Modes Generation of Reconfigurable Embedded Systems

Fatma Krichen*
ReDCAD
National Engineering School of Sfax
University of Sfax, Tunisia
fatma.krichen@redcad.org

Mohamed Jmaiel†
ReDCAD
National Engineering School of SfaX
University of Sfax, Tunisia
mohamed.jmaiel@enis.rnu.tn

Fairouz Fakhfakh‡
LAAS
University of Toulouse, France
fairouz.fakhfakh@laas.fr

ABSTRACT

In our work, we propose a process to develop Distributed Real-time Embedded (DRE) systems and particularly reconfigurable ones. Due to the complexity of the development of these systems and their hard temporal and resource constraints, our process offers two strategies of generation: code generation and modes generation. Code generation allows model transformations to generate code while modes generation allows to add all possible modes in our developed middleware from a high level model. In this paper, we detail the modes generation strategy.

1. INTRODUCTION

Embedded systems represent an integration of both software and hardware parts. These systems are frequently real-time and should respond to external events within pre-determined delays. Due to the environment change and the evolution of userrequirements, dynamic reconfigurations of embedded systems are required. They consist in evolving the system from its current configuration to another at runtime by either architectural or behavioral reconfigurations. Architectural reconfigurations consist in modifying the system topology such as adding or removing components or connections while behavioral reconfigurations consist in modifying the system behavior such as updating the properties of components.

Most DRE systems are not fully autonomous and require human intervention to respond triggered events and to be reconfigured. But, human interventions can cause errors and require more time and more efforts. Moreover, it is sometimes impossible to stop a critical real-time system for reconfiguration. Thus, dynamic reconfiguration is required to develop autonomous systems. Constructing autonomous reconfigurable DRE systems requires considerable efforts and is error prone. DRE systems are usually more difficult to design than other types of applications, in particular for reconfigurable ones. It is much tedious and complex to develop these systems without providing a high-level of abstraction. New modeling concepts and development processes are required to design and develop reconfigurable DRE systems.

In our work, we offer an approach allowing the development of reconfigurable DRE systems. For this, we propose a development process which presents two strategies of generation in order to conceive such systems: code generation and modes generation. Code generation presents an MDE-based approach [4] which presents a set of steps to be followed by the developer to get code of reconfigurable DRE systems while modes generation allows to enumerate the set of modes of each MetaMode compliant with the reconfiguration policies specified by the designer and to add all possible modes in our RCES4RTES middleware [6]. In this paper, we present the modes generation strategy.

The remainder of this paper is organized as follows. In Section 2, we describe our developed process to design and develop reconfigurable DRE systems with the two strategy of generation. Section 3 defines the meta-model infrastructure. The modes generation strategy will be described in Section 4. Given this, the Section 5 illustrates the effectiveness of the proposed approach by considering a case study having dynamic reconfiguration requirements: GPS(Global Positioning System). In Section 6, we briefly review related work that addresses the development process of embedded systems. Finally, Section 7 concludes this paper and presents some future work.

2. DEVELOPMENT PROCESS OF RECONFIGURABLE DRE SYSTEMS

We propose a development process allowing to design and develop reconfigurable DRE systems. In our approach, we specify reconfigurable DRE systems with a non-predefined number of configurations. For this, we introduce a new concept *MetaMode* which captures and characterizes a set of configurations (modes) instead of defining each one of them.

*ReDCAD,National Engineering School of Sfax,University of Sfax, Tunisia
Email: fatma.krichen@redcad.org
†ReDCAD,National Engineering School of Sfax,University of Sfax, Tunisia
Email: mohamed.jmaiel@enis.rnu.tn
‡LAAS, University of Toulouse, France
Email: fairouz.fakhfakh@laas.fr

A *MetaMode* is composed of structured components, connectors as well as non-functional and structural constraints. The modes belonging to a *MetaMode* are specified by the set of instances of structured components and connectors defined by this *MetaMode* and satisfying its constraints.

Our approach defines policy based reconfigurations. We specify dynamic reconfigurations using state machines which are composed of a set of *MetaModes* (states) and transitions between them. A *MetaMode* transition presents a set of reconfigurations between modes belonging to these *MetaModes* (as shown in Figure 1). When an event (presented as a *MetaMode* transition) is triggered, reconfigurations (i.e. presented as *mode* transition) are applied on the current mode to one of the modes belonging to the target *MetaMode*. Reconfiguration policies allow to automatically select the target mode. We consider reconfiguration policies: memory, CPU and bandwidth optimization.

