Combined Approach to System Level Performance Analysis of Embedded Systems

Simon Künzli
Siemens Building Technologies Group
Switzerland
kuenzli@ieee.org

Arne Hamann Rolf Ernst
TU Braunschweig
Germany
{arne|ernst}@ida.ing.tu-bs.de

Lothar Thiele
ETH Zürich
Switzerland
thiele@tik.ee.ethz.ch

ABSTRACT

Compositional approaches to system-level performance analysis have shown great flexibility and scalability in the design of heterogeneous systems. These approaches often assume certain system architectures and application domains, and are thus tailored to give tight analysis results for specific systems. We consider two different compositional analysis methods. Both methods have their particular strengths for different architectures and applications. In this paper, we aim to enhance the analysis capabilities for these techniques. A method for event model conversion allows us a seamless integration of the two methods. Finally, we present a detailed case study to show the applicability and benefits of the enhanced performance analysis technique.

Categories and Subject Descriptors

C.0 [**Computer Systems Organizations**]: General—*Modeling of Computing Architectures*

General Terms

Performance, Experimentation

1. INTRODUCTION

The increasing number of transistors that may be integrated into a single chip leads to more and more complex embedded systems. To properly design such a system, the developers need feedback on system properties of their design already in an early design phase. This is accomplished by performance analysis techniques that can be based already on very abstract system models.

Over the last decade, several analytical methods for performance analysis of embedded systems were proposed, examples can be found in [10, 19, 9, 16, 11, 4, 15, 6]. Most of them are specialized and dedicated to a specific application domain, whereas others try to be general enough to serve for many different application scenarios. These methods all have their particular strengths, e.g. in terms of supported scheduling techniques, or models of computation, but also expose weaknesses when dealing with systems they were not designed to cope with.

CODES+ISSS'07, September 30–October 3, 2007, Salzburg, Austria.

The heterogeneity of nowadays embedded systems challenges state-of-the-art performance analysis methods with respect to analysis capabilities and accuracy. Modular design approaches have been proposed that allow to compose analysis of different resource sharing strategies. They share a similar overall analysis strategy but use different computational models. They have demonstrated high modeling accuracy and simplicity for certain areas but are less appropriate for others.

For example, the tool, SymTA/S [6] (Symbolic Timing Analysis for Systems), can easily be extended with new analysis libraries like ERCOSek and CAN from the automotive domain due to the easy modeling of activation and execution dependencies while the Modular Performance Analysis toolbox (MPA) [18] based on Real-Time Calculus [1] offers a very intuitive and efficient way to capture scheduling hierarchies and distributed systems.

Rather than searching for a unified, more complex model, we exploit the common compositional system level analysis strategy to couple these two methods. In this paper, we explain the models of SymTA/S and MPA, the necessary interfaces, and the analysis integration. We give a complicated example with dependencies and feedback to demonstrate the seamless analysis interaction and the accuracy benefits over using the individual tools alone.

2. COMPOSITIONAL SYSTEM LEVEL ANALYSIS

In this section we explain the basic concepts of the so-called compositional system level analysis methodology, that we use in this paper to integrate the performance analysis approaches SymTA/S and MPA. We first introduce the task model (Section 2.1) which is utilized by local component analysis for performance analysis (Section 2.2). Finally, we explain the compositional system level analysis loop solving the system level performance analysis problem for distributed systems consisting of multiple dependent components (Section 2.3).

2.1 Task model

The task model utilized by state-of-the-art performance analyses assumes that tasks are associated with *best-case* and a *worst-case core execution times (BCET/WCET)*, i.e. minimum and maximum execution times assuming exclusive access to an executing resource.

Tasks are activated and executed due to activating events. Activating events can be generated in a multitude of ways, including expiration of a timer, external or internal interrupt, and task chaining. Each task is assumed to have one input FIFO. A task reads its activating data from its input FIFO and writes data into the input FIFO of a dependent task. A task may read its input data at any time during one

execution. The data is therefore assumed to be available at the input during the whole execution of the task. We also assume that input data is removed from the input FIFO at the end of one execution.

Note that the possible timing of activating events are captured by so-called *event models*. Two well known event models, the *standard event models* used by SymTA/S [6] and the *arrival curves* of the MPA [14, 15] will be presented in Sections 2.4.1 and 2.4.2, respectively.

