An Object-Oriented Design Process for System-on-Chip using UML

Qiang Zhu, Akio Matsuda,
Shinya Kuwamura, Tsuneo Nakata
Fujitsu Laboratories Limited
1-1, Kamikodanaka 4-chome, Nakahara-ku,
Kawasaki 211-8588, Japan
0081-44-754-2263
{shiyu, amatsuda, kuwa, nakata}@flab.fujitsu.co.jp

Minoru Shoji
Fujitsu Limited
1-1, Kamikodanaka 4-chome, Nakahara-ku,
Kawasaki 211-8588, Japan
0081-044-754-2391
shoji.minoru@jp.fujitsu.com

ABSTRACT
The object-oriented design process has been a hot topic in software development since it will improve product quality and productivity significantly, which is also a major issue in system-on-chip design. In this paper, a design process is proposed for hardware-software heterogeneous systems by reinforcing parallelism, structure, and timing. The management of design abstraction is also introduced for refinement of hardware. UML is used as a modeling language, and the reinforcement above is gracefully integrated into UML by its extensibility mechanism. An example of architecture exploration and performance analysis is illustrated through the application of the process to an image decoding design.

Categories and Subject Descriptors
C.4 [**PERFORMANCE OF SYSTEMS**]

General Terms: Performance, Design, Experimentation

Keywords: System Level Design, Design Process, UML, Object-Oriented Analysis and Design, System Level Performance Evaluation

1. INTRODUCTION
With increasing complexity of hardware-software heterogeneous systems such as SoC (System-On-Chip), SoC design becomes very difficult and costly. There are two risks in SoC design. The risk of functionality is caused by misunderstanding requirements from customers. If designers implement the system according to wrong specification, perhaps they have to make a huge effort to modify the design when they notice their misunderstandings. The risk of performance is caused by insufficient performance evaluation on the early stage of design. If designers finished the design but it cannot satisfy performance requirements. They have to take a long time to redesign it.

The latter can be resolved by using system level performance evaluation and architecture exploration technology such as Y-chart approach [1] and environments such as Polis [1], VCC [2] and Spade [3].

For avoiding the risk of functionality, we need a specification analysis and modeling techniques. In software community, the Object-Oriented Analysis and Design (OOAD) techniques [4] are used for analyzing the requirements from customer and the functionality of a target application efficiently. The UML (Unified Modeling Language) [5] is employed as a modeling language to characterize the artifact of the analysis and design obviously, clearly and comprehensively. We can make a set of graphical views for system with UML in order to confirm the correctness of analysis and design before implementation.

In this paper, we propose SLOOP (System Level design with Object-Oriented Process) design process that integrates modeling techniques with the system level performance evaluation and architecture exploration methodology. We show two contributions in this paper. Firstly, we establish a system level performance evaluation methodology for verifying the performance and functionality at system level. Secondly we extend UML using its extensibility mechanisms to model parallelism, structure and timing that are essential notions in SoC design.

In Section 2, we introduce the overview of SLOOP design process. The extension of UML notations is described in Section 3. We demonstrate an image decoding system application and experimental results in Section 4. Finally, we make a conclusion for SLOOP design process.

2. THE SLOOP DESIGN PROCESS
SLOOP employs four models to develop the SoC system incrementally before software and hardware implementation. Each model details three aspects of the target system – functionality, structure and timing [6]. Figure 1 indicates the design flow of SLOOP.

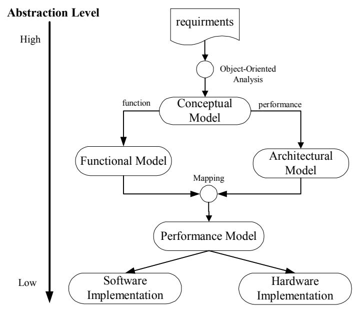

Figure 1. Design flow of SLOOP

Conceptual model describes the result of analysis of requirements from customer. In SLOOP, we analyze requirements from customer with OOAD techniques. The approach of analysis is same as that of application software. Conceptual model helps designers to grasp requirements from customers include functional requirements and non-functional constraints such as performance, area, shape and power consumption etc. The misunderstanding of requirements can be avoided using artifacts of conceptual model.

Functional Model focuses on the structure of function but not considers the physical architectures and timing. In functional model, we expose task level parallelism and make communication explicitly. Functional model consists of processes and communications between processes. In functional model, there are two indications for measuring the workload of the model.

