UML 2.0 Profile for Embedded System Design

Petri Kukkala[1], Jouni Riihimäki[1], Marko Hännikäinen[1], Timo D. Hämäläinen[1], and Klaus Kronlöf[2]

[1]Tampere University of Technology,
Institute of Digital and Computer Systems
P.O. Box 553,
FI-33101 Tampere, Finland

[2]Nokia Research Center
P.O. Box 407,
FI-00045 Nokia Group, Finland

Abstract

Unified Modeling Language (UML) 2.0 is emerging in the area of embedded system design. This paper presents a new UML 2.0 profile - called TUT-Profile - that introduces a set of stereotypes and design rules for an application, platform, and mapping. The profile classifies different application and platform components, and enables their parameterization. TUT-Profile concentrates on the structure of an application and platform, and utilizes standard UML 2.0 for the behavioral modeling. The application is seen as a set of active classes with an internal behavior. Correspondingly, the platform is seen as a component library with a parameterized presentation in UML 2.0 for each library component.

1. Introduction

The development of modern embedded systems requires reducing the gap between traditional hardware and software designs. Unified Modeling Language (UML) is widely used in software development, but now it is emerging in embedded system design. Especially the latest release of the language, UML 2.0 with its extension proposals, brings several advanced features to support also this domain. UML 2.0 holds a promise of a general design language that can be understood by system designers as well as software and hardware engineers.

This paper presents a new UML 2.0 profile, called TUT-Profile. TUT-Profile defines a set of stereotypes for extending UML metaclasses as well as design practices to describe applications, platforms, and mapping of them. It is especially targeted to automated system implementation using only UML 2.0 description. For this reason, TUT-Profile is used with a set of tools as depicted in Figure 1. Tools include Telelogic TAU G2 and a new custom UML profiling tool. Currently the physical target hardware platform is composed of Altera NIOS soft core processors and HIBI interconnection network [5] on single FPGA. However, the profile will not limit the type of platform and its components.

In this paper, we focus on the TUT-profile and demonstrate its use with a real implementation. In the next section, we start with highlighting the use UML 2.0 in embedded system design and give a brief review of related research. In addition, the TUT-Profile based design flow is briefly presented. Section 3 gives a detailed description about the stereotypes of TUT-Profile. Section 4, in turn, demonstrates the TUT-Profile in the design and profiling of a Wireless Local Area Network (WLAN) terminal. Finally, Section 5 concludes the paper.

2. UML 2.0 in Embedded System Design

A set of UML key properties can be identified for embedded system design [7]. One of the major advantages is that UML is not only a single language, but it allows the creation of languages for different purposes. To adapt UML 2.0, for example to different application and platform domains, sophisticated extension mechanisms are provided by the language.

Extension mechanisms in UML 2.0 can be roughly divided into first-class and second-class extensibility. The first-class extensibility is handled through the Meta Object Facility (MOF). This approach allows *modifications on the existing metamodels* and *creating new metamodels* with no restrictions.

The second-class extensibility does not allow modifications on existing metamodels. Instead, it enables adapting metamodels on specific purposes by *extending existing metaclasses*. The adaptations are defined *using stereotypes*, which are grouped in a profile. A profile contains constructs that are specific to a particular domain, platform, or method.

TUT-Profile utilizes only the second-class extensibility, which is considered to be powerful enough for applying UML 2.0 for embedded system design. Compared to the first-class extensibility, the use of stereotypes is a more lightweight mechanism, as it maintains the basic concepts and properties of the

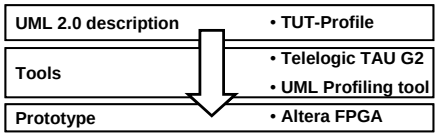

Figure 1. Design flow with TUT-Profile.

1530-1591/05 $20.00 © 2005 IEEE

standard UML 2.0. Consequently, this leads to better tool support.

2.1. Related Research

A number of extension proposals have been presented for real-time and embedded system design. The proposals can be roughly divided into three categories: *system and platform design*, *performance modeling*, and *behavioral design*. Next, the main related proposals are presented.

The Embedded UML [6] is a UML profile proposal suitable for embedded real-time system specification, design, and verification. It represents a synthesis of concepts in hardware/software co-design. It presents extensions that define functional encapsulation and composition, communication specification, and mapping for performance evaluation.

A UML Platform profile is proposed in [1], which presents a graphical language for the specification. It includes domain-specific classifiers and relationships to model the structure and behavior of embedded systems. The profile introduces new building blocks to represent platform resources and services, and presents proper UML diagrams and notations to model platforms in different abstraction levels.

