

# Analyzing Automotive Networks using Virtual Prototypes

Sebastian Graf, Martin Streubühr, Michael Glaß and Jürgen Teich  
Hardware/Software Co-Design, Department of Computer Science, University of Erlangen-Nuremberg, Germany  
{sebastian.graf, streuebuehr, glass, teich}@cs.fau.de

## Abstract

Even small changes in automotive networks might have harmful impact to the correctness of the functionality due to changed timing behavior. In this paper, we present an approach to investigate the functional and timing-behavior of an automotive network at a very early design stage. To reach this goal, a virtual prototype of an automotive network is created. The prototype is used as a model for a simulation. The simulation allows a detailed analysis of the timing behavior of the system. As a case-study, a possible migration from a network architecture consisting solely of CAN and LIN to a FlexRay/CAN/LIN solution was investigated. We simulated different configurations and networking topologies and analyzed the results. It could be shown that it is advantageous to move components to the FlexRay bus, but certain issues concerning the configuration of the system, especially the networking topology, should be regarded carefully.

## 1 Introduction

The ever-growing complexity of electrical/electronic architectures (E/E architectures) becomes a major challenge for the design and analysis of future automotive systems. Key driver are advanced driver assistance systems like adaptive cruise control with high bandwidth demands, low response time and also the need for dependable timing. This forces significant changes in the network topology and, particularly, the communication infrastructure. One possible solution is the use of the FlexRay bus which offers high-speed communication with strict determinism. But the integration of new bus systems and the mixture of time-triggered and event-triggered components cause effects that may hardly be investigated manually. Changed timing behavior may affect the functional behavior and even lead to hazardous system states. Investigating timing properties, however, requires taking functional and architectural behavior into account concurrently. Given this situation, an efficient investigation requires the modeling and analysis of virtual prototypes assisting the designer during the development of the architecture.

Our approach is based on simulations. For the use in an early design time, a *virtual prototype* of the automotive system is needed. The required results, which are used for the analysis of timing behavior, especially the latency of control loops, are gathered during the execution of the simulation. Therefore, this work will present how the functionality can be modeled in an actor-oriented *functional network*. Furthermore the various possibilities for modeling the architecture in a *component network* are described in detail. The virtual prototype combines the functional and the component network and therefore gives the opportunity to evaluate the timing behavior in conjunction with functional behavior. After the execution of the simulation, the results, especially concerning the timing, need to be analyzed in detail. For that, methods for the evaluation and analysis of the timing behavior are presented.

The whole methodology will be shown with the help of a small case-study. The main goal of this study was to find an architecture that allows a control loop to lower its response time and to integrate a more dependable functionality than in the currently used configuration.

The rest of the paper is outlined as follows: Section 2 discusses related work. While the problem targeted in this paper is outlined in Section 3, the used approaches are presented afterwards. The concepts of modeling and simulation of the functional and the architectural parts are introduced in Section 4. The analysis of the automotive networks is presented in Section 5. Section 6 presents a case-study from the automotive domain while Section 7 discusses the experimental results. The work is concluded in Section 8.

## 2 Related Work

Recent publications cover the analysis of the functional behavior. For example, for AUTOSAR software components, KRAUSE ET AL. present a SystemC-based approach to simulate the timing on different layers of abstraction [1]. They developed transformation rules to migrate an AUTOSAR system description to an executable SystemC equivalent. They extend their SystemC-based model with existing transaction level modeling (TLM)-based models. Some timing dependent features were introduced in the TIMMO-Project [2], which extends the AUTOSAR standard. This extended standard was used in [3] where they present a possible tool-flow with integrated TIMMO-features within a case-study. KRAUSE ET AL. aim also at another goal in their publication [4], where they try to verify AUTOSAR based software by SystemC-based virtual prototyping. The virtual prototype is again built on TLM-based components and thus integrates abstract timing behavior in the model. But, in the case of verification, they require a fully specified AUTOSAR system. But this is normally not available at an early design time and, thus, is

just partly comparable with our presented work where high-level models are used.

