Real-time Management in Emergent Systems

Steffen Stein, Arne Hamann, Rolf Ernst
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig
D-38106 Braunschweig / Germany
{stein | hamann | ernst}@ida.ing.tu-bs.de

Abstract: Integrating new functionality into complex embedded hard real-time systems requires considerable engineering effort. Emerging formal analysis methodologies and tools from real-time research assist system engineers solving this integration problem. For future organic computer systems, however, it is desirable to integrate these approaches into running systems, enabling them to autonomously perform e.g. online acceptance tests and self-optimization in case of system or environmental changes. This results in high system robustness and extensibility without explicit engineering effort.

In this paper, we present an approach adapting formal compositional analysis techniques to realize self-awareness and self-adaptation in embedded systems with respect to real-time properties such as latency constraints, buffer sizes, etc.

We introduce a framework for distributed online performance analysis running on embedded real-time systems. Based on this framework we implement an acceptance test for the integration of new functionality into an existing embedded real-time system. Furthermore, we present an online optimization algorithm based on the same framework. In a case study, we demonstrate the applicability of the approach and show that online optimization can increase the acceptance rate with reasonable computational effort.

1 Introduction

Organic computing systems show emergent system behavior and architecture. With growing system complexity the benefits of emergence, such as self learning, self healing, robustness etc., let organic computing become interesting even for applications with real-time or safety requirements. Such applications require controlled emergence, i.e. the system must comply with fundamental constraints, such as deadlines or minimum dependability.

This paper will focus on the real-time properties of organic computing systems. In many applications, simple process deadlines are insufficient. Rather, end-to-end deadlines, jitter, and transient load requirements are needed for high quality media or control engineering. Furthermore, a networked system will integrate many functions on the same system platform, many of them with emergent behavior. The organic computing system will have to control the system emergence in such a way that these complex timing constraints are met at all times. Here, we can identify three challenges. The system must be aware of the current architecture and the current set of applications on the organic computing system, it must anticipate the effect of a planned emergence and, finally, there must be an online mechanism to control emergence.

Many online algorithms for real-time analysis have been proposed [FGB01] because they are often needed for dynamic scheduling strategies. However, complex timing constraints for embedded systems are usually verified by simulation or prototyping since current approaches to online timing analysis do not cover heterogeneous networked systems. Those techniques, however, are not applicable in organic computing systems.

In the last couple of years, offline analysis algorithms [Ric04, CKT03] have been proposed that can analyze such complicated systems and constraints requiring access to the global system state. They have been used in system sensitivity analysis [RHE06, BNB06] and system optimization [HJRE06]. In this paper, we propose a first step to extending such analysis to an online analysis and optimization system for organic computing systems.

The remainder of the paper is organized as follows. Section 2 introduces the basic concepts of offline performance analysis used as basis for our online analysis approach. Afterwards, in Section 3, the underlying ideas of our distributed performance analysis are presented. In Sections 4 and 5 we then introduce algorithms needed to perform an acceptance test and self-adaptation for integrating new functionalities into a system. In Section 6 we demonstrate the applicability of the approach and show that self-adaptation can increase the acceptance rate.

2 Offline Compositional System-level Performance Analysis

The compositional performance verification methodology proposed by Richter et al. [Ric04, HHJ+05], called SymTA/S, solves the global system-level performance verification problem by alternating (classical) local scheduling analysis and *event model propagation* along *event streams* connecting functional dependant tasks.

Event models describe the possible I/O timing of tasks. *Input event models* capture event patterns leading to task activations. Based on *input event models* for all tasks mapped on a resource, local scheduling analyses can derive response times, i.e. the time between task activation and task completion, and *output event models* for each task. These *output event models* are propagated to connected resources, where they are used, in turn, as input event models for local scheduling analysis.

A popular event model is the so-called *standard event model*. For example, a *periodic* event model has one parameter $\mathcal{P}$ and states that each event exactly arrives periodically every $\mathcal{P}$ time units. This simple model can be extended with the notion of jitter, leading to a *periodic with jitter* event model. Such an event model is described by two parameters $(\mathcal{P}, \mathcal{J})$. It generally occurs periodically, but it can jitter around its exact position within a jitter interval $\mathcal{J}$. If the jitter is larger than the period, then two or more events can occur at the same time, leading to bursts. To describe a *bursty* event model, the *periodic with jitter* event model can be extended with a d^- parameter that captures the minimum distance between events within a burst.

