Influence of Different System Abstractions on the Performance Analysis of Distributed Real-Time Systems

Simon Perathoner [1], Ernesto Wandeler [1], Lothar Thiele [1]
Arne Hamann [2], Simon Schliecker [2], Rafik Henia [2], Razvan Racu [2], Rolf Ernst [2]
Michael González Harbour [3]
[1] Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland
[2] Institute of Computer and Communication Network Engineering, TU Braunschweig, Germany
[3] Grupo de Computadores y Tiempo Real, Universidad de Cantabria, Santander, Spain

ABSTRACT

System level performance analysis plays a fundamental role in the design process of real-time embedded systems. Several different approaches have been presented so far to address the problem of accurate performance analysis of distributed embedded systems in early design stages. The existing formal analysis methods are based on essentially different concepts of abstraction. However, the influence of these different models on the accuracy of the system analysis is widely unknown, as a direct comparison of performance analysis methods has not been considered so far. We define a set of benchmarks aimed at the evaluation of performance analysis techniques for distributed systems. We apply different analysis methods to the benchmarks and compare the results obtained in terms of accuracy and analysis times, highlighting the specific effects of the various abstractions. We also point out several pitfalls for the analysis accuracy of single approaches and investigate the reasons for pessimistic performance predictions.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Real-time and Embedded Systems*; C.4 [**Computer Systems Organization**]: Performance of Systems—*Modeling techniques*

General Terms

Performance, Design, Verification

Keywords

Performance Analysis, System Abstraction, Benchmarking

1. INTRODUCTION

One of the major challenges in the design process of real-time embedded systems is to accurately predict performance characteristics of the final system implementation in early design stages. Reliable predictions on end-to-end delays of events, memory demands and resource usages are essential to guarantee that the designed system meets all given performance requirements before time and resources are invested for the actual implementation of the system. In addition, an accurate and fast performance analysis is necessary to drive the design space exploration and thus support important design decisions.

The system level performance analysis of modern embedded systems is generally a difficult task as the architectures are increasingly heterogeneous, parallel and distributed. Complex input event streams, resource sharing, interferences among tasks and independent scheduling decisions of the distributed computing and communication nodes make the analysis process challenging even for apparently simple systems.

The need for reliable and accurate performance predictions in early design stages as well as the mentioned challenges of the analysis have driven research for many years. Most of the approaches for performance analysis proposed so far can broadly be divided into the two main classes of analytic techniques and simulation based methods. There are also stochastic methods for performance analysis which we will however not consider in this context. The main advantage of simulation is the large modelling scope, as various dynamic and complex interactions can be taken into account. However, most simulation based performance estimation methods suffer from insufficient corner case coverage. To determine guaranteed performance limits, analytic methods must be adopted. These methods provide hard performance bounds, but they are typically not able to model complex interactions and state-dependent behavior, which can lead to pessimistic (but still correct) analysis results.

Several models and methods for formal performance analysis have been presented so far; they are based on essentially different abstraction concepts. The first idea was to extend well-known results of the classical scheduling theory to distributed systems. This implies the consideration of the delays caused by the use of shared communication resources, which cannot be neglected. This combined analysis of processor and bus scheduling is often referred to as holistic scheduling analysis. The first approach in this direction was made by Tindell and Clark in [20] where the authors combine fixed priority (FP) preemptive scheduling on the processing resources of a distributed system with time division multiple access (TDMA) scheduling on the interconnecting communication bus. In [22] Yen and Wolf presented an analysis approach for distributed systems that considers data dependencies and in [14] Pop *et al.* took into account also control dependencies. Later several holistic analysis techniques for various other combinations of input event models, resource sharing policies and communication arbitration have been investigated, see e.g. [15].

A more general approach to extend the concepts of classical scheduling theory to heterogeneous distributed systems was presented by Richter et al. in [17]. In contrast to holistic methods that extend classical scheduling analysis to special classes of distributed systems, this approach applies existing analysis techniques in a modular manner: the single modules of a distributed system are analyzed with classical algorithms and the local results are propagated among the system through appropriate interfaces relying on a limited set of event stream models. Several extensions of this performance analysis framework have been worked out [6].

A completely different modular performance analysis approach that does not rely on classical scheduling theory was presented by Thiele et al. in [19]. The method uses Real-Time Calculus which extends the basic concepts of Network Calculus [7] to analyze the flow of event streams through a network of computation and communication resources. This method is not restricted to a few classes of input event models, but can model any event stream using the abstraction of arrival curves. A similar abstraction, the so-called service curves, permit to model any availability of computation or communication resources. Also for this approach various extensions have been presented [21].

While the previously mentioned formal performance analysis methods provide hard upper bounds for the worst-case performance and hard lower bounds for the best-case performance of a system, there are also approaches that determine the exact worst-case and best-case results. For instance timed automata models can be used for exact scheduling analysis [2]. Hendriks and Verhoef have presented an approach for the analysis of distributed systems based on model checking of timed automata networks in [5].

The set of available abstractions for performance analysis of distributed embedded systems is not limited to the methods cited above. The various approaches are very heterogenous in terms of modelling scope, modelling effort, tool support, accuracy and scalability. There is a lack of literature on their classification and comparison. In particular, it is very difficult for a designer to determine which performance analysis method is most suitable for a given system.

A direct comparison of performance analysis methods is difficult because several important aspects of the abstractions can only hardly be quantified, such as scalability or progression of the end-user's learning curve. Moreover the modelling scopes of the various approaches do only partially intersect. This means that an abstraction often allows to model scenarios that are not covered by other abstractions. Nevertheless, the comparison of different abstractions for performance analysis is necessary because it permits to highlight the specific effects of the various analysis methods and it helps to determine modelling difficulties and analysis pitfalls. Moreover, the comparison of different approaches serves to better understand the relation between models and analysis accuracy as well as to improve analysis methods by combining ideas and abstractions. Such a comparison based on a set of characteristic benchmark problems is not available so far, despite of its essential importance for any future work on performance analysis methods for distributed embedded systems.

The contributions of this work can be described as follows:

- We define a set of benchmarks for the evaluation of abstractions for performance analysis of distributed embedded systems.
- We apply different abstractions to the benchmarks and compare the results obtained in terms of accuracy and analysis times.
- We point out several pitfalls for the different abstractions and investigate the reasons for deviating analysis results.

