Semantics for Rollback-Based Continuous/Discrete Simulation

Luiza Gheorghe, Gabriela Nicolescu, Hanifa Boucheneb
Ecole Polytechnique de Montreal
luiza.gheorghe@polymtl.ca

ABSTRACT

Modern device trends present greater challenges to design [1] because many of them integrate continuous and discrete sub-systems and therefore their design involves specific global modeling and validation approaches. This paper proposes the operational semantics for rollback-based synchronization model that may be used in continuous/discrete systems simulation. The paper also addresses the formal representation of the behavior of the continuous/discrete simulation interfaces with respect to this mode. This representation enables the definition of generic and language independent co-simulation tools that can be used to provide global simulation models for continuous/discrete heterogeneous systems. The model was validated through simulation, using UPPAAL toolbox and its formal verification was realized by defining and checking the main properties.

1. INTRODUCTION

The past decade has observed the shrinking of the chips' size concurrently with the expansion of the number and the heterogeneity of components integrated on the same chip. Currently, most of the Systems-on-Chip (SoC) consist of pre-existing designed blocs. This enables cost-efficient solutions, an advantageous time-to-market and high productivity. However, one will notice the increase of the variability of design related parameters. Given their unmanageable complexity (which is the result of the diversity of concepts being manipulated), the global design specification and validation are extremely challenging. The heterogeneity of these systems makes the elaboration of an executable model for the overall simulation more difficult.

This work focuses on heterogeneous continuous/discrete (C/D) systems and their simulation models. These models are very complex; they include the execution of different components, the components adaptation and the interconnects interpretation. Their design requires tools with different models of computation and paradigms as well as the definition of *new models providing adaptations between components*. These adaptations are provided by the simulation interfaces that are in charge with the synchronization.

In a global C/D simulation model, the continuous and discrete models interact via events. The time stamps associated with these events are synchronization and communication points between the different simulators involved in a global simulation. The events exchanged between the simulators are:

- *discrete events* that are timed events scheduled by the discrete simulator.
- *state events* that are unpredictable events generated by the continuous simulator. Their time stamp depends on the values of state variables (e.g. a zero-passing or a threshold crossing).

The two main synchronization models that can be found in C/D simulation are:

- the Full Synchronization Model (FSM). In FSM the synchronization is realized at each discrete step and state event occurrence. The advantage is that this model is general; it respects the generic canonical synchronization model where the continuous simulator runs before the discrete simulator [2]. One of this model's disadvantages is the synchronization overhead caused by the number of unnecessary synchronisation steps.This model is detailed in [3].
- the Rollback-based Synchronization Model (RSM). In RSM the synchronization is realized only at the occurrence of unpredictable discrete events and/or state events. The discrete simulator has to backtrack if the continuous simulator generates a state event. This model reduces the number of synchronization steps and consequently the synchronization overhead. This property can be exploited if rollback featured discrete simulators are available.

Most of the simulators that support rollback are continuous simulators and by consequence many popular co-simulation approaches use FSM because it avoids the rollback. However, RSM is useful for the co-simulation of systems that integrate more than one continuous simulator and one discrete simulator as well as for the co-simulation of systems where real parallelism is required (e.g. distributed simulation).

This paper presents the operational semantics as well as the formal representation of the behavior of the C/D simulation interfaces, for a light rollback synchronization model. In a light rollback synchronization model, the discrete simulator will perform only a backup of the memory data segment, processor registers as well as input and output signal values for each output discrete event time stamps used as checkpoints. By representing the model formally, the system's requirements are precisely characterized. This representation allows for the definition and the formal verification of the synchronization model. Moreover, it constitutes the foundation for the definition of generic simulation tools that can provide global simulation models for C/D systems. In order to model, validate and check our model we used UPPAAL [4].

The article is structured as follows. Section 2 presents the main approaches for the C/D systems simulation. Section 3 introduces some of the basic concepts such as the discrete event formalism and timed automata. Section 4 details the synchronization model with rollback and its operational semantics while Section 5 shows the behavior of the discrete interface and its formal representation. Section 6 gives the experimental results; more precisely the model simulation and validation as well as the properties verification are detailed. Finally, section 7 presents our conclusions.

