Early Architecture Exploration with SymTA/S

Kai Richter, Marek Jersak, Rolf Ernst

Institute of Computer and Communication Network Engineering
(Institut für Datentechnik und Kommunikationsnetze, IDA)
Technical University at Braunschweig
Hans-Sommer-Strasse 66
D-38106 Braunschweig
Germany
kai.richter@tu-bs.de

Abstract: With increasingly parallel development of a system's hardware-software architecture on the one hand, and its functionality on the other hand, system integration, verification, and test happens ever later in the design process. In order to ultimately avoid costly re-designs, the system architecture has to more or less meet all requirements on the first try. In other words, the system architects face the challenge to make sufficiently good estimates and choices very early in the design when the implementation is not yet or -in case of re-used or supplied parts- at most partially available. This paper addresses the major limitations of the state-of-the-art benchmarking approach and outlines a structured and systematic architecture evaluation procedure. Based on the SymTA/S tool, the proposed approach explicitly supports estimated data and thereby enables a variety of architectural options to be explored and optimized in early design stages.

1 Introduction

Designing an embedded system is a complex task, and designers face a large variety of serious design challenges. Even before the functions are actually implemented, system architects have to select an appropriate hardware-software architecture out of the large number of available embedded controllers and networks, buses and memories, operating systems and drivers, basic software and libraries, sensors and actuators, etc.. This architecture has to meet a large variety of requirements. Key questions include: Does the communication framework provide the necessary bandwidth? How much bandwidth is necessary? Do the processing units have sufficient computation performance? Can all timing and performance constraints such as end-to-end deadlines be met? Is the power-consumption sufficiently low? Can the system be manufactured at a competitive price? And many more...

Selecting the right components is critical. Over-dimensioning the architecture increases the price and reduces market share. Under-dimensioning the architecture increases the risk of violating performance constraints, thus compromising product quality, and again reducing market share. This shows that the early architectural choices have a dominant impact on the success (or failure) of the project. Essentially, system architects must make sufficiently "good" choices, otherwise the project will simply fail to reach the expected profit.

1.1 How to Make Good Choices Early?

Let us take a brief look at the related field of performance verification. The ITRS [ITRS03] names system level performance verification as one of the top-three IC design issues. The same problem has been recognized by the “AUTOSAR development partnership” (www.autosar.org), in which large parts of the European automotive industry aim at establishing an open standard for automotive E/E architectures. The leading German electronics magazine [Ar04] says “networking and the increasing software complexity pose key challenges on future automotive system design, and requires re-consideration of integration practice, and co-operations”.

Today, satisfaction of performance constraints is checked as a side-effect of functional verification and test, which requires the system to be (almost) fully implemented. Consequently, performance verification happens late in the design process. This increases the risk of late architectural changes, which introduce costly delays and can be project-killing in the worst case. Therefore, a key question is what system architects can do to explore system configurations and gather representative data about the quality of alternative choices early, when the implementation is not yet or -in case of re-used or supplied parts- at most partially available? The answer is simple: if detailed data is not available, system architects must use estimations. Later, however, when more detailed about the implementation become available, it must be possible to seamlessly refine the estimation results.

1.2 Possibilities for Early Estimation

We use an example from the automotive industry. An ECU (embedded control unit) supplier such as Bosch, SiemensVDO, Magneti Marelli, etc., wants to evaluate new processor developments from several competitors (e.g. Infineon, Motorola, Texas Instruments, Phillips, ...). Each semiconductor vendor offers a certain core operating at a range of frequencies with a choice of configurable peripherals and coprocessors. The memory structure including cache is also open and configurable.

