A General Framework supporting Co-Simulation for BOM and DEVS

Bin Chen, Xiao-gang Qiu, and Ke-di Huang
School of Mechatronics and Automation,
National University of Defense Technology, Changsha,
nudtcb9372@gmail.com

***Abstract*—Domain Specific Modeling brings the problem of simulator differences in simulation. Model Transformation and Co-Simulation are used to solve the problem. Compared to Model Transformation, Co-Simulation integrates the different simulations without the loss of model features. In this paper, we present a general framework supporting Co-Simulation for Base Object Model(**BOM**) and Discrete EVent System specification (**DEVS**). The framework is constituted of **BOM**-based simulation within the **DEVS Proxy** and **DEVS**-based simulation within the **BOM Proxy**. The embedded proxies are used to do the time synchronization and data transfer. The Time Automata is used to represent the models so that the models can be checked and verified by UPPAAL. The **TimeStamped Atomic Model** and General Simulation Data Collect (**GSDC**) algorithm are devised to implement the Co-Simulation framework. The adaptability and validity of the framework are testified by the Aircraft and Control Tower example. The experimental results show that the framework works well in supporting Co-Simulation.**

***Keywords*—Co-Simulation, BOM Proxy, DEVS Proxy, Timed Automata.**

I. Introduction

Domain Specific Modeling is more and more popular in modeling and simulation. Lots of formalisms have been proposed to build the precise model in domains. And the large and complex systems have been asking for the combination of multi-formalism simulations. **Model Transformation** and **Co-Simulation** are the two appropriate methods to resolve the simulator differences. **Model Transformation** syncretizes differences of formalisms in the modeling view. The transformation rules are extracted from both the source and target meta-models[1]. The source models are transformed to the target models. The unified simulating engine can afford the whole simulation.

As known, even the most complete transformation rules cannot fully cover all the models. The generalization is always constructed at the cost of adaptability. So the source models is possible to lose some important features during the generic transformation. As a result we switch to Co-Simulation to find a well-adapted method.

Lots of work has been done on Co-Simulation. [2] presents a formal representation of a continuous/discrete synchronization model. The model is independent of language but the time synchronization is still a sequential one. [3] presents an Co-simulation Backbone on the basis of High Level Architecture (HLA), in which the models have to be wrapped again into federates. [4] presents the M/CD++ system, the system supports the simulation using Modelica. The continuous models in Modelica have to be described in **DEVS** before the discrete event simulation.

We propose a General Framework of Co-Simulation for **BOM** and **DEVS**. The customized proxy represents the source models in the target simulation. We embed the proxies into the simulations without changing the models and the simulations at all. The rest of the paper is organized as follows. Section 2 introduces the **DEVS**-based and **BOM**-based simulation. Section 3 presents the models of **BOM Proxy** and **DEVS Proxy**. Section 4 presents the Co-Simulation framework for **BOM** and **DEVS**. Section 4 gives a case study and shows the experimental results in the framework. Section 5 concludes the paper.

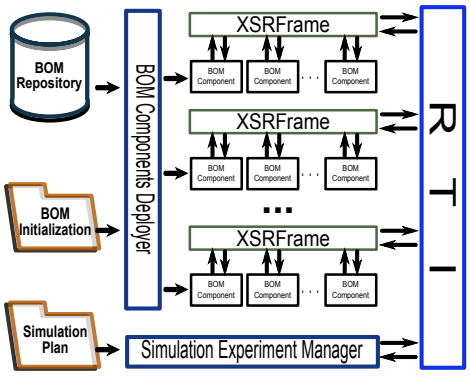

Fig. 1. BOM-based Simulation System.

II. BOM-based and DEVS-based based Simulation

A. BOM-based Simulation

The Base Object Model Template Specification defines a template for representing **BOM** Components. It provides a component framework to facilitate the interoperability, composability and to help enable rapid development of models, simulations and federations. A **BOM** Component is composed of four elements: Model Identification, Conceptual Model Definition, Model Mapping and Object Model Definition, as described detailedly in [5].

