Cluster Simulation – Support for Distributed Development of Hard Real-Time Systems using TDMA-Based Communication

Thomas M. Galla and Roman Pallierer
Institut für Technische Informatik,
Technical University of Vienna
Treitlstraße 3/182/1
A-1040 Vienna, Austria

email: {tom,roman}@vmars.tuwien.ac.at
fax: +43 (1) 5869149

Abstract

In the field of safety-critical real-time systems the development of distributed applications for fault tolerance reasons is a common practice. Hereby the whole application is divided into several distinct sub-systems, which are afterwards mapped onto a set of associated processors called nodes.

Cluster simulation provides a cheap and useful technique to test single nodes of a distributed application in isolation. Compared to alternative test approaches where the whole distributed system has to be built up the hardware requirements imposed by cluster simulation are rather small.

We present an approach together with a prototype implementation for cluster simulation in distributed hard real-time systems using TDMA-based communication. This cluster simulator has proven to be a valuable development support tool for single nodes within a distributed application by providing a versatile platform for validation of the nodes behavior against the rest of the distributed system.

1. Introduction

Distributed hard real-time systems consist of a number of nodes that interact with each other by means of communication. The correct operation of such systems depends not only on the correct execution of the application tasks within the nodes, but significantly on latencies and jitter imposed by the communication sub-system. In case of safety-critical applications, the correct operation of the system is of utmost importance and therefore the impacts of the communication sub-system are crucial.

Due to the distributed architecture the development of such systems is organized in a more and more decentralized manner. Different sub-system supplier develop various components which are then assembled by the system integrator. In the automotive industry, for example, car manufacturer integrate a large number of components developed by different supplier companies. The complexity of this integration and validation process tremendously increases with the use of such systems for safety-critical applications, such as brake-by-wire or steer-by-wire applications [4]. An approach to solve this problem has been introduced in [7]. If nodes depend on the temporal properties of the data exchanged among each other, a precise specification of the interfaces in the value *and* in the time domain is required to verify the behavior of a stand-alone developed component. A TDMA-based communication system allows such a precise specification of the temporal behavior of a node, since the points in time when information is exchanged are determined at design time.

In this paper we present a tool that supports the stand-alone development and validation of nodes that are interconnected by such a TDMA-based communication system. CANoe [2] is a similar tool for CAN-based systems, which *estimates* the bus load and latency times of the whole system by means of simulation in order to test the correct behavior of a node. In our approach, however, we want to exploit the TDMA-based paradigm using the *precise specification* of the points in time when a node transmits messages in order to give *guarantees* of the correct behavior of the stand-alone developed component.

The paper is structured as follows: Section 2 provides a list of objectives and requirements for a tool set facilitating cluster simulation. Section 3 gives a short overview over the the used system architecture and describes the main parts it consists of. Section 4 focuses on the basic concepts of the proposed cluster simulation architecture. Section 5 illustrates the implementation details of the cluster simulator and describes the hardware and software setup. In Section 6 we describe our experiences with the presented cluster simulation approach. Section 7 concludes the report with some future visions regarding the field of cluster simulation.

2. Objectives and Requirements

This sections gives an overview of the objectives and requirements of the cluster simulator.

The objective of the cluster simulation is to provide extensive testing facilities of the behavior of single nodes without the need to setup the whole system. It should provide a cheap and efficient way for the distributed development of single components by different sub-system suppliers. In case of safety-critical systems, benefits for the system validation process are given if the tested functionality of stand-alone developed nodes is not influenced and therefore does not require to to be re-tested upon system integration. To achieve such a *composable* system [3] a precise specification of the interfaces in the value and time domain is a prerequisite.

When designing a distributed system the implementation of a single node is often not yet determined or should be interchangeable. In a distributed braking system, for example, various implementations of the node computing the brake force on the wheels (e.g., including ABS or not) may be considered. Although the concrete implementation of a node might be unknown, the functional and temporal behavior of its interface has to be specified. The weaker the specification the less confidence can be achieved from the stand-alone testing process and the more complex the system integration and the validation process is. One approach to solve this problem is to perform an iterative refinement of the specification during the development.

