Real-Time Syst (2006) 33: 101–137
DOI 10.1007/s11241-006-6884-x

A framework for modular analysis and exploration of heterogeneous embedded systems

Arne Hamann · Marek Jersak · Kai Richter · Rolf Ernst

Abstract The increasing complexity of heterogeneous systems-on-chip, SoC, and distributed embedded systems makes system optimization and exploration a challenging task. Ideally, a designer would try all possible system configurations and choose the best one regarding specific system requirements. Unfortunately, such an approach is not possible because of the tremendous number of design parameters with sophisticated effects on system properties. Consequently, good search techniques are needed to find design alternatives that best meet constraints and cost criteria. In this paper, we present a compositional design space exploration framework for system optimization and exploration using SymTA/S, a software tool for formal performance analysis. In contrast to many previous approaches pursuing closed automated exploration strategies over large sets of system parameters, our approach allows the designer to effectively control the exploration process to quickly find good design alternatives. An important aspect and key novelty of our approach is system optimization with traffic shaping.

Keywords Real-time · Embedded · Distributed systems · System-on-chip · Performance verification · Scheduling analysis · Compositional · Optimization · Design space exploration · Traffic shaping · Evolutionary algorithms

1. Introduction

A major challenge of heterogeneous system optimization is the lack of coherent models and systematic search techniques. For this reason it is important to evaluate a large number of architectures and implementation alternatives. Ideally, the designer would try all possible alternatives and choose the best regarding specific system requirements. Unfortunately, this is not possible because the high number of design parameters in complex systems leads to a very large design-space, prohibiting an exhaustive search. Consequently, good exploration techniques are needed to find optimal, or at least good, design alternatives.

A. Hamann (✉) · M. Jersak · K. Richter · R. Ernst
Institute of Computer and Communication Network Engineering, Technical University of Braunschweig, D-38106 Braunschweig, Germany
{hamann, jersak, richter, ernst}@ida.ing.tu-bs.de

Manual design space exploration heavily reduces design productivity. It is highly desirable to automate at least part of the process. Of course, even automatic exploration cannot search the whole design space in reasonable time. Therefore, it is important to find an appropriate sub search space containing good solutions. Restriction of the search space to crucial system parameters is necessary to allow an efficient search for good design alternatives.

In this paper we present a framework for design space exploration and system optimization using SymTA/S (Hamann et al., http://www.symta.org/), a software tool for formal performance analysis. In contrast to previous approaches, our framework does not perform a closed global exploration over a large set of design parameters. Instead it provides the designer with the possibility to perform several successive exploration steps, modifying (i.e. extending or restricting) the search space in every step as a reaction to the obtained results. This user-controlled exploration approach allows the designer to guide the exploration process and provides him insight to system-level performance dependencies. Based on this knowledge she can identify step-by-step interesting design sub-spaces, worthy to be searched in-depth or even completely. In order to enable such an user-controlled exploration approach, our framework utilizes a compositional encoding of the search space and allows the dynamic modification of the search space during exploration without losing already obtained results.

An a priori global exploration does not permit such a flexibility and neglects the structure of the design space, giving the designer no possibility to modify and select the exploration strategy. In the worst-case, when the composition of the design space is unfavorable, this can lead to non-satisfying results with no possibility for the designer to intervene. In many approaches the only possibility for the designer in such a case consists in restarting the exploration, hoping for better results.

An important aspect and one key novelty of our design space exploration approach is the optimization of component interactions and dependencies with traffic shaping. Traffic shaping weakens functional and non-functional performance dependencies between components in the system and allows finding working system configurations, which are not possible without traffic modulation. Consequently, traffic shaping can broaden the solution space considerably leading to increased exploration efficiency.

The remainder of this paper is structured as follows. In Section 2 we first give an introduction into the formal core of SymTA/S, including the application model, the utilized standard event models, the compositional analysis methodology and the concept of event stream adaptation (i.e. traffic shaping). In Section 3 we then give a survey of related work in the domain of exploration and optimization of heterogeneous embedded systems. Afterwards, in Section 4, we explain the main concepts of our design space exploration approach. These are the compositional encoding of the search space, the component interaction optimization with traffic shaping, and the user-controlled exploration strategy. In Section 5 we then present the concrete realization of our exploration framework based on multi-objective evolutionary exploration techniques, including encoding and exploration strategies for different parts of the search space, optimization objectives and the design space exploration loop. Afterwards, in Section 6, we describe a synthetical SoC example and perform several exploration steps in order to optimize its performance (Section 7). Finally, in Section 8, we perform experiments to evaluate the efficiency of our exploration approach.

2. The SymTA/S approach

SymTA/S (Hamann et al., http://www.symta.org/) is a formal system-level performance and timing analysis tool for heterogeneous SoCs and distributed systems. The application model

of SymTA/S is described in Section 2.1. The core of SymTA/S is a technique to couple local scheduling analysis algorithms using event streams (Richter and Ernst, 2002; Richter et al., 2002). 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 2.2. The analysis composition using event streams is described in Section 2.3. A second key property of the SymTA/S compositional approach is the ability to adapt the possible timing of events in an event stream. The event stream adaptation concept is described in Section 2.4.

2.1. 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.

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 2.2.

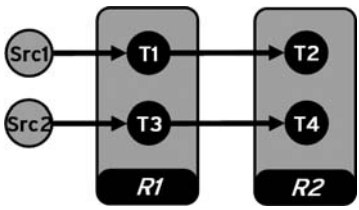


Fig. 1 System modeled with SymTA/S

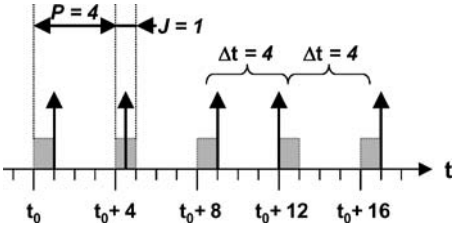


Fig. 2 Example of an event stream that satisfies the event model ($\mathcal{P} = 4,\ \mathcal{J} = 1$)

2.2. 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.

Periodic with jitter event models are well suited to describe generally periodic event streams, which often occur in control, communication and multimedia systems (Richter et al., 2003a). 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 (Richter et al., 2003a). 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 Richter et al. (2003b).

2.3. Analysis composition

In the SymTA/S compositional performance analysis methodology (Richter et al., 2003a,b), 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.

2.3.1. 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 (Eq. (1)).

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

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^-.

2.3.2. *Analysis composition using standard event models*

In the following, the compositional analysis approach is explained using the system example in Fig. 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 2.3.1). 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 Fig. 3.

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 (Richter, 2004). 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 (Tindell and Clark, 1994). 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 (Richter et al., 2003b). In the first phase local scheduling analysis is performed for each resource and output event models are calculated (Section 2.3.1). 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

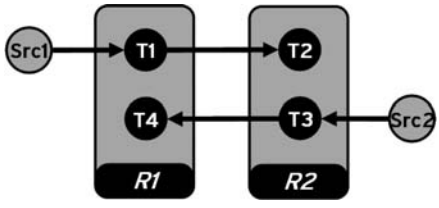


Fig. 3 Example of a system with cyclic scheduling dependency

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.