2. COMPARISON METHODOLOGY

Comparing different abstractions for performance analysis of distributed embedded systems is not trivial, as the various approaches are incompatible for various reasons. First of all there are differences with respect to the analyzable performance metrics. For instance most abstractions focus on the analysis of timing properties like end-to-end delays of events or response times of tasks, while only some of the approaches can also handle other performance metrics like buffer sizes and resource usage. Moreover, there are substantial differences with respect to the modelling power of the various abstractions. There are many different system characteristics that can be taken into account by some approaches but not by others. Hierarchical scheduling, blocking times or multiple events per task activation are just a few examples of numerous system properties that differentiate the modelling scope of the various abstractions.

In this work we focus on systems in the intersection of the various modelling scopes, in order to highlight the specific effects of the abstractions. We also intentionally keep the benchmarks small with the purpose to isolate the influence of different system characteristics and expose specific analysis difficulties of the various abstractions. In order to produce meaningful evaluations, we do not restrict the analysis problems to a single system configuration but repeat the performance analysis for changing values of relevant parameters in the systems.

Furthermore we would like to point out that this work is not intended as competition between performance analysis methods. Apart from the fact that such a competition could hardly be fair given the large heterogeneity of the modelling capabilities, we underline that our main motivation is not the ranking of analysis approaches but rather the detection and investigation of analysis difficulties.

3. BENCHMARK PROBLEMS

In this section we present a set of benchmark problems that we will use for the evaluation of different abstractions for performance analysis. Some of the described benchmark problems were discussed, among others, at the ARTIST2 Workshop on Distributed Embedded Systems 2005[1].

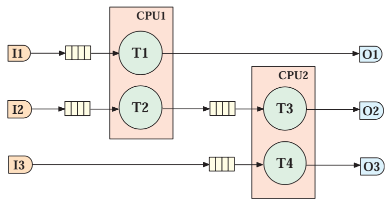
[1]`http://www.tik.ee.ethz.ch/~leiden05/`

Every problem is tailored to a particular analysis issue and consists of a simple system architecture (involving only few event streams, tasks and resources) and a performance characteristic to determine. The set of proposed benchmarks is by far not exhaustive, since there are lots of system configurations that lead to challenging analysis problems. However, our work defines several orthogonal analysis issues aimed at the evaluation of abstractions for performance analysis of distributed systems.

For the sake of simplicity we propose benchmarks with constant task execution times. We would like to point out that this choice is made in order to permit an easier interpretation of the analysis results, but is not strictly necessary, since all the analysis abstractions described in Section 4 can handle variable task execution times, typically specified as intervals [BCET, WCET]. Furthermore, in the described benchmarks the input streams are fully asynchronous and the buffering of events does not affect the performance of the system, i.e. we consider unbounded and infinitely fast buffers.

3.1 Benchmark 1: Complex activation pattern

The intention of this benchmark specification is to compare the behavior of different analysis abstractions with respect to a complex task activation pattern. By *complex* we mean an activation pattern that cannot be described by one of the so-called *standard event models* like periodic activation, periodic activation with jitter or periodic activation with burst (see Section 4.2.1 for more details on standard event models).



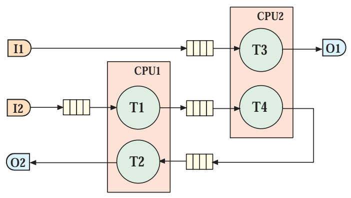
Input streams	I1: periodic (P = 60ms) I2: periodic (P = 5ms) I3: periodic (P = [60..110]ms)
Resource sharing	CPU1: FP preemptive, CPU2: FP preemptive
Execution times	T1: 35ms, T2: 2ms, T3: 4ms, T4: 12ms
Scheduling parameters	priority T1: high, priority T2: low priority T3: low, priority T4: high

Figure 1: Specification of benchmark 1

Figure 1 depicts the topology of the system. Three periodic event streams are processed by four tasks running on two CPUs that implement preemptive fixed priority scheduling. The performance characteristic to determine is the worst-case response time of task T3 as function of the period of stream I3. The activation pattern of task T3 is not periodic anymore, since task T1 can preempt task T2. In particular, the output behavior of task T2 is complex, i.e. cannot be precisely described by a standard event model. Thus, we expect pessimistic analysis results for abstractions relying on standard event models.

3.2 Benchmark 2: Variable feedback

The purpose of this benchmark specification is to confront the different analysis abstractions with a feedback loop and



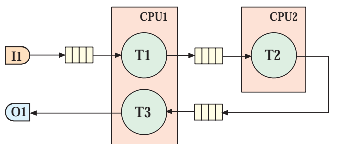
Input streams	I1: periodic (P = 100ms) I2: periodic (P = 5ms)
Resource sharing	CPU1: FP preemptive, CPU2: FP preemptive
Execution times	T1: 2ms, T2: 2ms, T3: [2..22]ms, T4: 1ms
Scheduling parameters	priority T1: high, priority T2: low priority T3: high, priority T4: low

Figure 2: Specification of benchmark 2

the consequent correlations among the activation times of the involved tasks. The system architecture is shown in Figure 2. A periodic event stream I2 is processed serially by three tasks running on two CPUs and forming a feedback loop. Besides, CPU2 processes also a second periodic event stream I1 of higher priority. The performance metric to determine is the worst-case delay from I2 to O2. In order to vary the correlation among the task activation times in the feedback loop, the execution time of task T3 is gradually increased. We expect pessimistic analysis results for abstractions that do not take into consideration such correlations among task activations.

3.3 Benchmark 3: Cyclic dependencies

The intention of this specification is to examine the capability of different performance analysis methods to deal with cyclic dependencies.



