Formal Methods for Automotive Platform Analysis and Optimization*

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

***Abstract*— There have been major advances in formal methods and related tools in embedded system design in recent years that support analysis and optimization of heterogeneous automotive architectures. We give an introduction of the tool SymTA/S and demonstrate its application to an automotive example where we analyze system sensitivity and explore the design space. Such results cannot be obtained by simulation or prototyping.**

I. Introduction

With increasing automotive system complexity, formal methods become more important as a complement to simulation and prototyping. Formal methods have already been used in early architecture design, e.g. to optimize bus architectures as such models can be applied before executable models are available. Recent advances in real-time system analysis have extended the scope of formal models to heterogeneous networks with different protocols and gateways, and with electronic control units (ECUs) running different scheduling algorithms. These formal models can be used to optimize those systems and analyze the sensitivity to later changes in the design process or to run time events such as retransmission due to an error. In this paper, we give a brief introduction to the underlying model and algorithms of the tool, SymTA/S, and give a small example for its application to system integration.

The remainder of this paper is structured as follows. First, we will give a short overview of existing formal techniques for system level performance analysis (section II). Afterwards, we give an introduction into the formal core of SymTA/S, including the application model, the utilized standard event models and the compositional analysis methodology based on event model propagation (section III).

We then discuss the most important concepts which are taken into account by the high-accuracy analyses techniques of SymTA/S for the $ERCOS^{EK}$ operating system and the CAN bus protocol (section IV). Both technologies are well established in the automotive industry and deployed in various car types and product lines.

Finally, we introduce a small automotive example system consisting of two independent subsystems (section V) and demonstrate the application of the SymTA/S exploration and sensitivity analysis frameworks to system integration (section VI to IX).

*This work was supported by the German DFG under ER 168/18-1

II. Formal Techniques in System Performance Analysis

In this section, we briefly review existing analysis approaches from real-time research for formal system performance analysis of heterogeneous distributed systems and MpSoC.

The holistic analysis approach developed by Tindell [16] systematically extended the classical local analysis techniques, considering the scheduling influences along functional paths in the system. He proposed a performance verification model for distributed real-time systems with preemptive task sets communicating via message passing and shared data areas. Eles *et al.* [7] extended this approach for systems consisting of fixed-priority scheduled CPUs connected via a TDMA scheduled bus. Later on, Palencia *et al.* [5], [6] extended the analysis for tasks with precedence relations and activation offsets.

Gresser [2] and Thiele [14] established a different view on scheduling analysis. The individual components or subsystems are seen as entities which interact, or communicate, via event streams. Mathematically speaking, the stream representations are used to capture the dependencies between the equations (or equations sets) that describe the individual components timing. The difference to the holistic approach (that also captures the timing using system-level equations) is that the compositional models are well-structured with respect to the architecture. This is considered a key benefit, since the structuring significantly helps designers to understand the complex dependencies in the system, and it enables a surprisingly simple solution. In the "compositional" approach, an output event stream of one component turns into an input event stream of a connected component. Schedulability analysis, then, can be seen as a flow-analysis problem for event streams that, in principle, can be solved iteratively using event stream propagation.

III. The SymTA/S approach

SymTA/S [3] is a formal system-level performance and timing analysis tool for heterogeneous SoCs and distributed systems. A key novelty of the SymTA/S approach is that it uses intuitive *standard event models* (section III-B) from real-time systems research rather than introducing new, complex stream representations. Periodic events or event streams with jitter and bursts [15] are examples of standard models that can be found in literature. The SymTA/S technology allows to extract

this information from a given schedule and automatically interface or adapt the event stream to the specific needs within these standard models, so that designers and analysts can safely apply existing subsystem techniques of choice without compromising global analysis.

The application model of SymTA/S is described in section III-A. The core of SymTA/S is a technique to couple local scheduling analysis algorithms using event streams [10], [13]. Event streams describe the possible I/O timing of tasks. Input and output event streams are described by standard event models which are introduced in detail in section III-B. The analysis composition using event streams is described in section III-C.

A. Application Model

A task is activated due to an activating event. Activating events can be generated in a multitude of ways, including expiration of a timer, external or internal interrupt, and task chaining. Task communication in SymTA/S is modeled either using FIFOs or registers.

In the case of FIFO communication, 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. SymTA/S assumes that input data is removed from the input FIFO at the end of one execution.

Register communication in SymTA/S requires that the sender task writes the data into register before the receiver task initiates the read routine. Therefore, this type of communication is only suited for time-triggered protocols. Note that in the case of register communication causal dependencies between communicating tasks cannot be exploited.

A task needs to be mapped on a *computation* or *communication resource* to execute. When multiple tasks share the same resource, then 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. *Scheduling analysis* calculates 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. Scheduling analysis guarantees that all observable response times will fall into the calculated [best-case, worst-case] interval. Scheduling analysis is therefore conservative. A task is assumed to write its output data at the end of one execution. This assumption is standard in scheduling analysis.

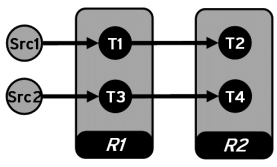


Fig. 1 - SYSTEM MODELED WITH SYMTA/S

Figure 1 shows an example of a system modeled with SymTA/S. The system consists of 2 resources each with 2 tasks mapped on it. $R1$ and $R2$ are both assumed to be priority scheduled. $Src1$ and $Src2$ are the sources of the external activating events at the system inputs. The possible timing of activating events is captured by so-called *event models*, which are introduced in section III-B.