2.4. Event stream adaptation

A key property of the SymTA/S compositional performance analysis approach is the ability to adapt the possible timing of events in an event stream (expressed through the adaptation of an event model (Richter et al., 2003b)). There are several reasons to do this. It may be that a scheduler or a scheduling analysis for a particular component requires certain event stream properties. For example, rate-monotonic scheduling and analysis (Liu and Layland, 1973) require strictly periodic task activation. Alternatively, an integrated IP component may require certain event stream properties. External system outputs may also impose event model constraints, e. g. a minimum distance d^- between output events or a maximum acceptable jitter. Such a constraint may be the result of a performance contract with an external subsystem (Tindell et al., 2003).

Event stream adaptation can also be done for the sole purpose of *traffic shaping* (Richter et al., 2003b). Traffic shaping can be used e. g. to reduce transient load peaks, in order to obtain more regular system behavior. Practically, event model *adaptation* is distinguished from event model *shaping* in SymTA/S (Hamann et al., http://www.symta.org/). Adaptation is required to satisfy an event model constraint, while shaping is voluntary to obtain more regular system behavior. Two types of event adaptation functions (EAF) are currently used in SymTA/S: *periodic* EAFs, producing periodic event streams from *periodic with jitter* input event streams, and d^--EAFs enforcing a minimum distance between output events.

In the following we will briefly explain the concept of traffic shaping using d^--EAFs. Compared to full synchronization, d^--EAFs provide promising peak load reduction and load balancing capabilities with smaller buffers and delays. Larger d^- values result in more balanced system load and better schedulability, while they increase delays and buffering requirements along task chains (or paths).

2.4.1. Traffic shaping with d^--EAFs

Scheduling and data dependent behavior induce jitter to the input-output timing of processes and communication (Richter et al., 2003b). Such jitters accumulate in the system and can lead to event bursts. Both effects increase timing uncertainty and worst-case peak load.

Such peak loads caused by bursty streams can be controlled by modulating the maximum number of events per time, called traffic shaping. Traffic shaping reduces the impact of an event stream on other streams at the cost of a potentially increased latency of the controlled stream. The shaping effects are rather complex and require special modeling considerations that will be explained in the following.

A bursty event stream is defined by three parameters, an average period T, a maximum allowed jitter J, and a minimum event distance d^- during bursts. As a popular measure of system load in scheduling analysis, the $\eta^+(\Delta t)$ function determines the maximum number of events η^+ for a given interval of time Δt. Small time intervals are dominated by bursty behavior, where the system load is only limited by the minimum event distance d^-. Larger observation intervals reveal the generally periodic nature of the event stream. The *arrival curve* (Thiele et al., 2000) in Fig. 4 illustrates the two different regions. Both regions, i.e. the periodic region and the region dominated by the burst, can be separately described with

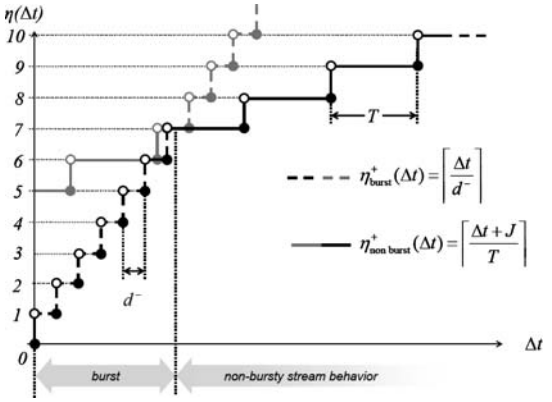


Fig. 4 Event arrival curve of incoming event stream

equations. The η^+ function of the stream, illustrated by the black curve in Fig. 4, is the minimum of them:

$$\eta^+_{\text{in}}(\Delta t) = \min\left(\left\lceil \frac{\Delta t}{d^-} \right\rceil, \left\lceil \frac{\Delta t + J}{T} \right\rceil\right). \tag{2}$$

Using time-out buffers, designers can deliberately enforce an additional bound on the minimum event distances. Such time-out buffers represent *traffic shapers* that are inserted in the design between two application components. The time-out mechanism buffers incoming events such that no two successive events are released earlier in time than d^-_{timeout}.

According to the extended real-time calculus approach of Thiele et al. (2001), the shaper defines a sporadic upper-bound *service curve* (Thiele et al., 2000) with $\eta^+_{\text{timeout}}(\Delta t) = \lceil \frac{\Delta t}{d^-_{\text{timeout}}} \rceil$. The shapers output arrival curve can be calculated from both, input arrival curve $\eta^+_{\text{in}}(\Delta t)$ and shaper service curve $\eta^+_{\text{timeout}}(\Delta t)$. In case of traffic shapers the usually complex real-time calculus equations can be easily reduced to

$$\begin{aligned}\eta^+_{\text{shaped}}(\Delta t) &= \min\left(\eta^+_{\text{timeout}}(\Delta t), \eta^+_{\text{in}}(\Delta t)\right) \\ &= \min\left(\left\lceil \frac{\Delta t}{d^-_{\text{timeout}}} \right\rceil, \left\lceil \frac{\Delta t}{d^-_{\text{in}}} \right\rceil, \left\lceil \frac{\Delta t + J}{T} \right\rceil\right).\end{aligned}$$

The larger value of d^-_{in} and d^-_{timeout} will dominate the other, and we can further reduce the $\eta^+_{\text{shaped}}(\Delta t)$ function to the η^+ function of an event stream with burst as introduced by Eq. (2). In case of $d^-_{\text{timeout}} \leq d^-_{\text{in}}$, the shaper does not actually represent an additional constraint. In other words, the shaper is "inactive", no events are buffered and the output arrival curve equals the input arrival curve.

Obviously more interesting is the case of $d^-_{\text{timeout}} > d^-_{\text{in}}$. Input events are buffered and the shaper "flattens" the burst slope of the output arrival curve according to d^-_{timeout}: $\eta^+_{\text{shaped}}(\Delta t) = \min(\lceil \frac{\Delta t}{d^-_{\text{timeout}}} \rceil, \lceil \frac{\Delta t + J}{T} \rceil)$. Figure 5 illustrates this behavior. The arrival curve with a minimum distance of d^-_{in} is above the service curve with a minimum event distance of d^-_{timeout}. The block arrows indicate buffering. Thiele et. al. already recognized that the vertical distance between the arrival and the service curve captures the so called backlog (Thiele et al., 2000), i. e. the number of buffered events at a given point in time: $backlog(\Delta t) = \eta^+_{\text{in}}(\Delta t) - \eta^+_{\text{timeout}}(\Delta t)$.