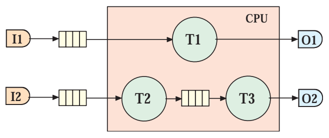
Input stream I1	periodic with burst (P = 10ms, J = [0..50]ms)
Resource sharing	CPU1: FP preemptive
Execution times	T1: 1ms, T2: 4ms, T3: 4ms
Scheduling parameters	1) priority T1: high, priority T3: low 2) priority T1: low, priority T3: high

Figure 3: Specification of benchmark 3

Figure 3 represents the system to analyze. A periodic event stream with bursts is processed by a sequence of three tasks running on two resources. On CPU1 a preemptive fixed priority scheduler is used to schedule the tasks T1 and T3. The performance characteristic to determine is the worst-case delay from I1 to O1 for increasing values of the input jitter and in two different scenarios. In scenario 1 T1 has higher priority than T3. In scenario 2 T3 has higher priority than T1, which means that there is a cyclic dependency among the two tasks (T1 indirectly triggers T3, however T3 preempts T1). We expect this cyclic dependency to make the analysis difficult for compositional system abstractions.

3.4 Benchmark 4: Data dependencies

The purpose of this benchmark specification is to clarify the ability of different analysis approaches to handle systems



Input streams	I1: periodic (P = 80ms) I2: periodic (P = 50ms)
Resource sharing	CPU: FP preemptive
Execution times	T1: [15..30]ms, T2: 20ms, T3: 10ms
Scheduling param.	priority T1: high, priority T2: medium, priority T3: low

Figure 4: Specification of benchmark 4

with data dependencies among tasks. The system specified below was first presented as example in [22] by Yen and Wolf. Figure 4 depicts the topology of the system. Two periodic event streams are processed by three tasks on a CPU that implements preemptive fixed priority scheduling. The data dependency is given by the execution sequence T2-T3. The performance characteristic to determine is the worst-case delay from I2 to O2 as function of the execution time of T1. For this benchmark we expect pessimistic analysis results for abstractions that do not take into consideration data dependencies among tasks.

4. ABSTRACTIONS FOR PERFORMANCE ANALYSIS

In this section we briefly describe four different abstractions for formal performance analysis of distributed embedded systems in early design stages.

4.1 Holistic scheduling - The MAST approach

Rather than a specific performance analysis method, holistic scheduling is a collection of techniques for the analysis of distributed embedded systems. The common principle is to extend concepts of the classical scheduling theory to distributed systems, integrating the analysis of processor and communication infrastructure scheduling. Every technique is tailored towards a particular combination of input event model, resource sharing policy and communication arbitration. The resulting large and heterogeneous collection of analysis methods makes it rather difficult to use holistic scheduling analysis in practice. This problem was relieved with the release of the Modelling and Analysis Suite for Real-Time Applications (MAST) [4] that aggregates several holistic analysis techniques. MAST is an open model for the description of event-driven real-time systems, together with a set of open source tools that enables the designer of a real-time application to verify its timing behavior.

The MAST suite includes schedulability analysis tools for the analysis of single processor and distributed systems with fixed priority, earliest deadline first (EDF) or EDF within priorities scheduling. The toolset includes offset-based schedulability analysis techniques [12, 11], optimized priority assignment, automatic calculation of blocking delays caused by mutual exclusion synchronization and sensitivity analysis. All the techniques include analysis capabilities for arbitrary deadlines, handling of input and output jitter and different variations of the scheduling policies, such as preemptive and non-preemptive, polling servers, sporadic servers, etc.

In MAST a real-time system is modelled as a set of transactions. Each transaction is represented through a graph that models the event flow among the activities executed by the system. The external events triggering the transactions can be of different kinds: periodic, unbounded aperiodic, sporadic, bursty, or singular (arriving only once). The execution of an activity generates an event that may in turn trigger other activities. Internal events may have timing requirements associated with them.

Figure 5 shows one of the key aspects of the MAST model, which is the separate modelling of the execution platform, the software modules, and the transactions that define a particular configuration or mode of the real-time system. This separation simplifies the use of the model in a component-based design process and separates the description of overhead parameters such as processor-, network- and driver overheads from the actual application model.

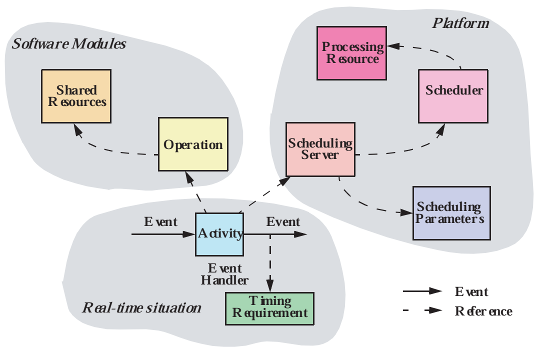


Figure 5: Elements of the MAST model

Using a UML tool it is possible to describe a real-time view of a system by adding appropriate real-time classes and objects [9]. The application design is linked with the real-time view to get a full description of the system including its timing requirements. The corresponding MAST model can then be automatically extracted from the UML description.

The MAST suite is available as open source software.[2]

4.2 Compositional Analysis Approaches

The basic idea of compositional analysis approaches is visualized in Figure 6. Compositional system level analysis alternates local component analysis and output event model propagation. More precisely, in each global iteration of the compositional system level analysis, local analysis is performed for each component to derive the output event models. Afterwards, the calculated output event models are propagated to the connected components, where they are used as input event models for the subsequent global iteration. Obviously, this iterative analysis represents a fix-point problem [17]. For the case that after an iteration all calculated output event models stay unmodified, convergence is reached and the last calculated task response times are valid. This holds for analysis techniques that do not contain a state in the analytical model, otherwise not only unchanged output event models, but also steady internal state has to be considered as convergence criterion. In the case where no convergence is reached, no statement can be made about the analyzed system.

[2] `http://mast.unican.es`

In the following sections we will shortly introduce two compositional analysis methodologies, namely SymTA/S (Section 4.2.1) and MPA-RTC (Section 4.2.2). Both methods differ in the way local component analysis is performed, as well as how event models are represented and propagated between dependent components.

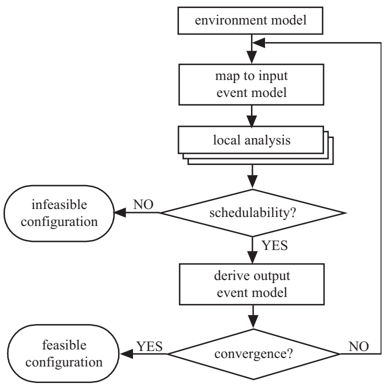


Figure 6: Compositional system level analysis

4.2.1 The SymTA/S approach

The system level performance analysis approach SymTA/S[3] (Symbolic Timing Analysis for Systems) is based on the principles of compositional system level analysis.