2. RELATED WORK

Some of the previous works in this field propose the utilization of a single language for the specification of the C/D system. These languages may be obtained by extension of existing HDLs [5][6][7]. Using these methods leads to the abandonment of certified efficient tools for the continuous domain (ex. Simulink).

There are tools in which the systems are designed by assembling together different components, each with its own design language [8][9]. However, the different sub-systems and components need to be developed in the same environment in order to be compatible thus they do not solve the problem of IP reuse in system design. Moreover, the formal verification of the simulation models is not considered.

A different approach for systems validation is based on the formal representation of the C/D systems. In [10], the authors propose a formal classification framework that makes it possible to compare and express differences between models of computation. In [11] the author proposes the formalization of the heterogeneous systems by separating the communication and the computation aspects. However, the formal verification of the interfaces between domains was not taken into consideration.

In [12] the author presents a formalism defined for the modeling and simulation of discrete event systems (Discrete EVent System Specifications - DEVS) where the time advances on a continuous time base. This approach can be used to build the models, using hierarchy and modularity. It allows the definition of the operational semantics for a system but not its formal verification.

The rollback is also presented in several works. [13] proposes a rollback algorithm for optimistic distributed simulation systems. In [14] the authors detail an incremental checkpoint mechanism that allows the system's rollback in order to recover the data. [15] presents a "time warping" algorithm that allows the rollback to a point where data consistency is guaranteed. However, the formalization and verification of the rollback mechanism in the context of the C/D simulation was never addressed.

The contributions of this paper are:

- The definition of the operational semantics for a C/D simulation models based on RSM
- The formal representation of the behavior of C/D simulation interfaces with respect to this synchronization model.
- The formal verification of the behavior of C/D interfaces in the context previously presented.

3. BASIC CONCEPTS

This section introduces some of the basic concepts that were used in this work. These concepts range from a discrete events formalism to timed automata.

3.1 Discrete Event Systems Specification (DEVS)

Discrete Event Systems Specifications (DEVS) is a formalism supporting a full range of dynamic system representation. The abstraction separates modeling from simulation and provides atomic models and the mechanisms for the definition of an operational semantics for the C/D synchronization model [12].

A DEVS is defined as a structure :

$DEVS = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, t_a \rangle$ where

$X = \{(p_d, v_d) | p_d \in InPorts, v_d \in X\, p_d\}$ set of *input* ports and their values in the discrete event domain,

S = set of *sequential states*

$Y = \{(p_d, v_d) | p_d \in OutPorts, v_d \in Y\, p_d\}$ set of *output* ports and their values in the discrete event domain.

$\delta_{int} : S \rightarrow S$ the *internal transition* function

$\delta_{ext}: QxX \rightarrow S$ the *external transition* function, where:

$Q = \{(s,e) | s \in S, 0 \le e \le t_a(s)\}$ set of total state,

e is the *time elapsed* since the last transition

$\lambda: S \rightarrow Y$ output function

t_a:S$\rightarrow R^+_{0,\infty}$ set of positive reals with 0 and ∞.

In the work presented here the DEVS formalism is used for a schematic formalism of the interface between the continuous and the discrete domain interfaces.

3.2 Timed Automata

A timed automaton (TA) is a formalism for modeling and verification of real time systems. It can be seen as classical finite state automata with clock variables and logical formulas on the clocks (temporal constraints) [16].

Timed automata have characteristics that make them desirable for our formal model. They are as follows:

- simplicity and flexibility for the systems' modeling,
- expressivity that is required in order to model time constrained concurrent systems.

Moreover, one can find of a whole range of powerful tools based on timed automata, that are already implemented and that allow different verification techniques.

Our formal model needs to support concurrency between C/D systems thus it was represented as a parallel composition of several timed automata with no constraints regarding the time spent in the locations.

4. ROLLBACK-BASED C/D SYNCHRONIZATION MODEL -

The simulation of continuous model, described by differential and algebraic equations, requires solving these equations numerically. A widely used class of algorithms discretizes the continuous time line into an increasing set of discrete time instants, and computes numerical values of state variables at these ordered time instants.