The cluster simulator we present in this paper requires the following interface specification: the points in time when messages are sent, the semantics and the value range of the messages. If the node under test, using the example above, computes the actual brake force on the wheels, the update period and the value range need to be given for all input data (desired brake force derived from brake pedal, yaw rate information, vehicle status information etc.) and all output data (actual brake force of the wheels) of the node. The used static transmission scheme must be chosen in a way that the specified update period is achieved. The value range might be given by upper and lower bounds or by simplified functions computing the value. During the stand-alone development of a node, the cluster simulator has to support recording of the value range in order to refine the specification in the value domain (The time domain is already sufficiently specified by the static transmission scheme). In an iteration step this new specification is fed back to all cluster simulators in order to perform a stand-alone test of the nodes developed in a distributed manner. Due to this iteration mechanism, a more and more accurate specification can be gained during the development to achieve high confidence in the testing process performed with the cluster simulator.

3. System Overview

This section introduces the system architecture of the hard real-time systems for which our approach has been designed.

The hard real-time system consists of a number of nodes which are interconnected by a communication medium. A node itself consists of the host sub-system executing the application tasks and the communication sub-system. The Communication Network Interface (CNI) is the node-internal interface between these two sub-systems. It has memory interface semantics and could be, for example, implemented as DPRAM.

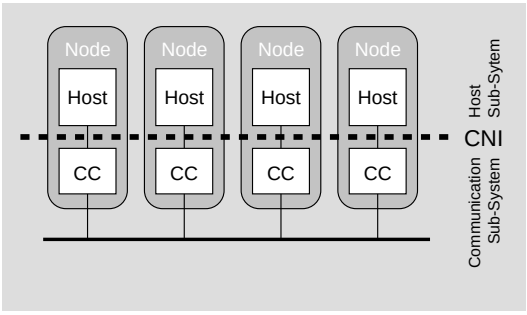

Figure 1. Structure of a TTP/C Cluster

Access to the communication medium is controlled by a cyclic time-division multiple access (TDMA) scheme derived from a global notion of time. This scheme divides the whole capacity of the communication medium into so-called *TDMA slots*, where each

TDMA slot is uniquely assigned to one specific node. This assignment is generated at design time and stored as static control data within every communication subsystem. The sequence of *TDMA slots* in which each node sends at most one message forms a *TDMA round*. All TDMA rounds exhibit the same temporal access pattern, but different messages may be sent in different TDMA rounds. The number of different TDMA rounds determines the length of the *cluster cycle*.

4. Concepts

This section focuses on the main concepts of the presented cluster simulator giving a description of its different parts and the interfaces between these parts.

As already mentioned in Section 2 the cluster simulator facilitates the development and the testing of single nodes of a distributed real-time application. Figure 2 illustrates the setup for this kind of testing.

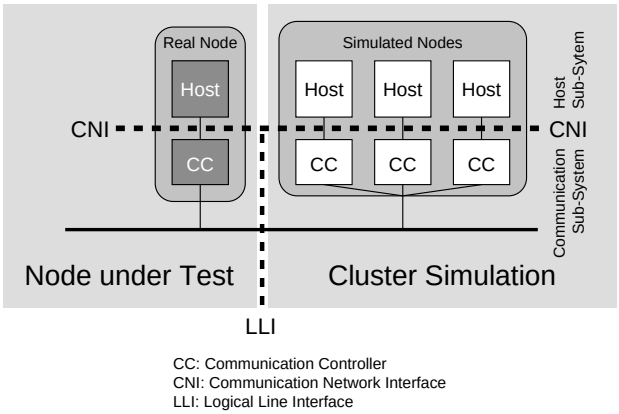

Figure 2. Cluster Simulation Setup

As indicated by Figure 2 the cluster simulator is connected to the node under test using a common communication medium. Thus the interaction between the cluster simulator and the node under test takes places via the *logical line interface (LLI)*. In order to provide a simulation that is transparent for the node under test, the cluster simulator's behavior at the LLI must match the behavior of the simulated nodes.

In conformance with the generic system architecture presented in Section 3, the cluster simulator itself can be divided into two distinct sub-systems – a host subsystem and a communication sub-system – as well. The interaction of these two sub-systems takes place via the CNI.

For the internal hardware setup of the cluster simulator basically two contrary approaches are possible. On the one hand the cluster simulator can be built out of a *multitude of nodes* (one physical node for each simulated one). On the other hand an approach where the cluster simulator consists of only *one node* is possible as well. For reasons of simplicity of the hardware setup, easy maintenance and last but not least cost reasons, the second alternative is proposed in this paper.

This second alternative however causes a scheduling problem that in general cannot be solved. Tasks that were executed in parallel on multiple nodes must now be allocated to a single node, which might yield that the task set is no longer schedulable. A solution to this problem is to *abstract* from the actual functionality of the simulated nodes by making *proper simplifications*. Section 4.2 describes how this is done in the presented cluster simulator.

