Electronic Communications of the EASST
Volume 50 (2011)

Recent Advances in Multi-paradigm Modeling (MPM 2011)

Model Based Engineering for the support of Models of Computation: The Cometa Approach

Papa Issa Diallo, Joel Champeau, Vincent Leilde

12 pages

Guest Editors: Vasco Amaral, Cécile Hardebolle, Hans Vangheluwe, László Lengyel, Peter Bunus
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Model Based Engineering for the support of Models of Computation: The Cometa Approach

Papa Issa Diallo[1], Joel Champeau[2], Vincent Leilde[3]

[1] papa_issa.diallo@ensta-bretagne.fr, [2] joel.champeau@ensta-bretagne.fr, [3] vincent.leilde@ensta-bretagne.fr
ENSTA Bretagne, MBE Research Departement Brest, France

Abstract: The development of Real Time Embedded Systems (RTES) increasingly requires the integration of several parts with different purposes. Consequently, the heterogeneous appearance of such systems creates a need to manage their growing complexity mainly due to the difficulty to interconnect the different parts composing them. Model Based Engineering (MBE) has significantly participated in recent decades to find solutions in terms of methodologies and technical support tailored to the design of RTES. Indeed, several models are used to represent different aspects of the system. However, the interconnection of different modeling paradigms is still a difficult challenge. The handling of such problems requires a clear definition of the execution and interconnection semantics of the different models composing the system. Indeed, the abstraction of the execution semantics of machines (Models of Computation) can highlight properties for the whole systems execution. In this paper, we propose an approach that captures these semantics at the earliest modeling phases with the aim of exhibiting properties that ease the design space exploration and performance analysis of systems. Our approach extends the Modeling and Analysis of Real Time Embedded Systems profile (MARTE) by providing means to express communication semantics of models. We also review existing approaches for defining such execution semantics.

Keywords: RTES, MBE, MoC (Model of Computation), Cometa, MARTE.

1 Introduction

Significant improvements have been made for embedded systems in their ability to integrate new technologies. Indeed, they are increasingly able to integrate elements from different areas of expertise, that are suitable for various treatments. In particular, the System on Chip (SoC) architectures are more complex and heterogeneous due to the integration of several components such as micro controllers, Digital Signal Processing, memories, Field-Programmable Gate Array.
Thus, one current challenge of the Electronic System Level (ESL) community is to provide effective methodologies and techniques based on appropriate formalisms in order to reduce the increasing complexity of systems. To this purpose, techniques such as the exploration of architecture, reuse of IP (Intellectual Properties) and the separation of concerns are favored by the new design approaches.
Elsewhere, the realization of the system on several levels of abstraction reduces the design com-

plexity, and also allows making property analysis and performance analysis at the earliest possible time.
Model-Based Engineering offers solutions to address these issues. For example, the separation of concerns as recommended by the Model-Driven Architecture [OMG03] specification separates the representation of application aspects from platform aspects which also eases the architecture exploration and reuse of application parts.
However, the MDA solution has not succeeded completely to address these issues due to shortcomings regarding the expression of semantics. Indeed, few tools or existing languages specify clearly the semantics of communication and synchronization of the various modules forming the system. These semantic aspects are also called Models of Computation. In fact, having such semantic information is mandatory for the coherent conception of the system, since it brings details on the different execution semantics of the system parts and the communication and synchronization techniques used for their interaction.
In this paper, we propose an approach that aims to offer solutions to capture the semantics of existing MoCs but also to define new semantics. Specifically, we propose the use of models for defining communication and synchronization of system parts. We propose an extension of the MARTE profile (Modeling and Analysis of Real-Time and Embedded Systems)[OMG07a] with focus on the explicit definition of communication and concurrency.
The paper is organized as follows: the second section presents the background of our approach, in Section 3 we present our contribution in two parts. First we talk about solutions in terms of MoC expressiveness, and we discuss the heterogeneous combination of semantics with linkage to other approaches. In Section 4 we present the related works to position our work, and finally we conclude by presenting the perspectives of our work.

