Verification of System coherency at early Architecture Design Stage

Antoine Delautre (Thales Communications, Colombes, France, antoine.delautre@fr.thalesgroup.com)
Jean-Etienne Goubard (Thales Communications, jean-etienne.goubard@fr.thalesgroup.com)
Guy Gogniat (Lester, Lorient, France, guy.gogniat@univ-ubs.fr)
Christophe Moy (Mitsubishi Electric ITE-TCL, Rennes, France, moy@tcl.ite.mee.com)
Nicolas Bulteau (Softeam, Rennes, France, nicolas.bulteau@softeam.fr)

ABSTRACT

This paper intends to demonstrate the feasibility of performing non-functional coherence verification of software radio architecture specifications and application requirements with UML based models, before beginning any development phase. Right at the UML modeling step, it will be possible to investigate the array of potential solutions enabling selection of some by verification of the coherency. This approach will enable design cost saving by drastic reduction of time and minimization of the number of prototypes. Moreover, it will enable impact analysis of software or hardware architectural modifications with regards to the system functional and non-functional requirements. The modeling of software radio application and execution platform is presented using UML activity and deployment diagrams. The mapping of software components on hardware components is also specified with these diagrams. The attributes required to perform non-functional verifications are highlighted. Finally coherence and performance constraints to be verified in order the meet quality of service for software radio application are presented.

1. INTRODUCTION

One of the main objectives of software radio is to enable the development of waveform components independently of the hardware on which they will be executed. Even if this allows the functional portability, this does not guaranty at any cost that the waveform components will behave optimally and identically on two different platforms in term of quality of service. One of the main challenges will be to verify during the design phase (and more especially on the different models described) if a waveform application can be executed on a specific platform in the required level of quality of service: execution time, latency, memory, ….

In the A3S approach, we highlight the constraints composition part, focusing on a heterogeneous Digital Signal Processor (DSP) and Field programmable Gate Array (FPGA) based system, and we place our interest in the verification at an early stage. We will thus propose an extension of the software radio profile elaborated by the OMG, allowing to design some software radio platform specific and independent models that take into account non-functional and quality of service aspects of the components behavior.

The paper is arranged as follows: Section 2 reviews existing methods and tools to improve the design cycle. Section 3 focuses on UML, it highlights the benefits of using UML and its associated profiles to perform the design of software radio systems. Section 4 provides the information related to PSM in the context of MDA technology. In this section a brief presentation of the attributes required to characterize and to verify software radio systems are exposed. The UML models that we propose are also presented. Section 5 deals with constraints verification in order to validate a design at each step of the specification. Section 6 introduces the A3S project and gives its expected contributions and finally section 7 concludes the paper.

2. TRENDS TO IMPROVE TIME AND MONEY EFFICIENCY FOR SOFTWARE RADIO DESIGN

2.1. Methods of performances verification

Designers have access to plenty of high-level functional simulation tools: C, C++, Matlab Simulink, Synopsys CoCentric Studio... but almost exclusively processor dedicated. Possibility exists to deal with some kind of heterogeneity, involving FPGA but it is quite limited. For instance, Simulink and Xilinx propose pre-developed equivalent algorithms for both processor and FPGA, which support so that the simulated results are effectively coherent with the implemented version. But this is limited in terms of the variety of algorithms and the hardware targets (components and platforms).

SystemC tries to impose itself as a standard for heterogeneous co-design. But it is only usable for functional co-simulation and not for developing nor for debugging environment. Moreover, synthesis from SystemC is not completely validated. If SystemC is convenient for functional simulation and verification, there is a break in the design flow to go through the implementation on embedded platforms, which are often multi-processing and heterogeneous or software radio. This break implies risks of mistake and the impossibility to re-introduce in the high-level design modifications done at embedded domain level. This impacts in term of development time, design quality and consequently cost.
Other ways have been investigated. The solution to directly work on the embedded system level itself is rejected because of the complexity and non-effective way of working it imposes. The above reasons lead to the conclusion that the high-level CAD tools have to be enhanced to encompass the multi-processing

heterogeneous domain at both functional and non functional level.
Either it will be integrated directly in the simulation environment, or as an independent tool, with possibility of a direct bidirectional bridge with the simulation tool.