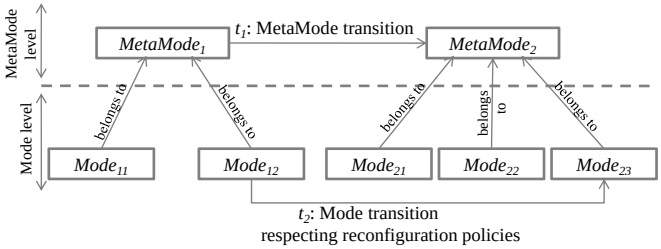

Figure 1: MetaMode modeling

Each *MetaMode* must be allocated to the hardware architecture. As the hardware architecture is unchanged, the allocation is defined from software architecture models (*MetaModes*) to execution supports. The hardware architecture is specified in terms of hardware components (such as processor, memory, etc.) using MARTE profile [8]. Some allocation constraints should be presented in order to specify the allocation policies defining the mapping from software models to hardware instances. The allocation constraints are described using VSL (Value Specification Language) language of MARTE profile.

All previously described concepts allow to model reconfigurable DRE systems. For this, we have defined a new metamodel RCA4RTES (Reconfigurable Component Architecture For Real Time Embedded Systems) [5] which presents these described concepts.

Our development process offers two strategies of generation as described in Figure 2: code generation and Modes generation. The code generation allows model transformations to get code based on our RCES4RTES middleware [6] while modes generation allows to enumerate the set of modes of each MetaMode compliant with the reconfiguration policies specified by the designer and to add all possible modes in our RCES4RTES middleware. The RCES4RTES middleware provides a set of routines in order to perform the dynamic reconfiguration of DRE systems. It supports both architectural and behavioral reconfigurations using a small memory footprint. It also ensures the monitoring and the coherence of reconfigurable systems and preserving the real-time constraints.

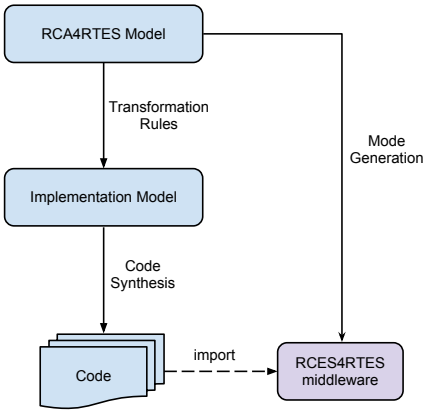

Figure 2: Code and mode generation

In order to generate modes, we have developed an algorithm allowing to select modes which are conform to reconfiguration policies specified in RCA4RTES models. This algorithm allows to directly add the selected modes to RCES4RTES middleware without passing through the implementation model. This paper focus on the modes generation.

3. RCA4RTES MODEL-BASED APPROACH

In the following, we detail our RCA4RTES meta-model presented in Figure 3. Our proposed meta-model allows to describe the dynamic reconfigurations of DRE systems. The *SoftwareSystem* meta-class has a set of *MetaModes* and *MetaMode* transitions. A transition presented by the *MetaModeTransition* meta-class allows switching the system from a *MetaMode* to another when an event is triggered. The *MetaModeChangeEventKind* enumeration presents two kinds of events that can trigger a *MetaMode* transition: an *application event* and an *infrastructure event*. An application event presents a configuration change in accordance with user requirements while an infrastructure event presents a variation of situation in the infrastructure. The proposed reconfiguration policies (e.g, CPU usage, memory usage, and bandwidth usage) are defined as properties of the *SoftwareSystem* meta-class. Each property has a value from the enumeration *reconfigurationPolicyKind* to indicate if the corresponding property will be optimized. For each transition, an activity of reconfiguration is associated. It represents an algorithm for switching from the current configuration (Mode) to the target one. A *MetaMode* transition presents an abstraction of a set of mode transitions.