At the component level SymTA/S uses formal analysis techniques based on the busy window technique proposed by Lehoczky [8]. SymTA/S can therefore directly reuse scheduling analysis techniques from real-time research, and does not need to adapt these techniques to a specific model. Currently, SymTA/S offers local analysis techniques for FP scheduling (preemptive and non-preemptive), TDMA, Round Robin, EDF, CAN, and ERCOSek.

SymTA/S uses so-called *standard event models* as interface to couple local component analyses according to the compositional system level analysis methodology. Standard event models are described by three parameters P, J and D. A *periodic* event model has one parameter P and states that each event arrives exactly every P time units. This simple model can be extended with the notion of a jitter J, leading to the event model *periodic with jitter*. In this model events occur periodically on the long term, but their actual arrival instant can jitter around the ideal periodic arrival within an interval of size 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 such a *bursty* event stream, the model *periodic with jitter* is extended with a parameter D that captures the minimum distance between the arrival times of events within a burst. Standard event models capture only key timing aspects of event streams but ignore more detailed stream properties. They therefore represent an simple interface for the coupling of heterogeneous performance analysis techniques.

In order to enable the performance analysis of distributed systems with feedback between two or more components, SymTA/S uses a so-called *starting point generation* to determine initial input event models for all components. Basically the external event models are propagated along all task chains until an activating event model is available for each task. Since the scheduling of tasks on a shared resource does not change the periods of the involved event streams and cannot decrease their maximum jitter, the starting point generation is safe. More details about starting point generation and output event model calculation can be found in [16].

[3]`http://www.symta.org`

The end-to-end latency along task chains is calculated by a simple sum over the local worst case response times. While this allows an easy composition of the local analysis results, it leads to an overestimation of end-to-end delays in the presence of bursty event streams, because the worst-case interference is assumed for each task along the chain and this is generally not realistic. In recent work this has been ameliorated by considering the pipelined execution of multiple events. The results of this improvement are included in the experiments of Section 5 and labelled as *SymTA/S path*, while the performance analysis based on the simple sum of local worst case response times is denoted by *SymTA/S add*.

4.2.2 Modular Performance Analysis with Real-Time Calculus

Modular Performance Analysis with Real-Time Calculus (in short called MPA-RTC) [18] is a framework for performance analysis of distributed embedded systems that has its roots in network calculus [7]. MPA-RTC analyzes the flow of event streams through a network of computation and communication resources in order to derive performance characteristics of the system.

Event stream model.

Event streams are described using a pair of arrival curves $\bar{\alpha}^u(\Delta)$, $\bar{\alpha}^l(\Delta) \in \mathbb{R}^{\geq 0}$, $\Delta \in \mathbb{R}^{\geq 0}$ which provide an upper and a lower bound on the number of events in *any* time interval of length Δ. Figure 7(a) shows an example pair of arrival curves. If $R[s,t)$ denotes the number of events that arrive in the time interval $[s,t)$, then the following inequality is satisfied:

$$\bar{\alpha}^l(t-s) \leq R[s,t) \leq \bar{\alpha}^u(t-s) \quad \forall s < t$$

where $\bar{\alpha}^l(0) = \bar{\alpha}^u(0) = 0$. This abstraction is much more general than standard event models: any event stream can be modelled by an appropriate pair of arrival curves.

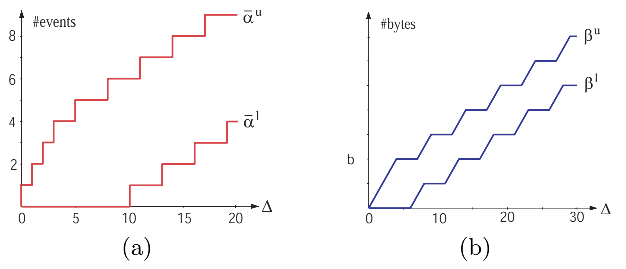


Figure 7: Examples of arrival and service curves

Resource model.

In a similar way, resource streams are described using a pair of service curves $\beta^u(\Delta)$, $\beta^l(\Delta) \in \mathbb{R}^{\geq 0}$, $\Delta \in \mathbb{R}^{\geq 0}$ which provide an upper and a lower bound on the available service in any time interval of length Δ. Figure 7(b) shows an example pair of service curves. The service is expressed in an appropriate unit, for instance number of cycles for computing resources or bytes for communication resources. If $C[s,t)$ denotes the number of processing or communication units available from the resource over the time interval $[s,t)$, then the following inequality holds:

$$\beta^l(t-s) \leq C[s,t) \leq \beta^u(t-s) \quad \forall s < t$$

Any resource availability can be modelled by an appropriate pair of service curves. The described abstraction treats the resource usage as first class citizen of the analysis approach and makes it modular also in terms of resource composition.

Performance components and Real-time Calculus.

Performance components are the basic building blocks to construct a performance model of a system. They model the processing of an event stream by an application process that is running on a shared resource, e.g. a computing or a communication subsystem. In the MPA-RTC framework an incoming event stream, represented as a pair of arrival curves α^l and α^u, is processed by a resource with availability β^l and β^u. On its output, the component generates an outgoing stream of processed events, represented by a pair of arrival curves $\alpha^{l'}$ and $\alpha^{u'}$. Resources left over by the component are made available again on the resource output and are represented by a pair of service curves $\beta^{l'}$ and $\beta^{u'}$. Outgoing arrival and service curves are determined from incoming arrival and service curves according to equations defined by Real-Time Calculus [3].

Performance components are combined to form performance models of distributed embedded systems. Scheduling policies on shared resources can be modelled by the way performance components are linked and resource streams are distributed among them. Possible resource sharing mechanisms are FP, EDF, TDMA and generalized processor sharing (GPS). Global performance characteristics such as end-to-end delays are determined by propagating the local analysis results through the system. MPA-RTC is available in form of a free Matlab toolbox.[4]

4.3 Timed automata based analysis

Timed automata [1] are a popular formalism for the specification and analysis of real-time systems. Timed automata can be applied for the schedulability analysis of event driven systems [10]. In this work we focus on a performance analysis approach for distributed embedded systems presented by Hendriks and Verhoef in [5], which relies on formal verification of timed automata networks by means of reachability analysis using the Uppaal model checker[5].

