Patterns System for the Design of Partial Reconfigurable Applications on FPGA

Nissaf Fredj[1], Mhamed Saidane[2], Yessine Hadj Kacem[3] and Mohamed Abid[4]
[1]*ISITCOM, CES Laboratory, ENIS, Sfax, Tunisia*
[4]*CES Laboratory, ENIS, Sfax, Tunisia*
[2]*TIM Laboratory, Monastir, Tunisia*
[3]*College of Computer Science, King Khalid University Abha, Saudi Arabia*

Keywords: DPR, Patterns System, Behavioral Pattern, Architectural Pattern, DP-RTE Systems.

Abstract: During the last few years, the Dynamic Partial Reconfiguration (DPR) has been introduced to the embedded systems as a key technique that aims at improving the flexibility of Field-Programmable Gate Array (FPGA)-based system reconfiguration. However, the design of these systems is a hard task using low-level functions where the design of the hardware side precedes that of the software. Recently, Model-Driven Engineering (MDE) based approaches have emerged. They aim at simplifying the modeling of the dynamically set systems and keep a design flow where DPR application and architecture are designed in parallel. In fact, there is a lack of reusable and generic models that allow the improvement of the designers' task and the decrease of the development costs. In order to overcome these issues we propose in this paper an additional featuring or abstraction level in the DPR design flow introduced by these approaches. Our aim is to suggest for designers a method (process and models) which allows reusing recurrent application models and sharing experience-owned knowledge. The proposed method is a patterns system which is a combination of architectural and behavioral patterns dedicated to the Dynamic Partial reconfigurable Real-Time Embedded (DP-RTE) systems.

1 INTRODUCTION

The real-time embedded systems have become so useful in our life. They are mainly related to the image and signal processing applications in which a significant quantity of data is regularly processed through repetitive calculations. Embedded systems, and particularly DP-RTE systems, are more complex and challenging to develop compared to software systems. In addition to requiring high computing power at considerable speeds, they are subject to a multitude of constraints such as resource limitations and execution time (Henzinger et al., 2007). The growing complexity and the high design costs of these systems pushed the designers to apply the DPR technique. Indeed, a DP-RTE system offers a high functional flexibility while maintaining good performance (Marques, 2012). These systems can be reconfigured for an unlimited number of times. They offer the ability to add new features and make changes to the system after its creation. However, the design of such systems has become an expensive task in terms of time. It also requires a broad knowledge of the technical details of the target platforms. Facilitating the work of DP-RTE systems designers and reducing the development costs and time, is a major challenge in this field. In response to these issues, several high-level design approaches (Beux, 2007) (Ochoa-Ruiz et al., 2012) have been emerged: It is a high-level co-design under the Y-model where the application and the architecture are designed in a parallel way. The development offers for designer's flexibility, reusability and automation as well as it hide technical details. It is based on MDE (Schmidt, 2006) and MARTE (OMG, 2011) (Modeling and Analysis of Real-Time and Embedded Systems) profile. Nevertheless, there is a lack of reusable and generic models that speeding and facilitating the reuse of these complex systems. A pattern presents an applied solution to share experience owned knowledge and generic terms to obtain fast and widely used designs (Gamma et al., 1995). However, most of the research studies based on pattern solution are oriented towards software systems. Furthermore, they do not deal with DP-RTE system development and ignore important

Fredj, N., Saidane, M., Kacem, Y. and Abid, M.
Patterns System for the Design of Partial Reconfigurable Applications on FPGA.
DOI: 10.5220/0006383503250335
In *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2017)*, pages 325-335
ISBN: 978-989-758-250-9

adaptation proprieties of these systems.

In this paper, we propose a new method for the DP-RTE systems design under the Y-model. Our contribution is formalized as a patterns system formed with process patterns that present fragments of demarches based on proposed criteria to assist and guide the designer to product patterns. Our objective allows DPR designers to reuse recurrent solutions that have been validated by experience and handle the real-time features of the DPR process. This work is characterized by a new novelty which is the description of the proposed patterns using MARTE profile. Using a rich terminology for the specification and analysis of embedded systems, MARTE enables a joint design of the hardware and software parts of embedded systems.

This paper is organized as follows: the second section summarizes the work on the high-level design of dynamically reconfigurable systems and the pattern-based adaptation. The proposed patterns system is presented in the third section and it will be illustrated by a case study in the fourth section. Finally, the last section is dedicated to conclude the paper and present our future work.

2 RELATED WORKS

2.1 High Level Design of DPR

Several approaches have investigated the design of DPR process such as the design of the reconfigurable application. This design has been evolved from one work to another through the tagged design flow. Previously proposed solutions were mainly based on the MDE approach. The main contributions in (Cherif, 2013) are the modeling of a deployment level, a physical platform and a control level. The design flow is inspired by GASPARD's work (Gamatie et al., 2008). This flow begins with a joint modeling of the application and the architecture as well as the mapping between them. Next, the authors in (Cherif, 2013) proposed an UML/MARTE design flow for the automatic generation of RTL code to be implemented on dynamically reconfigurable FPGAs. They added to the MARTE profile a set of stereotypes such as the ReconfigurableRtUnit stereotype to model the reconfigurable tasks of applications in the DPR process on Xilinx FPGAs. Other approaches, which have addressed the modeling of reconfiguration control, have been integrated into GASPARD project. First results in (Cherif et al., 2011) proposed a high level modeling of a distributed modular controller to manage the reconfiguration in FPGAs. The MARTE profile- based approach was improved by (Chiraz, 2012) to model the semi-distributed control as a set of distributed modular controllers which performs the observation, decision-making, reconfiguration tasks and coordination between distributed controller-made decisions. This Modeling aims at respecting global constraints and system objectives. Moreover, the authors in (Quadri, et al., 2010) developed a co-design approach for Soc (System on chips), in the GASPARD framework. The work made the automation of code generation from high-level MARTE models. Indeed, they use an intermediate level which provides different mechanisms to link the low level of implementation with high-level models. Furthermore, the authors extended the MARTE profile with a set of concepts to specify the DPR in modern FPGAs.

