A Standardized Co-simulation Backbone

Braulio Adriano de Mello[1,2] and Flávio Rech Wagner[1]
[1]*Instituto de Informática – UFRGS – Porto Alegre – Brazil:* [2]*URI – Brazil*

Abstract: In the field of co-simulation, the construction of a bridge between different simulators and the solution of problems like synchronization and data translation are some of the main challenges. This paper discusses the advantages of the HLA (High Level Architecture) standard to solve these problems and presents a generic architecture to support environments for geographically distributed co-simulation, called Distributed Co-simulation Backbone (DCB), which is based on the HLA. This architecture is very flexible and does not enforce code modifications to the simulators to be integrated into the environment.

Key words: distributed co-simulation, simulation backbone, cooperation, HLA

1. INTRODUCTION

Co-simulation is used to make experiments and get information on the behavior of heterogeneous systems aiming at the validation of their design or at the evaluation of performance. Heterogeneous systems are characterized by a combination of hardware and software parts or by descriptions in different languages and/or at different abstraction levels [1]. In order to validate the design of embedded electronic systems, research in co-simulation mainly emphasizes the cooperative simulation of hardware and software parts.

Therefore, one of the major challenges in co-simulation is the construction of a mechanism for a consistent cooperative simulation involving those parts. This challenge has been increased by the evolution of technologies for communication and distributed processing [2].

Current techniques, environments, and tools for co-simulation have shown that one of the main bottlenecks is the communication interface. The

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35597-9_40
M. Robert et al. (eds.), *SOC Design Methodologies*

large variety of technologies and their continuous evolution make it difficult to conceive an adaptive (or generic) mechanism, which promotes the cooperation between heterogeneous simulators without imposing restrictions regarding their data formats or behavior.

This paper presents an architecture for a distributed co-simulation backbone, called DCB, which is based on the High Level Architecture (HLA) [3]. HLA has its roots on defense programs and has been recognized as a standard by the IEEE in 2000. It proposes rules and mechanisms for the interoperability of distributed, heterogeneous simulators. In particular, the Run Time Infrastructure, which is part of the standard, offers facilities that give an important contribution for the construction of a generic mechanism supporting distributed co-simulation environments.

The DCB architecture presents three definite advantages over other co-simulation environments: it is based on a public standard; it is far more flexible than current approaches, allowing an easier definition of particular environments; and it allows the integration of existing simulators without imposing modifications on their internal structures.

This paper is organized as follows. An overview of co-simulation is presented in Section 2. Section 3 discusses related work and introduces the contribution of DCB. The HLA standard, and its Run Time Infrastructure in particular, are introduced in Section 4. Section 5 thus presents DCB, which is a generic architecture to support the cooperation between heterogeneous simulators in distributed environments. Finally, Section 6 gives final remarks and discusses future work.

2. AN OVERVIEW OF CO-SIMULATION

The complexity of current embedded systems have raised so much that their design cannot be performed with a single design language nor at a single abstraction level. These systems usually aggregate hardware and software cooperating parts. Their designs ask for models combining descriptions at different abstraction levels, and multiple specification languages are needed for representing those parts and/or abstraction levels. Moreover, these parts are usually developed and validated by means of separate design processes. For this, the validation of the whole system must take communication and cooperation aspects into account, becoming extremely complex because of the system heterogeneity.

Aimed at the validation and performance evaluation of heterogeneous systems, the co-simulation approach has been the target of an increasing number of research groups. Its fundamental principle is the cooperative execution of different simulators [1]. Each simulator is responsible for a

different system part. The simulators may be executed in a single machine or in multiple machines (in LANs or WANs) [4]. Research is lately emphasizing co-simulation in wide area networks, due to potential benefits such as the management of intellectual property and cooperative design. In this scenario, each simulator may send and receive data through a co-simulation interface that must handle communication, synchronization, and data format conversions. The literature presents the construction of this bridge between heterogeneous simulators as one of the main challenges of co-simulation [5].

