# MCI – MULTILANGUAGE DISTRIBUTED CO-SIMULATION TOOL

F.Hessel, P.Le Marrec, C.A.Valderrama, M.Romdhani, A.A.Jerraya

System-Level Synthesis Group
TIMA Laboratory
Grenoble, France

*Nowadays the design of complex systems requires the cooperation of several teams belonging to different cultures and using different languages. It is necessary to dispose of new design and verification methods to handle multilanguage approaches. This paper presents a multilanguage co-simulation tool that allows co-simulation of multilanguage specifications for complex systems. The main idea of our approach is to allow validation of the functional completeness of the system at a behavioral level. MCI starts with a configuration file that describes the interconnection between modules written in different languages. It generates automatically a software co-simulation bus and the interfaces required to connect the different simulators in a distributed way. The proposed tool is used to assist the design of an adaptive speed control system that was described in three different languages (VHDL, SDL and MatLab).*

## 1. Introduction

The design of complex electronic systems such as those embedded in a modern car or aircraft may require the cooperation of several design groups belonging to different companies with different cultures. Of course, different parts of the design may require different concepts such as hardware/software, control/data, continuous/discrete, synchronous/asynchronous and etc.

The specification of each part of the system is usually made by cooperating groups using different languages. Of course, a natural question arouses. Is it possible to find a specification language that may accommodate all kinds of applications at different abstraction levels?

Experiments with system specification language [2,3,6] show that there is not a unique universal specification language to support the whole life cycle (specification,

F. J. Rammig (ed.), *Distributed and Parallel Embedded Systems*

design and implementation) for all kind of applications. New specification and design methods are necessary to handle these cases where different languages and methods need to be used within the same design. These are multilanguage specification design and verification tools.

The concept of multilanguage specification aims at coordinating the different cultures through the unification of the languages, formalisms and notations. Nowadays, the use of more than one language correspond to current need in embedded system design. In most system houses, software design groups are separated from hardware design groups. Multilanguage specification is driven by needs of modular and evolutive design because of the increasing complexity of designs. Modularity helps in mastering this complexity, promotes design re-use and more generally encourages concurrent engineering development.

The basic idea behind multilanguage design is to allow modular specification. The communication scheme is described separately from the rest of the system. This allows hiding the details related to the inter-processes communication. All accesses to the interface of a communication unit is made through specific communication procedures that fix the protocol of exchanging parameters between the subsystems. Specific communication units may be necessary when complex protocols are used for exchanging data.

The goal of this work is to show a multilanguage distributed co-simulation environment, called MCI (**M**ultilanguage **C**osimulation **I**nterface). The objective of the environment is to run concurrently simulators for the simulation of the different subsystems. The inter-simulators communication is automatically generated starting from of a configuration model. This approach is based on a multilanguage co-simulation backplane.

The next section introduces the state of the art. In section 3, we give a brief overview of automatic generation of co-simulation interfaces. The section 4 describes the MCI tool. The above mentioned concepts will be clarified by a real example in section 5. And finally, in section 6, we conclude with perspectives and directions of future work.

## 2. Multilanguage Co-Simulation

Multilanguage co-simulation is a key issue of codesign methodology. Through co-simulation it is possible to validate the functional correctness of the system specification. Co-simulation can also be used to gain information about an embedded system before the prototype is actually build [7,9]. Furthermore, co-simulation allows designers to perform experiments that could be impractical in a prototyping environment.

As codesign proceeds, the level of co-simulation evolves from the behavioral level to the implementation level in which timed co-simulation would be necessary to validate timing requirements. In the behavioral level, the functional completeness of an implementation can be effectively verified with untimed simulation [10].

Co-simulation aims at using several simulators concurrently. The key issue for the definition of a co-simulation environment is the communication between the different simulators.

Several researchers have described frameworks and methodologies for the co-simulation of heterogeneous system specifications, typically C-VHDL co-simulation model [1,4,8,13]. Few recent works addressed the problem of multilanguage co-simulation [5].

