Preprints of the 2nd IFAC Conf. on Analysis and Design of Hybrid Systems (Alghero, Italy), 7-9 June 2006

CODIS – A FRAMEWORK FOR CONTINUOUS/DISCRETE SYSTEMS CO-SIMULATION

Gabriela Nicolescu, Faouzi Bouchhima, Luiza Gheorghe

Ecole Polytechnique de Montreal

Abstract: This paper presents CODIS, a co-simulation framework for continuous/discrete systems. Based on a well defined synchronization model and a generic architecture for continuous/discrete simulation models, this framework enables easy specification and automatic generation of simulation models. The supported simulators are Simulink for continuous components and SystemC for discrete components.

Keywords: models, simulation, heterogeneity, accuracy, performance.

1. INTRODUCTION

Modern systems that may be found in various domains like automotive, defense, medical and communications, integrate continuous and discrete models. In a recent ITRS study covering the domain of mixed continuous discrete systems, the conclusion is a "*shortage of design skills and productivity* arising from lack of training and poor automation with needs for basic design tools" as one of the most daunting challenges in this domain (ITRS, 2003).

One of the main difficulties in the definition of CAD tools for continuous/discrete (C/D) systems is due to the heterogeneity of concepts manipulated by the discrete and the continuous components. Therefore, in the case of validation tools, several execution semantics have to be taken in consideration in order to perform global simulation:

- In discrete models (DM), the time represents a global notion for the overall system and advances discretely when passing by time stamps of events, while in continuous models (CM), the time is a global variable involved in data computation and it advances by integration steps that may be variable.
- In discrete models, processes are sensitive to events while in continuous models processes are executed at each integration step.

Currently, co-simulation is a popular validation technique for heterogeneous systems. This technique was successfully applied for discrete systems, but very few applied it for C/D systems. The co-simulation allows joint simulation of heterogeneous components. This requires the elaboration of a global execution model, where the different components communicate through a co-simulation bus via simulation interfaces performing adaptation.

For C/D systems co-simulation, the simulation interfaces have to provide efficient synchronization models in order to cope with the heterogeneous aspects cited above. This implies a complex behavior for these interfaces; their design is time consuming and an important source of error. Simulation interfaces play also an important role in accuracy and performance of global simulation. Consequently, the definition of new co-simulation tools able to provide simulation interfaces is mandatory.

This paper presents CODIS (COntinuous/DIscrete Systems simulation), a co-simulation framework for C/D systems validation. This framework assists designers in building global simulation models. The supported simulators are Simulink for continuous models and OSCI SystemC simulator for discrete models.

2. CODIS FRAMEWORK

1.1 *Synchronisation and generic architecture for C/D simulation in CODIS*

For an accurate synchronisation, each simulator involved in a C/D co-simulation must consider the events coming from the external world and it must reach accurately the time stamps of these events. We refer to this as *events detection*. These time stamps are the synchronization and communication points between the simulators involved in the co-simulation. Therefore, the continuous simulator, Simulink, must detect the next discrete event (timed event) scheduled by the discrete simulator, once the latter has completed the processing corresponding to the current time. In case of SystemC, these events are: clock events, timed notified events, events due to the *wait* function. This detection requires the adjustment of integration steps in Simulink (see Fig. 1).

The discrete simulator, SystemC, must detect the *state events.* A state event is an unpredictable event, generated by the continuous simulator, whose time stamp depends on the values of the state variables (ex: a zero-crossing event, a threshold overtaking

event, etc.). This implies the control of the discrete simulator advancement in time: in stead of advancing with a normal simulation step, the simulator has to advance precisely until the time stamp of the state event (see Fig. 1).

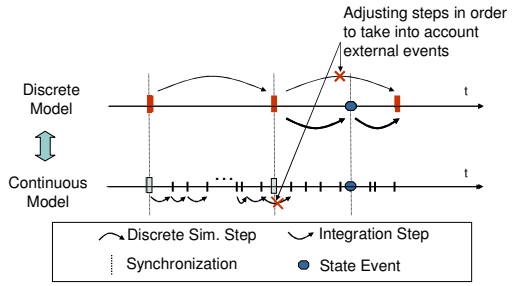

Fig. 1. C/D Synchronisation in CODIS

Fig.2 illustrates the generic architecture used in CODIS for the C/D simulation. CM and DM communicate through a co-simulation bus via simulation interfaces. Each simulation interface presents two main *layers*:

- The *synchronization layer* provides the synchronisation requirements discussed above. For both CM and DM, this layer is composed of three sub-layers, each of them achieving an elementary functionality for synchronisation.
- The *communication layer* is in charge of sending/receiving data between CM and DM.