Modelling the environment

Several timed automata models have been presented for different event stream types. For instance, Figure 8 shows a timed automaton that models a periodic event stream with period P and jitter $J \leq P$.

Modelling the system

Each hardware resource is modelled by a separate automaton. Several different resource sharing strategies can be modelled with appropriate timed automata. For instance, Figure 9 depicts a timed automaton that models a CPU executing two tasks and implementing preemptive fixed priority scheduling.

[4]http://www.mpa.ethz.ch/Rtctoolbox

[5]http://www.uppaal.com

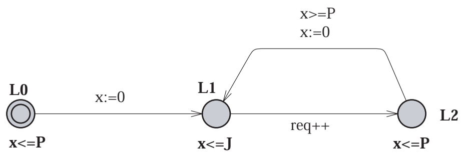


Figure 8: Periodic event stream with jitter

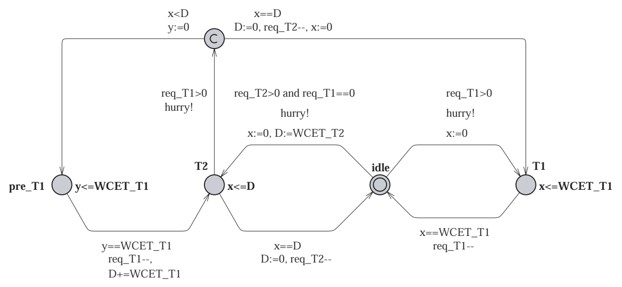


Figure 9: Preemptive FP resource with two tasks

Performance analysis

The models of the various system components are aggregated into a timed automata network. The key idea for the modelling of distributed systems is to use global variables and channels for the interaction of the different automata. The performance of the system is derived by formal verification of properties of the timed automata network. For instance, to ensure that the maximum backlog of a certain task does not exceed a given value b, it is sufficient to verify the following property with Uppaal:

$$AG \quad (req \leq b)$$

where 'AG' stands for 'always generally' (= invariantly) and *req* is the global variable that counts the activation requests of the corresponding task. It is also possible to derive the *exact* maximum backlog by finding the smallest b that satisfies the above property. This is done by using a binary search strategy.

The verification of end-to-end delays is done using particular timed automata models for event stream generators that are synchronized with the system output over a global channel and can keep track of the amount of time that elapses between the generation of an event and its output from the system. For a detailed explanation of the corresponding models we refer the reader to [5].

5. ANALYSIS RESULTS

In this section we present the results obtained by applying the formal performance analysis methods described in Section 4 to the benchmarks of Section 3. We compare the performance bounds obtained and discuss deviating results and analysis pitfalls. The models adopted are available online[6]. For their analysis we have used the RTC Toolbox v1.0, SymTA/S v1.1, Uppaal v4.0.3 and MAST v1.3.6, respectively.

[6]http://www.tik.ee.ethz.ch/~leiden05/index2.html#publications

We also include the results obtained by a simple SystemC simulation. For periodic input streams with jitter/burst the simulator generates events as early as possible (at the beginning of the jitter interval) with a probability of 5%, as late as possible (at the end of the jitter interval) with a probability of 5% and uniformly distributed over the jitter interval with a probability of 90%. This is done in order to increase the corner case coverage of the simulation with respect to a fully uniform event distribution. The time length of the simulated system execution is indicated in brackets in the graphs.

For additional benchmarking results and details about the simulation tool used we refer the reader to [13].

5.1 Benchmark 1: Complex activation pattern

In this experiment we evaluate the accuracy of the different formalisms when the event patterns significantly deviate from the patterns of standard event models that are used in SymTA/S. For this purpose, we tap the event stream between the tasks T2 and T3 in the system of Figure 1, where a distortion of the periodic event pattern occurs due to the influence of task T1. Figure 10 shows the analysis results for the worst-case response time of task T3.[7] The performance values derived with timed automata models are verified through model checking and represent the exact worst-case response time of task T3.

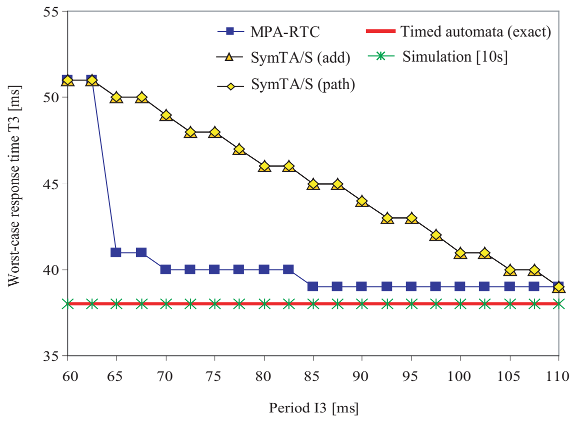


Figure 10: Analysis results for the WCRT of T3 in benchmark 1

The graph shows that the compositional analysis approaches provide pessimistic predictions for the worst-case response time of task T3 and it points out that there is a remarkable difference between the results obtained by SymTA/S and MPA-RTC. This can be explained by the different event models adopted by the two abstractions. While MPA-RTC accurately models the complex output pattern of T2 by an appropriate pair of arrival curves, SymTA/S approximates the output of T2 by a periodic event stream with burst.

Figure 11 shows the effect of the two different event models on the analysis accuracy for $P_{I3} = 65$ ms. The worst-case response time of T3 is given by the maximum horizontal distance between the worst-case resource availability (dashed curve) and the worst-case execution demand (solid curves). The graph shows that the approximation adopted by SymTA/S leads to an overestimated response time of task T3. Interestingly, however, the conservative results of SymTA/S disappear when the whole task chain from I2 to O2 is considered. This is because the adopted path analysis detects that the total worst-case delay from I2 to O2 is smaller than the sum of the two single worst-case delays. For the worst-case delay I2-O2 all considered methods determine the exact performance results.

[7]The MAST tool does not support the analysis of local response times in the current release and was thus not considered for this analysis problem.