All the approaches described above are beneficial because they facilitate and fix the development of dynamically reconfigurable systems. However, they have some deficiencies. First, proposed solutions depend on the hardware platform as they are based on specific concepts and low-level models that describe the Xilinx design methodology. These approaches are only interested in the modeling without addressing the actual DPR process behind monitoring features, reconfiguration decisions and system features management. Then, these studies do not provide support for the evaluation and validation of real-time constraints and resources. Finally, they are not generic enough because they handle specific reconfiguration problems which prevent their reusability from being adapted to the new requirements and constraints of the system. The development of design patterns is a promising alternative approach to deal with the latest problem. A pattern favors the extensibility and reuse of design and gives an abstraction view of a recurring problem.

2.2 Patterns based Adaptation

In literature, there are few works that have presented patterns for the adaptation of embedded systems. Regarding the software architecture design, Gamma and al in (Gamma et al., 1995) proposed a design patterns that define the running of application and aim at specifying a dynamic behavior for predefined types of software architectures which are the master/slave, centralized, decentralized, client/server architectures. The contributions of (Schmidt et al., 2000) are the basis of a pattern language that handles problems related to concurrency and networking.

The proposed patterns consist in defining a service provided by a middleware and then specifying a generic implementation of this service. In (Corsaro et al., 2002), the author proposed a Virtual component pattern that allows the developers of middleware to have a large set of functionalities to their users. The suggested pattern, which permits the adaptation of a distributed application to the embedded systems' memory constraints, has been applied in a variety of middleware, such as the JVM (Java virtual machines).

Most of the previously described works proposed pattern solutions that are oriented software system design. Some of these works describe the embedded systems design but they ignored the hardware side which is as essential as the software. Furthermore, these latter didn't deal with real time characteristics of the system adaptation. In (Said et al., 2014), the authors proposed a pattern-based specification for adaptive embedded systems. They have developed patterns for the MAPE (Monitor, Analyzer, Plan and Execute) adaptation (Chess, 2003) loop which consists of four adaptation modules. Separately in other contexts, these modules allow the promotion of their reuse. They also promote reusability and modularity designs. The patterns take into account the management of adaptation performance as well as the evaluation of real time characteristics of adaptation modules which are important in the modeling the embedded systems. This work proposed behavior patterns of the adaptation process; however, it is not concerned with the architecture of adaptation process which is hidden behind the adaptable components and the communication activity between them.

In this state of the art, we are discussing the studies that present approaches to DPR process design and the modeling of adaptation process using patterns. The main contribution of these studies is a high-level modeling of DPR using MDE based approach under MARTE profile to both guarantee abstraction and automation. In the following, we present our contribution which is built on a new pattern-based design flow that will be the subject of the next section.

3 THE PROPOSED PATTERNS SYSTEM

Before going on to explain the contributions of our method, we begin by presenting the pattern concept: According to (Buschmann et al., 1996), it is a template which is seen as a normative model to be copied or used. There are dependency relationships between patterns. These relations form a system that connects the different consecutive patterns. As reported by (Buschmann et al., 1996), a patterns system is a collection of patterns accompanied by a guide for their implementation, use and combination. The patterns must be weaving together in a cohesive whole that shows the inherent structures and relationships in each of its components to achieve a common goal. In the case of a complex system with tens patterns, it is necessary to have classification criteria.

Our approach focuses on the application part in the DPR design flow (Cherif, 2013). The method is formalized as a patterns system called PDPR (Patterns for Dynamic Partial Reconfiguration) dedicated to the DP-RTE systems; it is illustrated in figure 1. Our goal is to provide the designers with a method (process and models) to reuse recurrent application models and share experience-owned Knowledge. The system is composed of process patterns (step 2 in figure 1) and product patterns (step 3 in figure 1).

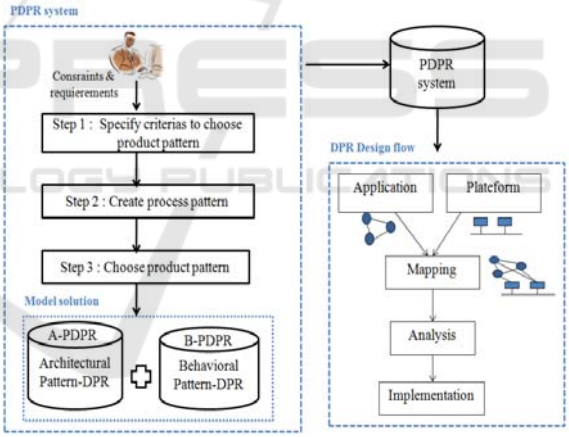


Figure 1: The Proposed PDPR system for DPR design flow.

The process pattern proposes to the designer a demarche to navigate the collection of the proposed product patterns and choose the best one that meets his needs through classification criteria (step 1 in figure 1). The product patterns allow the capitalization and reuse of model solutions. Absolutely, the model solution is composed of two types of product patterns: First, the patterns for DPR architecture (A-PDPR: Architectural Pattern for Dynamic Partial Reconfiguration) which represent the components of the system to be reconfigured as well as the communication between them. Second,

the patterns for the behavior of DPR (B-PDPR: Behavioral Pattern for Dynamic Partial Reconfiguration) that describe the reconfiguration engine process of these components. In what follows, we will present the three steps of our method: we begin by listing the proposed classification criteria. In step 2 (see Section 3.2) and step 3 (see Section 3.3), we present the different patterns of the PDPR system under the P-sigma formalism (Schmidt et al., 2000).