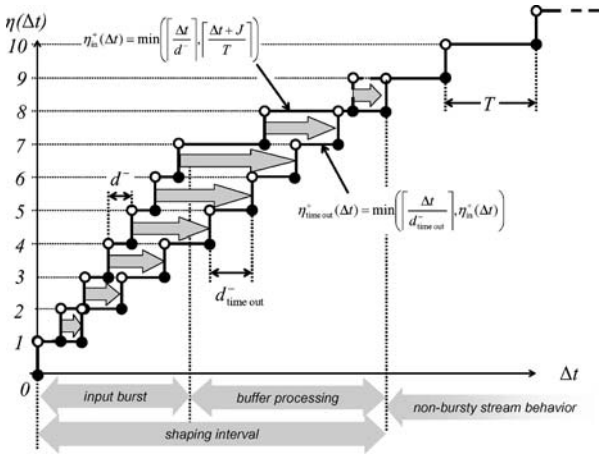


Fig. 5 Event arrival curve of output event stream

The horizontal distance between the curves, i. e. the length of the arrows in the figure, represents the delay of the corresponding event. The calculations are slightly more sophisticated than the *backlog*, although the specialties of traffic shaping reduce the complexity of the general real-time calculus theory (Thiele et al., 2000). We recently introduced another function $\delta^-(n)$ that determines the minimum distance between n successive events (Richter, 2004). Roughly speaking, $\delta^-(n)$ is the inverse of $\eta^+(\Delta t)$ since it returns the earliest time Δt at which the nth event ($n \geq 2$) can arrive after the first one. For the bursty arrival curve and the sporadic service curve, these are given by $\delta^-_{\text{in}}(n) = \max((n-1)d^-_{\text{in}}, (n-1)T - J)$ and $\delta^-_{\text{timeout}}(n) = (n-1)d^-_{\text{timeout}}$. Hence, the delay is given by: $delay(n) = \delta^-_{\text{timeout}}(n) - \delta^-_{\text{in}}(n)$.

The sought-after maxima of $backlog_{\max} = \max_{\Delta t>0} backlog(\Delta t)$ and $delay_{\max} = \max_{n\geq 2} delay(n)$ can be calculated through linearization of the discrete η^+ and δ^- functions. Details can be found in Richter (2004). For this paper, the following qualitative explanation shall be sufficient. It should not surprise that the worst-case buffering and delay situation appears at the end of the input burst. At that time, *the most events* are stored "waiting" for being processed until the buffer is empty and the behavior returns to "non-bursty". And clearly the last event of the input burst has to *wait longest*.

3. Related work

There is a large body of work in the area of design space exploration and optimization of heterogeneous MpSoC and distributed systems. In the following we give an overview of approaches for the optimization of different system parameters as well as frameworks allowing to explore given systems at different levels of abstraction.

The approach described in Thiele et al. (2002) uses an analysis technique, called the *real-time calculus* (Thiele et al., 2000), to estimate end-to-end packet delays and queuing memory in network processor architectures. Based on this analysis technique a measure is defined to characterize the performance of such architectures under different usage scenarios. By means of design space exploration pareto-optimal architectures are searched trading good performance under several usage scenarios versus cost. The exploration is performed using multi-objective evolutionary algorithms running a closed optimization over all relevant search parameters, including type and number of resources in the target architecture and the mapping of the tasks to the resources for each considered scenario

along with appropriate priority assignments. The presented results in a case study show the efficiency of the approach. Unlike the approach in this paper, which works on configurable chromosome strings, and thus allows to interactively constrain the search space, the exploration in Thiele et al. (2002) covers all dimensions of the search space in a closed automated approach. Constraining the design space is often required in practical designs with a large design space. Also, the approach presented in this paper covers a couple of important additional design features such as time slot assignment and traffic shaping.

In Maxiaguine and Künzli (2004) the authors treat the reverse problem. Instead of determining worst-case buffer requirements and output stream properties for given input streams and scheduling policies, the authors search for the input stream rates that can be supported by a given stream processing architecture without violating on-chip buffer constraints. The authors propose the integration of this technique into a tool for automated design space exploration for fast performance evaluation of different stream processing architectures.

Garcia and Harbour (1995) presents a heuristic algorithm for priority assignments in distributed hard real-time systems. The algorithm tries to find priority assignments on each resource so that all global end-to-end constraints in the system are satisfied. Therefore, global deadlines are iteratively decomposed into artificial local deadlines, which are then used to assign deadline monotonic priorities on each resource. Once a working system configuration is found the algorithm tries to find better solutions. Thereby, the quality of a solution is expressed by a so-called scheduling index, which is defined as a function of the difference between the worst-case end-to-end delays and the global deadlines for each constraint (lateness). The approach is limited to the optimization of end-to-end latencies. Also it does not determine different (pareto-optimal) design trade-offs between latencies along multiple paths. The utilized one dimensional metric offers only a narrowed view on the quality of a system configuration.

The approach in Pop et al. (2004) focuses on system optimization in the domain of multi-cluster embedded systems interconnected via gateways. The authors present a heuristic approach to map applications, modeled as sets of directed acyclic graphs, onto given architectures consisting of event triggered (ET) and time triggered (TT) clusters interconnected via TDMA and priority scheduled communication resources. Thereby, each graph is associated with an activating period and an end-to-end deadline, which has to be smaller than the period. The optimization objective is to find a system configuration satisfying all graph deadlines. The method proposed is a closed optimization over all parameters in the system, i.e. the partitioning of the application to the ET and TT clusters, the mapping of the tasks on resources within the clusters and the optimization of the bus access of the TDMA and priority scheduled interconnecting buses. In order to break down the complexity of the problem, the authors segment the optimization into three steps. First an initial configuration of all system parameters is generated. If the system configuration is not working an iterative heuristic is applied trying to optimize the partitioning and the mapping of the application. In the third step, as a last measure to reach schedulability, a heuristic optimizing the bus access is applied. In experiments the authors show that their optimization strategy is capable of finding working configurations for more than 80% of randomly generated applications with up to 250 tasks. Due to the restriction that the deadlines of a task graph must be smaller or equal than its activating period, heuristics are adequate to tackle the given optimization problem, since only few complicated scheduling effects can occur under the given application model. However, the situation is different if we allow deadlines greater than the period and activation jitter. Also the optimization approach does not reveal design trade-offs between different system constraints for the case that multiple solutions exist.

Givargis and Vahid (2002) describes the *Platune* framework allowing performance and power tuning of a specific parameterized SoC platform. For a given application to be mapped on the target SoC, *Platune* determines all sets of architectural parameter values representing pareto-optimal solutions regarding power and performance. In order to speed up the exploration process running on a large set of design parameters, the authors introduce a parameter dependency model, which is used to cluster the search space into independent parts. In the first step of the exploration the authors use an exhaustive algorithm to determine all local pareto-optimal configurations for each of these independent parts. In the second exploration step, the local pareto-optima are then merged iteratively to obtain pareto-optimal solutions for the entire configuration space. In experiments the authors show the efficiency of their approach for a given SoC platform, which can be clustered into small independent parts. However, parameter dependencies are target platform specific and might be difficult to determine in the general case. Also the system parameters of a given target architecture might offer only few independencies prohibiting an efficient clustering of the search space. In the latter case, exhaustive exploration becomes infeasible and needs to be replaced by more sophisticated exploration methods.

The *Spacewalker* (Snider, 2001), part of the *PICO* project from HP Labs, searches for pareto-optimal embedded systems for given applications. The search space is explored using a hierarchical divide-and-conquer approach. In the first step different subsystem are explored independently. From the sets of obtained pareto-optimal subsystems global systems are constructed and evaluated in the second step. This hierarchical exploration approach seems to work well for the architecture presented in the paper. However, for performance dependent subsystems the combination of local pareto-optima rarely leads to global pareto-optima.

