An MDE-based approach for reconfigurable DRE systems

Fatma Krichen[1,2], Amal Ghorbel[2], Brahim Hamid[1], and Bechir Zalila[2]
[1]IRIT, University of Toulouse, France
Email: Fatma.Krichen@irit.fr
[2]ReDCAD, University of Sfax, Tunisia

***Abstract*—In this paper, we propose a methodology to design 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, a development process is required to make easier the construction of these systems. For this, we propose an MDE-based approach which offers a development process from model to code.**

I. Introduction

The development of embedded systems consists in designing and integrating both software and hardware parts. Due to the environment change and the evolution of user requirements, the dynamic reconfigurations are required for these systems. The dynamic reconfigurations consist in evolving the system from its current configuration to another at runtime by either architectural or behavioral reconfigurations.

Most DRE systems are not fully autonomous and require the human intervention to respond to triggered events and to be reconfigured. But, human interventions can cause errors and require more time and much efforts. Moreover, it is sometimes impossible to stop a critical real-time system for reconfiguration. Thus, the dynamic reconfiguration is so required to develop autonomous systems. Constructing 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. Facing the exponential evolution of reconfigurable DRE system requirements, developers have very little time to conceive and market their systems. This constraint presents a very important factor to have a competitive advantage. For this reason, developers should conceive a system as fast as possible with guaranteeing the required performance.

To cope with the growing complexity of embedded systems design, several refinement approaches have been proposed. The most popular one is MDE (Model Driven Engineering) [1] which is a standard defined by OMG. Using a common modeling language in MDE, models represent the main artifacts to be constructed and maintained. In the MDE context, software development consists of transforming a model into another one more refined until a final model is reached. This final model is related to a specific platform and it is ready to be executed.

In this paper, we aim to offer an approach allowing the development of reconfigurable DRE systems. For this, we propose an MDE-based approach which presents a set of steps to be followed by the developer to design reconfigurable DRE systems. This approach presents a set of transformation rules allowing model to model (M2M) transformations.

The remainder of this paper is organized as follows. In Section II, we describe our MDE-based approach to design and develop reconfigurable DRE systems. Section III defines the meta-model infrastructure while Section IV presents tools. Given this, Section V illustrates the effectiveness of the proposed approach by considering a case study having dynamic reconfiguration requirements. In Section VI, we briefly review related work that address the development process of embedded systems. Finally, Section VII concludes this paper and presents some future work.

II. Our Proposed MDE-based Approach

We propose an MDE-based approach allowing to design reconfigurable DRE systems. In our approach, we specify reconfigurable DRE systems with a non-predefined number of configurations. For this, we introduce the new concept *MetaMode* which captures and characterizes a set of configurations (modes) instead of defining each of them. The *MetaMode* is described by 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* 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 as reconfiguration policies: memory, CPU and bandwidth optimization.

Then, each *MetaMode* must be allocated to the hardware architecture. As the hardware architecture is unchanged, the

1524-4547/12 $26.00 © 2012 IEEE
DOI 10.1109/WETICE.2012.57

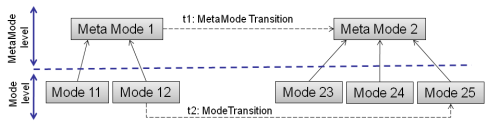

Fig. 1. MetaMode modeling

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 [2]. 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) of MARTE profile [2].

All previously described concepts allow to model reconfigurable DRE systems. To ensure the code generation, these models will be translated to implementation models which are based on RCES4RTES middleware model. The RCES4RTES middleware [3] 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 ensures the monitoring and the coherence and preserves the real-time constraints.

We define an MDE-based approach which presents a development process of reconfigurable DRE systems as shown in Figure 2. This process defines five levels:

- **Reconfigurable application model:** modeling of the dynamic reconfiguration using state machines. Reconfiguration policies should be also specified to select the target mode.
- ***MetaMode* model:** modeling of the system *MetaModes*.
- **Hardware model:** modeling of the fixed hardware architecture in terms of hardware components such as processor.
- **Software/Hardware mapping model:** allocation of system *MetaModes* to the fixed hardware architecture.
- **Implementation model:** generation of models which present the system implementation.