3. RELATED WORK

WESE (Web-based Environment for Systems Engineering) [6] is a collaborative and distributed environment for systems engineering. It supports distributed simulation with the cooperative execution of remotely located components. Components are stored in repositories called *factories* and are specified by the SSL language, which is specialized for web-based design. The environment is also suited for handling IP components.

IPCHINOOK [7] is a component-based design tool for distributed, embedded systems. Its main feature is a set of design abstractions to raise the level at which the designer interacts with the design environment. Communication and synchronization details are synthesized by the tool. The designer, however, must follow some modeling and synthesis rules.

The JavaCAD [8] tool supports the web-based reuse of IP components, guaranteeing the confidentiality of information for suppliers and clients. JavaCAD is implemented as a simulation backplane that supports the evaluation of IPs during the design process. For this, the user may instantiate IP components from multiple remote suppliers and simulate them transparently with proprietary blocks. For performing a simulation, the user must initially specify his/her design through a JavaCAD client, thus somehow restricting reuse flexibility.

Multilanguage approaches are used for the design of systems consisting of heterogeneous components, by offering environments that support multiple (but limited) languages. An example is the MCI tool [1]. Starting from an abstract description of a communication interface among subsystems, it automatically generates a specialized co-simulation environment. MCI allows a geographically distributed co-simulation of subsystems.

The Pia co-simulation tools [4] supply a specification mechanism that allows the interconnection of nodes through a geographically distributed network. Nodes are connected by sockets to the tool, so that other simulators, compilers, or even physical devices may be linked together.

A co-simulation approach based on the Ptolemy project [9] is presented in [10]. Following this approach, software and hardware simulators communicate through a master process (a backplane, in fact). This process manages communication between simulators, offering a standard interface that is based on a set of rules.

In general, existing approaches and tools are adequate for solving a set of predefined problems. Frequently, however, unexpected problems must be handled, regarding the integration of new tools. In this case, solutions that are proprietary or that are based on a given set of restrictions may impose an excessive burden, in terms of redesign of tools. Examples of restrictive conditions may be found in all approaches: particular techniques for model specification and synthesis, as in IPCHINOOK; use of predefined functions or structures, as in JavaCAD [8] and Pia sockets; limitations on the number or types of languages, as in multilanguage tools; and proprietary communication interfaces, as in some solutions based on backplanes.

The fundamental idea behind the architecture of DCB – Distributed Co-simulation Backbone is to remove the restrictions that are imposed upon independent simulators, considering aspects of interface, cooperation, and synchronization, thus adding great flexibility to co-simulation environments. Preserving the integrity of each simulator, DCB more easily handles unexpected situations, without loosing fidelity and precision. Besides that, the explicit use of non-proprietary, distributed solutions has not been addressed as a fundamental issue of other co-simulation approaches.

4. HLA AND CO-SIMULATION

4.1 Introducing HLA

HLA - High Level Architecture was initially conceived within the community of "distributed interactive simulation", considering special needs that were observed in military training [11]. Its development process involved government, academia, and industry, and its main concepts have been defined in 1995.

HLA offers a common architecture for the cooperative and distributed execution of individual simulations, also on WANs, as well as a structure for reusing simulations in existing or new applications. In order to enhance reuse and interoperability, HLA proposes the reduction of design restrictions and the implementation of fairly independent simulations, by using the object-oriented paradigm and introducing the idea of a simulation *federation* [3]. This concept introduces a high flexibility in the modeling process and eases the definition of interfaces, functionality, and cooperation [11]. HLA

is not a standard with a fixed set of rules or restrictions. Instead, it may be considered as a meta-standard, which defines meta-rules for building particular distributed simulation environments. HLA is specified by three main parts:

- the interface specification;
- the Object Model Template (OMT) [3], which defines a common and structured format for *federates* (the individual simulators) and *federations* (environments built from cooperating federates); and
- a set of rules for the definition and management of the behavior of federation and federates.

The interface specification defines the communication mechanism between the federates and includes the RTI – Run Time Infrastructure. The task of RTI is to offer services for communication and cooperation between federates, since they cannot communicate directly with each other. For establishing communication between federates, both RTI and the federates must offer communication methods, and the *ambassador's* paradigm may be used [12], as will be explained later on. A federate may represent a simulation model in a computer, a dedicated physical simulator, or an interface to a physical object or human.