The main rubrics used are identifier, problem, solution, application, use, require. The rubric use and require expresses the dependency relationship between the different patterns of the PDPR system.

3.1 Step 1: Classification Criteria

In a DP-RTE system, an application is a set of tasks communicating with one other. We define an initial list of criteria to characterize tasks and communication between them. These criteria subsequently facilitate the search and selection of the product pattern that meets the designer's needs. These criteria are the synthesis of our study of the art.

- Action type: We can distinguish three types of actions (see figure 2) sent by a task: (1) Signal send: The signals are equivalent to global variables ensuring communication between the real-time units. (2) Function call: Functions are blocks of instructions that return a value. (3) Data exchange: A task can communicate with another to send or receive data. These three types of actions can invoke a DPR process (Marques, 2012) in the system.
- Task State: In a DP-RTE system and during a DPR process, we distinguish two kinds (Cherif, 2013) of tasks under their states. A static task always handles the same algorithm and sends the resulting data to the same task with which it communicates. For some input data, the calculation to be carried out is always the same regardless of the previous data or the change of environment. A dynamic task is characterized by a behavior that varies due to external factors. In other words, it is adapted to the environment.
- Synchronization: Communication between two real-time units (tasks) can be carried out in different modes: (1) the synchronous mode where the task waits for the end of the client task's execution before proceeding. (2) The asynchronous mode where the task does not wait for the execution of the client task.

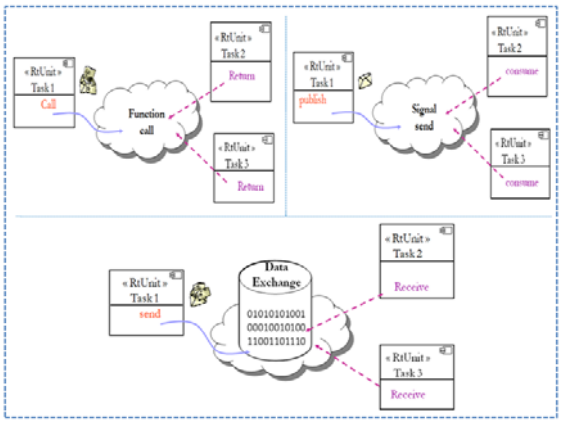


Figure 2: Action type criteria.

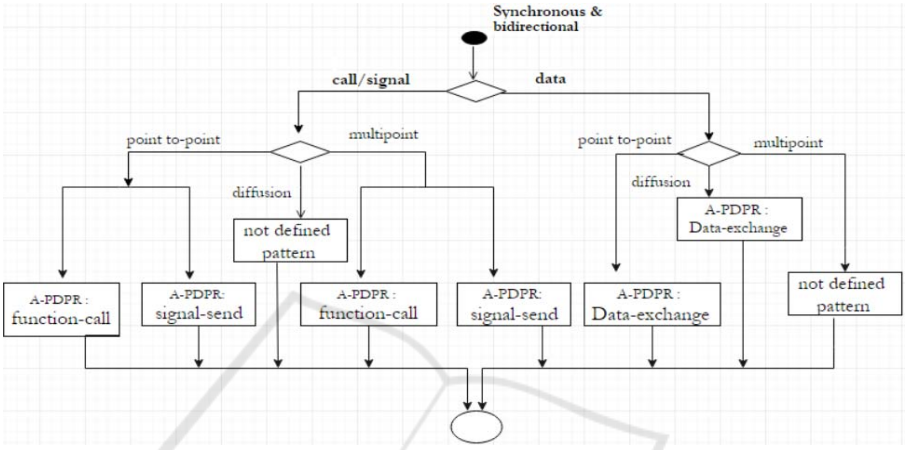
- Type of communication (exchange): We present three types (Marques, 2012) of communication. (1) A point-to-point communication between only two tasks at the same time. (2) A multipoint exchange, where the sender task sends the message simultaneously to a limited number of tasks that requested an exchange. (3) A broadcast (diffusion) communication, where the sender task sends the message simultaneously to all the tasks participating in the communication whereas those that are not concerned with ignore it.
- Direction of Communication: The exchange between two tasks can be unidirectional or bidirectional.

In our patterns system, the values of such previously presented criteria change from one product pattern to another like a set of constraints which are expressed using OCL Language (Group, 2003). The corresponding constraints are shown in see table 2 and table 5.

3.2 Step 2: Process Patterns

As already mentioned, the PDPR system consists of a set of process patterns that guide the designer to search and select one of the proposed product patterns. The entry process pattern to the PDPR system is named Input-PDPR which coordinates the use of other process patterns. The navigation in the different process patterns is carried out using the previously described criteria. Each process pattern proposes a fragment of demarche and orients the designer to a process or product pattern in the patterns system. The different process patterns are represented as an activity diagram. In what follows, one example of process patterns will be presented in table 1.

Table 1: "PDPR: Exchange-Synchronous" process pattern.

Identifier	PDPR: Exchange-synchronous
Context	This pattern requires the use of the Input-PDPR pattern.
Problem	Orients the designer when the type of synchronization between tasks is synchronous.
Demarche Solution	A simultaneous connection can only be bidirectional. This connection leads to two types of patterns, depending on the action type criteria: When the action type is a call/signal exchange. The designer is oriented towards two types of product patterns: The "A-PDPR: Signal-send" pattern or the "A-PDPR: Function-call" pattern. When the action type is a data exchange, the designer is oriented to the "A-PDPR: Data-exchange" pattern. Such patterns are not defined at this level. 
Use	A-PDPR:Signal-send, A-PDPR:Function-call and A-PDPR:Data-exchange pattern.
Require	Input-PDPR pattern.

