MULTILANGUAGE DESIGN OF HETEROGENEOUS SYSTEMS

P. COSTE, F. HESSEL, PH. LE MARREC, Z. SUGAR, M. ROMDHANI, R. SUESCUN, N. ZERGAINOH, A.A. JERRAYA

TIMA Laboratory, 46 avenue Félix Viallet, 38000 Grenoble France

Pascal.Coste@imag.fr

ABSTRACT

Multilanguage solutions are required for the design of heterogeneous systems where different parts belong to different application classes e.g. control/data or continuous/discrete. The main problem that needs to be solved when dealing with multilanguage design is the refinement of communication between heterogeneous subsystems. This paper discusses the basic concepts of multilanguage design and introduces MUSIC a Multilanguage design approach. The paper also shows the application of this approach in the case of a mechatronic system.

Keywords

Multilanguage, Codesign, Heterogeneous Systems, Cosimulation.

1. INTRODUCTION

Experiments with system specification languages [21] show that there is not a unique universal specification language to support the whole life cycle (specification, design, implementation) for all kinds of applications. A plethora of specification languages exists. Each claims superiority but excels only within a restricted application domain. The design of a complex system may require the cooperation of several teams belonging to different cultures and using different languages. New specification and design methods are needed to handle these cases where different languages and methods need to be used within the same design. These are multilanguage specification, design and verification methods.

A typical application domain for multilanguage design is Mechatronics. In fact, the use of electronics within cars, for example, is becoming more and more important. It is expected that the electronic parts will constitute more than 20% of the price of future cars [19]. The joint design of various mechanical parts with electronic parts and specially micro-controllers is a very important area for the Mechatronics design. In traditional design approaches, the different parts are designed by separate groups and the integration of the overall system is made at the final stage. This scheme may induce extra delays and costs because of interfacing problems. The necessity for more efficient design approaches allowing for a joint design of different parts is evident. Multilanguage design constitutes an important step in this direction. It gives the designer the ability to validate the whole system's behavior before the implementation of any of its parts. Multilanguage system design offers many advantages including efficient design flow and shorter time to market. The key idea is to allow for early validation of the overall system through co-simulation.

This paper introduces a complete design methodology and a set of tools used for Mechatronics design. The next two sections discuss system level modeling strategies, multilanguage design concepts and previous work. Section 4 introduces MUSIC, a multilanguage design environment. Section 5 shows the feasibility of the multilanguage method through an example. Finally, section 6 provides our conclusions.

2. Multilanguage Design

Most of the existing system specification languages are based on a single paradigm. Each of these languages is more efficient for a given application domain. For instance some of these languages are more adapted to state-based specifications (SDL or Statecharts [5]), some others are more suited for data flow and continuous computation (LUSTRE, Matlab [15]), while many others are more suitable for algorithmic description (C, C++).

When a large system has to be designed by separate groups, they may have different cultures and expertise with different modeling styles. The specification of such large designs may lead each group to use a different language which is more suitable for the specification of the subsystem they are designing according to its application domain and to their culture.

The key issue for the design of such a system is the validation of the overall system and the synthesis of the interfaces between the different subsystems. Of course, most of these subsystems may include both software and hardware. In this case, a multilanguage design is needed. The key issues with such a scheme are validation and interfacing. The use of a multilanguage specification requires new validation techniques able to handle a multiparadigm model. Instead of simulation we will need cosimulation and instead of verification we will need coverification. Additionally, multilanguage specification brings about the issue of interfacing subsystems which are described in different languages. These interfaces need to be refined when the initial specification is mapped onto a prototype.

Figure 1 shows a generic flow for codesign starting from a multi-level specification. Each of the subsystems of the initial specification may need to be decomposed into hardware and software parts. The codesign process also needs to tackle the refinement of interfaces and communication between subsystems.