The union of the federates with RTI is defined by a Federation Object Model (FOM). In turn, for each federate a Simulation Object Model (SOM) is defined. The FOM specifies a contract between federates, regarding the types of objects that will be visible among them and the interactions they will perform.

Although there is an expressive, inherent flexibility in HLA, it still imposes some restrictions:

- federates must implement methods for communication with RTI;
- federates and federation must follow a given set of rules (which may be, however, specialized for each federation);
- RTI may not accept a given communication protocol.

These restrictions apply mainly when new models or simulators have to be added to a certain federation and they have not been developed considering this federation. This problem is also found in co-simulation, where the heterogeneity of simulators must be handled in order that interoperability is achieved. These restrictive aspects in the construction of co-simulation environments have motivated research looking for more flexible approaches [12]. Nevertheless, the HLA standard has three definite advantages as a basis for co-simulation environments:

- it is a *standard*, so that we may see in near future commercial simulators that follow its guidelines and have an intrinsic interoperability;
- it is far more flexible than current approaches, allowing an easier definition of particular federations; and

- it allows the integration of existing simulators without imposing severe modifications on their internal structures, provided that they support some basic features.

4.2 Supporting co-simulation with the RTI

RTI specifies services to be offered to the federates, as well as services to be offered by the federates. These services can be grouped into three categories: federation management, data management, and time management. The federation management services control the dynamic inclusion and removal of federates and implement other global operations for the federation that are similar to those found in distributed co-simulation environments.

The main goals of the data management services are: control the data to be communicated between the federates, establishing origin – destination relationships; assign ownership of simulation objects to federates; offer support for the creation, removal and identification of data; and implement data routing between federates. Time management services control the time progression in the federates. These services must be also offered by mechanisms supporting distributed co-simulation environments, even if specified or implemented with alternative (usually proprietary) approaches.

The specification of RTI includes APIs that define how these services are accessed. The use of APIs, however, suggests adapting the code of a new simulator to be integrated into a federation. As an alternative to maintain transparency and to avoid the need of severe modifications in the simulator implementation, *gateways* between the federates and RTI may be introduced [12], as explained later on.

5. DISTRIBUTED CO-SIMULATION BACKBONE

DCB may be seen as a simulation-specific coordination layer for supporting distributed co-simulation. Its main goal is to offer a generic mechanism supporting communication and cooperation services between heterogeneous federates. The word "federate" refers to any computational sub-model, simulation sub-model, physical device, or even human actor.

DCB has been defined in the scope of the SIMOO project. SIMOO is a general-purpose, object-oriented simulator for discrete systems, which is being applied to the design of electronic embedded systems [13,14]. A non-distributed co-simulation tool, integrating SIMOO and VHDL, has been already implemented [15]. An adaption of SIMOO to the HLA standard is now underway [16]. The DCB implementation must support a transparent cooperation between models generated by SIMOO.

As strongly based on the HLA standard, DCB considers its definitions, especially those related to RTI, for building a co-simulation backbone. DCB has three main functions that are closely related to the RTI functionality: interface specification, synchronization management, and management of data exchanges between federates. In order to give a better support to co-simulation, however, DCB implements special strategies in these three parts, extending the RTI functionality and also removing some of its features that would inhibit flexibility.

5.1 Overall architecture

In order to handle the complexity of communication interfaces between heterogeneous federates and maintain flexibility, DCB uses the concept of *gateway* [12]. Gateways translate data formats, according to the destination of the data sent through DCB. Gateways are not implemented directly within the simulators or DCB, but as part of *ambassadors* [3]. Figure 1 shows the DCB architecture.

According to the DCB architecture, the simulators don't need to invoke communication primitives of DCB, as needed when using the RTI, for sending data. The simulator transfers output data to its ambassador. This, in turn, makes these data available at its interface, and the DCB ambassador will check for the data. In the opposite direction, the same tasks are performed by the two ambassadors.