4.1. Communication Sub-System

The communication sub-system part of the cluster simulator is responsible of the *timely transmission* of the simulated messages.

In TDMA-based communication systems it is not required to encode the sender and the type of a message in the message itself. The TDMA slot in which the message is sent uniquely identifies both the message type and the sender. In order to enable all non-simulated nodes to derive the correct message type and the correct simulated sender of the messages sent by the cluster simulator Obligations 1 and 2 must be fulfilled.

Obligation 1 *The cluster simulator must transmit messages in every simulated TDMA slot.*

Obligation 2 *The cluster simulator must not transmit any messages in other TDMA slots than the simulated ones.*

In order to ensure that other non-simulated nodes are able to decode the cluster simulators transmission, the following obligation must hold as well.

Obligation 3 *The used bus speed must be agreed by all nodes in the cluster including the cluster simulator itself (i.e., the cluster simulator must use the same bus speed as the simulated nodes).*

Additionally to the previous obligations the following must be fulfilled to prevent messages from getting discarded upon reception due to incorrect message length.

Obligation 4 *Each message transmitted by the cluster simulator must have exactly the same length as the corresponding message transmitted by the simulated node.*

The re-integration of failed nodes and the integration upon cluster startup of a single node is restricted to its own TDMA slot. – The node is only allowed to (re-)join a running cluster by transmitting its initial frame in its own sending slot.

In order to accomplish a behavior of the cluster simulator that matches the behavior of the simulated nodes Obligation 5 must hold.

Obligation 5 *The cluster simulator must be allowed to integrate in every simulated TDMA slot.*

Obligation 6 must be fulfilled to prevent the cluster simulator from disturbing ongoing transmissions of already integrated nodes upon its own (re-)integration.

Obligation 6 *It must be prohibited that the cluster simulator integrates in any other TDMA slots than the simulated ones.*

4.2. Host Sub-System

The responsibility of the host sub-system lies in the *provision of simulated message data* at the CNI prior to the actual transmission by the communication sub-system. Ideally (i.e., in a perfect simulation) this data should match the data provided by the simulated nodes. This however is not possible for the following reasons:

- As indicated earlier the cluster simulator consists of only one physical node and may therefore be equipped with less processing power than the the sum of the simulated nodes. Due to this limited processing power it is in general not possible for the cluster simulator to perform all application tasks of the simulated nodes.
- During the development phase the exact functional specification of the simulated nodes is not available. In the most cases only the types of messages and the semantics of their contents are defined.

Thus it is inevitable to introduce a *level of abstraction* to *simplify* the tasks that have to be performed by the cluster simulator. The presented cluster simulator uses a *table-driven approach* for this simplification.

The host-subsystem operates synchronously to the TDMA scheme of the communication sub-system. A statically defined *dispatching table* which contains an entry for each TDMA slot is used to control the behavior of the host part of the cluster simulator. TDMA slots where no messages have to be sent and where no actions by the host part of the cluster simulator have to be performed are left empty.

Each entry of the dispatching table entries can itself consist of multiple entries of the following two different entry types:

Pre-Defined Messages This entry type consists of up to 16 bytes of *user-defined data* and the corresponding *CNI address* where this data has to be copied to in the specific TDMA slot.

The user-defined data is generated off-line (either by hand or by an appropriate design tool) based on the functional specification of the tasks performed by the simulated nodes.

User-Defined Functions These functions allow the user to specify arbitrary tasks that are dispatched within the specific TDMA slot by the cluster simulator. These task can on the one hand be used to generate message data in the CNI. On the other hand these tasks can be used for the communication with peripheral device (e.g., sensors and actuators) connected to the host sub-system.

While pre-defined messages provide a very controlled and well-define way to specify the behavior of the cluster simulator, the user-defined functions introduce a great flexibility as far as the cluster simulator's behavior is concerned.

In order to *guarantee* the *timely provision of simulated message data* at the CNI, the execution time of the operations generating this data (i.e., user-defined functions and copying of pre-defined message data) must be bounded and known.