The ACCORD/UML profile [10] defines a methodology for model mappings during the different development stages. ACCORD/UML proposes a real-time modeling based on the real-time active objects. The modeling features introduced by the ACCORD/UML method define a set of modeling rules involving model transformation techniques.

The UML Profile for Schedulability, Performance and Time (or the Real-time UML Profile) is standardized by OMG [8]. The profile defines notations for building models of real-time systems with relevant Quality of Service (QoS) parameters. The profile supports the interoperability of modeling and analysis tools. However, it does not specify a full methodology, and the profile is considered to be very complex to utilize.

The UML-RT profile [9] defines execution semantics to capture behavior for simulation and synthesis. The profile presents capsules to represent system components, which internal behavior is designed with state machines. The capabilities to model architecture and performance are very limited in UML-RT, and thus, it should be considered complementary to the Real-time UML profile. HASoC [3] is a design methodology that is based on UML-RT. It proposes also additional models of computation for the design of internal behavior.

These proposed profiles contain several features for utilizing UML in embedded system design, but they miss the completeness for combining application and platform.

2.2. TUT-Profile Approach

TUT-Profile extensions are used to define the structure and parameters of an application and platform as well as their mapping. Correspondingly, the system design is divided into three parts: *application description*, *platform description*, and *mapping*. Both the application and platform descriptions can be developed independently of each other.

TUT-Profile mainly concerns the structure of an application and platform. The application is seen as *a set of active classes* with an internal behavior. The platform is seen as a *component library* with a parameterized presentation in UML 2.0 for each library component. The profile does not restrict the behavioral modeling, and by default, it utilizes standard UML 2.0 concepts for this.

TUT-Profile classifies different application and platform components by defining various stereotypes and strict rules how to use them. The objective is to enhance the support of external tools for automatic analyzing, profiling, and modifying the UML 2.0 description of an embedded system. The classification also assigns defined parameters to proper components.

In practice, the profile is applied in a tool framework depicted in Figure 2. The platform mapping can be explicitly performed by the designer, or assisted with tools. For the latter, we have developed a UML profiling tool that combines the UML 2.0 description (model parsing) and simulation statistics (simulation log-file) that is obtained during the verification phase. Based on the profiling, also the application description can be modified for example to fulfill real-time constraints. When the verification is completed, executable application for the implemented platform is automatically generated from the UML 2.0 description.

3. TUT-Profile Stereotypes

TUT-Profile contains several stereotypes to support embedded system design. The structure of the TUT-Profile is presented in Figure 3. An *application* is composed of *application components*, which are instantiated as *application processes*. Next, application

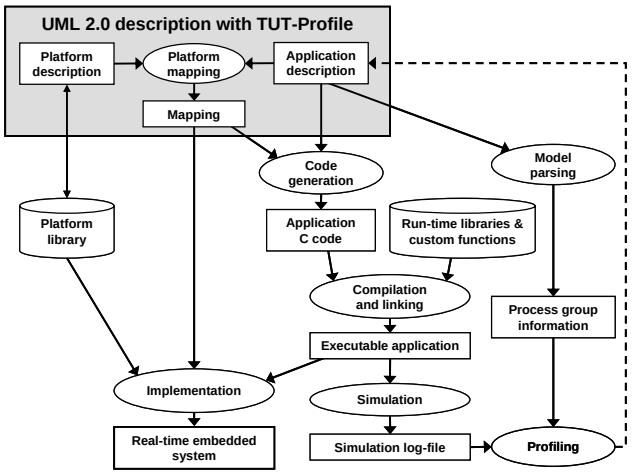

Figure 2. TUT-Profile design and profiling flow.

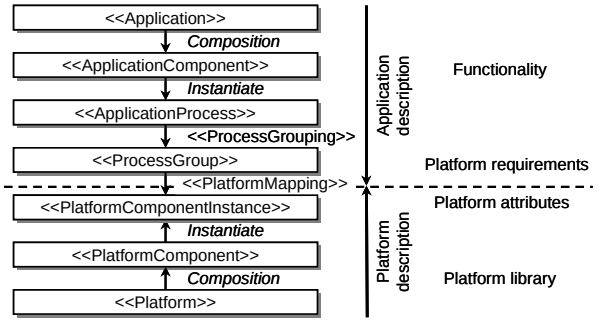

Figure 3. TUT-Profile hierarchy.

processes are grouped into *process groups*. Correspondingly, a *platform* is composed of *platform components*, which are instantiated as *platform component instances*. Finally, process groups are mapped to platform component instances using *platform mapping*. The summary of the profile stereotypes is presented in Table 1.