2.2 Local component analysis

Tasks are mapped on *computation* or *communication resources* to execute. Usually, multiple tasks share the same resource, therefore two or more tasks may request the resource at the same time. In order to arbitrate request conflicts, a resource is associated with a *scheduler* which selects a task to which it grants the resource out of the set of active tasks according to some scheduling policy. Other active tasks have to wait.

Instead of considering each task activation individually, as simulation does, *local component analysis* [8, 5, 1] abstracts from individual activating events to *event streams*. Based on these event streams, analysis can systematically derive worst-case scenarios, which in-turn allow to calculate worst-case (sometimes also best-case) task response times, i.e. the time between task activation and task completion, for all tasks sharing a resource under the control of a scheduler.

Note that the above mentioned analyses guarantee that all observable response times fall into the calculated [best-case, worst-case] interval. These analyses are therefore called conservative.

Additionally, local component analyses determine the communication behavior at the output of the analyzed tasks by considering the effect of scheduling. Therefore, it is usually assumed that tasks produce events at their output at the end of each execution. Just like the timing of the activating events, the output timing behavior is also captured by event models. The way in which these *output event models* are calculated depends on the underlying analysis technique.

2.3 Compositional System Level Analysis Loop

The basic idea of the compositional system level analysis is visualized in Fig. 1, see e.g. [6, 13, 1, 15].

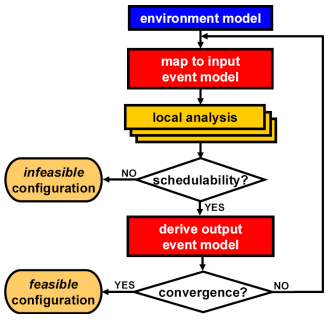

Figure 1: Compositional system level analysis loop.

Compositional system level analysis alternates local scheduling analysis as explained in Section 2.2 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 response times and output event models. Afterwards, the calculated output event models are propagated to the connected components, where they are used as activating event models for the subsequent global iteration.

Obviously, this iterative analysis represents a fix-point problem. 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 (e.g. for the models presented by [2]). In the case where no convergence is reached, no statement can be made about the analyzed system.

Note that for the compositional system level analysis to be applicable, the input event models of all components need to be known or must be computable by local component analysis. For systems containing feed-back between two or more components this is not the case, and thus system level analysis cannot be performed without additional measures. Thereby, the concrete strategy to overcome this problem depends on the specific component types and their input event models. One possibility is the so-called *starting point generation* of SymTA/S which is explained in Section 2.4.1.

2.4 Formal Analysis Techniques

The compositional system level analysis methodology presented in Section 2 defines very clear component interfaces. The only requirements which are imposed to components included into the system level analysis loop is that they accept event models at their inputs and deliver output event models as local analysis results. The compositional system level analysis therefore represents an adequate framework to integrate different performance analysis approaches which we exploit in this paper.

In this section we shortly introduce two performance analysis techniques that we integrate in this paper: SymTA/S [6] and MPA [14, 1, 15]. Note that the technical details of the integration and the benefits of the resulting integrated performance analysis are discussed in Sections 3 and 4, respectively.

2.4.1 SymTA/S

The system level performance analysis approach SymTA/S (Symbolic Timing Analysis for Systems) is based on the principles of compositional system level analysis explained in Section 2. At the component level SymTA/S uses formal analysis techniques based on the busy window technique proposed by Lehoczky [8]. Currently, SymTA/S offers local analysis techniques for fixed priority 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 (Section 2).

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 exactly arrives periodically every 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 P, J. It generally occurs periodically, but it can jitter around its exact position within a jitter interval 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.

Standard event models only capture key timing aspects such as periods, send/receive message jitters, etc., and ignore details of the concrete components. They therefore represent a suitable and easy interface for the coupling of heterogeneous performance analysis approaches.

In order to enable a system level performance analysis of distributed systems with feed-back between two or more components, SymTA/S uses the so-called *starting point generation* to determine initial input event models for all components. The standard starting point generation in SymTA/S consists of propagating the external event models along all task chains in the system until an initial activating event model is available for each task [13]. This approach is safe since on one hand scheduling cannot change an event model period. On the other hand, scheduling can only *increase* an event model jitter [16]. Since a smaller jitter interval is contained in a larger jitter interval, the minimum initial jitter assumption is safe. Details about the output event model calculation and the starting point generation in SymTA/S can be found in [6, 13].

