Hardware - Software co-design of embedded telecommunication systems

using multiple formalisms for application development

N. S. Voros¹, S. Tsasakou¹, C. Valderrama², S. Arab², A. Birbas¹, M. Birbas¹, V. Mariatos¹, A. Andritsou³

¹*Department of Electrical and Computer Engineering – University Campus of Rion 26 500, Greece, Tel: +30 61 97312, Fax: +30 61 997333 , E-mail: voros@ee.upatras.gr*

²*TIMA/INPG 46 Ave Felix Viallet F-38031 Grenoble Cedex, France, Tel: +33 476 574 870, Fax : +33 476 473 814, E-mail: CA.Valderama@imag.fr*

³*INTRACOM S.A, P.O. Box 68 Paiania Attikis 190 02, Greece, Tel : +30 1 690082, Fax : +30 1 6860312, E-mail: anda@intranet.gr*

Key words: HW-SW co-design, multiple formalisms, embedded telecommunications systems

Abstract: In this paper a co-design methodology based on multiformalism modelling is presented. It defines a platform that integrates different notations and, the necessary mechanisms to handle different in nature models in a coherent way. The supported formalisms cover a wide area of application domains, allowing the designer to select the notations that are mostly appropriate for his/her application. The proposed co-design development cycle provides a full blown path from system specification to a virtual prototype of the system under construction. The methodology has been applied for the design and implementation of a MAC layer protocol, named MASCARA, for providing ATM QoS over wireless connections.

1. INTRODUCTION

During the last years, a new generation of design methods for embedded systems, referred to as co-design [8, 10, 12, 13, 14, 15, 17, 18], has emerged and matured. These methods are able to handle mixed hardware/software systems starting from behavioural-level down to implementation. They are based on co-design tools that are capable of providing concurrent design of the different parts of heterogeneous distributed systems, automation of H/W-S/W partitioning and automation during the integration steps at the lower levels of the development cycle.

In the next paragraphs we present a co-design development cycle that supports different specification formalisms. The goal is to propose a methodology that embraces a wide range of application domains [6] under a consistent co-design

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: [10.1007/978-0-387-35394-4_29](https://doi.org/10.1007/978-0-387-35394-4_29)

S. Budkowski et al. (eds.), *Formal Description Techniques and Protocol Specification, Testing and Verification*

© IFIP International Federation for Information Processing 1998

framework that leads to a virtual prototype of the final system. The proposed methodology framework is exhibited through the design of MASCARA protocol, a MAC layer protocol for wireless ATM networks.

2. CO-DESIGN USING MULTIPLE FORMALISMS FOR SYSTEM DEVELOPMENT

The increase of complexity in embedded systems and the demand for virtual prototypes as early as possible in the design cycle, are putting strict requirements on methodologies and the tools supporting them. The co-design approach proposed in this paper aims at building a coherent couple of methodology-toolset for the entire design cycle of embedded systems. Particular emphasis is given on the phases of system-level co-modelling, co-simulation, partitioning and co-targeting.

Co-modelling aims at formalising the specifications of an embedded system. Such specifications usually include several evolving models described by means of different notations. *Co-simulation* can be used at this stage, before partitioning, to conduct an early validation of the specifications. Co-simulation, as it says, involves simulation of at least two different models that interact among each other e.g. exchange data dynamically. Then comes *partitioning* to decide which parts of the system are going to be implemented in S/W and which parts in H/W. Partitioning takes into account the target platform (e.g. processors, distributed architecture etc.) to assign each system part. Co-simulation can also be used at this stage to estimate the validity of the choices. Finally, once the partitioning is agreed, *targeting* takes place. It involves S/W code generation, H/W synthesis, realisation of interfaces between H/W and S/W modules, and implementation of the whole system into the target platform.

The proposed methodology-toolset materialises a design environment for hybrid (continuous and discrete) H/W - S/W embedded systems. It supports a global design flow, based on multi-formalism modelling and executable prototypes to validate the system design. The toolset accompanying the proposed methodology has been built from existing products and prototypes, which have been adapted and extended for the needs of industrial end-users. Interfaces have also been developed for connecting external tools such as VHDL tools for H/W design. The co-design flow detailed in Figure 1 outlines the methodological stages involved in the design of embedded systems.