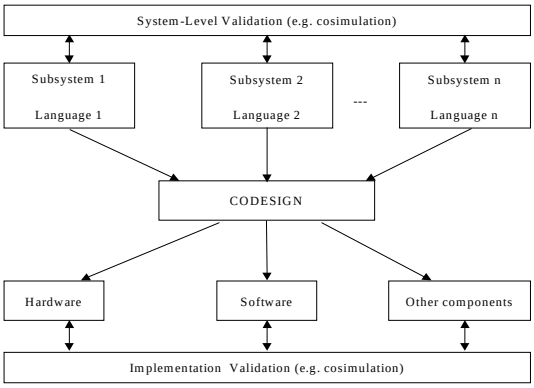

Figure 1 : Multilanguage Codesign Flow

3. Previous work

The success of cosimulation techniques [23] has made the Multilanguage approach very popular within the research community. There are two main approaches for multilanguage design : the compositional approach and the cosimulation-based approach.

The compositional approach aims at integrating the partial specification of sub-systems into a unified representation which is used for the verification and design of the global behavior. This allows to operate full coherence and consistency checking, to identify requirements for traceability links, and to facilitate the integration of new specification languages [3]. POLIS [1], [13], JavaTime [27], SPI [28],[6] and SpecC [4] introduce a compositional-based codesign approach. Both Polis and SPI use an internal model for composition. Both JavaTime and SpecC use another specification language (respectively Java and SpecC) for composition.

The cosimulation-based approach consists in interconnecting the design environments associated to each of the partial specifications. Compared with the deep specification integration accomplished by the compositional approaches, cosimulation is an engineering solution to multilanguage design that performs just a shallow integration of the partial specifications. However it allows for modular design.

In this approach the different modules described in different languages may be designed using different environments. During the design flow, Cosimulation is used for the validation of the overall system through the codesign steps. The main Codesign steps in this case are interlanguage communication synthesis and subsystem refinements [16]. Most of existing tools in this area provides few refinements and they start the design at quite a low level e.g. RTL for Hardware and C language for Software. Coware [22], Seamless [9] and [24] are typical environments supporting such a codesign scheme. They start from a mixed description given in VHDL or VERILOG for hardware and C for software. All of them allow for cosimulation. However, only Coware allows for interface synthesis [2]. Only few systems in the literature tried to tackle the Cosimulation-based multilanguage codesign approach at a higher level. These are RAPID [20], Ptolemy [14] and SEA [10]. Unfortunately most of these systems provide only Cosimulation facilities [20],[14].

The work presented in this paper overcomes these limitations. It allows to start the multilanguage approach at the system level providing both system-level refinement and high-level interfaces synthesis. The interlanguage communication synthesis is based on the concept of high-level communication synthesis similar to the concepts used in Codesign tools such as those used in LYCOS [11], CHINOOK [25] and the work reported in [17].

4. MUSIC : a multilanguage approach for heterogeneous system design

This section introduces MUSIC, a complete design methodology and a set of tools for cosimulation-based multilanguage codesign. Although the design flow supports more languages covering several application domains we will restrict this presentation to the Mechatronics area. Figure 2 shows the flow used for the design of a Mechatronics system including hardware, software and mechanical parts. The design starts with an analysis of the system requirements and a high-level definition of the various functions of the system. The mechanical part is modeled in Matlab and the electronic part is modeled in SDL. At this stage, we obtain a multilanguage system-level model given in SDL-Matlab. At the top level, partitioning is done manually because we have no formal specification.

The design flow handles three abstraction levels : the system level, the system architecture level and the RTL or cycle accurate level.

