Object-Oriented Modeling and Co-Simulation of Embedded Systems

F.R.Wagner, M.Oyamada, L.Carro[1] and M.Kreutz
Universidade Federal do Rio Grande do Sul
Instituto de Informática, [1]Departamento de Engenharia Elétrica
Caixa Postal 15064, 91501-970 Porto Alegre, Brazil
e-mail {flavio,marcio,kreutz}@inf.ufrgs.br, [1]carro@iee.ufrgs.br

Key words: embedded systems, object-oriented modeling, co-simulation

Abstract: This paper presents the modeling and co-simulation capabilities of S^3E^2S, a design environment for electronic systems that can be built as a combination of analog and digital parts and software. S^3E^2S is based on a distributed, object-oriented system model, where abstract objects are initially used to express complex behavior and may be later refined into digital or analog hardware and software. Co-simulation of any heterogeneous model developed during a stepwise refinement process is supported. These capabilities are illustrated by the modeling of a crane and its embedded control.

1. INTRODUCTION

Embedded electronic systems contain a combination of software and hardware, both analog and digital. For simple systems, a single off-the-shelf processor might be sufficient, since there are various available architectures (microcontrollers, DSP, RISC processors) with different cost / performance ratios. However, more complex systems, which represent the current trend in the market, usually have more critical requirements, such as a combination of behaviors (scalar and DSP processing) and tight low-level characteristics (area, speed, power dissipation). Typical examples are voice-modems, cellular phones, and embedded controllers for automotive applications.

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: 10.1007/978-0-387-35498-9_57
L. M. Silveira et al. (eds.), *VLSI: Systems on a Chip*

The design of complex embedded systems should ideally proceed from an initial, abstract specification, going through a sequence of successive refinements, until a final detailed solution is achieved. Intermediate, heterogeneous descriptions generated during this stepwise refinement must be validated. Co-simulation is the most usual validation tool.

Most existing environments are oriented towards a particular application domain and have a fixed target architecture. They usually follow one of two possible approaches for system specification and co-simulation. In the first approach, system modeling is performed using a single language from a given domain, such as C, VHDL, or a higher-level language, and partitioning is performed in a later stage. In the second approach, the initial specification already considers a given partitioning, and each part is modeled using a language which is more appropriate for the corresponding domain.

The S^3E^2S (Specification, Simulation, and Synthesis of Embedded Electronic Systems) environment combines the advantages of both approaches. Complex systems may be modeled as combinations of objects specified at different domains – abstract object-oriented specification, digital hardware, analog hardware, and software – and at multiple abstraction levels. Co-simulation is performed by coupling different simulation engines, so that the validation of any heterogeneous model developed during a process of stepwise refinement is supported. Furthermore, the environment allows an easy exploration of the design space at a multi-processor level, selecting a combination of processors which best matches the design requirements. This paper covers the modeling and co-simulation features of S^3E^2S. The synthesis capabilities are discussed in detail elsewhere [1].

This paper is organized as follows. An overview of the design environment can be found in Section 2. The modeling and co-simulation capabilities of S^3E^2S are introduced in Section 3. Section 4 presents a case study that fully illustrates the application of the environment. Section 5 compares S^3E^2S to other current approaches for modeling and simulation of embedded systems. Section 6 draws conclusions and discusses future work.

2. AN OVERVIEW OF THE S^3E^2S DESIGN ENVIRONMENT

Figure 1 presents the design flow in S^3E^2S. The initial specification is a set of objects, described as C++ code. A set of support libraries allows not only the sequential modeling of digital hardware and software behavior, but also the modeling of linear or discrete time systems (to model analog domain behavior). If the user wishes, a set of high level VHDL constructs is also available, like a FIR filter, for example.