2.2. Available tools and their use

Only some academic tools currently try to offer solutions to the issue. Several of them are discussed here.
At a platform level, CoFluent[1].Studio enables developers to capture and verify a fully-timed functional model of their system through graphics and C code, and explore various heterogeneous architectures.
The SoC Environment (SCE)[2] provides an environment for modeling, synthesis and validation. SCE represents a flow that allows designers to capture system specification as a composition of C-functions. It is based on a set of tools to facilitate the design flow and perform refinement steps.
SynDEx[3] extracts itself the potential parallelism available in a system, based on measurement or estimation of the algorithms execution and communication timing. It makes scheduling and partitioning for multi-processing platforms involving DSP, GPP and FPGA (currently only as co-processors), and automatically generates the corresponding code for each of the processors (C or assembly for processors, VHDL for FPGA). The tool tries to optimize the processing efficiency and the user checks if it respects its constraints (in term of execution duration here)
At the level of a FPGA, GAUT[4] generates a pipelined architecture implementing the VHDL-expressed digital signal processing function under timing constraints and a target technology. GAUT extracts the potential parallelism at the behavioral level e.g. before selecting, allocating, assigning, and scheduling hardware operations. It generates a processing unit containing the arithmetic and a control unit piloting the processing unit.

2.3. Return on investment

The concept of the a priori system coherency verification is to ease and accelerate the design phase, a simple solution is to use the most known modeling language in order to facilitate the control of the language. For that reason we will address UML[5] because it is a well-established standard, whose concept was proved on most of project designs, and is understood by a huge community of designers and developers. A tool dedicated to the system design and verification may be of great interest because it may guide the designer in its conception thankfully to some automations. So the gain is twice: sparing time for modeling language control, and for design time (due to the tools automations). Moreover UML is a formal language enabling refinement of the description of the system up to code generation. Main tools provide code generation for C/C++, Java and also some dedicated language through bridge or plug-in like ESTEREL [6].

3. UML, AN ORGANIZED FAMILY OF STANDARDS

3.1. UML in global projects conception

UML is a very commonly used language in the system design process, but the diagrams it allows to take into account are differently used.

- The Use Case diagrams are frequently used to express the user’s need. They describe the actors interacting with a system and the main use cases of these actors on the system.
- The sequence diagrams allow to describe the interaction sequences between many objects.
- The Collaboration diagrams, often used conjointly with objects diagram, allow to describe the objects, their links, values and cooperations.
- The Class diagrams are the most used in UML. They allow to describe the main notions of a system, their links, their attributes and their classifications. Used at all development steps, they allow to obtain some conceptual, architectural or close to code diagrams.
- State and Activities diagrams are used to describe the dynamicity of a system. While State diagrams are mainly used for technical specification of real time applications, Activity diagrams allow to model the process described by activity chains with information transmission, connection management, and activity responsibility description. They are used in tertiary applications, where process description is primordial, but also in the technical and real time application world

3.2. UML profiles for extending models to specific domains

UML profiles enable to specialize UML for each work context by introducing some notions more adapted to the current work. An UML profile regroups coherently the extensions of the UML model and defines their coherency rules. There may be dependencies, inheritances, or groupings between UML profiles, the main interest being the reuse of domain specific notions in a standard way. Some standards profiles are emerging, each of those being an UML profile dedicated to an application domain or a technical environment.

The Real time, Scheduling and Performances profile[7] which describes the characteristics, is an OMG standard, focalized on the representation of properties bound to the time, like the duration, the performance and the planning. The goal of this profile is to allow the description of these temporal properties and to be able to predict the temporal aspects of the software before any development. Figure 1 shows the meta-model described into the real-time profile for non-functional characteristics representation.

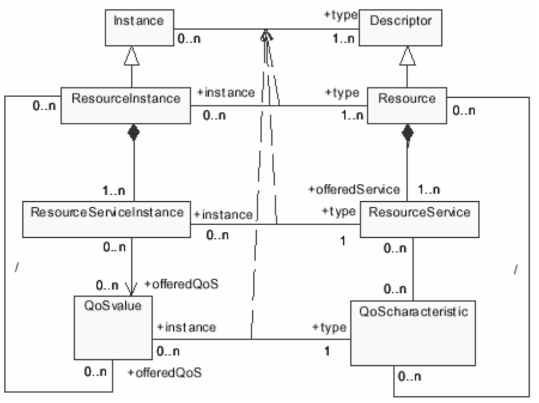

Figure 1: meta-model for non-functional characteristic representation

The UML profile for software radio[8] is in elaboration inside the OMG, it shall allow the description of the technology used in the software radio of the PIM model form, which specifies the interfaces between the wave form components and an environment constituted of radio devices (amplifier, antenna), radio services (filters, converters), radio management components (channels assignation) and operating system. Figure 2 describes one of the meta-models introduced by the software radio profile.