In Dick and Jha (1998) an approach for the co-synthesis problem, called *MOGAC*, is presented. For a given embedded system specification the authors use a multi-objective genetic algorithm to determine an optimal system architecture, i.e. hardware/software processing elements and communication links, as well as the mapping of the application onto this architecture including a non-preemptive static schedule for each processing element. The application model used is similar to the one used in Pop et al. (2004), i.e. DAGs with associated periods and deadlines. Optimization objectives during exploration are system price, power consumption and processing time. The approach represents an interesting method addressing the co-synthesis problem with stochastic search techniques. However, the authors perform a closed exploration over all possible system parameters without providing a methodology to reduce the huge search space or to control the exploration. Consequently, the approach might quickly reach its limits as system size and complexity increases. Another drawback of the *MOGAC* approach is the utilized application model assuming non-preemptive static scheduling on the processing elements restricting its applicability to many real-world examples.

The *Sesame* framework, part of the *Artemis* (Pimentel et al., 1995) project, is used in Erbas et al. (2003) to tackle the mapping decision problem of complex applications onto heterogeneous embedded system architectures. The authors use evolutionary exploration techniques to search for solutions, which are pareto-optimal regarding maximum processing time, power consumption and system price. These solutions are then input to a simulation framework for further evaluation. The *Sesame* approach differs form the other approaches in the sense that it uses Kahn process networks to model applications rather than task graphs and event stream models. Also it does not target system synthesis and does not create schedules as exploration result. An interesting aspect in this approach is the explicit distinction of working and non-working pareto-optimal design alternatives, preventing the

possible convergence of the exploration towards a set of pareto-optimal infeasible solutions, which could happen, for instance, using MOGAC.

4. Compositional design space exploration approach

In the following sections the main concepts of our compositional exploration approach, which is based on multi-objective evolutionary exploration techniques (Deb, 2001), are presented. Note that realization details are presented in Section 5.

First, we discuss and motivate the compositional encoding of the search space used in our exploration framework (Section 4.1). Afterwards, we will explain the concept of optimizing component dependencies and interactions with traffic shaping and show by means of a small example, that it can broaden the solution space considerably (Section 4.2). Finally, we will present our user-controlled exploration approach allowing the designer to effectively control the exploration to quickly cover large search spaces (Section 4.3).

4.1. Compositional search space encoding

Figure 6 shows the compositional search space encoding concept of our exploration framework. According to the underlying compositional scheduling analysis described in Section 2 the system is seen component wise for system exploration. During exploration, components and event streams belonging to the search space are encoded as separate *chromosomes*. The designer can interactively combine arbitrary chromosomes to compose the desired search space. In the example, for instance, the search space consists of the scheduling parameters on the resources $CPU1$, $CPU2$ and BUS as well as traffic shaping on the event stream connecting $T4$ and $C2$. Note that during exploration, sets of concrete chromosome instances (phenotypes) represent specific system configurations called *individuals*.

In addition to the encoding of the represented search space part, a chromosome defines the local exploration strategy for the underlying component. More precisely, it possesses evolutionary variation operators (mutation and crossover) for combination with other chromosomes of its type. During exploration, these operators are applied chromosome-wise to create new candidate system configurations.

We have chosen to split the overall system exploration into several entities, i.e. chromosomes, controlling the exploration on local components rather than performing a closed exploration over all search parameters for several reasons.

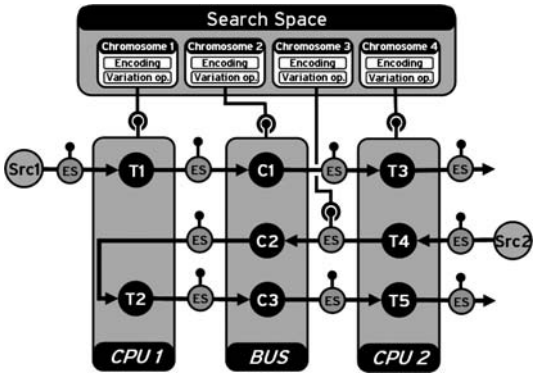


Fig. 6 System exploration example

First, it is easier to establish a constructively correct encoding on a small subset of design decisions. Consequently, the compositional encoding scheme ensures that all chromosome values correspond to valid decisions such that any chromosome variation is constructively valid. This improves the exploration process as it greatly reduces the effort of checking a generated design for validity. It allows using the analysis engine of SymTA/S which requires correct design parameters to apply analysis (e.g. sum of time slots no longer than the period, legal priority setting, etc.).

Secondly, the compositional view on the exploration allows to integrate new component analyses into system level exploration by simply adding a corresponding chromosome to the exploration framework. This is important since SymTA/S is not limited to a fixed set of component analyses with a common application model, but allows coupling arbitrary local component analyses to system level analysis. Current component analyses in SymTA/S use the application model presented in Section 2.1 and differ only in the utilized scheduling policy (i.e. static priority preemptive, TDMA, EDF, etc.). However, SymTA/S is also capable of coupling analysis techniques based on completely different application models like for instance Kahn process networks.

Thirdly, the compositional encoding scheme leads to high flexibility of the exploration framework. Each chromosome allows to define the specific encoding and exploration strategy for the search space part it represents. In the simplest case, binary encoding and binary variation operators like single-point crossover can be utilized resulting in uniform search over all possible configurations. However, chromosomes can be encoded in more intuitive ways using arbitrary data structures and problem-aware variation operators. In Hamann and Ernst (2005) we have shown for the case of TDMA time slot optimization that such tailored chromosomes can significantly increase exploration efficiency.

Finally, the compositional encoding approach allows adding and removing system parameters to design space exploration, even dynamically, which we exploit in our approach allowing the designer to effectively guide the exploration process to quickly find interesting design alternatives (see Section 4.3).

4.2. Component interaction optimization

One key property of design space exploration in SymTA/S is the optimization of component dependencies and interactions using traffic shaping. Like explained in Section 2, components in SymTA/S are connected via event streams. The manipulation of event streams via event stream adaptation represents an interesting optimization possibility as it breaks open, resp. weakens, performance dependencies between connected components.

An extreme measure would be to use *periodic* EAFs on every event stream in the system, enforcing strictly periodic event models between all components. Clearly, this completely decouples all performance dependencies between components, reducing the global optimization problem to local optimizations on the single components with respect to global system constraints. However, since *periodic* EAFs induce high latency on the underlying event streams and require large buffers, such a measure would surely lead to unacceptably high end-to-end latencies and high buffering costs in the resulting system. Therefore, mainly d^--EAF are useful for system optimization and exploration. Compared to *periodic* EAFs providing full synchronization, they allow to trade the grade of component performance decoupling and peak load reduction versus increased delays and buffer sizes along the shaped event stream.