3.3 Step 3: Product Patterns

The product patterns of the PDPR system provide a level of abstraction that allows the DPR designers to reason about the general behavior of an application without giving the details of implementation. They propose a double description: An architectural description that describes the structure of the application, which includes the tasks and interactions between them, and a behavioral description that follows and organizes the behavior of the tasks in collaboration. The patterns of the PDPR system are based on the combination of formal and semi-formal languages. The joint use of UML/MARTE and OCL to specify the model solution of the product patterns makes it possible to: increase the reuse of the proposed patterns, facilitate the understanding of the overall architecture and specify the constraints that allow controlling its reuse and its adaptation. The five proposed product patterns are illustrated respectively by table 2, table 3, table 4, table 5 and table 6.

3.4 Extensible Patterns System

The classification criteria and the demarches allow guiding the designer (DPR engineer) in his choice of the most appropriate model solution. Once he has chosen his demarche, the designer may be confronted with two situations (see figure 3) : (1) The configuration demarche leads to various models (product pattern): In this case, the reconfiguration (DPR) engineer can suggest to the patterns system (PDPR) engineer the addition of one or several criteria in order to distinguish between two neighboring demarches. This implies the addition of new demarches and the update of the different process patterns constituting it.

(2) The product pattern is not adequate to the engineer application's needs: It is necessary to elaborate a new product pattern representing the expected model solution. The application engineer can also propose new criteria to differentiate his architecture from that proposed to him. The figure illustrates the set of steps that the reconfiguration engineer and the patterns engineer must follow.

The next section is devoted to the instrumentation and the validation of the proposed PDPR system in a concrete example.

Table 2: "B-PDPR:Follow-behavior" product pattern.

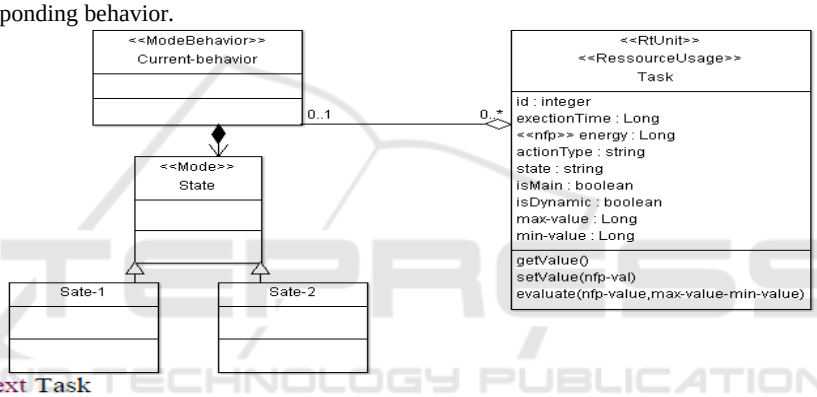
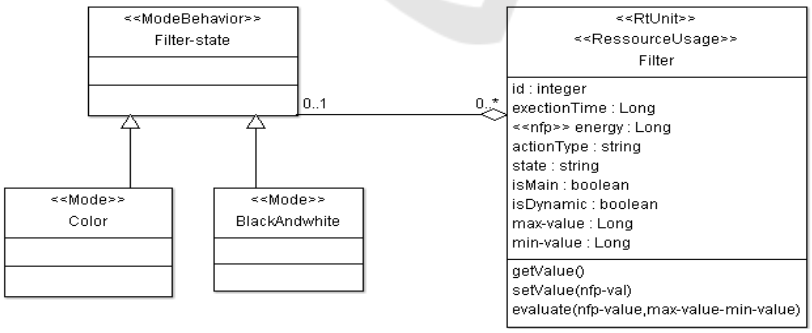
Identifier	B-PDPR:Follow-behavior
Context	If the task state is dynamic.
Problem	The pattern is applied when the designer wants to change the behavior of a dynamic task in the DPR process. It intercepts his behavior over the time and informs the rest of the application about his new state.
Model Solution	Compared to the patterns presented by the research studies [(Buschmann et al., 1996), (Schmidt et al., 2000), (Corsaro et al., 2002)], this pattern is dedicated the DP-RTE systems adaptation behavior and deals with real-time proprieties of these systems. A task is stereotyped with MARTE:RtUnit and MARTE:ResourceUsage concepts. The ResourceUsage provides a set of non-functional properties representing the consumed values of the resources. A task class defines a method evaluate () that checks whether a non-functional property has been optimized during a DPR process, the evaluation is based on minimum and maximum values. A non-functional property to be evaluated by this pattern is the consumed energy which is stereotyped with MARTE:NFP concept. The pattern is based on: (1) State interface which is stereotyped with MARTE:Mode concept, defines the behavior that specifies a set of mutually exclusive modes. (2) Concrete states (state-1, state-2) which implement the behaviors. (3) Context (Current-behavior) which is stereotyped with MARTE:ModeBehavior concept, stores the current state and calls the corresponding behavior.  context Task inv max-value, min-value > 0 context Task :: evaluate(nfp-value : Long) pre : nfp-value> 0 post : nfp-value >= min-value, nfp-value <= max-value
Application example	 A filter task can be in two different states: color or black and white. When a filter task receives requests from other tasks, it responds differently according to his current state. The pattern describes how the filter task behaves differently in each state. The key idea of this pattern is to introduce a Filter-state abstract class to represent the states of the filter. It declares a common interface to all the classes representing the different operating states. The sub-classes of Filtre-state implement specific behaviors. For example, the Color and BlackAndWhite classes implement a particular behavior for the color and the black and white states of the filter task.
Use	A-PDPR:Signal-send, A-PDPR:Function-call and A-PDPR:Data-exchange pattern.