A **BOM**-based simulation system has all its simulation models implemented in the **BOM** Component. Figure 1 shows a **BOM**-based simulation framework which consists of **BOM** Repository, **BOM** Components Deployer, Simulation Experiments Manager, XSRFrame and Runtime Infrastructure (RTI)[6].

The **BOM** Repository stores the Atomic and Composed **BOM** Components. These components can be downloaded to the **BOM** Components Deployer which assembles the components and deploys them to the simulation nodes. The XSRFrame loads and manages the life cycle of the **BOM** assemblies, and it is plugged into the RTI to interchange the data during the simulation. The RTI clock is the unique time in the whole simulation system.

The components are initialized when the **BOM** Components Deployer generates the **BOM** assemblies. The behavior of each **BOM** assembly during the simulation is decided in the initialization phase. The Simulation Experiment Manager is responsible for the management of all the **BOM** assemblies

978-0-7695-3941-6/10 $26.00 © 2010 IEEE
DOI 10.1109/ICCMS.2010.225

in the simulation nodes. The Simulation Plan is a collection of the time stamped instructions that control the simulation. The Simulation Experiment Manager controls and observes the simulation according to the instructions in the Simulation Plan.
The simulation experiments can be done automatically with the help of these tools. Users only need to work out the **BOM** Initialization and Simulation Plan based on the simulation objective.

B. The DEVS formalism and DEVS-based Simulation

The **DEVS** formalism was introduced in the late seventies by Zeigler as a rigorous basis for the compositional modeling and simulation of discrete event systems[7]. It has been successfully applied to the design, performance analysis, and implementation of a plethora of complex systems.
DEVS provides a theoretic base of modeling the discrete event models. The formalism specifies the discrete event model in a hierarchical, modular manner. The Atomic Model, AM, is specified as follows:

$$AM = \langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t_a \rangle \quad (1)$$

X is the set of inputs and Y is the set of outputs;
S is the set of sequential states;
$\delta_{int} : S \to S$ is the external state transition function;
$\delta_{ext} : Q \times X \to S$ is the external state transition function;
$\lambda : S \to Y$ is the output function;
$t_a : S \to$
$mathbbR_0^+ \cup \infty$ is the time advance function;
$Q = (s, e)|s \in S, 0 = e = t_a(s)$ is the set of total sets.
Several atomic models could be coupled in the coupled model. The coupled model could also be added into a larger coupled model according to the closure of it. The hierarchy is constructed by building coupled models. The Coupled Model, CM is defined as follows:

$$CM = \langle X, Y, D, M_d, I_d, Z_{i,d}, Select \rangle \quad (2)$$

X: a set of input events and Y: a set of output events;
D: a set of component references;
M_d: a Classic **DEVS** model;
I_d: a set of influences of d;
For each i in I_d
$Z_{i,d} : Y_i \to X_d$ to d output translation function;
$Select$ is the subsets of $D \to D$: tie-breaking function.
The semantics for a coupled model is, informally, the parallel composition of all the sub-models. A priori, each sub-model in a coupled model is assumed to be an independent process, concurrent to the rest. There is no explicit method of synchronization between processes. Blocking does not occur except if it is explicitly modeled by the output function of a sender, and the external transition function of a receiver. There is however a *serialization* whenever there are multiple sub-models that have an internal transition scheduled to be performed at the same time. The modeler controls which of the conflicting sub-models undergoes its transition first by means of the *select* function.
The protocols of simulator and coordinator are devised to support the simulation for **DEVS** models. The typical sequential **DEVS**-based simulation is constructed in Root Coupled model that couples all the atomic and coupled models. The life circle of the simulation depends on the termination checking in Root Coupled model. But the sequential simulation cannot satisfy the requirements of large scale simulation. Thus we implement the parallel techniques on **DEVS** to improve the simulation efficiency. **DEVS** Simulator is modified to implement Time Warp algorithm, the newly simulator supports State Storage, Fault Detection and Fossil Collection. The models are loaded in local simulators in every node. The events are sent and received between simulators on different nodes with the help of parallel communication mechanism. The independent model advances arbitrarily until meets the Global Virtual Time (**GVT**). Simulation is rolled back if a straggler event or an anti-event is received. The approximate Time window is used to improve simulation when the rollback happens very often. As a result, in Co-Simulation, we select the parallelized **DEVS**-based Simulation in discrete event side.