The simulation of discrete models is based on *events* ([15]). At each simulation cycle, the first event with the smallest time stamp is processed and the processes sensitive to this event are executed. This may generate other events causing execution of other processes. Once all events with discrete time stamp equal to the current time have been treated, the simulator advances the time to the nearest discrete scheduled event.

4.1 Rollback-based synchronization model

Figure 1 presents the light rollback synchronization model for the C/D simulation interfaces.

For a rigorous synchronization, the discrete domain has to detect the events generated by the continuous domain and the continuous simulator must detect the scheduled events from the discrete domain. The simulators have to be controlled by the simulation interfaces in order to provide the functionalities described below.

At a given time the discrete simulator is in the state (x_{dk}, t_{dk}) with x_{dk} the location and t_{dk} the k^{-th} discrete time (that can be seen also as the k^{-th} event in the queue of events in the discrete domain). At this point the discrete simulator had executed all the processes sensitive to the event, advances to the time of the next event t_{dk+1} (arrow 1 in Figure 1) and a new state (x_{dk+1},t_{dk+1}), sends the data and the time of the event t_{dk+1} to the continuous simulator and switches the context to the continuous simulator (arrow 2 in Figure 1).

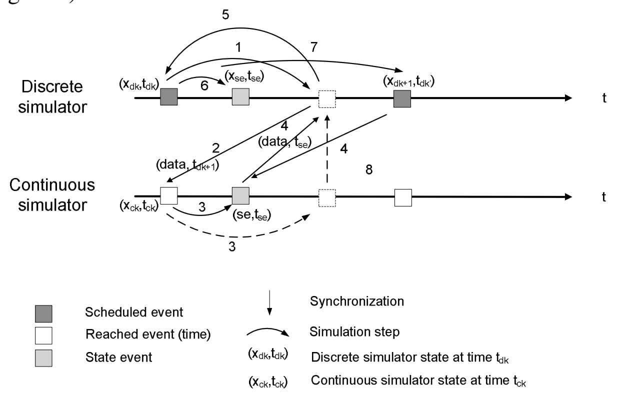

Figure 1. The light rollback synchronization model with state event.

The state of the continuous simulator is (x_{ck},t_{ck}) and the advance in time of the simulator cannot be further then t_{dk+1}, the time sent by the discrete simulator.

The behavior of the continuous interface can be described by the following transition state (arrow 3 in Figure 1):

$$\left(x_{ck},t_{ck}\right)=\begin{cases}(x_{ck+1},t_{ck+1}) \text{ if } t_{ck+1}=t_{dk+1} & \textbf{\textit{(1)}}\\ (se,t_{se}) \text{ if } t_{ck+1}<t_{dk+1} & \textbf{\textit{(2)}}\end{cases}.$$

where the state (x_{ck+1}, t_{ck+1}) is the state of the continuous simulator when no state event was generated in the time interval $[t_{ck}, ,t_{ck+1}]$. The state (se,t_{se}) represents the state of the continuous simulator when a state event *se* was generated and t_{se} represents the time when the state event occurred. In both situations the continuous simulator will stop and send the data to the discrete simulator and then switch the context to (x_{dk+1},t_{dk+1}), (arrow 4 in Figure 1). The event taken into consideration is the event generated within the time interval $[t_k,t_{k+1}]$.

The case described by (1) is the case without state event where after switching the context, the continuous simulator will solve the equations that characterize the continuous components for the time interval $[t_{dk},t_{dk+1}]$. At the time t_{dk+1} the continuous solver will send the data to the discrete domain interface, switch the context to the discrete domain and the cycle restarts.

Equation (2) describes the case where a state event occurred. The continuous simulator will send not only the data but also the time when the state event occurred t_{se} (arrow 4 in Figure 1). The discrete simulator backtracks to the previous state (x_{dk},t_{dk}) (arrow 5 in Figure 1) and restores the saved data for the time stamp t_{dk}. After the state restoration, the simulator starts over, taking into account the state event and advances to the time stamp t_{se} (state event detected by the discrete simulator) where will execute all the processes sensitive to the event (arrow 6 in Figure 1). The cycle restarts, the discrete time advances to the next discrete event (arrow 7 in Figure 1). The time stamp of this event can change after a state event; it can take any value bigger than t_{se} .