Coware [15] and Siera [16] are typical environments supporting such codesign scheme. They start from a mixed description given in VHDL or VERILOG for hardware and C for software. All of them allow for co-simulation. These approaches restrict the specification for hardware and software by using specific languages.

A general multilanguage approach requires the combination of any kind of languages for the specification of modules. Only few systems in the literature allow such codesign model. These include the PTOLEMY and the RAPID systems [5] and the work described in [6]. As in the case of heterogeneous specification, the problem of interfacing and multilanguage validation need to be solved.

The main problems of multilanguage specification are:

(1) How to decide the most appropriated paradigm for each part of global specification,

(2) How to compose these subsystem specifications in a unified format suitable for codesign,

(3) How to verify if the subsystem specifications and the unified format correspond to the requirements and constraints of global specification.

The co-simulation approach (Figure 2.1) consists in interconnecting the simulation environments associated to each of the partial specifications. Co-simulation is an engineering solution to multilanguage validation that performs just a shallow integration of partial specifications [4].

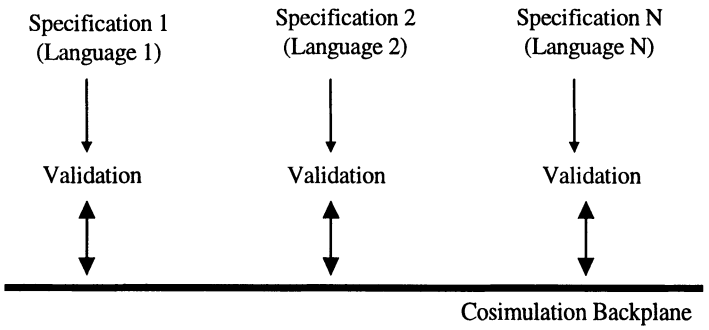



Figure 2.1: Validation through co-simulation

## 3. Automatic Generation of Co-simulation Interfaces

When dealing with co-simulation, the most tedious error-prone procedure is the generation of the link between different simulation environments. In order to overcome this problem automatic co-simulation interface generation tools are needed. This kind of tools take as input a user defined configuration file, which specifies the desired configuration characteristics (I/O interface, interconnection modules debugging tools) and produces a ready-to-use multilanguage co-simulation interface [13].

Figure 3.1 shows a generic inter-simulator interface generation scheme.

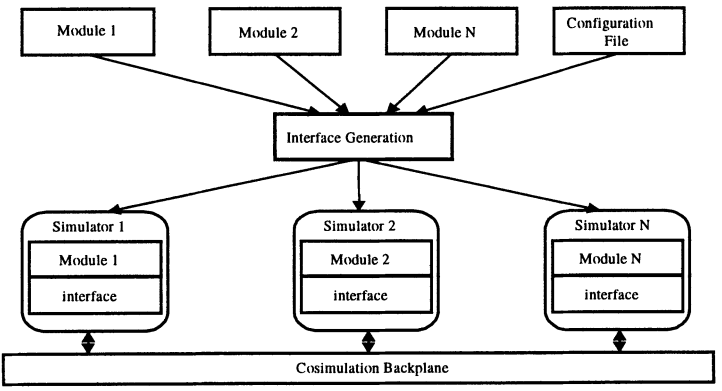



Figure 3.1: Inter-simulator interface generation

Based on the information provided by the configuration file, the interface generation process produces automatically the interface between the modules and the co-simulation bus.

The configuration file specifies the inter-module interactions. These interactions may be specified at different levels ranging from the implementation level where communication is performed through wires to the application level where communication is performed through high level primitives independent from the implementation. Figure 3.2 shows three levels of inter-module communication [14].