2 Background and Motivation

The growing complexity of RTES induces new challenges for their realization. Indeed, the ESL community has to find new solutions to handle the realization complexity of the new generation of RTES that combines various business domains. To this end, raising the abstraction level of current languages and the early validation of properties can be part of the solution.
On the one hand, efforts are currently made by the ESL community in order to raise the level of abstraction of current implementation languages in order to perform property analysis at the earliest possible time and reduce complexity of system design. This approach reduces also the existing gap with the teams in charge of high-level system design. For instance, languages like SystemC [HV07] define a system level language TLM (Transaction Level Modeling) for high-level system design.
On the other hand, there are several new techniques and methodologies developed by the MBE community to tackle issues related to the modeling of the different parts of a system. For instance, Domain-Specific Languages (DSLs) are focused on architectural and specific semantic aspects that help describe how a system (or sub-system) is realized; they allow defining all the needed concepts and how these concepts are associated to match the objective of a part of the system.
Unfortunately, only few solutions address the issue of heterogeneity. Even though the known

semantics are well-tooled currently (data flow systems [LP02], continuous time based systems [Liu98], or discrete event based systems [Mul99]), their combination is still a difficult task, resulting often in emergent behaviors.
One solution according to MBE is to model systems on several levels of abstraction while performing analysis at each level of abstraction. Moreover, several studies, including the project MOPCOM / SOC [Kou09], aim to define several levels of abstraction in the development process, where design and analysis can be performed. Indeed, from a high-level of abstraction, the platform system designers "virtualize" the hardware platform by describing the underlying topology of such architectures, enriched with communication semantics and the level of concurrency.
Indeed, one needs to explain and/or resolve the interconnections of the different modules. To this end, one should focus on the structural and semantic aspects of the whole system. Several systems are based on hierarchical communicating components. These components are often seen as concurrent entities. The way the different components of the system are interconnected is defined using connectors, ports and interfaces.
Many languages can be used with such methodology. For instance, the MARTE profile [OMG07b] allows specifying the concurrency among different components by defining the concept of "RTUnit", the communication among components is described from the use of "RTConnector". SysML (Systems Modeling Language)[OMG10] is a profile that is often combined with MARTE for system modeling. With SysML, systems can be described based on hierarchical blocks (components), defining the application model.
Other approaches, propose Architecture Description Language semantics for the description of architecture with specific semantics.
However, most of the known approaches and languages have shortcomings in regards to the communication and synchronization expressiveness, mostly lacking flexibility to express those concerns. For instance in MARTE, how the communication between modules should be executed is defined implicitly, and does not have the flexibility to express the communication scheme differently. In SysML such information is not present and has to be imported.
Our contribution is motivated by the need to provide additional solutions for the modeling of highly parallel and heterogeneous embedded systems. More precisely, we recall possible solutions for high-level analysis and execution of systems.
In this paper, we propose a metamodel that aims to provide two main solutions. On the one hand, we propose a component-based architecture which is similar to other component-based approaches in the MBE community. On the other hand we present a way to capture and add semantics to those structural elements in order to ease their execution within a given execution environment. Plus, we define possibilities to combine meaningful MoCs and discuss irrelevant MoC interconnections. The second objective of this paper is to show the possibility to transform such semantics to equivalent standard semantics in the literature.

3 Contribution

In this section, we present our contribution by defining the key concepts of the language Cometa, which permits to capture semantics to support heterogeneous multi-MoCs system models. Then,

we argue the possible representation of the adaptation among different Models of Computation. Finally, we present an example application of our approach.
The language Cometa addresses two aspects: The representation of system architecture, and the capture of MoC semantics. Regarding architectural concerns, the Cometa metamodel defines concepts similar to any Architecture Description Language (ADL): components, connectors, ports, interfaces. Those elements are basic building blocks to any system assembly. Figure 1 shows an excerpt of the language defining the main concepts for modeling of the architecture:

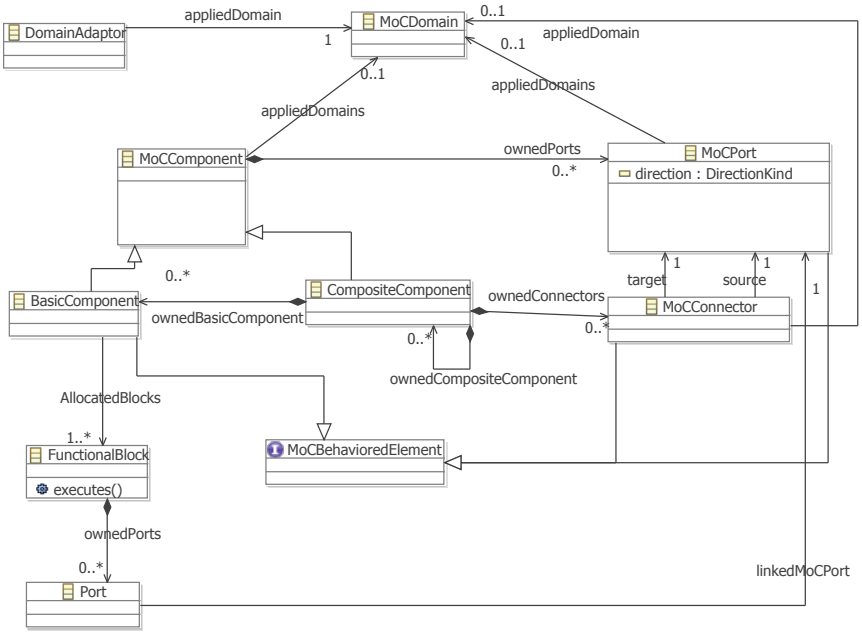

Figure 1: Excerpt of Cometa Metamodel for Architecture Description

- A *MoCComponent*: represents an executable concurrent entity providing / requiring services from the external world through ports and connectors. A MoCComponent can be of different nature (Basic, Composite, and Translator). The Basic Component represents any entity that has a specific behavior; the behaviors defined inside a Basic Component are executed sequentially, while at a same level, different Basic Components can be executed concurrently. The content of a Basic Component can be programmed functions, state machines or any behavior defined with a dedicated language from a domain. Composite Components are elements executed concurrently and highlight architectural concerns. They can help to define structured architectures with several levels of containment. In other words, it can contain other Composite Components, Basic Components, Translator Components, Ports or Connectors. A Translator Component is an element that adapts any semantics from one domain to another domain whenever such translation is possible.
- A *MoCPort*: represents an interaction point of a *MoCComponent*. Depending on the do-

main, the MoCPort can have different meanings (reference, data store, etc.). To offer more flexibility to the definition of Ports, we add the possibility to a MoCPort to eventually have behaviors. Such definition helps to have more control on the ports behaviors. This can be useful if one wishes to define synchronous or asynchronous communications. Ports can also have interfaces that provide or require services. The defined services are strongly related to the domain for which these ports are defined.

- A *MoCConnector*: represents a connection between two concurrent entities. Depending on the domain, the MoCConnector can have different meanings (data path, synchronization point in time, etc.). We also add the possibility to add behaviors to MoCConnectors with the same aim as in the ports to facilitate expressiveness of synchronization.

- A *MoCDomain*: represents a concept that helps to capture the semantics of a specific Model of Computation and helps to link architectural elements to specific semantics specifications. The captured semantics can be bound to the architecture via a MoCDomain. The MoC domain also has the Scheduling policy for a specific MoC, or architectural element (parallel or sequential). Each MoCComponent has an attached MoCDomain.

The defined concepts enable the design of a virtual platform that can support semantics. In our approach we give flexibility to the designers to describe their own semantics, and the possibility to test several semantics in order to find the ones that are efficient to express their needs. Each of the above described concepts relies on a domain or (MoCDomain) which provides a specific semantics. The Semantics of a specific MoC are defined according to the four axes defined by the Rugby conceptual model [Jan04]. In our metamodel those axes are called "schemes". Thus we define:

- A *Behavioral Scheme* : it defines the kind of supported behaviors depending on the domain (Discrete Behavior, Continuous Behavior). Currently, only Discrete Behavior is addressed by the metamodel and is captured with the help of Finite State Machines. For instance, for control concerns, different functions or components can be activated (or fired) by events sending from a defined control element. They can also be used for communication concerns like the representation of synchronization.