Table 1. Operational semantics for continuous/discrete synchronization model with light rollback

Rules	Arrows in Figure 1
$\dfrac{synch=1\wedge flag=1\wedge back=1\wedge s_{dk+1}=\delta_{\text{int}}(s_{dk})}{(s_{dk},e_{dk})\xrightarrow{\delta_{\text{int}}(s_{dk})}(s_{dk+1},0)\xrightarrow{(DataFromBus,t_a(s_{dk}))!;flag:=0}(s_{dk+1},t_a(s_{dk}))}$	arrow 1
$\dfrac{synch=1\wedge flag=0\wedge q=\delta_{ext}(q)}{q\xrightarrow{(DataFromBus,t_a(s_{dk}))?;synch:=0}q}$	arrow 2
$\dfrac{synch=0\wedge flag=0\wedge back=1\wedge\neg statevent(t)\wedge q'=\delta_{\text{int}}(q)}{q\xrightarrow{\delta_{\text{int}}}q'\xrightarrow{DataToBus!;flag:=1}q'}$	dashed arrows 3 and 4 - no state event
$\dfrac{synch=0\wedge flag=1\wedge back=1\wedge\neg statevent\wedge s_{dk+1}=\delta_{ext}(s_{dk+1})}{(s_{dk+1)},t_a(s_{dk}))\xrightarrow{DataToBus;\lambda(s_{dk+1})?;synch:=1}(s_{dk+1},0)}$	dashed arrow 4 - the receiving end
$\dfrac{synch=0\wedge flag=0\wedge back=1\wedge statevent\wedge q'=\delta_{\text{int}}(q)}{q\xrightarrow{\delta_{\text{int}}(q)}q'\xrightarrow{DataToBus!;t_{se}!;flag:=1}q'}$	arrow 3 and 4 – state event
$\dfrac{synch=0\wedge flag=1\wedge back=1\wedge statevent\wedge s_{dk+1}=\delta_{ext}(s_{dk+1,}t)}{(s_{dk+1},e_{dk+1})\xrightarrow{DataToBus?;t_{se}?;\lambda(s_{dk+1});synch:=1;back:=0}(s_{dk+1},0)}$	arrow 4 the receiving end
$\dfrac{synch=1\wedge flag=1\wedge back=0\wedge s_{dk}=\delta_{\text{int}}(s_{dk+1})}{(s_{dk+1},e_{dk+1})\xrightarrow{\delta_{\text{int}}(s_{dk+1});back:=1}(s_{dk},e_{dk})}$	arrow 5
$\dfrac{synch=1\wedge flag=1\wedge back=1\wedge s_{dse}=\delta_{\text{int}}(s_{dk})}{(s_{dk},t_{se})\xrightarrow{\delta_{\text{int}}(s_{dk})}(s_{dse},0)\xrightarrow{(DataFromBus,t_a(s_{dse}))!;flag:=0}(s'_{dk+1},t_a(s_{dse}))}$	arrow 6 and 7

4.2 Operational semantics for the rollback-based synchronization model

The operational semantics (OS) for C/D systems requires the rigorous representation of the relation between the simulators (communication/synchronization, data exchanged between the continuous and the discrete simulators) as well as their high level and dynamic representations. The OS for the light rollback synchronization model is given by the set of rules presented in Table 1, using DEVS (as it was presented in Section 3.1). *DataToBus* is the output function from the discrete domain interface, and *DataFromBus* is the output function from the continuous domain interface. The semantics of the global variable "flag" is related to the context switch between the continuous and discrete simulators. When "flag" is set to '1', the discrete simulator is executed. When it is '0', the continuous simulator is executed. The global variables "synch" and "back" are used to impose an order. When "back" is 1 the discrete simulator advances to the next time stamp while when it is 0, it backtracks to the previous time stamp while "synch" is 0 for the switch context between the discrete and the continuous simulator and 1 for the advancement of the discrete simulator (it eliminates a potential decidability problem for the discrete simulator when receiving data from the continuous simulator).