III. The Models of Proxy for Co-Simulation

A. Time Synchronization

BOM-based simulation is usually used to simulate continuous model(i.e. dynamics of vehicles, climate change). And the simulation time advances by steps. **DEVS** is inherent used to simulate the discrete event models. In order to obtain the high performance, we parallelize the **DEVS**-based simulation by partitioning coupled models onto different machines. As a result, the **BOM**-based and **DEVS**-based Co-Simulation is a typical case of Discrete and Continuous Co-Simulation. Gheorghe et al. had introduced the time synchronization interfaces for the Co-Simulation[8]. However the sequential algorithm does not support Parallel simulation. Thus we give a novel method to parallelly synchronize simulation time in **DEVS**-based and **BOM**-based simulation. The method devises the **BOM Proxy** and **DEVS Proxy**. The proxies are responsible for the data transfer and time synchronization between two simulations. The principles of the proxies are list as below:

- The **DEVS Proxy** is seemed as a Local Process(LP) in **DEVS**-based simulation. The current time in **DEVS Proxy** is the Local Virtual Time(**LVT**) of this LP.
- **BOM**-based simulation uses conservative time synchronization algorithm. The **BOM Proxy** is a constrained and control federate, in consequent the time in proxy is consistent to all the models in **BOM**-based simulation.
- The time from **BOM Proxy** is kept less than LVTs of all the coupled models. The frequency of Time Synchronization is determined by the time step of **BOM Proxy**. **GVT** is calculated just before **BOM Proxy**'s advancing.
- The lookahead of **BOM**-based simulation is identical to the Time Window of **DEVS**-based simulation. The timestamps of messages and events are restricted in a limited range.
- **BOM Proxy** transforms messages into two types of events: Regular Event and Rollback Event, while **DEVS Proxy** transforms events into time stamped messages.
- The data transition is needed to match the data formats from messages to events, or vice versa.

Figure 2 describes the time synchronization for Co-Simulation. Coupled **DEVS** Models are distributed on different machines. A Time Warp algorithm is realized here for time synchronization. The time stamps of activities are limited inside the Time Window from **GVT**. The synchronization happens periodically, the **LVT** from **BOM** simulation is used to calculate the **GVT**. Messages are divided into Regular Event and Rollback

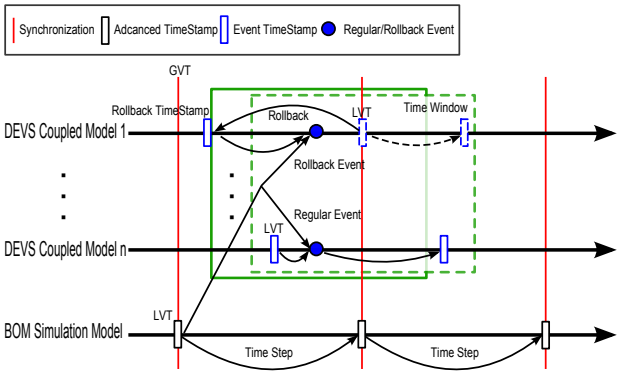

Fig. 2. Time Synchronization for **DEVS** and **BOM** Co-Simulation.

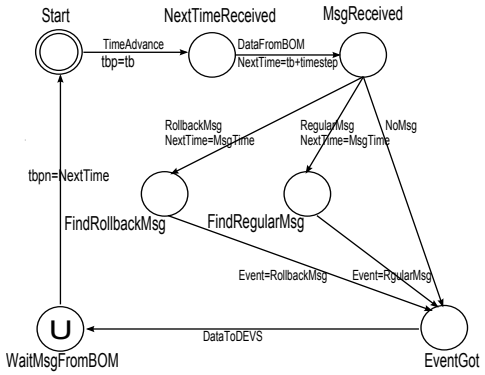

Fig. 3. **BOM Proxy** Model.