- A *Time Scheme*: defines the underlying model of time (causal, discrete, continuous, etc.). For the moment, we just rely on the MARTE model of time which defines discrete time, and continuous time (as a dense discrete time). We abstract in the metamodel the concepts for defining time structures and the concepts for time access (Logical Clock, Chronometric Clock, etc). Currently, more efforts are needed to specify more precisely these concepts.

- A *Data Scheme*: it defines the kind of data manipulated (abstract data types, bit vectors, etc.). Depending on the domain, data can be represented differently. In Cometa we have classified and reified data into four types (abstract data types for the definition of high-level data types such as Integer or String; the physical data types gathering the definition of physical types such as weight, speed, volume; the signals and the bit vectors generally representing wires in a physical platform. We assume that these types can be the bases to

create any types of data. We can also add new concepts to describe more complex data types.).

- A *communicationScheme* (cf. figure 2): it defines concepts for the description of the communication elements such as ports and connectors. The main idea of this scheme is to specialize these elements for a specific domain through the description of their behaviors (according to a behavior described in the behavior scheme), and through the description of the services they require or provide (described from interfaces) and finally through the definition of extra parameters that offer flexibility to match any specific MoC.

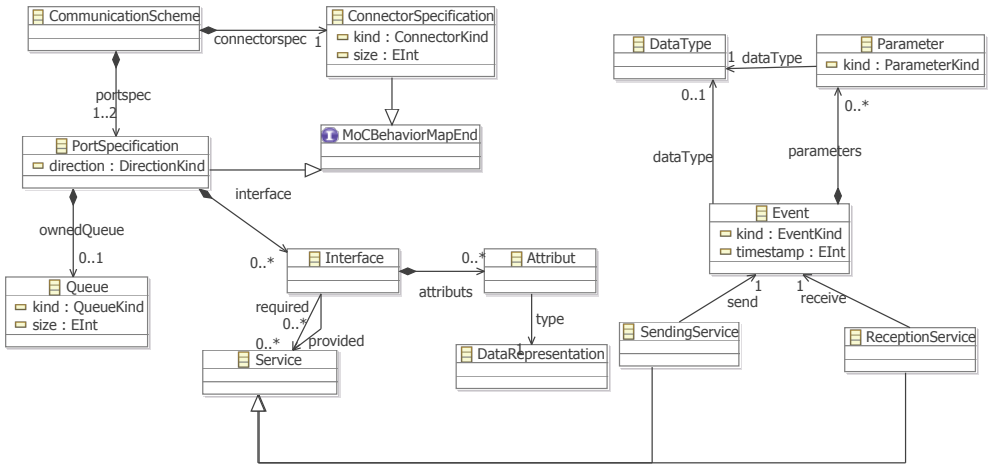

Figure 2: Excerpt of Cometa Metamodel for the Communication Scheme Description

In our approach, several exercises have been successfully done to express known MoCs semantics like KPN (Khan Process Network) or CSP (Communicating Sequential Process) or SDF (Synchronous Data Flow). Furthermore, we allow the user to define their own MoCs. We have described two known MoCs: the CSP [Hoa83] and KPN [Neu04] with Cometa.
In summary, one can describe the architecture of components that are concurrent by definition. It is possible to define two types of architectures: architecture based on hierarchical composition of MoCs but also architecture where the execution semantics are defined in a flat way in the ports and connectors.
However, the contents of each Basic Component are executed sequentially. The rules of communication and synchronization are defined using ports and connectors defining behaviors (FSM) and interfaces.
The composition of semantics is done by the addition of Translators that complement a *source* MoCDomain with missing properties to comply with a *target* MoCDomain. In the case where there is a hierarchy of MoCs, the Translator can tailor the source MoCDomain by adding to the various Schemes the missing information to have a complete semantic description of the *target* MoC. Therefore, to ease the representation of MoC Combination, we rely on the classification