- Computation workload is statistics of the number of invocation of each process.
- Communication workload is used for calculating the number of communicated tokens.

Architectural model represents the physical resources of architecture. Resources consist of processing resources and communication resources.

- Processing resources such as processors, DSPs, ASICs are used for implementing processes in function model.
- Communication resources such as buses, memories are employed to realize the communication channels among processes in function model.

Each resource in architectural model is parameterized with their attributes. For example, processor is parameterized with the multi-task scheduler. Bus is parameterized with bus width, transfer latency, arbitration algorithm. The architectural model in SLOOP is provided as a template class library, so that designers can manipulate it through making a class instance simply.

Performance model maps processes in functional model onto processing resources of architectural model explicitly, as well as assembling a communication channel with communication resources. Using performance model, system level trade-offs can be performed by evaluation of performance for a selected architecture. Using the statistics of computation workload and communication workload, designers can find the bottleneck of the selected architecture easily, so that it can help designers to improve the system to satisfy the performance requirement of design. In performance model, designers give the run-time of each process, which is mapped onto processors or ASICs. The run-time can be obtained either from a lower level model of the processing resource (ISS model, High Level Synthesis Tools), or can be estimated by an experienced designer. In performance model, both of functionality, structure and timing must be considered. Consequently performance model can be used for both function and performance verification. After performance model, the partition of hardware and software will be known, and can be implemented respectively after performance model shown in Figure 1.

SLOOP uses C++ and SystemC [8] as a description language to implement each model. Designers can confirm function and performance through simulation-based verification. In SLOOP, we employ UML as a specification language before implementation of each model. We expect two effectives via using UML:

- Clarification of specification – Because UML can model design specification using graphical diagrams, designers can confirm the correctness of design before coding using UML.
- Language independent design – Because UML does not depend implementation language, we can implement UML model using any implementation language such as C, SystemC, HDL.

For reasons above, SLOOP introduces two phases to realize each model. Modeling phase specifies results of analysis and design using UML. Implementation phase implements the UML model into C++/SystemC as an executable model shown in Figure2.

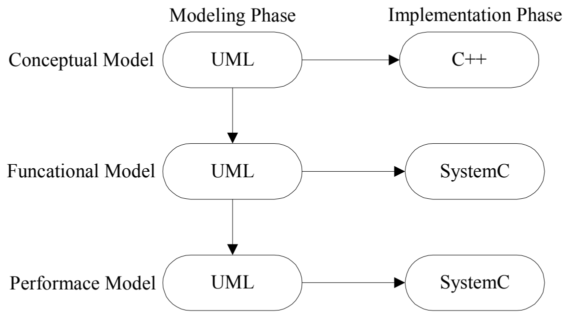

Figure 2. SLOOP design process

SLOOP adopts C++ as an implement language for conceptual model. For realizing the parallelism, timing and software-hardware modeling, we adopt SystemC, that is a C++ library and run-time environment for modeling systems both at the RT level and at more abstract levels in SLOOP for implementing functional model, architectural model and performance model.

3. THE EXTENSION OF UML

UML is a collection of graphical notations for capturing a specification of a software system. The UML is widely used in software community as a modeling language to specify the requirement, document the structure, and analyze the target system successfully. The notations of UML have formal syntax defined by the OMG [5]. They are primarily graphical, with textual annotation. Consequently, designers can certainly grasp the specification, which is described in UML and employ UML as a standard to communicate each other.

For reasons above, we also employ UML as a modeling language in SLOOP. However, the standard UML insufficiently models the software-hardware heterogeneous systems because of lacking in describing parallelism, architectures and timing, which are indispensable notions for describing heterogeneous systems. The Modeling for software-hardware heterogeneous systems is similar to model real-time software systems. The ObjecTime ROOM methodology [10] is developed specifically for dealing with distributing real-time systems by using the extensibility mechanisms of UML. Rose RealTime [9] is a commercial tool from Rational for modeling real-time software systems based on ROOM technology. The Embedded UML [11] is proposed to deal with the embedded system.