More details about the compositional system-level performance analysis can be found in [Ric04, HHJ+05].

Event models only capture key integration aspects such as send/receive message jitters, etc., and ignore details of the concrete computation or communication resources. They therefore represent a suitable abstraction for the realization of a light-weight distributed online system-level performance analysis, which will be discussed in the following sections.

3 Distributed Online Performance Analysis

3.1 Motivating Example

Consider the example setup shown in Figure 1. It contains four different computational units connected by a bus. Two applications are already mapped on the architecture. A video application (solid chain) gathers data from a camera controlled by the micro controller uC, performs preprocessing on the DSP and post-processing on the PPC core. The second application (dashed chain) reads data from a sensor, which is first processed on the ARM core and then forwarded to the PPC core, which, in turn, controls an actor. Both applications have constrained end-to-end latencies (Table 1(c)).

Suppose the ARM processor as well as the DSP are not fully loaded with these applications, leaving room for integrating a third application into the system. We propose the

integration of a second streaming application that runs on the ARM processor core and also uses the DSP for processing. In Figure 1, the task chain is shown under the block diagram, the white arrows indicate the desired mapping of the tasks on the system architecture. This application also has a constrained end-to-end latency (Table 1(c)).
The computational resources (PPC, uC, DSP, and ARM) are all scheduled according to the static priority preemptive policy, and the interconnecting bus is arbitrated by the CAN protocol. Core execution and core communication times as well as priorities of all tasks and exchanged messages are given in Tables 1(a) and 1(b), respectively. The periods of incoming data at the system inputs (Cam, $Sens$, and S_{in}) are specified in Table 1(d).

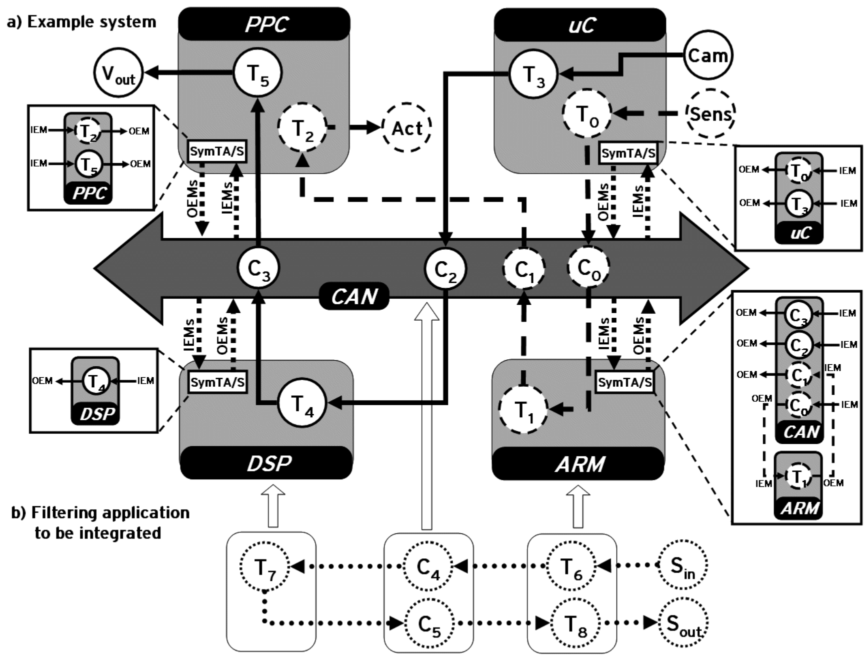

Figure 1: Example system architecture

Channel	*CCT*	*Priority*
C0	[10.4, 12.4]	4
C1	[12, 14.4]	2
C2	[20, 26.4]	1
C3	[15.2, 22.4]	3
C4	[10.4, 14.4]	5
C5	[15.2, 26.4]	6

(a) Core Communication Times (CCT)

Task	*CET*	*Priority*
T5	[30.9, 43.1]	1
T2	[15.8, 27.4]	2
T3	[36.9, 46.3]	1
T0	[20.1, 48.5]	2
T4	[13, 86]	1
T7	[40.3, 44.5]	2
T1	[27.1, 154.9]	1
T6	[14.2, 63.6]	2
T8	[11.5, 291.2]	3

(b) Core Execution Times (CET)

Path	*Deadline*
$Sens \rightarrow Act$	850
$Cam \rightarrow V_{out}$	1100
$S_{in} \rightarrow S_{out}$	1000