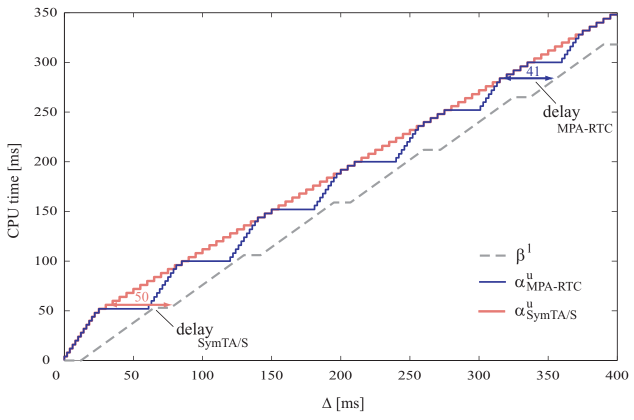


Figure 11: Influence of different event models on the analysis of the WCRT of T3 for $P_{I3} = 65$ ms

5.2 Benchmark 2: Variable feedback

In benchmark 2 depicted in Figure 2 the behavior of the feedback stream I2-O2 depends strongly on the execution time of task T3, as task T3 may preempt task T4 and thus affect its response time. In particular, increasing the worst-case execution time of T3 at a constant rate causes the correlation effects between T1 and T2 to vary in a periodic manner.

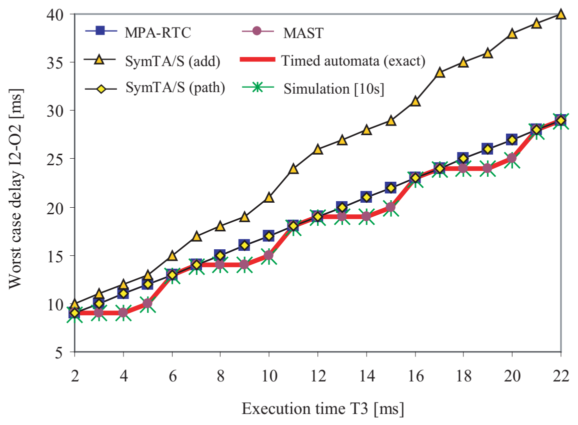


Figure 12: Analysis results for the worst case delay I2-O2 in benchmark 2

This effect is shown in Figure 12 by the exact values of the worst-case delay I2-O2 determined with the analysis approach based on timed automata. The graph shows that also the MAST tool provides the exact worst-case performance values for all the parameter configurations of the specified system. However, the compositional analysis approaches MPA-RTC and SymTA/S do recurrently overestimate the exact performance of the system and provide pessimistic predictions for several parameter values. A closer analysis of the behavior of the feedback loop reveals that this overestimation happens for those parameter configurations that lead to the worst-case delay I2-O2 without a full preemption of task T2 on CPU1. Since the compositional abstractions do not take into consideration the correlation between the activation times of T1 and T2, the corresponding analysis methods have no means to recognize the missing or partial preemption and suppose that a full preemption is possible in the worst case, which leads to pessimistic performance predictions.

Another interesting property of the system in Figure 2 is that the worst-case delay I2-O2 is smaller than the sum of the single worst-case delays at T1, T4 and T2. This phenomenon is common for systems with a sequence of tasks that process bursty event streams and is generally referred to as 'Pay burst only once' [7]. In such cases a purely modular analysis based on the sum of the local worst case response times leads to overly pessimistic performance predictions, as shown in Figure 12 by the results of the analysis mehtod *Symta/S add*.

5.3 Benchmark 3: Cyclic dependencies

In the first scenario of the specification depicted in Figure 3 task T1 has higher priority than task T3 and thus there is no cyclic dependency in the system. However, correlation effects as described for benchmark 2 are present. For instance, depending on the input stream properties, it may happen that task T3 is not preempted by task T1. Such correlations are not fully exploited by all analysis abstractions, as described above. While the timed automata model permits to determine the exact worst-case latencies of the system, other formal analysis methods like MPA-RTC and SymTA/S slightly exceed the exact performance results and their pessimism grows with increasing input jitter values, as shown in Figure 13.

The poor performance predictions of the MAST tool have another cause. Holistic analysis methods compute the worst-case delay not referred to the actual release time of an event which varies within the jitter interval, but referred to the ideal periodic release time. In other words, the release jitter is considered already part of the delay and thus the predicted worst-case end-to-end latency cannot be smaller than the maximum input jitter. This explains why the pessimism of the predictions provided by MAST increases for increasing values of the input jitter. Unfortunately this deviation with respect to the other analysis abstractions cannot be simply adjusted after the analysis, since the actual release instant leading to the worst-case performance is generally unknown. However, we would like to point out that the different interpretation of latency adopted by MAST in the presence of input jitter can be useful in other settings. For instance in a system where the activation jitter of a task is caused by a low resolution clock it is more appropriate to refer the deadline for the response time of the task to the real activation request rather than to the actual activation instant.

The graph in Figure 13 also shows that simulation can in general not be used to guarantee hard performance bounds: for some input configurations the corner-cases leading to the worst-case performance are missed by the executed simulation.

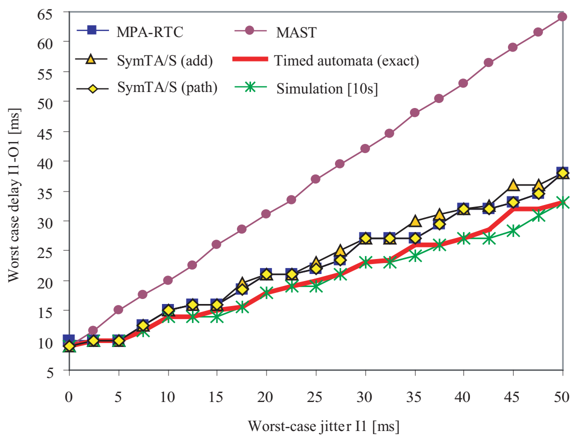


Figure 13: Analysis results for the worst case delay I1-O1 in benchmark 3 (scenario 1)