3.1 Notations

In SLOOP, we propose an extension of UML derived from ROOM and Rose RT. The key notions in functional decomposition as defined in SLOOP are:

- Module – A structural entity, which can contain a process, ports channels or other modules. The module is an active object used for realizing the concurrency and parallelism. The concept of module is similar to the concept of 'Capsule' in ROOM and Rose RT. In UML, a module is represented by the <<module>> stereotype of class.
- Interface – Provides a set of method declarations, but provides no method implementations and no data fields. The interface also defined in standard UML concepts. The mechanism of interface can improve the reusability of models. In UML, an interface is modeled by the <<interface>> stereotype of class.
- Channel – Implements one or more interfaces, and serves as a container for communication functionality. Channels represent the communication channels between processes. This is similar to the nation of 'Connector' as defined in ROOM and Rose RT. A Channel is described by the <<channel>> stereotype class in UML.
- Port – An object through which a module can access a channel's interface. Ports are objects whose purpose is to act as boundary objects to a module instance. The concept of port also defined in ROOM and Rose RT. In UML, a port is represented by the <<port>> stereotype of class.

Note that these notions are also proposed in SystemC2.0 [13].

Figure 3 indicates an example of a single port named *p* belonging to module class *ModuleA*. This port depends on the interface of channel defined by channel class *ChannelA*.

In SLOOP, we employ the role model [10] to describe the structure of function model and performance model. We call it *structure diagram*. Figure 4 shows an example of structure diagram that describes the structure of modules.

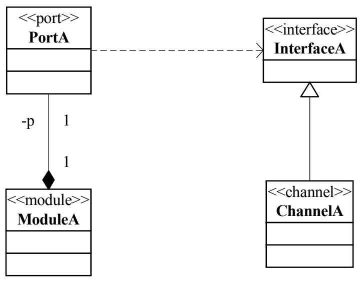

Figure 3. Modules, interfaces, ports and channels

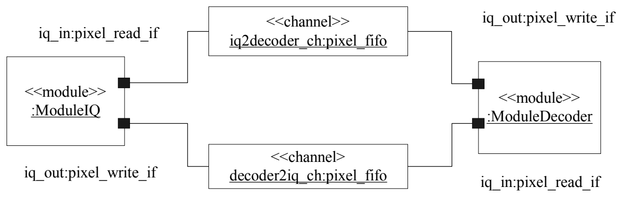

Figure 4. Structure Diagram

In structure diagram, objects are represented by the appropriate classifier roles - sub-module by module roles, ports by port roles and channels by channel roles. To reduce visual clutter, the port roles are generally shown in iconified form, represented by black-filled squares shown in Figure 4.

Using these notations, each model in SLOOP can be modeled by extension of UML. In next Section, we introduce application of SLOOP via an image decoding system.

3.2 Reusability of Models

For improving the reusability of each model, we introduce interface-based design methodology [14], namely, separating behavior from communication. In functional model, we define behaviors as processes (modules), communications as channels. The process and channel can be refined into performance model respectively. In performance model, we must evaluate various architectures within a short period, the reusability of models becomes very important to shorten the design cost. Interface-based design methodology can help designers to evaluate different communication channels without modifying behaviors.

The notions of interfaces and channels can help us to separate behavior from communication. The details can be found in [13] and [15].

Using interface-based design methodology, we can realize performance model easily by modifying functional model with less effort.

4. A CASE STUDY

We have applied SLOOP design process to an image decoding system. In conceptual model, we analyzed the decoding algorithm with Object-Oriented analysis techniques. We created functional model using Kahn Process Network (KPN) [12]. We selected bus architecture as an architectural model. After mapping functional model onto architectural model, we evaluated throughput of image decoding system to explore an adequate architecture using simulation.

4.1 Conceptual Model

The conceptual model of image decoding system is started with the use case analysis to find the boundary of target system and associate the external stimulus 'actors' and 'use-cases' of system via the use of relationships. Figure 6 indicates a use case diagram of image decoding system. In this case, there is only one use case "Decode Image" and three actors. In conceptual model, class diagrams are employed to describe the data structure, and sequence diagrams are used to represent scenarios of use case. Because the approach of analysis is same as that of application software, we skip the explanation of the details.

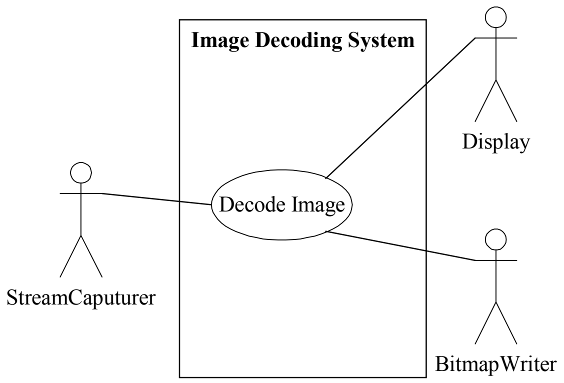

Figure 6. A Use Case diagram of image decoding system

4.2 Functional Model

In the functional model of image decoding system, we employ KPN model of computation. In the KPN model, parallel processes communicate via unbounded FIFO channels. The function in conceptual model is partitioned into processes that communicated each other via unbounded FIFO channels. Each process performs sequential computation on its private state space. Figure 7 illustrates an example of the KPN. Process P1 and Process P2 communicates with an unbounded FIFO channel, which has a single write port and single read port. The KPN fits nicely with signal processing applications as it conveniently models stream processing and as it guarantees that no data is lost in communication [16].