From the formalism point of view, Figure 2 depicts how the associated supporting tools are combined in the context of the proposed approach. The circles depict the co-modelling and co-simulation modules that have been developed for the need of our methodology. They implement MATRIXx-SCADE co-modelling, SDL-SDL co-execution, MATRIXx-Statechart co-simulation, SDL-MATRIXx-Statechart co-simulation and C-VHDL co-simulation [6]. Therefore multi-formalism co-simulation is possible both at the specification level and at the implementation level. Two more arrows have been drawn to indicate improvements dealing with software distribution using SEA and VHDL generation from SDL using COSMOS. The latter

platforms are used in the partitioning phase: COSMOS produces C for S/W parts and VHDL for H/W parts, while SEA generates C from Statechart and MATRIXx models for distribution on a given H/W platform. Code generation and targeting is realised by taking into account the constraints attached to embedded systems such as e.g. communication efficiency and memory limitations.

Figure 1. H/W-S/W co-design using multiple formalisms for embedded system design

Regarding the tools that match with the various phases of the proposed methodology, the following could be mentioned. For co-modelling, the user can employ four notations: Lustre [6], SDL [1, 3, 6], Statecharts [7, 6] and MATRIXx [6] diagrams. Each of these notations is supported by a corresponding tool and the proposed toolset provides for their efficient interconnection. With respect to partitioning, the user can rely either on the COSMOS [4, 11, 17] approach (covering SDL, Statechart, C and VHDL formalisms), or on the SEA [5] approach (involving MATRIXx, Statechart and C notations) for software distribution on hardware. As already mentioned, co-simulation is possible both at the specification and at the implementation levels, for the purpose of model validation and partitioning assessment respectively. Application targeting is generally covered by the partitioning tools and also by the CASE/CAD tools.

3. SDL-STATECHARTS CO-MODELLING PRIMITIVES

The effectiveness of the proposed approach has been tested over a number of different application domains including automotive, aerospace and telecommunications. Since the primary goal of this paper is to present the adaptation of the proposed methodology/toolset to embedded telecommunication systems, we will restrict our analysis to the tools usually involved in such systems. The following sections introduce the idea of developing telecommunication systems using SDL and Statecharts as complementary formalisms for system specifications. A set of transformation rules between Statechart and SDL is also presented, and proposes the required extensions to couple the two languages.

Figure 2. Tool coupling for multiformalism system specification

3.1.Statechart notation vs. SDL notation

The Statechart notation is an extension of the Finite State Machine notation. The primary objective of this modelling technique is to describe the system behaviour by means of a hierarchy of state diagrams. Concepts such as state refinement and concurrent activities are supported, allowing to represent a system behaviour at different levels of abstraction. The Statechart model is now popular and is present in well-know systems and software engineering techniques such as UML [2] or ROOM

[16]. It is used to model real-time systems that are centralised (i.e. without communication and synchronisation) and have a static architecture (i.e. no dynamic task creation).

The SDL notation has been defined to model and develop distributed real-time systems. The concept of dynamic objects that can be created/suppressed, is supported, so the system at run-time may behave as a dynamic network of communicating processes. For this purpose, SDL supports the necessary mechanisms for communication. The simulation and dynamic checking of SDL models are possible because of the underlying formal semantics of the language. The production of executable code is also possible, to be compared for example with the Statechart technique where C/C++ code has to be entered to describe the actions in the transitions. SDL is a graphical language as Statechart is, but does not include state abstraction facilities, and transition diagrams are purely flat diagrams, hierarchical states are not possible, neither entry and exit actions.

3.2.Complementary features between Statechart and SDL

From the conceptual point of view, the Statechart and SDL notations seem rather complementary than in competition. Statechart provides facilities for model abstraction, whereas SDL provides a complete graphical language to describe the dynamic behaviour at the lowest level. Statechart can be used early in the system development, to sketch a first description of the behaviour that can be given to non-expert users. Then this model can be refined with SDL at the detailed design stage, to provide a complete behavioural description. The latter can be transmitted to the development engineers for production, and to the validation teams for checking the conformance to the specification constraints.

Therefore a solution that integrates both notations to take advantage of their respective facilities is welcome. If most of the Statechart concepts are interesting for introduction in SDL, the concept of concurrent activity seems not complementary at all with SDL. The SDL designer usually describes the concurrent objects – that exist at run-time – by processes. Internal concurrency seems unnecessary for these objects; if parallel actions must be described, the right way would be to refine the object in two separate processes which can run in parallel. Besides it must be noted that internal concurrency is forbidden or not recommended in most Statechart techniques such as UML or ROOM. At last, it is worth to remind that internal concurrency is the unique way to structure the system in the original Statechart notation.