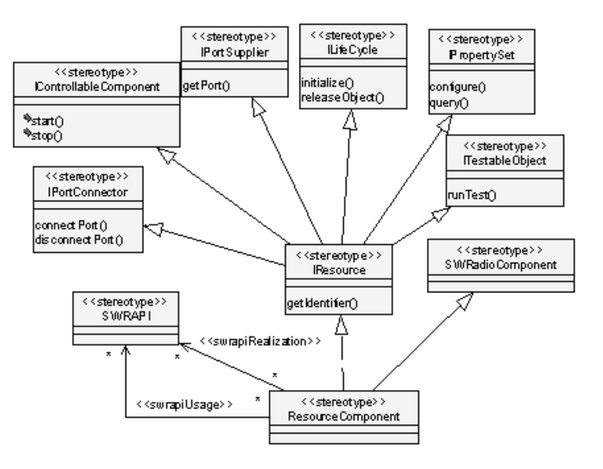

Figure 2: meta-model for software radio resource representation

3.3. Current UML place in the software radio

Currently, major software radio projects are described by UML class diagrams for architecture and sequence diagrams for chaining behavior. The UML profile for software radio proposes a set of stereotypes for allowing the description of platform independent or dependent architectures of radio systems on a functional view side. To allow the fine definition of signal processing behaviors, it can be extended by the additions of stereotypes coming from QoS profile[9] and Real Time profile on each of the components addressed by the Software Radio profile, describing their quality of service behavior offered or desired. In order to address the DSP/FPGA specific domain of study, it can be possible to extend the software radio profile by introducing DSP and FPGA stereotypes representing DSP and FPGA components derived from the processor stereotype of the software radio profile. These new stereotypes will be then tagged with stereotype extracted from the QoS profile to describe the quality of service behavior of the DSP and the FPGA. In the software radio profile and real time, performance and scheduling profile, some aspects are described in a CCM (Corba Component Model) view, allowing to perform Corba [10] implementations.

3.4. Legacy Component

Using standardized profiles and the components they introduce, we will allow the designer to reuse some legacy components by wrapping them into a standard component exhibiting the compliant interfaces. It will make the designer possible to focus on the architecture or system composition, instead of being compelled to discover and/or create new components from scratch. This method is already used for a long time by software developers to de-couple from third-party provided components. Such an approach is the only way to enable a smooth transition from existing methods to new one. It also makes possible the integration of non-compliant external provided component.

4. UML MODEL BASED OF VERIFICATION OF COHERENCE ARCHITECTURE - APPLICATIONS, THE CONCEPT

4.1. The MDA technology in software radio design

MDA allows to integrate all the middleware technologies (such as CORBA, EJB, XML, SOAP, .NET), the languages and the type of applications, federating them around the application model.

The MDA principle consists on defining some domain adapted models, independent of the implementation technology, called PIM (Platform Independent Model), and to transform these models in some more specialized models closer to the technology (PSM = Platform Specific Models) until being able to produce automatically the final code. The Software Radio profile of the OMG is looking forward this way in order to specify some stereotypes that may be used to produce software radio PIM and PSM models. One of the goals of A3S is to apply non-functional description elements on PSMs and on PIMs and to confront them during the verification phase.

4.2. Modeling of hardware characteristics and application requirements

Signal processing engineers use functional simulation tools to design radio systems. This approach enables to verify the algorithms consistency and accuracy for a dedicated situation of radio transmission. This is usually done in a high-level language like C or C++. But the implementation complexity associated with these algorithms is often not considered, neither the potential targeted device that should be considered to run the algorithm (DSP, FPGA, ASIC...).
As software radio offers a quasi-direct access from simulation to implementation on a hardware embedded platform, A3S aims at adding these considerations into the design flow. Concretely, this consists in adding to each of the algorithms that have been functionally checked, non-functional characteristics depending on the hardware target it will be supposed to run on. These non-functional characteristics are numerous and a choice has to be made.
Note that several solutions of implementation (DSP, FPGA, ASIC) can be considered for each algorithm, as algorithms are built in software components they can be easily moved on the multi-processing architecture. We can briefly give examples of what kind of parameters are of interest for software and hardware components.