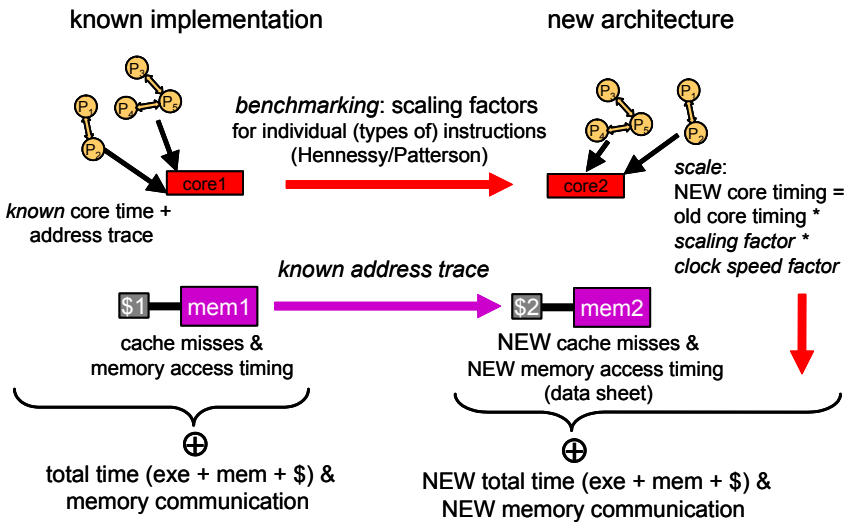

Figure 1 Benchmarking yields timing and memory access patterns of new architecture.

Experienced system architects can project the performance of a known implementation from a previous project to an unknown processor core and new memory configuration. Figure 1 shows how benchmarking helps considerably to derive scaling factors for individual types of instructions or basic functions that allow predicting the timing on the new hardware. In addition, known memory access traces can be re-used and analyzed with new cache hit/miss models to predict the new memory timing.

1.3 Limitations

Unfortunately, benchmarking as described in the previous paragraph is practically limited to simple architectures such as 8 to 16 bit CISC microcontrollers with simple memories. But automotive ECUs are far from simple. For instance, the popular Infineon TriCore and the Motorola MPC555 (see Figure 2) incorporate multiple bridged buses that connect pipelined 32-bit RISC cores to co-processors, caches, partially independent peripherals, and several external memories. Multi-processor ECUs are about to entering the markets. Each ECU, however, forms only one node in today's distributed automotive networks. With such increasing system complexity, the mutual influences due to caching, scheduling, peripherals, bus contention etc. result in ever less reliable estimations and benchmarking is eventually replaced by "guesstimation".

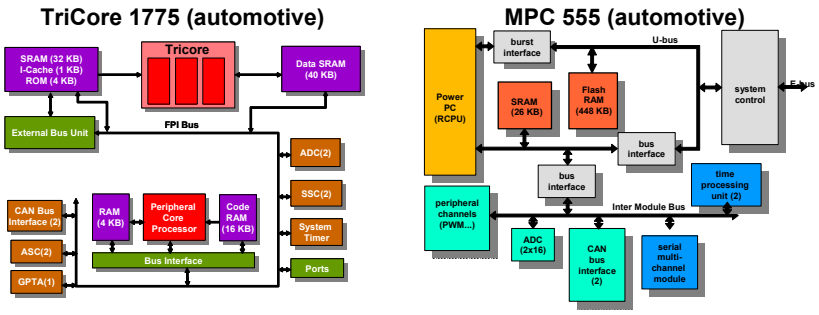

Figure 2 Two complex controllers popularly used in automotive systems.

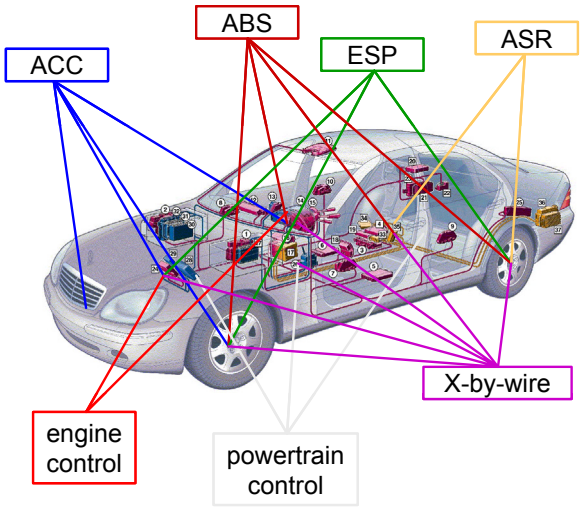

Figure 3 Automotive platforms are heterogeneous, highly integrated, multi-vendor systems.

The problem is in fact a lot worse for automotive OEMs who need to integrate the resulting car platform of heterogeneous ECUs from several competing suppliers, distributed applications with functions partitioned onto several ECUs, and multiple bridged networks running variety of protocols (CAN, LIN, MOST, TTP, Flexray). Furthermore, each car platform has several versions and configurable variants. Is it obvious that benchmarking is not applicable anymore, and even experienced designers can at most roughly guess about the system performance, simply because the variety of dependencies can not be fully overseen by anyone in a design team. Figure 3 illustrates the partitioning or distribution of functions such as automatic cruise control, electronic stability program, and others.