(c) End-to-End Constraints

System Input	*Period $\mathcal{P}$*
$Sens$	500
Cam	100
S_{in}	1000

(d) Input Event Models

Table 1: System Parameters

The key challenge is to successfully integrate the new functionality into the system without violating the timing constraints of the already mapped applications. Offline performance analysis as introduced in the previous section offers techniques to verify timing behavior prior to integration. Currently, these methodologies are only integrated in offline analysis

tools (e.g. SymTA/S [HHJ$^+$]), thus new functionality can only be safely integrated after modeling, testing, and possibly adaptation of the remaining system by an engineer.
In order to enable integration of new functionality by non-experts, we suggest to integrate performance analysis methodologies on the embedded system itself. Our approach to online distributed performance analysis suggests, that on every computational resource in the system, a performance analysis application instance is running. Each of these instances has knowledge of part of the global model corresponding to the application running on the system.
In Figure 1 these analysis instances are depicted using white boxes labeled "SymTA/S". The boxes next to them show the parts of the global model to be analyzed that are known to the respective instances. Notice that each instance has a model of the tasks running on the same processing unit as itself. As the bus has no computational power that can be used for analysis, one instance has to take over its analysis - in this case the instance on the ARM core. Similarly, online analysis could also be performed if only a few nodes contain instances of the analysis framework.
A distributed online performance analysis using the instances running on the embedded system, yields runtime performance data that can be used to implement an online acceptance test for the integration of new functionality into an embedded system. The same data can also be used to implement distributed optimization algorithms automatically adapting system parameters to changing system requirements and extended functionalities.
In the following sections we describe our approach to distributed online performance analysis and outline an acceptance test algorithm (Section 4) as well as an online optimization algorithm (Section 5). These algorithm will be used in Section 6 to successfully integrate the filtering application presented in Figure 1(b) into the system described in Figure 1(a).

3.2 Distributed system modeling

For distributed online analysis, distributed models of the application running on the embedded system as well as its architecture are needed for the analysis instances running on the computational resources. Basically, it is desirable that the analysis instances themselves are capable of generating these models using data obtainable from the system like measured execution times and observed communication paths or task dependencies. This way, the designer effort for changing system specifications or integrating new functionalities into a running system would be minimized.
However, an automatic generation of these models from measured data in an embedded system is a complex problem and not in the scope of this paper. Therefore, we assume in the remainder of this paper, that there exists a complete distributed model of the architecture of the embedded system and the application running on it. Additionally, we assume that new functionalities which are integrated into the system come along with an appropriate model.

3.3 Distributed analysis

The basic compositional performance verification model, as used in the SymTA/S tool suite [Ric04, HHJ$^+$05, HHJ$^+$], needs to be extended to support distributed descriptions of a real-time system. We introduce the concept of *Remote Resources* to model remote parts of the model that are connected to the local instance. Remote Resources are containers that represent remote parts of the model specified in other instances of the analysis framework. Since two partial models may be connected by multiple event streams, each Remote Resource may contain multiple *External Sinks* and *External Sources*. An External Sink-Source pair acts as a tunnel for analysis information (i.e. event models) being passed along the connected event streams. Note that the actual exchange of event model data is achieved over communication resources of the underlying architecture. In the case of the example system in Figure 1 the CAN-bus is used to exchange the necessary input event models (IEMs) and output event models (OEMs).

3.4 Analysis strategy

As described in Section 2, our proposed analysis methodology alternates local scheduling analysis on resources and event model propagation along event streams. The order in which resources are analyzed can be chosen arbitrarily. During offline analysis the order is determined based on topological information to minimize computational effort.

For a distributed system, we suggest to analyze each resource as soon as new input event model data is available for at least one task mapped on the resource. This approach does not require any topological knowledge or central coordination of the analysis. In the remainder of the paper we refer to this approach as "live analysis".

A problem that arises from this "live" distributed setup is to determine analysis convergence. As new functionality can be added at random points of time, successive analysis iterations may be caused by an ongoing analysis or the integration of new functionality at a remote part of the system. Thus, it is hard to determine when analysis has converged, and thus supplies valid performance data for the current system setup.

However, in existing systems an analysis run converges significantly faster compared to the average time interval between system changes. For instance, the software of a car may be updated every few months, whereas an analysis run converges in seconds. Consequently, it is safe to assume that analysis has converged if no resource in the local instance had to be analyzed within a reasonable timeout interval.