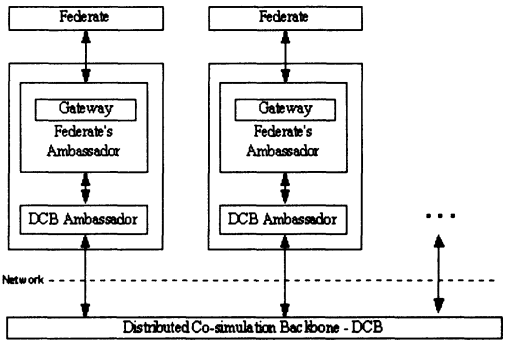

Figure 1. Architecture of the DCB

Because of this architecture, two ambassadors (the simulator and the DCB ones) must be developed when a new simulator is integrated into a co-simulation environment. They are specific for a particular application model, because they depend on the particular data to be exchanged through the sub-model interfaces. It is thus much more appropriate to talk about the

integration of a new *federate* (or sub-model) into an environment, and not of a new *simulator*.

If a federation includes N federates, even if they are implemented by the same simulator type (e.g. a VHDL simulator), 2N different ambassadors will need to be developed. These 2N ambassadors, however, are very similar, following the same control patterns, and differing only by the attributes they must control.

Although there may be some implementation effort in the development of ambassadors, they avoid modifications both in the communication and synchronization infrastructure (the DCB kernel) and in the simulator. These modifications would be also very costly, if compared to the construction of ambassadors.

The DCB infrastructure is general-purpose and is not affected by particular federates to be integrated into a federation. Furthermore, a simulator can be easily integrated into a DCB co-simulation environment because its code doesn't need to be reprogrammed at all. There are, however, certain requirements that must be met by federates, as discussed in the following sub-sections, in order to integrate them.

5.2 Interface specification

When a new federate is integrated into the co-simulation environment, the environment designer must explicitly declare the *attributes* that will be used for data exchanging and for synchronization at the sub-model interface. These attributes are used by DCB and the federate (in fact through their ambassadors) as structures for sending and receiving data.

In the attribute specification (a process called *interface configuration*), the designer must include the following attributes:

- the simulator execution mode (synchronous or asynchronous - a constant value);
- TLS – time of the last state saved by the federate;
- LVT – local virtual time of the federate;
- set of output variables of the federate; and
- set of input variables of the federate.

The execution mode and the LVT are standard attributes for all federates. The TLS is also mandatory for asynchronous simulators. The remaining attributes may vary in quantity and purpose and depend on the particular application model.

Therefore, according to the DCB approach, a simulator may be integrated into a co-simulation environment only if it can make both TLS and LVT available at its interface. When a federate is included in the environment, the designer must also indicate the other federates with which

the new one must interact and through which attributes the cooperation will take place. With this information, DCB implements the data exchanges, guaranteeing consistency regarding synchronization and interfaces.

Attributes are handled by the ambassadors, which are responsible for their management. Ambassadors must keep track of attribute values, by monitoring their changes, translating them to other data formats, and sending their contents to the respective destinations.

A new federate that is added to a co-simulation environment receives a unique identifier from DCB. This feature doesn't allow the recognition of two identical (replicated) federates in the same federation. In DCB, each federate (or simulator instance) is unique.

5.3 Data management

Data management in DCB reflects the basic principles prescribed by the HLA standard but is adapted to the purposes of generality and heterogeneity that are fundamental to DCB.

RTI offers six different classes of services [3], and four of them are related to data management: Declaration Management; Data Distribution Management; Object Management; and Ownership Management. Since some of these services define requirements that affect the implementation of the federates, their use in DCB would be restrictive, considering that code modifications in the simulators to be included in a federation are not desirable. Therefore, these particular services are not implemented by DCB.

The *ownership management* service, in turn, is very important. As RTI, DCB defines and manages through this service a responsibility over the attributes that it shares with all federates. Only a federate that *owns* a given attribute may update its vale, and the ownership of the attribute may vary along the time. It is thus possible for DCB to implement a mutual exclusion policy regarding the use of attributes by the federates, so avoiding undue attribute value updates, performed either by the local or by a remote federate. Ownership management is essential also for synchronization purposes, as explained next.