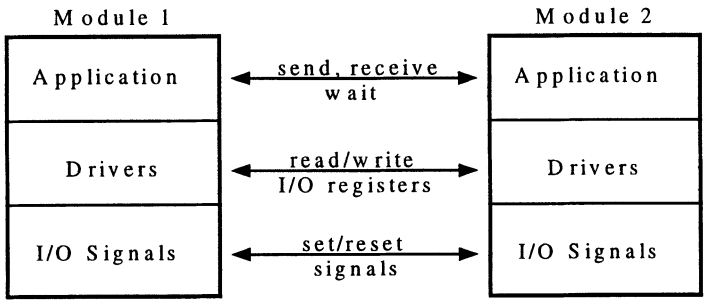



Figure 3.2: Abstraction levels of inter-module communication

## 4. MCI: A Multilanguage Co-simulation Interface

The figure 4.1 shows the global view of MCI. The proposed approach starts from a multilanguage specification of the system and a user defined configuration file. This file specifies the desired configuration characteristics, as a list of the subsystem specifications, a list of interconnections between subsystems, a list of properties concerning the machine that will execute each subsystem specification and the language of each subsystem was described. The simulator names are included into the subsystem characteristics for to give information about what simulator to use for the model debugging tools during the co-simulation process.

The MCI tool (figure 4.1) takes the configuration file to produces a ready to use multilanguage co-simulation environment. The environment is composed of a cosimulation backplane and a simulation interface for each simulator that will be run. The co-simulation backplane is based on a communication network protocol that is used as a co-simulation bus. All simulators communicate with one another by means of this cosimulation bus. The cosimulation bus is in charge of transferring data between the different simulators via the router process.

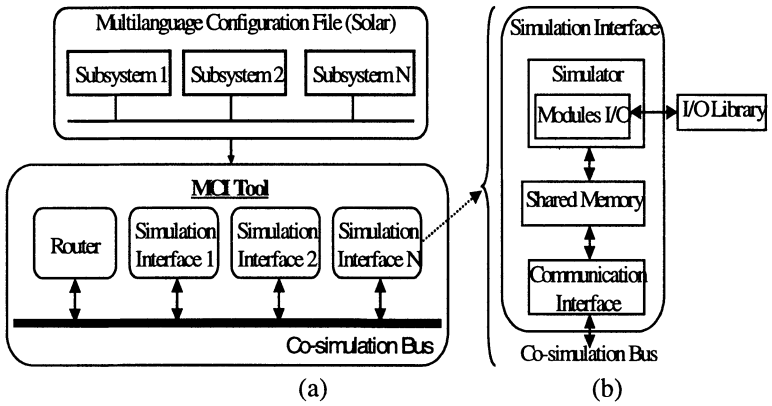



Figure 4.1: MCI functioning model

The cosimulation backplane is implemented onto UNIX sockets. MCI reads on the configuration file the computer name that will execute the simulation interface, the type of simulator that is required to run the subsystem simulation, and performs all initializations to run each simulation interface process (figure 4.1b). Then MCI loads the router process that establishes the scheme of the data interconnection and manages the computed data transferring between different simulators. Then the simulation interfaces are automatically loaded by remote connections with the local host name where MCI is running as parameter. This allows to connect each communication interface to the router by means of sockets. When the simulation interface is in the running mode, it loads the associated simulator.

Before starting the simulation processes, each subsystem specification is linked to a specific library that allows to send/receive data to/from the external world. The computed data that are sent to the external world are stored in a shared memory. This shared memory is dynamically created by the first external port computed by the

simulator. After the creation of the shared memory all external ports write the values in this memory and change an update flag (data modified or data not modified). The communication interfaces extract out of the shared memory the modified data only specific flag is setting has been modified. Then it sends the new data to the router via the connected socket. The router receives this modified data and, respecting the data interconnection schema, sends the new data to the right communication interface. The communication interface receives the new data, stores it into the shared memory of the simulation interface and changes the flag of the data as modified. Finally the simulator, via the external input, receives the new data and changes the flag for not modified data.

Once the system is loaded, the designer may interact with any simulator, setting breakpoints, examining registers, change parameters, etc. This can be done during the simulation, so it can be easy to parameterize a complete system. In its current version MCI makes use of four simulators (VHDL, C, SDL, and Matlab).

Figure 4.2 shows a configuration file. We used a SOLAR intermediate form [11], an EDIF like format for the specification of configuration files. In this configuration, we can view two modules: send and receive. The send module is a SDL system and the receive module is a VHDL system. The SDL system sends a data to the VHDL system. The VHDL system receives this data, decodes it and sends it to an ack signal if the data is right or a nack signal if the data is wrong.