For further clarification, we detail here the first rule, corresponding to the arrow 1 in Figure 1. The premises of this rule are: the variables "synch", "flag" and "back" have the value '1', and there is an internal transition function (δ_{int}) for the discrete model. The discrete model is initially in the total state (s_d, e_d), this means it is in the state s_d from the time e_d. In this state the discrete simulator performs the following actions:

- send the data and the value of its next time stamp (this action is expressed by *(DataFromBus,* $t_a(s_d)$*)!*.

- switch the simulation context to the continuous model (this action is expressed by *flag = 0*).

For the same rule, the continuous model is in state *q* and performs the following actions:

- receive the data and the value of the time stamp from the discrete simulator (expressed by *((DataFromBus,* $t_a(s_d)$*)?*.

- set the global variable synch to '0' (action expressed by *synch=0*) in order to respect the premise of the rule corresponding to the arrow 4.

The actions expressed by this rule will be executed by the discrete simulator when the context will be switched to it.

5. DISCRETE DOMAIN SIMULATION INTERFACE

The C/D simulation interfaces are formed of two distinct, domain specific interfaces, one for the continuous domain and one for the discrete domain. The continuous domain interface is the same for the case of the light rollback and the canonical synchronization model and was detailed in [17]. This chapter details the discrete domain interface.

Figure 2 presents the flowchart of the behavior of the discrete domain interface when the light rollback synchronization mode is used. Based on this flowchart we formalized the discrete simulation interface.

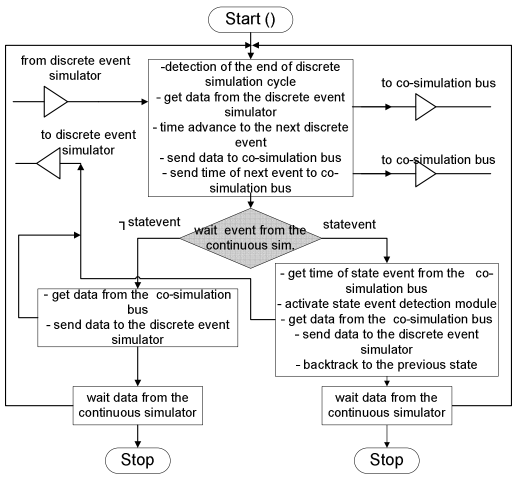
Start ()
from discrete event simulator
-detection of the end of discrete simulation cycle
- get data from the discrete event simulator
- time advance to the next discrete event
- send data to co-simulation bus
- send time of next event to co-simulation bus
to co-simulation bus
to discrete event simulator
to co-simulation bus
¬statevent
wait event from the continuous sim.
statevent
- get data from the co-simulation bus
- send data to the discrete event simulator
- get time of state event from the co-simulation bus
- activate state event detection module
- get data from the co-simulation bus
- send data to the discrete event simulator
- backtrack to the previous state
wait data from the continuous simulator
wait data from the continuous simulator
Stop
Stop

Figure 2. Flowchart for discrete domain simulation interface

Figure 3 shows the formal model (using timed automata) for the discrete domain interface.

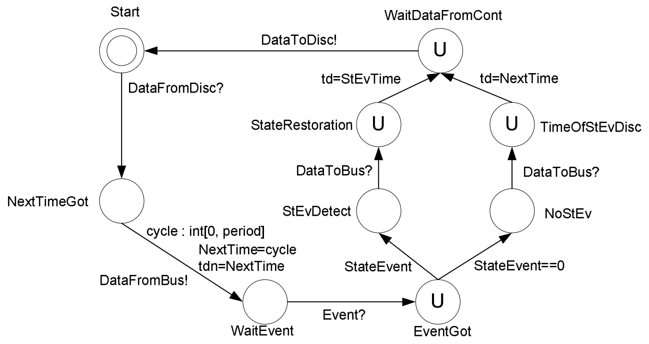

Figure 3. The discrete domain interface model

The model has only one initial location (a double circle in Figure 3) *Start*. The discrete interface will change location from *Start* to *NextTimeGot* following the transition $\textbf{\textit{Start}} \xrightarrow{\textit{DataFromDisc?}} \textbf{\textit{NextTimeGot}}$. This is an external transition realized with zero time and it is triggered by the receiving of the data (that is also synchronization between the discrete simulator and the interface) from the discrete simulator (`DataFromDisc?`).