4 Acceptance Test for new Functionality

Acceptance tests are a vital tool for real-time system integration. A new function can be accepted to be integrated into a real-time system if its integration does not overload the system, does not cause constraint violations for applications already mapped on the system, and meets its own constraints.

Implementing an acceptance test on embedded devices themselves adds self-protection properties to these systems, as they can deny modifications compromising their current functionality.

For the purpose of implementing an acceptance test using the performance analysis methodologies described above, we need to be able to specify end-to-end latency constraints for distributed applications that affect multiple analysis instances. Similarly, it must be possible to determine global end-to-end latencies for these distributed applications.

In this section, we introduce techniques to distribute constraints throughout the system and collect global information about, for instance latencies or aggregated buffer sizes along paths. For simplicity, we outline the algorithms for constrained path latencies. Note that the same algorithms are also applicable for other performance data.

For distributing constraints inside the system, we assume that there exists a broadcast mechanism to send messages to all analysis instances currently present in the system.

Constraints are specified as three-tuples $\{\mathcal{P}, \mathcal{T}_\mathcal{P}, \mathcal{C}_\mathcal{P}\}$, where $\mathcal{P}$ specifies the unique name of the path, $\mathcal{T}_\mathcal{P} = t_0, \dots, t_n$ is the ordered set of tasks contained in $\mathcal{P}$, and $\mathcal{C}_\mathcal{P}$ specifies the constraint for $\mathcal{P}$. These tuples are broadcasted to all analysis instances, which store paths and constraints relevant to them. Once, $\mathcal{T}_\mathcal{P}$ is known in the system, constraints may be updated using the tuple $\{\mathcal{P}, \mathcal{C}_\mathcal{P}\}$ to reduce communication overhead.

To be able to detect violations of path latency constraints, each analysis instance must be able to compute the total latency along a given path. We propose the following algorithm to compute this latency.

Whenever an analysis instance needs to compute the latency along a specific path, it sends a request for path latency calculation along $\mathcal{P}$ towards $t_0 \in \mathcal{T}_\mathcal{P}$. From this source, a path latency discoverer is sent along the path towards its sink. This discoverer accumulates latencies along the path. Once, the discoverer reaches the end of the path, the latency calculation ends and the total path latency is stored in the discoverer. This information is then sent back along the path towards the source. This way, all analysis instances alongside the path get updated latency information. Algorithm 1 shows the pseudo-code for the algorithm applied to non-end tasks in the path. To avoid duplicate latency calculations, t_0 drops any calculation request between starting a discoverer and receiving the corresponding result.

Algorithm 1 Calculate Path Latency

```
INPUT: Name of path $\mathcal{P}$
  Task $t_i \in \mathcal{T}_\mathcal{P}, i > 0 \wedge i < n$ for current latency accumulation.
  State $s \in \{REQUEST, DISCOVER, BC\_RESULT\}$
  Current latency $l$
  if $s == REQUEST$ then
    forward request to $t_{i-1}$
  else if $s == DISCOVER$ then
    $l \leftarrow l +$ latency imposed by $t_i$
    forward discoverer to $t_{i+1}$
  else
    store $l$ as current latency of $\mathcal{P}$
    forward result to $t_{i-1}$
```

Using the mechanisms described above, an acceptance test can be implemented. If new functionality is to be integrated into a system, its timing properties, i.e. core execution times, sizes of exchanged messages, etc., and constraints must a priori be specified in a format convertible to a SymTA/S compliant model. This description is integrated into the existing distributed model of the running system. After convergence of the online analysis, the new functionality can be accepted if (a) no resource is overloaded and (b) no path constraints are violated.

We propose that similar to the concept of "live analysis", an acceptance test is executed every time convergence of analysis has been detected. Each analysis instance runs the acceptance test by checking all contained resources for overload situations and querying the relevant paths for their metrics and comparing them to given constraints. In case a resource is overloaded or a constraint is violated, a NOK (not OK)-message is broadcasted throughout the system to indicate that the current configuration cannot be accepted.

These messages can be used to trigger self-protecting or self-adapting algorithms to resolve the detected problems. These algorithms can range from denying a proposed update over graceful degradation to self-optimization as described below.

5 Heuristic Online Optimization

In this section we present a heuristic online optimization algorithm which can be used in conjunction with the acceptance test introduced in Section 4 to increase the acceptance rate for the integration of new functionality into a given organic system. In Section 6 this algorithm is used to successfully integrate new functionality into the example system given in Figure 1.