If we look at verification, we can observe similar challenges, even if the entire car platform is fully specified and implemented. Performance simulation and/or test suffer from increasing corner-case coverage problems. The large number of complex perfor-mance dependencies leads to corner cases and bottlenecks that are extremely difficult to find and debug, and it is even more difficult to find test patterns to cover them all. In other words, today system-level performance verification has become a second design bottleneck.

2 What Else can We Do?

We have seen that the individual pieces of a complex system are manageable in the small, and that platform and system integration is the major source of complexity. This indicates an urgent need for a systematic, structured procedure to handle system performance estimation. We have thoroughly researched this area for several years with a particular focus on practically useful approaches. We recognized that the real-time systems community has developed a variety of formal (i.e. systematic) techniques to structure the entire problem, and concluded that a layered approach is the most promising solution.

Figure 4 shows four architectural levels of complexity. The mentioned benchmarking and projection strategies can be adequately applied at the bottom level where individual task timing and communication is separated from the complex architectural influences. Alternatively, formal WCET (worst-case execution time) analysis can be applied, as proposed by Wolf [Wo02]. AbsInt provides the "aiT" WCET analyzer tool (http://www.absint.com/wcet.htm), that combines abstract interpretation and detailed pipelines models.

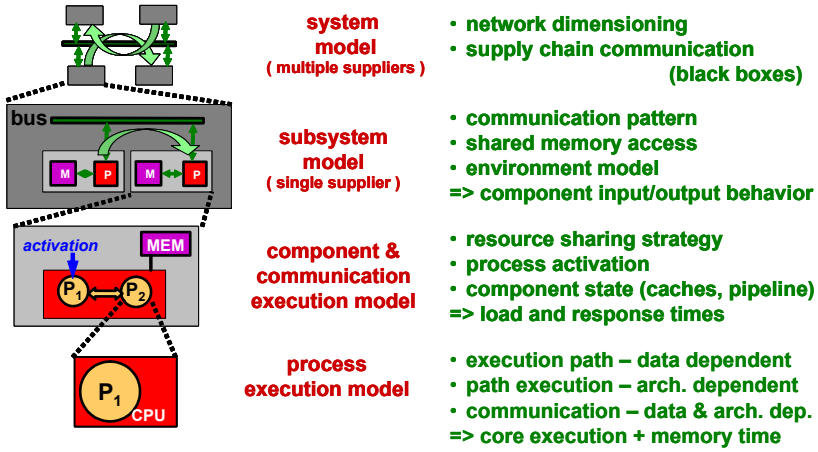

Figure 4 Four structured levels of architecture performance estimation

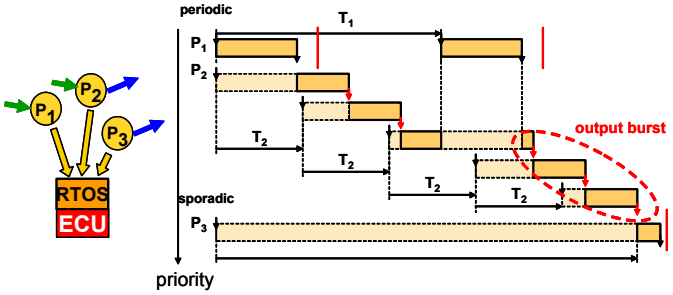

Figure 5 Scheduling diagrams visualize the influence of operating systems on task timing,

The next level (component & communication) already includes mutual dependencies between several tasks including scheduling by an operating system, cache dependencies, and shared-peripheral access. This makes benchmarking less appropriate. Instead, there exist promising approaches, some of them already known for some decades [LL73, JLT85], that use abstract task and activation models and formal analysis methods to determine processor and bus load, task and communication response times, frequencies and jitter, and sometimes the remaining component flexibility. Figure 5 shows the scheduling diagram of a system with three tasks that are scheduled periodically. Scheduling is preemptive and follows static priorities. The highest-priority process P_1 preempts P_2 and P_3, resulting in a complex execution scenario exhibiting jitter and burst process outputs.

Detailed operating system models are starting to become available, and a few real-time analysis tools have already been established, especially in the automotive area. Examples include the Real-Time Architect tool family from LiveDevices (an ETAS Company: http://en.etasgroup.com/products/rta/index.shtml) and Vectors CANalyzer (http://www.vector-cantech.com/products/canalyzer.html). As an additional benefit, these techniques do not require the system to be fully implemented but can also use estimated data, e.g. early estimations of task execution times. This considerably supports system architects during architecture exploration.