B. SymTA/S Standard Event Models

Standard event models represent the possible timing of activating events of tasks in SymTA/S. They are described using several parameters. For example, a *strictly 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 $\mathcal{J}$. Consider an example where $(\mathcal{P}, \mathcal{J}) = (4, 1)$. This event model is visualized in figure 2. Each gray box indicates a jitter interval of length $\mathcal{J} = 1$. The jitter intervals repeat with the event model period $\mathcal{P} = 4$. The figure additionally shows a sequence of events which satisfies the event model, since exactly one event falls within each jitter interval box, and no events occur outside the boxes.

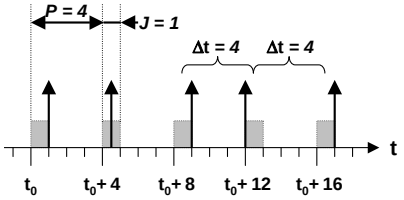


Fig. 2 - EXAMPLE OF AN EVENT STREAM THAT SATISFIES THE EVENT MODEL ($\mathcal{P} = 4$, $\mathcal{J} = 1$)

Periodic with jitter event models are well suited to describe generally periodic event streams, which often occur in control, communication and multimedia systems [11]. If the jitter is zero, then the event model is strictly periodic. 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.

Additionally, *sporadic* events are also common [11]. Sporadic event streams are modeled with the same set of parameters as periodic event streams. Note that *jitter* and d^- parameters are also meaningful in sporadic event models, since they allow to accurately capture sporadic transient load peaks.

A more detailed discussion about the event models used in SymTA/S can be found in [12].

C. Analysis composition

In the SymTA/S compositional performance analysis methodology [11], [12], local scheduling analysis and event model propagation are alternated, during system-level analysis. This requires the modeling of possible timing of output events

for propagation to the next scheduling component. In the following, first the output event model calculation is explained. Then the compositional analysis approach is presented.

1. Output event model calculation

The SymTA/S standard event models allow to specify simple rules to obtain output event models that can be described with the same set of parameters as the activating event models. The output event model period obviously equals the activation period. The difference between maximum and minimum response times (the response time jitter) is added to the activating event model jitter, yielding the output event model jitter (equation 1).

$$\mathcal{J}_{out} \quad = \quad \mathcal{J}_{act} + (t_{resp,max} - t_{resp,min}) \qquad (1)$$

Note that if the calculated output event model has a larger jitter than period, this information alone would indicate that an early output event could occur before a late previous output event, which obviously cannot be correct. In reality, output events cannot follow closer than the minimum response time of the producer task. This is indicated by the value of the *minimum distance* parameter d^-.

2. Analysis composition using standard event models

In the following, the compositional analysis approach is explained using the system example in figure 1. Initially, only event models at the external system inputs are known. Since an activating event model is available for each task on $R1$, a local scheduling analysis of this resource can be performed and output event models are calculated for $T1$ and $T3$ (section III-C.1). In the second phase, all output event models are propagated. The output event models become the activating event models for $T2$ and $T4$. Now, a local scheduling analysis of $R2$ can be performed since all activating event models are known.

However, it is sometimes impossible to perform system level scheduling analysis as explained above. This is shown in the system example in figure 3.

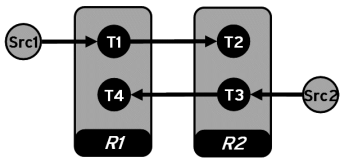


Fig. 3 - EXAMPLE OF A SYSTEM WITH CYCLIC SCHEDULING DEPENDENCY

Figure 3 shows a system consisting of 2 resources, R_1 and R_2, each with 2 tasks mapped on it. Initially, only the activating event models of $T1$ and $T3$ are known. At this point the system cannot be analyzed, because on every resource an activating event model for one task is missing. I.e. response times on R_1 need to be calculated to be able to analyze R_2. On the other hand, R_1 cannot be analyzed before analyzing R_2. This problem is called *cyclic scheduling dependency*.

One solution to this problem is to initially propagate all external event models along all system paths until an initial activating event model is available for each task [9]. 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 [15]. Since a smaller jitter interval is contained in a larger jitter interval, the minimum initial jitter assumption is safe.

After propagating external event models, global system analysis can be performed. A global analysis step consists of two phases [12]. In the first phase local scheduling analysis is performed for each resource and output event models are calculated (section III-C.1). In the second phase, all output event models are propagated. It is then checked if the first phase has to be repeated because some activating event models are no longer up-to-date, meaning that a newly propagated output event model is different from the output event models that was propagated in the previous global analysis step. Analysis completes if either all event models are up-to-date after the propagation phase, or if an abort condition, e. g. the violation of a timing constraint has been reached.

IV. AUTOMOTIVE EMBEDDED TECHNOLOGY

In the following sections we give a brief overview about the most important concepts taken into account by the high-accuracy analyses techniques of SymTA/S for the $ERCOS^{EK}$ operating system (section IV-A) and the CAN bus protocol (section IV-B). Both technologies are well established in the automotive industry and deployed in various car types and product lines.

A. $ERCOS^{EK}$

The $ECROS^{EK}$ operating system builds upon the core ideas of static-priority preemptive (SPP) scheduling. However, this underlying scheduling policy is extended by a variety of additional concepts.

$ERCOS^{EK}$ distinguishes *hardware tasks (interrupts)*, *preemptive software tasks* and *cooperative software tasks*. Software tasks are comprised of processes that are executed sequentially. In contrast to preemptive software tasks, cooperative software tasks preempt each other only at process boundaries. This reduces the context switch overhead but results in additional blocking for the higher priority cooperative tasks.

Certain scheduling-related OS routines can request a considerable amount of execution time at various priority levels. For instance, the *activate task* and *terminate task* routines are called by the OS before and after task execution, respectively. Both OS routines are executed with the so-called *kernel priority*, which is higher than the priority of all software tasks.