Table 3: "B-PDPR:Abstract-behavior" product pattern.

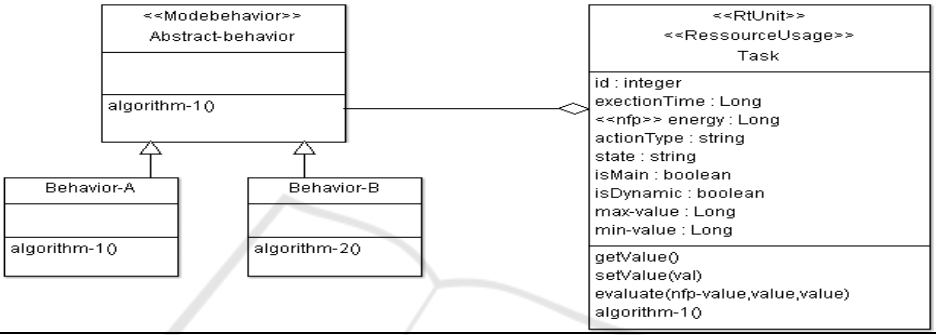
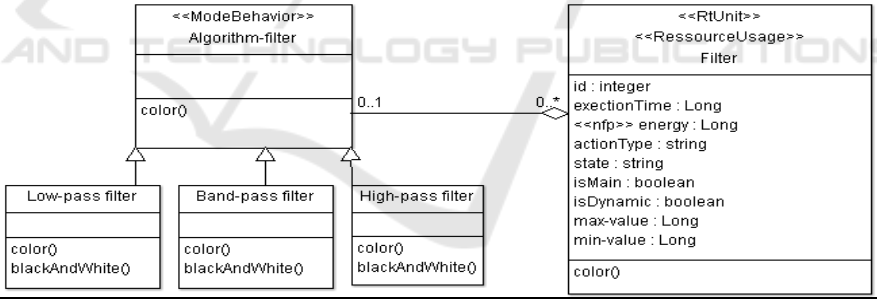
Identifier	B-PDPR:Abstract-behavior
Context	If the task state is dynamic.
Problem	The pattern is applied to a dynamic task in the DPR process, is used when a task has different behaviors. The pattern mainly seeks to separate the main class from its behaviors (algorithms) by encapsulating them into different classes.
Model Solution	This pattern which is dedicated to the DP-RTE system adaptation behavior, deals with the real time proprieties of these systems. A task is stereotyped with MARTE:RtUnit and MARTE:ResourceUsage concepts. It may have one or more behaviors. The class task maintains a reference to abstract behavior. It is configured with a concrete algorithm. A behavior is stereotyped with MARTE:ModeBehavior concept. A non-functional property to be evaluated by this pattern is the consumed energy which is stereotyped with MARTE:NFP concept. The class Abstract-behavior which is the standard interface of all algorithms is used by a task to call a particular algorithm. The classes Behavior-A, Behavior-B are specific algorithms. 
Application example	Filtering strategies are not implemented by the Filter class but by the subclasses of the Algorithm-filter abstract class. The subclasses use different algorithms: Low-pass filter, High-pass filter and Band-pass filter. A Filter task preserves a reference to an Algorithm-filter object. When a filter task is executed, it passes the responsibility to its Algorithm-filter object which specifies the algorithm that must be used. 
Use	A-PDPR:Signal-send, A-PDPR:Function-call and A-PDPR:Data-exchange pattern.

Table 4: "A-PDPR:Function-call" product pattern.

Identifier	A-PDPR:Function-call
Classification	call/signal AND (synchronous OR asynchronous) AND bidirectional AND (point-to-point OR multipoint)
Problem	Compared to the patterns presented by the research studies [(Buschmann et al., 1996), (Schmidt et al., 2000)] which are software-oriented systems design, this pattern is dedicated to the DP-RTE systems adaptation architecture. In these systems, a function call can trigger a DPR process. This pattern is dedicated to the communication between tasks by calling functions. The sender task may or may not wait for the response of the receiver before continuing its execution. However, it must receive a response; hence the sense of communication is necessarily bidirectional. The communication is either synchronous or asynchronous. A component can call one component or several; so the type of communication is either point-to-point or multipoint.

Table 4: "A-PDPR:Function-call" product pattern (cont.).

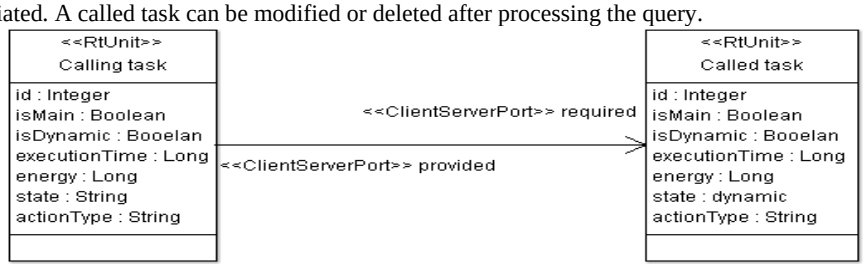
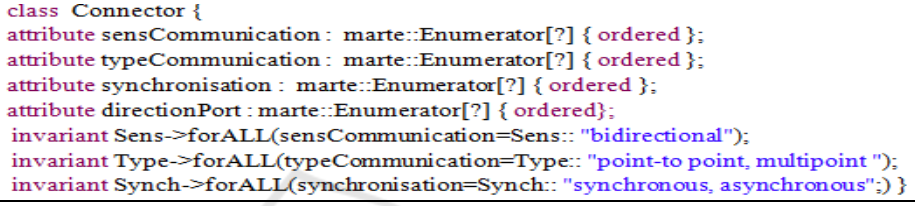
Model Solution	A task is stereotyped with MARTE:RtUnit. The calling task sends a call via a connector through a specific required port which is stereotyped with MARTE:ClientServerPort. This port allows passing a call that is forwarded to the other task that responds via the provided port kind. A calling task can be modified or removed from the configuration after completing the request it initiated. A called task can be modified or deleted after processing the query.  
Use	B-PDPR:Follow-behavior, B-PDPR:Abstract-behavior patterns