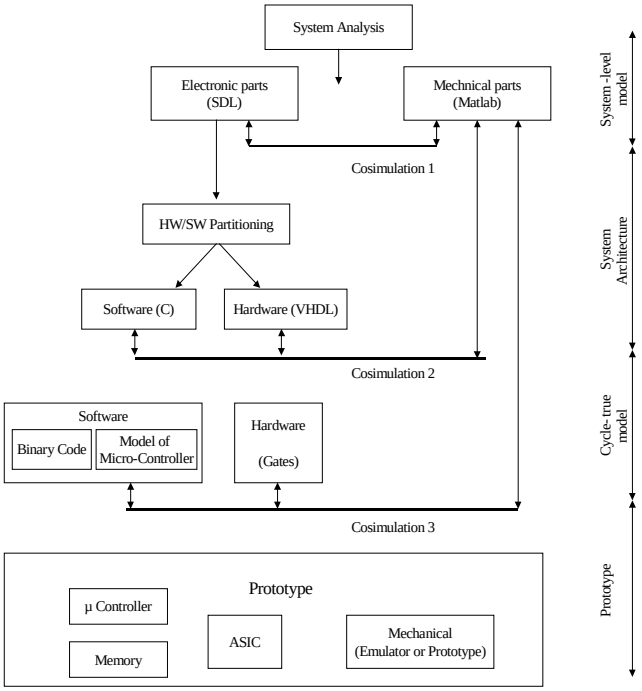

Figure 2 : Multilanguage design flow for Mechatronics

At the system level, communication is described at the application level. The validation of the overall system may be done using system level co-simulation of SDL-Matlab models. This model may be used as a mock-up of the system in order to fix the final specification. At this level the SDL-Matlab communication needs to be refined. Additionally the electronic module needs to be

partitioned into hardware and software. This may be performed automatically using the COSMOS tools [26]. This step produces a mixed hardware/software model of the electronic part.

After the partitioning step and the interlanguage communication synthesis, we obtain a system-architecture Model. Hardware is modeled as behavioral VHDL, software is modeled as C-programs, Hardware/Software communication is performed through generic wires and the mechanical part remains as a Matlab model. At this level the interlanguage communication (C, VHDL, Matlab) is described at a level where all the protocols are explicit in the model. Cosimulation may be achieved in order to validate the partitioning and the communication protocols. Some timing verifications may be achieved at this level.

During the next step the design is refined to the cycle-true level. Hardware is refined as an RTL model. The software is executed on a model of the final processor and the mechanical part may be kept in Matlab. At this level all interfaces are refined to the physical level. HW/SW interfaces should include drivers in the software part and some hardware adapters to link the processor to the application. The model of the processor may be an implementation model (e.g. a gate model or a synthesizable VHDL) or a high-level model (e.g. a C program). At this level, cosimulation may be used to check timing at the clock-cycle level [18].

The final step in this multi-language design flow is prototyping. At this level, a prototype of the electronic system needs to be built. The mechanical part may be emulated [12],[8].

5. Application Example

This section shows the results of our multilanguage codesign flow in the case of a multilanguage model. The example is a large application, a robot arm controller. In the example we will illustrate the overall design flow of MUSIC, from a system-level specification given in SDL and Matlab to an implementation.

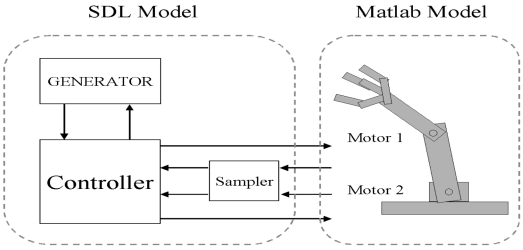

Figure 3 : Motors controller application

5.1 The application

The application is a robot arm controller. The system can be divided into two parts, the robot arm's motors and the controller. SDL is used to model the controller and Matlab is used to model motors physical behavior.

The robot arm controller can adjust the position and speed parameters of eighteen motors belonging to a robot arm. In this paper we will restrict the model to two motors only.

Four signals are exchanged between the Matlab model and the SDL model, two for each motor. The first signal controls the motor and the second one provides the current position. The basic bloc diagram of the system is shown in Figure 3. The *GENERATOR* bloc computes a trajectory for all motors of the robot arm. The *Sampler* bloc receives Motors' positions and transmits them to the Controller. The *Controller* bloc is in charge of giving orders to Motors only when it is possible.