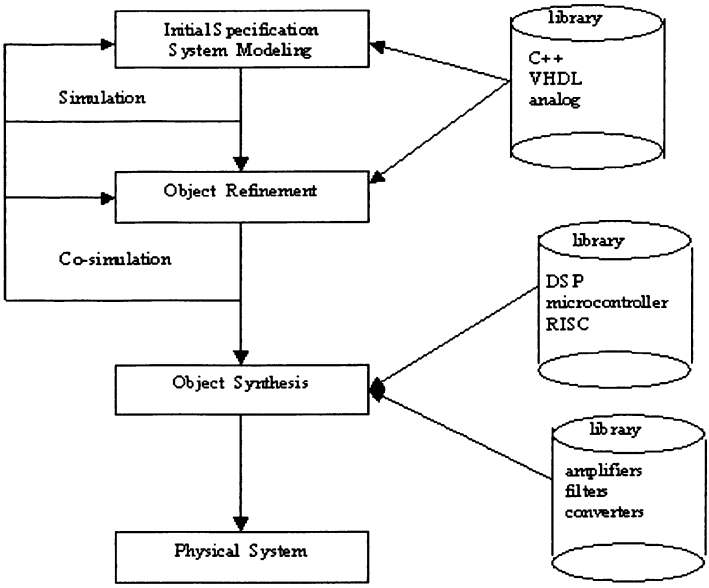

Figure 1. The S^3E^2S design flow

Once the initial specification is defined, it can be validated using the simulation engine behind S^3E^2S. Each object that used a pre-defined set of methods can then be translated into a VHDL component or a set of analog components. Simulation can be done mixing hardware descriptions in VHDL, software descriptions in C++ and analog descriptions developed as a set of analog components (such as operational amplifiers, converters and filters). Each intermediate description obtained during this stepwise refinement process can be co-simulated, whereby each object is simulated by a dedicated simulator for the corresponding abstraction domain.

There are 3 alternatives for the synthesis of the digital parts. The first one is based on the library of VHDL methods used during refinement. If taken from the S^3E^2S library, all methods are synthesizable. Another possibility is the use of the initial C++ specification alone. In this case, an internal compiler analyzes the code of each objet and maps it to an off-the-shelf processor [1]. A library of processors with different characteristics (microcontrollers, digital signal processors, RISC machines) is available. Mapping is done based on the characteristics of the object that best match processor characteristics. S^3E^2S performs the parsing of the C++ object description, obtaining a control-data flow structure. Each function of each object is checked, and the number of memory accesses, arithmetic operations and control instructions is verified. Based on the statistics of resource usage, on the organization of the object code and on the available time to execute a task, a processor best matching these criteria is chosen.

A third synthesis possibility is to let the system map some objects to off-the-shelf processors, leaving some other objects as synthesizable VHDL code. The resulting system is a processor with one or more dedicated ASICs. A typical situation could be a microcontroller and a dedicated filter.

Presently, in order to synthesize analog circuits, the designer must use only blocks that are available from the library, which can mapped to physical components.

3. MODELING AND CO-SIMULATION

S^3E^2S is built on top of SIMOO [2], an integrated environment for object-oriented modeling and simulation of discrete systems. SIMOO is composed of a class library and a model editor. The editor supports the description of the static and dynamic aspects of the model. The static structure is described graphically, while the dynamic structure is described either directly in C++ using the library resources or by means of a state diagram annotated with C++ code. The editor implements extensions to diagrams usually proposed by object-oriented design methodologies, in order to handle simulation-related aspects. From the model description, the editor automatically generates the necessary executable code.

A model is composed of *interface* and *autonomous elements*. Interface elements support tracking of the simulation execution, visualization of simulation results, interactive input of data, and dynamic modification of parameters during the experiments. Autonomous elements, on the other side, are used to model concrete entities. An autonomous element is an active object, i.e., an object with its own execution thread and a message queue. It may interact with other autonomous and interface elements only through messages. The model does not support shared variables, so that it may be also used in distributed environments.

Different objects of the same model may follow different paradigms [3]. A paradigm is defined as a combination of the following modeling approaches: event orientation or process orientation for the description of the object behavior, messages or ports for the communication between objects, and active or passive message handling. These approaches may be extended or specialized by inheritance.