RCA4RTES meta-model also introduces the *MetaMode* meta class which is composed of a set of structured components, connectors as well as structural and non-functional constraints. We define the *StructuredComponent* meta-class composed of a set of interaction ports. The structured components can be periodic, sporadic or aperiodic threads (*DispatchProtocolKind* enumeration), or a composition of structured components. A connector which is presented by the *Connector* meta-class links two or more interaction ports. It can be a delegation connector (between two output ports or

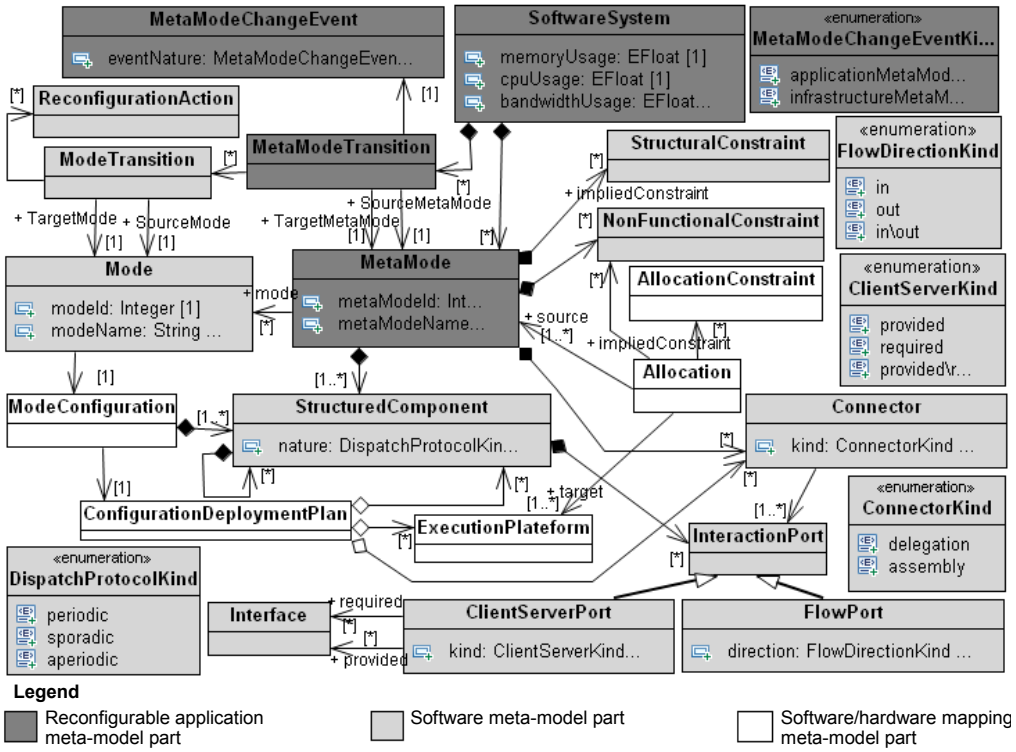

Figure 3: RCA4RTES meta-model

two input ports) or an assembly connector (between an input and output ports). We treat two kind of ports: flow port and client server port. A flow port presented by the *FlowPort* meta-class has been introduced to describe the data flow-oriented communication between components while a client server port which is presented by the *ClientServerPort* meta-class has been added to define a request/reply communication paradigm between components such as operation calls or signals.

Each *MetaMode* has several instances (modes). For each mode, a configuration relates the mode to the deployment plan. A deployment plan describes a configuration by a set of structured components, the connections between them, their configuration, and their allocation to physical nodes.

We introduce the *Allocation* meta-class to specify the allocation of *MetaModes* to execution supports (e.g. the allocation of software models to a static hardware architecture). This allocation has non-functional and allocation constraints that must be respected.

In fact, our meta-model presents three kinds of constraints: structural constraints which are related to the structure of component based architectures, non-functional constraints which specify conditions on the non-functional properties associated with *MetaMode* elements (components, connectors), and allocation constraints which specify the policies used for the allocation of software Models (*MetaModes*) to a static hardware architecture (execution supports).

To handle reconfiguration requirements of DRE systems, a UML profile has been derived from the RCA4RTES meta-model.

