

Design and Implementation of a WLAN Terminal Using UML 2.0 Based Design Flow

P. Kukkala, M. Hännikäinen and T.D. Hämäläinen

Tampere University of Technology, Institute of Digital and Computer Systems,
P.O. Box 553, FIN-33101 Tampere, Finland
petri.kukkala@tut.fi

Abstract. This paper presents a UML 2.0 based design flow for real-time embedded systems. The flow starts with UML 2.0 application, architecture and mapping models for our TUTWLAN terminal with its medium access control protocol. As a result, the hardware/software implementation on Altera Excalibur FPGA is achieved. Implementation utilizes eCos real-time operating system, and hardware accelerators for time-critical protocol functions. The design flow is prototyped in practice showing rapid UML 2.0 application model modification, real-time protocol processing in an image transfer application, and execution monitoring.

1 Introduction

Several design frameworks have been proposed for high-level systems design to meet the challenges of ever-increasing system complexity. However, most of the approaches concentrate on a specific flow aspect, but rarely cover the whole flow from an abstract model to physical implementation.

UML 2.0 is converging on a general design language that can be understood by system designers as well as software and hardware engineers [8]. Many UML profiles have been proposed to adapt UML to this purpose. For instance, the UML Platform profile [2] introduces a way to model architecture resources and services, and the UML Profile for Schedulability, Performance, and Time [9] defines notations for building models of real-time systems with Quality of Service (QoS) parameters.

This paper presents a novel UML 2.0 based design flow that uses a custom UML profile, called TUT-Profile [7]. The flow comprises the UML application, architecture and mapping models, and includes automatic code generation, real-time execution monitoring, model profiling and performance back-annotation to the UML models. The main contribution of the approach are the methods to govern the whole flow using UML 2.0.

This paper shows the use of UML 2.0 in the implementation of a time-critical embedded system including both hardware and software. The target is the implementation of a proprietary TUTWLAN terminal on Altera Excalibur FPGA with an embedded processor core and eCos Real-time Operating System (RTOS). Rapid model modification is performed by changing the functionality of the terminal to illustrate the use of the design flow in practice.

The paper is organized as follows. First, TUTWLAN is introduced in Section 2. The design flow to implement the TUTWLAN terminal is presented in Section 3. Thereafter,

the implemented terminal, real-time execution monitoring, and rapid model modification are presented in Section 4. Finally, Section 5 concludes the paper.

2 TUTWLAN and the TUTMAC Protocol

TUTWLAN [6] is a proprietary WLAN developed at Tampere University of Technology (TUT). TUTWLAN solved the problems of scalability, QoS and security present in standard WLANs. The wireless network has a centrally controlled topology, where one base station controls and manages multiple portable terminals. Several configurations have been developed for different purposes and platforms. In this we present one configuration of the TUTMAC protocol.

TUTMAC is a dynamic reservation Time Division Multiple Access (TDMA) based Medium Access Control (MAC) protocol for TUTWLAN. The protocol supports negotiated QoS parameters for the data transfer, and is capable for managing the dynamically changing network topology.

The protocol contains functions for Cyclic Redundancy Check (CRC) and encryption. CRC is performed for headers with CRC-8 algorithm, and for payload data with CRC-32 algorithm. Encryption is performed for payload data using an Improved Wired Equivalent Privacy (IWEP) algorithm [5]. The algorithm encrypts payload data in 64-bit blocks, and uses an encryption key of same size.

The functions are performed for every packet sent and received by a terminal. Thus, their performance become significant, especially, when the data throughput increases and several packets are simultaneously processed by the protocol. Depending on the implementation, the algorithms may need hardware acceleration to achieve adequate delays for data. Further, the radio channel access has to maintain accurate frame synchronization in the TDMA scheduling, which sets tight real-time constraints and need for prioritizing the protocol processing.

3 UML 2.0 Design Flow for the TUTWLAN Terminal

The flow to design and implement the TUTWLAN terminal is presented in Fig. 1. The flow starts with three UML models for an application, architecture and mapping, which describe the TUTWLAN terminal as a whole. At least an application must be provided by a designer, but the architecture and mapping can also be produced using an architecture exploration tool, such as Koski [10].

The *application model* implements the functionality of the terminal. Software for TUTMAC is automatically generated based on the application model. External C functions can be included in the model if needed. Software is executed using an RTOS, and the thread configuration is retrieved from the *mapping model*. In this case, the thread configuration is fixed and manually retrieved, but the task can also be automated. The *architecture model* is composed of existing hardware components available in a library. In this case, the implementation is fixed, but the component selection can also be dynamic and automated.