In this paper the abstraction of standard event models to model component interactions will be used to integrate the Real-Time Calculus presented in Section 2.4.2 into the compositional analysis engine of SymTA/S.

2.4.2 Modular Performance Analysis MPA

In [14], Thiele et al. present Real-Time Calculus (RTC), a mathematical model for system-level performance analysis of embedded systems. The corresponding modular analysis method for distributed embedded systems [1, 15] is based on arrival curves and service curves which characterize workload and processing capabilities, respectively. A toolbox for the modular performance analysis (MPA) is available, see [18], which has been used for the experiments in this paper.

For a given event stream, let $R(s,t)$ denote the number of events that arrive in the time interval $[s,t)$. The upper arrival curve, denoted by $\alpha^u(\Delta)$ gives an upper bound on the number of events in any interval Δ. Similarly, a lower bound on the number of events arriving is given by a lower arrival curve $\alpha^l(\Delta)$. R, α^u and α^l are related as follows:

$$\forall s \; : \quad \alpha^l(\Delta) \leq R(s, s+\Delta) \leq \alpha^u(\Delta) \qquad (1)$$

Arrival curves describe timing properties of a whole class of event streams, for example the average rate, burstiness, long-term and short-term behavior. In a similar way, properties of resources are described by service curves, see [14]. Upper and lower arrival curves can also be given for the standard event models that are used in SymTA/S, see for example Fig. 2.

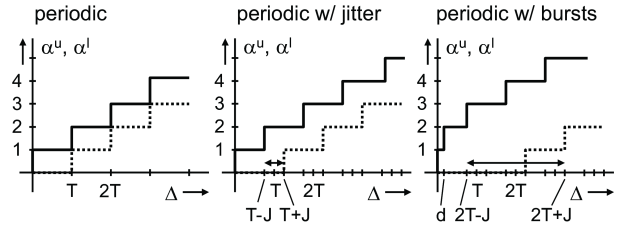

Figure 2: Arrival curves for Standard Event Models.

For efficiency of computation, arrival and service curves can be represented by a finite aperiodic and a periodic part, see [17, 18]. The first is used to describe the bursty behavior, the latter describes the long term behavior and is repeated periodically with a period k. Figure 3 shows an upper arrival curve with an aperiodic part up to time interval Δ_0 and a periodic part with a period k that is repeated infinitely often. This specific representation of arrival and service curves significantly decreases the computational effort of its implementation in the MPA toolbox.

Using MPA, it is possible to determine worst-case bounds on on-chip memory requirements, overall throughput and

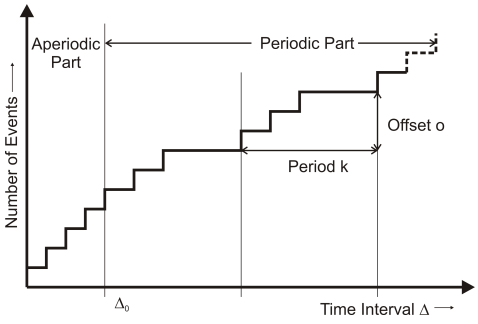

Figure 3: Arrival curve with aperiodic and periodic part.

delay, based on the arrival curves capturing the workload, a task graph describing the structure of the application, a mapping of the tasks to processing and communication units and resource sharing policies on these resources. MPA supports the analysis of computation and communication tasks, several activation schemes such as OR and AND, and several scheduling policies such as Fixed-Priority, EDF, GPS or TDMA. As resource capabilities are first class citizens of the analysis method, hierarchical scheduling methods can be analyzed by an arbitrary combination of the above basic schemes. In addition, various forms of timing and event correlations between streams and within streams can be considered. Furthermore, it is possible to give bounds on the utilization of the processing elements in the system and to get a description of the output event stream after being processed in the form of an arrival curve. For seamless integration into the compositional analysis loop provided by the SymTA/S tool, we need now to convert these output arrival curves into standard output event models, which is described in the next section.