5.2 The design process

- The SDL System-Level Specification

The SDL specification uses three mains blocs that include six SDL processes, a *generator*, a *distributor*, one *controller* per motor, and one *sampler* per motor.

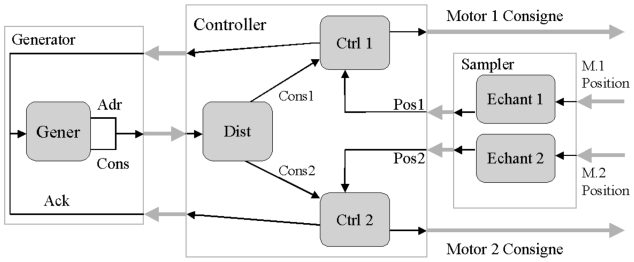

Figure 4 : SDL controller specification

Figure 4 shows the processes' structure of the controller system. The *Gener* process produces two signals, an order which is called *Cons* and an address, which is an identifier of a motor. The *Dist* forwards orders to the called motor controller by the address. A *Ctrl* process gives the order to the motor and scans its position. When a motor is about to accomplish its order, the *Ctrl* process informs the *Gener* process that it can produce a new destination. Because a position signal is a "continuous signal", each *Echant* process represents a digital acquisition. In SDL all processes are concurrent and communicate through asynchronous queues.

- The Matlab Specification

In this case a Matlab model is used to specify the mechanical part. This model acts as a testbench for the SDL model during all the codesign steps.

- The Overall System

The overall system's interconnections, including SDL and Matlab instances, are described using a configuration file that gives the interconnection between the blocs. Figure 5 shows the configuration of the SDL-Matlab Models. It is made of two blocs interconnected through abstract channels. We use the SOLAR format [7] for the description of the configuration file. In SOLAR, a system is modeled as a set of design units interconnected through high-level channels or physical signals. MUSIC uses this configuration file in order to produce a cosimulation run where the different blocs will be executed using the corresponding Simulators.

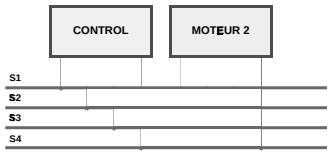

Figure 5 : SDL-Matlab blocs configuration

The same file will be used to refine the communication protocols between SDL and Matlab. In fact, the global configuration of the system is a kind of "system-level netlist" that specifies the interconnection between different subsystems. Since the different languages are based on different concepts for data exchange, the

interpretation of the link between heterogeneous subsystems will need a specific communication synthesis step.

- System-level Validation (SDL-Matlab Cosimulation)

In order to validate the system at this stage of the design flow, an SDL-Matlab cosimulation is performed. During Cosimulation, debuggers and GUI can be launched to control the simulation and analyze all parts of the system. Figure 6 shows a running cosimulation of the SDL-Matlab model. The left part shows the SDL simulator window and the right part shows the Matlab/Simulink windows.

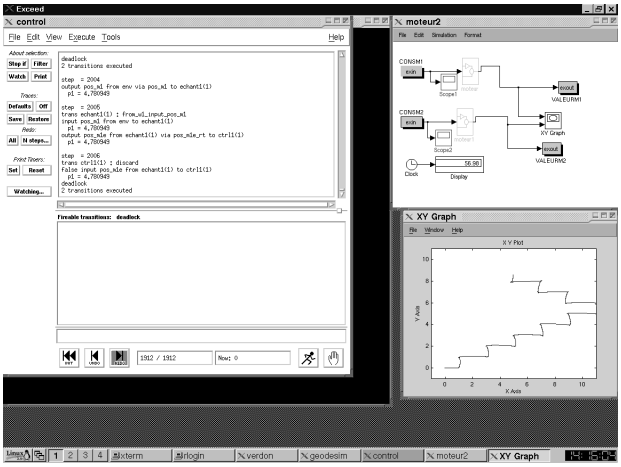

Figure 6 : controller system-level running cosimulation

- Partitioning and Communication Synthesis

The system-level specification contains two blocs, one Matlab bloc, the mechanical part, and one SDL bloc, the controller. As described in section 4, several steps are used to refine the controller specification down to a prototype.