3.1. Application Description

Application stereotypes are applied to an application description containing a platform independent structure and behavior of an application. The application stereotypes mainly concern the structure and parameterization, as the TUT-Profile does not extend the behavioral modeling.

The application stereotypes classify the different types of classes and parts used in an application description. The parameterization of an application is performed using tagged values. The parameters may be manually added by a designer, or automatically derived from the behavioral part of the application description by the profiling tool.

Table 1. TUT-Profile stereotype summary.

Stereotype name (*extended Metaclass*)	**Description**
Application (*Class*)	Top-level application class
ApplicationComponent (*Class*)	Functional application component (active class, has behavior)
ApplicationProcess (*Structural feature*)	Instance of a functional application component
ProcessGroup (*Structural feature*)	Group of application processes
ProcessGrouping (*Dependency*)	Dependency between an application process and a process group
Platform (*Class*)	Top-level platform class
PlatformComponent (*Class*)	Defines features of a platform component
PlatformComponentInstance (*Structural feature*)	Instantiated platform component
CommunicationWrapper (*Dependency*)	Defines wrapper parameters of a communication agent
CommunicationSegment (*Structural feature*)	Interconnection structure of communicating agents
PlatformMapping (*Dependency*)	Dependency between a process group and a platform component instance

Table 2 tabulates the tagged values for each application stereotype in TUT-Profile.

The *stereotype* <<*Application*>> is applied to define the top-level class of an application description. The top-level class has a class hierarchy defining different classes of an application. The active classes having behavior are called *functional components*. Passive classes are *structural components*, which do not have behavior, but instead, define composite structures and *data structures* storing application data.

Process grouping is a part of an application description and it defines the structure for an application. The structure is implementation-oriented and may thus differ from the composite structure of an application. However, the grouping is platform independent as process groups do not represent any specific platform components.

The grouping can be performed according different criteria, such as the preliminary *scheduling* of application processes, *workload distribution*, *communication* between process groups, *dependencies* between process groups, and *size* of a process group (code size, memory requirements). The grouping is used for the *analysis* and *architecture exploration*, which can be performed using simulations or static analysis. The composed groups are mapped to platform components specified later.

The grouping can be fixed either at the level of a single process, or at the level of a process group. In the latter case, processes cannot be added or removed from a group. Currently, the grouping is done manually by the designer, but tools for automatic grouping according to the profiling information and process types will be implemented.

Table 2. Tagged values of application stereotypes.

Tagged values	**Description**
***Stereotype* <<*Application*>>**	
Priority	Execution priority of an application
CodeMemory	Required memory for application code
DataMemory	Required memory for application data
RealTimeType	Type of real-time requirements (hard/soft/none)
***Stereotype* <<*ApplicationComponent*>>**	
CodeMemory	Required memory for application component code
DataMemory	Required memory for application component data
RealTimeType	Type of real-time requirements (hard/soft/none)
***Stereotype* <<*ApplicationProcess*>>**	
Priority	Execution priority of application process
CodeMemory	Required memory for application process code
DataMemory	Required memory for application process data
RealTimeType	Type of real-time requirements (hard/soft/none)
ProcessType	Type of process (general/DSP/hardware)
***Stereotype* <<*ProcessGroup*>>**	
Fixed	Defines if the group is fixed (true/false)
ProcessType	Type of processes in a group (general/DSP/hardware)
***Stereotype* <<*ProcessGrouping*>>**	
Fixed	Defines if the grouping is fixed (true/false)

3.2. Platform Description

The platform description is more abstracted compared to the application description. The platform description is not targeted for functional synthesis of hardware, but it is used for composing a platform containing components using an existing platform library. In TUT-Profile, properties, capabilities, and limitations of a component are parameterized. The summary of tagged values in platform stereotypes is tabulated in Table 3.

The parameterized description models are used to perform a high-level hardware/software co-simulation. In that case, the execution of application processes is guided with the properties of the platform components.

The *stereotype* <<*Platform*>> is applied to define the top-level class of a platform description, composed of available platform components. Platform components and their parameterization can be refined by the further specialization of platform stereotypes.

In the platform description, the communication between active platform components, called *processing elements*, is performed via *communication elements*. Communication elements are implemented as *communication wrappers* that are used to connect processing elements to *communication segments*.