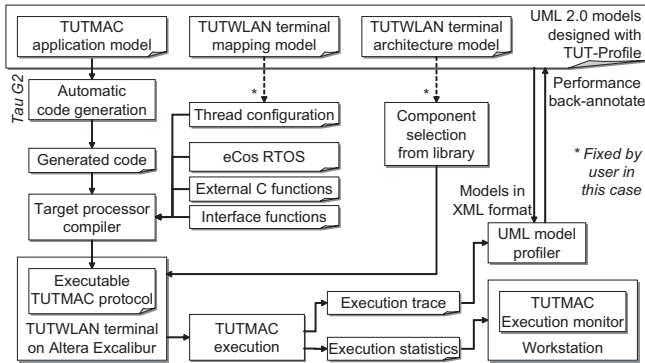


Fig. 1. UML 2.0 based design flow for the implementation of the TUTWLAN terminal

During execution, a trace and statistics are collected using custom functions. The model profiling uses the trace to back-annotate performance information to the UML models. Statistics are used for the real-time execution monitoring.

The terminal is implemented on Altera Excalibur FPGA on EPXA1 development board [1]. The FPGA contains an ARM9 processor and a Programmable Logic Device (PLD) connected by AMBA Bus (AHB) and dual-port memory. A 2.4 GHz MACless Intersil HW1151-EVAL radio transceiver is connected to the development board with an expansion header. On PLD, custom hardware accelerators and interfaces to access external devices are implemented using VHDL.

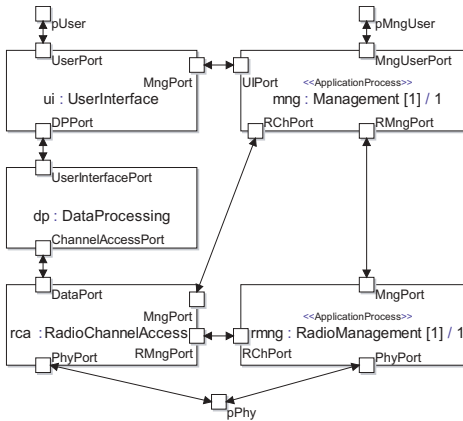
Telelogic Tau G2 [11] is used as an UML design tool. The tool also enables automatic C code generation for an application. The generated code supports POSIX standard for threaded execution of software. eCos RTOS [3] is utilized to the software execution. eCos provides a POSIX compatibility layer supporting the standard Application Programming Interface (API) to interface the kernel.

3.1 UML 2.0 Models

The UML models are constrained by our custom TUT-Profile [7], which is targeted to the design of real-time embedded systems using a model based approach. The profile defines a set of stereotypes for extending the standard UML metaclasses, and design practises to describe an application and architecture as well as their mapping. The stereotypes have tagged values for the real-time constraints.

TUTMAC Application Model. The class hierarchy of TUTMAC is designed using *class diagrams*. Thereafter, *composite structure diagrams* are used to describe the structure of the protocol in a more detailed way. The *parts* (class instances) communicate sending *signals* via *ports* and *connectors*.

The composite structure diagram of the top-level class Tutmac_Protocol is presented in Fig. 2(a). The *mng* and *rmng* parts are instances of the functional components, and they represent the processes of the application model. The *ui*, *rca*, and *dp* parts are instances of the structural components, which are further hierarchically modeled using class diagrams and composite structure diagrams.



(a) Top-level composite structure diagram

| Diagram type | Amount |
|------------------------------|--------|
| Class diagrams | 18 |
| Composite structure diagrams | 5 |
| Statechart diagrams | 41 |

| Property | Amount |
|----------------------------|--------|
| State machines (processes) | 20 |
| Ports | 52 |
| Signal types | 40 |

(b) Amount of diagrams and properties

Fig. 2. TUTMAC UML 2.0 application model with TUT-Profile

In TUTMAC, the behavior of functional components is expressed using statechart diagrams combined with the UML 2.0 textual notation. Statecharts are asynchronous communicating Extended Finite State Machines (EFSM) [4].

Figure 2(b) presents the amount of UML diagrams and main properties of the TUTMAC UML model to illustrate the size and complexity of the model.