Table 5: "A-PDPR:Signal-send" product pattern.

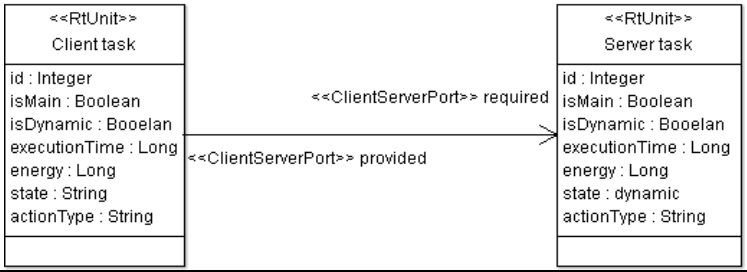
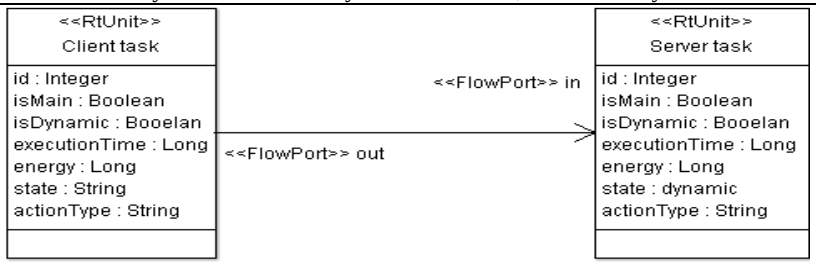
Identifier	A-PDPR: Signal-send
Classification	call/signal AND (synchronous OR asynchronous) AND (bidirectional OR unidirectional) AND (point-to-point OR multipoint OR diffusion)
Problem	This pattern is dedicated to the design of DPR process architecture which is based on sending signals. A signal can trigger a DPR process. The components of this pattern are the client that publishes a signal and the server that consumes it. The type of communication is multipoint, diffusion or point-to-point. The communication between the tasks can be synchronous or asynchronous. The direction of communication is bidirectional, but it can also be unidirectional.
Model Solution	The signal exchange is passed via a connector port. The client issues a signal via the required port kind. The server consumes the signal via the provided port. A task is stereotyped with MARTE:RtUnit and the port with MARTE:ClientServerPort which allows passing a signal. A client task can be modified or removed from the configuration after completing the request it initiated. A server task can be modified or deleted after processing the query. The basis of this pattern is to delete, add or modify a task. 
Use	B-PDPR:Follow-behavior and B-PDPR:Abstract-behavior pattern

Table 6: "A-PDPR:Data-exchange" product pattern.

Identifier	A-PDPR: Data-exchange
Classification	Data AND (synchronous OR asynchronous) AND bidirectional AND(point to point OR multipoint)
Problem	This pattern is used to allow tasks accessing a storage system. Access is managed by a server task. This pattern is similar to the "client/server" architecture except that the client requests a server task for a query. The latter has the client role for the storage system. Communication is processed either in a synchronous or an asynchronous mode, but it is always bidirectional.
Model Solution	 Compared to the patterns presented by the research studies [(Buschmann et al., 1996), (Schmidt et al., 2000)] which are software-oriented systems design, this pattern is dedicated to the DP-RTE systems adaptation architecture. The principle of this pattern is to delete, add or modify a task; it is used when sending data between client and server task which can be modified, deleted or added. Communication is carried out according to a communication protocol via a connector port. The client invokes the server through its out port. The server receives the request through it's IN port. A task is stereotyped with MARTE:RtUnit and the port with MARTE:FlowPort which allows passing the data.
Use	B-PDPR:Follow-behavior and B-PDPR:Abstract-behavior pattern

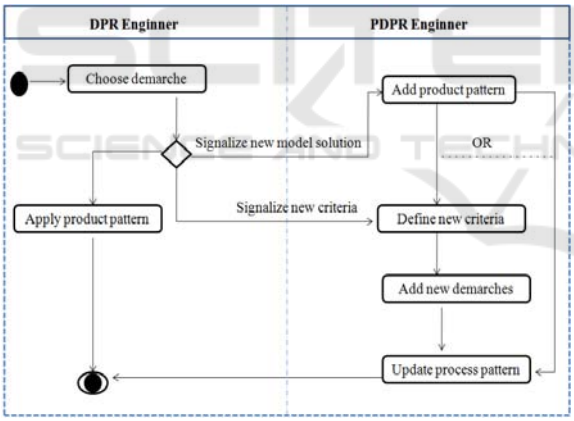


Figure 3: Actors features of PDPR-Tool

4 CASE STUDY

All the engineering systems are based on several methods. C. Rolland (Roland et al., 1988) describes one of them by defining three corresponding components: models, demarches and tools (or techniques). At this stage, we have addressed the first two components, the demarche which is the process patterns of our patterns system and the models that are the product patterns. In this section, we briefly present the last component, namely, the PDPR-tool. Then, we proceed to illustrate the proposed method (PDPR system, PDPR-tool) through a case study of a dynamically adaptive image processing application. The features of the PDPR-Tool are intended to the DPR engineer (reconfiguration engineer) and the PDPR system engineer. The features are illustrated in figure 4.