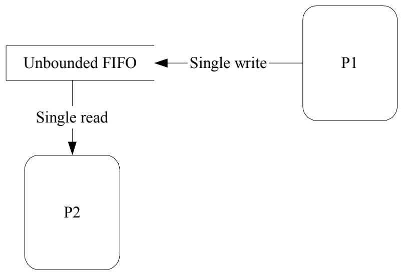

Figure7. Kahn Process Network

For modeling the functional model of the image decoding system, we used the structure diagram of UML to describe the KPN, and implement it as an executable model with SystemC. Figure 8 depicts the structure diagram of the image decoding system. The image decoding system has seven modules drawn with solid-outline rectangles. Modules are connected by unbounded FIFO channels drawn with solid line. Each module has ports notated with black-filled squares.

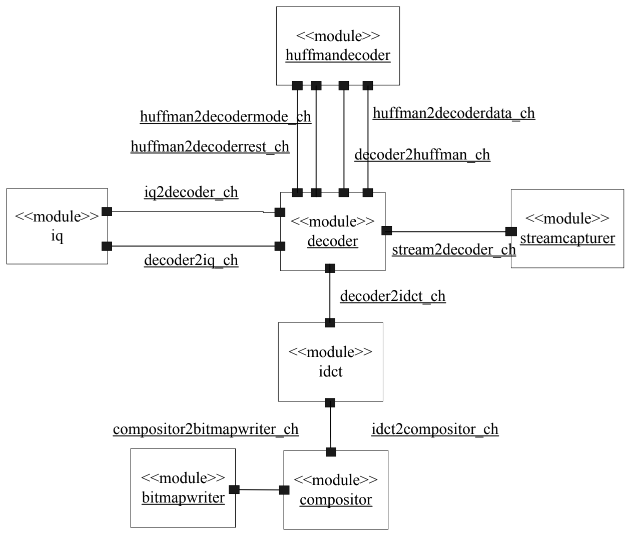

Figure 8. The Structure Diagram of image decoding system

The functional model is also implemented with SystemC as an executable model. The computation workload and communication workload can be obtained by execution of the functional model. Table 1 shows the computation workload, and Table 2 shows the communication workload of the functional model of the image decoding system. The computation workload and communication workload help designer to decide the initial parameters of the architectural model.

Table 1. Computation Workload

Module	Operation	Number of invocation
streamcapturer	Processing stream data	8,100
decoder	Processing stream data	8,100
	Processing VLD data	119,596
	Processing IQ data	119,596
…	…	...

Table 2. Communication Workload

Channel	Token	Number of tokens
stream2decoder_ch	EncodedBlock	8,100
decoder2huffmandecoder_ch	BitVector	13,585
...	…	…

4.3 Architectural Model

In this case, we select an architecture that consists of a processor, five hardware modules and a RAM to implement the functional model of the image decoding system. The processor has a priority-based scheduler as a multi-task scheduler when more than two tasks are mapped. We select a 32 bits bus to connect these components. For the shared memory we selected an SRAM-type memory of size 32KB with read and write latency of 10 ns respectively. The architectural model is modeled using deployment diagram of UML shown in Figure 9. The node notation of UML describes components of the architectural model. The parameters of each component are defined with attributes of each node shown in Figure 9.

