A Framework for Rapid Application Development of Distributed Embedded Real-Time Systems

R. Obermaisser, P. Peti
Vienna University of Technology
Vienna
Austria
email: {ro,php}@vmars.tuwien.ac.at

Abstract— **There is a strong need for a system design approach for distributed embedded real-time systems with supporting tools that reduces complexity and development time. In real-time systems timing constraints should function as a driving force throughout the design process, instead of employing temporal constraints only at the starting and end point of the design process (i. e. within the specification and the validation of the implementation). We describe a platform based design methodology aiming at safety-critical distributed real-time applications. This methodology incorporates the idea of the platform-independent and platform-specific model of OMG's Model Driven Architecture. For functional design activities at the application space, the framework employs the MATLAB/Simulink tool suite. The framework offers dedicated blocksets for modeling the communication between nodes of a distributed time-triggered system, while taking the temporal behavior of communication and processing activities into account. Automatic code generation maps this model onto the platform space. The platform space comprises a real-time Linux variant as the operating systems and the Time-Triggered Architecture as the communication system.**

Keywords— **Real-Time Systems, Real-Time Linux, Platform-based Design, Model Driven Architecture, Matlab/Simulink.**

I. INTRODUCTION

EMERGING safety-critical applications (e. g., X-by-wire) in the automotive and avionics domain require a deterministic behavior of message transmissions and a guaranteed transmission latency even at peak-load. Time-triggered communication protocols can provide this deterministic behavior. In addition to hard real-time performance they support temporal composability and dependability. As a consequence time-triggered protocols are becoming more and more accepted as the communication infrastructure for safety-critical applications [1], [2]. Furthermore, business realities in many application domains result in shortened product cycles (e. g., automotive industry [3]). Hence, there is a strong need for a system design framework that reduces complexity and development time. Reuse of validated functions and ensuring portability between various architectures are major concerns.

This paper describes a design framework for distributed embedded real-time systems that addresses the above mentioned requirements. This model is based on the platform based design methodology proposed by [4]. In our framework, the Time-Triggered Architecture in combination with middleware services and a real-time Linux extension forms the system platform. Successive refinements of the specification results in a platform specific specification tailored to a particular platform. During the specification process we distinguish between a platform-independent and a platform-specific model based on the OMG Model Driven Architecture (MDA) [5].

We present a framework that exploits the MATLAB/Simulink tool suite for enabling the designer to devise a platform independent functional model from the specification. Subsequently, the designer can exploit communication blocksets to construct the platform-specific model out of the platform-independent functional model. Based on the resulting model the MATLAB/Simulink Real-Time Workshop automatically generates code for the real-time Linux variant RTAI, which functions as the operating system in our network platform.

The structure of the paper is as follows. Section II describes supporting technologies used in the remainder of the paper. In Section III we present a design model for distributed embedded real-time systems in accordance with the platform based design methodology. Section IV describes the appliance of the model in a framework. Section V concludes the paper.

II. SUPPORTING TECHNOLOGIES

This section summarizes the concepts underlying the system design framework described in the paper.

A. Distributed Embedded Real-Time Systems

A distributed system consists of a set of nodes interconnected via a common network (communication system). A node can at least be partitioned into two subsystems, the communication controller and the host computer. The interface between these two is called Communication Network Interface (CNI) [6].

In the context of embedded real-time systems a complete node seems to be the best choice for a component [7], since the component-behavior can then be specified in the domains of value and time. Thus, a component is considered to be a self-contained computational element with its own hardware (processor, memory, communication interface, and interface to the controlled object) and software (application programs, operating system), which interacts with its environment by exchanging messages across Linking Interfaces (LIFs) [8].

B. Platform-based Design

Platform-based design is increasingly becoming accepted as a well suited design methodology for embedded systems [9], [10] According to [4, p. 5] a platform is defined as an *abstraction layer in the design flow that facilitates a number of possible refinements into a subsequent abstraction layer (platform) in the design flow*.

Platform based design provides an abstraction of the underlying hardware and software foundation and offers the following advantages:

- **Reuse:** The reuse of application software is supported by offering the same system platform independently of the actual hardware/software platform (e. g., processor, operating system).
- **General Purpose Components:** It is desirable to use general purpose processors that work for several designs, instead of custom-designed hardware. The main motivation for this approach is the high setup cost for chip production, since development cost of the processor amortizes over large number of produced units.