3.3. Mapping

When both an application and platform description have been defined, each group of application processes is mapped to a platform component instance. Mapping is performed by defining a dependency between a process group and a platform component instance. In order to account performance issues, the performance related parameterizations specified in the application and platform descriptions are combined.

The *stereotype* <<*PlatformMapping*>> is applied to describe how a process group is mapped to a platform component. When the mapping is fixed (indicated by a tagged value), it cannot be changed automatically by profiling tools during the design process. This can be used when a designer knows that a certain process group has to be placed into a certain processing element.

Table 3. Tagged values of platform stereotypes.

Tagged values	Description
Stereotype <<PlatformComponent>>	
Type	Type of a component (general/DSP/HW accelerator)
Area	Area of a component
Power	Power consumption of a component
Stereotype << PlatformComponentInstance>>	
Priority	Execution priority of a component instance
ID	Unique ID of a component instance
IntMemory	Amount of internal memory
Stereotype << CommunicationSegment>>	
DataWidth	Data width (in bits) of a communication segment
Frequency	Clock frequency of a communication segment
Arbitration	Arbitration scheme (e.g. priority or round-robin)
Stereotype << CommunicationWrapper >>	
Address	Address of a wrapper
BufferSize	Buffer size of a wrapper
MaxTime	Maximum time a wrapper can reserve the segment

4. TUT-Profile Case: WLAN Terminal

This section presents TUT-Profile utilization with the design of a custom WLAN terminal. The examples show how UML 2.0 diagrams are used with TUT-Profile, and how the stereotypes are applied in design.

The application is a custom Medium Access Control (TUTMAC) protocol [4] of a proprietary WLAN (TUTWLAN) [2]. The TUTMAC protocol has been modeled with UML 2.0 meeting the TUT-Profile, and the software parts of the protocol have been implemented with the automatic code generation for the platform. The platform is Altera Stratix FPGA with NIOS soft processor cores. In addition, the platform library contains implementations of some time critical algorithms, such as Cyclic Redundancy Check (CRC), that can be used for hardware acceleration of protocol functions.

4.1. TUTMAC Application Description

The design of an application description starts from the definition of the class hierarchy. The top-level application class and its components are created, and the associations between components are defined. A TUTMAC *class diagram* is presented in Figure 4.

The `Tutmac_Protocol` class is the top-level class of the application. Thus, it is stereotyped as <<Application>>. The class is composed of five classes. The `Management`, `RadioManagement` and `RadioChannelAccess` classes are functional components containing behavior, and can be instantiated as application processes. Thus, they are stereotyped as <<ApplicationComponent>>. The `UserInterface` and `DataProcessing` classes are structural components having composite structure without behavior. Consequently, they can not be instantiated as application processes, and are not stereotyped.

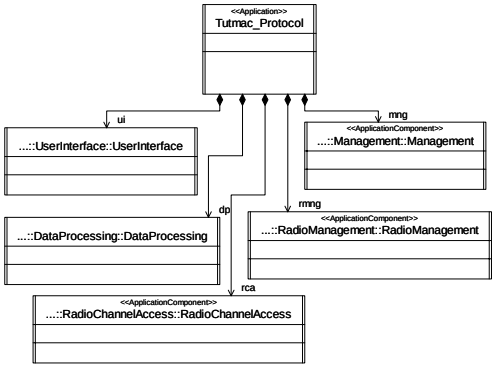

Figure 4. TUTMAC class diagram of an application description.

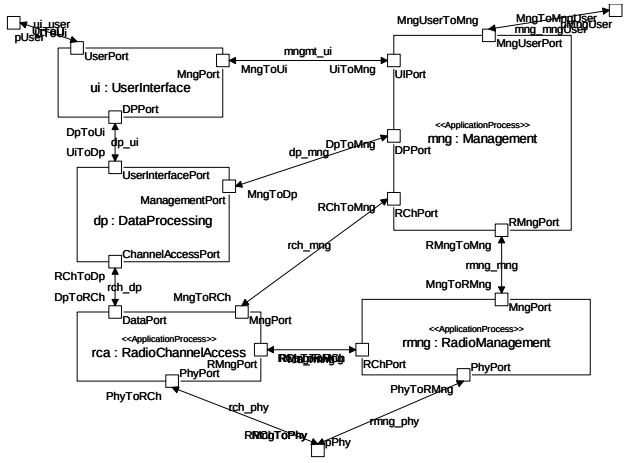

Figure 5. Composite structure diagram of `Tutmac_Protocol` class in the TUTMAC application description.