Concerning the software components, whatever the nature of the hardware device that will execute the algorithm, some non functional parameters are necessary like I/O data type or repetition of the algorithm.
But some characteristics are more or less important regarding the device on which the algorithm will be executed. For example, for algorithms running on DSP, characteristics like code size, data size, execution time) are of major interest, but for algorithms executed on FPGA, characteristics such as gate occupation, internal dedicated RAM block occupation and internal dedicated multipliers block occupation will be more relevant.

Concerning the hardware components, different characteristics will be used regarding the hardware device we use. For example a DSP will be described at least by the clock frequency, code memory size, data memory size, co-processor number and nature, I/O ports number and nature, DMA..., but an FPGA will be described at least by its gate availability, internal dedicated RAM block availability, internal dedicated multipliers block availability, and clock frequency.
But communication means (and external memory) like FIFO, bus, dual-port memory, DMA, must not be forgotten... and can be described by their data throughput, size, and direction Another point to be considered is the middleware of the platform which is a part of the system itself, and may have influence on the system performance. Its non-functional aspects can be described, through the following criterias :

- Board Support Package, RTOS, for which services, support of Java Virtual Machine, CORBA support...

4.3. UML representation

A two diagrams form representation has been selected to specify the application and the platform. This couple can support the entire description of both software and hardware characteristics in an integrated and clear manner. This could be rapidly described as a hardware graph that deals with the platform and a software graph dealing with the application.
The hardware graph, which is a UML 1.4 deployment diagram, describes the physical connections that exist between the hardware devices located on the platform. The parameters of each hardware component are filled as shown by the Figure 3.

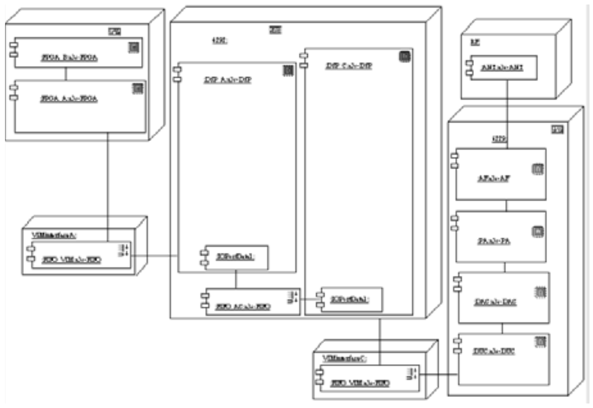

Figure 3: deployment diagram - platform description

The software graph of Figure 4, which is an UML 1.4 activity diagram, addresses the logical links between the different software components that constitute the system radio functionalities. It includes the non-functional characteristics that are independent of the nature of the hardware device that will execute the application (for example the number of I/O of a software component, its period and its number of execution).

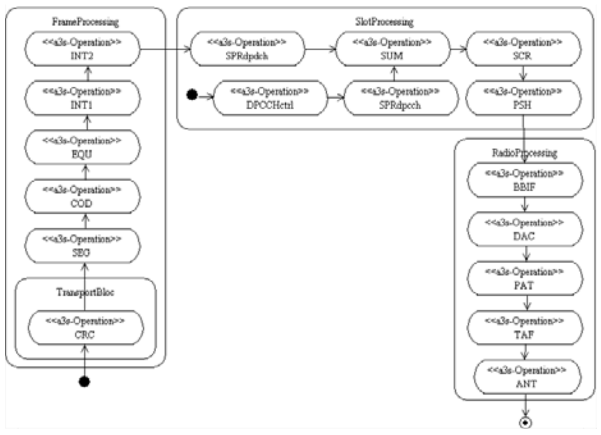

Figure 4: activity diagram for a UMTS transmitter - application description

The targeted hardware devices for a software component are defined in a table where each instance of the software component of the activity diagram is described. This allows to fill the parameters which are dependent of the hardware devices concerning each software component. Moreover, this approach enables to highlight on the deployment diagram the repartition of the software components on the hardware platform (Figure 5).

At each step of the design flow: application specification, platform specification and hardware-software mapping the designer needs to verify the coherency and the performance of his solution. These informations are provided through a constraints composition approach as resented below.