Here the interface receives the data from discrete simulator and the time of the current event in the discrete domain. The location changes then to *WaitEvent*. The discrete interface sends to the continuous interface the time of the current event (the synchronization `DataFromBus!`*)*. The variable `NextTime` represents the time of the events in the discrete domain. This variable takes the value `cycle`. This value is then assigned to the variable `tdn` that represents the time stamp of the event. The theory normally assumes equidistant sampling intervals. This assumption is not usually achieved in practice. For an accurate simulation we assume that the cycle takes random values in an interval defined here as *[0, period]*. In *WaitEvent* location, the context is switched from the discrete to the continuous simulator. When the context is switched back to the discrete simulator, the location is changed to *EventGot* following the synchronization transition: $\textit{WaitEvent} \xrightarrow{\textit{Event?}} \textit{EventGot}$. During this transition the discrete interface receives from the continuous interface the synchronization `Event?`. In this location the occurrence of a state event in the continuous domain is considered. Two cases are possible:

1) When no state event was generated by the continuous domain, the location changes from *EventGot* to *NoStEv*. The transition $\boldsymbol{EventGot} \xrightarrow{StateEvent\ ==0} \boldsymbol{NoStEv}$ is annotated in this case only with the guard `StateEvent==0`. This state changes to *TimeOfStEvDisc* (that is an urgent location) following the transition $\boldsymbol{NoStEv} \xrightarrow{DataToBus?} \boldsymbol{TimeOfStEv\ Disc}$. This is an external transition realized with zero time and it is triggered by the receiving of the data (that is also synchronization between the discrete and the continuous interfaces) from the continuous interface (`DataToBus?`). During this transition only the data is sent to the discrete simulator. The system will immediately change the state to *WaitDataFromCont* while updating the time in discrete with the time stamp of the current event (`td=NextTime`).

2) When a state event was generated by the continuous domain the location changes from *EventGot* to *StEvDetect* following the transition: $\boldsymbol{EventGot} \xrightarrow{StateEvent} \boldsymbol{StEvDetect}$. This transition is annotated with a guard (`StateEvent`). This state changes to *StateRestoration* following the transition: $\boldsymbol{StEvDetect} \xrightarrow{DataToBus?} \boldsymbol{StateResto\ ration}$. This is also an external transition realized with zero time. During this transition the data and the time of the state event t_{se} are sent to the discrete simulator. The system will immediately change the state to *WaitDataFromCont* while updating the time in discrete with the time stamp of the state event (`td=StEvTime`).

From *WaitDataFromCont* state the location changes to *Start*. The discrete interface sends to the discrete simulator the data and the time of the events (state event or discrete event) and is represented here by the synchronization `DataToDisc!`.

6. Model Validation

The formalization and verification of the simulation interfaces behavior stage can be divided into three steps: formalization (that can be the formal specification of the heterogeneous system and it was already presented in section 4.2), the validation by simulation and the formal verification. This section presents the last two steps, the simulation and the formal verification.

6.1 Formal Model Simulation

The UPPAAL tool allowed the validation of the system's expected behavior regarding functionality: synchronization, conflicts, communication. The main advantage of UPPAAL is that the product automaton is computed on-the-fly during verification and reduces the computation time and the required memory space. We simulated all the possible dynamic executions of our model.

Figure 4 shows a screenshot with the simulator. We observe in the left panel the variables. It displays the values of the data and clock variables in the current location or transition selected in the trace of the simulation control panel (the symbolic traces).

The right panel allows the visualization of the message sequence chart (also known as simulator). The vertical lines in the simulator window represent the transitions between the locations while the horizontal lines are the synchronization points. In this figure the communication between the interfaces as well as the communication between the simulators and the domain specific interfaces are represented by the same horizontal lines.