made in Ptolemy that highlights the possible semantics combination between directors into tree degree of compatibility (*loosest*, *loose*, and *tight*)[GBA+07]. The translation must take into account the level of compatibility of the source and target semantics. A first rule being that source semantics must have a less restrictive level of compatibility than the target semantics. With this classification, we define the possible translations.
If one is interested in an adaptation of a flat architecture, the execution semantics applies at the ports and connectors. Thus, the translator modifies the behavior and interfaces of the elements of communication to suit the communication elements of the target MoCDomain.
For example, for the translation of semantics from a set of components communicating asynchronously to a component that is governed by a semantics Synchronous Data Flow, parameters such as the rate of consumption and production are added at the ports and interfaces, then the behavior of the port is modified to behave corresponding to SDF semantics.
In the next section we present an example use of the Cometa principles to express communication aspects. In this example, we consider two applications that send or receive information without any assumption on the nature of communication. If one wishes to have synchronous communication between these two elements, it is possible to capture such semantics with an FSM that we can add to the connector or in the ports. In our example we will define synchronization between the two applications using the CSP semantics. In the CSP MoC, each time a producer emits data, it locks until this data has been consumed by the consumer. At the same time, when a consumer requires data, it locks until this data has been produced by the producer. In this synchronous model, both read and write operations are blocking. Figure 3 shows the state machine for such semantics.

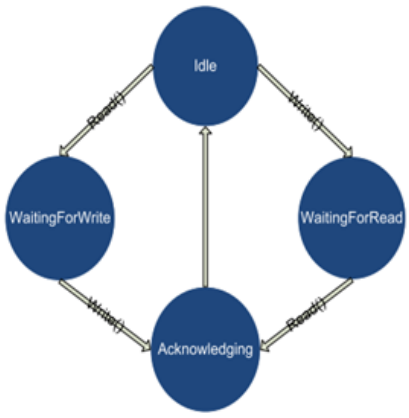

Figure 3: Definition of Synchronous Communication with FSM

The two applications are included in Basic Components linked with ports and connectors that have the interface information and the above defined state machine. The whole description is then transformed to corresponding elements within an execution environment like Rhapsody.

In Rhapsody, figure 4 shows the different exchanges made between the two applications. Application 1 shows a request from application 2 that reaches a ”waitingForWrite” status. Once the data is sent by application 1, the synchronization is made and an acknowledgment is sent to the applications to free the connector for a new synchronization. In this example we can see that specific events have been defined in the ports to communicate with an external element.