When the class hierarchy is defined, *composite structure diagrams* are used to describe the connections between parts (class instances). The parts communicate with each other by signals via their ports. Ports are connected by connectors carrying signals. The composite structure diagram of the top-level class `Tutmac_Protocol` is presented in Figure 5.

In Figure 5, the `mng`, `rmng` and `rca` parts are instances of the functional components stereotyped as <<ApplicationComponent>>. They represent the processes of the application, and are thus stereotyped as <<ApplicationProcess>>. `ui` and `dp` parts are instances of the structural components, and therefore they do not represent processes.

The structural components are hierarchically modeled using class diagrams and composite structure diagrams, until the behavior of the functional components can be expressed. The behavior of the functional components in TUTMAC is described using *statechart diagrams* combined with the UML 2.0 textual notation. TUTMAC statecharts are modeled as asynchronous communicating Extended Finite State Machines (EFSM).

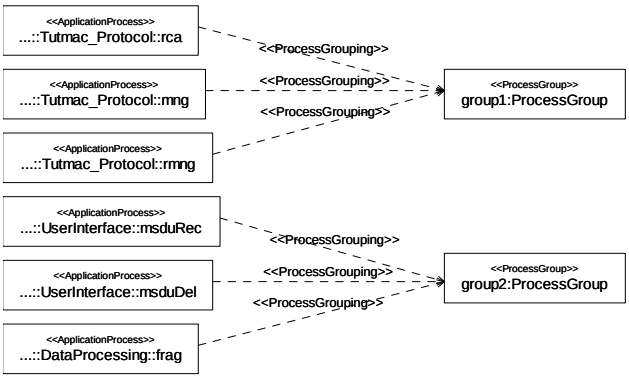

Figure 6. TUTMAC process grouping using composite structure diagram.

The TUTMAC application description is finished by grouping the main processes as depicted in Figure 6. The objective in grouping has been to minimize the communication between process groups, which enhances the performance if groups are mapped to different processing element.

4.2. TUTWLAN Platform Description

When describing hardware platform, the designer selects suitable components from the TUT-Profile library and connects components together. Figure 7 presents a platform description for the TUTWLAN terminal containing four processing elements connected with three communication segments. Three of the processing elements are similar NIOS processors. The fourth is a hardware accelerator for CRC-32 calculation.

The employed communication channel is a HIBI bus [5]. For HIBI, the platform stereotypes are specialized, thus including more detailed information of the platform specific communication elements. The specialized stereotypes are <<HIBIWrapper>> from <<CommunicationWrapper>>, and <<HIBISegment>> from <<CommunicationSegment>>. The specialized information contains sizes of buffers, bus arbitration, and addressing.

The communication between processing elements is carried out through a hierarchical bus structure. As presented in Figure 7, `processor1` and `processor2` are connected to the same bus segment called `hibisegment1`, which is connected to a bridge segment. The remaining two processing elements, `processor3` and `accelerator1`, are connected to `hibisegment2`, which is also connected to a bridge segment.

4.3. Mapping of TUTMAC to a Platform

The mapping in Figure 8 integrates the TUTMAC application description and the platform description of a TUTWLAN terminal. As seen, two process groups, `group1` and `group3`, are mapped to `processor1`. This indicates that the designer prefers the processes of the two process groups to be implemented on the same processor, although they were divided into separate process groups in the application description. `group4` has processes that can be implemented on an existing hardware accelerator, and thus, the process group is mapped to `accelerator1`.

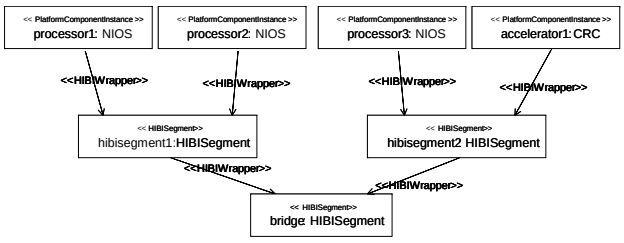

Figure 7. Stereotyped composite structure diagram for an example TUTWLAN platform.

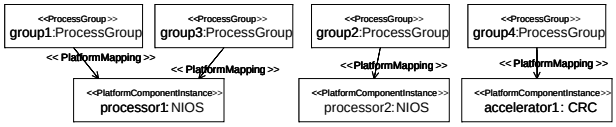

Figure 8. Mapping the TUTMAC protocol to TUTWLAN platform.