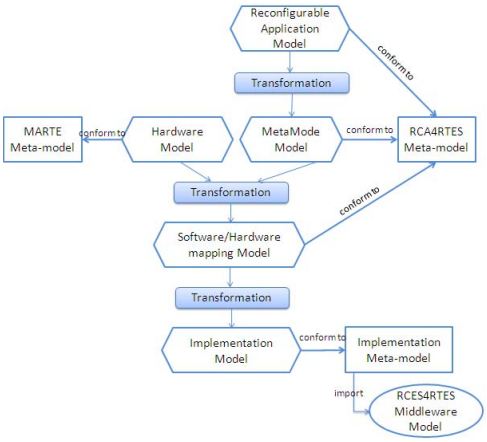

Fig. 2. An MDE approach for reconfigurable DRE systems

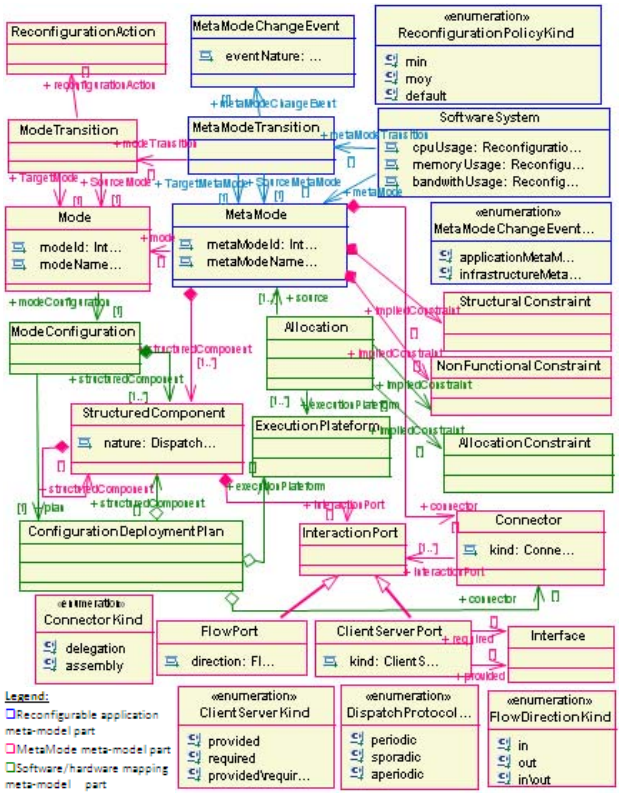

Fig. 3. RCA4RTES meta-model

III. META-MODELING INFRASTRUCTURE

A. RCA4RTES Meta-model

In the following, we detail our RCA4RTES (Reconfigurable Computing Architecture for Real Time Embedded Systems) meta-model presented in Figure 3. This 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 be triggered: an *application event* and an *infrastructure event*. The proposed reconfiguration policies (e.g, CPU, memory 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 mode transition, a set of reconfiguration actions is associated. It represents an algorithm for switching from the current configuration 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 and structural and non-functional constraints. We define the *StructuredComponent* meta-class com-

posed 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 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. 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 topology of component based architectures, non-functional constraints which specify conditions on the non-functional properties associated with *MetaMode* elements and allocation constraints which specify the policies used for the allocation of software Models to a fixed hardware architecture.

B. Implementation Meta-model

The implementation meta-model described in Figure 4 allows the representation of system implementation models. Each distributed system implementation which is conform to our implementation meta-model has a set of processes. For this, we define both *System* and *Process* meta-classes. The system processes communicate to exchange data. The meta-class *Connector* describe the communication between two processes (sender and receiver) through buses described by the *Bus* meta-class. Each bus is characterized by a communication and transport protocols (*CommunicationProtocolKind* and *TransportProtocolKind* enumerations).