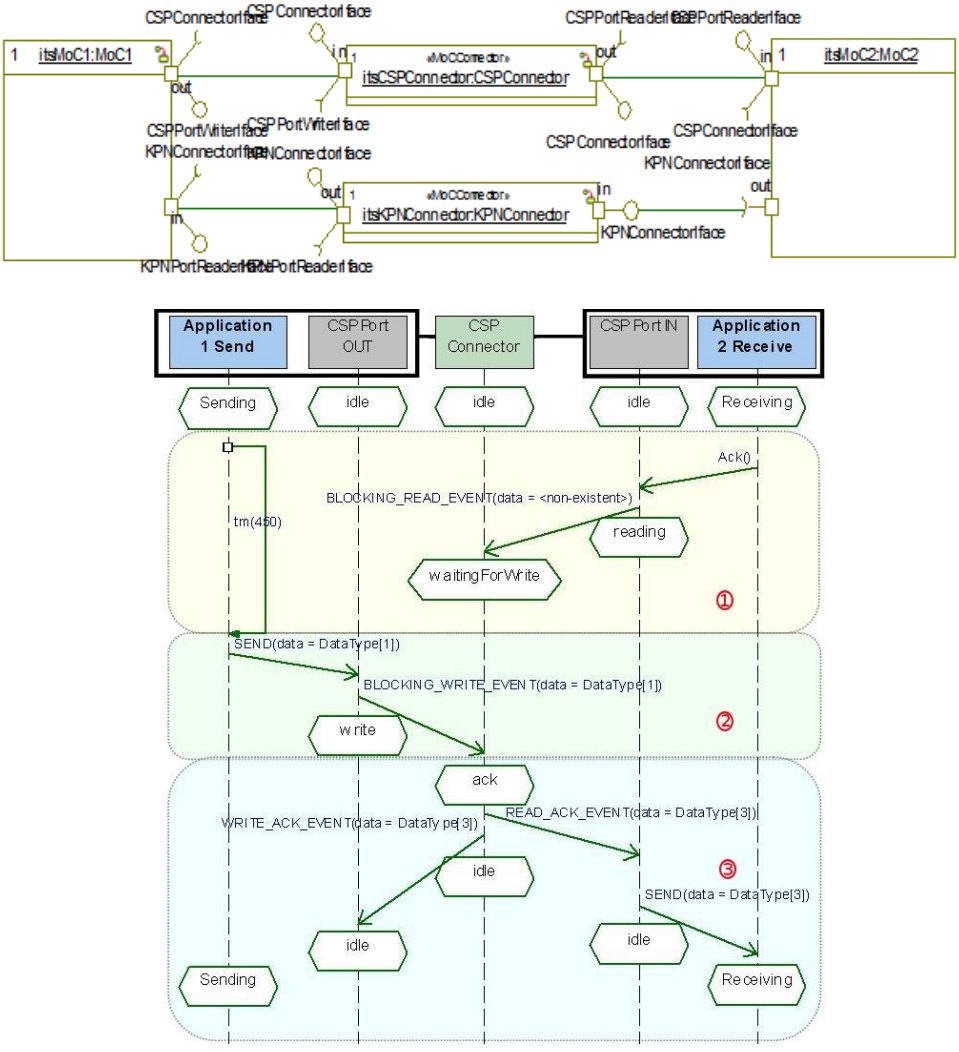

Figure 4: Application model and Simulation Results in Rhapsody

4 Related Works

In section 2, we presented some of the major issues related to MoCs that we try to solve. In the following paragraphs, we present some of the main contributions achieving the same goal. This presentation will give to the readers insights to position our work as well as relevant comparison points for discussion. Indeed the approaches presented in section 2 have several drawbacks. In particular, they do not favor a clear separation of concerns and they do not provide means to cope with heterogeneity of RTES. To this end, several languages have been proposed to tackle issues related to systems heterogeneity. Those languages aim to provide an unified framework for modeling and simulating heterogeneous systems. Among existing approaches aiming to provide a unified framework for heterogeneous specification, the Ptolemy project [EJL+03] is the first to provide a complete component-based framework for heterogeneous systems design and analysis. A design in Ptolemy is made of concurrent atomic or composite entities, called *Actor*, communicating through communication entities called *Port* (Interface). Viewing the system as a structure of actors emphasizes its causal structure as well as its concurrent activities along with their communication and data dependencies. In contrast to OOP, where objects communicate through method calls transferring control to each other's code, actors may or may not be related to the flow of control. In this approach, heterogeneity is handled through specification of Models of Computation (MoC). A MoC is associated with a domain and defines specific *Directors* that establish execution rules for actors (Scheduling and synchronization) of the same hierarchical level. Such an approach favors their reuse in different contexts. Unfortunately, introducing new domains in Ptolemy has limitations as it requires a good knowledge of the underlying language (Java). Moreover, heterogeneity in Ptolemy is handled through hierarchy, which imposes strong architectural constraints. Besides from being a component-based approach, Cometa defines a flat architecture model and emphasizes the use of translator elements to interconnect different MoCs. Also, the ESL (Electronic System Level) community has proposed a language to raise designs at the system level: the SystemC language, standardized by the IEEE (IEEE1666-2005). The goal of this language is to provide a means to build a system incrementally. Each increment is captured by models integrating more details related to specific requirements about computation, communication, data and time. At each level of abstraction, the system is specified through interconnected components and verified by specific kinds of analysis. Such an approach allows a better design space exploration avoiding errors related to early partitioning between software and hardware parts [Arn00]. In SystemC, a system is made of modules (Black-box IPs) and processes communicating through channels. The core of the language provides features required to model heterogeneous systems, e.g., primitive channels, events or time notification. In [Ghe06], authors present an extension of the language called TLM (Transaction Level Modeling) which aims to provide high-level libraries of interfaces and channels to raise levels of abstraction. But according to [HV07], the TLM library does not provide the complete and unambiguous communications semantics required by many MoCs and their integration. To address this issue, they provide then additional facilities to cover syntactical and semantic deficiencies of the SystemC core language through their HetSC (Heterogeneous SystemC) extension. Cometa approach is similar to the TLM approach by providing library of MoC elements to a given platform; in addition it offers the flexibility to define new MoCs. We are also wishing to be able to generate SystemC code in future works. In [BBS06], the authors address the meaningful