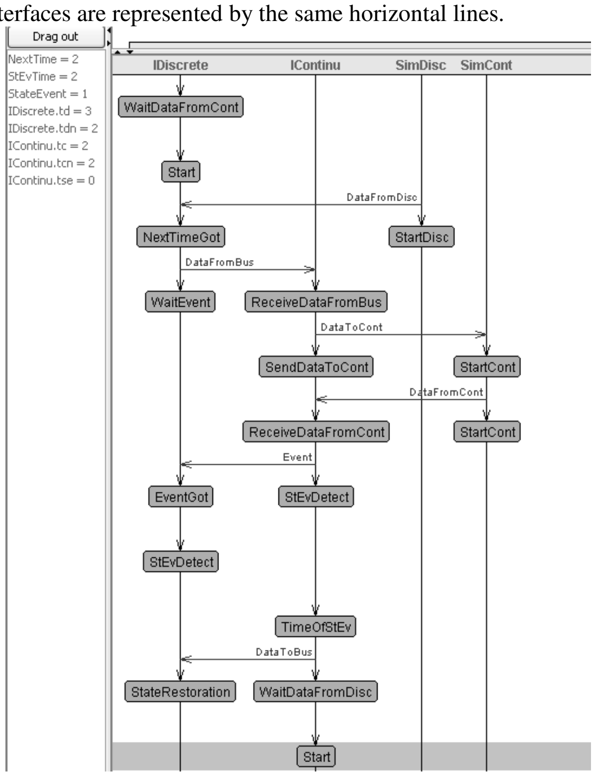
NextTime = 2
StEvTime = 2
StateEvent = 1
IDiscrete.td = 3
IDiscrete.tdn = 2
IContinu.tc = 2
IContinu.tcn = 2
IContinu.tse = 0

Figure 4. Simulation screen capture

As shown here, the simulation was stopped by the user after the state event was indicated to the discrete simulator and the time of the state event (`StEvTime=2`) was sent from the continuous to the discrete interface. The variable panel shows that the variable `StateEvent=1`, the time of the state event `StEvTime=2` while the discrete time is `td=3 and t`he next discrete time is `tdn = 2`. The system must backtrack and this can be seen in the simulator panel where the discrete model changes the state to *StateRestoration.* The next step is the advancement of the discrete to the `NextTime` that is 2, the time of the state event and implicitly its detection.

6.2 Formal Verification

The two main techniques that can be used for the formal verification of the interfaces are [18]:

- *model checking* where the system descriptions are given as automata, the specification formulas are given as temporal logic formulas and the checking consists in the verification if all models of a given system description satisfy a given specification formula.
- *theorem proving* where the verification plan is manually designed and the correctness of the steps in the plan is verified using theorem provers.

In this work we used UPPAAL that is based on model checking. The formal verification consists of checking properties of the system for a broad class of inputs . In our work we checked properties that fall into three classes [19]:

- Safety properties - the system does not get into an undesirable configuration, (i.e. deadlock)

- Liveness properties - some desired configuration will be visited eventually or infinitely (i.e. expected response to an input) .
- Reachability properties – the system always has the chance of reaching a given situation (some particular situation can be reached) .

The properties verified in order to validate the synchronization model are described below.

P0 Absence of deadlock ***(safety property)***

A state is a deadlock state if there are no outgoing action transitions either from the state itself or any of its delay successors [18].

```
A[] not deadlock
```

P1 State event detected by the discrete domain ***(liveness property)***

The indication of a state event by the continuous interface and its detection by the discrete interface are very important for C/D heterogeneous systems. We defined the property in order to check this behavior that is stated as follows:

Definition: A state event detected in the continuous domain `leads` to a state event detected in the discrete.

```
IContinu.StEvDetect --> IDiscrete.StEvDetect
```

P2 No state event in discrete if no state event in continuous domain ***(safety property)***

In order to avoid false responses from the discrete simulators, we defined a safety property to verify if the system will “detect” a state event in the discrete when it was not generated (and indicated) by the continuous domain:

Definition: Invariantly a state event detected in the discrete domain imply state event in the continuous.

```
A[](IDiscrete.StEvDetect imply StateEvent)
```

P3 Synchronization between the interfaces ***(reachability property)***

One of the most important properties characterizing the interaction between the continuous and the discrete domains is the communication and implicitly the synchronization. This property verifies that after a cycle executed by each model, both are at the same time stamp (and by consequence are synchronized)

Definition: Invariantly both processes in the *Start* location (initial state) imply the time in the continuous t_c is equal or smaller then the time in the discrete t_d.

```
A[]( (IDiscrete.Start and IContinu.Start)
imply ( IContinu.tc - IDiscrete.td <=
period))
```

P4 Synchronization between the interfaces when a state event was detected ***(reachability property)***

This property verifies that there is synchronization between the interfaces even when a state event is detected.