5.4 Synchronization

According to the HLA definition, while each federate has its own local virtual time, RTI manages the global simulation time. RTI offers pre-defined control functions that must be used by the federates in order to receive a grant for advancing their local times. This approach, however, makes the integration of new heterogeneous simulators into a federation a more complex task, since each simulator must be coded so that it calls those

control functions. Therefore, although using the HLA global mechanism for time advancement, DCB does not follow RTI's approach of control functions.

As prescribed by the HLA standard, DCB supports an hybrid synchronization (synchronous or asynchronous time advancement) [11]. In distributed environments, the asynchronous approach usually presents less overhead. In order to implement an asynchronous mode, however, a simulator must support rollback to a previous safe state [17], because of events it receives with past time stamps. The simulation performance may be degraded if federates must execute rollbacks very often, but this is highly dependent on the application.

In the synchronous mode, DCB uses the ownership management service together with the LVT of the federates in order to guarantee that only safe events are executed. Suppose two federates A and B, and A sends a message that modifies a value of an attribute X of B. DCB will grant the ownership of X to A, thus allowing A to modify the value of B. Besides that, each federate may only increment its LVT if the ownership of the LVT is granted to it by DCB. From this strategy, it becomes clear that a synchronous simulator may be integrated into a DCB federation only if it may declare the LVT at its interface and wait for a grant signal from the ambassador to advance the LVT's value.

In order to optimize performance in a synchronous operation, DCB supports the look-ahead mechanism [18], both static and dynamic. If L is the look-ahead value, then the update of an attribute X of a federate B, by another federate A, is safe if $(LVT(A)+L) \geq (LVT(B))$.

The same ownership management mechanism is used for time advancement in the asynchronous mode. The only difference is that, in this case, DCB supports unsafe events and implements a rollback mechanism. For this, federates need to store safe states. DCB, on the other side, must store messages sent by the federates with future time stamps, if compared to the global time, and send anti-messages [17] in case of causality errors.

6. CONCLUSIONS AND FUTURE WORK

One of the most relevant challenges in co-simulation is the construction of an adequate mechanism for the cooperation between heterogeneous simulators. This paper presented a generic architecture, called DCB, which is based on the HLA standard and supports distributed environments consisting of heterogeneous simulators. DCB presents flexible strategies for the management of data interactions and synchronization between the simulators.

Generality and flexibility have been the main requirements in the development of the DCB. New simulators may be easily introduced into an heterogeneous and cooperative environment, without regard to their implementation technologies or languages and without needing any modification of their internal code.

The simulators, however, must show a few fundamental features, which have been discussed in the paper, in order to be able to participate in a federation. Nevertheless, these features are very simple and may be considered as unavoidable, if one wants to integrate a simulator into a cooperative environment.

Because of its generality regarding the management of the integration of heterogeneous simulators, DCB may be used as a basis for the implementation of distributed simulation environments in a wide range of application domains.

Different domains, however, may have distinct performance requirements, and synchronization requirements and temporal restrictions may vary heavily. Consider for instance the distinctions between a pure software co-simulation of electronic designs, a co-simulation environment with physical mechatronic components-in-the-loop, a co-simulation environment with physical electronic components-in-the-loop, and a training simulation environment with human-in-the-loop. These very distinct situations considering real-time requirements may enforce different implementations of the DCB architecture.

Using DCB, the tool designers must interact just with the ambassadors, because DCB's main goal is to provide and support cooperation among any existent simulators. On the other side, with commercial tools, such as from Cadence, Coware, Cossap, and Synthesia, the tool designers must interact with a more complex design environment and are usually restricted to a given number of description languages, such as C++, VHDL, or Verilog.

For the validation of the DCB architecture, a particular federation for the co-simulation of electronic embedded systems is being implemented. In parallel, a supporting environment for the DCB development methodology is being built. It will offer services and resources for the configuration of interfaces of federates and for the semi-automatic generation of the ambassadors. The object-oriented features of the SIMOO modeling and simulation framework [13] will be basic for this supporting environment.