To illustrate the benefit of controlling the component interaction with traffic shaping, we consider the example system given in Figure 6. It consists of two CPUs connected via a BUS,

Table 1 System parameter

(a)Core execution times

Computation task	Core execution time
T1	[20, 20]
T2	[40, 40]
T3	[30, 30]
T4	[25, 25]
T5	[25, 25]
Communication task	**Core communication time**
C1	[10, 10]
C2	[20, 20]
C3	[15, 15]

(b)Input event models

Input	Event model
$Src1$	periodic, $\mathcal{P}_{Src1} = 100$
$Src2$	periodic, $\mathcal{P}_{Src2} = 100$, $\mathcal{J}_{Src2} = 400$

Table 2 Path latency constraints

Constraint #	Path	Maximum latency
1	$Src1 \rightarrow T3$	800
2	$Src2 \rightarrow T5$	600

all scheduled according to the static priority preemptive policy. The best-case and worst-case execution times and the external activating event models are given in the Table 1(a) and (b). In order to function correctly, the system has to satisfy the path latency constraints listed in Table 2.

We can easily verify that there exists no priority assignment leading to a functioning system satisfying the path constraints. However, if we add traffic shaping to the search space we are able to find working system configurations.

Let us, for instance, consider the following priority assignment:

- *CPU1*: $T2 > T1$
- *BUS*: $C2 > C3 > C1$
- *CPU2*: $T4 > T5 > T3$

Without traffic shaping this system configuration yields the following end-to-end latencies: 1630 time units for the path $Src1 \rightarrow T3$ and 680 time units for the path $Src2 \rightarrow T5$. However, we can improve the system behavior tremendously by performing traffic shaping with d^--EAFs.

Let us take a look at the worst-case response time of $C1$ with and without traffic shaping at the output of $T4$. Figure 7(a) visualizes the worst-case scheduling scenario of $C1$ without traffic shaping in the system. Figure 7(b) shows the improved worst-case scheduling scenario of $C1$ with a d^--EAF at the output of $T4$, extending the minimum distance of successive events from 25 to 50 time units. Note that the given activating event models for $C1$, $C2$ and $C3$ are analysis results of the iterative compositional scheduling analysis performed by SymTA/S (see Section 2).

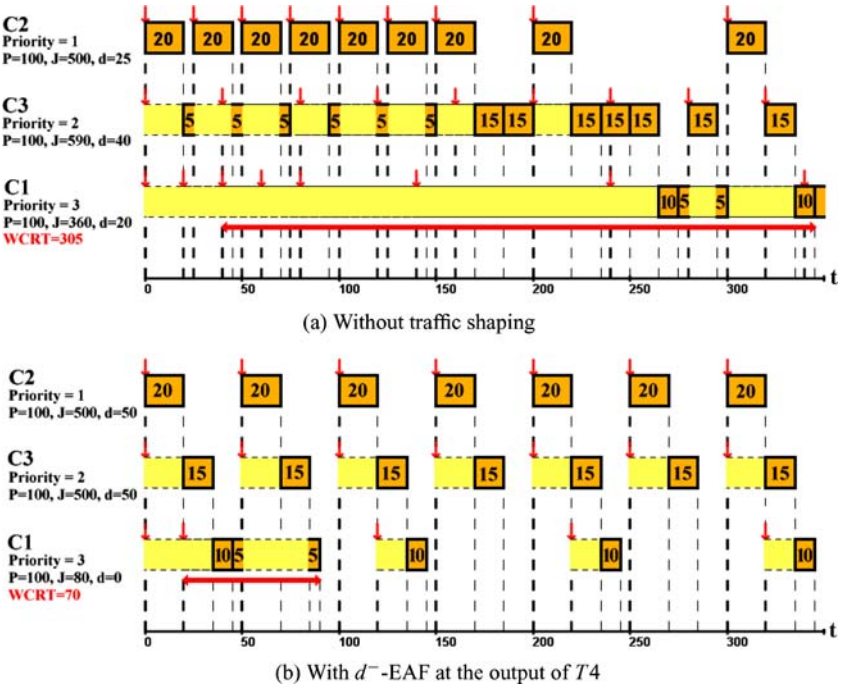


Fig. 7 Worst-case scheduling scenarios $C1$

We observe that the inserted d^--EAF leads to the reduction of the worst-case response time of $C1$ from 305 to 70 time units. This is due to two effects. First of all, the d^--EAF relaxes the activation burst of $C2$, leading to more freedom for the lower priority tasks $C1$ and $C3$ to execute. This results in less preemption, and thus earlier completion for $C1$ and $C3$. Secondly, we observe smaller activation jitters for $C1$ and $C3$. The reason for that is, that the positive effect of the d^--EAF does not only lead to shorter worst-case response times on the *BUS*, but is also propagated through improved output event models, i.e. containing less response time jitter, to the neighboring components. In our case, for instance, less activation jitter is produced for $T2$ on *CPU1*. The lower priority task $T1$ is profiting from the reduced activation jitter of $T2$ in terms of a shorter worst-case response time, and thus less output jitter. In the considered case the output jitter of $T1$, and thus the input jitter of $C1$, was reduced from 360 to 80 time units due to the effects of the inserted d^--EAF at the output of $T4$.

Figure 8 visualizes the global impact, expressed by the end-to-end delays along the paths $Src1 \rightarrow T3$ and $Src2 \rightarrow T5$, of d^--EAFs at the output of $T4$ enforcing different d^- values. Possible d^- values lie between 25 and 100 time units, given by the best-case execution time and the activating period of $T4$, respectively.

We observe that the latency of the path $Src1 \rightarrow T3$ falls with growing d^-. This is not surprising, since all tasks along the path have the lowest priority on their resources, and are thus profiting highly from the inserted traffic shaper. For the path $Src2 \rightarrow T5$ the situation looks different. We observe that first its latency falls, reaching a minimum for d^- values between 40 and 45 time units. Afterwards, its latency increases again. The reason for this behavior is, that the traffic shaper does not only improve the systems timing behavior, but also introduces latency on the path $Src2 \rightarrow T5$. However, up to a d^- value of 69 time units the positive effect of the traffic shaper dominates the introduced

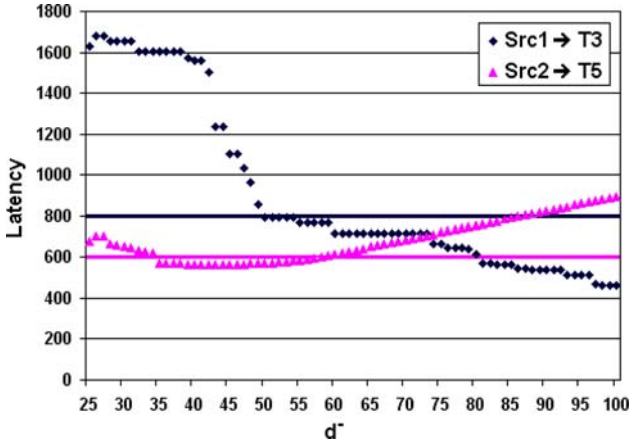


Fig. 8 System behavior with d^--EAFs at the outputof $T4$

latency, leading to smaller latencies compared to the original configuration without traffic modulation. Altogether, d^--EAF enforcing d^- values between 50 and 57 time units lead to working system configurations satisfying the given end-to-end constraints.

For the discussed example system the global impact of the inserted d^--EAF is rather high, since the system behavior is dominated by the large input burst generated by $Src2$. In the general case optimization through traffic shaping is not applicable in such a straight forward manner.