The first step is the translation of SDL to the internal model of MUSIC (SOLAR) in order to apply system-refinement transformations. The specification obtained is presented in Figure 7, this model is made up of seven blocs. Six come from the SDL model and the seventh corresponds to the Matlab bloc.

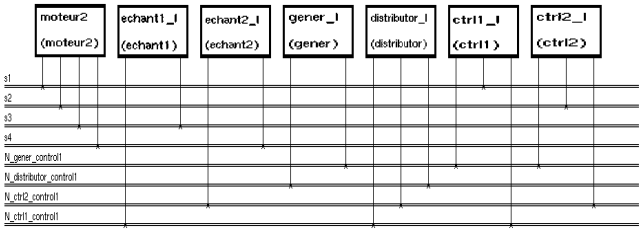

Figure 7 : SOLAR specification after SDL translation

The next step consists in the communication synthesis. The MUSIC tool assists the user in mapping abstract communication channels on explicit communication protocols. In this case we have two kinds of communication channels. First of all we have channels that correspond to the communication between SDL processes : The four bottom channels of Figure 7 link electronic modules. In this case we decided to implement them as “rendez-vous” protocols.

Then we have the four top channels of Figure 7 that correspond to communications between the mechanical part (Matlab) and the electronic system. They need specific transceivers able to connect heterogeneous blocs. The result of the communication synthesis is shown in Figure 8. Four extra communication controllers were inserted automatically during the communication synthesis. They correspond to adapters that implement the communication between mechanical parts and electronic parts. For both kinds of communication syntheses we use the library-based protocol mapping method described in [26].

The next step is Hw/Sw partitioning. MUSIC allows to decompose the system into three kinds of blocs : Hardware blocs, Software blocs and IP blocs. The IP bloc corresponds to parts of the system that will be handled as black boxes. As shown in Figure 9 the initial specification was reorganized into a Hardware module (in black), an IP module (in white) and two Software modules (in gray).

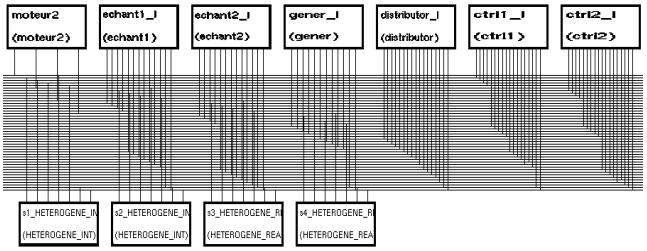

Figure 8 : Explicit Communication Specification

- Hardware/Software Synthesis

The next step is code generation. MUSIC generates VHDL-RTL code for hardware blocs and C code for software blocs. This new model can be used for low-level cosimulation or for prototyping.

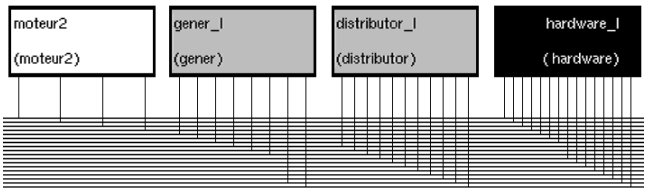

Figure 9 : C/VHDL generated system

- Architectural Validation (C-VHDL-Matlab cosimulation)

The architectural controller specification is composed of VHDL, C and Matlab blocs. MUSIC is used again to cosimulate this model. VHDL blocs are executed on Synopsys VSS simulator, C blocs are compiled and executed on a workstation and Matlab executes the models of the motors. Figure 10 shows a screen running C, VHDL and Matlab. This simulation allows to check that the produced system has the same behavior as the initial specification. According to the design flow in Figure 2 another simulation step may be performed to check the behavior of the system at the clock-cycle level.