3.3.An approach to combine Statechart and SDL notations

The approach should take as basis the complementary nature of Statechart and SDL, as discussed above. The natural way of working is to use Statechart first for conceptual modelling. Basically, a Statechart model is a hierarchy of state diagrams. The next step is to transform them into SDL constructs and to derive an architectural model. Finally the latter will be completed in SDL in order to obtain a detailed

design model that can be simulated. It must be noted that those considerations on the use of the concepts are independent of the support provided by the tools.

Therefore the main task is at language level to identify which Statechart constructs must be included in SDL, and how to transform them in standard SDL. For the ease of integration, assume a notation for Statechart that is a superset of UML, without preventing the use of the basic Statechart.

3.4.Mapping between Statechart and SDL

Table 1 makes the correspondence between Statechart constructs and SDL standard constructs. The basic primitives of mapping SDL constructs to Statecharts and vice versa are explained as follows:

Table 1. Correspondence between Statechart and SDL constructs

Statechart construct	Corresponding SDL construct
State name	State name
Entry action	No direct correspondence
Exit action	No direct correspondence
Activity	No direct correspondence
Initial state	Start
End state	Stop
Macro-state	No direct support
Internal concurrency	Not supported within the same process; basically supported between processes
Transition	Transition
Event with parameters	Signal with parameters
Provided (condition)	Enabling (condition)
Action	SDL transition body
Output	Output
Internal transition	Usual transition
Delay	Timer (set, reset and expiration)

- **Activities** : Activities i.e. sequential processing without the triggering of events, are not supported in SDL finite state machines. In SDL, actions are only performed on transitions. However it can be shown that activities can be easily modelled in the SDL notation extended by entry/exit actions as introduced below.
- **Internal Transitions with Entry/Exit Actions** : In order to support internal transitions i.e. execution of internal actions during all the time an object stays in the same state, it will be possible to inhibit automatic expansion of entry and exit actions by means of a specific syntactic rule for state and next state naming.
- **Internal Concurrency** : The usual way in SDL to express internal concurrency i.e. parallel activities defined within the same Statechart, is to refine the object in separate processes, each process implementing one parallel activity. Internal concurrency within the same SDL process has not been implemented.

- **Inheritance** : The Statechart notation does not support inheritance. Therefore inheritance will only be allowed in standard SDL constructs, but not used in the Statechart extensions that have been implemented.
- **Dynamic Semantics** :
 - Each concurrent class (process) has its own Statechart, i.e. a Statechart corresponds to a process. This differs from pure Harel's Statechart – one Statechart model for the whole model – but this conforms to the UML Statechart variant.
 - The dynamic semantics will conform to the SDL standard: asynchronous point-to-point communication, no broadcasting, each finite state machine has one queue that stores the events received but not yet consumed, dynamic process instance creation.

4. SYSTEM PARTITIONING

One of the key issues in the co-design of complex systems is the action of partitioning. The decision of which parts will be implemented in software and which will be transferred in hardware is crucial for the efficient implementation of the final system. Even though several prototype tools already exist for system partitioning, most of them are nothing but simple SDL-to-VHDL translators that produce a description that may feed existing RTL synthesis systems. In the context of the proposed methodology, the hardware-software partitioning is performed using COSMOS toolset, which is presented in the sequel.

4.1. User guided system transformational partitioning

The approach for SDL to C/VHDL proposed within COSMOS environment is built around an intermediate format called SOLAR [4, 11], designed to represent both the system concepts as well as those used by hardware description languages. The intermediate format SOLAR constitutes an intermediate representation permitting the unification of different specifications described in either hardware, software or system description languages. Thus, the different parts of software and hardware of a system can be unified within a single SOLAR format. SOLAR supports three levels of abstraction: the system level, the behavioural level and the Register Transfer Level. In addition, this model facilitates the reuse of existing sub-systems in the form of a library of components and communication models. SOLAR is used as an intermediate format during synthesis and is completely transparent to the designer.

The basic construction within SOLAR is a state table denoted by the keyword `StateTable`. It permits the specification of hierarchical and communicating finite state machines. In addition, other structures are added to give a modular specification and facilitate the communication between the processes. The design unit (`DesignUnit`) is introduced for structuring a system into a set of interconnected subsystems. The channel unit (`ChannelUnit`) permits the specification of communication between the communicating subsystems.