The main difficulty that MCI should overcome is the lack of openness of the co-simulation environments and the heterogeneity of the data types they use. However, MCI aims to ensure a distributed co-simulation model in such way that none of the tools should master the global simulation. We used co-simulation without a synchronization of the simulators (untimed co-simulation) [10], each simulator running under of the local clock. In this case the exchange of data between simulators are controlled by events. Only the order of operations is checked, the timing aspects are ignored. This model of co-simulation schedules and verifies the behavior of the system, and establishes the coherence of the interconnection between the sub-systems, showing a global flashover of the system functionality. The fidelity of a simulation, as the contrary notion to abstractness, is not only defined by its completeness in regard to modeling of the functionality, but as well by the precision used to model the value/time relation of its signals and variables [12]. In untimed co-simulation the time proceeds in both simulators independently and thus the value/time relation is merely modeled correctly in intervals of the appropriate time scale [10].

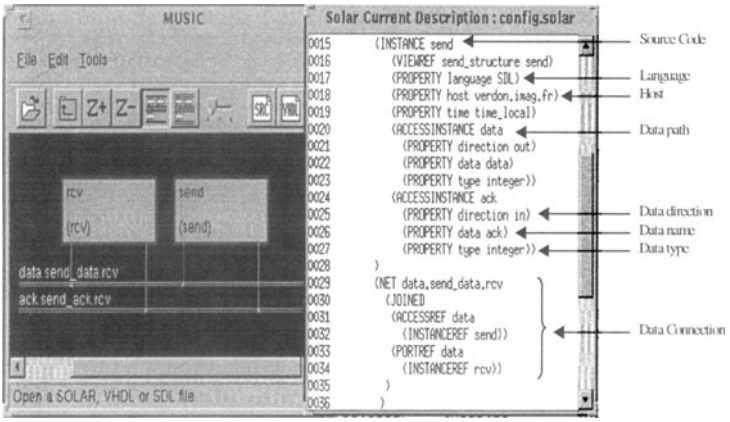



Figure 4.2: Solar Configuration File

## 5. Application Example

This section describes an experiment performed with the MCI tool. In this example, we modeled an Adaptive Motor Speed Control system. The system manages the motion of the set of motors in order to adjust their actual trajectory. For example, the control in a 2-D space needs a motor for each axis (X and Y) and an associated control system for a continuous movement. The system can handle up to 18 motors. However, in order to simplify the presentation, we will explain a single motor application.

As figure 5.1 shows, the Adaptive Motor Speed Control is composed of two sub-systems, Distribution and Speed Control. The Distribution sub-system provides the travelling distance to the Speed Control. The Speed Control sub-system computes the number of speed control pulses, using the specified final position and the current state of a motor, and translates them into motor control signals. This computation uses a regulation algorithm.

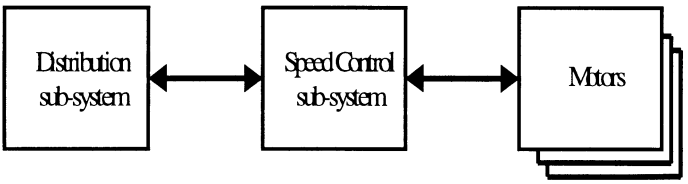



Figure 5.1: Adaptive Motor Speed Control system

The system is modeled into three sub-systems. The Distribution sub-system is modeled in SDL; the Speed Control sub-system is modeled in VHDL and the Motors are modeled in Matlab.

The entire system was simulated at the behavioral level using the MCI tool. For this purpose, we used a configuration file that describes the interfaces of each sub-

system and the connections between them (Figure 5.2), a VHDL simulator for the Speed Control sub-system, a SDL simulator for the distribution sub-system and a Matlab simulator for the Motors. The communication between the sub-systems and the motors is made by means of a co-simulation bus and the simulation interface processes, described in previous sections. The results of co-simulation are the waveforms generated by the VHDL simulator, the SDL execution traces and a graphical generated by the Matlab simulator that shows the evolution of the motors.