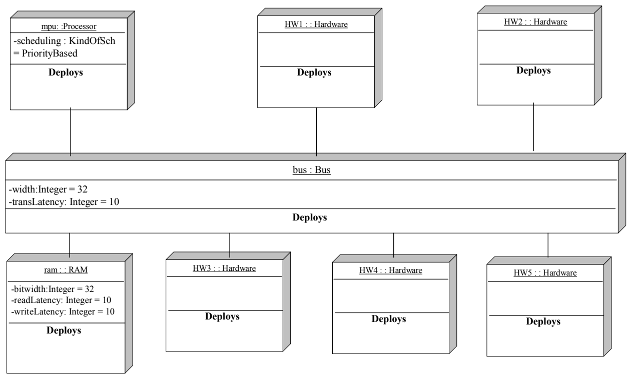

Figure 9. Deployment Diagram of Architectural Model

4.4 Performance Model

We map the functional model shown in Figure 8 onto the architectural model shown in Figure 9 in order to obtain the performance model of image decoding system. Figure 10 shows the result of mapping with deployment diagram of UML. Each module of the functional model is mapped onto the node of the architectural model. The run-time is added into the behavior of each module. In this case, the run-time is given from designers and implemented by the "*wait (run-time)*" statement using SystemC. The unbounded FIFO channel of the functional model must be transformed into the bounded FIFO by fixing its size. In this case the channels are realized with bus and RAM.

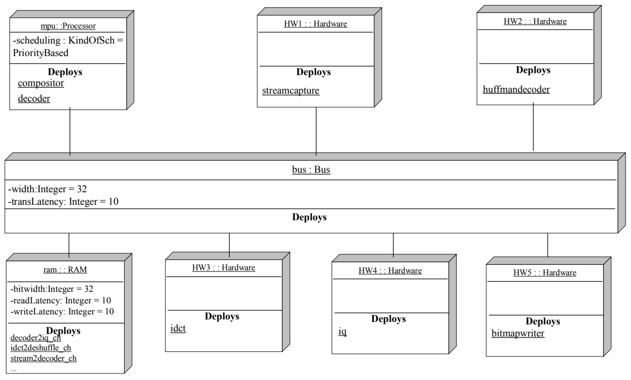

Figure 10. Mapping Functional Model onto Architectural Model

After implementation of the performance model with SystemC, we can obtain the performance evaluation results to analyze that the selected architecture is satisfied the performance requirement or not. If not, the bottleneck analysis can help us to improve the selected architecture.

4.5 Performance Evaluation Results

In this section we present the experimentation and explain how SLOOP help us to evaluate the performance of alternative architectures effectively.

In the experiment, the 'throughput' is a performance metric for evaluating the image decoding system. We measured this throughput with frames per second. The requirement of performance is 30 frames per second.

Experiment 1: We mapped the module 'decoder' and the module 'compositor' to processor, and other modules onto hardware components shown in Figure 10. The communication channel between modules was realized with RAM and bus. The parameters of each component were set shown in Figure 9.

The simulation results show the average throughput is 12 frames per second that was well below the required throughput of 30 frames per second. Figure 11 shows the ratio between the execution time and the I/O wait time of each module in performance model. Figure 12 indicates the utilization of each communication channel between processes. The bus utilization is 57%. The focus of this experiment is thus on improving the performance. Figure 11 shows the execution time of the module 'decoder' and 'compositor' is very longer than other modules and Figure 12 indicates that channels connected with the module 'decoder' and the module 'compositor' take very long time for reading data form the FIFO channel. These mean that the run-time both of 'decoder' and 'compositor' bottleneck the throughput of system.

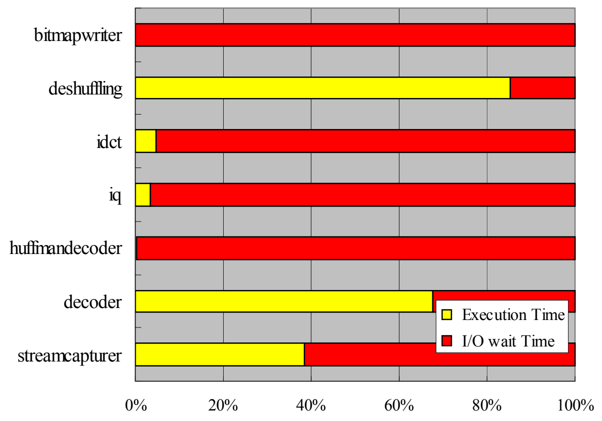

Figure 11. Utilizations of modules in Experiment 1

Experiment 2: For improving the throughput of experiment 1, we decided to remove the module 'decoder' from processor and map it onto a new hardware component to shorten the run-time of 'decoder'. We evaluated this modified architecture in order to see how the performance had changed. The throughput was improved to 24 frames per second but did not satisfy the requirement yet. The bus utilization was 73%. By analyzing the experimental results, we found that the bus was the bottleneck.