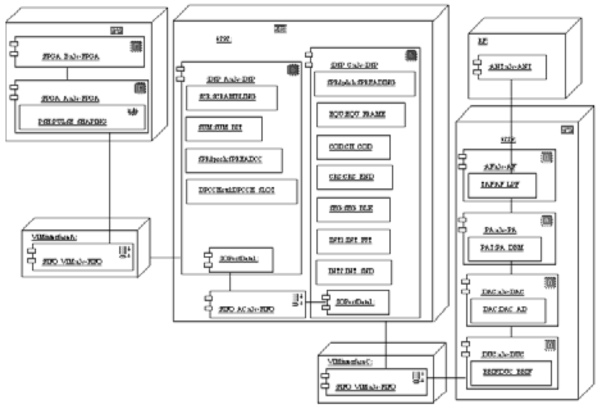

Figure 5: deployment diagram - platform description with software components

5. CONSTRAINTS COMPOSITION

Any system is subjected to a set of constraints, depending of the application that must be carried out and of the resources used for its implementation. Different types of constraints must be considered in order to validate the quality of service of a system: functional and non-functional constraints. Functional constraints verification leads to verify (mainly by simulation or execution) that the application well realizes the computation for what it was conceived. Non-functional constraints verification, aims to verify (mainly by simulation, prototyping, analytically or formally) that the system satisfies constraints related to its coherency and its performances (mainly timing, area and consumption). In the following, we focus on the non-functional constraints for each step of the design flow (application, platform and mapping).

5.1. Software architecture verification

The software architecture modeling, which consists of determining and connecting software components, requires a phase of analysis to be able to validate the representation carried out, as errors can occur in the representation. It is thus necessary to be able to identify and to provide them to the designer. These errors can be related to the symmetry of the inter-connected functions. A certain number of constraints of coherence must thus be analyzed. For example, it is necessary that, on both sides of inter-connected software components, the interface as well as the type of the exchanged data are identical (throughput, width of data).

Modeling must also bring answers in term of feasibility. Hence, with the attributes, on which we rely once the application is specified, it is possible to check if the model of execution and the coherence of the periods of the various software components are corrects. The constraint being that the interdependent functions should not overlap, otherwise system dysfunctions will occur. Some components can also execute an operation depending on a signal resulting from another component, which results in constraints of synchronization that must also be taken into account.

The verifications done at the software architecture level are related to specification and execution model coherency.

5.2. Hardware architecture verification

The platform modeling is of the same kind than the application's one, except that instead of having a representation of the software components, it leads to the representation of the hardware components. Thus, the same kinds of verification are essential. In fact, structural constraints of the system are concerned since it is the platform coherency that needs to be highlighted. Hence, in a similar way, the good coherency of the types and connections must be ensured. It is also important to check if some entries are not connected to other entries without being connected to an exit. Architecture must be coherent. If not, this constitutes an error case.

5.3. Hardware-Software mapping verification

The mapping of a software architecture on a hardware's one enables to verify if the system meets the required performance constraints. Within this stage, all the attributes are fulfilled and the system can be validated. We can then have a large number of informations on the performances of the system. Those are obtained after the preliminary checking of the coherence's constraints since a software component that requests a certain quantity of resources cannot be implemented on a hardware component if this last one does not offer such resources.

As a software component placed on a specific hardware component do not have the same performances as if it is placed on another hardware component the evaluation of the system performances constraints must be carried out. The verification corresponds first to the execution time of the application, which is the combination of execution time[11-12] of each software component running on its corresponding hardware component. This step takes into

account the constraints of operation, communication, memory, OS and middleware. Furthermore, in a complex system where several functions can be carried out in parallel in a heterogeneous environment, it is important to provide informations on the resources used. Indeed, if the system does not respect the constraints, localization of critical points (overuse of a resource whereas others are available) are important in order to find solutions which may improve the system. The addition of traces goes in this direction. The traces give the temporal evolution of a certain number of selected parameters. It is thus possible to see the temporal resources occupation, the functions activation and duration.
The functions are also subjected to constraints of scheduling which depends on the architecture (interconnection and computation resources) and on the performances that the designer wants to obtain. They are also related to the constraints of data dependences. Indeed, an execution of a task cannot take place if the data that it requires are not available. It is thus necessary to identify the possible dependency constraints in order to try to reduce or eliminate their consequences by finding scheduling solutions that overcome the problem.
All these verifications enable the designer to validate early in the design cycle his hardware-software mapping. If constraints are not met he has the possibility to try another mapping or to define a new platform in order to converge rapidly to an efficient solution.
Virtual prototyping gives the possibility to check a priori the adequacy between the application and the platform. This is carried out by computing the performances of the system and by verifying the constraints thanks to the attributes identified at each level of the system's design. It is thus easier, due to the flexibility of the approach, to compare the effectiveness of an application being carried out on a platform, with the same one being carried out on another platform, by the simple fact of inverting a component by another. The same analysis can be performed to the application tested, where each software component can be replaced by another (for example a IIR filter can be replaced by a FIR filter). By this high level approach, as well the platform as the application can easily be modified without asking for a large effort of development.