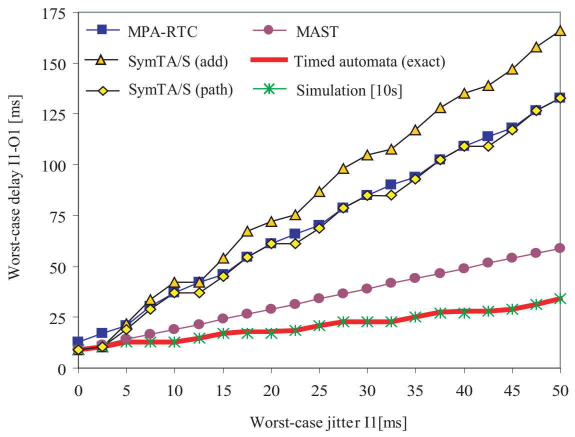


Figure 14: Analysis results for the worst case delay I1-O1 in benchmark 3 (scenario 2)

In scenario 2 T3 has higher priority than T1 and thus there is a cyclic dependency: the output behavior of T1 depends on the CPU availability left over by the activity of T3 while at the same time the activity of T3 depends on the output behavior of T1. For holistic system abstractions this dependency does not make the analysis more difficult and Figure 14 shows that the performance predictions provided by MAST do not deviate more significantly from the exact values than in the previous scenario (note the different scaling of the ordinate axes). However, for compositional abstractions the cyclic dependency complicates the analysis process. Both MPA-RTC and SymTA/S use a fixed-point calculation to handle it, but the graph shows that this leads to overly pessimistic performance predictions.

5.4 Benchmark 4: Data dependencies

Figure 15 represents the analysis results obtained applying the different abstractions to benchmark 4 specified in Figure 4. The chart shows that MPA-RTC and SymTA/S largely overestimate the worst-case delay I2-O2, while MAST determines the exact worst-case performance of the system. The overly pessimistic performance prediction of the former two approaches results from the disregard of data dependencies in the system. In particular, the activation times of the tasks T2 and T3 are not independent. The data dependency forces the two tasks to be executed in a fixed order and imposes a temporal offset between their activation. Let us consider, for instance, the system configuration with an execution time of 15 ms for T1. It is simple to verify that in this configuration T1 can only preempt either T2 or T3, but not both in a single execution and also T2 cannot preempt T3. Hence, the worst-case latency I2-O2 is 45 ms. However, MPA-RTC and SymTA/S ignore the data dependencies between T2 and T3 and consider their activation times as independent. Thus, they suppose a worst-case response time of 35 ms for T1 and 45 ms for T2 and estimate the worst-case latency I2-O2 with 80 ms, the sum of the two delays. In contrast, the MAST tool implements offset based analysis methods, that are designed to detect and exploit data dependencies among tasks in order to determine tighter performance bounds.

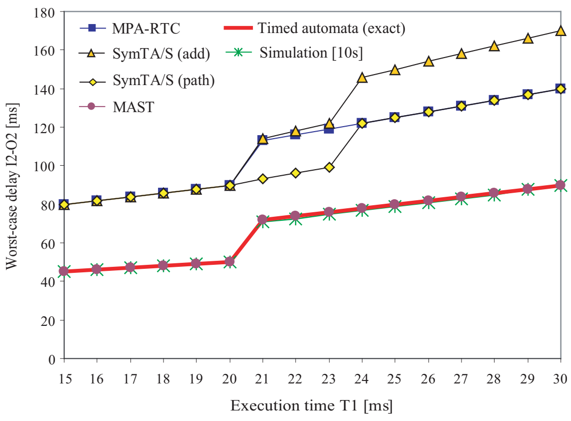


Figure 15: Analysis results for the worst case delay I2-O2 in benchmark 4

5.5 Analysis times

		B1	B2	B3(1)	B3(2)	B4
MPA-RTC	min	0.60	0.03	0.01	0.04	0.03
	med	1.06	0.04	0.01	0.15	0.05
	max	19.72	0.08	0.04	0.30	0.20
SymTA/S	min	0.05	0.03	0.03	0.03	0.06
	med	0.09	0.05	0.06	0.34	0.09
	max	1.50	0.23	0.09	0.80	0.31
MAST[a]	min	-	<0.5	<0.5	<0.5	<0.5
	med	-	<0.5	<0.5	<0.5	<0.5
	max	-	<0.5	<0.5	<0.5	<0.5
Timed aut.[a, b]	min	18.0	<0.5	<0.5	<0.5	<0.5
	med	34.5	<0.5	1.0	<0.5	<0.5
	max	60.5	<0.5	52.0	5.5	<0.5
Simulation[a]	min	1.0	<0.5	0.5	0.5	<0.5
	med	1.0	<0.5	0.5	0.5	<0.5
	max	1.0	<0.5	0.5	0.5	<0.5

[a]For MAST, Timed automata and Simulation we have timed the analysis duration by hand since the corresponding tools do not support automatic measuring of the analysis time. For these methods a '<0.5' in the table stands for a value below the measuring accuracy of 0.5 seconds.

[b]For the analysis approach based on timed automata the analysis times are referred to one single step of binary search.

Table 1: Analysis times in seconds

In this subsection we report the measured analysis times for all the considered benchmarks. Table 1 sums up the minimum, median and maximum analysis time for each performance analysis tool and benchmark. The values show that most of the considered abstractions permit a fast performance analysis for all the specified benchmarks. However, the analysis approach based on model checking of timed automata networks forms an exception, as in two out of four benchmarks it suffers from very long verification times. For instance in the first scenario of benchmark 3 the maximum running time of the Uppaal model checker is more than a hundred times larger than the analysis times of the other methods. Especially in the presence of large jitters the state space of the timed automata models grows considerably and leads to long analysis times.

Another interesting observation is that in some cases there is a remarkable difference between the minimum, median and maximum analysis time. This shows that in general the analysis times of the different approaches may depend highly on the particular system parameters.

6. DISCUSSION

The results of Section 5 show that the accuracy of the performance predictions determined with each abstraction varies considerably for the different benchmarks. The only exception is given by the analysis approach based on timed automata, which provides the exact performance predictions for all the considered benchmarks. However, we would like to point out that the exact results are often paid for by a large analysis effort, i.e. may require long verification times. Thus, considering not only the achieved accuracy, but also the necessary analysis times, we can state that none of the considered abstractions performed best in all the benchmarks.