Beside simulative approaches, timing could also be analyzed by static analysis. This is often based on WCET-analysis like shown in [5] and could be extended to real-time analysis of distributed systems, which is already available in commercial tools from Syntavision [6] or Inchron [7]. Some exemplarily work for this kind of analysis can be found in [8] and [9]. One disadvantage of these tools is that they need a complete specified and refined system that should be analyzed and, thus, it is difficult to use these approaches early in the design process. In contrast to these requirements, our approach is able to analyze high-level models and, thus, only needs abstract system configurations. Moreover, these configurations could be changed very quickly, so the analysis of the timing can give estimations about the expectable performance and functional behavior for various architectures.

### 3 Problem Definition

Timing and the dependability of guaranteeing timing constraints are major challenges during the integration phase of the E/E-architecture of a car and are tackled with various approaches. One major problem is that functionality is often designed without taking care of the later used architecture of the real system and, thus, uses ideally timing premises or assumptions about the expected timing-behavior. This development is mainly based on models built in MATLAB/Simulink [10], which are designed to work under specified timing behaviors and constraints (e.g. the control loop is executed every 100 ms and can handle an end-to-end latency of maximum 20 ms). Due to these assumptions, an implementation has to provide these requirements in order to guarantee a correct functional behavior. It can hardly be estimated what will happen if timing assumptions are exceeded or dropped.

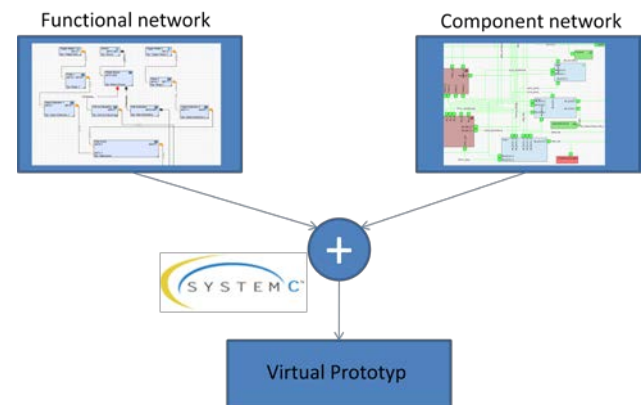
The architecture of the system is often dimensioned with respect to estimated or expected communication and computation demands, often driven by worst-case execution time assumptions. Due to these worst-case estimations, the architecture is in part over dimensioned (and thus more expensive). But the functionality implemented on this architecture might also be compromised by timing errors. E.g. a scheduling anomaly may cause unexpected communication-bursts, when the task execution times are significantly shorter than expected by WCET analysis. Then, the best-case for computation becomes the worst-case for communication. Other examples for timing related effects are unsynchronized communication partners, unexpected peak workload caused by a mixture of event- and time-triggered activations etc.

To avoid expensive and error-prone modifications late in the development process, mainly at the integration phase, analysis of the system should be regarded as soon as possible. Of course it is not possible to model the complete system with cycle-accuracy at this early stage, methods for abstract modeling of those systems at system-level

must be used. At this time only first versions or concepts of the functions as well as estimated execution times and abstract communication profiles between software components exist. The architecture is often also not fully developed at this time and might further be adapted in the future. But usually, it is possible to outline the currently planned networking architecture and ECU configuration. With this high-level information, it is possible to assist the developer with first estimations regarding the system and timing behavior. With this knowledge, it is also possible to give recommendations for the ongoing development process in order to improve the system configuration and lower the risk of error-prone systems.