Figure 4 shows the complex OS priority set-up of $ERCOS^{EK}$ in SymTA/S with different priority regions for the mentioned task types.

Furthermore, task activation can be initiated in a variety of ways. So called “Time Tables” allow the specification of periodic tasks with phase offsets (startup delays) between them. “Alarms” use more dynamic time-out mechanisms and can be issued and disabled at any point in time. Finally, tasks can be activated from software tasks and interrupts (hardware tasks) dynamically and bursty.

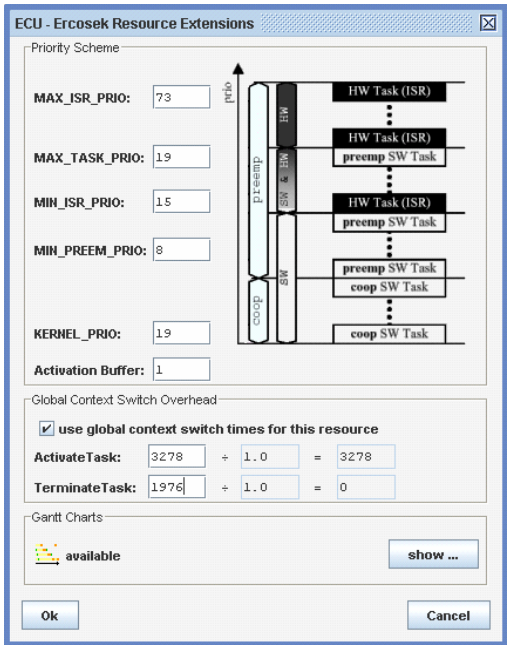


Fig. 4 - COMPLEX $ERCOS^{EK}$ PRIORITY SET-UP

Figure 5 shows the worst-case scenario of the cooperative task $T2$ scheduled on an $ERCOS^{EK}$ arbitrated resource. All tasks in the Gantt-chart are periodically activated by a time table without startup delays.

First of all we observe the cascading calls of the activate task functions and the terminate task functions preempting all other execution requests. Furthermore, we observe the cooperative behavior of $T2$ blocked by the process $P0$ of the lower priority cooperative task $T4$.

B. CAN (Control Area Network)

The CAN bus protocol is also based on static priorities but the message transmission is non-preemptive, as typical for serial line protocols.

Compared to the $ERCOS^{EK}$ behavior, the actual CAN protocol is relatively simply. However, due to cost reasons, CAN interfaces are typically realized with a very limited number of sender buffers, so-called *message objects*. A message, once written into such a buffer, can not be "overtaken" by another higher-priority message that is generated later. This behavior can turn the static-priority scheme into a complex queuing scheme when it comes to scheduling analysis.

Furthermore, CAN messages inherit the time-table-like behavior of the tasks that generate the messages. In combination with dynamic phase shifts between request and acknowledge frames in an end-to-end path this leads to complex best- and worst-case scheduling scenarios.

Finally, transmission errors can enforce retransmissions that add to the overall load and message latency.

V. AUTOMOTIVE EXAMPLE SYSTEM

Figure 6 shows a SymTA/S model of two electronic subsystems. The two subsystems are functionally independent and are designed separately by two different electronics suppliers.

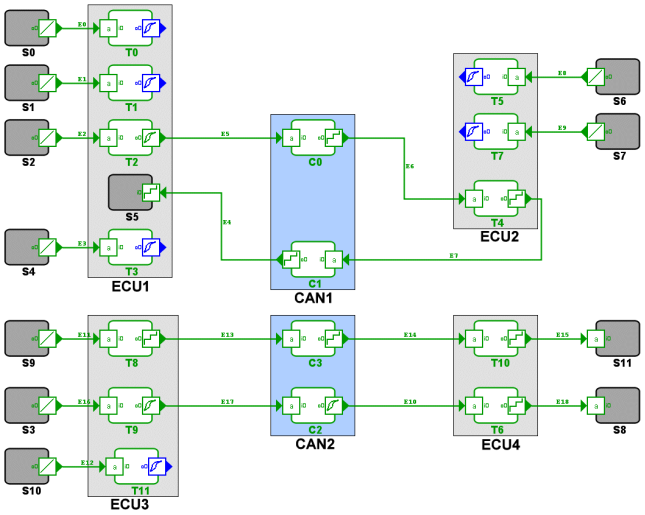


Fig. 6 - AUTOMOTIVE SYSTEM EXAMPLE: INDEPENDENT SUBSYSTEMS

The automotive OEM would like to integrate both subsystems in a vehicle.

Note that $ECU1$ is arbitrated by the $ERCOS^{EK}$ operating system, whereas all other $ECUs$ are arbitrated according to the static priority preemptive policy (SPP). The buses used to exchange messages between the ECUs in both subsystems are running the CAN protocol.

The core execution times, i.e. without scheduling influences, of the software-functions running on the ECUs are given in table (a). The communication delays of the messages exchanged over the CAN buses, assuming no concurrent communication requests, are given in table (b). The tables (a) and (b) additionally contain the priorities of each software-function as well as the IDs of the CAN messages (correspond to priorities). Except for the $ERCOS^{EK}$ scheduled software-functions lower values correspond to higher priorities.

The gray, rounded boxes model the activation of the software-functions (in this case, timers). The activation periods are given in table (c).

Note that both subsystems need to satisfy certain timing constraints in order to function correctly:

- deadline of 15 time units for the path $S2 \rightarrow S5$
- deadline of 15 time units for the path $S9 \rightarrow S11$
- deadline of 15 time units for the path $S3 \rightarrow S8$

Figure 6 shows both subsystems before integration. Both the upper and the lower subsystem are implemented on 2 ECUs (vertical square boxes on the left and right) that exchange messages over a dedicated CAN bus (vertical square boxes in the middle).