3. COMBINATION OF APPROACHES

As described above, the compositional system analysis is based on either standard event models for SymTA/S, or on arrival curves in the case of MPA. In order to integrate MPA into the analysis loop used in the SymTA/S tool, we have to find a method to convert arrival curves into standard event models and vice versa. In addition to this conversion, a new approach to determine a feasible starting point for the iterative analysis needs to be developed. These topics are discussed in the following sections.

3.1 Event Model Conversion

In this section, we describe how the event model used in MPA, i.e. the arrival curves, can be obtained from SymTA/S event models and vice versa. The translation from SymTA/S event models to MPA arrival curves is lossless and straightforward, as PJD models can directly be mapped to arrival curves. For example, a periodic event model with jitter can be represented by staircase functions with a step width equal to the period T, and a gap between the upper and lower arrival curve that is equal to the jitter $2J$. For a graphical representation and other examples see Fig. 2.

The opposite direction is more involved. The following algorithm describes the necessary steps to obtain SymTA/S PJD models from general arrival curves. Please note, that the translation algorithm is not lossless but provides a safe approximation.

ALGORITHM 1. *Approximation algorithm for PJD model from arrival curve*

Input: α^u,α^l *Upper and lower arrival curve*

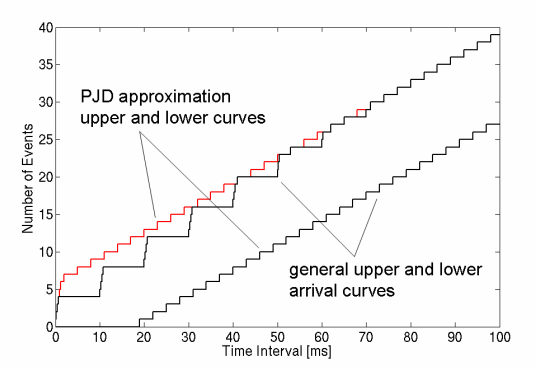

Figure 4: Approximated Standard Event Model (PJD Curve) for generic Event Arrival Curves.

Output: *P,J,D(Period, Jitter, and minimum Distance)*

Step 1: Determine D: *If* $\alpha^u(0+) > 1$ *then* $D = 0$. *Otherwise,* $D = \min\{D_0 : \alpha^u(D_0) = 2\}$.

Step 2: Determine P: *With the period* k *of the periodic part of* α^u, *the starting point* Δ_0 *and the offset* o *with* $o = \alpha^u(\Delta_0 + k) - \alpha^u(\Delta_0)$, *we set* $P = \frac{k}{o}$.

Step 3: Determine initial J: *Define an upper curve* γ^u *corresponding to an event stream with period* P, *i.e.* 1 *event every* P *time units. Then,* $J^u = \min\{J_0 : \gamma^u(\Delta + J_0) \geq \alpha^u(\Delta), \ \forall 0 \leq \Delta \leq \Delta_0 + k\}$.

Step 4: Adjust J: *Define a lower curve* γ^l *corresponding to an event stream with period* P, *i.e.* 1 *event every* P *time units.* $J^l = \min\{J_0 : \gamma^l(\Delta - J_0) \leq \alpha^l(\Delta), \ \forall 0 \leq \Delta \leq \Delta_0 + k\}$. *Set* $J = \max\{J^u, J^l\}$.

Algorithm 1 describes a general method to derive a PJD model from any given arrival curve with $o > 0$, i.e. a non-zero average rate. Otherwise, the approximation with a PJD model does not make sense and a purely bursty approximation would fit the curves better. If we know the period already for some reasons, we can actually skip Step 2. For example, a local component with a single input stream does not change its period.

In Fig. 4 an example of an arrival curve and its PJD approximation is shown. Algorithm 1 delivers PJD models that represent conservative approximations of the given arrival curves. This can be observed graphically in Fig. 4 as the arrival curves representing the PJD model coincide or are larger than the upper curve, and coincide or are smaller than the lower curve, respectively.

3.2 Starting Point Generation

As mentioned in Section 2.3 a so-called starting point needs to be generated under certain circumstances to successfully apply compositional system level performance analysis to a distributed embedded system.