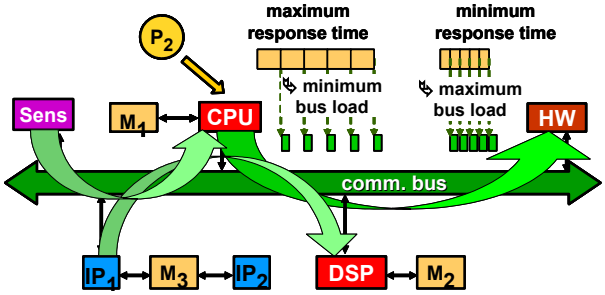

Figure 6 Scheduling anomalies can result from system integration.

For a long time, there was no support for the two remaining, most complex levels in Figure 4, namely subsystem-integration with multiple processors, and system-level integration along the supply-chain. Overseeing the impact of multi-ECU or multi-processor integration that access other peripherals has been a practically unsolved problem requiring detailed I/O patterns to be known to detect overload situations and resolve so-called scheduling anomalies (shown in Figure 6), identify bottlenecks and dimension networks and buffers. The problem is even worse at the system level, where only little internal component details are known due to IP protection. Because of the corner-case coverage problem, neither simulation, nor prototyping, nor test provide sufficient estimation and verification support.

Figure 6 illustrates a so called scheduling anomaly which illustrates the complexity of the overall task of performance analysis and estimation. Recall the P_3 bursts from Figure 5 and consider that P_3's execution time can vary from one execution to the next. There are two corner cases: the minimum execution time for P_3 corresponds to the maximum transient bus load, slowing down other components' communication, and vice versa.

3 SymTA/S - A New Technology

We have recently developed a technology and a tool called SymTA/S (http://www.symta.org) that brings approaches from real-time analysis theory to the system level by an intuitive global event flow modeling and analysis technique. SymTA/S does not require fully detailed system specifications but can use estimated data such as task execution times or communication volume. Alternatively, bench-marking, simulation and WCET-analysis can provide more accurate numbers (bottom level in our figure).

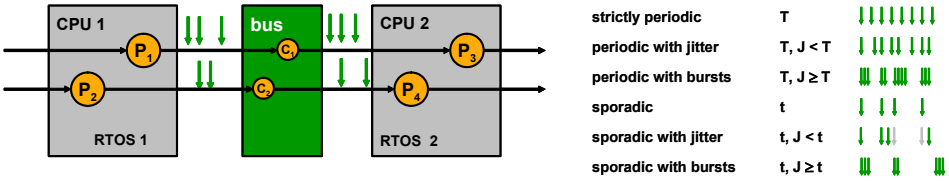

Figure 7 The use of event stream models and their classification.

At the next (component) level, SymTA/S uses approaches from real-time analysis theory to consider scheduling and arbitration dependencies. Only a small number of parameters such as priorities or time slots are sufficient to provide meaningful information about resource utilization, bandwidth and response times. We have mentioned that tools are available at this level (level 2), but SymTA/S goes much further. It extracts key information from a given schedule and determines the production of system workload, e.g. packets, interrupts, and communication patterns. These influence the global inter-actions between the components at the system-level (levels 3 and 4), and must essentially be analyzed in order to comprehensively capture the system-level dependencies.

In order to keep track of these dependencies which can usually not be fully overseen by anyone in a design team, SymTA/S uses intuitive workload models or "event stream models" [RE02] that can be used for both scheduling analysis and network analysis. These models capture abstract interaction timing properties such as periods, jitters, and bursts, and provide an adequate understanding of the dynamic system behavior without requiring internal details. Hence the approach is also applicable to black-box integration analysis (level 4).

Figure 7 illustrates the application of event stream models to capture the interaction timing between components in the system, processes P and channels C in the example. We define two classes of models, periodic and sporadic, with three models in each class: strict, with jitter, and with burst. This six-class model set is an efficient compromise between model simplicity and completeness, since these models are sufficient to cover a wide range of systems in practice.

Controlling the properties of these streams when integrating several tasks and subsystems is key, since they allow system-level performance corner-cases to be found, and bottlenecks to be identified, e.g. overload situations and constraint violations. Based on "event streams", SymTA/S identifies buffer overflows and missed deadlines as a result of transient overload. The influences on other components are automatically detected and propagated further [Ri02].