Timing and performance analysis of both subsystems with SymTA/S [3] reveals the following worst-case delays for the constrained paths:

- 12.01 time units for the path $S2 \rightarrow S5$
- 5.12 time units for the path $S9 \rightarrow S11$
- 9.92 time units for the path $S3 \rightarrow S8$

If we compare these worst-case delays with the deadlines imposed by system specification, we observe that both suppliers implemented their subsystems so that all timing constraints are satisfied.

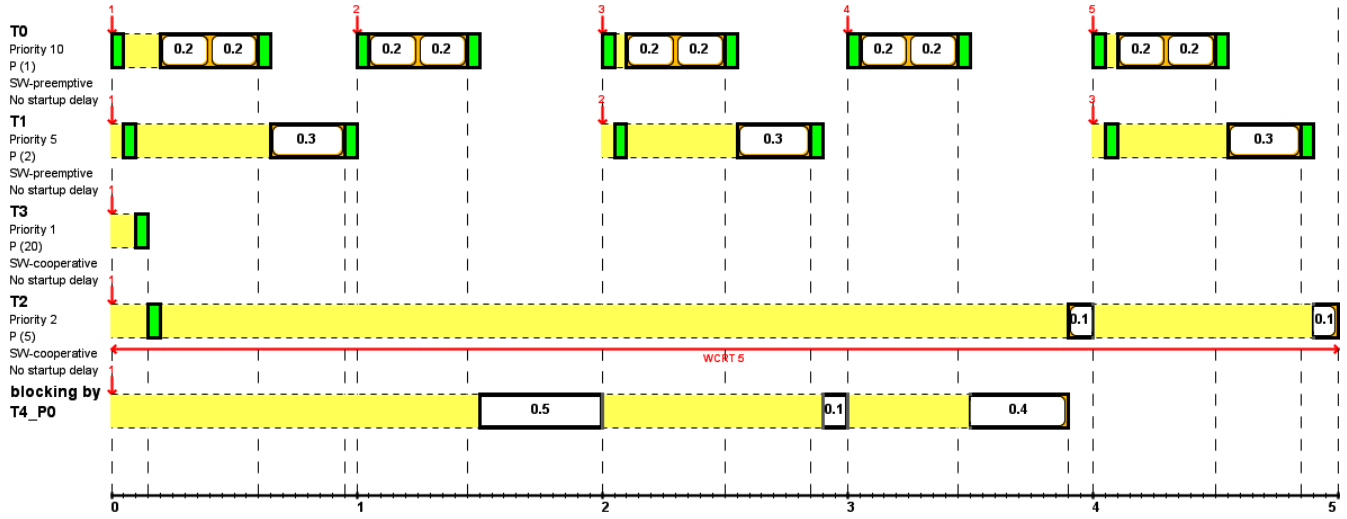


Fig. 5 - $ERCOS^{EK}$ WORST-CASE SCHEDULING EXAMPLE

VI. Sensitivity Analysis

The robustness of an architecture to parameter changes is a major concern in the design of embedded real-time systems. Robustness is important in early design stages to identify if and in how far a system can accommodate later changes or updates. Furthermore, system robustness is an important metric in the later design phases during subsystem or third-party component integration. In general, the system robustness is defined by the available headroom or slack corresponding to different properties of the system components. These can be task execution demands, channel communication times, the parameters of the activation models, the speed of the computation resources or the throughput of the communication buses.

Sensitivity analysis allows the system designer to permanently keep track of the system robustness, and thus to quickly asses the impact of changes of individual hardware or software components on system performance. Section VI-A gives a short overview on the sensitivity analysis framework implemented in SymTA/S. In Section VI-B we determine the robustness of the independent subsystems presented in Figure 6 with respect to variations of different parameters.

A. Sensitivity Analysis Framework

The sensitivity analysis framework implemented in SymTA/S combines a binary search technique and the compositional analysis model presented in Section III-C.2. The binary search technique is known as a simple and fast search algorithm used to determine a specific values within an ordered set of data. Since the variations of specific system parameters like execution demands, activation periods, resource speeds have a monotonic impact on the set of system timing properties, the binary search can quickly determine the values of these parameters that leading to conforming system configurations. A detailed description of the sensitivity analysis framework is presented in [8].

Parameters for sensitivity analysis are any system properties that may vary during the design process. Very common are variations of execution times, the parameters of the activation models, like period, jitter and offset, communication volumes, bus and processor speeds. The variations of these parameters affect different system performance metrics, like task response times, end-to-end deadlines, output jitters, buffer sizes or deadline miss-ratio in case of soft real-time systems.

B. The Robustness of the Independent Subsystems

In this section we determine the robustness of the independent subsystems presented in Figure 6. Firstly, we investigate the available slacks of the execution times of the tasks mapped on the ECUs and of the communication channels mapped on the CAN resources. The results are presented in Figure 7(a). We observe that tasks $T2$, $T5$, $T7$ and channel $C2$ have a very large flexibility compared with the other tasks and channels. This is explained by the fact that these tasks belong to functional paths without or with loose timing constraints.

Figure 7(b) shows the flexibility of the operational speeds of computation and communication resources. The minimum speed of these resources is determined on one hand by the utilization factor, and, on the other hand, by the timing constraints defined for the tasks executed on these resources.

The last investigated set of parameters are the task execution rates defined at system inputs. Figure 7(c) shows the maximum decrease permitted for task activation periods without violating the set of constraints or the system schedulability.