In order to support a progressive replacement of SIMOO objects by VHDL entities or by analog components, and to model interactions with the real analog world, a co-simulation strategy combining SIMOO abstract models, VHDL descriptions, and analog models is needed.

The SIMOO simulation environment is coupled to the VSS simulator from Synopsys. Figure 2(a) shows two SIMOO objects (A and B) that

communicate with each other. Object B is to be refined into a VHDL entity. The co-simulation environment then automatically generates necessary interfaces in the SIMOO and VHDL domains. In the SIMOO domain, this includes an interface element. In the VHDL domain, on the other side, the interface specification of the entity corresponding to object B and an interface file written in C are generated, as shown in Figure 2(b). These interfaces in both domains are responsible for data exchange and for synchronizing the simulators.

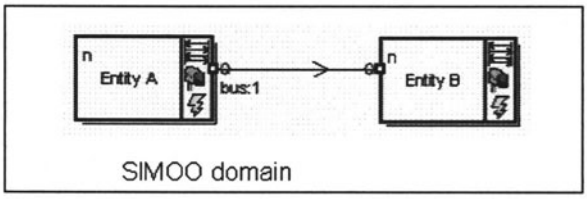

Figure 2(a). An initial specification in the SIMOO domain

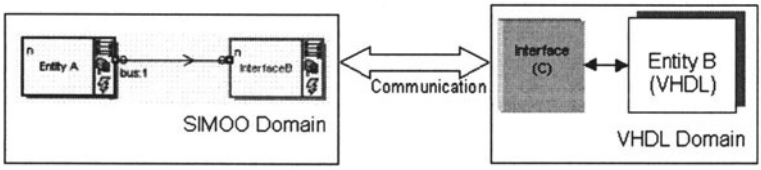

Figure 2(b). The same model, transformed after generation of interfaces for co-simulation

The communication between the SIMOO and VSS simulators is performed via sockets. This allows a distributed solution, where each simulator may run on a different node of a network. In the current version, a conservative approach [4] is adopted for synchronization between the SIMOO and VSS simulators. The base time unit is defined by the VSS simulator, and both simulators advance together their simulation times at each time step. A more efficient implementation of this co-simulation mechanism is currently under investigation. In this more optimistic approach [4], each simulator may run with its own clock and the synchronization is not performed at every clock cycle.

An analog part may be modeled by a set of differential equations that define the object behavior at every possible time value. The co-simulation strategy of the SIMOO environment follows a signal-flow approach, where objects are modeled as mathematical functions from the inputs to the outputs, as opposed to a structural approach, where objects are described as interconnections of analog components. This signal-flow approach is more

convenient for systems that don't have a physical implementation yet, or for systems whose design has a stronger emphasis at higher abstraction levels, like PID controllers, converters, filters, transfer functions, etc.

For the integration of an analog part into a SIMOO model, it is thus necessary that the SIMOO object encapsulating this part implements the numerical method needed to solve the differential equations, such as Euler or Runge-Kutta. The SIMOO object must contain attributes such as the time step for the numerical resolution of the equations, the equations themselves, and methods for handling state variables.

4. CASE STUDY

In order to illustrate design possibilities using S^3E^2S, we have modeled a crane and its embedded control. This system has been proposed in [5] as an attempt of benchmarking in the area of system-level modeling and synthesis.

The physical plant is composed of a crane with a load, moving along a track, as depicted in Figure 3. The modeling of the physical system is done by a set of differential equations, which describe the behavior of the crane with a load and external forces being applied. The control of the system involves a set of sequential procedures and the control algorithm itself, which will assure a smooth behavior while the car is moving.