System architects directly benefit from the ability to use estimations in several places such as task execution times, amount of communicated data, communication patterns, etc... These parameters need not be fixed, since SymTA/S uses interval notations with upper and lower bounds from which the system-level corner cases are systematically derived and analyzed.

Furthermore, SymTA/S explicitly supports the exploration process because it is very flexible with respect to the amount of architectural details. System architects can focus only on their upfront design issues while ignoring unnecessary details such as the pipelining effects or the final bus width. Quite to the contrary, system parameters as well as resource configurations including priorities and mapping of tasks to resources can be changed freely. Since SymTA/S runs extremely fast, it allows evaluating a large number of different architectural choices. SymTA/S supports optimization through a variety of methods that automatically search the design-space for promising solutions based on hard constraints and optimization criteria. The design-space can be freely configured by the user to focus on certain aspects, or to omit certain alternatives because parts of the system have already been fixed. An overview on SymTA/S can be found in [Ha04].

4 The SymTA/S Tool - A Short Overview

An easy-to-use GUI allows to configure the analysis in SymTA/S. Figure 8 shows a screenshot of the tool. The user tasks and communications are edited and connected in the main "drawing area". The environmental is modeled as virtual source and sink tasks.

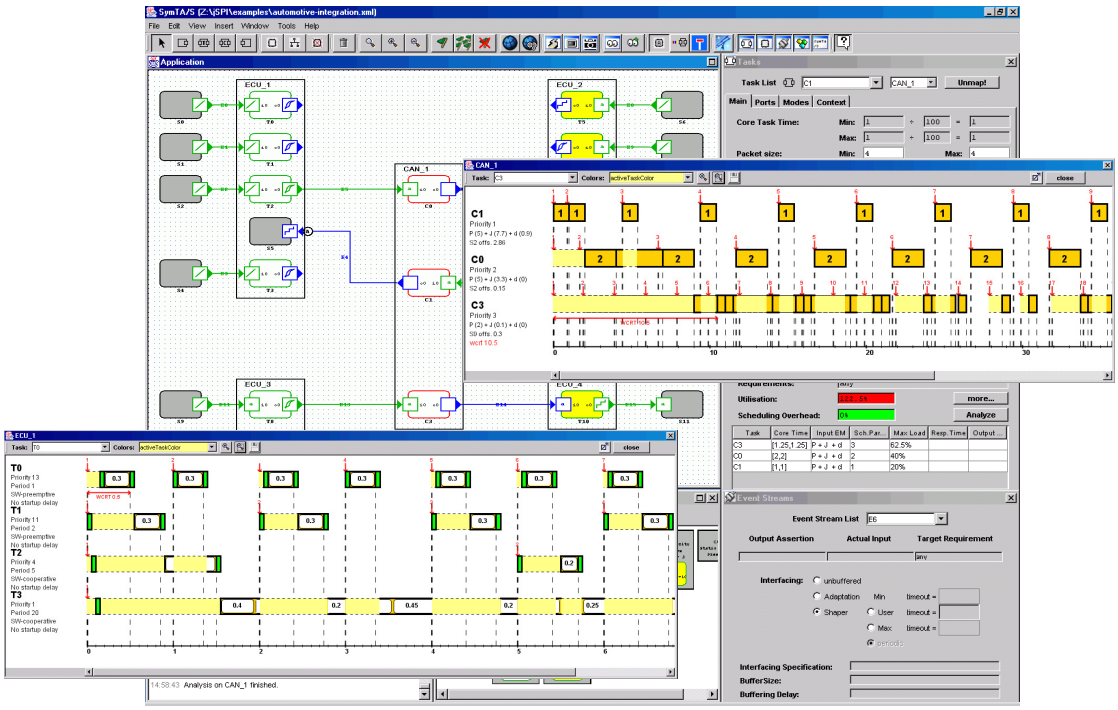

Figure 8 SymTA/S Screenshot

Few parameter windows allow the task parameters such as the core execution time and scheduling parameters as well as bus protocol and operating system configuration, scheduling strategy, OS overhead, etc.. to be modified. Different OS and protocol types are available as analysis libraries. The analysis library currently contains abstract scheduling models for the most popular and practically used process scheduling and bus arbitration policies. Specifically, these are preemptive and non-preemptive priority-based scheduling, time-division multiple access (TDMA), and Round Robin. Detailed. Models of real-world operating systems and bus protocols can be included as library elements. So far, we have done this for an OSEK based operating system and the CAN bus protocol. Interfaces to established Tools are currently being developed.