The resulting slacks of investigated system parameters represent an additional argument for the feasible integration of the two independent subsystems. The available resource headroom allows all messages to be transmitted on a single bus without disturbing too much the performance of the other system components.

VII. Integrating both subsystems

We now integrate both independent subsystems. Figure 8 shows the system after integration. Instead of utilizing a dedicated CAN bus for each of the subsystem, all messages are now transmitted over a single CAN bus. Of course, this

ECU1 ($ERCOS^{EK}$)		
Task	core exec. time	Priority
T0	[0.2,0.3]	10 (preem)
T1	[0.2,0.3]	5 (preem)
T2	[0.1,0.2]	2 (coop)
T3	[1.2,2.3]	1 (coop)
ECU2 (SPP)		
Task	core exec. time	Priority
T4	[0.7,0.8]	2
T5	[0.1,0.2]	3
T7	[0.1,0.6]	1
ECU3 (SPP)		
Task	core exec. time	Priority
T8	[0.3,0.4]	2
T9	[1,2]	1
T11	[3,5]	3
ECU4 (SPP)		
Task	core exec. time	Priority
T6	[1.1,1.5]	2
T10	[0.3,0.6]	1

(a) Computational Tasks

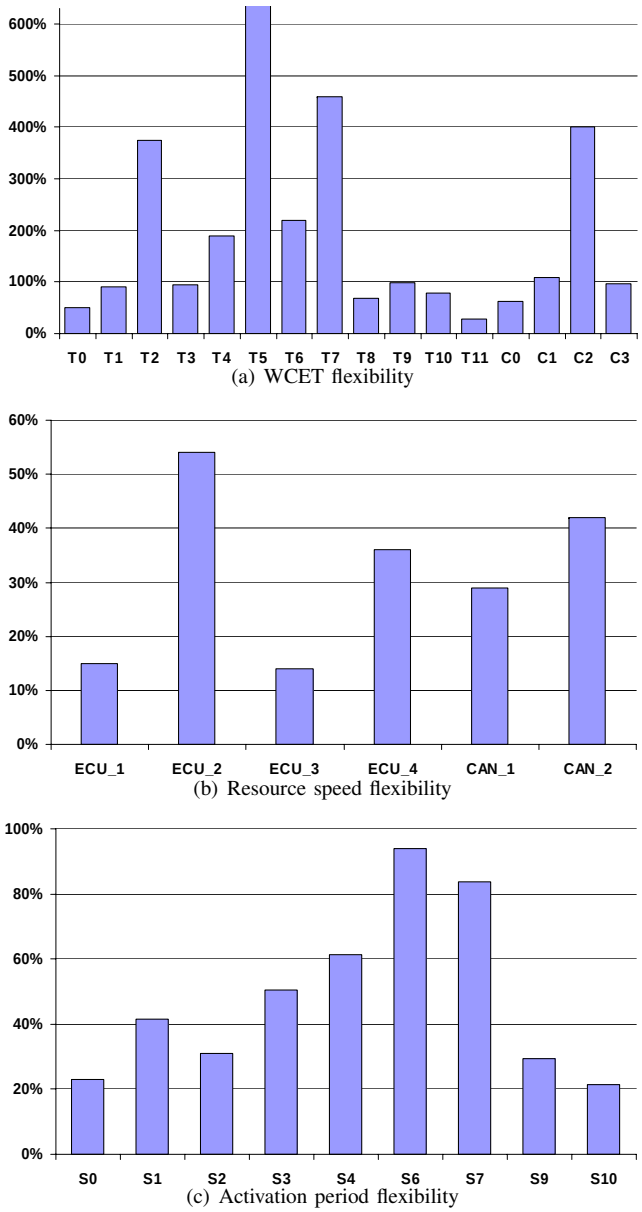
CAN1 ($ControlAreaNetwork$)		
Channel	core comm. time	Priority
C0	[1.08,1.32]	1
C1	[0.76,0.92]	3
CAN2 ($ControlAreaNetwork$)		
Channel	core comm. time	Priority
C2	[0.6,0.72]	4
C3	[0.68,0.82]	2

(b) Communication Tasks

Input	Event Model
$S0$	periodic, $\mathcal{P}_1 = 1$
$S1$	periodic, $\mathcal{P}_1 = 2$
$S2$	periodic, $\mathcal{P}_2 = 5$
$S3$	periodic, $\mathcal{P}_3 = 15$
$S4$	periodic, $\mathcal{P}_3 = 20$
$S6$	periodic, $\mathcal{P}_3 = 5$
$S7$	periodic, $\mathcal{P}_3 = 7$
$S9$	periodic, $\mathcal{P}_9 = 2$
$S10$	periodic, $\mathcal{P}_{10} = 10$

(c) Input Event Models

TABLE I - SYSTEM PARAMETERS



(a) WCET flexibility

(b) Resource speed flexibility

(c) Activation period flexibility

Fig. 7 - FLEXIBILITY OF THE INITIAL SYSTEM CONFIGURATION

leads to additional load and potentially longer blocking of low-priority messages.

Again we verify performance and timing of the system with SymTA/S [3], and obtain the following worst-case delays for the constrained paths:

- 20.18 time units for the path $S2 \rightarrow S5$
- 7.92 time units for the path $S9 \rightarrow S11$
- 33.09 time units for the path $S3 \rightarrow S8$

We observe that the constrained paths $S2 \rightarrow S5$ and $S3 \rightarrow S8$ exceed their deadlines by 34.5% and 120.6%, respectively.

VIII. SYSTEM OPTIMIZATION

In section VII we have seen that the integration of two independently working subsystems is usually not possible in a straight-forward manner. In practice, system parameters like CAN message IDs, task priorities, and time slots need to be adapted to successfully integrate several subsystems.