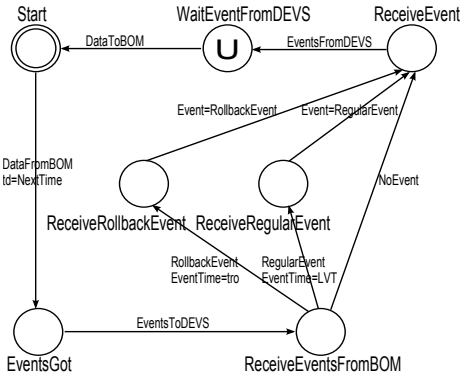

Fig. 4. **DEVS** Proxy Model.

Event because the different **LVT**s in coupled models. The Regular Event only changes the next external time stamp, but the Rollback Event triggers off the rollback. The current states are canceled and the model is rolled back to the last state with the time stamp less than the Rollback Event. Consequently, the **LVT** is recalculated and it is worth to note that the **GVT** guarantees that the rollback in **DEVS**-based simulation will not lead to the rollback of **BOM**-based simulation.

B. Timed Automata

Timed Automata is a formalism for modeling and verification of the real time systems. It is well-adapted to high-level models for the design of software specification. A variety of formal methods have been developed to prove that a Timed Automaton satisfies basic correctness properties and timing properties. The verification covers not only continuous but discrete event systems[9].

A timed automaton can be considered as a classical finite state automata with clock variables and logical formulas on the clock. The behavior of the automaton is restricted by the constraints on the clock. The clock constraints are the guards and transitions, and the actions are used for synchronization. Bengtsson et al. give a simple example of a timed automaton[10]. A Guard is the condition decided by the limited ranges of clocks. The transitions are triggered when the guard is satisfied. The actions are executed during the transitions and after the transition the clocks are reset.

The models of **BOM Proxy** and **DEVS Proxy** are designed using **Timed Automata**. The proxies are responsible for the time synchronization and data transfer between the two simulations. The Safety properties, the Liveness and the Reachability properties of proxy models are checked by UPPAAL. UPPAAL is an integrated tool environment for modeling, simulation and verification of **Timed Automata** [11].

C. BOM Proxy Model

BOM Proxy is embedded in **DEVS**-based simulation. It works on data transfer and synchronization with **BOM**-based simulation. The behavior of **BOM Proxy** is described as below:

1) Detect the time advance in **BOM**-based simulation.
2) Reflect the simulation time of **BOM**-based simulation. The time is used to send to **DEVS Proxy**.
3) Identify and groups the messages from **BOM**-based simulation. Messages are transformed to events in target format.
4) Send the translated events to **DEVS Proxy**.
5) Receive messages from **DEVS Proxy** and sends them to **BOM**-based simulation.

Figure 3 illustrates the **BOM Proxy** model represented in **Timed Automata**. The transition from location ***Start*** to ***NextTimeReceived*** is triggered by the **TimeAdvance**. The synchronization action helps ensure time t_{bp} in **BOM Proxy** consistent with t_b, the current time of **BOM**-based simulation. We initially set t_b to be 0 at the starting point. The location is changed to ***MsgReceived*** by the transition with action **DataFromBOM**. The **NextTime** is increased by the t_b plus time step brought by the advance action.

The location will be changed to ***EventGot***, with the **NextTime** is set to be the translated event time. The middle locations ***FindRollbackMsg*** and ***FindRegularMsg*** are referred to on the condition that the transition is triggered by receiving **RollbackMsg** or **RegularMsg**. The transition with **DataToDEVS** synchronizes the time between **BOM Proxy** and **DEVS**-based simulation. The last change from ***WaitMsgFromBOM*** to ***Start*** reset the **BOM proxy** to restart another cycle.

D. DEVS Proxy Model

DEVS Proxy is similar to **BOM Proxy**, it is used to do the time synchronization and data transfer in **BOM**-based simulation for **DEVS**-based simulation. The behavior is summarized as below:

1) Receives time stamp from **BOM Proxy** and sets it to be the **LVT** for **DEVS Proxy**.
2) Receives events from **BOM Proxy** and sends them to designated **DEVS** models via ports connections.