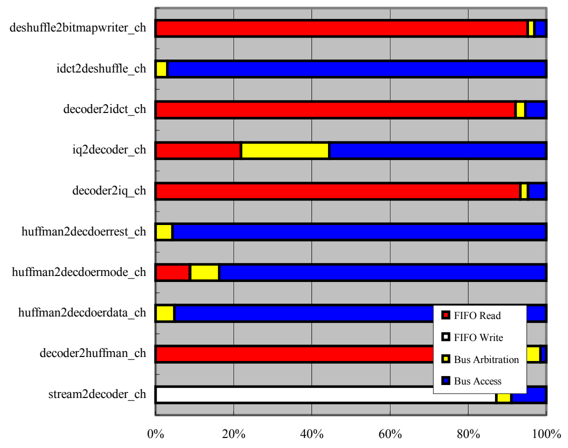

Figure12. Utilization of channels in Experiment 1

Experiment 3: We connected 'decoder', 'huffmandecoder', and 'iq' directly without utilizing bus. Furthermore we expend the bus width to 64 bits. As the experiment result, the throughput rose to 30 frames per second and the bus utilization was down to 50%.
According to the results of the exploration we decided to select the architecture shown in experiment 3 to implement.

5. CONCLUSIONS

In this paper, we have presented SLOOP design process for SoC development. We demonstrated a case study of an image decoding system application to show the effectiveness of SLOOP design process.

There are two key technologies in SLOOP shown in this paper. One is system level performance evaluation methodology. This can help us to explore the architecture at system level in order to avoid the performance risk. Another key technology is modeling with UML. For modeling results of analysis and design in SLOOP, we extended the standard UML using stereotype mechanism of UML.

We also demonstrated how to establish conceptual model, functional model, architectural model and performance model via an image decoding system. The experimental results showed the adequate architecture could be found easily through performance evaluation of performance model.

In future work, we will establish the verification process and develop the tools that are necessary in SLOOP.

6. REFERENCE

[1] F. Balarin, A. Sangiovannni-Vincentelli al. *Hardware-Software Co-design of Embedded Systems – The Polis approach*, Kluwer Academic Publisher, 1997.

[2] Cadence Virtaul Component Co-desing (VCC), http://www.cadence.com/datasheets/vcc.html

[3] P. Lieverse, T. Stefanov, P. van der Wolf, Ed Deprettere, "System Level Design with Spade: an M-JPEG Case Study," *IEEE/ACM International Conference on Computer Aided Design ICCAD2001*, pp26-32, November 2001.

[4] J. Rumaugh, M. Blaha, W. Lorensen, F. Eddy. *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

[5] OMG home page, http://www.omg.org/

[6] D. Verkest, J. Cockx, F. Potargent, G. D. Jong. "On the use of C++ for system-on-chip design", *IEEE Computer Society Workshop on VLSI'99*, Orlando, Florida, April 8-9, 1999.

[7] Baudoin, Claude & Hollowell, Glenn. *Realizing the Object-Oriented Lifecycle*. Upper Saddle River, NJ: Prentice Hall, 1996.

[8] Stan Liao, Steve Tijiang, Rajesh Gupta, "An Efficient Implementation of Reactivity for Modeling Hardware in the SCENIC Design Environment,", *proceedings of the Deisgn Automation Conference DAC'97*, pp.70-75, June 97.

[9] Rational Rose RealTime Homepage, http://www.rational.com/products/rose/real_time/rtrose.jsp

[10] B. Selic and J. Rumbaugh, "Using UML for Modeling Complex Real-Time Systems", white paper, Rational, March 11, 1998.

[11] G. Martin, L. Lavagno, J. Louis-Guerin. "Embedded UML: a merger of real-time UML and co-design". http://www.gigascale.org/pubs/101.html, March 2001.

[12] G. Kahn, "The semantics of a simple language for parallel programming," *Proc. of the IFIP Congress 74.* 1974.

[13] FUNCTIONAL SPECCIFICATION FOR SYSTEMC2.0, http://www.systemc.org

[14] J. A. Rowson, Alberto Sangiovanni-Vincentelli, "Interface-Based Design", *proceedings of the Deisgn Automation Conference DAC'97*, pp178-183. June 97.

[15] D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, S. Zhao, *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, 2000.

[16] P. Lieverse, P. Wolf, E. Deprettere, "A Trace Tansformation Technique for Communication Refinement", Proc. 9th Int. Symposium on Hardware/Software Design, Copenhagen, Denmark, April 2001.