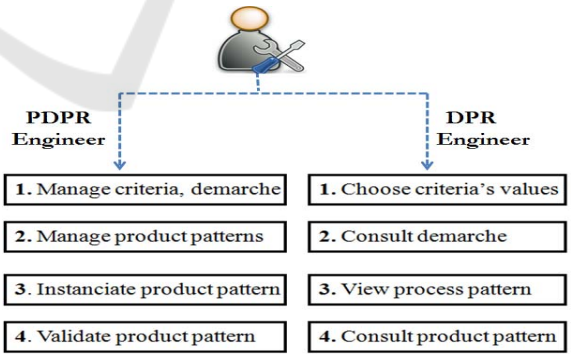


Figure 4: Actors features of PDPR-Tool.

The demonstration is based on a dynamically reconfigurable FPGA and an input/output video. We present the PDPR-Tool through a video streaming consisting of two tasks, executed in sequence. The first task is named Binarization, which is used to binarize the values of an image and produce a binary image. The second task named Inversion is used to

invert the resulting binary image of the binarization task. The example is illustrated in figure 5.

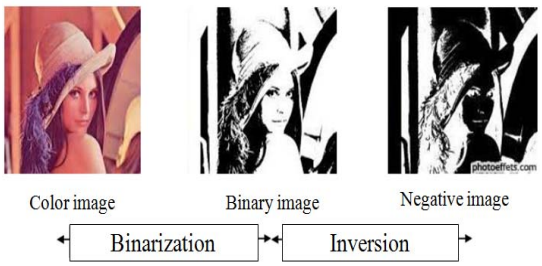


Figure 5: Demonstration Example.

The DPR engineer begins by specifying some values of the criteria according to his needs. The PDPR-Tool automatically combines the selected values of the tests with the values of the unspecified one. Therefore, we obtain the possible demarches that will be saved in a database and displayed on the interface of the PDPR-Tool. The designer, then, chooses the right demarche from the interface. The PDPR-Tool generates the corresponding process pattern (s). The designer (DPR engineer) can view them as an activity diagrams form. The process pattern leads to the product pattern (s). The obtained product patterns for our example are "APDPR: Data-exchange" and "B-PDPR:Follow-behavior". The next step is the instantiation of the model solution.

Figure 6 and figure 7 show the instantiation and the validation of the "APDPR:Data-exchange" pattern : The Inversion task launches the communication and requests bitstreams for its execution. It sends a request via its output port (out) and receives a response via its input port (in). The Binarization task receives the request through its input port (in), handle the request and returns the result to the client task via its out port. The Binarization task is responsible for providing the necessary data that the client task needs. The communication connectors receive requests from the client task via the input port (in) and return the request to the server task via the out port.

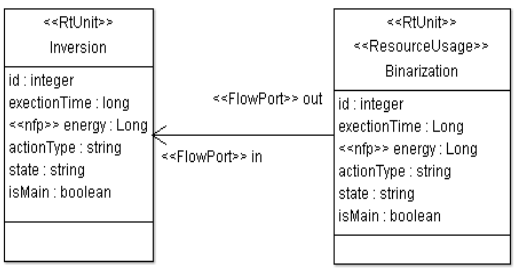


Figure 6: Instanciation of "A-PDPR: Data-exchange" pattern.

Figure 8 and figure 9 show the instantiation and the validation of the "B-PDPR:Follow-behavior" pattern for our example. Only one ModeBehavior is defined and dedicated to the behavior of the Binarization task which is the reconfigurable task (task state criteria = dynamic.), in our example. Transitions that make the passage from one mode to another are stereotyped with the MARTE:ModeTransitions. The events are Reconfigure2S1 and Reconfigure2S2. We have defined two implementations (Threshold 1, Threshold 2) for the Binarization task. Each defined implementation is attached to one mode (S1, S2).

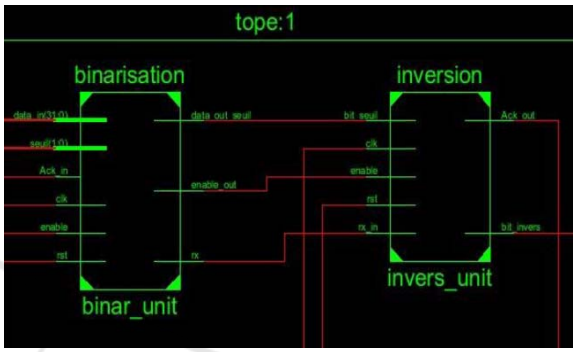


Figure 7: Validation of "A-PDPR: Data-exchange" pattern.

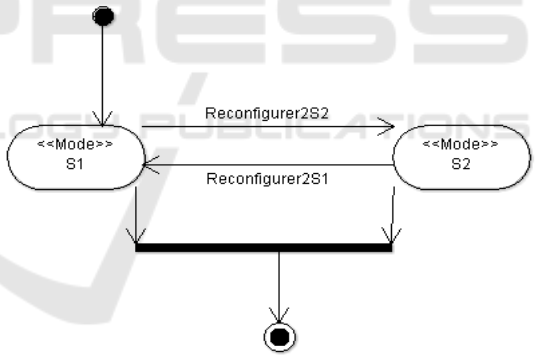


Figure 8: Instanciation of "B-PDPR: Follow-behavior" pattern.