4. MODES GENERATION

This section presents the generation of modes of each MetaModes respecting reconfiguration policies from RCA4RTES model. Our approach takes in consideration three kinds of thread:
- Periodic thread is defined by a constant time interval between two successive activations. It is defined by a deadline (D_p), a period (P_p) and an execution requirement (C_p).
- Sporadic thread can be executed at arbitrary times with defined minimum inter-arrival times between two consecutive activations (P_{sp}). It is also identified by a deadline (D_{sp}) and an execution requirement (C_{sp}).
- Aperiodic thread is activated only once and it is characterized by an arrival time and an execution requirement (C_{ap}).

We define algorithms allowing the generation of modes respecting memory usage, CPU usage and bandwidth usage. The modes that will be generated from memory usage, CPU usage and bandwidth usage algorithms together will be added to our RCES4RTES middleware.

4.1 Generation of modes respecting memory usage

To generate modes respecting memory usage, we calculate the used memory rate of each mode. If the used memory rate of mode during system execution is less than the memory rate defined at RCA4RTES model, the corresponding mode

will be generated. We develop an algorithm to calculate and verify if the used memory rate does not exceed the specified rate. The used memory rate is the ratio between the used memory size and the available memory size. For each node, the used memory size is computed based on allocated thread types. For this, three cases are treated to correctly compute the used memory size:
1) Only periodic and/or sporadic threads co-exist: the used memory size presents the sum of memory sizes allocated by all periodic and sporadic threads defined in a DRE system. The memory allocation of periodic and sporadic threads is permanent during system execution.
2) Only aperiodic threads co-exist: we will detect the overlap of time interval of aperiodic threads which requires the highest memory size as shown in Figure 4. Aperiodic threads are activated only once and their memory space is allocated only when they exist. Elementary intervals are created according to arrival and end times of aperiodic threads. Then, we calculate for each interval the memory size allocated by these aperiodic threads. We consider the highest allocated memory size as the used memory size.

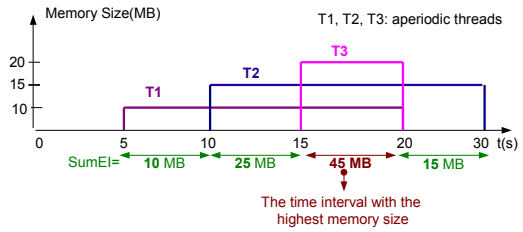

Figure 4: Time intervals of aperiodic threads

3) Periodic and/or sporadic threads co-exist with aperiodic threads: the used memory size is computed by the sum of the two values previously determined in the two first cases (i.e. the sum of the allocated memory sizes of periodic and/or sporadic threads and the highest allocated memory size of the time intervals of aperiodic threads).

4.2 Generation of modes respecting CPU Usage

To generate modes respecting CPU usage, we calculate the processor usage factor of each mode. The modes that the CPU usage factor does not exceed the specified rate will be generated. The processor usage factor is computed with the formula $\sum_{i=0}^{n}(Ci/Pi)$ [7], where n is the number of periodic threads, and Ci and Pi define respectively the execution time and the period of the thread i. Since we also handle sporadic and aperiodic threads, we consider sporadic and aperiodic threads as periodic threads. The two following paragraphs describe how sporadic and aperiodic threads are considered as periodic threads.

To consider sporadic threads as periodic thread, we consider the worst case execution of sporadic threads. This worst case execution is determined when sporadic threads are launched as often as possible (i.e. each two consecutive executions are separated with the minimum of inter-arrival time). That means a sporadic thread is regarded as a periodic thread with C_p=C_{sp}, P_p=P_{sp} and D_p=D_{sp}.

An aperiodic thread can be seen as a periodic thread with a period P at the WCEI. Given this, two cases are handled:
- Only aperiodic threads co-exist: the period P is the end time of the last achieved aperiodic thread.
- Periodic or sporadic threads co-exist with aperiodic threads: the period P is equal to the least multiple of the meta-period (i.e. the LCM (Least Common Multiple) of periodic and sporadic threads periods) and containing the end time of the last achieved aperiodic thread.

4.3 Generation of modes respecting bandwidth usage

To generate modes which respect bandwidth usage, we should verify for each mode that the consumption rate of each bus does not exceed the the rate defined at the RCA4RTES model. For this, we consider that all software connectors will be executed at the same time.