More details on synchronization and CODIS simulation architecture may be found in (Bouchimma et al., 2005)

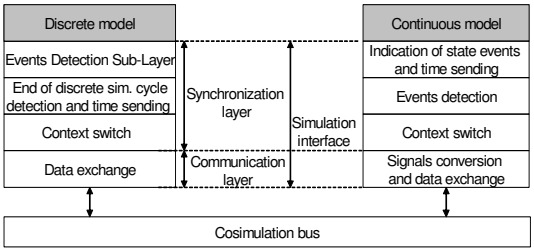

Fig 2. Generic architecture for accurate C/D simulation

1.2 Simulation model generation in CODIS

Based on the presented synchronization model and the generic architecture, a flow for automatic generation of simulation models was implemented in CODIS (see Fig. 3). The inputs of the flow are the CM in Simulink and the DM in SystemC. The output of the flow is the global simulation model, including the co-simulation bus and the simulation interfaces. The interfaces are generated by composing elements from the CODIS library. These elements implement the layers in Fig.2. They may be customized in terms of number of ports and their data type.

The interfaces for DM are automatically generated by a script generator that has as input user defined parameters. The model is compiled and the link editor calls the library from SystemC and a static library called "simulation library" (see Fig. 3).

The interfaces for Simulink are functional blocks programmed in C++ using S-Functions. These blocks are manipulated like all other components of the Simulink library. The user starts by dragging the interfaces from the library into the model's window, then parameterizes them and finally connects them to the model inputs and outputs. Before the simulation, the functionalities of these blocks are loaded by Simulink from the ".dll" dynamically linked libraries (see Fig. 3).

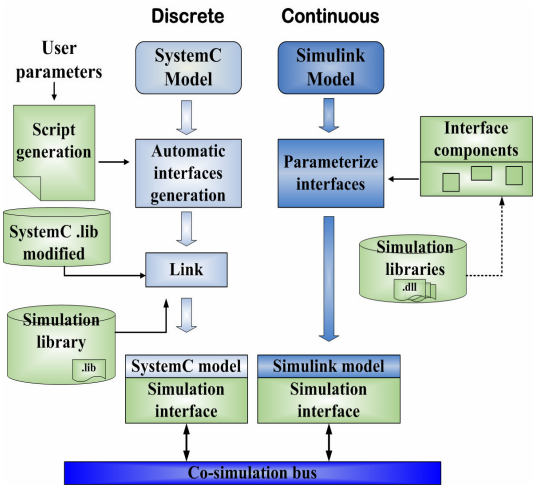

Fig. 3. Flow for simulation models generation

3. EXPERIMENTAL RESULTS

To analyze the capabilities of the proposed framework, we used two illustrative applications: a manipulator arm controller and a Δ/Σ converter.

To evaluate the performances of simulation models generated in CODIS, we measured the overhead given by the simulation interfaces. The overhead caused by the Simulink integration step adjustment when detecting a SystemC event has been measured in a maximum of 10% of total simulation time. The overhead caused by IPC (Inter Process Communication) used for the context switch and the communication layers has been measured in order of maximum 20% of the total simulation time. The cost of the added synchronization functionality in the case of SystemC is negligible and does not exceed 0.02% of the total simulation time.

4. CONCLUSION

This paper presented a simulation framework enabling continuous and discrete models integration. These models may be described using powerful tools for the two domains: Simulink and SystemC.

The experiments have shown a synchronization overhead of less than 30 % in simulation time.

REFERENCES

International Technology Roadmap for Semiconductor Design (2003), available at http://public.itrs.net/.

F. Bouchhima, *et al.* (2005) In: Discrete–Continuous Simulation Model for Accurate Validation in Component-Based Heterogeneous SoC Design, *Proceeding of RSP Conference*, (IEEE)