A system platform is defined as the combination of a hardware platform and a software platform. A hardware platform is a family of architectures that satisfy a set of architectural constraints that are imposed to allow the re-use of hardware and software components [11]. A software platform is a software layer that provides a real-time operating system and middleware services.

C. *OMG's Model Driven Architecture (MDA)*

The Model Driven Architecture [12] has been devised by the OMG. It separates the application logic from the underlying platform technology. It distinguishes between a Platform Independent Model (PIM) and a Platform Specific Model (PSM). The PIM provides *formal specifications of the structure and function of the system that abstracts away technical details*. The PSM is expressed in terms of the specification model of the target platform. The distinction between the PIM and the PSM is motivated by the desire for interoperability, portability, and short design cycles.

D. *Real-Time Operating Systems*

The primary role of an operating system is the management of resources while meeting the demands of application software. In real-time applications, the operating system must also ensure timeliness and predictability.

The design of a real-time operating system (RTOS) is a balance between providing a reasonably rich feature set for application development and deployment without sacrificing predictability and timeliness. Examples for corresponding standards are [13], [14].

A comparison of real-time operating systems features is available in [15]. This paper compares existing real-time operating systems with respect to scheduling, synchronization, interprocess communication, interrupt handling, memory management, and timers.

III. METHODOLOGY BASED ON PLATFORM-BASED DESIGN

This section describes the platform-based design model for distributed embedded real-time systems. As depicted in Figure 1 the model distinguishes between application and platform space. The application space is further subdivided into the specification, the platform-independent model, the platform-specific model and the resulting executable representation. The platform space consists of the software platform (middleware layer, operating system) and the hardware platform (host computer, communication system).

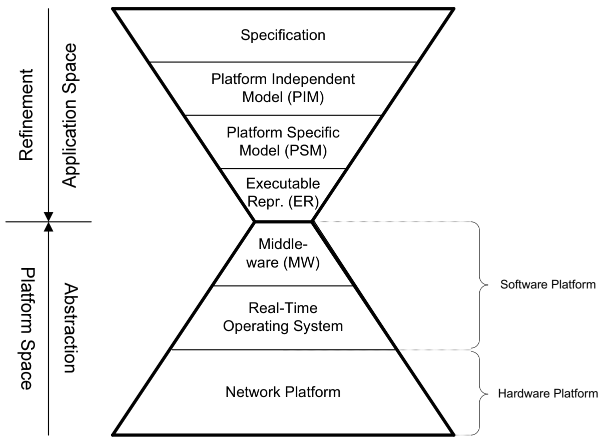


Fig. 1. Methodology based on Platform-Based Design

A. *Specification*

The specification is the result of the requirement analysis phase and provides input for later design phases. In the context of real-time systems a precise temporal specification of the provided services must be included.

B. *Platform Independent Model*

The PIM constitutes a functional description of the data transformations of the system. The functional model abstracts from the details of the platform and the implementation. By abstracting from the implementation, the designer is enabled to gain a thorough understanding of the problem space.

However, in the context of real-time systems, one cannot abstract away from time. The functional model must include the temporal constraints of services.

C. *Platform Specific Model*

The PIM is mapped to a particular platform, thereby reaching the PSM. It substantiates functional units to actual components. In the context of this paper, the term component refers to a self-contained hardware/software unit (system component) [16]. The abstract communication relationships of the PIM become either interprocess communication relationships or communication activities via a common network. The latter applies in case the functional units are allocated to separate components, the first if functional units share a component.

D. *Executable Representation (ER)*

The executable representation may either be created by an automatic code generation tool or by a human programmer. Currently software for embedded systems is mainly written by using low-level programming languages like C or even assembly language in order to meet the resource and performance requirements in embedded applications.

Wybo et. al compared code generation tools from different with respect to traceability, readability and efficiency (code size, RAM utilization, performance). The result of the analysis shows that the generated code is close to being practical for memory constrained environments [17].

E. Middleware

According to [18] a middleware service is a *general-purpose service that sits between platforms and applications*. The middleware provides a standard interface for applications. Middleware is connectivity software that consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network. It offers services of the underlying communication system on an abstract level.