Since the duration of the copy operation solely depends on the number of bytes to be copied and on the duration of the copy operation for one byte (τ_{byte}), the exact execution time for a given pre-defined message entry (m) can be calculated. Assuming that the worst case execution times (WCETs) [11, 12, 10] of the cluster simulator core (i.e., the part of the cluster simulator that is responsible for the interpretation of the dispatching table and for the invocation of the user-defined functions and the copy operation) itself (τ_{core}^{WCET}) and of the user-defined functions (τ_{f}^{WCET}) are know, an *upper bound* on the execution time of all actions that have to be performed by the cluster simulator (τ_{csim}^{WCET}) can be given (Equation 1).

$$\tau_{csim}^{WCET} = \tau_{byte} \cdot \sum_{m \in \mathbb{M}_i} bytes(m) + \sum_{f \in \mathbb{F}_i} \tau_{f}^{WCET} + \tau_{core}^{WCET} \quad (1)$$

Hereby $\mathbb{M}_i$ denotes the set of pre-defined messages and $\mathbb{F}_i$ the set of user-defined functions for slot i. The operator $bytes(m)$ is used to determine the number of bytes the data of message m consists of.

In case the time difference between invocation time of cluster simulator tasks (t_i^{incovc}) and the transmission time of the messages (t_i^{trans}) for a specific TDMA slot i is larger than the previously mentioned upper bound on the sum of the execution times of the cluster simulator tasks, it can be *guaranteed* that the message data will be present in the CNI prior to the transmission by the communication sub-system part of the cluster simulator.

$$\tau_{csim}^{WCET} < t_i^{trans} - t_i^{incovc} \quad (2)$$

5. Implementation

This section focuses on a concrete implementation of the cluster simulator for architectures based on TTP/C [5]. It describes how the obligations defined in the previous sections are actually met by the implementation.

Just like the previous section this section will be divided into a part covering the communication subsystem and a part dealing with the host sub-system.

5.1. Communication Sub-System

We have implemented the communication part of the cluster simulator using a prototype TTP/C controller [6].

In conformance to Section 3 TTP/C is TDMA based and operates in a time-triggered fashion. Messages are sent according to a static control structure (i.e., the so-called message descriptor list (MEDL)) which defines which node is allowed to transmit in each specific TDMA slot.

In order to fulfill Obligation 1 the MEDL of the cluster simulator must define the cluster simulator as the active sender in each of its simulated slots.

Obligation 2 requires that the MEDL prohibits any transmission by the cluster simulator in any other slot than the ones to simulate.

Since the transmission speed on the bus and the length of each message are governed by the communications sub-systems static MEDL, Obligations 3 and 4 can be fulfilled by ensuring that both the transmission speed and the message lengths specified by the cluster simulator's MEDL match the transmission speed and the message lengths specified by MEDLs of all other nodes in the cluster.

Similar to Obligations 1 and 2 the fulfillment of Obligations 5 and 6 can simply be achieved by an appropriate MEDL configuration because the set of TDMA slots where a single node is allowed to integrate upon startup or after a node failure is statically defined in the MEDL as well. Thus it is sufficient to ensure that the MEDL of the cluster simulator allows integration in every simulated slot and prohibits integration in all other slots.

5.2. Host Sub-System

The host sub-system is implemented on an IP360 motherboard (an IP motherboard with a Motorola MC68360 CPU) [9] which hosts the TTP/C controller in one of its IP [1] slots.

From the software point of view the host sub-system consists of a generic *simulator core* which is tailored to the specific application using and appropriate *dispatching table*.

The host sub-system's operation can be divided into three distinct phases:

Download The host sub-system performs a download of the dispatching table from a workstation via an ethernet link.

Simulation This phase is the actual simulation phase where message data is copied into the CNI according to the dispatching table.

Upload After the simulation has finished, possible monitoring information is uploaded to the workstation.

Each of the three phases will be described in more detail in the following sections.

5.2.1. Download

During the download phase the communication subsystem is not yet activated. The only purpose of this phase is to transfer the dispatching table from a workstation (where the table is generated with an appropriate design tool) to the host sub-system of the cluster simulator. Currently a light-weight ethernet protocol [8] is used to accomplish this task. The download via standard protocols like FTP or HTTP however is under development.

5.2.2. Simulation

Once the dispatching table is present at the host subsystem (either because the transfer from the workstation has been completed or because the table has been

statically linked to the cluster simulator application) the communication sub-system is activated and the simulation phase is entered.

In the simulation phase the host sub-system works synchronously to the communication sub-system (i.e., the actions for each TDMA slot entry of the dispatching table are triggered with the start of a TDMA slot).

The dispatching table itself is defined in using a so-called *slot description language* which on the one hand can be translated into native C code, to be compiled and linked to the cluster simulation application, and on the other hand can be used to generate a core image of the table to be dynamically downloaded into the host sub-system.