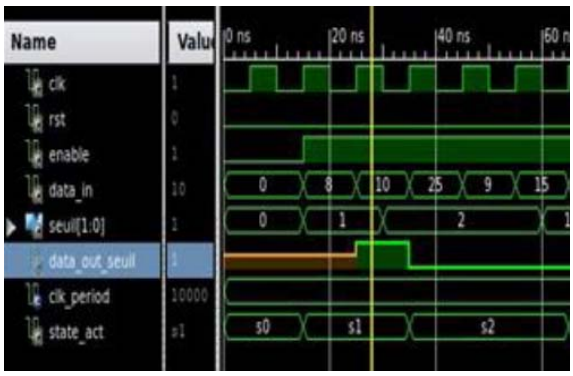


Figure 9: Validation "B-PDPR: Follow-behavior" pattern.

5 CONCLUSIONS AND FUTURE WORK

This paper deals with the high abstraction level design of the DP-RTE systems using a patterns system that allows generating model solutions. Our method focuses on the application part in the DPR design flow. It allows reusing recurrent application models and sharing experience-owned knowledge. We have proposed patterns that limit the vocabulary of the UML/MARTE and identify the essential concepts for the specification of the model solutions. Our method enables the DPR to be processed very early in the design flow, in contrast to the Xilinx flow that only offers it during the phase of placement of the reconfigurable zones on FPGA. Indeed, since the creation of the application model, the concepts of MARTE have enabled to define what the reconfigurable tasks are. Our method was illustrated on a concrete example of an image processing application, starting with the modeling of the patterns system until the instantiation and the implementation of the patterns.

In our future works, we plan to complete and validate the specifications of the patterns system. This refers to the refinement of the classification criteria as well as the proposed product and the process patterns. Then, we intend to integrate the proposed patterns in an MDE-based approach for the automatic generation of DP-RTE systems. Similarly, we seek in our future work, to apply the models @runtime (Thomas et al., 2011) on the adaptation process design of the DP-RTE systems to solve a particular problem related to the complexity and the wealth of the information associated with the execution. This is useful to check the designer's needs and the non-functional properties, support dynamic behavior monitoring and fix the errors during the execution.

REFERENCES

Beux, S. L. (2007). Un flot de conception pour les applications de traitement du signal systematique implementees sur fpga a base d ingenierie dirigee par les modeles. Universite des Sciences et Technologie de Lille.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1996). Pattern- oriented software architecture. Wiley.

Cherif, S. (2013). Approche basee sur les modeles pour la conception des systèmes dynamiquement reconfigurables: de MARTE vers RecoMARTE. University of Science and Technology of Lille.

Cherif, S. Chiraz, T., Samy, M., and Dekeyser, J. (nov 2011). High level design of adaptive distributed controller for partial dynamic reconfiguration in fpga. Conference on Design Architectures for Signal Image Processing: DASIP, pages 308–315.

Chiraz, T. (2012). Controle materiel des systemes partiellement reconfigurables sur fpga: de la modelisation a l implementation.

Corsaro, A., Schmidt, D. C., Klefstad, R., and ORyan, C. (2002). Design pattern for memory-constrained embedded applications. Proceedings of the 9th Conference on Pattern Language of Programs.

Gamatie, A., Beux, S. L., Piel, E., Etien, A., Atitallah, R. B., Marquet, P., and Dekeyse, J.-L. (aout 2008). A model driven design framework for high performance embedded systems. INRIA Journal.

Gamma, E., Helm, R., and Johnson, R. (1995). Design patterns: Elements of reusable object-oriented software. Addison Wesley. .

Group, O. M. (2003). UML 2.0 OCL Specification. OMG Adopted Specification ptc/03-10-14. Object Management Group.

Group, O. O. M. (June 2011). A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, ptc/2011-06-02. Object Management Group.

Henzinger, T. A. and Sifakis, J. (2007). The discipline of embedded systems design. IEEE Society Computer.

Imran Rafiq Quadri, Abdoulaye Gamatie, S. M. J.-L. D. H. Y. E. R. (jan 2010). Targeting reconfigurable fpga based socs using the uml marte profile: from high abstraction levels to code generation. INRIA Journal, pages 308–315.

Marques, N. (2012). Methodologie et architecture adaptative pour le placement efficace de taches materielles de tailles variables sur des partitions reconfigurables.

M. Chess, J. O. K. D. (2003). The vision of autonomic computing. IEEE Computer Society. Ochoa-Ruiz, G., Labbani, O., Bourennane, E.-B., Soulard.

Ochoa-Ruiz G, Ouassila L, El-Bay B, Philippe S and Sana C. (2012) A High-level Methodology for Automatically Generating. Springer Verlag (Germany).

Rolland, C. Foucault, O. and Guillaume, B. (1988). Conception des Systèmes d'Information – la méthode REMORA. Editions Eyrolles.

Said, M. B., Kacem, Y. H., Kerboeuf, M., Amor, N. B., and Abid, M. (July 2014). Design patterns for selfadaptive systems specification. International Journal of Reconfigurable Computing.

Schmidt, D., Stal, M., Rohnert, H., and Buschmann, F. (2000). Pattern-oriented software architecture, patterns for concurrent and networked objects, volume 2. Wiley.

Schmidt, D. C. (2006). Model-driven engineering. IEEE Computer, 39(2). Vogel, T., Seibel, A., and Giese, H. (2011). The role of models and megamodels at runtime. Springer.

Thomas V., Andreas S and Holger G. (2011). The Role of Models and Megamodels at Runtime. Springer.