- System Implementation

The final step is implementing the system into a Hardware / Software architecture. This part may be done using classical methods [18].

6. Conclusion

This paper discussed multilanguage design and introduced a multilanguage design flow called MUSIC. The application of MUSIC on mechatronic design was also discussed.

The main contribution of this work is to start from a multilanguage specification made of SDL and Matlab for the design of a mechanical system. The design flow combines SDL-Matlab Cosimulation, Interlanguage Communication Synthesis and classical Hardware/Software Codesign flow.

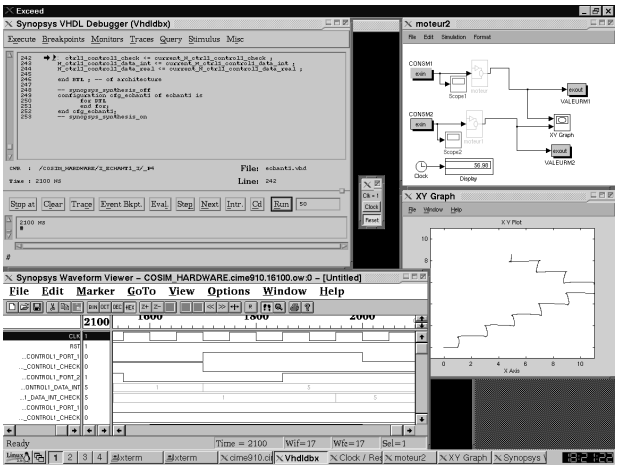

Figure 10 : architectural specification running cosimulation

Acknowledgement

This work was supported by France-Telecom/CNET, ESPRIT-OMI program under project CODAC 24139, SGS-Thomson, Aerospatiale, PSA, ESPRIT program under project COMITY 23015 and MEDEA program under projects SMT AT-403 and CIME 452.

REFERENCES

[1] **Felice Balarin et al**. 1997 (May). *Hardware-Software Co-Design of Embedded Systems, The POLIS Approach.* Kluwer Academic Publishers.

[2] **I. Bolsens, B. Lin, K. Van Rompaey, S. Vercauteren and D. Verkest**. 1995 (June). Co-Design of Dsp Systems, from *In NATO ASI Hardware/Software Co-Design*. Tremezzo, Italy.

[3] **A. Davis**. 1995. *Software Requirements: Analysis and Specification.* Elsiever. NY.

[4] **D. Gajski, Rainer Domer and Jianwen Zhu**. 1998 (Aug.). IP-Centric Methodology and Design with the SpecC Language, from *Contribution to NATO-ASI Workshop on System Level Synthesis.* Il Ciocco, Barga, Italy.

[5] **D. Harel**. 1987. Statecharts : A Visual Formalism for Complex Systems. *Science of Computer Programming*, **8**, pages 231-274.

[6] **J. Henkel and R. Ernst**. 1995 (Sept.). A Path-Based Technique for Estimating Hardware Runtime in Hw/Sw- Cosynthesis, from *8th Intl. Symposium on System Synthesis (SSS).* Cannes, France. Pages 116-121.

[7] **A.A. Jerraya and K. O'Brien**. 1994. SOLAR: An Intermediate Format for System-level Modeling and Synthesis, from *Computer Aided Software/Hardware Engineering.*, ed. J.Rozenblit and K.Buchenrieder IEEE Press.

[8] **B. Kienhuis and E. Deprettere and K. Vissers and P. Von der Wolf**. 1998. The Construction of a Retargetable Simulator for an Architecture Template, from *6th International Workshop on Hardware/Software Co-Design (Codes/CASHE'98).* Pages 33-37.

[9] **R. Klein**. 1996. Miami: A Hardware Software Co-Simulation Environment, from *RSP'96.* IEEE CS Press. Pages 173-177.