Nevertheless, the results permit to give some indications on which abstractions are more appropriate than others for the analysis of a certain performance characteristic in a given system. For instance benchmark 1 indicates that the approximation of complex event streams with standard event models can be inappropriate for precise performance predictions at a local level. Benchmark 3 emphasizes that at the current state systems with cyclic dependencies represent a serious pitfall for the accuracy of compositional analysis methods. The benchmarks 2 and 4 indicate that holistic analysis approaches are generally more appropriate than modular abstractions in the presence of correlations among task activations and data dependencies. On the other hand, holistic analysis methods are less appropriate for the analysis of timing properties that are referred to the actual release time of an event within a large jitter interval, as described in the interpretation of the results obtained for benchmark 3. Overall, it is advisable for a system designer to use at least two different performance analysis methods, to prevent stepping in one of the mentioned analysis pitfalls.

Moreover, the results show that for the benchmarks considered simulation often provides more accurate results but these results are not necessarily correct (i.e. valid performance bounds), as the underestimation of the worst case performance of benchmark 3 shows. While in general this might be tolerable for soft real-time systems, it is not for systems with hard real-time requirements.

We would also like to emphasize that most of the encountered pitfalls for analytic techniques are not related to system characteristics that are conceptually impossible to integrate in the respective abstraction. Rather, the analysis difficulties point out aspects that have not yet been considered for the corresponding methods. In this sense poor analysis results indicate potential research directions for the improvement of the single performance analysis approaches.

Furthermore, there are also questions that the proposed set of small benchmarks cannot answer. For instance it would be useful to analyze larger systems with the aim to examine the scalability of the different abstractions with respect to analysis accuracy and analysis times. It could also be interesting to consider the combination of several system properties that have been isolated in the single benchmarks.

7. CONCLUSIONS

We defined a set of benchmarks for the evaluation and comparison of abstractions for performance analysis. We applied a number of system level performance analysis methods to the benchmarks and examined the results in terms of accuracy and analysis times. We pointed out several analysis pitfalls for the different analysis abstractions and investigated the reasons for pessimistic performance predictions. We showed that the analysis results of different approaches are remarkable different even for apparently basic distributed systems and that the choice of an appropriate analysis abstraction matters. Moreover, we showed that the analysis accuracy of the various approaches depends highly on the particular system characteristics and that none of the analysis methods performed best in all cases. Hence, the problem to provide accurate performance predictions for general systems is still far from solved, despite the availability of promising formal analysis approaches.

8. ACKNOWLEDGEMENTS

This research has been supported by the Swiss National Science Foundation (SNF) under the project *Modular Performance Analysis of Distributed Embedded Real-Time Systems* (grant 200021-103580/2), by the Plan Nacional de I+D+I of the Spanish Government under the *THREAD* project (grant TIC2005-08665-C03) and by the *Artist2 Network of Excellence* (EU grant IST-004527).

9. REFERENCES

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. UPPAAL 4.0. In *Proc. of the 3rd Intl. Conference on the Quantitative Evaluation of SysTems*, IEEE Computer Society, pages 125–126, 2006.

[3] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. of 6th Design, Automation and Test in Europe*, pages 190–195, 2003.

[4] M. González Harbour, J. J. Gutiérrez García, J. C. Palencia Gutiérrez, and J. M. Drake Moyano. Mast: Modeling and analysis suite for real time applications. In *Proc. of 13th Euromicro Conference on Real-Time Systems*, pages 125–134. IEEE Computer Society, 2001.

[5] M. Hendriks and M. Verhoef. Timed automata based analysis of embedded system architectures. In *Workshop on Parallel and Distributed Real-Time Systems*, 2006.

[6] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the SymTA/S approach. *IEE Proc. Computers and Digital Techniques*, 152(2):148–166, March 2005.

[7] J. Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet.* Springer-Verlag New York, Inc., 2001.

[8] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proc. of the Real-Time Systems Symposium*, pages 201–209, 1990.

[9] J. L. Medina, M. González Harbour, and J. M. Drake. Mast real-time view: A graphic uml tool for modeling object-oriented real-time systems. In *Proc. of the 22nd Real-Time Systems Symposium*, pages 245–256. IEEE Computer Society Press, 2001.

[10] C. Norström, A. Wall, and W. Yi. Timed automata as task models for event-driven systems. In *Proc. of the 6th Intl. Conference on Real-Time Computing Systems and Applications*, page 182. IEEE Computer Society, 1999.

[11] J. C. Palencia and M. González Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proc. of the 20th Real-Time Systems Symposium*, pages 328–339. IEEE Computer Society Press, 1999.

[12] J. C. Palencia Gutiérrez and M. González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proc. of the 19th Real-Time Systems Symposium.* IEEE Computer Society, 1998.

[13] S. Perathoner, E. Wandeler, and L. Thiele. Evaluation and comparison of performance analysis methods for distributed embedded systems. Technical Report 276, Computer Engineering and Networks Laboratory, ETH Zurich, Mar. 2006.

[14] P. Pop, P. Eles, and Z. Peng. Performance estimation for embedded systems with data and control dependencies. In *Proc. of the 8th intl. workshop on Hardware/software codesign*, pages 62–66. ACM Press, 2000.

[15] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proc. of the 10th intl. symposium on Hardware/software codesign*, pages 187–192. ACM Press, 2002.

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

[17] K. Richter, M. Jersak, and R. Ernst. A formal approach to mpsoc performance verification. *IEEE Computer*, 36(4):60–67, 2003.

[18] L. Thiele, S. Chakraborty, M. Gries, A. Maxiaguine, and J. Greutert. Embedded software in network processors - models and algorithms. In *Proc. of the 1st intl. Workshop on Embedded Software*, pages 416–434. Springer-Verlag, 2001.

[19] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. Intl. Symposium on Circuits and Systems*, volume 4, pages 101–104, 2000.

[20] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems)*, 40:117–134, 1994.

[21] E. Wandeler and L. Thiele. Characterizing workload correlations in multi processor hard real-time systems. In *Proc. of the 11th Real Time on Embedded Technology and Applications Symposium*, pages 46–55. IEEE Computer Society, 2005.

[22] T. Y. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. In *Proc. of the 1995 Intl. Conference on Computer Design*, pages 64–71. IEEE Computer Society, 1995.