TUTWLAN Terminal Architecture Model In this case, the architecture of the TUTWLAN terminal is fixed. An existing library contains UML models for the fixed components of Excalibur FPGA, including the processor, AHB and dual-port memory, as well as custom hardware accelerators and interfaces. The components are stereotyped using TUT-Profile to parameterize each component. The architecture model for the TUTWLAN terminal is presented in Fig. 3.

TUTWLAN Terminal Mapping Model The mapping model connects an application with an architecture. Mapping is performed in two stages. First, application processes are grouped, and thereafter, the groups are mapped to an architecture. Grouping can be performed according to different criteria, such as workload distribution, communication between groups, and the size of groups.

A process group maps a number of application processes to a platform component instance. In software, a process group is implemented as a single thread, and in hardware, as a single hardware accelerator, depending on the case.

Figure 4 shows the process grouping, which is based on the features of the processes. Most of the processes are divided into *LowPrio* and *HighPrio* groups that correspond to low and high priority threads in the software implementation. In addition, the processes related to the CRC-32 calculation and encryption are grouped into separate *CRC32* and

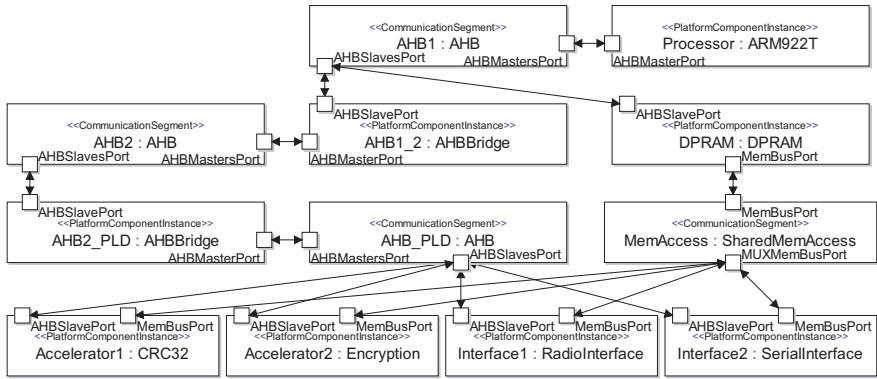


Fig. 3. TUTWLAN terminal architecture

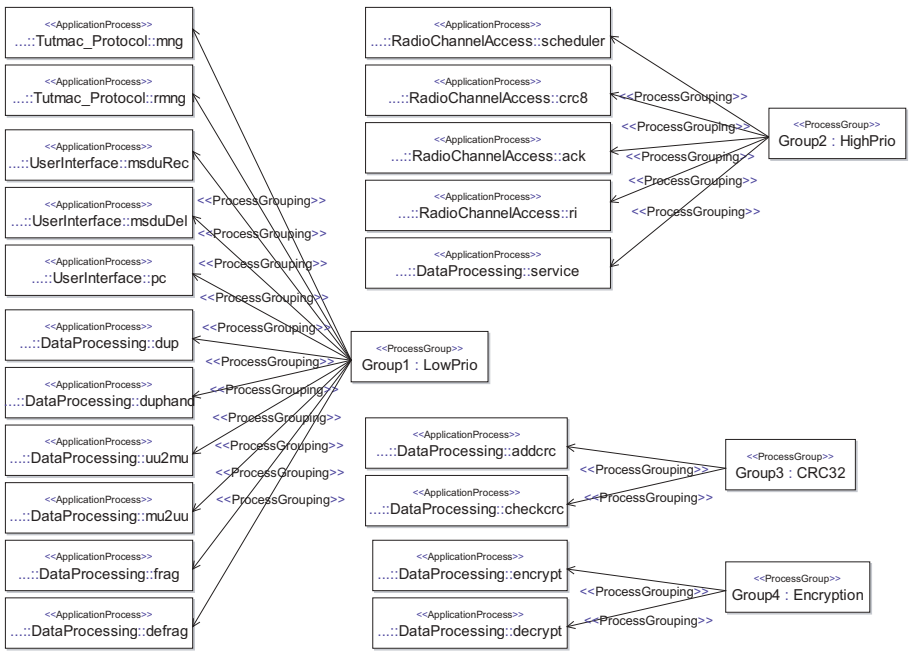


Fig. 4. TUTMAC process grouping

Encryption groups. It should be noted that the flow does not restrict the number of groups.

Figure 5 presents the platform mapping of TUTMAC. The process groups defined above are mapped on the architecture model. In this case, the mapping model is created manually, but the task can be automated using an architecture exploration tool, such as Koski [10]. If the designer has created a preliminary mapping when using architecture exploration, the tool optimizes the mapping, but retains fixed mappings.