[10] **B. Kleinjohann**. 1998 (sept.). Multilanguage Formalism. *MEDEA/ESPRIT conference HW/SW Codesign*, pages X.1.1-X.1.22.

[11] **P.V. Knudsen and J. Madsen**. 1998. Communication Estimation for Hardware/Software Codesign. *CODES98*, pages 55-59.

[12] **G. Koch and U. Kebshull and W. Rosenstiel**. 1994 (Sep.). A prototyping Environment for Hardware/Software Codesign in the COBRA Project, from *IWHSC*. Grenoble, France.

[13] **L. Lavagno, A. Sangiovani-Vincentelli and H. Hsieh**. 1996. Embedded System Codesign: Synthesis and Verification Kluwer Academic, Boston. Pages 213-242.

[14] **B. Lee and A. Lee**. 1998. Hierarchical Concurrent Finite State Machines in Ptolemy. *Proc. Of International Conference on Application of Concurrency to System Design.* Pages 34-40.

[15] **MathWorks**. 1998. *Matlab*. http://www.mathworks.com

[16] **V. Mooney and T. Sakamoto and G. De Micheli**. 1997. Run-Time Scheduler Syntheis for Hardwrae-Software Systems and Application to Robot Control Design, from *CHDL'97*. IEEE. Pages 95-99.

[17] **R.B. Ortega and G. Borriello**. 1998 (nov.). Communication Synthesis For Distributed Embedded Systems, from *ICCAD98*. San Jose, CA, USA.

[18] **P. Paulin**. 1995 (Sept.). High Level Synthesis and Codesign Methods: An Application to a Videophone Codec, from *Proc. of EuroDAC/EuroVHDL*. Brighton.

[19] **A. Rault, Y. Bezard, A. Coustre and T. Halconruy**. 1996. *Systems Integration in the Car Industry.* PSA, Peugeot-Citroen. 78140 Velizy Villacoublay, France.

[20] **N.L. Rethman and P.A. Wilsey**. 1993 (Apr.). RAPID: A Tool For Hardware/ Software Tradeoff Analysis, from *Proc. CHDL'93*. Elsevier Science, Otawa,Canada.

[21] **M. Romdhani et al**. 1995 (Sept.). Evaluation and Composition of Specification Languages, an Industrial Point of View, from *Proc. IFIP Conf. Hardware Description Languages (CHDL).* Pages 519-523.

[22] **K.Van Rompaey, D. Verkest, I. Bolsens and H. De Man**. 1996 (Sept.). Coware - a Design Environment for Heterogeneous Hardware/Software Systems, from *The European Design Automation Conference*. Geneve.

[23] **J.A. Rowson**. 1994 (june). Hardware/Software Co-Simulation, from *DAC94.* San Diego, CA, USA. Pages 439-440.

[24] **D.E. Thomas and J.K. Adams and H. Schmit**. 1993 (Sep.). A Model and Methodology for Hardware-Software Codesign. *IEEEDTC*, **10**(3), pages 16-28.

[25] **F. Vahid and L. Tauro**. 1997. An Object-Oriented Communication Library for Harware-Software Co-Design, from *5th International Workshop on Hardware/Software Co-Design (Codes/CASHE'97).* Pages 81-86.

[26] **C. Valderrama et al**. 1997. Hardware and Software Co-design : Principles and Practice KLUWER. Chap. COSMOS : A Transformational Codesign Tool for Multiprocessor Architectures, pages 307-357.

[27] **J.S. Young et al**. 1998. Design and Specification of Embedded Systems in Java Using Successive, Formal Refinement, from *DAC98.* Pages 70-75.

[28] **D. Ziegenbein, R. Ernst, K. Richter, J. Teich and L. Tiele**. 1998. Combining Multiple Models of Computation for Scheduling and Allocation, from *6th International Workshop on Hardware/Software Co-Design (Codes/CASHE'98).* Pages 42-46.