In the following we first (section VIII-A) introduce the design space exploration framework of SymTA/S [4], assisting the designer in exploring the configuration space of complex distributed systems and pointing out pareto-optimal system configurations with respect to an arbitrary numbers of optimization criteria, including timing properties, power consumption, buffer sizes, etc. In the second part (section VIII-B), we then use this framework to explore the integrated automotive example system.

A. Design Space Exploration Framework

Figure 9 shows the design space exploration framework [4] of SymTA/S [3]. The *Optimization Controller* is the central element. It is connected to the scheduling analysis of SymTA/S and to an evolutionary multi-objective optimizer. SymTA/S checks the validity of a given system parameter set, that is represented by an individual, in the context of the overall

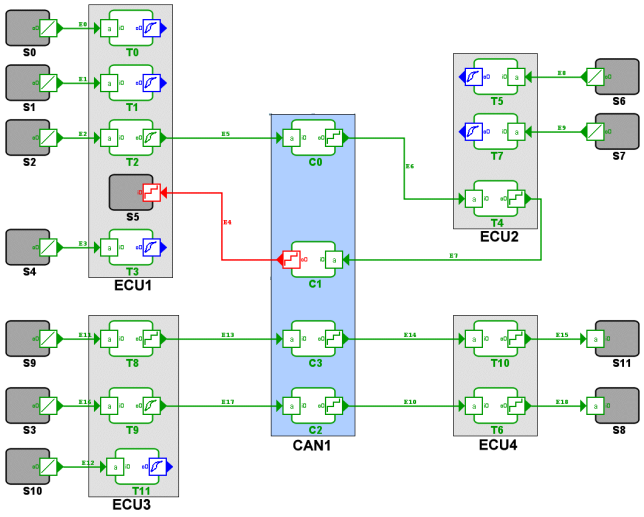


Fig. 8 - AUTOMOTIVE SYSTEM EXAMPLE: AFTER SYSTEM INTEGRATION

heterogeneous system. The evolutionary multi-objective optimizer is responsible for the problem-independent part of the optimization problem, i.e. elimination of individuals and selection of interesting individuals for variation. Currently, we use SPEA2 (Strength Pareto Evolutionary Algorithm 2) [17] for this part, which is coupled via PISA (Platform and Programming Language Independent Interface for Search Algorithms) [1].

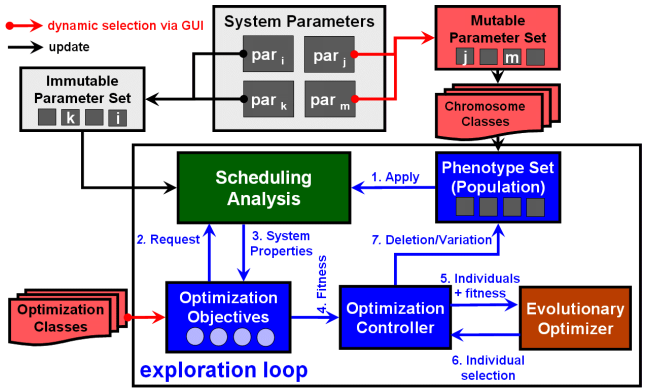


Fig. 9 - EXPLORATION FRAMEWORK

Different parameters of a system, such as priorities or time slots, are encoded on separate *chromosomes*. The user selects a subset of all parameters for optimization. The chromosomes of these parameters form an *individual* and are included in the evolutionary optimization while all others are fixed and immutable. The variation operators of the evolutionary algorithm are applied chromosome-wise for these individuals. More details on the optimization system can be found in [4].

B. Exploring the integrated system

In this section we use the design space exploration framework of SymTA/S (section VIII-A) to explore the integrated automotive system (section VII).

The search space of our exploration consists of the priority assignments on all ECUs as well as the assignment of CAN message IDs on the bus. Since the straight-forward integration of the two subsystems lead to the violation of hard timing constraints, our primary optimization objective is to find a working system configuration. Additionally we are interested in pareto-optimal trade-offs between the three constrained end-to-end paths.

In the real world, no single design team has full control over the entire system. Instead, numerous design teams from different companies contribute along an automotive supply chain. Each team is in control of only part of the system. Therefore, system-level exploration across team-boundaries is a complicated task. The OEM as the bus integrator, for instance, usually controls CAN message IDs, but has very limited insight into the configuration of the ECUs.

Unfortunately, dynamic behavior between several subsystems usually cannot be observed until late in the design process when first ECU prototypes become available. By that time, it is very costly to re-assign system parameters like priorities of software-functions or CAN message IDs.

In order to account for the difficulty and the high cost of system parameter modifications at system integration time, we add the minimization of parameter changes to the optimization objectives considered during design space exploration. More precisely, we are interested in finding working system configurations with as few parameter changes as possible.

Table VIII-B shows the pareto-optimal system configurations obtained by an exploration considering 1000 system configurations (40 generations with 25 individuals each). Note that this exploration took approximately 70 seconds on a standard PC running at a clock-rate of 2.4 GHz.

#	$S2 \rightarrow S5$	$S9 \rightarrow S11$	$S3 \rightarrow S8$	# param. changes
1	12.24	10.12	11.52	9
2	12.24	13.72	10.72	7
3	12.86	10.12	11.52	7
4	12.99	10.12	8.87	8
5	12.99	13.72	8.07	6
6	13.84	8.42	12.97	8
7	13.84	10.12	6.57	10
8	13.84	12.12	14.57	5
9	13.84	13.82	5.77	7
10	14.06	9.37	12.97	7
11	14.46	9.37	12.97	5
12	14.46	11.07	6.57	7
13	14.46	12.97	14.57	4

TABLE II - PARETO-OPTIMAL SOLUTIONS

We observe that we found 13 pareto-optimal working system configurations, each of them representing an optimal trade-off between the constrained timing properties and the number of parameter changes with respect to the initial configuration.