The way in which such a starting point is generated depends on the considered component. The starting point generation in the case of standard SymTA/S components was explained in Section 2.4.1. A similar approach can be pursued to generate a starting point for Real-Time Calculus analyzed components within the SymTA/S engine. One possibility is to propagate the largest period, i.e. generating the least average load, occurring at the component inputs to its outputs and to assume a jitter of zero.

This black-box approach leads to a conservative starting point since the application model utilized by Real-Time Calculus cannot lead to a period increase. The only period changing application property utilized in the Real-Time Calculus model is the OR task activation. This OR activation, however, leads to an decrease of the period [7], and thus to a higher average load.

Note that in the case of components analyzed by a performance analysis approach supporting for instance rate transitions, such a black-box approach is not possible and the responsibility for the starting point generation must be delegated to the components themselves.

Path	MPA	SymTA/S	Combination	Constraint
S1→S4:	170	170	170	200
S2→S5:	430	376	376	400
S3→S6:	412	422	389	400

Table 1: End-to-End path latencies obtained with the different approaches.

4. EXPERIMENTAL RESULTS

In this section we consider the distributed example system in Fig. 5 to demonstrate the efficiency and benefits of the proposed combined performance analysis approach.

The system consists of two computational resources interconnected via a round robin arbitrated bus. CPU 1 executes four tasks which are scheduled according to a hierarchical scheduling policy with TDMA resource sharing at the top level. T1 and T2 are dispatched by a priority scheduled server and share 60% of the CPUs service capacity. T3 and T4 are each assigned 20% of the available CPU time. As the scheduling scheme in CPU 1 is hierarchical, MPA is used for the combined performance analysis approach.

CPU 2 executes four tasks, which are scheduled by the priority based ERCOSek operating system. The speciality of CPU 2, which needs to be considered for performance analysis, is the cooperative behavior of the tasks T7 and T8. Generally, T7 has a higher priority than T8. However, due to the cooperative policy T7 can interrupt T8 only at specific points in time once T8 is executing. In the given example system T8 can be interrupted by T7 only after 2, 3, 4, 6, or 11 time units during its execution. More details about ERCOSek can be found in [3]. For the combined performance analysis approach CPU2 as well as the BUS are analyzed by SymTA/S.

For correct operation the system has to satisfy three end-to-end constraints: the maximum latency along the paths $S2 \rightarrow S5$ and $S3 \rightarrow S6$ must be less than 400 time units, and the maximum latency along the path $S1 \rightarrow S4$ must not exceed 200 time units.

4.1 Path Latency Estimation

In order to see the influence of the combined approach, we analyzed the example system with (1) MPA only, (2) SymTA/S only, and (3) the combination of the two approaches. The results of the three approaches are given in Tab. 1. From the analysis results obtained for cases (1) and (2) we find that the path latency constraints are violated. Therefore, we would redesign the system, although the combined analysis shows that the system perfectly meets the constraints.

We observe that the worst-case latencies predicted by the combined approach are more than 10% tighter than the results achieved by the single methods for some end-to-end paths. Considering the small size and the simplicity of the considered example system this represents a remarkable improvement of the analysis accuracy.

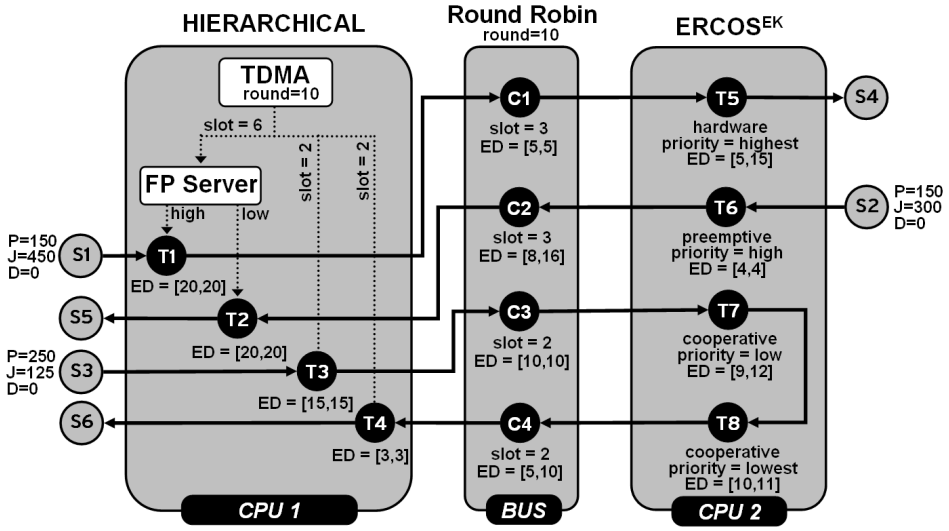

Figure 5: Distributed example system.