The translation approach adopted is based on the correspondence between the basic concepts in the two languages SDL and SOLAR. Concepts used in SOLAR and in SDL, permits us to observe certain similarities. Nevertheless, SDL describes the behaviour at a level sufficiently abstract, whereas, the SOLAR descriptions, even if they are at a high level, are implementation oriented. The translation process proposed essentially works with the behavioural, communication and structure aspects of the specified system. Since SDL design system is in front of COSMOS, basic concepts of SDL are supported in COSMOS. The following table lists the subset SDL supported by COSMOS. Synthesis is the final stage performed on the SOLAR representation and translation into VHDL, for the hardware components and into C, for the software components.

Table 2. SDL primitives to SOLAR primitives mapping

SDL		SOLAR
Structure	System, blocks, processes	Done
Communication		Done
Hierarchical specifications		Done
Declarations	Data type	Limited to: Dcl, synonym, newtype, literal, structure sort
	Exp/imp procedures	Done
	Shared variables	Done
	ADTs + operators	Done
Process/Procedure bodies	State tables	Done

The organisation of partitioning into several small steps reduces the complexity of the problem. The designer controls the partitioning history within an interactive environment, through fine grain control of the synthesis process. This methodology can be seen as a human guided compilation where the designer spends additional effort to produce an efficient implementation. To facilitate the user's interaction for incremental transformations, all these refinements make use of a set of primitives called *split*, *merge*, *move*, *flat* and *map* allowing to decompose and transform objects. Figure 3 summarises these partitioning primitives.

The user guided transformational partitioning assumes that the designer starts with an initial specification and an architectural solution in mind. System design from specification to implementation is performed through a set of primitives allowing the designer to transform the system, following an incremental refinement scheme, in a distributed model that matches the architectural solution. All the refinement transformations are performed automatically. Additionally, the decisions are made by the designer who uses his knowledge and experience to achieve the desired solution.

4.2. Architecture generation

The architecture generation translates the intermediate format used in COSMOS into executable code (C and VHDL) to allow the co-simulation and co-synthesis of the system. The virtual prototype generated is a heterogeneous architecture composed of a set of distributed modules, represented in VHDL for hardware elements and in C

for software elements, communicating through communication modules from a library of components. The virtual prototype is a simulatable model of the system that implements or emulates the initial specification. The generated model allows to separate the behaviour of the modules (hardware and software) and the communication units. Inter-module interaction is abstracted using communication primitives that hide the implementation details of the communication units.

Figure 3. Partitioning primitives used in COSMOS

4.3.C/VHDL co-simulation interface

In the context of the proposed methodology, co-simulation is reckoned as the joint simulation of a system description based on heterogeneous notations, since different parts of the system may be described using different notations. The rationale of adopting multiformlism is that, (a) when dealing with overall system specification, the use of multiple formalisms is desirable since it allows describing different aspects of the system with the most appropriate notation, and (b) when dealing with the virtual prototype of the system we need at least two different notations for describing efficiently the hardware and the software parts respectively.

The COSMOS environment generates a distributed hardware/software co-simulation environment for heterogeneous systems prototyping. The environment provides as features: distributed hardware/software co-simulation and automatic hardware/software interface generation. Hardware components can be described at different levels of abstraction and generic/specific software debuggers can be used. Starting from a brief description of the interface of the interconnected modules, an operation named *communication synthesis* produces automatically the interface between hardware and software parts.

The distributed co-simulation environment is composed of a VHDL simulator (VSS: VHDL SYNOPSIS Simulator) for hardware components and C-debuggers (any tools in the market) for software modules communicating via a software bus. This bus relies on the UNIX IPC layer. The software bus and the elements used to interface VHDL and C modules during simulation are automatically created as result of communication synthesis.

Figure 4. Overview of a wireless ATM system

5. DESIGN OF A MAC LAYER PROTOCOL USING MULTIPLE FORMALISMS FOR SYSTEM SPECIFICATION

The methodology proposed within this paper has been used to develop a MAC layer protocol named MASCARA [9] for offering ATM services over wireless connections. The wireless ATM architecture consists of *ATM switches*, *Access Points* and *Mobile Terminals* as shown in Figure 4. *ATM switch* is a standard customer premises access node, containing also mobility specific software and minimum hardware modifications; *Access Point* is the network element connected to the ATM switches with standard ATM connections; finally, *Mobile Terminal* is the end-user equipment that contains the *wireless ATM radio adapter* card, interfacing the 20 Mbits/sec air-interface.