F. Operating System

The operating system hides the details of the underlying hardware. It provides system calls as the interface to upper layers. For embedded real-time systems the operating system should provide support for interrupt handling, task dispatching, clock synchronization, interprocess communication, and error detection. An example for an error detection mechanism is the monitoring of tasks with respect to their worst-case execution times [19].

G. Network Platform

The network platform is formed by the node computers and the communication system. The primary function of the network platform is to provide an infrastructure for the exchange of information between components. Essential services of the network platform in embedded real-time systems are a temporally predictable message transport, error-containment and clock synchronization [20].

The purpose of communication system is to transport messages from the communication network interface (CNI) of the sender to the CNI of the receiver. In order to meet the temporal constraints imposed by the specification, the transmission latencies and transport in the transmission of messages from the sender's CNI to the receiver's CNI must be bounded and low. The application software executing in the host computer exploits the services of the communication system by accessing the CNI.

IV. Framework

This section describes the framework for distributed embedded real-time systems (see Figure 2), which is based on the Time-Triggered Architecture and the real-time Linux variant RTAI. In accordance with the design model presented in the previous section, we distinguish between platform-independent and platform-specific design phases. Out of the platform-specific model code generation yields an RTAI kernel module.

We will present the framework with an accompanying example (see Figure 3), which is a simplified version of a anti-lock braking system serving for demonstration purposes only.

A. Specification

The design model for the case study builds on top of a specification as described in the previous section. The specification in the anti-lock braking example describes the prevention of blocking of wheels in emergency-braking conditions. Depending on the wheel revolution and the driver's brake pressure the system needs to individually calculate the braking force for each wheel.

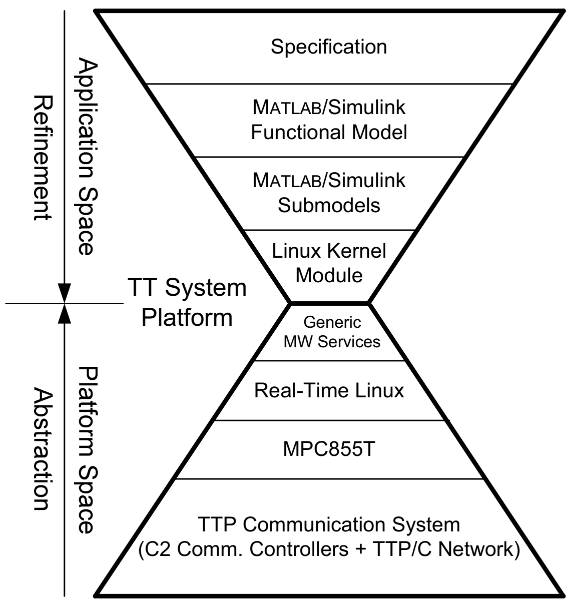

Specification
MATLAB/Simulink Functional Model
MATLAB/Simulink Submodels
Linux Kernel Module
Generic MW Services
Real-Time Linux
MPC855T
TTP Communication System (C2 Comm. Controllers + TTP/C Network)
Application Space Refinement
TT System Platform
Platform Space Abstraction

Fig. 2. Case Study for the System Design Model

B. Platform Independent Model

In the framework the MATLAB/Simulink model represents the PIM. MATLAB/Simulink[1] is a software package widely used in industry to model and simulate dynamic systems. In MATLAB/Simulink complex dynamical systems can be created by connecting functional blocks available from libraries and new functionality can be added to the model by defining new block sets.

At this stage MATLAB/Simulink facilitates gaining confidence in the adequacy of the network platform independent model by running simulation runs.