Note that the presented algorithm is tailored for optimizing the priority assignment of tasks or messages mapped on priority scheduled resources, i.e. static priority preemptive scheduled CPUs and CAN arbitrated buses. However, we are currently working on extending it to various additional scheduling strategies, including time-slice based schedulers (e.g. TDMA, Round Robin), and automotive RTOSes (e.g. $ERCOS^{EK}$).

Algorithm 2 optimizes the priorities of tasks running on a single priority scheduled resource. In order to realize a system-level priority optimization, each resource needs to run its own instance of the optimization algorithm.

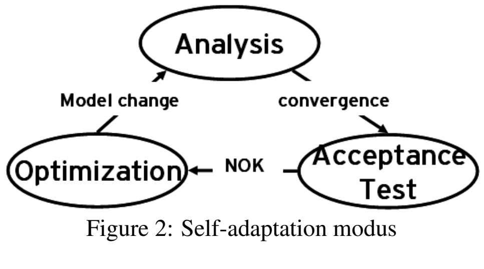

Figure 2: Self-adaptation modus

During run-time of the organic system, the system can be set into the *self-adaptation modus* (Figure 5) by broadcasting a message through the system.

In self-adaptation modus the SymTA/S instances running on the resources in the system react to NOK-messages of the acceptance test (Section 4) by triggering an online optimization step. Note that the necessary input data for Algorithm 2, i.e. path latencies etc., can be collected during the execution of the acceptance test.

During an online optimization step model parameters (i.e. task and message priorities) are modified, which, in turn, leads to the triggering of the live analysis. Once the live analysis has converged, a new acceptance test is performed. For the case that the system still violates constraints, it broadcasts a NOK-message, which again triggers an online optimization step. This analysis-optimization loop is repeated until a working system configuration is found, i.e. no NOK message occurs for a sufficiently long amount of time (time out), or until a certain number of unsuccessful optimization steps have been performed.

Algorithm 2 k-th Online Resource Optimization Step

```
INPUT: CPU $\mathcal{C}$, tasks $\mathsf{t}_1 \dots \mathsf{t}_n$ mapped on $\mathcal{C}$ with response times $\mathsf{r}_1 \dots \mathsf{r}_n$, constrained paths $\mathcal{P}_1 \dots \mathcal{P}_m$ with deadlines
    $\mathcal{D}_1 \dots \mathcal{D}_m$ and end-to-end response times $\mathcal{R}_1 \dots \mathcal{R}_m$,
    histories $\mathcal{H}_1 = \{\mathcal{H}_1^1, \dots, \mathcal{H}_1^{k-1}\}, \dots, \mathcal{H}_n = \{\mathcal{H}_n^1, \dots, \mathcal{H}_n^{k-1}\}$,
    where $\mathcal{H}_i^j = \{[\mathsf{pls}^j_{i,x_i^1}, \mathsf{rpl}^j_{x_i^1}], \dots, [\mathsf{pls}^j_{i,x_i^{s_i}}, \mathsf{rpl}^j_{x_i^{s_i}}]\}$
    history depth $\mathsf{h}_{depth}$
OUTPUT: Priority assignment for $\mathsf{t}_1 \dots \mathsf{t}_n$
 1: for ($i = 1$ to $m$) do
 2:   $\mathsf{rpl}_i^k = \frac{\mathcal{R}_i - \mathcal{D}_i}{\mathcal{D}_i}$
 3:   for ($j = 1$ to $n$) do
 4:     if ($\mathsf{t}_j \in \mathcal{P}_i$) then
 5:       $\mathsf{pls}_{j,i}^k = \frac{\mathsf{r}_j}{\mathcal{R}_i}$
 6:       add $[\mathsf{pls}_{j,i}^k, \mathsf{rpl}_i^k]$ to $\mathcal{H}_j^k$
 7: for ($i = 1$ to $n$) do
 8:   $\mathsf{relevance}_i = 1$
 9:   for ($j = k$ downto $k - \mathsf{h}_{depth} + 1$) do
10:     for ($q = 1$ to $s_i$) do
11:       $\mathsf{relevance}_i = \mathsf{relevance}_i \times (1 + \mathsf{pls}^j_{i,x_i^q}) \times (1 + \mathsf{rpl}^j_{x_i^q})$
12: assign highest priority to process with highest relevance, etc.
```

Algorithm 2 heuristically determines the ”importance” of each task on the considered resource and distributes the priorities accordingly. Important values included into the importance evaluation are the *process latency share (pls)*, denoting the share of the process response time on the overall path latency, and *relative path lateness (rpl)*, denoting the percentual lateness of a path, i.e. the percentage by which a path misses its deadline (negative lateness means the response time of the path is inferior to its deadline). Additionally, Algorithm 2 uses a history to ”average” the task importance over several iterations of the optimization.