A process is composed of a set of threads defined by the *Thread* meta-class. These threads can be periodic, sporadic and aperiodic threads defined respectively by *PeriodicThread*, *SporadicThread* or *AperiodicThread* meta-classes which inherit respectively from *PeriodicTask*, *SporadicTask* and *AperiodicTask* classes of RCES4RTES middleware model. Each thread is characterized by a set of non-functional properties such as priority and has a set of input and output ports whose types are respectively *PortIn* and *PortOut* classes of RCES4RTES middleware model. These ports allow to send and receive data whose type is *GeneratedType* class of RCES4RTES middleware model through a port router (*PortRouter* class of RCES4RTES middleware model). The functional part of each thread will be added by the developer in *threadJob*

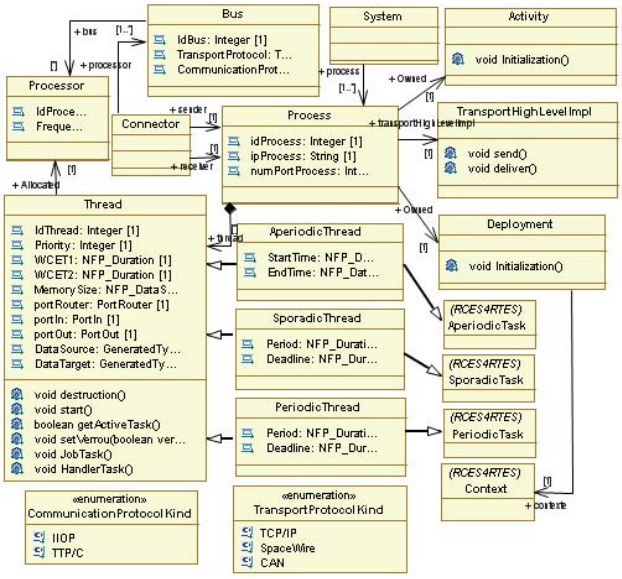

Fig. 4. Implementation meta-model

operation after the generation of code. Each thread is allocated to a processor chosen according to the allocation constraints specified in the previous design level.

To ensure the dynamic reconfiguration of DRE systems, the following meta-classes are defined: (1) *Deployment* meta-class presenting the deployment of the initial mode, (2) *TransportHighLevelImpl* meta-class handling both sending and receiving data for each thread, and (3) *Activity* meta-class allowing the starting of system threads and also the starting of the thread which ensures the dynamic reconfiguration of system (instance of *ReconfigurationTrigger* class of RCES4RTES middleware model).

C. Transformation Engine

The transformation from a model to another is ensured by rules defined using Atlas Transformation Language (ATL) [4]. In our approach, we use two kind of transformations [5]: endogenous transformations in which source and target models are conform to the same meta-model and exogenous transformations with different source and target meta-models.

Both transformation from reconfiguration application model to *MetaMode* model and transformation from *MetaMode* model to software/hardware mapping model are endogenous. These models (reconfiguration application model, *MetaMode* model and software/hardware mapping model) are conform to the RCA4RTES meta-model (as shown in Figure 2). The transformation from software/hardware mapping model to implementation model is exogenous. Software/hardware mapping model is conform to the RCA4RTES meta-model while implementation model is conform to the implementation meta-model (as presented in Figure 2).