## 4 Modeling and Simulation

The presented approach uses abstract simulation models at system-level. They are used to evaluate the system with respect to communication and execution times (see Section 5). The simulation models itself are based on the Y-chart approach introduced by GAJSKI ET AL.[11], which clearly separates the functionality from the architecture. As could also be seen in **Figure 1**, the functional network of the system, that emulates the functional behavior of the system, is modeled using a SystemC-based framework. In our case, the modeling uses an extended version of the SystemMoC library [12], which merges the abilities of SystemC with a well defined model of computation. It will be described in detail in Section 4.1. The functional network is mapped to a component network. The components are



**Figure 1** The Y-chart approach consists of the functional network and the component network. Together, both sides specify the abstract system representation in terms of a SystemC-based virtual prototype which can be used to evaluate the system or the timing behavior

responsible for modeling temporal behavior, representing task execution times and communication delays. Furthermore, they model scheduling of tasks mapped to ECUs and arbitration of shared buses. The modeling of the *E/E architecture* as a component network is based on the *Virtual Processing Components* [13], which were adapted to the requirements of the automotive networks and systems in this work. The required extensions as well as basic modeling concepts are presented in Section 4.2. Together,

both parts result in a virtual prototype of the system that is evaluated in case of timing.

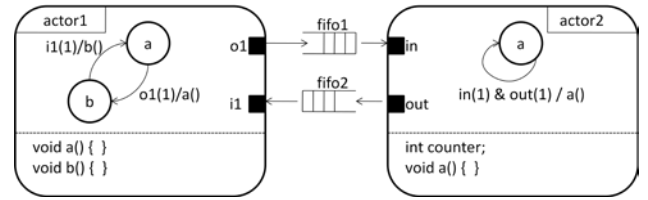
## 4.1 Functional network

Modeling the functional network uses an actor oriented model based on the SystemoC library [12]. In this model, functionality is encapsulated in *actors* which are connected via *ports* through *channels* (see **Figure 2**). Every actor contains a finite state machine (FSM) with one or more states. Below the dashed line, *actions* and internal variables are shown. Actors access the channels via named ports (black rectangles in Figure 2). Every transition in the FSM is annotated with an activation pattern and an action, separated by a slash. If all conditions in an activation pattern are fulfilled, the transition can be taken. If the transition is fired, the corresponding action is executed atomically. For example, the transition *in(1) & out(1) / a()* in *actor2* is activated if there is at least one token available in channel *fifo1* and channel *fifo2* is able to accept at least one token. If the transition is taken, action *a()* is executed. Afterwards, tokens are consumed and produced according to the activation pattern.

SystemoC was previously used for SoC development [14]. There, the primary used model of computation is dataflow with event-triggered activation. In automotive functional networks, dataflow is important too. Nearly all kinds of functions are based on such kind of activation pattern: as soon as input data is available, the functionality is executed. Moreover, Remote Procedure Calls, which are part of the AUTOSAR-standard, can be modeled with this kind of actors. But within the automotive networks, due to increasing demands for real-time behavior and dependability, more and more time-triggered execution of functions is used. Thus, some extensions to the actor oriented approach are required. To support the modeling of functions with time-triggered activation, we introduced a new category of actors: the *periodic\_actor*. It consists, beside a simple controlling FSM, of an initial release time (offset  $\tau_{offset}$ ) and a period  $\tau_{period}$ . So the time of the *n*-th activation of the actor  $\tau_{release,n}$  can be calculated by:

$$\tau_{release,n} = n * \tau_{period} + \tau_{offset}$$

Nowadays, the function network usually consists of several function blocks that are connected with other function blocks by *logical connections*. The function blocks mainly stand for dedicated software tasks, AUTOSAR software components or at least a runnable entity. This kind of function blocks and connections are well suited for a transformation to SystemoC actors and the logical connectors between the function block can be modeled in SystemoC by channels. Moreover, the functional network of current cars consists of a mixture of functions with varying activation patterns. Therefore, actors with different activations and various refinements can be used in the same model. Due to this, it is possible to implement a huge range of functionalities. The possibility of a further refinement also fits the demands of the development of the E/E-architecture at an early design stage. During the ongoing development process it is possible to advance the



**Figure 2** The SystemoC model consists of various actors that are connected via channels. Each actor has a finite-state-machine that controls its functional and communication behavior. The functionality is modeled within the methods of the actor (e.g. *b()* in actor1).

model. This gives the ability to simulate, analyze and verify the system behavior after each modification with advancing accuracy.