The Listing 1 presents a procedure to compute the bandwidth for a software connections mapped on a bus. For each conection type, we multiply its bandwidth (*BWcnxType*) with the number of possible instances of this connection (*typenbMaxInstances*) mapped on the corresponding bus. As a bus can connect more than two processors, we compute the bandwidth for all connection types between each pair of CPUs.

```
SumBW=0
for each couple of cpu connected by the bus do
   for each software connection type i do
         sumBw = sumBw + (BWcnxType(i)*nbMaxInstances(i))
    endFor
endFor
```

Listing 1: Bandwidth verification of a bus

5. CASE STUDY

To illustrate our MDE-based approach, a GPS (Global Positioning System) [5] case study has been considered. A GPS is a radio navigation system which provides accurate navigation signals to any place on Earth. It helps the user to determine the road to be followed from his current place to some specified destinations using information provided by a satellite. The satellite sends to the ground an encrypted signal which contains various information useful for localization and synchronization. The control base receives and sends information to satellites in order to synchronize the clocks of satellites. We only define the following use cases: (1) GPS with insecure functioning: consists of a traditional use of a GPS; (2) GPS with secure functioning: represents a restricted use of a GPS with some safety requirements.

Following our MDE-based approach and in the first step, we begin by defining a state machine specifying the dynamic reconfigurations by a set of *MetaModes* and *MetaModes* transitions. We use UML state machine diagram (Figure 5). We define two *MetaModes* of GPS: (1) *Insecure GPS MetaMode* and (2) *Secure GPS MetaMode*. The transition from one *MetaMode* to another is ensured by a triggered event. For example, the switch from *Insecure GPS MetaMode* to *Secure GPS MetaMode* occurs when the monitor commands to drive in secure state.

The reconfiguration policies are also defined as tagged values of SoftwareSystem stereotype (Figure 5). For example, the memory consumption should not exceed 40% of hardware memory size. The modes conforming to the specified reconfiguration policies are then generated and added to our middleware.

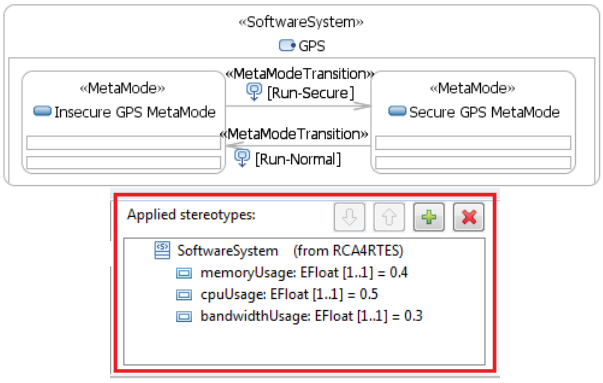

Figure 5: State Machine of GPS

In the second step of our approach, we specify each system *MetaMode* by a set of structured components, connectors and non-functional and structural constraints. For the sake of simplicity, many functionalities of this case study have been omitted. Both satellite and control base are represented by basic components (resp. *GpsSatellite* and *GpsControlBase*). In this paper, we only describe the *GpsTerminal* architecture which consists of five components for the *Insecure GPS MetaMode*: *Position* for receiving the satellite signal, *Receiver* for converting the analog signal into a digital signal, *Decoder* for decoding digital information and separating between the information to calculate distance and time information, *TreatmentUnit* for computing the distance from the satellite in order to obtain the position, and *Encoder* for encoding time and position information.

Then and in the third step, we specify the static hardware architecture followed by the allocation of each *MetaMode* to the specified static hardware architecture. Figure 6 presents the allocation of *Insecure GPS MetaMode* to GPS terminal hardware and GPS satellite hardware. The top part of Figure 6 presents the *Insecure GPS MetaMode* while the down part presents the hardware architecture of GPS terminal and GPS satellite. For lack of space, the GPS control base is not presented. We use the MARTE profile to specify the hardware part. The memory size of both GPS Terminal node and GPS Satellite node is 10 MB. The frequency of each processor is 800 MHz while the bandwidth of each bus is 200 Mb/s. The allocation constraints describe the policies of allocation of models to hardware instances. For example, the allocation of instances of structured component `Encoder` is devised between the two processors `cpu1` and `cpu2` of GPS terminal (as shown in figure 6).