During the debug phase, we can execute the VHDL and SDL simulators step by step in order to follow the execution. The figure 5.3 is a screen capture showing a simple co-simulation session for the Adaptive Control Speed system consisting of the execution of the SDL simulator (top-left side), VHDL simulator (top-right side) and Matlab simulator (bottom-left side and bottom-right side). The Distribution sub-system was simulated in a Sun Sparc station, the speed control sub-system was simulated in a Sun Sparc-5 station and the motors were simulated in a Sun Ultra-Sparc station.

After co-simulation, the next steps of the codesign (partitioning, communication synthesis) can be made by means of the existing tools, as MUSIC framework, and a second co-simulation can be performed to verify the time constraints.

The main advantages of this approach are the flexibility and modularity of specifications, as well as the verification of the functional completeness. The execution of appropriates symbolic or high level debuggers can be used to debug each sub-system specification. The limitations of this approach are related to the real-time applications, when the timed co-simulation is necessary. In this case, the capability of synchronizing the execution of simulators must be added in the MCI tool.

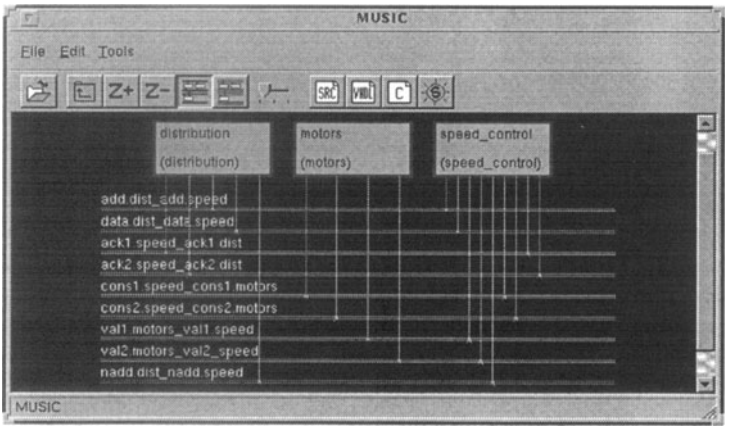
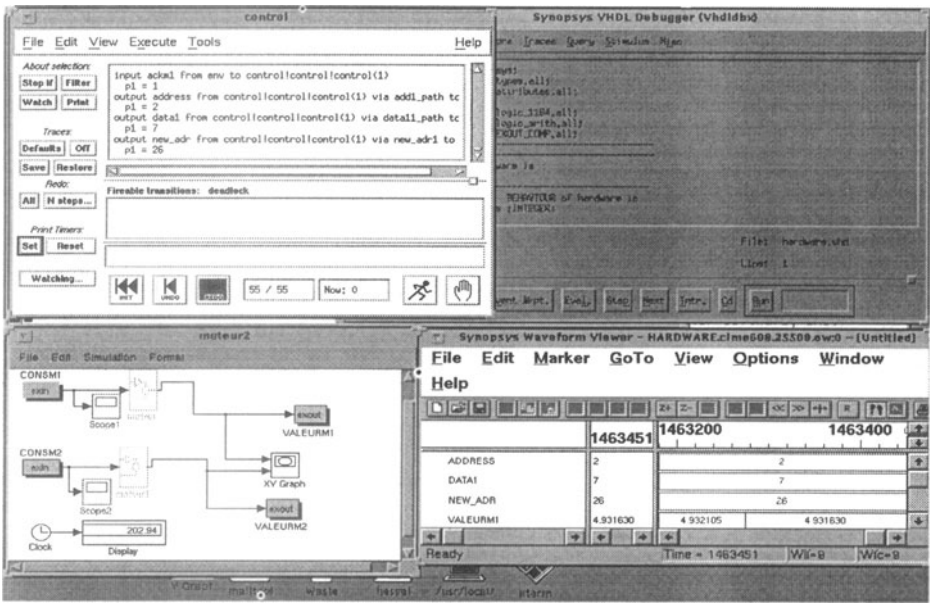



Figure 5.2: Solar configuration file




5.3: Adaptive Speed Control System co-simulation

## 6. Conclusions and Future work

This work presented a multilanguage distributed co-simulation tool, called MCI. It allows to co-simulating multilanguage specifications, starting with a Solar configuration file. The current version of MCI allows parallel-execution of SDL-C-Matlab-VHDL simulators and it is possible to run on a single CPU or across networks on different platforms without any additional implementation effort. Other languages can be plugged in MCI.