However, the exploration of a small but realistic example system in Section 7 shows, that traffic shaping can lead to the discovery of interesting design alternatives, which are not possible without traffic modulation. Additionally, experiments with synthetical systems in Section 8.1 indicate, that traffic shaping can broaden considerably the solution space leading to decreased exploration time to find working system configurations.

4.3. User-controlled exploration strategy

In Section 4.2 we have seen how component interactions and dependencies can be optimized using traffic shaping within the compositional exploration methodology of SymTA/S. Traffic shaping improves the system behavior by weakening performance dependencies between components and reducing the global impact of transient load peaks. Consequently, the system becomes more predictable through traffic shaping, and thus easier to optimize and explore.

However, traffic shaping might not be sufficient as a control mechanism for an efficient design space exploration in SymTA/S. Especially when facing large systems with a multitude of parameters, design space exploration can hardly cover the complete search space in adequate time, even with efficient stochastic search techniques. Consequently, it is crucial to find appropriate sub search spaces containing good solutions.

The idea to restrict the search space to speed up exploration is nothing new and some previous approaches contain techniques to do so. Common techniques try to automatically partition the entire search space into (independent) parts and perform a hierarchical exploration, i.e. local exploration on single components and subsequent recombination of the locally pareto-optimal solutions (Givargis and Vahid, 2002). Limitations of such approaches

include, that the search space might contain only few independencies and the difficulty for the designer to identify these without further aid. In other words, the exploration needs information, which the designer wants to obtain by means of exploration. Consequently, dynamic parameter dependencies are often heuristically ignored (Pop et al., 2004; Snider, 2001), which might lead to the incapacity of the underlying exploration algorithm to find good solutions to the optimization problem.

The compositional exploration approach in SymTA/S pursues another strategy to increase exploration efficiency. Instead of performing a closed automated exploration over all system parameters or taking a priori heuristic assumption about the structure of the search space, the control over the search process is transferred to the designer. Thereby, the exploration concept consist in performing several successive exploration steps with modification of the search space in every step as a reaction to previously obtained results. This concept enables the designer to identify step-by-step interesting design sub-spaces, worthy to be searched in-depth or even completely. A closed global exploration (Dick and Jha, 1998; Thiele et al., 2002) does not permit such a flexibility and neglects the structure of the design space, giving the designer no possibility to modify and select the exploration strategy. For large search spaces this can easily lead to the incapacity of the design space exploration to find working system configurations in adequate time.

Dynamic modifiability of the search space during exploration without losing already obtained results is key to this user-controlled design space exploration approach. Search space modification consists in adding and/or removing chromosomes to/from the search space. Both operation can be easily performed on system configurations encoded according to the compositional encoding scheme. In case of search space modification, pareto-optimal configurations of the currently running exploration step are used as starting point for the exploration step with the modified search space.

The following operations are performed in case that the search space of a running design space exploration is modified by the designer:

1. Pause exploration
2. Discard non pareto-optimal configurations
3. Adapt remaining configurations
4. Reevaluate remaining configurations
5. Continue exploration

After modification of the search space by the designer the exploration is paused (step 1) and all non-pareto optimal configurations are discarded from the set of currently considered system configurations (step 2). Note that this is the default behavior of our exploration strategy. Alternatively, the designer may choose keeping arbitrary system configurations. Afterwards, the remaining system configurations need to be adapted (step 3). For the case that the search space is extended, each remaining configuration must be complemented with parameters for the added part of the search space. By default our exploration strategy initializes the added part of the search space with (valid) random parameters. For the opposite case, i.e. restriction of the search space, the designer must choose one configuration as common basis for the removed search space part for further exploration. After adaptation, the remaining configurations are reevaluated (step 4), i.e. scheduling analysis is performed and performance metrics (optimization objectives) are recalculated. Once this is done the exploration can be continued with the modified search space (step 5).

Figure 9 shows an example exploration with two search space modifications.

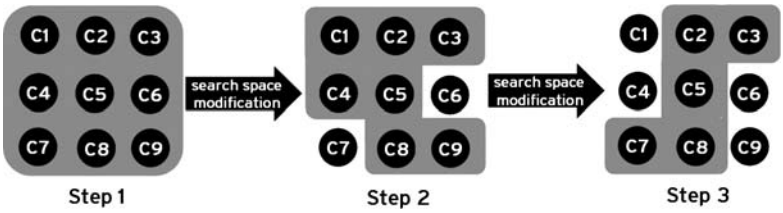


Fig. 9 Example exploration with search space modifications

The first step of the exploration in the example is performed including the whole search space represented by the chromosomes $C1$ to $C9$. Including the whole search space in the first exploration step is advisable, since its starting point usually consists of random system configurations representing poor designs in most cases. However, once design space exploration has run for a while on the whole search space and successively begins to find better system configurations, the designer might want to narrow the search space in order to speed up the exploration process. We assume that in the given example the designer observed, that the search space part represented by the chromosomes $C6$ and $C7$ showed only little differences for many pareto-optimal system configurations obtained in the first exploration step. Consequently, she chooses to fix them and to start a second exploration step with the resulting reduced search space. After analyzing the obtained results of the second exploration step, the designer decides to fix the chromosomes $C1$, $C4$ and $C9$ for similar reasons as before. However, she also decides to reinsert the chromosome $C7$ into the search space. Reasons for this decision might be, for instance, the observation that global constraints strongly influenced by the system parameters represented by $C7$ are violated in many obtained system configurations.

Of course, the search space modifications performed in the above described exploration example represent heuristic assumptions based on partial knowledge of the search space. However, for the case of search space restriction, the designer can always reverse his decisions if exploration results are not satisfying or if he discovers that important system parameter dependencies are neglected.

Experiments with large synthetical example systems show the efficiency of the user-controlled exploration approach (see Section 8).

5. Realization of the SymTA/S exploration framework

In this section we will give a survey of the realization of the compositional design space exploration framework in SymTA/S, which is based on evolutionary exploration techniques.

We will first describe several specific chromosomes representing sets of system parameters with according exploration strategies, which can be composed by the designer to precisely define the search space (Section 5.1). We will then introduce some example metrics expressing desired or undesired system properties used as optimization objectives during exploration (Section 5.2). Finally, we will explain in detail the iterative design space exploration loop performed by our framework (Section 5.3).

5.1. Search space

We see the entire system as a set of independent *chromosomes*, each representing a distinct subset of system parameters. A chromosome carries variation operators necessary for combination with other chromosomes of its type. In SymTA/S we currently use the standard operators mutation and crossover which are independently applied to the chromosomes during exploration.

Besides an exploration methodology allowing to narrow the search space (see Section 4.3), the strategy to guide the exploration through the search space, which is implemented by the variation operators of the specific chromosomes, is also very important to achieve high exploration efficiency.

In the following sections some chromosomes used in our exploration framework are presented.

5.1.1. Priority assignments on priority-scheduled resources

The optimization of priority assignments is a discrete permutation problem. In the context of evolutionary optimization such permutation problems are well studied, and thus efficient coding techniques and variation operators achieving good optimization results are known.

There exist several exact models of evolutionary algorithms based on a binary string representation of the problem. However, in our exploration framework we encode the priority assignment directly as a permutation. That means, the priority assignment on a resource is encoded as a list of integers containing one entry per process denoting its priority level (with 1 representing the highest priority).

