Enhance the Reusability of Models and Their Behavioral Correctness

Papa Issa Diallo, Joël Champeau, and Loïc Lagadec

Lab-STICC, ENSTA Bretagne, UEB - 2, Rue F. Verny 29806 Brest cedex 9, France
{papa_issa.diallo,joel.champeau,loic.lagadec}
@ensta-bretagne.fr

Abstract. In the context of growing number of design activities, tools and formalisms for system design, it becomes difficult to ensure that an entry system model will retain its consistency and coherence after passing through different tools. Indeed, the design tools often rely on different formalisms, syntaxes and semantics. Therefore, the preservation of the semantics and correctness of models is difficult to ensure. In this article, we are especially interested in the mechanisms to maintain the correct behavior of models through different tools possessing different execution semantics. We propose to show an experiment we conducted to connect the UML Rhapsody specification tool with the Formal Design and simulation ForSyDe-SystemC tool.

Keywords: Model-Driven Engineering, Model of Computation, Tool Interoperability

1 Introduction

Embedded systems design complexity has evolved in the last few decades. Therefore, to guarantee correct functioning of systems and to reduce the validation efforts, such complexity must be addressed at design time and through several abstraction levels, where many paradigms and languages interact. For this particular point, Model Driven Engineering (MDE) offers better abstractions and is gaining credibility in the early design stages as evidenced by the growing willingness to use languages such as UML [1].
Besides, in MDE the analysis of high level system models is another key element for successful development processes. In fact, today's design processes include many analysis tools that means using several analysis paradigms and languages at each design step. In classical processes, the design activities are mainly organized in a vertical flow. Therefore, each step between the process activities can be implemented by transformations or code generations. At each level, to take into account several aspects of the system, we can define a global model and create interactions between languages [1].

[1] For example, to validate a model which has a part dedicated to software properties and the other part for the properties of the hardware component of a platform

For this matter, the *execution semantics* of the models is a major factor to preserve the models behavior towards different analysis steps potentially using different analysis tools. The overall approach requires handling and representing the semantics (static and dynamic) of the models. The dynamic semantics (*execution semantics*) is provided by the execution engine of the analysis tools with regards to the MoC they implement. Such concern is mandatory to ensure that the combination of semantics used to analyze the models is appropriate and preserve the global model's behavior. This is particularly true when the tools have different runtimes based on different execution semantics (e.g. IBM Rational Rhapsody Modeler [2] with Discrete Events [3] semantics, or Modelica [4] with Continuous-Time [5] semantics).
For instance, the presented *Adder Ring* [2] example is representative of the analysis issues encountered when trying to simulate a unique model in environments with distinct execution semantics.
Here, we want to address the synchronization problem between different concurrent components. For instance, in this example, all the modules cannot be launched at the same time without some kind of execution control (or execution priority definition) because the sequence of values expected as output might be erroneous if the modules are launched in parallel with the same priority.
Besides, not using the parallelism proposed by multi-processor platforms ends up with slower systems. Therefore, the idea is to find mechanisms that will permit to take advantage of parallelism while enforcing the system to be conform to its overall behavioral requirements. Such concerns are not well addressed by the currently known MDE approaches.

In this paper, our purpose is to contribute to the reduction of the existing shortcomings within MDE regarding the description and preservation of the models behavior between tools and to reduce the gap between MDE tools and formal design tools. We present our approach to weave an execution semantics (from *ForSyDe-SystemC* tool) to a UML model. The following points summarize our contribution in this article:

- Using the Cometa DSML [6] to express MoC annotations and MoC implementation models: for consistent analysis and simulation in *UML Rhapsody Modeler* [3] [2] as well as for model enrichment to connect with *ForSyDe-SystemC* [4] formal design framework [7]..
- A design flow connecting the *UML Rhapsody Modeler* and the *ForSyDe-SystemC*.

[2] This motivational example is the support of our experiment and is presented in the Section 2. It represents four modules that perform additions successively and are exchanging values.
[3] http://www-03.ibm.com/software/products/us/en/ratirhapfami/
[4] https://forsyde.ict.kth.se/trac

2 Enhance the Reusability of Models and Their Behavioral Correctness

The Design Flow presented in this paper is based on the use of the UML Rhapsody Modeler and the ForSyDe-SystemC Simulator. Each of these methodologies are full-fledged and already responds to several concerns (e.g. specification, simulation, etc). Our goal is to consistently integrate them while preserving the behavior of the models.

Rhapsody [2] is proprietary tool that provides a system development environment based on the use of UML language and profiles. The tool integrates UML component model to specify communicating concurrent entities and its execution semantics is based on events (signals) exchanges (Discrete Events). Unfortunately, its main drawback is the lack of explicitly defined rich MoC semantics, only the DE MoC is implicitly defined.