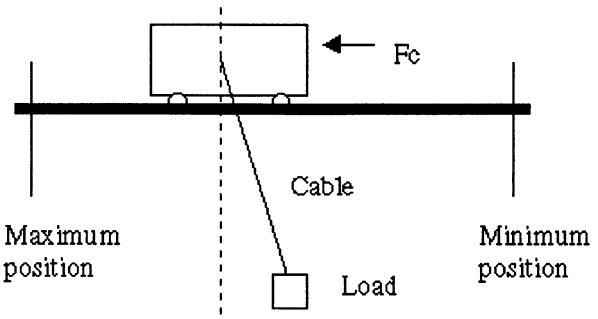

Figure 3. Crane moving along its track with load [5]

A first version of the modeling of the complete system can be seen in Figure 4. Object *Plant_rk* is the physical plant itself. It has been described as a set of differential equations that are solved in continuous time. Object *Actuators* is also modeled in the linear (analog) domain. Object *M_Control* performs the discrete step control algorithm and sensor checking. It receives sensor inputs at each 2 ms, covering the position of the car with some precision, the limits of the displacement for the car, the angle of the cable,

and the desired position of the load. All these inputs are used to complete other tasks. For example, the initialization phase of the control algorithm itself and the sensor verification are performed by this major block.

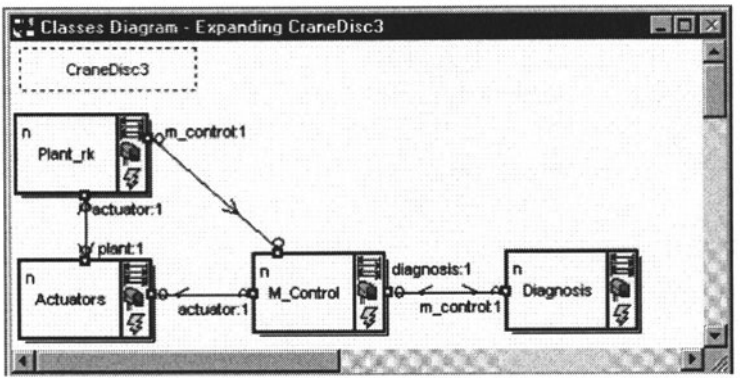

Figure 4. The initial crane and embedded control model

The control algorithm itself is implemented as a discrete computation of the state-variable method. In the control algorithm the goal is to move the crane with a linear displacement, without bumps and oscillations. A set of matrix multiplications must be performed at a fixed time step of 10 ms. If $q_n = [q1_n, q2_n, q3_n, q4_n, q5_n]^T$ is the discrete state vector of the crane, then

$$q_{n+1} = A*q_n + B*[\text{Motor\_Voltage Car\_Position}]^T \quad (1)$$

is the next discrete state of the control algorithm. Coefficient matrixes A and B have dimensions 3x5 and 2x5, respectively.

The object *M_Control* is also responsible for performing sensor checking. When the system enters this mode, the car is driven until extreme positions are found, where sensor inputs should detect its presence. This mode thus allows checking if any sensor function is missing.

Object *Diagnosis* is responsible for continuously checking plausibility of sensor values in parallel to the control algorithm. The position of the car and the value of the angle of the load with regard to the vertical axis are informed by the object *M_Control* and checked for plausible values. In case any discontinuity is verified, the emergency mode is entered immediately.

The control algorithm outputs the value of the force to be applied to the crane, and this is passed to the object *Actuators*. This object is responsible for driving the dc motor that controls the speed, the breaks, and the emergency break, that stops the crane until a power-on-reset is performed.

It is interesting to notice that the proposed benchmark problem allows various modeling solutions, as in almost any real life situation. For example, modeling of the crane behavior should be developed in the analog domain, by solving a set of differential equations that model its behavior. Besides, the

control itself could also be developed in the analog world, as some control systems still are. On the other hand, even if one modeled the control circuit as a differential equation in the analog domain, some digital hardware would still be present, due to the finite state machines required to perform sensor checking, system diagnosis, and emergency control. S^3E^2S allows any combination of these different modeling domains and a stepwise refinement of the solution, as it will be described in the next paragraphs.

In Figure 4, which showed our initial modeling of the proposed system, the behavior of all objects is described as C++ methods. Table 1 illustrates the methods implemented for the object *Plant_rk*. Figure 5 shows the main code for this same object.