composition of heterogeneous components to ensure their correct inter-operation. The spectrum of component types they address is very wide: it goes from fully synchronized components to completely asynchronous components. They propose a framework for modeling heterogeneous real-time components called BIP (Behavior, Interaction and Priority). The Modelica language [Tilel]has been proposed to model and simulate complex and heterogeneous systems. It is an object-oriented non-proprietary language providing special constructs supporting hybrid systems design and simulation. It supports modeling of continuous, discrete or hybrid time-based systems. In this approach, behaviors can be captured by Algebraic differential Equation (ACausal modeling) or Algorithms (Sequence of statements). In [BBHP06], authors suggest an extension of UML 2.0 for hybrid systems modeling. They state, that even if UML 2.0 provides language constructs for a large variety of systems and behaviors,element for hybrid systems modeling have not been considered. For example, real or rational numbers, essential in RTES designs, are not established as primitive types. They provide then a dedicated profile called *HybridUML Profile for UML 2.0*. This proposition aims to fill the lack of UML 2.0 to specify hybrid systems with unambiguous meaning. They also refine the UML Time model in order to model dense-time. The building block for describing systems architecture is introduced under the notion of *Agent* refined into *Primitive Agent* and *Composite Agent*. In [BH08], authors address the simulation of multi-formalism models. They propose a framework called ModHel'X that eases the combination of multiple modeling languages in models. Their approach relies on concepts of component-oriented and hierarchical modeling. In ModHel'X, components are reified under the notion of *Block*. This approach distinguishes three kinds of blocks: the *Atomic Block* (which encapsulates business behaviors), The *Composite Block* (which encapsulates instances of atomic blocks), and the *Interface Block* (which acts as an adapter among heterogeneous blocks). Like for the Ptolemy approach from which it is inspired, the hierarchy is used as a mean to combine the heterogeneous parts of the whole system. Then, the meaningful simulation of such heterogeneous system can be achieved on the providing of a semantic adaptation mechanism as well as an execution engine able to support interpretation of multi-formalism models. Cometa also proposes the use of semantic adaptation mechanism, but our approach is different since we do not define a system based on the hierarchy of MoCs.

5 Conclusion and Future Works

With the rapid evolution of techniques and technologies, the ESL community is becoming increasingly aware of the need to use high level abstractions for the design and analysis of RTES. Unfortunatly, the MBE community has failed to provide means to handle the inherent heterogeneity of the system under study, which hampers its adoption by the ESL commununity. Then, our work contributes to foster the usage of modeling for RTES design and analysis. To this purpose, we have presented in this paper an extension of the new UML for MARTE profile allowing the capture of the heterogeneity that characterizes the actual RTES. Our work contributes to provide reusable libraries of the most commonly used MoC patterns (KPN, CSP, SDF, etc.). Although we have successfully applied our approach onto a real industrial case, our future works aims to generalize our approach on wider range of MoC families (discrete / continuous, timed / untimed) as well as providing a more formal framework with clarification on the combination of

MoCs.