For connection admission MASCARA provides information on link conditions including e.g. the available bandwidth and cell error ratio. The actual call admission is made taking into account existing connections, the radio link condition and spare capacity required for handovers. The scheduler of the protocol is designed to preserve the admitted connections as conformant as possible within the negotiated connection QoS parameters. The service policy implemented by the scheduler can be tuned to meet the needs of a particular installation or a particular set of users.

5.1. System specification

The first stage of the protocol design involves requirements capturing using UML [2]. The reason for using UML lies in the fact that significant parts of the protocol are already available as standards in UML. So having available standard parts of the protocol, like SAR (Segmenting and Reassembly) unit, we are able to reduce the amount of work needed for designing them (we use them either as they offered or with minor changes). Figure 5, presents a high level specification of MASCARA using UML, while Figure 6 depicts the internal behaviour of the “Control Segmenting and Reassembly” object using Statecharts (UML uses Statecharts for describing the dynamic behaviour of the objects composing a system specification). Due to limited space, we have restricted our specification only to the essential parts of the protocol, while for shake of simplicity we have intentionally eliminated the references among the constituent parts of the protocol. Among the objects composing the MAC layer, *ATM Layer* corresponds to the ATM layers above MASCARA that implement the classical ATM services; *Physical Layer* implements functions pertaining to the actual physical layer (in our case the transmission medium is air) of a traditional protocol stack; the remaining objects are part of the MASCARA specification and incorporate aspects concerning mobility.

The final stage of the specification involves description of the protocol’s parts using SDL (Figure 7 reflects part of the UML specification presented in Figure 5). The objects have been transformed in blocks and their internal behaviour has been implemented using SDL structures (see Table 1). Having completed the system specification using objects, we transform them automatically to SDL adding the necessary implementation details for having a more detailed version of protocol’s specification. This way, we combine of-the-shelf components described in UML with newly created components described in a powerful language like SDL. Having a detailed system specification in SDL, we are able to exploit the power of state-of-the-art tools like ObjectGeode which incorporates a set of powerful utilities for SDL-to-Statechart translation, exhaustive simulation, deadlock detection etc. So, we are able to verify/validate the system’s specification early enough in the co-design flow, where the cost of repairing possible design flaws is relatively small.

Figure 5. The constituent parts of the MASCARA protocol in UML

Figure 6. Statechart representing the dynamic behaviour of "Control Segmenting and Reassembly" object

5.2. System partitioning

This section refers to H/W-S/W partitioning of the MASCARA protocol. We will illustrate the overall design flow in COSMOS, from the system-level specification given in SDL to the distributive hardware/software architecture described in C/VHDL. The use of SDL allows for fivefold reduction of the size of system's specification when compared to distributed C/VHDL models. All the transformation applied in this section are fast enough to look instantaneous during an interactive session.

Figure 7. Specification of a small part of the MASCARA protocol in SDL

The SOLAR intermediate format is obtained by an automatic conversion from the initial SDL description. During the conversion, the following operations are performed: processes are converted to design units, processes behavior are converted to state stables and interconnections among SDL processes are converted to abstract channels. Figure 8 depicts the initial SOLAR model. The refinement steps performed for the production of the MASCARA protocol's virtual prototype, are listed in the sequel: (a) *structural flat* to keep out the hierarchy of all the design units, (b) *protocol selection*, where a communication protocol is chosen from a

protocol library (the FIFO and rendez-vous protocols were chosen to implement all the abstract channels), (c) *channel map*, where the abstract channels were replaced by well defined signals interfaces, (c) *hardware/software assignment*, where (after a trial and error approach) the processes WDLC and SCH and the FIFO controller of WDLC are set to have a software implementation. The processes TB, MPDU and the others FIFO controllers are set to have an hardware implementation.

Summing up, the SDL specification has been transformed to SOLAR so as to enable efficient system partitioning. Each process has been represented as a design unit whose internal behaviour has been specified using a state table (which is the conceptual equivalence of the an FSM in SDL or a Statechart in UML). The communication scheme among the various design units have also been specified more accurately by selecting specific protocols for implementing their communication. For example, the concept of channels or signal routes that connect two system blocks in SDL are no longer abstract. In fact, the channel notion incorporates concepts such as communication protocols (e.g. FIFO, Rendez-Vous, etc) and specific bus (channel) arbitration schemes.