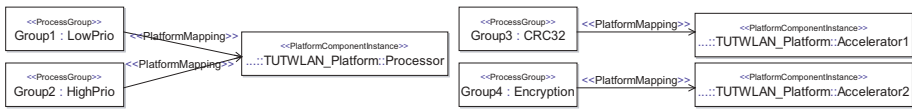


Fig. 5. TUTMAC platform mapping

3.2 Implementation Phase

C code for TUTMAC is automatically generated based on the application model. The generated code implements the functionality of state machines, but not the communication between the state machines. The generated code is complemented with run-time libraries implementing the state machine communication.

The generated code is compiled for ARM9 using Gnu C compiler. The compilation includes the codes for eCos, interface functions, external C functions, and the thread configuration. eCos enables the execution and scheduling of multiple threads with multiple priority levels, preemption and timeslicing.

The interface functions are platform dependent, and they connect the generated code with other software components and hardware. The functions take necessary actions for the signals coming out from the application model.

The external C functions are accessed inside the application model to use existing C implementations for algorithms, and to access hardware accelerators. In this case, CRC-8 is implemented as an external C function, and CRC-32 and encryption hardware accelerators are accessed using external C calling functions.

In the hardware implementation, a top-level VHDL code is written to instantiate the hardware components, which have existing VHDL implementations in the library of components. Next, the hardware is synthesized.

A configuration file for Excalibur FPGA is generated by combining the synthesized hardware and executable protocol. Finally, the configuration file is programmed to the flash memory on the development board, and the terminal is ready for use. If the executable protocol is modified, only the configuration file is necessary to regenerate, while the synthesized hardware is retained untouched.

3.3 Execution Monitoring and Model Profiling

The execution statistics include the data throughputs and delays of TUTMAC. The statistics are used for the real-time monitoring of performance, but they can also be stored into a file for the analysis of a larger set of data.

The execution trace contains information about the state transitions, signal transfers, timer events, and thread switching on the protocol. Time stamps for each event are stored. The *UML model profiler* analyses the trace, and combines it with the UML model. The profiler back-annotates the performance information to the UML model. In this way the designer gets feedback to the modeling level.

4 TUTWLAN Terminal Implementation

The final implementation of the TUTWLAN terminal on FPGA is presented in Fig. 6. The processes of TUTMAC are implemented in eCos threads. The processes are divided into two categories, high priority and low priority, each having an own thread with own priority. The high priority thread include processes having real-time constraints, including TDMA scheduling and data processing.