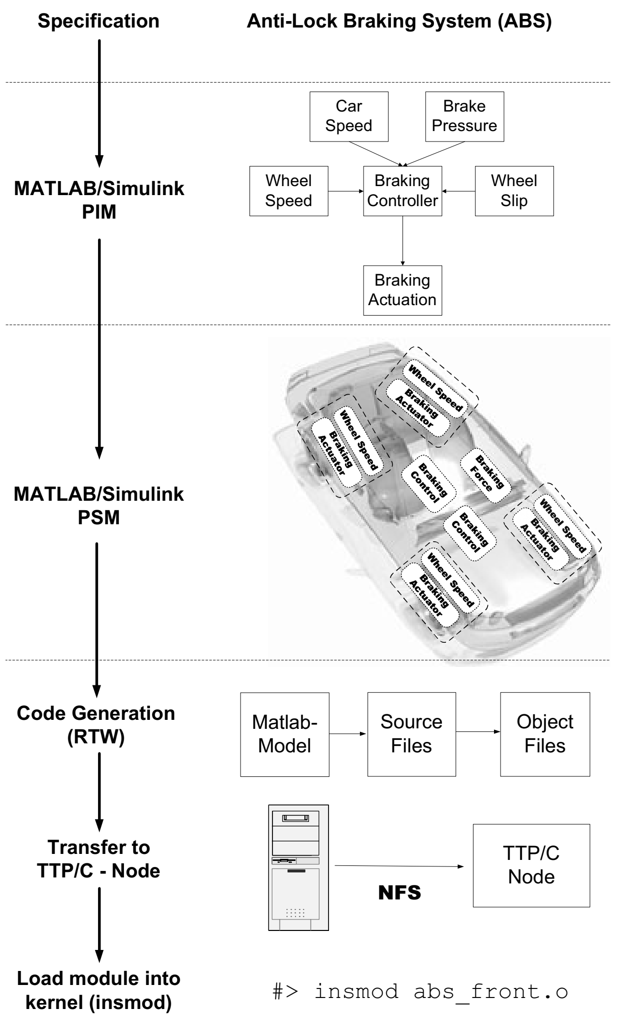
In the ABS example the PIM incorporates functionality of the braking system in the form of functional units. Dedicated functional units represent the braking system's inputs, namely the brake pressure (driver's input), wheel revolutions and frictions. The ABS algorithm is modeled by a functional unit denoted as braking control. The actuation via the brake cylinder control value is represented by the functional unit braking actuation.

C. Platform Specific Model

The migration of the PIM to the PSM maps functional units to actual components. The designer constructs a distinct submodel m_i for every component i. Every submodel m_i represents a task t_i that implements the functional units mapped to a particular node. It is essential to possess knowledge about the worst-case execution time (WCET) of the resulting tasks t_i. This WCETs must be determined by a corresponding tool as described in [21].

Our framework aims at a time-triggered network platform. Such a platform employs a time-triggered protocol that groups communication into rounds according to a TDMA (time division multiple access) scheme.

[1]created by the MathWorks, `www.mathworks.com`



```
#> insmod abs_front.o
```

Fig. 3. Design Example – Anti-Lock Braking System (ABS)

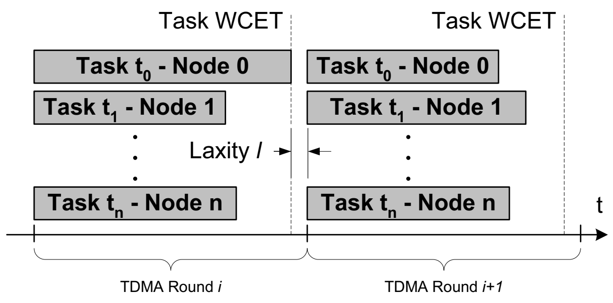


Fig. 4. Temporal Behavior of Task Activations

```
initialize memory
for each TDMA-round do
    read inputs
    compute outputs
    update memory
od
```

Fig. 5. Synchronous Sample Driven Task Execution Scheme

The following condition must hold (see Figure 4):

$$\max_i (t_i) \leq d_{\mathrm{TR}}$$

d_{TR} denotes the length of the TDMA round. The execution scheme shown in Figure 5 corresponds to the execution model described in [22].

In our example we provide seven components that implement the functionality of the platform-independent ABS model. As indicated in Figure 3, one components is located at every wheel. This components implements the functionality of wheel speed and friction measurement, as well as brake cylinder control. Two redundant braking control components execute the braking algorithms to determine the braking force for each wheel. The driver's input is captured by a corresponding component.

D. Executable Representation

Real-Time Workshop enables to automatically generate portable and customizable ANSI C code from MATLAB/Simulink models [23]. We apply the MATLAB/Simulink real-time workshop for automatically generating code for our platform out of the TNPSM.

In our case study, we provide a framework which supports the automatic generation of Linux kernel modules. These kernel modules can be dynamically loaded into the Linux kernel. The kernel loader performs linker and version checks [24].

E. Middleware

In the framework a fault-tolerance layer [25] offers redundancy management functionality. The fault tolerance layer groups nodes into fault-tolerant units. The basis for the grouping are the redundancy blocks (e. g., TMR block) inserted during the creation of the Timed Network Platform Specific Model (TNPSM).