4.4. Profiling TUTMAC Description

The profiling tool implemented for TUT-Profile contains three main stages that are implemented as TCL scripts. First, the XML presentation of the UML 2.0 model is parsed to gather process group information from the model. Next, the automatically generated application code is complemented with custom C functions to create simulation log-file during simulations. These phases are shown in the design flow presented in Figure 2.

Finally, after simulation, the profiling data in the simulation log-file and the process group information are combined and analyzed. The results are gathered to a profiling report. An example profiling report based on the TUTMAC simulations on the workstation processor is presented in Table 4. The report contains the execution times of the process groups (a), and the amount of communication between the groups (b). In addition, other metrics, such as transfers between individual application processes, are also available.

The report is used for improving the application description. The process groups and mapping are modified to improve performance including amount of communication and the division of workload between application processes. Further, the profiling report can be used to refine the parameterization of the application description as the performance information is gathered with simulations on a reference platform, such as a PC workstation.

5. Conclusions

This paper presents the TUT-Profile for embedded system design with UML 2.0. The TUT-Profile concentrates on the structure and parameterization of an application and platform, and the mapping of these.

Table 4. A profiling report based on the TUTMAC simulations.

(a)

Process group	**Total execution time**	**Proportion**
Group1	13 942 856 cycles	92.1 %
Group2	792 056 cycles	5.2 %
Group3	375 372 cycles	2.5 %
Group4	36 092 cycles	0.2 %
Environment	0 cycles	0.0 %

(b)

Number of signals between groups					
Sender/Receiver	Group1	Group2	Group3	Group4	Environment
Group1	805	8	2	0	269
Group2	8	4	3	0	4
Group3	2	0	8	4	0
Group4	0	4	0	0	0
Environment	274	4	0	0	269

The case design with a TUTWLAN terminal demonstrated the use of TUT-Profile in practice. The example illustrated the use of UML 2.0 diagrams, and the applying of the stereotypes. TUT-Profile and the profiling tool were used to improve performance of TUTMAC by minimizing the communication between process groups.

In the future, the TUT-profile will further be developed by the specialization of the stereotypes and the improvement of parameterization. The profile will also be evaluated for multiprocessor System-on-Chip co-design environment. In addition, real-time operating system will be used in system processors, which will also be accounted in the TUT-Profile.

Acknowledgements

The research work for the TUT-profile and custom tools has been carried out as part of ITEA (Information Technology for European Advance) project “Prompt2Implementation” (ITEA 010009) at Tampere University of Technology (TUT).

References

[1] R. Chen *et al.*, “UML and Platform-based Design”. UML for Real: Design of Embedded Real-Time Systems, Kluwer Academic Publishers, May 2003, pp. 107-126.

[2] M. Hännikäinen *et al.* “TUTWLAN - QoS Supporting Wireless Network”. Telecommunication Systems - Modelling, Analysis, Design and Management, Kluwer Academic Publishers, Vol. 23, No. 3-4, July/August 2003, pp. 297-333.

[3] P. N. Green, and M. D. Edwards, “The Modelling of Embedded Systems Using HASoC”. Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE’02), March 2002, pp. 752-759.

[4] P. Kukkala *et al.*, “UML 2.0 Implementation of an Embedded WLAN Protocol”. Proceedings the 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, September 2004.

[5] E. Salminen *et al.*, "HIBI v.2 Interconnection for System-on-Chip," LNCS 3133 Computer Systems Architectures, Modelling, and Simulation, A.D. Pimentel, S. Vassiliadis, (eds), Springer-Verlag, Berlin, 2004, pp. 412-422.

[6] G. Martin *et al.*, “Embedded UML: a Merger of Real-Time UML and Co-design”. Proceedings of the 9th International Symposium on Hardware/Software Codesign, April 2001, pp. 23-28.

[7] G. Martin, “UML for Embedded Systems Specification and Design: Motivation and Overview”. Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE’02), March 2002, pp. 773-775.

[8] OMG, “UML Profile for Schedulability, Performance, and Time Specification”. September 2003.

[9] B. Selic, and J. Rumbaugh, “Using UML for Modelling Complex Real-Time Systems, White Paper, Rational”. March 1998.

[10] P. Tessier *et al.*, “A Component-based Methodology for Embedded System Prototyping”. Proceedings of the 14th IEEE International Workshop on Rapid Systems Prototyping, June 2003, pp. 9-15.