Definition: The discrete process in the *StateRestoration* location and the `continuous` process in the *StEvDetect* location leads to the time in the continuous t_c is equal with the time in the discrete t_d.

```
(IDiscrete.StateRestoration and
IContinu.StEvDetect) --> (IContinu.tc-
IDiscrete.td == 0)
```

7. CONCLUSIONS

This paper introduced the operational semantics for a C/D simulation model with a rollback synchronization model. We also give the distribution of the synchronization functionality to the simulation interfaces and formal representation and verification of the behavior of the C/D simulation interfaces. The formalization was realized with respect to the presented synchronization model using timed automata. The model was validated through simulation. In order to verify the formal representation, some properties were defined and checked using the model checker UPPAAL.

8. REFERENCES

[1] International Technology Roadmap for Semiconductor Design. Available: *http://public.itrs.net/.*

[2] Ghasemi, H. R. 2005. An effective VHDL-AMS simulation algorithm with event. Int. Conf. on VLSI Design.

[3] Gheorghe, L. et al. 2006. Formal definitions of simulation interfaces in a continuous/discrete co-simulation tool. RSP06.

[4] Behrmann, G., David, A. and Larsen, K. 2005. A Tutorial on UPPAAL. Real-Time Systems Symposium, Miami.

[5] IEEE Standard VHDL Analog and Mixed-Signal Extensions (1999), IEEE Std 1076.1-1999

[6] Patel, D. H. and Shukla, S. K. 2004. SystemC kernel – Extensions for heterogeneous System modeling. Kluwer Academic Publishers.

[7] Vachoux, A., Grimm, C. Einwich, K. 2003. Analog and mixed signal modeling with SystemC-AMS.

[8] Ptolemy project. Available: *http://ptolemy.eecs.berkeley.edu/*

[9] Lee, E. A. and Zheng, H. 2005. Operational Semantics of Hybrid Systems. Proc. of Hybrid Systems Computation and Control (HSCC).

[10] Lee, E.A. and Sangiovanni-Vincentelli, A. L. 1996. Comparing Models of Computation. IEEE Proc. of the Int. Conf. on Computer-Aided Design (ICCAD).

[11] Jantsch, A. 2003. Modeling Embedded Systems and SoCs - Concurrency and Time in Models of Computation. Systems on Silicon. Morgan Kaufmann Publishers.

[12] Zeigler, B.P., Praehofer H. and Kim, T.G. 2000. Modeling and Simulation - Integrating Discrete Event and continuous complex dynamic systems. Academic Press, San Diego.

[13] Madisetti, V., Walrand, J. and Messerschmitt, D. 1988. WOLF: A rollback algorithm for optimistic distributed simulation systems. Proc of the 1988 Winter Simulation Conference.

[14] Feng, T. H. and Lee, E. A. 2006. Incremental checkpointing with application to distributed discrete event simulation. Proc of the 2006 Winter Simulation Conference.

[15] Cassandras, C. G., Lafortune, S. 2007. Introduction to discrete event systems. Springer.

[16] Alur, R. and Dill, D. 1990. Automata for modeling real-time systems. Proc. 17-th Int. Colloquium on Automata, Languages and Programming.

[17] Gheorghe, L. et al. 2007. A Formalization of Global Simulation Models for Continuous/Discrete Systems. Proc of the 2007 Summer Simulation Conference.

[18] Wang, F. 2004. Formal verification of times systems: a survey and perspective. Proc. of the IEEE, vol. 92.

[19] Monin, J-F. 2003. Understanding Formal Methods. Springer.