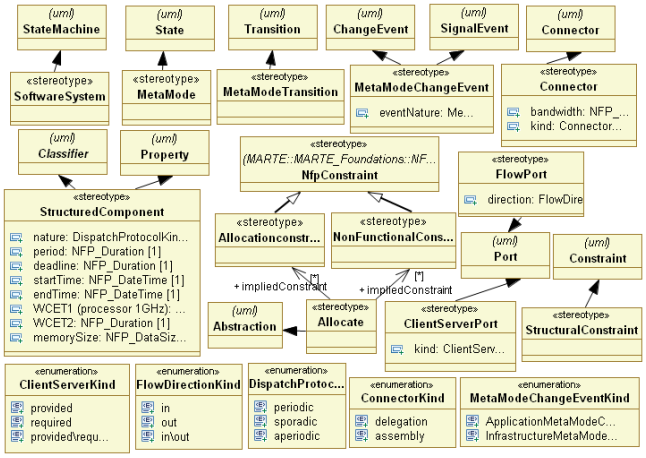

Fig. 5. RCA4RTES profile description

IV. Design and Tools

A. RCA4RTES profile

To handle reconfiguration requirements of DRE systems, we derive a profile from the RCA4RTES meta-model. This profile imports the NFP and VSL profiles of MARTE profile [2] to specify non-functional and allocation constraints and the Basic NFP types of MARTE library [2] to use the types defined in this library. The full profile description is given in Figure 5.

We define a set of stereotypes to specify the dynamic reconfigurations of software DRE systems. A reconfigurable DRE system is described by a state machine. For this reason, we introduce the *SoftwareSystem* stereotype which extends the *StateMachine* UML meta-class and presents the dynamic reconfigurations of DRE systems. These reconfigurations are described by transitions between *MetaModes*. We define the *MetaModeTransition* stereotype, which extends the *Transition* UML meta-class. These transitions are launched by events. The *MetaModeChangeEvent* stereotype is used to define the events that handle the system state machine. *MetaModeChangeEvent* extends both *SignalEvent* and *ChangeEvent* UML meta-classes. An enumeration *MetaModeChangeEventKind* is defined. It presents two kinds of events: application *MetaMode* change and infrastructure *MetaMode* change.

As mentioned previously, a *MetaMode* characterizes the system states by a set of structured components, connectors and structural and non functional constraints. Therefore, the *MetaMode* stereotype extends the *State* UML meta-class.

As we are interested in DRE systems, the structural components are considered as threads or a set of components. For defining and characterizing these threads, we define the following properties as tagged values of *StructuredComponent* stereotype which extends the *Component* UML meta-class:
- **Nature** defines the nature of a component: periodic, sporadic or aperiodic thread,
- **Period** defines the period of a periodic thread. It is used to describe the minimal time between two activations of a sporadic thread. Its type is *NFP_Duration* of MARTE library,
- **Deadline** defines the deadline for periodic and sporadic threads. Its type is *NFP_Duration* of MARTE library,
- **StartTime** defines the start time of an aperiodic thread. Its type is *NFP_DateTime* of MARTE library,
- **EndTime** defines the end time of an aperiodic thread. Its type is *NFP_DateTime* of MARTE library,
- **WCET**: presents the Worst Case Execution Time computed by the sum of *WCET1* and *WCET2* of a thread. *WCET1* defines the worst case execution time on a processor with 1 GHz of frequency while *WCET2* presents the time which does not depend on processor frequency but on other devices such as buses or memories.

Each structured component has a set of interaction ports which can be classified into flow or client server ports. Both *FlowPort* and *ClientServerPort* stereotypes extend the *Port* UML meta-class. To ensure the communication between components, ports are linked by connectors defined by the *Connector* stereotype which extends the *Connector* UML meta-class. Each connector is characterized by the *bandwidth* property which is defined as tagged value and whose type is *NFP_DataTxRate* of MARTE library. To ensure the allocation of *MetaModes* to execution supports, we define the *Allocate* stereotype which extends the *Abstraction* UML meta-class. This stereotype is associated with non-functional and allocation constraints.

The *StructuralConstraint* stereotype extends the *Constraint* UML meta-class in order to specify architectural constraints using OCL. Both *NonFunctionalConstraint* and *AllocationConstraint* stereotypes inherit from the *NfpConstraint* stereotype of NFP package of MARTE profile. This inheritance allows to use VSL which presents an extension of OCL and allows to specify non functional properties and constraints as well as the complex expressions of time.