## References

[1] D. Hermann, J. Henkel, and R. Ernst, "An approach to the adaptation of estimated cost parameters in the Cosyma system", in *Proc. Third Int'l Workshop on Hardware/Software codesign*, pp. 100-107, IEEE CS Press, 1994.

[2] W. Wolf, "Hardware-software co-design of embedded systems", *Proceedings of the IEEE*, vol. 82, pp. 967-989, July 1994.

[3] L. Lavagno, A. Sangiovanni-Vicentelli, and H. Hsieh, "Embedded system co-design: Synthesis and verification", in *Hardware/Software Co-Design* (G. DeMicheli and M. Sami, eds.), pp. 213-242, Kluwer, 1996.

[4] A. A. Jerraya, M. Romdhani, C. A. Valderrama, Ph. Le Marrec, F. Hessel, G. Marchioro and J. M. Daveau, "Languages for system-level specification and design", in *Hardware/Software Co-design: Principles and Practice* (J. Staunstrup and W. Wolf eds.), pp. 307-357, Kluwer, 1997.

[5] N. Rethman and P. Wilsey, "RAPID: A tool for hardware/software tradeoff analysis", in *Proc. IFIP Conf. Hardware Description Languages (CHDL)*, Elsevier Science, April 1993.

[6] M. Romdhani, R. Hautbois, A. Jeffroy, P. de Chazelles and A. A. Jerraya, "Evaluation and composition of specification languages, an industrial point of view", in *Proc. IFIP Conf. Hardware Description Languages (CHDL)*, pp. 519-523, September 1995.

[7] K. Hines, and G. Borriello, "Dynamic communication models in embedded system co-simulation", in *Proc. of Design Automation Conference*, pp. 395-400, 1997.

[8] S. Yoo and K. Choi, "Optimistic Timed HW-SW Co-simulation", in *Proc. of APCHDL'97*, pp. 39-42, 1997.

[9] W. Sung, M. Oh and S. Ha, "Interface design of VHDL simulation for hardware-software co-simulation", in *Proc. of APCHDL'97*, pp. 43-49, 1997.

[10] K. Hagen and H. Meyr, "Timed and Untimed hardware/software co-simulation: application and efficient implementation", in *CODES'93*, 1993.

[11] A. A. Jerraya and K. O'Brien, "Solar: An intermediate format for system-level design and specification", in *IFIP Inter. Workshop on Hardware/Software codesign*, Grassau, Germany, May 1992.

[12] K. Hagen and H. Meyer, "Modeling at different levels of abstraction within an ASIC design project", in *Modeling and Simulation (ESM)*, pp. 95-99, June 1993.

[13] C.A. Valderrama, A. Changuel, P.V. Raghavan, M. Adib, T. Ben Ismail and A.A. Jerraya, "A unified model for co-simulation and co-synthesis of mixed hardware/software systems", in *Proc. European Design and Test Conference (EDAC-EDTC-EuroASIC)*, IEEE CS Press, March 1995.

[14] D.E. Thomas, J.K. Adams and H. Schmit, "Model and methodology for hardware-software codesign", in *IEEE Design and Test of Computers*, 10(3):6-15, September 1993.

[15] H. DeMan, I. Bolsens, B. Lin, K. Van-Rompaey, S. Vercauteren, D. Verkest, "Codesign for DSP systems", in *NATO ASI Hardware/Software Codesign*, Tremezzo, June 1995.

[16] M. Srivastava, R. Brodersen, "Sierra: A unified framework for rapid-prototyping of system-level hardware and software", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 676-693, June 1995.