Another important part of the functional network are the channels. A channel is used to connect a pair of actors. Each channel exchanges data between two ports in terms of tokens. Usually the channel is modeled as a first-in-first-out buffer (*smoc\_fifo*) with limited capacity, which is comparable to the *client-server-communication* in the AUTOSAR specification. The channel is typed with standard C++ data types and data is stored as tokens. A sending actor produces a defined amount of tokens at the end of its execution and a reading actor consumes a defined amount of tokens from the channel. Thus, an actor can only be executed if there are enough tokens available on input channels and enough free space is available on output channels.

Another type of channel is the *smoc\_register*. It implements a register-like access scheme. Reading is performed non-destructive and allows reading a token multiple times. Writing is non-blocking and gives the writer the ability to rewrite a value. This kind of behavior of the channel is needed for the *sender-receiver-communication*, which is also a part of the AUTOSAR specification. Another area of use is the under sampling and oversampling of transferred packets, where a sent element might be read zero, one or even multiple times. This kind of access pattern is used within typical control-loops.

## 4.2 Architecture

The architecture is the second part of the Y-chart approach and is build from virtual processing components. The components are linked together via edges and, thus, build the virtual representation of the vehicles architecture. ECUs, sensors, actuators and even bus systems are modeled as dedicated components. Each component is able to manage the tasks mapped to it and the messages routed on it with the help of a scheduler.

Component scheduling and bus arbitration introduces additional wait cycles from resource contention as well as from task preemption. These effects cause variable execution and communication delays, known as jitter. For faster modeling, there are several different scheduling and arbitration techniques provided. A short, but not entire enumeration of some schedulers that help to abstractly model various components is provided:

- Priority based with and without preemption
- Time Division Multiple Access
- Round Robin
- First Come First Served
- FlexRay

With this selection of schedulers, it is possible to simulate ECUs, CAN, LIN, FlexRay and many other scheduling and arbitration techniques used in modern cars.

Especially the FlexRay scheduler with the ability of dual-channel operation, the mixture of time and event-triggered communication, and cycle-multiplexing behavior is very powerful and is used within this work. Beside this, nearly every kind of scheduling behavior can be implemented and be provided for timing simulation, so an extension to new domains (like infotainment with MOST-150 or Ethernet) is possible. The whole architectural description, used for enhancing the function network with timing properties, is included in an XML-based configuration file and augments the model of the functional network with an architectural network at the beginning of a simulation run. These two parts result in the virtual prototype of an automotive network, which is able to simulate the functional behavior including timing properties.

## 5 Evaluation and Analysis

After completing the functional (actor oriented) modeling and an architectural configuration, the simulation could be executed. This will simulate the functional behavior with respect to the modeled architecture. Results from a simulation run can be analyzed for several aspects. The results may be used for functional verification or for analyzing the timing behavior. The latter point is focus of this work. Thus, the presented techniques will primary take care of timing. Some of these methods were also used in our case-study, which is presented in Section 6.

Mainly two different steps are taken in order to assess the timing properties obtained from the simulation. The first step is the tracing and evaluation of the system during the execution of the simulation. In this step, required extensions to the simulation models were developed so that relevant data for the timing analyses can be captured. The second step processes the data obtained by the evaluation and therefore generates an estimation of the quality of the simulated solution. This step helps to compare the currently observed system with other solutions and can also be used to support the development for further system improvement in case of performance as well as in other objectives.

### 5.1 Evaluation

Part of this work was the definition of different evaluations which could be integrated within the simulation process. One possibility is to look at the functional behavior of the used control loops. If the actor-oriented modeling of the system is extended with an abstract control algorithm, possible faulty controlling can be detected during the simulation run. This is done by adding additional

timing information (timestamps) to the abstract message tokens. Due to high-level modeling, this is transparent and does not affect the behavior of the system and thus could be integrated without impact to the simulation results.