Starting from a random set of priority assignments, we solve the ordering problem given by the permutation encoding by using a simple mutation operator and several crossover operators from literature (for a small overview see Whitley and Yoo (1995)).

Mutating a priority assignment is simply achieved by exchanging the priority of two processes and letting all others untouched.

For the crossing of two priority assignments we use *Order Crossover* (Davis, 1985), *Order Crossover 2* (Syswerda, 1990) and *Position Based Crossover* (Syswerda, 1990). In experimental results they turned out to be effective in solving the problem of assigning priorities on priority scheduled resources in the context of distributed systems.

Order crossover intends to preserve position information during the crossover process and works as follows. The offspring inherits the priority assignments of the tasks between two randomly chosen crossover points in the priority list from the first parent. The remaining priorities are inherited from the second parent, beginning at the first position of its priority list and adding them to the offsprings priority list starting from the second crossover point. Thereby, priorities that are already assigned are skipped.

Parent 1	: 1	2	3	4	5	6
Parent 2	: 3	2	6	5	4	1
Cross Pts	:		*		*	
Offspring	: 6	1	3	4	5	2

Order crossover 2 differs from Order Crossover in that several crossover positions are chosen randomly and the order in which the priorities at these position occur in the second parent is imposed to the first parent in order to create the offspring. Example:

Parent 1	: 1	2	3	4	5	6
Parent 2	: 3	2	6	5	4	1
Cross Pts	: *		*		*	
Offspring	: 1	2	3	6	5	4

1, 3 and 5 are selected as crossover positions. The ordering of the priorities at these positions from parent 2 will be imposed on parent 1. The priorities from parent 2 at the selected positions are 3, 6 and 4. In parent 1 the same priorities are found at the positions 3, 4 and 6. In the offspring the priorities at these positions (i.e. 3, 4 and 6) are reordered to match the order of the same priorities in parent 2 (i.e. 3, 6 and 4). Therefore $offspring[3] = 3$, $offspring[4] = 6$ and $offspring[6] = 4$. The remaining priorities are directly copied from parent 1.

Position based crossover intends, just like Order Crossover, to preserve position information. But instead of inheriting the priorities between two selected positions of one parent, several random positions are chosen. The priorities at those positions are inherited from the first parent and the remaining priorities are taken from the second parent in the order they appear, skipping over all priorities already assigned in the offspring. Example:

Parent 1	: 1	2	3	4	5	6
Parent 2	: 3	2	6	5	4	1
Cross Pts	: *			*	*	
Offspring	: 1	3	2	4	5	6

5.1.2. Time slot sizes on TDMA scheduled resources

The search space of all time slot assignments for the tasks on a TDMA scheduled resource is very large, even if we fix the turn-length and the arithmetic precision. Turn-length variation, which is often necessary to find good solutions, adds another search dimension. Since it is unrealistic to try all possible time slot assignments and turn-lengths, a good strategy to walk through the search space is indispensable.

One approach frequently used for continuous optimization problems like TDMA time slot optimization in the context of evolutionary algorithms, is discretizing the desired search space into a power of 2 and using a binary string representation with binary variation operators, like i.e. single-point crossover.

However, in the SymTA/S exploration framework we decided to use a real number coding of the problem variables with arithmetic variation operators. This is suitable for the given problem because it gives much more control over the generated alternatives and allows to implement problem-aware variation operators guaranteeing the validity of the generated configurations.

In the following we introduce arithmetic real-coded variation operators tailored for time slot and turn optimization on TDMA scheduled resources. Thereby, our exploration strategy is split into two aspects: optimizing the admitted loads of the mapped tasks as well as

optimizing the TDMA turn-length. Both factors together define the quality of a time slot assignment.

The reason for separating these two problem parameters is the increased control over the optimization process. By configuring the probabilities for the use of the two different operator types for crossover and mutation, the designer can decide which of them is the preferred search parameter. In the extreme case, she can, for example, hold the turn-length constant and optimize only by varying the admitted loads.

According to the two problem aspects we introduce four variation operators. One crossover and one mutation operator that vary the admitted loads of the mapped tasks while letting the turn-length constant as well as one crossover and one mutation operator that vary the turn-length and make sure that the admitted loads of the tasks stay constant.

The crossover operators implement a heuristic strategy of converging towards solutions lying "between" individuals currently considered by the evolutionary algorithm, whereas the mutation operators serve to break out of local minima by increasing or decreasing the admitted loads and the turn-length, respectively, within a configurable limit.

Extensive experiments using synthetical applications have shown, that the proposed variation operators are superior to standard binary-coded operators performing a uniform search over all possible time slot assignments in respect of time needed to find valid time slot assignments for given systems. Thereby, the performance difference is particularly noticeable for systems with narrow deadlines. Details about the experiments can be found in Hamann and Ernst (2005).

Creation of the initial population For the creation of the initial population we specify an initial TDMA turn-length $turn_{init}$. Note that choosing a sub-optimal initial turn-length for the initial population does not lead to the incapability of the time slot chromosome to find valid solutions because the proposed variation operators are capable of adapting the turn-length in the course of optimization. Nevertheless, if the designer chooses a good initial turn-length the chromosome converges faster towards the solution space.

Let R be a TDMA scheduled resource subjected to optimization with the tasks $T_0, \ldots, T_{k-1}$ mapped on it. The worst-case execution time, i.e. assuming no interrupts, of T_i is denoted by $WCET_i$, its activating period by $period_i$ and the length of its time slot by $slot_i$. In the following we refer to a specific time slot assignment as *individual*.

In order to create only valid (i.e. resource not overloaded, etc.) individuals for the initial population, we have to ensure that for each task T_i its maximum load $load_{\max;i}$ does not exceed its admitted load $load_{adm;i}$.

$$load_{adm;i} \geq load_{\max;i} \Leftrightarrow \frac{slot_i}{turn_{init}} \geq \frac{WCET_i}{period_i}$$
$$\Leftrightarrow slot_i \geq \frac{WCET_i}{period_i} * turn_{init}$$

This implies for T_i a minimum time slot

$$slot_{\min;i} = \frac{WCET_i}{period_i} * turn_{init}.$$

Algorithm 1 is used to create the initial population which is uniformly distributed in the search space of all valid time slot assignments with a turn-length of $turn_{init}$. To do so, it randomly distributes the initial turn to the tasks $T_0, \ldots, T_{k-1}$. It respects the above

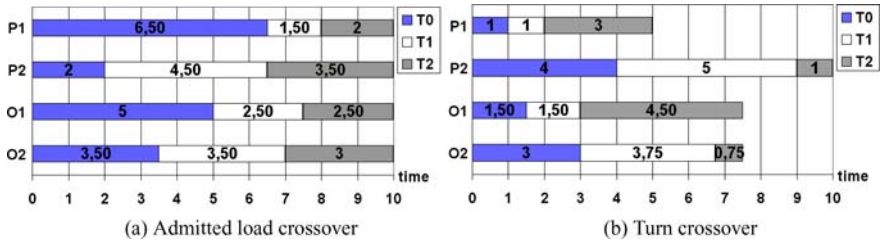


Fig. 10 Crossover operators

mentioned minimum time slot length to prevent the creation of non-schedulable individuals.