F. Operation System

The nodes employed in the framework use the embedded real-time Linux variant RTAI [26] (Real Time Application Interface) as their operation system. The RTAI version used for the prototype implementation combines a real-time hardware abstraction layer (RTHAL) with a real-time application interface for making Linux suitable for hard real-time applications [27]. RTAI introduces a real-time scheduler, which runs the conventional Linux operating system kernel as the idle task, i. e. non real time Linux only executes when there are no real-time tasks to run, and the real-time kernel is inactive. The conventional Linux kernel is modified to prevent it from blocking or redirecting hardware interrupts. Hence, Linux cannot add latency to the interrupt response time of the real-time system. RTAI performs these modifications by replacing the corresponding code in the Linux kernel (e. g., `cli`, `sti`, and `iret` statements) with calls to functions in the real-time hardware abstraction layer. This mechanism offers the possibility for software emulation of interrupt control hardware. When an interrupt occurs, the real-time kernel intercepts the interrupt and runs its own dispatcher, which invokes the corresponding real-time handler. In case the interrupt is shared with the conventional Linux kernel, an interrupt pending flag is set. The flag causes then appropriate Linux interrupt handler to be invoked, when the Linux kernel is activated. In order to prevent temporal fault propagation from the

Linux kernel to the real-time kernel, the real-time kernel is never blocked by the Linux side.

G. Host Computer

For the framework, a TTP/C [28] cluster with TTP monitoring nodes [29] is used. The TTP monitoring nodes are based on the TTP-C2 controller (AS8202). They are equipped with the Motorola embedded Power PC processor MPC855T, which features an on-chip fast Ethernet controller (100BASE-T), which can be used e. g., for TCP/IP connections to a PC. The monitoring node supports synchronous (25 Mbit/s) and asynchronous (MFM - 5 Mbit/s) bus interfaces for establishing the TTP/C communication. For the synchronous bus interface, it employs a 100BASE-TX physical layer with the media independent interface (MII) forming the bus towards the physical interfaces. Furthermore, a PCMCIA slot for cards type I/II/III is available on the monitoring nodes. The MPC855T host CPU employs a PowerPC core and runs at 80 MHz. It is equipped with 16 MBytes of dynamic RAM, and 8 MBytes of flash memory. The CNI memory of the TTP-C2 controller is mapped into the address space of the MPC855T host CPU.

H. Communication System

For communication the Time-Triggered Architecture (TTA) is used. The TTA provides a computing infrastructure for the design and implementation of dependable distributed embedded systems [30]. The basic building block is a node computer, which is a self-contained composite hardware/software subsystem. Two replicated communication channels connect the nodes to build a cluster (see Figure 6).

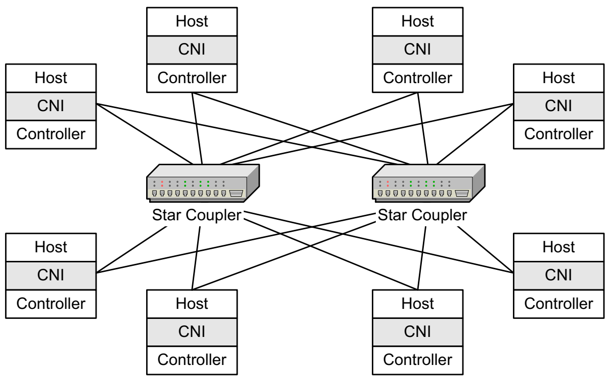


Fig. 6. TTP Network (Star Topology)

Communication is performed according to time division multiple access (TDMA) schedule. The TTA obtains its synchronous behavior by the progression of real time. The TTA establishes a global time, which is employed for arbitration to the communication medium.

V. Conclusion

Platform-based design is a design methodology especially suitable for the design of embedded systems. This paper presents rapid development framework incorporating TTA as the network platform. The framework is applied for the construction of an anti-lock braking system, thereby demonstrating that our framework represents a sound technical solution for the design of embedded systems.

The distinction between a platform-independent and a platform-specific model in accordance with OMG's Model Driven Architecture helps to ensure portability and reduces complexity. RTAI Linux as an open-source real-time operating system and TTA have turned out to be an effective system platform in the presented development framework.