Beside the analysis on application level, another possibility is to evaluate the timing behavior on the networking level. This is done with analysis of the transferred messages on the networking components and the duration of a complete round-trip time (end-to-end delay) of the control loop. For this, two different methods for acquiring the required data were realized.

- Tracing of messages over the complete path through a network (communication delay).
- Monitoring utilization of the communication resources.

Both methods generate log files with the aggregated timing information. Beyond that, it's also possible to analyze the raw timing sequences using waveform viewers.

Important parameters for evaluating timing properties are latency  $L_{msg}(n)$  and jitter  $J_{msg}(n)$ . Both could be calculated with the captured data. Within this work, the following definitions are used:

$$L_{msg}(n) = T_{msg}(n, stop) - T_{msg}(n, start)$$

$$J_{msg}(n) = T_{msg}(n, stop) - T_{msg}(n-1, stop) - P_{soll}$$

For the latency of the n-th message, the related send and receive times ( $T_{msg}(n, start), T_{msg}(n, stop)$ ) need to be subtracted. The jitter is defined as the difference of the time between two successive receive-times and the awaited period. Additionally, it is also possible to analyze the utilization and workload of the resources, as well as the activation patterns of the functions by post processing the log files.

### 5.2 Analysis

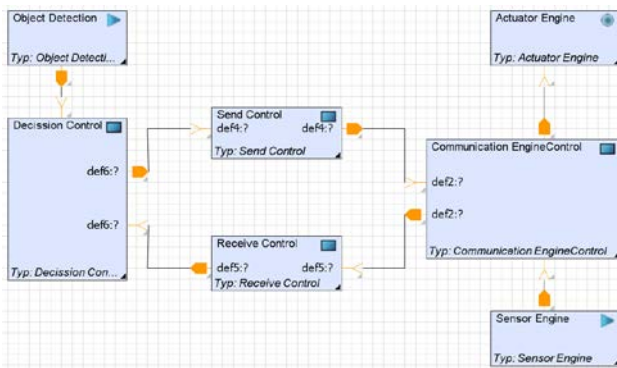
After collecting and evaluating the required data during the execution of the simulation, the results needed to be analyzed and construed. This is done by three different ways. The first one is the rating of the functional behavior. This consists, beside the correct functionality, of some specialized criterion which came from the application-side and won't be discussed here in detail. An example for this may be the constraint to undercut a maximum round-trip-time of 50 ms for a single control-loop. The other analysis methods are all based on the timing evaluations presented in Section 5.1. On the one hand, a quick reaction to a request is needed in order to archive a smooth and agile controlling. This results in the need of very low latencies and therefore requires a fast connection between the distributed and connected functions. This opportunity can be observed by the delay times of the involved messages and the measured end-to-end delay of the control-loop.

On the other hand, the latencies also need to be stable and, thus, predictable. For the stability of the control loops, it is better to have one solution with a predictable, but higher minimum latency then having a solution with a

very low latency, but having higher jitter values. In all these cases, given timing constraints concerning a maximum latency have to be fulfilled. But the main goal is typically to lower the timing values as an optimization criterion and thus enables the application developer to advance the control loop.

## 6 Case-study

To demonstrate the applicability of the presented approach, a case-study is presented. The problem definition is taken from the automotive domain and could be regarded as a system for adaptive cruise control and a brake by wire application. An abstract model of a controlled system in the power-train domain was modeled and examined. After the abstract implementation of the algorithm (see **Figure 3**) and the refinement of the functional behavior, the main goals were to evaluate the timing behavior of the function network under various architectural mappings. This is done in order to assess the effects caused by various latencies of the communication network. The model of a single control loop consists of two actors with functionality, one is activated time-triggered within the radar and (in case of a change in the detected volume) sends requests for changes of the current state of the engine (e.g. lower torque and rpm) or the braking system over the channel to the receiving actor. The succeeding actor receives and processes these requests and thus is activated by the communication events. This task prepares the requests for the engine control which will be regarded by the next injections or for the braking control. Because the injections are released at common angles of the camshaft, the execution of the adaption could be further delayed by an asynchronously triggered function. To improve the model, additional actors that represent the software-stacks were added in the functional network. In addition to the regarded control functions, other messages and function tasks were abstractly defined to model the remaining busload and the use of various ECUs.