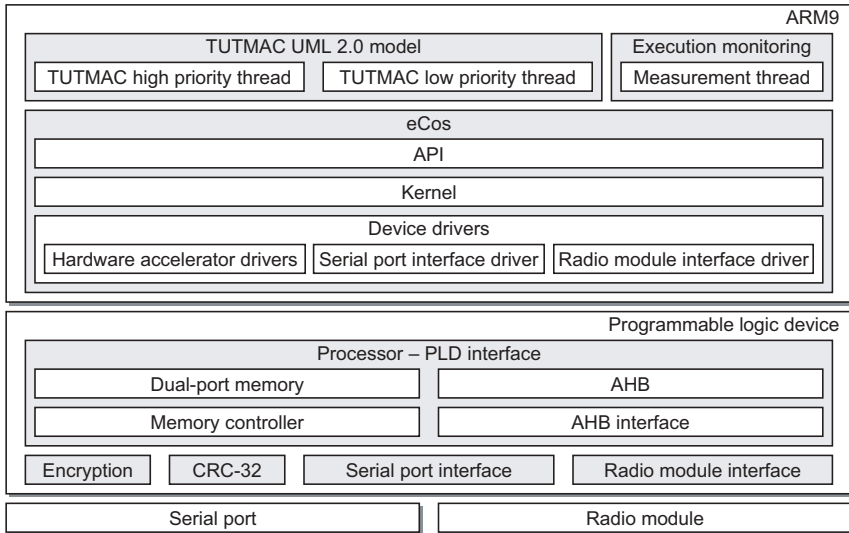


Fig. 6. TUTWLAN terminal implementation on the Excalibur FPGA

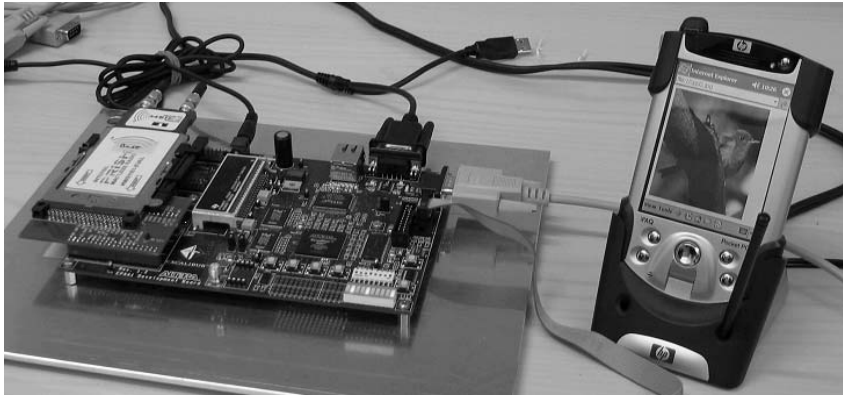
The execution statistics for real-time execution monitoring are collected in a measurement thread. The statistics, including data throughputs and delays, are measured continuously, and transferred to a workstation at a second interval.

Custom device drivers and API are included to eCos for accessing the hardware. The device drivers take care of the hardware component control, including transferring data and handling the interrupts. Respectively, API provides common functions and data structures for applications to utilize the hardware components. The hardware accelerators, serial port and radio module interfaces each have a device driver and corresponding API implementations.

Table 1 tabulates the size of the software implementation divided into code and data. In addition to the listed components, 5.6 kB of dynamically reserved memory is required for the general execution, and about 150 kB for data buffering in TUTMAC. The total size of the hardware is 3859 logic cells, which is 92.8 percents of the total PLD capacity of the used FPGA (4160 logic cells).

Table 1. Size of the software implementation

| <i>Component name</i> | <i>Code size</i> | <i>Data size</i> |
|----------------------------------|------------------|------------------|
| TUTMAC (generated code) | 18.5 kB | 2.0 kB |
| Run-time libraries | 14.7 kB | 0.8 kB |
| Interface functions & external C | 15.8 kB | 1.8 kB |
| eCos | 64.1 kB | 7.9 kB |
| Total | 113.1 kB | 12.5 kB |

**Fig. 7.** Prototype containing a TUTWLAN terminal and PocketPC device

4.1 Prototype

A prototype containing the TUTWLAN terminal and a PocketPC device with an image transfer application is presented in Fig. 7. In all, the prototype contains two TUTWLAN terminals, two PocketPC devices, and a workstation with the execution monitor. The prototype is used to illustrate the real-time execution monitoring and rapid model modifications.

The PocketPC application uses data transfer services of TUTWLAN, and has server and client sides. On the server side, a user selects the images to be transferred. On the client side, the images are received and shown on the screen.

4.2 Execution Monitoring

The execution monitor on a workstation contains three diagrams presenting TUTMAC *user data throughput*, *radio data throughput* and *reception delay* that are measured from the pUser and pPhy ports showed in Fig. 2(a).

The user data throughput gives the amount of data to the image transfer application. Correspondingly, the radio data throughput gives the gross data transferred between the radio transceiver and TUTMAC. The reception delay is the total execution time for processing a received packet between the radio transceiver and application. The