The **For**mal **Sy**stem **De**sign Framework (ForSyDe) [8] is a modeling and design framework targeting heterogeneous embedded system design. The system models are executable concurrent hierarchical process networks consisting of *processes* interconnected by *signals*. Processes are either *composite processes*, which are created by composing other processes, or *leaf processes*, which are directly created using *process constructors* (e.g. `mealy`, `delay`, `comb`). Each leaf process in the process network belongs to a specific MoC.

With the Adder ring example, our purpose is to show that Rhapsody UML modeler (Specification and Simulation in C++) should be enriched with Cometa MoC-based operational semantics to have a correct execution of the model. Further, if one wishes to perform supplementary analysis using ForSyDe, extra annotation properties (using Cometa) and transformation rules related to MoC definitions in ForSyDe are added to preserve the correctness of the models behavior within ForSyDe.

The Cometa [6] framework proposes way to capture execution semantics (ES) definitions on top of other low level ES such as the Discrete Events of Rhapsody. The framework is a tool independent DMSL aiming to provide formal definition of execution semantics into models by abstracting relevant concepts for the description of formal scheduling mechanisms of concurrent entities based on the theory of MoC to control the parallelism. It is seamlessly integrated with the IBM Rhapsody environment for simulation purpose.

The Adder ring model presents four components interconnected via ports and connectors. The components are named p0, p1, p2, p3; connectors are named fifo_1, fifo_2, fifo_3, fifo_4; besides these elements, we added a delay that enable to break the cycle induced by the ring and to introduce an execution order. In this model, the components and connectors are Cometa models defining behaviors and annotations for the control of the execution.

Here, the idea is to ensure that the requirements mentioned in the introduction are respected while performing analysis in Rhapsody. The Rhapsody execution environment being based on the DE semantics, we provide with Cometa behavioral layers as Event-Based Finite State Machine that adapt the execution of the

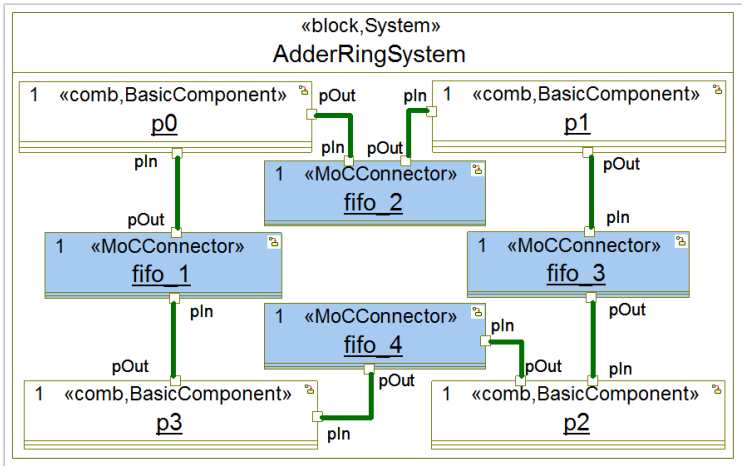

Fig. 1: Use Case connecting 4 Adders in a Ring.

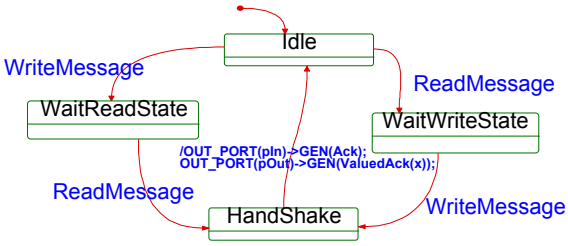

Fig. 2: FSM for the control of the execution implemented in MoCConnector.

concurrent entities to make sure that the different components execute concurrently with respect to their priorities. Here, the behavioral layers are represented by the *MoCConnector*'s (Blue components in Figure 1) and their behavior implementation is provided with the FSM shown in Figure 2.

The FSM has four states: *Idle*, *WaitWriteState*, *WaitReadState*, *HandShake*. The *WaitWriteState*, *WaitReadState* are blocking states that control and alternate Read and Write requests. This means that two requests of the same type (Read-Read or Write-Write) cannot happen which guarantee that each time a component produces one request it blocks until another request responding to the first request get to the HandShake point. Such synchronization is called communication synchronization.

The simulation of the Adder ring model [5], resulting from the enrichment with Cometa, shows that the various concurrent entities follow an execution order which corresponds with the pre-defined requirements. In the next step, we will show that it is possible to obtain the same results by defining a Design Flow con-

[5] see Figure 1 of the annex

```
private rule manageCompositeComponentContent (component : CompositeComponent,
     forsydeProcess : ProcessNetwork)
{
var allcomposites : MDWList = component.getInstances("CompositeComponent");
var allbasics : MDWList = component.getInstances("BasicComponent");
foreach(composite: cometa.CompositeComponent in allcomposites)
    {
      var newProcessNet : forsyde.ProcessNetwork = createProcessNetWork(composite.
           Name);
      @manageCompositeComponentContent(composite, newProcessNet,root);
      root.processNetworks.add(newProcessNet);
    }
foreach(basic : cometa.BasiCComponent in allbasics)
    {
      var newLeafProcess : forsyde.LeafProcess = createLeafProcess(basic.Name);
      @manageBasicComponentContent(basic, newLeafProcess);
      forsydeProcessN.leaf_process.add(newLeafProcess);
    }
    ...
}
```