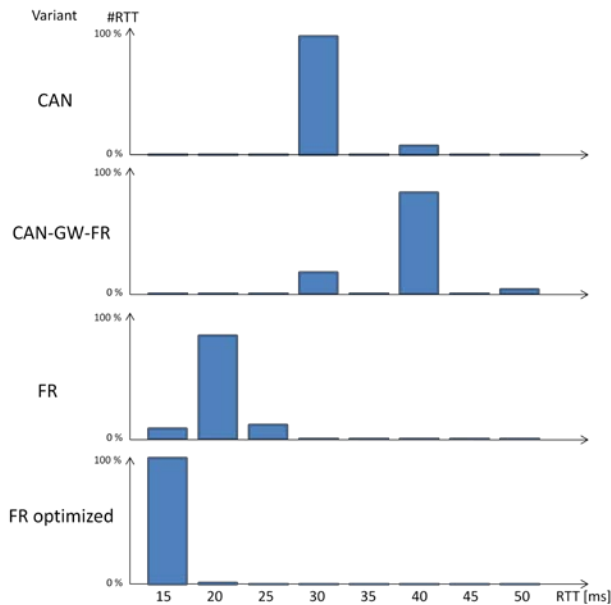
**Figure 3** Simplified example functional network of the used case-study. The observed control loop from a sensor task (*Object Detection*) to an actuator function (*Actuator Engine*) and reverse. One part of the system is angle controlled; others are time-triggered and event-triggered. One important timing property is the round-trip-time of the control loop. This indicates the reactivity of the system.

To prove the usefulness of the introduced simulative approach, a state-of-the-art architectural template was used as a basis for the different explored configurations. The architecture superset consists of various time- and event-triggered busses connected via a centralized gateway and various possible links to the ECUs. For the deeper look insight the functionality, mainly parts that were associated with ECUs from the power train domain and the gateway were modeled. The example was evaluated using different networking architectures, which were extracted from the inflated architecture and had various function mappings. Moreover, different scheduling and/or configurations of the busses were used to check their influence on the timing. To respect the effects from other networking components, important messages from other components or functions, which are not integrated in the functional network, were integrated as abstract messages. This implements an abstract kind of restbus simulation to consider the network traffic of all connected components. The deeper look insight the time-critical control loops gave us the possibility to evaluate the behavior in more detail. In addition to that it was possible to check the expectable consequences to the functional behavior.

The basic version, as a kind of reference version to which every other solution should be compared, was an architecture where all ECUs are connected to a CAN-Bus with a data-rate of 500 kbps. So the first task was to identify the given constraints concerning the timing in the currently applied system. After that, the simulations and analysis were performed with changed system-configurations and different networking architectures and evaluated equally. The main goal was to enhance the performance of the control loop.

## 7 Experimental Results

The results of the presented case-study support the architectural migration of some components from the CAN to a newly integrated FlexRay bus and considers taking care of some important conditions for the faultless integration and a further improvement of the timing behavior. For that we calculated the round-trip-times of the most important control loop and extracted the minimum and maximum values as well as the frequency of occurrence. For further analysis of the results, a deeper look insight the timing property of the tasks and messages was taken. Based on these results, an assessment of the various configurations was made to score the solutions. Some of the observational results regarding the timing behavior are shown in **Figure 4**. The first figure shows the situation of the latencies of the control loop in a CAN-based system. It could be seen that some constellations can be regarded where the expected value of 30 ms in some cases was surpassed, so it is a bit difficult for the designers to optimize the system for a fixed delay. This mainly consists of the time-triggered activation of the function network on an unsynchronized architecture and the existence of higher priority CAN-messages. A possible solution with a



**Figure 4** Normalized Round-trip-times with given statistical values are shown for different configurations. It can be seen that the architecture and the configuration of the busses has great impact to the timing.