4.1.1 *Analysis using MPA alone*

To approximate the ERCOSek scheduler running on CPU 2 in the example with MPA where no built-in analysis technique for ERCOSek exists, we used a fixed-priority scheduling for tasks T5 (highest priority) and T6 (second highest priority). In order to determine the best-case and worst-case response times of T7, it was assigned a higher and a lower priority than T8, resp. T8 was analyzed the same way, i.e. assuming to have a higher/smaller priority than T7.

Note that the analysis using these approximation for MPA was only possible, because of the simple ERCOSek task configuration chosen for the illustrative example. In fact only the timing consequences of two cooperative tasks (T7 and T8) needed to be considered. However, in the general case the ERCOSek task behavior can be far more complicated disabling an easy approximation by MPA. Examples are so-called "Time Tables" specifying phase offsets between periodic tasks. Furthermore, "Alarms" use dynamic time-out mechanisms and can be issued and disabled at any point in time. Finally, scheduling-related OS routines can request a significant amount of execution time at various priority levels.

4.1.2 *Analysis using SymTA/S alone*

Since SymTA/S does not directly support hierarchical scheduling policies, the timing behavior of the tasks running on CPU 1 needs to be approximated in order to analyze the example system using only SymTA/S. Figure 6 shows one possible approximation using standard SymTA/S components.

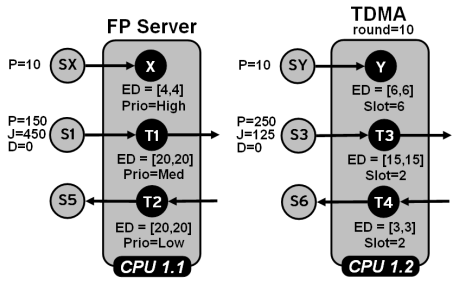

Figure 6: Approx. hierarchical scheduling on CPU1.

The FP server is modeled by the fixed-priority scheduled CPU 1.1. In order to account for the reduced service capacity of the FP server, which is due to the TDMA scheduling policy used at the top hierarchy level, we introduce the additional task X. X is activated every 10 time units and has an execution demand of 4 time units. It therefore reduces the service capacity of CPU 1.1 by the amount of resources claimed by $T3$ and $T4$. The timing behavior of the TDMA-scheduled tasks $T3$ and $T4$ is approximated by CPU 1.2. The TDMA slot, which is allocated to the FP server is modeled by the additional task Y.

Note that for the considered case the approximation of the hierarchical scheduling on CPU 1 by SymTA/S was quite straight-forward. However, in the general case, i.e. considering more complicated hierarchical scheduling policies with several levels of hierarchy, similar approximations are far more complicated and result in large overestimations of the timing behavior.

4.2 Robustness to execution demand variations

In addition to the path latency estimations presented in the last section, the SymTA/S tool also supports automatic sensitivity analysis. The seamless integration of MPA into the SymTA/S tool allows us to perform the sensitivity analysis across domain borders, e.g. enabling exploration of suitable regions of the task execution demand in our example system. In Fig. 7, the feasible regions for task execution demands for three tasks are given. If the task execution demand lies in the feasible region, the path latency constraints for the three paths in the system are met. For example, the task execution demand for task T5 must lie below 16 time units for the system to meet all timing constraints.

From Fig. 7 we can observe that especially for task T1 our example system exposes an interesting behavior to execution demand changes. Therefore, we further investigated the influence of T1's execution demand to the system path latencies. In Fig. 8, we can see the inter-dependencies of a variation in the task execution of T1 and the various path latencies. For instance, if we slightly increase the task execution time of T1 from 23 to 24 time units, the path latency of the feedback path from S3 to S6 drops significantly.

