgas," in *FPL*, 2006, pp. 1–6.

[2] E. Mainsah, "Autonomic computing: the next era of computing," pp. 2–3, 2002.

[3] OMG, "Portal of the model driven engineering community," http://www.planetmde.org.

[4] ——, *UML Superstructure*, 2009.

[5] ——, *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*, 2011.

[6] C. Trabelsi, S. Meftali, and J.-L. Dekeyser, "Semi-distributed control for fpga-based reconfigurable systems," in *15th Euromicro Conference on Digital System Design*, 2012.

[7] J. Vidal, F. de Lamotte, G. Gogniat, J.-P. Diguet, and P. Soulard, "Uml design for dynamically reconfigurable multiprocessor embedded systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10, 2010.

[8] 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*, ser. DASIP '09, 2009.

[9] O. Labbani, J.-L. Dekeyser, P. Boulet, and E. Rutten, "Introducing control in the gaspard2 data-parallel metamodel: Synchronous approach," 2005.

[10] I. R. Quadri, A. Gamatie, P. Boulet, and J.-L. Dekeyser, "Modeling of configurations for embedded system implementations in marte," in *1st workshop on Model Based Engineering for Embedded Systems Design - Design, Automation and Test in Europe*, ser. DATE '10, 2010.

[11] S. Pillement and D. Chillet, "High-level model of dynamically reconfigurable architectures," in *Conference on Design and Architectures for Signal and Image Processing*, ser. DASIP '09, 2009.

[12] R. Gumzej, M. Colnaric, and W. A. Halang, "A reconfiguration pattern for distributed embedded systems," *Software and System Modeling*, vol. 8, pp. 145–161, 2009.

[13] A. Gamatie, S. L. Beux, E. Piel, R. B. Atitallah, A. Etien, P. Marquet, and J.-L. Dekeyser, "A model driven design framework for massively parallel embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 10, 2011.

[14] S. Guillet, F. de Lamotte, E. Rutten, G. Gogniat, and J.-P. Diguet, "Modeling and formal control of partial dynamic reconfiguration," *Reconfigurable Computing and FPGAs, International Conference on*, vol. 0, pp. 31–36, 2010.

[15] F. Maraninchi and Y. Remond, "Mode-automata: a new domain-specific construct for the development of safe critical systems," 2003.