References

[1] J. Rushby. Bus architectures for safety-critical embedded systems. In *EMSOFT 2001: Proceedings of the First Workshop on Embedded Software*, volume 2211 of *Lecture Notes in Computer Science*, pages 306–323, October 2001.

[2] Hermann Kopetz. Why time-triggered architectures will succeed in large hard real-time systems. In *Proceedings of the 5th IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 2–9, Chenju, Korea, Aug. 1995.

[3] J. Barkai. Vehicle diagnostics–are you ready for the challenge? In *ATTCE 2001*, volume 5. SAE, October 2001.

[4] A. Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign of EETimes*, February 2002.

[5] OMG. Model Driven Architecture (MDA). Technical Report document number ormsc/2001-07-11, Object Management Group, July 2001.

[6] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[7] H. Kopetz. Component-based design of large distributed real-time systems. *Control Engineering Practice - A Journal of IFAC*, 6:53–60, 1998.

[8] H. Kopetz and N. Suri. Compositional design of RT systems: A conceptual basis for specification of linking interfaces. Research report, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2002.

[9] A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design and Test of Computers*, 18(6):23–33, November-December 2001.

[10] R. Goering. Platform-based design: A choice, not a panacea. *EETimes*, September 2002.

[11] A. Ferrari and A. Sangiovanni-Vincentelli. System design: traditional concepts and new paradigms. In *International Conference on Computer Design (ICCD '99)*, pages 2–12, October 1999.

[12] OMG. Model Driven Architecture (MDA). Technical Report document number ormsc/2001-07-11, Object Management Group, July 2001. Available at `http://www.omg.org`.

[13] OSEK/VDX. *Time-Triggered Operating System – Specification 1.0*, July 2001.

[14] IEEE Standards Publication. *International Standard for Real-Time: POSIX 1003.1*. ISO/IEC-9945-1.

[15] R. Yerraballi. Real-time operating systems: An ongoing review. In *Work-In-Progress Sessions of The 21st IEEE Real-Time Systems Symposium (RTSSWIP00)*, Orlando, Florida, November 2000.

[16] H. Kopetz and N. Suri. Compositional design of RT systems: A conceptual basis for specification of linking interfaces. Research report, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2002.

[17] D. Wybo and D. Putti. A qualitative analysis of automatic code generation tools for automotive powertrain applications. In *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design*, pages 225–230, Hawaii, USA, August 1999.

[18] P. A. Bernstein. Middleware: a model for distributed system services. *Communications of the ACM*, 39(2):86–98, 1996.

[19] P. Puschner and C. Koza. Calculating the maximum execution time of real-time programs. *Journal of Real-Time Systems*, 1(2):159–176, Sep. 1989.

[20] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[21] R. Kirner, R. Lang, G. Freiberger, and P. Puschner. Fully automatic worst-case execution time analysis for matlab/simulink models. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, pages 31–40, Vienna, Austria, June 2002. Vienna University of Technology, IEEE.

[22] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. In *Proceedings of the IEEE Special Issue on Modeling and Design of Embedded Software*, volume 91 of *1*, pages 64–83, January 2003.

[23] The Mathworks Inc. Real-time workshop user's guide, Version 4, June 2001.

[24] A. Rubini and J. Corbet. *Linux Device Drivers*. O'Reilly, 2nd edition, June 2001.

[25] T. Becker. Application-transparent fault tolerance in distributed systems. In *Proceedings of the Second International Workshop in Configurable Distributed Systems*. IEEE Computer Society Press, May 1994.

[26] D. Beal, E. Bianchi, L. Dozio, S. Hughes, P. Mantegazza, and S. Papacharalambous. RTAI: Real-Time Application Interface. *Linux Journal*, April 2000.

[27] RTAI Programming Guide, Version 1.0. Dipartimento di Ingegneria Aerospaziale Politecnico di Milano (DIAPM), Italy, September 2000. Available at `http://www.rtai.org`.

[28] H. Kopetz. *Specification of the TTP/C Protocol*. TTTech, Schönbrunner Straße 7, A-1040 Vienna, Austria, July 1999. Available at `http://www.ttpforum.org`.

[29] TTTech Computertechnik AG. TTP Monitoringnode - A TTP Development Board for the Time-Triggered Architecture, March 2002.

[30] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE Special Issue on Modeling and Design of Embedded Software*, January 2003.