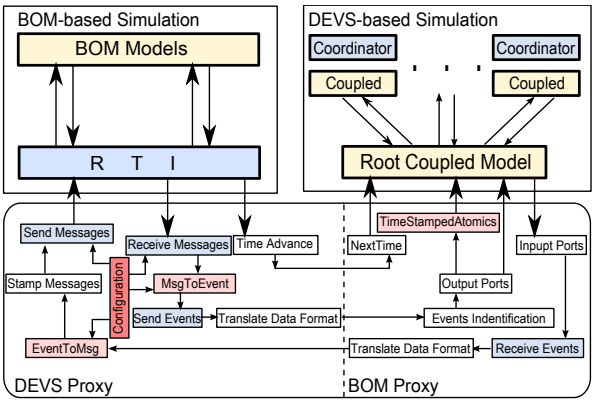

Fig. 5. Co-Simulation Framework for **DEVS** and **BOM**.

3) Receives the events from **DEVS** models and transforms them to messages with time stamp.
4) Sends the messages to **BOM Proxy**.

Figure 4 illustrates the model of **DEVS Proxy** with Timed Atutomata. The initial location ***Start*** changes to ***EventsGot*** by transition with **DataFromBOM** and reset t_d by **NextTime** from **BOM Proxy**. The clock t_d is seemed as the **LVT** of **DEVS Proxy**. The location changes to ***ReceiveEventsFromBOM*** following the transition with action **EventsToDEVS**. There exists 3 possible sub-locations after ***ReceiveEventsFromBOM***. The location movement depends on the receiving events. The **RollbackEvent** leads to the ***ReceiveRollbackEvent*** while the **RegularEvent** to the ***ReceiveRegularEvent***. The locations are all changed to ***ReceiveEvent***. **DEVS** models are synchronized by resetting the event time. The location moves to ***WaitEventFromDEVS*** by the transition of action to receive events from **DEVS** models. Finally, the location changes to ***Start*** and the cycle restarts.
The next section details the implementation of the proxy models verified by UPPAAL.

IV. Co-Simulation Framework for BOM and DEVS

We give the simulation framework of **BOM** and **DEVS** in Figure 5. The **DEVS Proxy** and **BOM Proxy** are actually combined to do the Co-Simulation. **DEVS Proxy** is embedded into RTI, the global simulation in **BOM**-based simulation, to join the Federation as shown. Messages are sent from RTI and received by **DEVS Proxy**, **Time Advance** is also controlled by the **Time Management** in RTI. The transformation from **DEVS** events to **BOM** messages is decided by the **Configuration** which gives the mapping instructions.
BOM Proxy is connected with Root Model in **DEVS**-based simulation, thus the events can be sent to designated **DEVS** models. The time interface and Event transfer interface are used to synchronize time and exchange events with **DEVS Proxy**. **Events Identification** groups events into Regular Events and Rollback Events. We devise **TimeStammped Atomic Models** to model the time delay. These models are supplied to the events translated from time stamped messages. **BOM Proxy** is a sub component in Root Coupled model, the proxy **LVT** is added into the calculation of the **GVT**. So that the time is synchronized with **DEVS Proxy**. By this approach, the time are consistent to the **BOM**-based simulation.
The **GSDC** algorithm and **TimeStamped Atomic Model** are described in detail in the next section.

A. GSDC Algorithm in Data Distribution

The objective of **GSDC** algorithm is to construct the mapping from events to messages by **Configuration**. The events come from **DEVS** models and messages are defined in FOM (Federation Object Model). On the basis of mapping, The **DEVS Proxy** declares the publish and subscribe relationship in RTI. **GSDC** algorithm falls into 4 steps:

1) **Configuration**. Select a message in FOM for each possible **DEVS** event that will be sent to **BOM** components. Likewise, map every message to **DEVS** model to an event. These mappings are maintained in **Configuration** settings.
2) **Subscribe** and **Publish**. Declare the Subscribe and Publish relationship for receiving and sending messages.
3) Receive messages and **Parse** data. Receive messages from RTI and parse the data according to the description of format in **Configuration**. The parsed data is transformed to event.
4) **Pack** data and Send messages. The data translated from event is packed in the format needed by **BOM** Component, then the data is sent to RTI.