Figure 3 shows part of a dispatching table as small example illustrating the use of the slot description language.

```
DEFINE_DATA_ENTRY(14, 0)
{
    0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
    0x88, 0x89, 0x8A, 0x8B, 0x8C, 0x8D, 0x8E, 0x8F
};

DEFINE_FUNCTION_ENTRY(14, 1, nSrcBytes, pSrcData,
                      nDstBytes, pDstData)
{
    int i;

    ASSERT(nSrcBytes == nDstBytes);

    for (i = 0; i < nSrcBytes; i++)
    {
        pDstData[i] = pSrcData[i] + 1;
    }
};

DEFINE_SLOT_ENTRY(14)
{
    DATA_ENTRY(14, 0, 0x0120, 16),
    FUNCTION_ENTRY(14, 1, 0x0132, 16, 0x0144, 16)
};
```

Figure 3. Example Usage of Slot Description Language

The example given in Figure 3 defines a slot entry for TDMA slot 14 consisting of a pre-defined 16 byte message which is copied to CNI location `0x0120` and a user-defined function which reads 16 bytes from CNI location `0x0132`, increments the value of each byte by one and stores the modified bytes in CNI location `0x0144`.

In addition to the generation of messages the host sub-system of the cluster simulator supports the recording of monitoring information (e.g., responses from the node under test) obtained from the CNI. This monitoring information can be stored using user-defined functions in order to facilitate an off-line analysis later on.

5.2.3. Upload

Once the simulation is finished an upload phase takes place during which the *recorded monitoring information* is transferred to the workstation (again using the light-weight ethernet protocol).

This monitoring information can be analyzed by appropriate tools in order to *validate the correct behavior* of the node under test. Additionally the monitoring information can be used by the system integrator to derive accurate *data for a dispatching table* for a cluster simulation including the the node under test. This dispatching table can then be provided to one of the other suppliers.

In this way an *iterative refinement* of the simulation starting from a rather rough simulation based on the functional specification of each node and ending in a precise simulation based on recorded monitoring data is supported (see Figure 4).

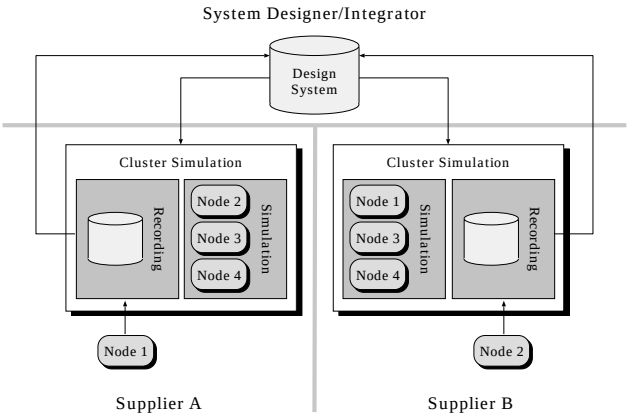

Figure 4. Iterative Refinement of Cluster Simulation

This approach matches the development process which successively refines the definition of the functionality of the different nodes from a pure interface definition to a detailed specification of the node-internal functions and tasks.

6. Evaluation

Within the Brite EuRam Project "X-By-Wire" [4] a steer-by-wire (i.e., electronic steering without mechanical backup) prototype application has been developed.

This application is distributed over three fault-tolerant units (FTUs) and consists of eight nodes in total which exchanged messages using the TTP/C protocol.

One FTU (consisting of three nodes for reasons of fault tolerance) is responsible for handling the sensors and actuators of the *road wheels*. An FTU consisting of two nodes services the *steering wheel* and another FTU (again made up of two nodes) is in charge of the *overall control loop*. In order to demonstrate the fault tolerance properties of the application a dedicated *monitoring node* has been introduced. Figure 5 shows a schematic representation of the steer-by-wire prototype.

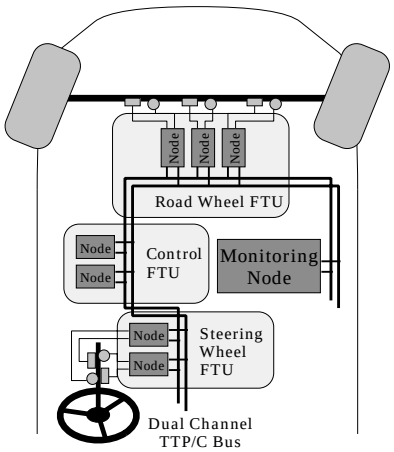

Figure 5. Steer-By-Wire Prototype (schematic)

Since the development of the different nodes has been distributed among the projects partners, support for the stand-alone test of a single node and simulation of the behavior of all other nodes is required.