In order to decide which system configuration to adopt, the system integrator needs to interpret the pareto-set. Depending on special requirements to the system she can consider additional information such as system sensitivity to property variations (see section IX) to make a decision.

Figures 10(a), 10(b) and 10(c) show the 2-dimensional pareto-fronts representing the optimal trade-offs between each of the constrained timing properties and the number of parameter changes necessary to achieve the latter.

If we consider, for instance, the path $S2 \rightarrow S5$, we can see from the pareto-front in figure 10(a), that the minimum

number of necessary parameter changes to obtain a working system is 4 (config. #13), corresponding to an delay of 14.46 time units, which is short of the constraint (15 time units).

Increasing the number of allowed parameter changes leads to shorter end-to-end delays. With 5 (config. #8) and 6 (config. #5) parameter changes we can obtain end-to-end delays of 13.84 and 12.99 time units for the path $S2 \rightarrow S5$, respectively. The shortest end-to-end delay for the path $S2 \rightarrow S5$, 12.24 time units, can be achieved with a minimum number of 7 parameter changes.

IX. Sensitivity Analysis of the Integrated System

In this section we determine the sensitivity of the pareto-optimal system configurations presented in Section VIII-B. Figure 11 shows the flexibility of the task execution times and channels communication times. From the set of pareto-optimal configurations we removed those that dominate the other configurations in at least one computed parameter slack. At a closer look we observe that configurations #4 and #7 determine the maximum available slack for most execution times. In general, comparing the results presented in Figure 11 with the results obtained for the two independent subsystems (Figure 7(a)) we observe that the slacks of the execution times of the system tasks have decreased only by a small amount after system integration. The slack of the communication channels has evidently decreased due to the load increase of $CAN1$.

Figure 12 shows the slack values obtained for the hardware resource speed. Again, we selected only those configurations that have a high overall robustness or those which dominate all other configurations in at least one computed parameter slack. Compared to the results presented in Figure 7(b) the only resource with a noticeable smaller slack is the communication bus, $CAN1$. The reason is again the increase of the overall resource utilization after system integration.

Lastly, we determine the available slack corresponding to the task activation periods. Figure 13 shows the values obtained for some configuration selected from the set of pareto-optimal configurations obtained in Section VIII-B. Comparing these results with the results obtained before system integration (Figure 7(c)) we observe that the only periods whose slacks clearly decreased are $S2$, $S3$ and $S9$. Since these periods obviously determine the communication rates of the channels on $CAN1$ and, consequently, automatically the utilization of this bus, and since the overall load on $CAN1$ has increased after subsystem integration, the available headroom of these periods decreased accordingly.

X. Conclusion

Formal models are an ideal basis to determine system properties that are not amenable to simulation, such as system robustness, and they allow rapid design space exploration. Therefore, formal models are considered to play a major role in automotive design in the future.

References

[1] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA — a platform and programming language independent interface for search algorithms. *http://www.tik.ee.ethz.ch/pisa/*.

[2] K. Gresser. An event model for deadline verification of hard real-time systems. In *Proceedings 5th Euromicro Workshop on Real-Time Systems*, pages 118–123, Oulu, Finland, 1993.

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

[4] A. Hamann, M. Jersak, K. Richter, and R. Ernst. Design Space Exploration and System Optimization with SymTA/S - Symbolic Timing Analysis for Systems. In *Proc. of the 25th IEEE Real-Time Systems Symposium (RTSS)*, Lisbon, Portugal, December 2004.

[5] J. C. Palencia and M. G. Harbour. Schedulablilty analysis for tasks with static and dynamic offsets. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid, Spain, 1998.

[6] J. C. Palencia and M. G. Harbour. Exploiting precedence relations in the schedulablilty analysis of distributed real-time systems. In *Proceedings of the 20th Real-Time Systems Symposium (RTSS)*, 1999.

[7] P. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Tenth International Symposium on Hardware/Software Codesign (CODES'02)*, Estes Park, Colorado, USA, May 2002.

[8] R. Racu, M. Jersak, and R. Ernst. Applying sensitivity analysis in real-time distributed systems. In *Proc. of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, San Francisco, California, March 2005.

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

[10] K. Richter and R. Ernst. Event model interfaces for heterogeneous system analysis. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Paris, France, March 2002.

[11] K. Richter, M. Jersak, and R. Ernst. A formal approach to MpSoC performance verification. *IEEE Computer*, 36(4), April 2003.

[12] K. Richter, R. Racu, and R. Ernst. Scheduling analysis integration for heterogeneous multiprocessor SoC. In *Proc. of the 24th IEEE Real-Time Systems Symposium (RTSS)*, Cancun, Mexico, December 2003.

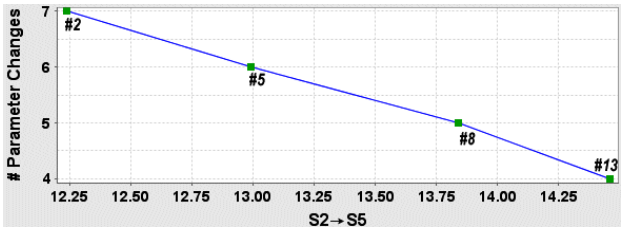
[13] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. Model composition for scheduling analysis in platform design. In *Proc. of the 39th IEEE/ACM Design Automation Conference (DAC)*, New Orleans, USA, June 2002.