B. Implementation profile

We propose a UML profile called implementation profile (Figure 6) derived from our proposed implementation meta-model. We define both *System* and *Process* stereotypes which extend the *Package* UML meta-class to present the system as a set of packages. As defined in our implementation meta-model, each process is composed by a set of threads. For this, we define *PeriodicThread*, *SporadicThread* and *AperiodicThread* stereotypes which extend the *Class* UML meta-class. We also define *Deployment*, *TransportHighLevelImpl* and *Activity* stereotypes which extend the *Class* UML meta-class.

V. Case Study

To illustrate our MDE-based approach, a GPS (Global Positioning System) [6] case study has been considered. A GPS is a radio navigation system which provides accurate navigation signals to any place on Earth. 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 satellite clocks. We only define the following use cases: (1)

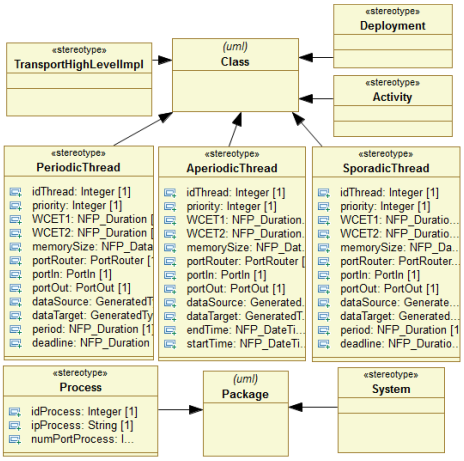

Fig. 6. Implementation profile description

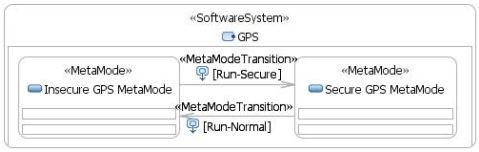

Fig. 7. The State Machine of GPS

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 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 7). We define two *MetaModes*: (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.

In the second step, we specify each *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 *GPS_Terminal* 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,
- *TreatmentUnit* for computing the distance from the satellite in order to obtain the position,
- *Encoder* for encoding time and position information.

Then and in the third step, we specify the fixed hardware

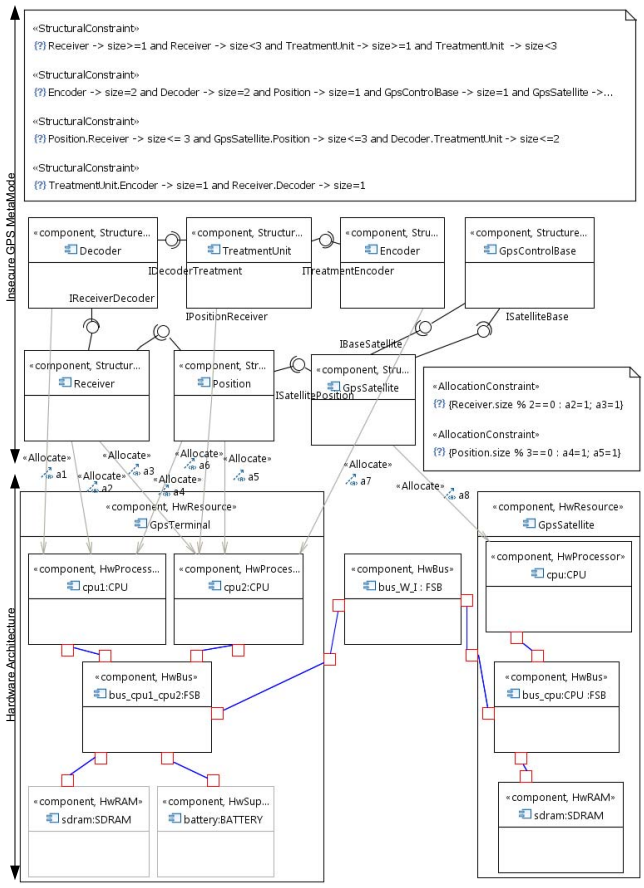

Fig. 8. Allocation of Insecure MetaMode to GPS terminal hardware and GPS satellite hardware

architecture followed by the allocation of each *MetaMode* to the specified fixed hardware architecture. Figure 8 presents the allocation of *Insecure GPS MetaMode* to GPS terminal hardware and GPS satellite hardware. The top part of Figure 8 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 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 8). Finally, the implementation model will be obtained by applying transformation rules in order to generate code.