We have developed an ECLIPSE plug-in allowing modes generation. This plug-in generates and selects modes respecting reconfiguration policies from models. For the GPS case study, seven modes conforming to reconfiguration policies are generated and added to RCES4RTES middleware

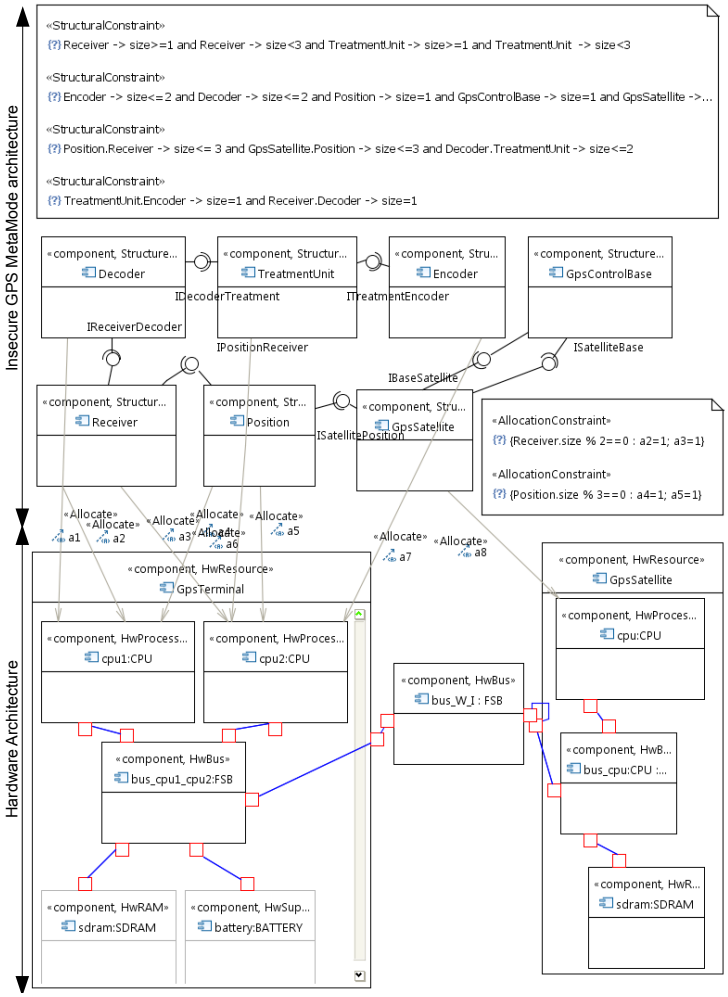

Figure 6: Allocation of Insecure MetaMode to GPS terminal hardware and GPS satellite hardware

(four modes of *Insecure GPS MetaMode* and three modes of of *Secure GPS MetaMode*). Figure 7 shows that the *mode 4* of *Insecure GPS MetaMode* respect reconfiguration policies while *mode 5* does not respect these policies. Therefore, *mode 4* will be added to RCES4RTES middleware while *mode 5* will be eliminated.

6. RELATED WORK

To cope with the growing complexity of embedded system design, several development processes have been proposed. The approaches defining in [9, 1, 2] offer development processes that allow to conceive real-time embedded systems. They present methodologies to be followed by developer from high level models to code. However, these approaches do not support reconfigurable systems. Thus, they define only one configuration of each system.

COMDES (COMponent-based Design of Embedded Software for Distributed Systems)[3] is a framework dedicated to the specification and the configuration of real-time embedded systems. Using this framework, an embedded application is built from reusable components implemented as ex-