Table 1. Methods for object *Plant_rk*

Method	Description
Eval	Called in each integration step; invokes functions to solve diff. equations
Rk4_Diff	Implements the Runge-Kutta integration method, to evaluate car distance
Diff	Describe the car position, over the time
Send_sensor	Send the plant state to objects *Control_A* and *AD*

```
/*state[ 0]  = xdot, state[ 1]  = x,
  state[ 2]  = alphadot, state[ 3]  = alpha  */

Parameters par;
MSGEA m;                    // port definition
double xdot, x, dxdot_dt, dx_dt;
double alphadot, alpha, dalphadot_dt, dalpha_dt;       // derivatives
xdot = state[ 0] ;          // initial values
x = state[ 1] ;
alphadot = state[ 2] ;
alpha = state[ 3] ;
t = Clock() / 1000;
par.Set(time-t);            // step
m.Set(Id(), actuator, "FORCE", NORMAL_PR, par);
SendReceiveNow(&m);
Fc = m.GetData().GetParAsReal(0);       // get data
dxdot_dt = (fc/mc) + (g*(ml/mc)*alpha) - ((dc/mc)*xdot);
dx_dt= xdot;
dalphadot_dt = ((((dc/mc)-(dl/ml))*(xdot/r))-g/5*(1+(ml/mc))*(alpha))
-((dl/ml)*alphadot) - (fc/(mc*r)) + (fd/(ml*r));    // differ.equations
dalpha_dt = alphadot;
state[ 0]  = dxdot_dt;     // next step
state[ 1]  = dx_dt;
state[ 2]  = dalphadot_dt;
state[ 3]  = dalpha_dt;
```

Figure 5. Differential equation describing the crane in object *Plant_rk*

In Figure 6 we have split *M_Control* into two different objects. Since the control algorithm has many arithmetic operations, we described it in a separate object *Control_A*. The finite state machine that performs sensor

checking is modeled in object *Job_Control*. All objects of the model are still described using C++ methods.

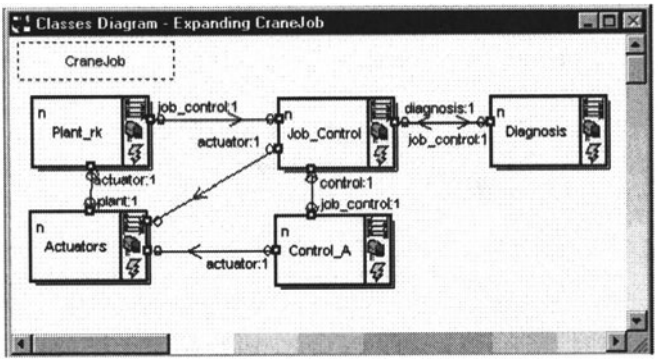

Figure 6. Refining the initial model: control algorithm as a separate object

In order to allow co-simulation and to start a first version of the synthesis of some system parts, *Job_Control* and *Diagnosis* are moved to VHDL. This requires the insertion of analog to digital and digital to analog converters, as shown in Figure 7. Notice that all FSMs, formerly implemented by separate objects *Control_A* and *Diagnosis*, have been grouped in the object *Control_Diag*, although this would not be mandatory. The collapsing of the different FSMs was done because possibly a better synthesis result could be obtained, since all FSMs have many common inputs.

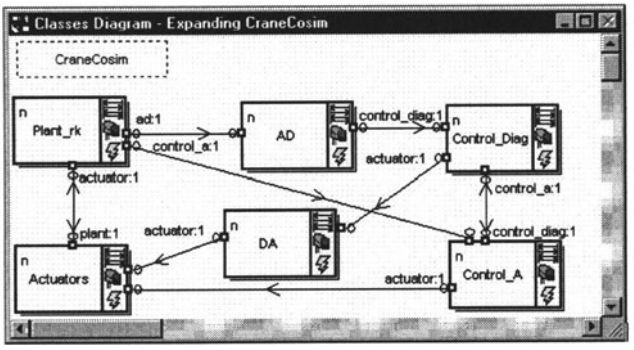

Figure 7. Synthesizing the system: *Control_Diag* as a VHDL object

The synthesis results for the FSMs are a total of 102 logic cells and 41 flip-flops, occupying an area of 8% of an Altera 10k10 FPGA.