The presented implementation of the cluster simulator has been successfully used for the development and testing of the monitoring node for the steer-by-wire prototype. In this scenario the cluster simulator had to simulate all nodes but the monitoring node itself.

The simulation encompasses the transmission of messages of the different simulated nodes to test whether the monitoring node displays the correct activity status of all nodes and the provision of pre-defined message data to check whether the monitoring node reads the messages from the right locations within the CNI and decodes the message data in the correct way.

Due to the use of the cluster simulator during the development and test phase it has been possible to validate the monitoring node's functionality prior to the actual integration of this node into the cluster of the remaining seven nodes. Thus a large number of implementation errors (e.g., accesses to wrong address offsets within the CNI) of the monitoring node has been detected and corrected, and the final integration of the monitoring node into the rest of the cluster has been performed with a minimum of effort.

7. Conclusion and Future Work

In this paper we presented an architecture for cluster simulation for TDMA-based time triggered real-time systems. The design objectives, the requirements and the problems of a such an architecture were discussed. Special focus was put on the problems resulting from the limited CPU power of the simulator. A detailed description on how these problems can be solved and how the requirements can be met was given. An example implementation based on TTP/C was described and the experiences with this implementation within a major project funded by the European Commission was given.

Currently an extension to a cluster design system to support automatic generation of the cluster simulation MEDLs is almost finished. Future work will focus on the development of a set of tools facilitating the generation of dispatching tables and on the interaction of the cluster simulator with off-the-shelf visualization tools.

8. Acknowledgments

This work was partly sponsored by the Brite EuRam Project "X-By-Wire" and by the Esprit OMI Project "TTA".

References

[1] GreenSpring Computers, Inc., 1204 O'Brien Drive, Menlo Park, CA 94025, USA. *IndustryPack Logic Interface Specification*, 1995. Revision 0.7.1.

[2] Vector Informatik. *CANoe – The Comprehensive Tool for CAN Projects*. Stuttgart, Germany, 1997.

[3] H. Kopetz. *Real-Time Systems*. Kluwer Academic Publishers, 1997.

[4] H. Kopetz, E. Dilger, L. Å. Johansson, M. Krug, P. Lidén, G. McCall, P. Mortara, B. Müller, U. Panizza, S. Poledna, A. Schedl, J. Söderberg, M. Strömberg, and T. Thurner. Towards an Architecture for Safety Related Fault Tolerant Systems in Vehicles. In *Procceedings of International Conference on Safety and Reliability (ESREL'97)*,

pages 1030–1021, Lisbon, Portugal, Jun. 1997. European Safety and Reliability Association, Elsevier Science Ltd.

[5] H. Kopetz and G. Grünsteidl. TTP – A Protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer*, pages 14–23, January 1994.

[6] H. Kopetz, R. Hexel, A. Krüger, D. Millinger, R. Nossal, A. Steininger, Ch. Temple, T. Führer, R. Pallierer, and M. Krug. A Prototype Implementation of a TTP/C Controller. In *Procceedings of SAE Congress 1997*, Detroit, MI, USA, Feb. 1997. Society of Automotive Engineers, SAE Press. SAE Paper No. 970296.

[7] H. Kopetz and T. Thurner. TTP – A New Approach to Solving the Interoperability Problem of Independently Developed ECUs. In *Procceedings of SAE Congress 1998*, Detroit, MI, USA, Feb. 1998. Society of Automotive Engineers, SAE Press. SAE Paper No. 981107.

[8] M. Kucera, I. Smaili, and E. Fuchs. *A Lightweight Ethernet Protocol to Connect a Time-Triggered Real-Time System to an INTERNET Server.* IOS Press, Washington DC, USA, 1998.

[9] MicroSys GmbH, Mühlweg 1, 82054 Sauerlach, Germany. *IP360 – User's Manual*, 4th edition, 1996.

[10] P. Puschner. A Tool for High-Level Language Analysis of Worst-Case Execution Times. In *Proceedings of Euromicro Workshop on Real-Time Systems*, pages 130–137, Berlin, Germany, June 1998.

[11] P. Puschner and Ch. Koza. Calculating the Maximum Execution Time of Real-Time Programs. *Real-Time Systems*, 1(2):159–176, Sep. 1989.

[12] P. Puschner and A. Schedl. Computing Maximum Task Execution Times – A Graph-Based Approach. *Real-Time Systems*, 13(1):67–91, July 1997.