Note that a detailed study of such sensitivity analysis plots allows the designer to judge the robustness of the overall system to changes of individual parameters. In our

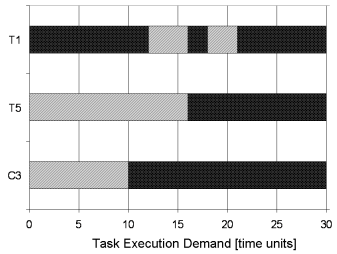

Figure 7: Feasible regions for task execution demands of T1, C3, and T5. Feasible regions are light, unfeasible regions are marked dark.

case study, we only varied one parameter at the time. However, SymTA/S also supports multi-parameter variation [12]. With the results presented in this paper these techniques can now also be applied across different analysis domains.

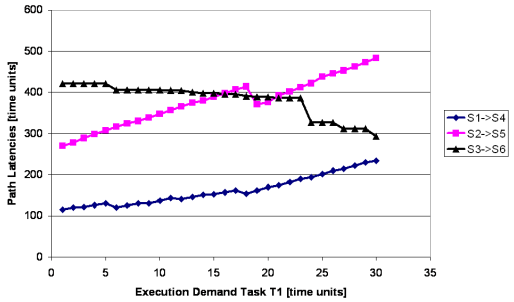

Figure 8: Path latencies vs. task execution demand for task T1.

5. ACKNOWLEDGMENTS

This work was partly funded through the European Network ARTIST2.

6. CONCLUSIONS

In this paper, we presented a combined approach to system level performance analysis of embedded systems. Using this combined approach, we found better analysis results compared to stand-alone analysis techniques. Furthermore, we were able to perform robustness analysis across analysis domain borders. We strongly believe that such combinations of heterogeneous methods could be explored much more, leading to further improved analysis capabilities and accuracy for complex distributed embedded systems.

7. REFERENCES

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

[2] S. Chakraborty, L. T. X. Phan, and P. S. Thiagarajan. Event count automata: A state-based model for stream processing systems. In *Proc. 26th IEEE International Real-Time Systems Symposium (RTSS)*, pages 87–98, Washington, DC, USA, 2005.

[3] ETAS: $ERCOS^{EK}$ V4.1 User's guide.

[4] M. G. Harbour, J. J. G. García, J. C. P. Gutiérrez, and J. M. D. Moyano. MAST: Modeling and analysis suite for real time applications. In *Proc. 13th Euromicro Conference on Real-Time Systems ECRTS*, page 125, Washington, DC, USA, 2001.

[5] M. G. Harbour and J. C. P. Gutiérrez. Response time analysis for tasks scheduled under EDF within fixed priorities. In *Proc. Real-Time Systems Symposium (RTSS)*, pages 200–209, 2003.

[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] M. Jersak, K. Richter, and R. Ernst. Performance analysis for complex embedded applications. *International Journal of Embedded Systems, Special Issue on Codesign for SoC*, 2004.

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

[9] A. Nandi and R. Marculescu. System-level power/performance analysis for embedded systems design. In *Proc. 38th conference on Design automation (DAC)*, pages 599–604, New York, NY, USA, 2001.

[10] C. Norström, A. Wall, and W. Yi. Timed automata as task models for event-driven systems. In *Proc. 6th International Conference on Real-Time Computing Systems and Applications (RTCSA)*, page 182, Washington, DC, USA, 1999.

[11] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proc. 10th international symposium on Hardware/software codesign (CODES)*, pages 187–192, New York, NY, USA, 2002.

[12] R. Racu, A. Hamann, and R. Ernst. A formal approach to multi-dimensional sensitivity analysis of embedded real-time systems. In *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*, Dresden, Germany, July 2006.

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

[14] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. International Symposium on Circuits and Systems*, pages 101–104, Geneva, Switzerland, March 2000.

[15] L. Thiele, E. Wandeler, and S. Chakraborty. A stream-oriented component model for performance analysis of multiprocessor dsps. *IEEE Signal Processing Magazine*, 22(3):38—-46, May 2005.

[16] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocess. Microprogram.*, 40(2-3):117–134, 1994.

[17] E. Wandeler. *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems*. PhD thesis, Swiss Federal Institute of Technology, September 2006.

[18] E. Wandeler and L. Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.

[19] T.-Y. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. In *Proc. International Conference on Computer Design (ICCD)*, pages 64–71, Washington, DC, USA, 1995.