5. COMPARISON WITH RELATED WORK

The specification and simulation of application-specific embedded systems is an area of active research. In the case of complex systems, which cannot be implemented by a single processor or controller and its associated software, it is difficult to specify the designer's intention. Many specification languages, or combinations of languages, are being used in industry [6].

The description of complex systems through a single, abstract language has been proposed [7,8]. Some approaches that follow this strategy adopt an object-oriented specification to describe both hardware and software [9-12]. In these cases, design partitioning is left to later design stages. The system is modeled as a set of objects, and each one of them may be later implemented as software or hardware, either digital or analog (as in [13]).

In these object-oriented approaches, system behavior is described by some procedural language, such as C++. The specification of embedded systems using procedural languages is a natural consequence of microprocessor-based system design. An important advantage of this approach is that the specification can be simulated and validated. However, this kind of specification is more appropriate for a software-based synthesis, where the target architecture is fixed and based on a given processor, and the specification is compiled into the processor's code. On the other side, these descriptions have an inherently sequential nature and are not appropriate for describing hardware semantics. Some extensions that allow the description of hardware features like parallelism and communication mechanisms have been developed [14]. Java has been also proposed, not only as a language for describing the abstract system behavior, but also as a basis for a target architecture based on multiple threads [15].

Although the S^3E^2S environment is also based on object-oriented specifications, where abstract behavior is given in C++, the drawbacks of this approach are partially avoided, because heterogeneous specifications mixing C++, VHDL and analog descriptions can be created.

This alternative approach of supporting modeling of heterogeneous systems is also followed by Ptolemy [16], an environment for simulation and prototyping of heterogeneous systems which also uses object-oriented technology. Ptolemy implements the combination of different simulation mechanisms, called domains (such as Synchronous Data Flow, Dynamic Data Flow, discrete event, and analog). Another environment allowing the specification and simulation of heterogeneous systems is described in [17], where a backbone in the operating system implements communication among dedicated simulators that are needed for heterogeneous objects specified in different languages.

S^3E^2S combines the advantages of the multi-language and heterogeneous simulation approach with the abstract, object-oriented specification. Objects can be modeled regardless of their future implementation as digital or analog hardware or software. Then, any object may be specified in any of these domains or refined into any of them, and every possible intermediate model generated during this stepwise refinement process may be co-simulated.

6. CONCLUSIONS AND FUTURE WORK

The automatic design of embedded electronic systems is an open area of research. A design environment must consider important aspects such as system specification, partitioning, validation, and synthesis. Regarding the specification, current design environments either propose a multi-language approach, where each component is modeled in a language which is suitable for a given implementation domain, or a single language approach, where all components must be modeled using a language from a single domain.

This paper presented S^3E^2S, an environment combining the advantages of both approaches in a flexible way. An abstract, object-oriented, initial specification may be created, and each object may be later refined into a different domain, using a suitable language (C, VHDL, differential equations). Any heterogeneous model created during the process of stepwise refinement may be co-simulated for validation. The paper illustrated these modeling and co-simulation capabilities by means of a concrete example.

We are currently implementing the automation of the synthesis capabilities of S^3E^2S. From a manual characterization of available processors and an automatic characterization of the application objects (specified as SIMOO abstract objects), the environment is capable of selecting the best multi-processor platform for implementing the application.

We are also developing a more efficient implementation of the co-simulation strategy. An optimistic synchronization mechanism will allow each object to have its own local time. Each object encapsulating an analog behavior will have a time step most appropriate for the numerical integration of the differential equations, while synchronization among objects will occur only as objects have to communicate events on interface signals.