VI. Related Work

To cope with the growing complexity of embedded system design, several development processes have been proposed. Ocarina [7] is a framework that allows developing, configuring and deploying DRE systems using a model driven approach.

From AADL model, Ocarina can perform scheduling and verification analysis to ensure the validity of the model. Then, an important part of the application code as well as a middleware layer devoted to specific needs of the application will be generated. However, this framework does not address the dynamic reconfiguration in DRE systems.

In the same direction, an MDA approach to address real-time software reusability, maintainability and portability issues is proposed in [8]. In fact, authors propose a Model-Driven Framework which defines a new methodology that makes easier the design and implementation of real-time embedded systems. First, the application model should be annotated with HLAM sub-profile of MARTE profile. The target platform model and the mapping model should be also specified. Then, the generated platform specific model is obtained through defined generic transformation rules. Finally, the executable code will be generated. As an outcome of this process, designers can obtain a real-time embedded system architecture that can be used for several platform implementations. However, this approach does not take into consideration the dynamic reconfiguration of such system.

ModES[9] is also an MDE-based approach devoted to embedded system design. It defines a set of meta-model representing (1) application to capture functionality by means of processes communicating, (2) platform to indicate available hardware/software resources, (3) mappings from application to platform, (4) and implementations oriented to code generation and hardware synthesis. The particularity in this approach is that the mapping meta-model does not specify only the allocation of application processes to fixed hardware components. This mapping also delimits a design space which corresponds to all possible implementations that can be obtained through the choice of sequences of transformations between models. Therefore, the set of transformations between models implements the possible mappings from application to platform. The ModES methodology includes a set of tools that support model-based design tasks starting from specification until software/hardware generation and synthesis. However, this approach does not support the reconfiguration of DRE systems.

The previously presented approaches [7], [8], [9] 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. In the same direction, COMDES (COMponent-based Design of Embedded Software for Distributed Systems)[10] is a framework dedicated to the specification and the configuration of DRE systems. It 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. Compared to our approach, using COMDES framework, the developer has a limited number of prefabricated components that are stored in a component directory. In our approach, we can create new instance of each component while structural constraints are respected. Moreover, this framework does not consider the resource optimisation in the reconfiguration process.

VII. Conclusions and Future Work

In this paper, we proposed an MDE-based approach to develop distributed real-time embedded systems and particularly reconfigurable ones. For this, we defined two meta-models (RCA4RTES and implementation meta-models). The RCA4RTES meta-model allows to specify reconfiguration application model, *MetaMode* model and software/hardware mapping model while the implementation meta-model allows to specify implementation model. The hardware model will be specified using MARTE profile. We are developing an ECLIPSE plug-in allowing the generation of code following the proposed process.

As Future work, we aim at extending our approach in order to enable validation and verification of reconfigurable DRE systems. In particular, we plan to handle the state of components and connectors during reconfiguration at runtime.

References

[1] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[2] OMG, "A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems,Beta 2," http://www.omgmarte.org, June 2009.

[3] 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*. Shanghai, China: Springer, 2012.

[4] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez, "Atl: a qvt-like transformation language," in *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. ACM, 2006, pp. 719–720.

[5] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Syst. J.*, vol. 45, no. 3, pp. 621–645, july 2006.

[6] 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.

[7] J. Hugues, B. Zalila, L. Pautet, and F. Kordon, "From the prototype to the final embedded system using the ocarina aadl tool suite," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 4, 2008.

[8] 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*. Las Vegas, Nevada, USA: IEEE Computer Society, 2011, pp. 45–54.

[9] 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*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 67–76.

[10] 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: IEEE Computer Society, 2008.