At first glance Algorithm 2 seems like a heuristic divide-and-conquer algorithm determining local optima and combining these with the hope to achieve a globally good solution. However, this is not fully true. Algorithm 2 heuristically evaluates the local impact of tasks and resources on global timing properties. Therefore, it rather represents a heuristic global optimization algorithm, which can easily be distributed to multiple resources.

6 Case study

In this section we take the example system shown in Figure 1 as case study to demonstrate the algorithms proposed in the previous sections.

Path	*wc lat.*	*OK*
$Sens \rightarrow Act$	707.4	$\surd$
$Cam \rightarrow V_{out}$	392.8	$\surd$
$S_{in} \rightarrow S_{out}$	n.a.	n.a.

(a) latencies before integration

Path	*wc lat.*	*OK*
$Sens \rightarrow Act$	861.9	X
$Cam \rightarrow V_{out}$	484.7	$\surd$
$S_{in} \rightarrow S_{out}$	2639.9	X

(b) latencies before optimization

Path	*wc lat.*	*OK*
$Sens \rightarrow Act$	795.8	$\surd$
$Cam \rightarrow V_{out}$	1067.7	$\surd$
$S_{in} \rightarrow S_{out}$	509.1	$\surd$

(c) latencies after optimization

Table 2: Latencies before and after integration and optimization

Suppose the filtering application consisting of the path $S_{in} \rightarrow S_{out}$ shown at the bottom of Figure 1 is to be integrated into the system. In order to not compromise any timing properties of the applications already running on the system, the tasks of the application are assigned the lowest priorites on the respective resources. This is already reflected in Table 1.

Integration of the additional functionality into the model triggers online analysis and an acceptance test as previously described. Online analysis detects that timing constraints of the new application as well as a constraint of the old application are violated (see Table 2), hence the acceptance test fails and broadcasts a NOK-message through the system.

If the system is in self-adaptation mode, this triggers an online optimization step, which may find a different configuration of the system for which no path constraints are violated. In this case, optimization yielded the following priority assigment leading to a working system after 3 optmization steps. CAN: $C_5 > C_4 > C_0 > C_1 > C_2 > C_3$, ARM: $T_8 > T_6 > T_1$, DSP: $T_7 > T_4$, PPC: $T_2 > T_5$, μC: $T_0 > T_3$.

7 Conclusion

In this paper we presented a new methodology introducing self-awareness and self-adaptation with respect to real-time constraints into complex distributed embedded systems. The ambition of our approach is to increase system robustness, flexibility, and extensibility without explicit engineering effort. We motivated and demonstrated our approach by means of a small but realistic case study.

References

[BNB06] E. Bini, M. Di Natale, and G. Buttazzo. Sensitivity Analysis for Fixed-Priority Real-Time Systems. In *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS)*, Dresden, Germany, July 2006.

[CKT03] S. Chakraborty, S. Künzli, and L. Thiele. A General Framework for Analysing System Properties in Platform-Based Embedded System Designs. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Munich, Germany, 2003.

[FGB01] S. Funk, J. Goossens, and S. Baruah. On-line scheduling on uniform multiprocessors. In *Proc. of the 22nd IEEE Real-Time Systems Symposium (RTSS)*, London, UK, 2001.

[HHJ$^+$] A. Hamann, R. Henia, M. Jersak, R. Racu, K. Richter, and R. Ernst. SymTA/S - Symbolic Timing Analysis for Systems. *http://www.symta.org/.*

[HHJ$^+$05] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System Level Performance Analysis - the SymTA/S Approach. *IEE Proceedings Computers and Digital Techniques*, 152(2):148–166, March 2005.

[HJRE06] A. Hamann, M. Jersak, K. Richter, and R. Ernst. A Framework for Modular Analysis and Exploration of Heterogeneous Embedded Systems. *Real-Time Systems Journal*, 33(1-3):101–137, July 2006.

[RHE06] R. Racu, A. Hamann, and R. Ernst. A Formal Approach to Multi-Dimensional Sensitivity Analysis of Embedded Real-Time Systems. In *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS)*, Dresden, Germany, July 2006.

[Ric04] K. Richter. *Compositional Performance Analysis*. PhD thesis, Technical University of Braunschweig, 2004.