Listing 1.1: Excerpt of Transformation Rule in MDWorkBench for composite components

necting the ForSyDe-SystemC and Rhapsody UML tool. Here, Rhapsody UML is as well used with Cometa to add annotations (e.g. *comb*, *BasicComponent*) in order to access the MoC semantics constructors of ForSyDe-SystemC. Since ForSyDe processes are built using process constructors, the equivalent concept (i.e. *MoCOrchestrator*) defined within each *BasicComponent* will be used in the semantics layers to represent the different process constructors. The semantics of these different constructors are presented in [8]. In our example, the `combSY` process constructor stands for combinatorial synchronous process.

We set equivalencies between concepts from both DSML's that guided us to the establishment of patterns of model transformation rules to connect ForSyDe and the Rhapsody Modeler. In listing 1.1, we show an excerpt representative of the type of rule used to transform an input UML model to an output model consistent with ForSyDe. In this rule, for a given *CompositeComponent*, all the elements it contains (*BasicComponent*, *CompositeComponent*, *MoCConnector*, etc.) are transformed into their corresponding elements in the target model (*ProcessNetwork*, *LeafProcess*, *Signal*, etc.). Similarly, the *MoCOrchestrator* associated with the Cometa components are parsed. Each *MoCOrchestrator* and its related parameters are reused to build the ForSyDe process constructors. Several rules based on the same approach were developed to translate an entire system model. The resulting model corresponds to a ForSyDe-XML model containing the structure and the functionalities which are automatically transformed into SystemC code by ForSyDe-SystemC compiler. Sample of the generated SystemC code is shown in listing 1.2 The simulation results [6] prove that the sequences of values expected are similar to those displayed after simulation with the Cometa UML Rhapsody model, which ensure the correct execution of the system in the two different environments. Here, we have shown the importance of the behavior adaptations to preserve the correct execution of models towards different tools based on many semantics.

[6] Figure 2 of the annex

```
SC_CTOR(System)
{
    // leaf processes
    auto p0 = ForSyDe::SY::make_comb("p0",p_func,fifo_2,fifo_1);
    auto p1 = ForSyDe::SY::make_comb("p1",p_func,fifo_3,fifo_2);
    auto p2 = ForSyDe::SY::make_comb("p2",p_func,fifo_4,fifo_3);
    auto p3 = ForSyDe::SY::make_comb("p3",p_func,fifo_1,fifo_4);
    auto delay0 =
    ForSyDe::SY::make_delay("delay0",abst_ext<int>(0),fifo_1,fifo_0);
 }
```

Listing 1.2: Excerpt of the generated SystemC Code

3 Conclusion

In this paper, we illustrate an example of models enrichment with concerns related to the formal execution semantics of models. The expected benefits are more abstraction levels for system models (ESL Community), more formal execution semantics satisfying the behavioral requirements and properties of models (MDE Community). It is clear from our experimentation that the models analysis activities must be reinforced with concerns related to the formal execution semantics (MoC) of tools. Consequently, the execution semantics precision and the implementation of execution semantics adjustments (where it is possible) is a considerable contribution to the analysis and preservation of models behavior. This is the purpose of Cometa that has been used here to serve as a semantic link between the Rhapsody modeling tool and the ForSyDe heterogeneous simulation tool.

Acknowledgements The research leading to these results has received funding from The ANR Project GEMOC (French Agency for Research, grant ANR-12-INSE-0011).

References

1. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. 2. edn. Addison-Wesley, Boston, MA (2005)
2. Telelogic: Rhapsody uml modeler. `http://www.telelogic.com/products/rhapsody/index.cfm`
3. Muliadi, L.: Discrete event modeling in Ptolemy II. Master's report, Dept. of EECS, University of California, Berkeley, CA (1999)
4. Tiller, M.: Introduction to Physical Modeling with Modelica. Kluwer Academic Publishers, Boston (2001)
5. Liu, J.: Continuous time and mixed-signal simulation in Ptolemy II. Technical Report UCB/ERL M98/74, Dept. of EECS, University of California, Berkeley, CA (1998)
6. Diallo, P.I., Champeau, J., Leilde, V.: Model based engineering for the support of models of computation: The cometa approach. In: International Workshop on Multi-Paradigm Modeling. MPM '11 (2011)
7. Attarzadeh *et al.*, S.H.: Formal heterogeneous system modeling with SystemC. In: Proc. of FDL. (2012)
8. KTH: Forsyde. Website `https://forsyde.ict.kth.se/trac`.