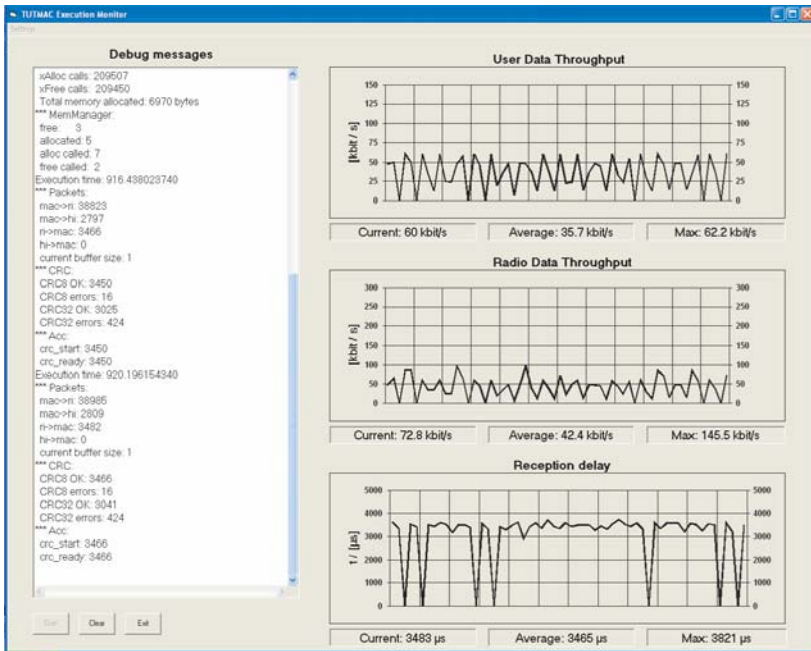


Fig. 8. User interface of the TUTMAC execution monitor

Table 2. Time for each phase of the rapid model modification

| <i>Task</i> | <i>Time</i> |
|-------------------------|---------------|
| (UML model modification | 30 s) |
| C code generation | 15 s |
| Compilation | 25 s |
| Flash programming | 17 s |
| Total | (30 s) + 57 s |

variation in delay is caused by the varying processing load on TUTMAC as well as the behavior of the application transferring data.

A screenshot of the execution monitor is presented in Fig. 8. The main window contains three diagrams presenting the execution statistics of TUTMAC, and a text window used to output debug messages.

4.3 Rapid Model Modification

The flow enables rapid modification of the functionality and structure of TUTMAC as well as mapping. This gives great possibilities to evaluate different implementations, and compare their performance. The real-time execution monitoring retrieves instant feedback about the modifications. The model profiling back-annotates the performance information to the UML models.

As an example of the rapid model modification, a forward error correction function is added to the UML model. Time for each phase is presented in Table 2. This kind of efficiency can be achieved as the UML models for the application processes already exists, and different combinations are evaluated.

5 Conclusions

This paper presented a new UML 2.0 based design flow that enables the implementation of a real-time embedded system from high-level models. Platform dependent functions are implemented only once, after which new functionality in UML can be created rapidly, and performance constraints verified. The flow was evaluated by implementing the TUTWLAN terminal on a single-processor platform with RTOS and hardware accelerators. A novel feature for performance verification is the back-annotation from the platform.

The ongoing work include multiprocessor implementation with the Koski design flow, and automatic allocation and scheduling of the application model processes on different processors.

References

1. Altera homepage, February 2005. <http://www.altera.com>.
2. Rong Chen, Marco Sgroi, Luciano Lavagno, Grant Martin, Alberto Sangiovanni-Vincentelli, and Jan Rabaey. *UML for Real: Design of embedded Real-time Systems*, chapter UML and platform-based design, pages 107–126. Kluwer Academic Publishers, May 2003.
3. eCos homepage, February 2005. <http://ecos.sourceforge.org>.
4. Stefania Gnesi, Diego Latella, and Mieke Massink. Modular semantics for a UML state-chart diagrams kernel and its extension to multicharts and branching time model-checking. *Journal of Logic and Algebraic Programming*, 51(1):43–75, 2002.
5. Panu Hämäläinen, Marko Hännikäinen, Timo D. Hämäläinen, and Jukka Saarinen. Hardware implementation of the improved WEP and RC4 encryption algorithms for wireless terminals. In *Proceedings of the European Signal Processing Conference*, volume 4, pages 2289–2292, September 2000.
6. Marko Hännikäinen, Tommi Lavikko, Petri Kukkala, and Timo D. Hämäläinen. TUTWLAN - QoS supporting wireless network. *Telecommunication Systems - Modelling, Analysis, Design and Management*, 23(3,4):297–333, 2003.
7. Petri Kukkala, Jouni Riihimäki, Marko Hännikäinen, Timo D. Hämäläinen, and Klaus Kronlöf. UML 2.0 profile for embedded system design. In *Proceedings of the Design, Automation and Test in Europe*, volume 2, pages 710–715, March 2005.
8. Luciano Lavagno, Grant Martin, and Bran Selic, editors. *UML for Real: Design of Embedded Real-time Systems*. Kluwer Academic Publishers, May 2003.
9. Object Management Group (OMG). *UML Profile for Schedulability, Performance, and Time Specification (Version 1.1)*, January 2005.
10. Erno Salminen, Vesa Lahtinen, Tero Kangas, Jouni Riihimäki, Kimmo Kuusilinna, and Timo D. Hämäläinen. HIBI v.2 communication network for system-on-chip. In *Proceedings of the International Workshop on Systems, Architectures, Modeling and Simulation*, pages 413–422, July 2004.
11. Telelogic homepage, February 2005. <http://www.telelogic.com>.