Time Management and **Data Distribution** are implemented in a flexible manner because of the configuration step. The publish and subscribe declaration in RTI changes with the **Configuration**. The parse and pack match the data format in different models. It is very convenient with **GSDC** algorithm to make **DEVS Proxy** communicate with the **BOM** Components.

B. TimeStamped Atomic Model

There are two kinds of messages in **BOM**-based simulation: Receive Order(RO) messages and Time Stamp Order(TSO) messages. In the RO case, messages are sent directly to a target just when they are generated. The delivery has nothing to do with the simulation time. Hence, the port-to-port event transfer mechanism of **DEVS** satisfy the RO message delivery. In the TSO case however, it is not efficient to just re-use the **DEVS** mechanism. The time stamp in TSO messages will be lost in this case. To solve the problem, a **Timestamped Atomic Model** used to model network delay is proposed here. We use it as a middle ware to retain the ordering of TSO messages.
The principle of a **Timestamped Atomic Model** is illustrated in Figure 6. The input port is connected to the model port outputting time stamped events. The *extTransition* puts the events into the Timestamped Event List. The list is sorted by the time stamps of arriving events. The *timeadvance* running after the *extTransition* retrieves the minimal time stamp t_m from Event List to calculate the δ. When the **Timestamped Atomic Model** advances to t_m, the internal transition is activated to pass the events stamped by t_m to *outputFunc*. Then, these events are popped to the output port in *outputFunc*. The construction of **Timestamped Atomic Model**s breaks down into two steps: Search time stamped ports and Construct **Timestamped Atomic Model**s. Time stamped ports are the ports used to input or output time stamped events. All the Atomic-CKs are traversed to collect the time stamped ports. **Timestamped Atomic Model**s are constructed to model the necessary time delay for behavior equivalence. They are inserted into every pair of output and input time stamped ports. The output port of a user model is connected to the input

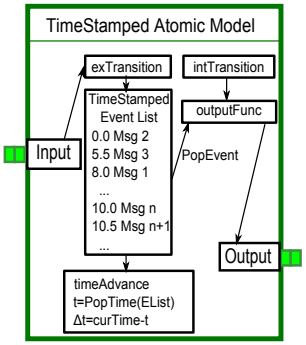

Fig. 6. The Principle of **Timestamped Atomic Model**.

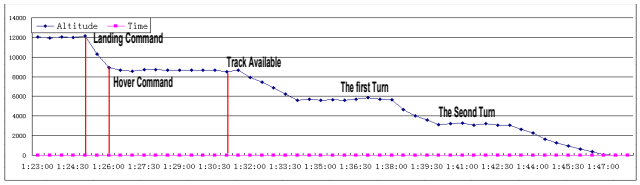

Fig. 7. The Altitude of Aircraft in Landing.

port of a **Timestamped Atomic Model**, and vice versa. These models help store the TSO events and send them to the target at related time stamp times. Consequently, the data distribution in **BOM**-based simulation can be replaced consistently by the **DEVS** framework with the help of **Timestamped Atomic Model**s.

V. Case Study

A. Model Description

The Aircraft and Control Tower (A/CT) models are simulated in the Co-Simulation framework. The A/CT models are the appropriate example for the Co-Simulation, in which the Aircraft model is a continuous model and the Plane Tower model is a typical discrete event model.

In our case, the Aircraft is composed by 3 **BOM** Components: Dynamics, Communicator and Controller. Dynamics calculates Aircraft's flight profile, Communicator outputs the Aircraft information and receives the commands from Control Tower. The Controller interprets commands and executes them in Dynamics. Likewise, We build Control Tower using **DEVS**. The CT model guides Aircraft to take off and land on the airport. CT model watches the trace of the flight and sends the command events accordingly in a discrete manner. It is obviously that the A/CT models generate at lease two types of events communicated between models in different simulation architectures:

- The positions of Aircraft calculated by Dynamics component. The outputting messages are transformed to the events in **DEVS Proxy**.
- The command events from CT model, the events are sent to **DEVS Proxy** by **BOM Proxy**. And **DEVS Proxy** transforms them to the time stamped messages.