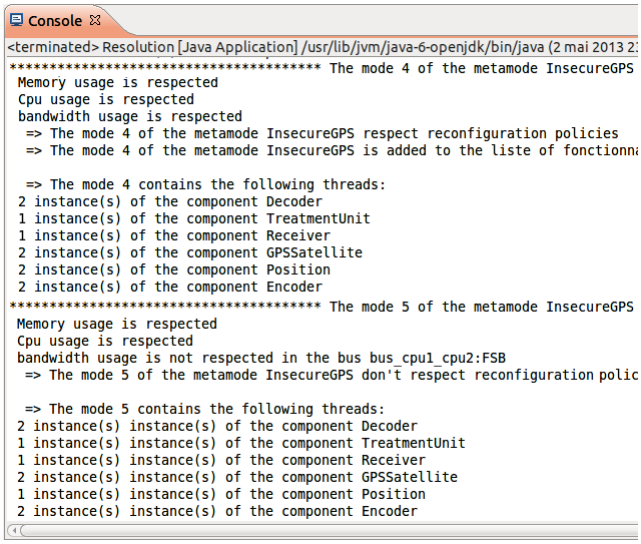

Figure 7: Generation of modes respecting reconfiguration policies of GPS

ecutable function blocks. COMDES defines a development process for embedded systems starting from design level until production of application code. A system is modeled in a high level of abstraction, and then the output model will be transformed to a COMDES model that will be generally enriched with information that guide code generation. Finally, the generated code will be deployed and tested. This framework defines two types of processes: configuration process and reconfiguration process. The configuration process allows to find components in the component repository and then to assemble them to configure an application model. A reconfiguration process allows adding, removing and updating components at runtime in order to update the application. However, using COMDES framework, the developer has a limited number of prefabricated components that are stored in a component directory, then the developer can not add a new component that does not exist in the directory. Moreover, the different system modes will be defined based in the prefabricated component.

The most proposed development processes allow to generate code. But, they do not ensure the generation of modes (configurations) of such system.

7. CONCLUSIONS AND FUTURE WORK

In our work, we proposed a development process to develop distributed real-time embedded systems and particularly reconfigurable ones. For this, we proposed two strategies of generation in order to conceive these systems: code generation and modes generation. Code generation presents an MDE-based approach which presents a set of steps to be followed by the developer to get code of reconfigurable DRE systems while Modes generation allows to enumerate the set of modes of each MetaMode compliant with the reconfiguration policies specified by the designer and to add all possible modes in our RCES4RTES middleware. In this paper, we presented the modes generation strategy. We are developing an ECLIPSE plug-in allowing to generate modes and to add them to our RCES4RTES middleware .

As Future work, we aim at extending our approach in order to enable validation and verification of reconfigurable distributed real-time embedded systems. In particular, we plan to investigate the verification of non-functional properties like CPU and Memory usage. Moreover, our approach should enable to verify that the integration of both hardware and software parts is coherent.

8. REFERENCES

[1] W. E. H. Chehade, A. Radermacher, F. Terrier, B. Selic, and S. Gérard. A model-driven framework for the development of portable real-time embedded systems. In *Proceedings of the 16th IEEE International Conference on Engineering of Complex Computer Systems*, pages 45–54, Las Vegas, Nevada, USA, 2011. IEEE Computer Society.

[2] F. A. M. do Nascimento, M. F. S. Oliveira, and F. R. Wagner. Modes: Embedded systems design methodology and tools based on mde. In *Proceedings of the Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, pages 67–76, Washington, DC, USA, 2007. IEEE Computer Society.

[3] Y. Guo, K. Sierszecki, and C. Angelov. A (re)configuration mechanism for resource-constrained embedded systems. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference*, Washington, DC, USA, 2008. IEEE Computer Society.

[4] F. Krichen, A. Ghorbel, B. Hamid, and B. Zalila. An MDE-Based Approach for Reconfigurable Embedded Systems. In *Proceedings of the 21st IEEE International Conference on Collaboration Technologies and Infrastructures*, pages 78–83, Toulouse, France, juin 2012. IEEE Computer Society.

[5] F. Krichen, B. Hamid, B. Zalila, and M. Jmaiel. Towards a model-based approach for reconfigurable distributed real time embedded systems. In *Proceedings of the 5th European Conference on Software Architecture*. Springer, 2011.

[6] F. Krichen, B. Zalila, M. Jmaiel, and B. Hamid. A middleware for reconfigurable distributed real-time embedded systems. In *Proceedings of the ACIS International Conference on Software Engineering Research, Management and Applications SERA (selected papers)*, Studies in Computational Intelligence, Shanghai, China, 2012. Springer.

[7] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, 20(1):46–61, 1973.

[8] OMG. UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. http://www.omgmarte.org, June 2011.

[9] B. Zalila. *Configuration et déploiement d'applications temps-réel réparties embarquées à l'aide d'un langage de description d'architecture*. PhD thesis, École Nationale Supérieure des Télécommunications, 2008.