Bibliography

[Arn00] G. Arnout. SystemC standard. In *ASP-DAC '00: Proceedings of the 2000 Asia and South Pacific Design Automation Conference*. Pp. 573–578. ACM, New York, NY, USA, 2000.
doi:http://doi.acm.org/10.1145/368434.368808

[BBHP06] K. Berkenktter, S. Bisanz, U. Hannemann, J. Peleska. The HybridUML profile for UML 2.0. *Int. J. Softw. Tools Technol. Transf.* 8(2):167–176, 2006.
doi:http://dx.doi.org/10.1007/s10009-005-0211-z

[BBS06] A. Basu, M. Bozga, J. Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *SEFM '06: Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*. Pp. 3–12. IEEE Computer Society, Washington, DC, USA, 2006.
doi:http://dx.doi.org/10.1109/SEFM.2006.27

[BH08] F. Boulanger, C. Hardebolle. Simulation of Multi-Formalism Models with ModHel'X. In *ICST '08: Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation*. Pp. 318–327. IEEE Computer Society, Washington, DC, USA, 2008.
doi:http://dx.doi.org/10.1109/ICST.2008.15

[EJL+03] J. Eker, J. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Sachs, Y. Xiong. Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE* 91(1):127–144, January 2003.
http://chess.eecs.berkeley.edu/pubs/488.html

[GBA+07] A. Goderis, C. Brooks, I. Altintas, E. A. Lee, C. Goble. Composing Different Models of Computation in Kepler and Ptolemy II. In *Proceedings of the 7th international conference on Computational Science, Part III: ICCS 2007*. ICCS '07, pp. 182–190. Springer-Verlag, Berlin, Heidelberg, 2007.

[Ghe06] F. Ghenassia. *Transaction-Level Modeling with Systemc: Tlm Concepts and Applications for Embedded Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Hoa83] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM* 26:100–106, January 1983.
doi:http://doi.acm.org/10.1145/357980.358021
http://doi.acm.org/10.1145/357980.358021

[HV07] F. Herrera, E. Villar. A framework for heterogeneous specification and design of electronic embedded systems in SystemC. *ACM Trans. Des. Autom. Electron. Syst.*

12(3):1–31, 2007.
doi:http://doi.acm.org/10.1145/1255456.1255459

[Jan04] A. Jantsch. *Modeling Embedded Systems and SoC's*. Systems on Silicon, 2004.

[Kou09] J. C. D. A. P. S. Koudri, Ali. MoPCoM/MARTE Process Applied To A Cognitive Radio System Design And Analysis. In *Model Driven Architecture - Foundations and Applications*. 2009.

[Liu98] J. Liu. Continuous Time and Mixed-Signal Simulation in Ptolemy II. Technical report, Dept. of EECS, University of California, Berkeley, CA, 1998.

[LP02] E. A. Lee, T. M. Parks. Readings in hardware/software co-design. In De Micheli et al. (eds.). Chapter Dataflow process networks, pp. 59–85. Kluwer Academic Publishers, Norwell, MA, USA, 2002.
http://portal.acm.org/citation.cfm?id=567003.567010

[Mul99] L. Muliadi. DISCRETE EVENT. Technical report, Dept. of EECS, University of California, Berkeley, CA, 1999.

[Neu04] S. Neuendorffer. PN Domain. *Design*, 2004.

[OMG03] OMG. MDA Guide Version 1.0.1. Technical report, Object Management Group, 2003.

[OMG07a] OMG. UML Profile for MARTE, Beta 1. Technical report ptc/07-08-04, Object Management Group, 2007.

[OMG07b] OMG. UML Profile for MARTE, Beta 1. 2007.
http://www.omg.org/omgmarte/Documents/Specifications/08-06-09.pdf

[OMG10] OMG. SysML 1.1. Technical report, OMG, 2010.

[Tilel] M. Tiller. *Introduction to Physical Modeling with Modelica*. Springer, Tiller, Michael.