ACKNOWLEDGMENTS

Special thanks are deserved to B.Copstein, J.F.H.Jornada and C.E.Pereira for the development of the SIMOO environment. Thanks are also due to F.B.Lima, who helped implementing the crane model.

REFERENCES

[1] L.Carro, M.Kreutz, F.R.Wagner and M.Oyamada. "System Synthesis and Processor Selection in the S^3E^2S Environment". In: 12th Symposium on Integrated Circuits and Systems Design. Natal, Brazil, 1999. Proceedings, SBC, 1999.

[2] B.Copstein, F.R.Wagner and C.E.Pereira. "SIMOO – An Environment for the Object-Oriented Discrete Simulation". In: 9th European Simulation Symposium. Passau, Germany, 1997. Proceedings, SCS, 1997. pp 21-25

[3] B.Copstein, C.E.Pereira and F.R.Wagner. "The Object-Oriented Approach and the Event Discrete Simulation Paradigms". In: 10th European Simulation Multiconference. Budapest, Hungary, 1996. Proceedings, SCS, 1996. pp 57-61

[4] R.Fujimoto. "Parallel Discrete Event Simulation". In: Communications of the ACM, Vol. 33, No. 10, Oct. 1990. pp 30-53.

[5] E.Moser and W.Nebel. "Case Study: System Model of Crane and Embedded Control". In: Design, Automation and Test in Europe, 1999. Proceedings, IEEE Computer Society Press, 1999. pp 721-723.

[6] W.Nebel and G.Gorla. "JAVA, VHDL-AMS, ADA or C for System Level Specification?" In: Design, Automation and Test in Europe, 1999. Proceedings, IEEE Computer Society Press, p. 720.

[7] J.K.Adams and D.E.Thomas. "The Design of Mixed Hardware/Software Systems". In: ACM/IEEE Design Automation Conference, 1996. Proceedings, ACM Press, 1996. pp 515-521.

[8] G.Martin. "Design Methodologies". In: Design, Automation and Test in Europe, 1998. Proceedings, IEEE Computer Society Press, 1998. pp 286-289.

[9] J.Böttger et al. "An Object-Oriented Model for Specification, Prototyping, Implementation and Reuse". In: Design, Automation and Test in Europe, 1998. Proceedings, IEEE Computer Society Press, 1998. pp 303-310.

[10]W.Wolf. "An Architectural Co-Synthesis Algorithm for Distributed, Embedded Computing Systems". IEEE Transactions on VLSI Systems, June 1997. pp 218-229.

[11]N.Woo, A.Dunlop and W.Wolf. "Codesign from Cospecification". IEEE Computer, Jan. 1997. pp 42-47.

[12]M.Aiguier et al. "ECOS: A Generic Codesign Environment for the Prototyping of Real Time Applications". In: Hardware/Software Co-Design and Co-Verification. Edited by M.Bergé, O.Levia and J.Rouillard. Kluwer Academic Publishers, 1997. pp 23-58.

[13]J.Z.Pino and K.Kalbasi. "Cosimulating Synchronous DSP Applications with Analog RF Circuits". In: 32nd Asilomar Conference on Signals, Systems and Computers, 1998.

[14] I.Page. "Design of Future Systems". In: Design, Automation and Test in Europe, 1998. Proceedings, IEEE Press, 1998. pp 343-347.

[15] M.Mrva, K.Buchenrieder and R.Kress. "A Scalable Architecture for Multi-Threaded Java Applications". In: Design, Automation and Test in Europe, 1998. Proceedings, IEEE Computer Society Press, 1998. pp 868-874.

[16] A.Kalavade and E.Lee. "Hardware/Software Codesign Using Ptolemy". In: Codesign. Edited by J.Rozenblit and K.Buchenrieder. IEEE Press, 1995. pp 397-413.

[17] A.A.Jerraya and R.Ernst. "Multi-language System Design". In: Design, Automation and Test in Europe, 1999. IEEE Computer Society Press, 1999. pp 696-699.