B. Experiments and Analysis

We choose the altitude of Aircraft to show the experimental results in Figure 7. The altitude changes with time from cruising altitude to ground. The attached red line means the events from CT model. It is shown that the **Landing Command**, **Hover Command** and **Track Available** events are received respectively at 1:25:30, 1:26:30 and 1:32:00. The Aircraft reacts to the events by lowering the Altitude.

In summary, the A/PT models benefits from the Co-Simulation architecture. The **BOM Proxy** and **DEVS Proxy** work well to synchronize the logical time and data transfer. Although the **DEVS** models lose some efficiency in case of the synchronization with **BOM**-based simulation, the models in different formalisms are simulated together without any modification in this Co-Simulation framework.

VI. Conclusion

The objective of Domain Specific Modeling is to avoid the accidental complexity. The multi-formalisms brings not only convenience in modeling, but also the problem of simulator difference at run-time. The model transformation may lose some model features, therefore we switch to Co-Simulation. The **BOM Proxy** and **DEVS Proxy** are customized for Co-Simulation between **BOM**-based simulation and **DEVS**-based simulation.

The **Timed Automata** models of proxies are used to check the Safety, Liveness and Reachability of the proxy models. A general Co-Simulation framework for **BOM** and **DEVS** is constructed on the implementation of the proxies. The **Timestamped Atomic Model** and **GSDC** algorithm are designed to solve the problems exists in data transfer and time synchronization. Finally the case study gives a typical example. According to the experimental results, the framework within proxies is testified to be successful in the Co-Simulation for **BOM** and **DEVS**.

References

[1] K. Czarnecki and S. Helsen, "Classification of Model Transformation Approaches," in *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of MDA*, 2003.

[2] L. Gheorghe, F. Bouchhima, G. Nicolescu, and H. Boucheneb, "A Formalization of global simulation models for Continuous/Discrete systems," in *SCSC: Proceedings of the 2007 summer computer simulation conference*. San Diego, CA, USA: Society for Computer Simulation International, 2007, pp. 559–566.

[3] B. A. d. Mello and F. R. Wagner, "A Standardized Co-simulation Backbone," in *VLSI-SOC '01: Proceedings of the IFIP TC10/WG10.5 Eleventh International Conference on Very Large Scale Integration of Systems-on/Chip*. Deventer, The Netherlands, The Netherlands: Kluwer, B.V., 2002, pp. 181–192.

[4] M. D'Abreu and G. Wainer, "M/CD++: modeling continuous systems using Modelica and DEVS," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005. 13th IEEE International Symposium on*, Sept. 2005, pp. 229–236.

[5] SISO, *Base Object Model (BOM) Template Specification*. SISO-STD-003-2006: Simulation Interoperability Standards Organization, 2006.

[6] J. Gong, "Research on System Framework of Extensible BOM-based Simulation," Ph.D. Thesis, School of Mechatronics and Automation, National University of Defense Technology, Changsha, China, 2007.

[7] P. B. Zeigler, P. Herbert, and K. T. Gon., *Theory of Modeling and Simulation, Second Edition*. Academic Press, 2000.

[8] B. H. G. L. B. F. Nicolescu, G., "Methodology for efficient Design of Continuous/Discrete-events Co-Simulation tools," in *Anderson, J., Huntsinger, R. (eds.) High Level Simulation Languages and Applications - HLSLA*, San Diego, CA, 2007, p. 172C179.

[9] H. P. Dacharry and N. Giambiasi, "A formal verification approach for DEVS," in *SCSC: Proceedings of the 2007 summer computer simulation conference*. San Diego, CA, USA: Society for Computer Simulation International, 2007, pp. 312–319.

[10] J. Bengtsson and W. Yi, "Timed Automata: Semantics, Algorithms and Tools," 2004, pp. 87–124.

[11] G. BEHRMANN, "A Tutorial on UPPAAL," *Proc, of SFM-RT'04*, 2004.