The analysis fully hides the mathematical background from the user, so she/he can concentrate on the key tasks, that is integration, analysis, and optimization. In order to allow fast and easy tool control, key information about the analysis status (schedulable or not) visualized using an intuitive color code: green=success, red=failure. The timing parameters of central concern such as response times, buffer sizes, and jitters can optionally be shown as tool tips when the mouse is moved over a specific element in the drawing area.

In addition to the specific system properties such as task deadlines, global response times, buffering delays, and resource load and utilization, SymTA/S generates worst-case scheduling diagrams, that visualize the complex interdependencies and thereby dramatically increase user understanding. Two scheduling diagrams are shown in Figure 8.

Due to its analysis speed --turn-around times are in the range of seconds-- a variety of different architectures, task mappings, and scheduling decisions can be explored quickly. A one-click mechanism allows to e.g. toggle priorities or change time-slots. And we have added automatic system optimization to SymTA/S that uses highly efficient genetic algorithms.

Finally, the sensitivity analysis plug-in in particular supports exploration and optimization in two ways. Firstly, it detects unused performance reserves in *working* systems and analyzes how the system properties can be modified whilst still meeting all constraints, e.g. how much a task's run-time is allowed to exceed its specification without violating some deadline. Secondly, it detects possible sources of overload and non-schedulability in *non-working* systems by providing information about which task, CPU, or bus must be optimized by what amount to make the system working, e.g. a CPU might be required to run 50% faster to meet all constraints.

4 Conclusions

Early architecture exploration is a critical task with a huge impact on the success (or failure) of a design project. It requires appropriate estimates of the expected architecture performance for a specific application. The state-of-the-art benchmarking approach, however, can not cope with the increasing complexity of today's systems, and more systematic and structured approaches are needed.

The SymTA/S approach provides this structure, at the same time requiring only few key parameters which can be provided as estimates. For the reasons mentioned above (can use estimated data, allows abstraction from details, supports exploration of alternatives, and provides quick evaluation of architectural changes) SymTA/S is a promising technology for system architects in the early exploration process. And once critical decisions have been made, the SymTA/S specification can be communicated along component supply chains to support the performance verification process throughout the whole design cycle. Models can be refined as new implementation details become available, allowing SymTA/S to verify implementations and detect critical bottlenecks earlier than simulation-based and benchmarking techniques.

We have successfully applied the tool and the technology to several verification and exploration problems in automotive, telecommunications, and multimedia industries where we could detect and solve serious system integration problems. We consider our approach to be a serious alternative to performance simulation and test. The new technology allows comprehensive system integration and provides reliable performance estimates extremely early and with very little computation time.

References

[Ar04] Heinz Arnold. Thema der Woche (topic of the week): Automotive electronics, in German. Markt & Technik, (36):16–20, 2004.

[ITRS03] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors, 2003,* http://public.itrs.net/Files/2003ITRS/Design2003.pdf.

[Wo02] F. Wolf, *Behavioral Intervals in Embedded Software*, Kluwer Academic Publisher, Boston, 2002.

[LL73] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real time environment. *J. ACM*, vol. 20, no. 1, pp. 46-61, 1973.

[JLT85] E. Jensen, C. Locke, and H. Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems, *Proc. 6th IEEE Real-Time Systems Symp*. (RTSS85), IEEE Computer Society Press, Los Alamitos, Calif., 1985, pp. 112-122.

[RE02] K. Richter and R. Ernst, "Event Model Interfaces for Heterogeneous System Analysis," *Proc. Design, Automation and Test in Europe Conf.* (DATE02), IEEE CS Press, Los Alamitos, Calif., 2002, pp. 506-513.

[Ri02] K. Richter, D. Ziegenbein, M. Jersak, R. Ernst, "Model Composition for Scheduling Analysis in Platform Design," *Proc. Design Automation Conf*. (DAC02), ACM Press, New York, 2002, pp. 287-292.

[Ha04] A. Hamann, R. Henia, R. Racu, M. Jersak, K. Richter, R. Ernst. "SymTA/S - Symbolic Timing Analysis for Systems". In *WIP Proc. Euromicro Conference on Real-Time Systems 2004 (ECRTS '04)*, pages 17-20. Catania, Italy, June 2004.