6. THE A3S PROJECT

6.1. Goal of the project

The A3S Project responds to the priorities of the RNRT 2002 call for proposal, by highlighting the high level step of the systems design. It aims at realizing a tight coupling between the specification and the different constraints to enable an a priori verification of the software architecture adequacy to the target platform in the software radio domain, especially for the 3G and 4G telecom. Its goals are to promote the tools and development environment enabling:

- Virtual representation of heterogeneous reconfigurable systems (DSP, GPP, Logical devices),
- High level system design sustained by languages able of both software and hardware,
- Tight coupling with development environments.

6.2. Constitution of an UML profile dedicated to non-functional coherence verification

The capabilities presented above correspond to the basic tools needed to design any software radio system. Furthermore, it is the choice and the accuracy of the set of parameters associated with each hardware or software component that makes the design a success or not.
This is what aims at providing A3S: an effective profile for SDR.
Coupled with already existing efforts on the domain done at the OMG for instance around the Swradio DSIG and the SCA. An organization of SDR design around such a tool favors reuse and design time inside a company, as well as cooperation between companies using the same tool and method.

6.3. The Demonstrator

A UMTS-FDD communication system for dedicated channels has been chosen to investigate the requirements of such a design and to highlight the features expected from the A3S tool.
This example provides a complexity and a set of constraints high enough to be considered as a good evaluator of the A3S tool accuracy.
Confronting the tool to such a use case will prove the efficiency of the solution. It will concretely show to the software radio designers of 3G systems the benefits affordable with appropriate tools.

Major challenges offered by the use case are:

- processing speed demanding,
- embedded memory allocation to be optimized,
- hardware (reconfigurable or not) circuits surface to be evaluated,
- communications means between processing units dimensioning,
- embedded middleware to be chosen and associated overhead,
- taking into account power consumption constraints.

7. CONCLUSION

The project is already running for six months and strong progresses have been made, demonstrating the basis of our approach. The team involving some major radio manufacturers as well as a well-known academic research

team and a leader of UML workshop editors has achieved the definition of the basics of the case study. Work will be finalized in the next months and published for wide dissemination.
Our purpose is to expose the benefits of our research to the Software Radio community and to promote our results for standardization.
We propose to give our publishes and results to the SDR Forum and to collect their feedbacks for the benefit of the A3S project but also for the benefit of the SDR community.

ACKNOWLEDGEMENTS

This work has been performed in the framework of the RNRT French funded project A3S. The authors would like to acknowledge the contribution of their colleagues from A3S consortium.

REFERENCES

[1] CoFluent Design: http://www.cofluentdesign.com/
[2] A. Gerstlauer, R. Dömer, J. Peng, D. D. Gajski, *System Design: A Practical Guide with SpecC*, Kluwer Academic Publishers, Boston, MA, ISBN 0-7923-7387-1, June 2001.
[3] SynDEx: http://www-rocq.inria.fr/syndex/
[4] E. Martin, O. Sentieys, H. Dubois, J. L. Philippe, "GAUT, an Architecture Synthesis Tool for Dedicated Signal Processors", Proceedings of International EURO-DAC Conference, pp. 14-19, 1993.
[5] OMG Unified Modeling Language Specification, final adopted specification January 2002 UML1.4 OMG document
[6] http://www-sop.inria.fr/meije/esterel/esterel-eng.html
[7] UML profile for Schedulability, Performance, and Time Specification – ptc/02-03-02 OMG Adopted Specification
[8] UML profile for Software radio – OMG draft
[9] UML profile for Quality of Service and Fault Tolerance Characteristics and mechanisms – OMG revised submission
[10] http://www.corba.org
[11] Ali Dasdan, Dinesh Ramanathan, Rajesh K. Gupta
"Rate Derivation and its applications to reactive , real-time Embedded Systems"
Proc. the 35th Design Automation Conference (DAC) pp. 263-8 , June 1998
[12] Richard Gerber, Seongsoo Hong, Manas Saksena
"Guaranteeing Real-time Requirements with Resource-Based Calibration of Periodic Processes"
IEEE transactions on software Engeneering, July 1995.