[14] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS)*, Geneva, Switzerland, May 2000.

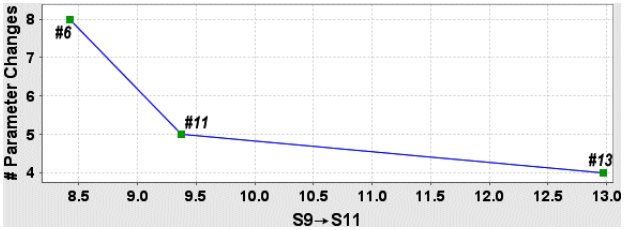
[15] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing & Microprogramming*, 50(2-3):117–134, April 1994.

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

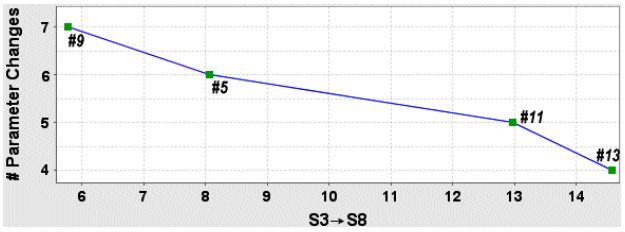
[17] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.



(a) Pareto-Front: #Parameter Changes - $S2 \rightarrow S5$



(b) Pareto-Front: #Parameter Changes - $S9 \rightarrow S11$



(c) Pareto-Front: #Parameter Changes - $S3 \rightarrow S8$

Fig. 10 - OPTIMIZATION RESULTS

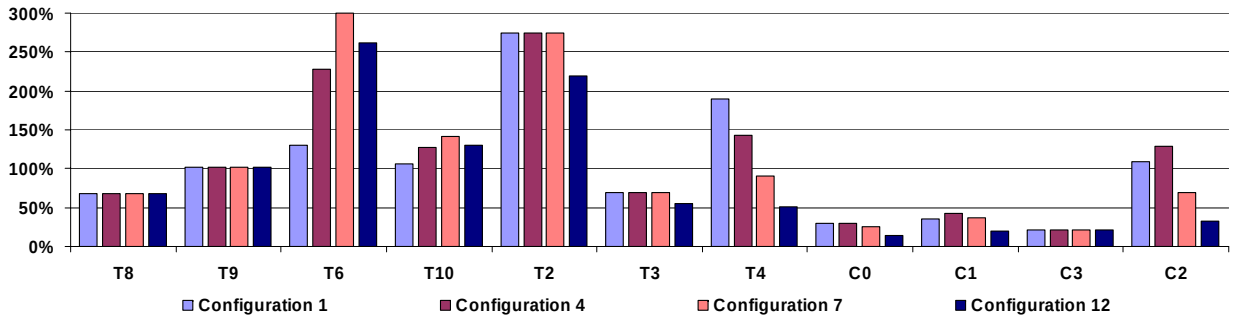


Fig. 11 - WCET FLEXIBILITY

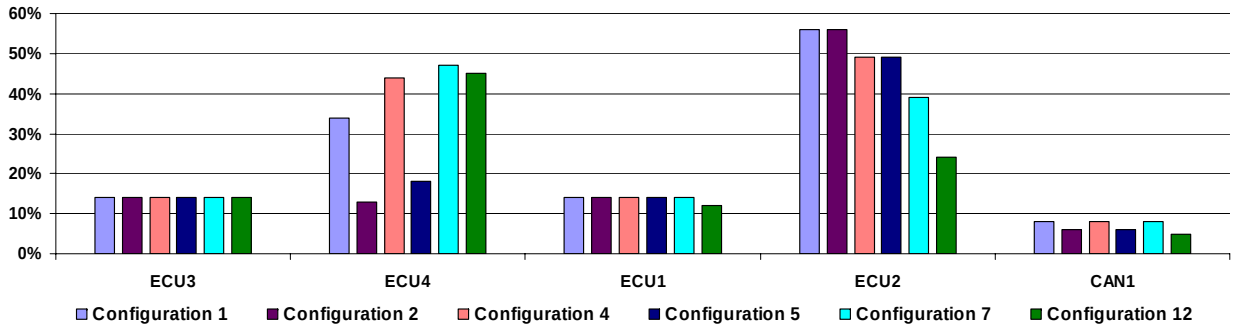


Fig. 12 - RESOURCE SPEED FLEXIBILITY

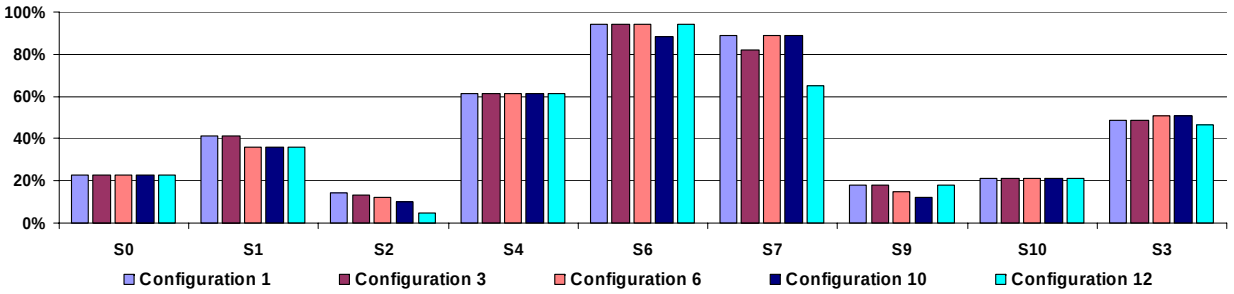


Fig. 13 - ACTIVATION PERIOD FLEXIBILITY