communication from the CAN-bus over the gateway to a FlexRay and reverse will even worsen the timing behavior. Just migrating everything to the FlexRay bus will lower the mean value of the obtained control loop, but the distribution is contrary to one of the objectives of the optimization. As a deeper look insight the timing behavior shows, the newly proposed architecture (*FR optimized*) is able to achieve tremendous decrease of the latency and beyond that is able to guarantee the occurrence of that latency. To reach that goal, the communication network needs to be changed from CAN to FlexRay with an optimized configuration. FlexRay needs to be configured with a cycle time of 5 ms and the scheduling of the static slots needs to be optimized. In addition, the involved ECUs must be synchronized with the FlexRay bus to let the communication tasks exactly hit their configured static slots. If that is not done, some unintended effects like escalating jitter of the message delay would occur.

## 8 Conclusion

As presented in this paper, our new methodology can build a virtual prototype of an automotive network. It has the ability to evaluate the timing-behavior of various function networks under different networking architectures. This is very important, especially if a migration of a system to a newly developed networking architecture is planned. The early modeling of a simulation-assisted validation of the correct functional behavior helps to detect possible error-prone configurations at a very early design stage and thus helps to save money by preventing cost-intensive redesigns. Moreover, it is possible to further refine the models during the development process and thus

allows an ongoing check of the correctness of the selected system configuration with increasing accuracy.

## 9 Acknowledgment

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) under project 01BV0914-SEIS.

## 10 Literatur

- [1] M. Krause, O. Bringmann, A. Hergenhan, G. Tabanoglu, and W. Rosentiel, "Timing simulation of interconnected AUTOSAR software components," in Proceedings of DATE 2007, pp. 1–6.
- [2] TIMMO (TIMing MOdel), "<http://www.timmo.org>."
- [3] K. Klobedanz, C. Kuznik, A. Thuy, and W. Mueller, "Timing modeling and analysis for AUTOSAR-based software development: a case study," in Proceedings of DATE 2010, pp. 642–645.
- [4] M. Krause, O. Bringmann, and W. Rosenstiel, "Verification of AUTOSAR Software by SystemC-Based Virtual Prototyping," in Hardware dependent Software. Springer Netherlands, Jan. 2009, pp. 261–293.
- [5] S. Byhlin, A. Ermedahl, J. Gustafsson, and B. Lisper, "Applying static wcet analysis to automotive communication software," in Proceedings of the 17th Euro-micro Conference on Real-Time Systems 2005, pp. 249–258.
- [6] Syntavision, "<http://www.syntavision.de>."
- [7] Inchron, "<http://www.inchron.de>."
- [8] T. Kramer and R. Mützenberger, "Modellierung und Echtzeitanalyse komplexer Wirkketten in Fahrerassistenzsystemen" in Proc. of 3. AutoTest, 2010.
- [9] N. Feiertag, K. Richter, J. Nordlander, and J. Jonson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in Proceedings of RTSS 2008.
- [10] The MathWorks, MATLAB/Simulink "<http://www.mathworks.com/products/simulink>"
- [11] D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin, High-level synthesis: introduction to chip and system design. Norwell, MA, USA: Kluwer Academic Publishers, 1992.
- [12] SysteMoC, "<http://www12.informatik.uni-erlangen.de/research/scd/systemoc.php>"
- [13] M. Streubühr, J. Falk, C. Haubelt, J. Teich, R. Dorsch, and T. Schlipf, "Task-Accurate Performance Modeling in SystemC for Real-Time Multi-Processor Architectures," in Proc. of DATE 2006, pp. 480–481.
- [14] J. Keinert, M. Streubühr, T. Schlichter, J. Falk, J. Gladigau, C. Haubelt, J. Teich, and M. Meredith, "SYSTEMCODESIGNER - An Automatic ESL Synthesis Approach by Design Space Exploration and Behavioral Synthesis for Streaming Applications," ACM Transactions on Design Automation of Electronic Systems, vol. 14, no. 1, pp. 1–23, 2009