Figure 8. Intermediate model resulting from the partitioning

6. CONCLUSIONS

In the previous sections we presented a co-design methodology for embedded systems design, where the concept of system specification using multiple formalisms was introduced. A platform for coupling different notations was introduced and, the necessary mechanisms to handle different in nature models in a coherent way were presented. The specification languages supported refer to a wide area of application domains, enabling selection of the mostly appropriate formalisms for a specific application. The co-design development cycle presented, provides a fully fledged methodology that leads from system specification to a virtual prototype of the system under construction, where the software parts are implemented in C and the hardware parts are implemented in VHDL. The efficiency of the proposed co-design approach is depicted through its use for the design and implementation of a MAC layer protocol, named MASCARA, for providing ATM QoS over wireless connections. The protocol has been designed using UML and SDL as complementary formalisms for requirement capturing. The system partitioning and the creation of a virtual prototype of the final system has been based on the COSMOS toolset and the system description in the SOLAR notation, the underlying formalism supported by COSMOS.

7. REFERENCES

- [1] "ITU-T Specification and Description Language SDL", Recommendation Z.100 (SDL-92). ITU General Secretariat, Geneva, 1992. <<http://www.itu.ch/ITU-T>>.
- [2] "UML Semantics & UML Notation Guide - Version 1.1", OMG Document, September 1997. <<http://www.rational.com/uml>>.
- [3] "ObjectGeode: Method Guidelines", 1997
- [4] A.A. Jerraya, K. O'Brien, "SOLAR: an intermediate format for system-level modelling and synthesis", in J. Rozenblit, K. Buchenrieder (Eds) IEEE Press, Piscataway, NJ., 1994.
- [5] B. Kleinjohann, E. Kleinjohann, J. Tacke, "The SEA Language for System Engineering and Animation", Conference in Application and Theory of Petri Nets, LNCS 1091, Springer Verlag, 1996.
- [6] COMITY project, Deliverable, ESPRIT/23015/WP2/JUN97/D20.1, "COMITY User Documentation, V1", June 1997
- [7] D. Harel, "Statecharts: A visual formalism for complex systems", Science of Computer Programming 8(3), pp.231-274, July 1977.
- [8] D.Gajski, F.Vahid, S.Narayan, J.Gong, Specification and Design of Embedded Systems, Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1994.
- [9] Frederic Bauchot, Stephane Decrauzat, Gerard Marmigere, Lazaros Merakos, Nikos Passas, "MASCARA, a MAC Protocol for Wireless ATM", ACTS Mobile Summit, 1996, pp. 647-651
- [10] J.Buck, S.Ha, E.Lee, Ptolemy: A framework for simulating and prototyping heterogeneous systems, International Journal of Computer Simulation, January 1994.

- [11] Jean-Marc Daveau, Gilberto Fernandes Marchioro, Tarek Ben-Ismaïl, Ahmed Amine Jerraya, "Protocol selection and interface generation for H/W-S/W co-design", IEEE transactions on VLSI systems, Vol. 5, No. 1, March 1997
- [12] M. Srivastava, R.Brodersen, SIERA: A unified framework for rapid-prototyping of system-level hardware and software, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 676-693, June 1995.
- [13] OMI/MCC, "Hardware Software Co-design Study Report", October 1996.
- [14] P.A.Subrahmanyam, W.Wolf, Hardware-Software Codesign for Embedded Systems, in ASP-DAC'95, Makuhari Messe, Chiba, Japan. pp. 1-72, 1995.
- [15] R.Ernst, J.Henkel, Th.Benner, W.Ye, U.Holtmann, D.Herrmann, M.Trawny, The COSYMA Environment for Hardware/Software Cosynthesis, Journal of Microprocessors and Microsystems, Butterworth-Heinemann, 1995.
- [16] Selic B., Gullekson G., Ward P., "Real-Time Object-Oriented Modelling" (ROOM), John Wiley, New York, NY, 1994.
- [17] T.Ben-Ismaïl, K.O'Brien, A.A.Jerraya, PARTIF: Interactive System-level Partitioning, VLSI Design Vol. 3 no 3-4, pp. 333-345, 1995.
- [18] W.Wolf, Object-Oriented Co-Synthesis of Distributed Embedded Systems, in Proceedings CHDL'95, IFIP, 1995.