7. REFERENCES

[1] F.Hessel, P.L.Marrec, C.Valderrama, M.Romdhani, and A.A.Jerraya, "MCI: Multilanguage Distributed Cosimulation Tool". In: F.J.Rammig (ed.), *Distributed and*

Parallel Embedded Systems. Kluwer Academic Publishers, 1999. (Proceedings of DIPES'98)

[2] G.Borriello and R.Want. "Embedded Computation Meets the World Wide Web", *Communications of the ACM*, Volume 43, n. 5, May 2000.

[3] F.Kuhl, R.Weatherly, and J.Dahmann. Creating Computer Simulation Systems: An Introduction to the High Level Architecture. Prentice Hall, MITRE Corporation, 2000.

[4] K.Hines and G.Borriello. "A Geographically Distributed Framework for Embedded System Design and Validation". *In: Proceedings of the 35th Design Automation Conference*, June 1998.

[5] K.Kim, Y.Kim, Y.Shin, and K.Choi. "An Integrated Hardware-Software Cosimulation Environment with Automated Interface Generation". *In: 7th International Workshop on Rapid Systems Prototyping*, June 1996.

[6] R. Dhananjai, V.Chernyakhovsky, and P. A.Wilsey, "WESE: A Web-based Environment for Systems Engineering". In: *Proceedings of the 2000 International Conference On Web-Based Modelling & Simulation (WEBSIM 2000)*. January 2000.

[7] P.Chou, R.Ortega, K.Hines, K.Partridge, and G.Borriello. "IPChinook: An Integrated IP-based Design Framework for Distributed Embedded Systems". In: *Proceedings of the 36th Design Automation Conference*, New Orleans, June 1999.

[8] M.Dalpasso, A.Bogliolo, and L.Benini. "Virtual Simulation of Distributed IP-based Designs", *Proceedings of the 36th Design Automation Conference*, New Orleans, June 1999.

[9] E.A.Lee et al. "Overview of the Ptolemy Project". *Technical Memorandum UCB/ERL M01/11*. University of California, Berkeley, 2001.

[10] W.Sung and S.Ha. "Hardware Software Cosimulation Backplane with Automatic Interface Generation". *Proceedings of the ASPDAC'98*, 1998.

[11] J.S.Dahmann. "High Level Architecture for Simulation". *Proc... of the 1st International Workshop on Distributed Interactive Simulation and Real-Time Applications*, 1997.

[12] S.Strassburger, T.Schulze, U.Klein, and J.Henriksen. "Internet-based Simulation Using Off-the-shelf Simulation Tools and HLA". *Proceedings of the Winter Simulation Conference*. Washington, USA, December, 1998.

[13] F.R.Wagner, M.Oyamada, L.Carro, and M.Kreutz. "Object-Oriented Modeling and Co-simulation of Embedded Electronic Systems". In: L.M.Silveira, S.Devadas, and R.Reis (eds.), *VLSI: Systems on a Chip*. Kluwer Academic Publishers, 2000.

[14] L.Carro, M.Kreutz, F.R.Wagner, and M.Oyamada. "A Design Methodology for Embedded Systems based on Multiple Processors". In: B.Kleinjohann (ed.), Architecture and Design of Distributed Embedded Systems. Kluwer Academic Publishers, 2001. (Proceedings of DIPES'2000)

[15] M.Oyamada and F.R.Wagner. "Co-simulation of Embedded Electronic Systems". In: *Proceedings of 12th European Simulation Symposium*. Hamburg, Germany, October 2000.

[16] D.Wildt and F.R.Wagner. "Adapting Simulation Environments to HLA: Discussion and Case Study". In: *Proceedings of the European Simulation Multiconference*, Prague, Czech Republic, June 2001.

[17] M.Elnozahy et al. "A Survey of Rollback-Recovery Protocols in Message-Passing Systems". *Technical Report CMUCS99148, Department of Computer Science*. Carnegie Mellon University, September 1999.

[18] R.M.Fujimoto. "Parallel Discrete Event Simulation". In: *Communications of the ACM*, Vol. 33, n. 10, October 1990.