Algorithm 1 Create valid initial individual

Input : initial turn-length: $turn_{init}$
minimum time slots: $slot_{min;0}, \ldots, slot_{min;k-1}$
Output: valid time slot assignment for $T_0, \ldots, T_{k-1}$

1: $free = turn_{init}$;
2: $set = \{0, 1, \ldots, k-1\}$;
3: **while** $(set \neq \emptyset)$ **do**
4: choose random $r \in set$;
5: $set = set \setminus r$;
6: **if** $(set = \emptyset)$ **then**
7: $slot_r = free$;
8: **else**
9: $slot_{\max} = free - \sum_{x \in set} slot_{min;x}$;
10: choose random $slot_r \in [slot_{min;r}, slot_{\max}]$;
11: $free = free - slot_r$;
12: **end if**
13: **end while**

Crossover operators Algorithm 2 describes the crossover operator varying the admitted loads while letting the turn-length constant. As input it takes two parent individuals from which it creates two offsprings. Its optimization strategy is related to a binary search method.

The admitted loads of the offsprings are placed evenly (i.e. at $\frac{1}{3}$ and $\frac{2}{3}$) in the respective admitted load interval defined by the two parents. The time slots of offsprings 1 and offspring 2, respectively, are then calculated according to the turn-length given by parent 1 and parent 2, respectively. Figure 10(a) gives an example for this crossover operator.

The crossover operator varying the turn-length is described by Algorithm 3. It also pursuits a binary search related strategy. Given the two parent individuals it calculates their average turn-length (lines 1–7). Offspring 1 and offspring 2, respectively, is then created by adapting the time slots of parent 1 and parent 2, respectively, to the average turn-length letting the admitted loads untouched (lines 8–11). Figure 10(b) visualizes the functionality of this crossover operator by means of an example.

The described crossover operators lead to the convergence of the obtained time slot assignments towards (locally) optimal solutions contained "between" individuals considered by the evolutionary algorithm. Of course, it is possible that the variety of the initial

Algorithm 2 Crossover admitted load

Input : time slots of parent p_1: $[slot_{p_1;0}, \ldots, slot_{p_1;k-1}]$
time slots of parent p_2: $[slot_{p_2;0}, \ldots, slot_{p_2;k-1}]$
Output: time slots of offspring o_1: $[slot_{o_1;0}, \ldots, slot_{o_1;k-1}]$
time slots of offspring o_2: $[slot_{o_2;0}, \ldots, slot_{o_2;k-1}]$

1: $turn_{p_1} = 0$;
2: $turn_{p_2} = 0$;
3: **for** $(i = 0;\ i <= k-1;\ i = i+1)$ **do**
4: $\quad turn_{p_1} = turn_{p_1} + slot_{p_1;i}$;
5: $\quad turn_{p_2} = turn_{p_2} + slot_{p_2;i}$;
6: **end for**
7: **for** $(i = 0;\ i <= k-1;\ i = i+1)$ **do**
8: $\quad load_{adm;p_1;i} = slot_{p_1;i}/turn_{p_1}$;
9: $\quad load_{adm;p_2;i} = slot_{p_2;i}/turn_{p_2}$;
10: $\quad difference = \mid load_{adm;p_1;i} - load_{adm;p_2;i} \mid$;
11: $\quad$ **if** $(load_{adm;p_1;i} < load_{adm;p_2;i})$ **then**
12: $\quad\quad slot_{o_1;i} = (load_{adm;p_1;i} + difference/3) * turn_{p_1}$;
13: $\quad\quad slot_{o_2;i} = (load_{adm;p_2;i} - difference/3) * turn_{p_2}$;
14: $\quad$ **else**
15: $\quad\quad slot_{o_1;i} = (load_{adm;p_1;i} - difference/3) * turn_{p_1}$;
16: $\quad\quad slot_{o_2;i} = (load_{adm;p_2;i} + difference/3) * turn_{p_2}$;
17: $\quad$ **end if**
18: **end for**

population is insufficient to find good solutions only by using these crossover operators. Additionally, the exploration may get stuck in a local optimum, without the possibility to reach globally better solutions.

Therefore, we introduce two mutation operators, enabling the evolutionary algorithm to break out of local optima and to reach parts of the search space not yet considered.

Algorithm 3 Crossover turn

Input : time slots of parent p_1: $[slot_{p_1;0}, \ldots, slot_{p_1;k-1}]$
time slots of parent p_2: $[slot_{p_2;0}, \ldots, slot_{p_2;k-1}]$
Output: time slots of offspring o_1: $[slot_{o_1;0}, \ldots, slot_{o_1;k-1}]$
time slots of offspring o_2: $[slot_{o_2;0}, \ldots, slot_{o_2;k-1}]$

1: $turn_{p_1} = 0$;
2: $turn_{p_2} = 0$;
3: **for** $(i = 0;\ i <= k-1;\ i = i+1)$ **do**
4: $\quad turn_{p_1} = turn_{p_1} + slot_{p_1;i}$;
5: $\quad turn_{p_2} = turn_{p_2} + slot_{p_2;i}$;
6: **end for**
7: $turn_{new} = (turn_{p_1} + turn_{p_2})/2$;
8: **for** $(i = 0;\ i <= k-1;\ i = i+1)$ **do**
9: $\quad slot_{o_1;i} = slot_{p_1;i}/turn_{p_1} * turn_{new}$;
10: $\quad slot_{o_2;i} = slot_{p_2;i}/turn_{p_2} * turn_{new}$;
11: **end for**

Mutation operators The mutation operator varying the admitted load while letting the turn-length constant is described by algorithm 4. As input it takes one parent individual from which it creates one offspring. After initialization, $\frac{r}{2}$ pairs of tasks are chosen (lines 6–7).

For each of these pairs the first task gives a part of its disposable time slot (i.e. the time slot it can dispense without overloading the resource) to the second (lines 8–13). The percentage of the disposable time slot dispensed is randomly chosen in the interval $]0, d_{\max} \leq 1]$, where $d_{\max}$ is configurable. Figure 11(a) shows the functionality of this mutation operator by means of an example.

Algorithm 5 describes the mutation operator varying the turn-length. First the target turn-length is chosen, by increasing or decreasing the turn-length of the parent by a percentage randomly chosen in the interval $]0, d_{\max} \leq 1]$, where $d_{\max}$ is configurable (lines 1–11). The offspring's time slot assignments are then calculated to sum up in the target turn-length without altering the admitted loads given by the parent's time slot assignment (lines 12–14). Figure 11(b) shows the functionality of this mutation operator for a turn-length reduction of 20%.

Algorithm 4 Mutate admitted load

Input : time slots of parent p: $[slot_{p;0}, \ldots, slot_{p;k-1}]$
max. % of disposable time slot dispensed: $d_{\max}$
Output: time slots of offspring o: $[slot_{o;0}, \ldots, slot_{o;k-1}]$

1: $turn_p = 0$;
2: **for** $(i = 0;\ i <= k-1;\ i = i+1)$ **do**
3: $\quad slot_{o;i} = slot_{p;i}$;
4: $\quad turn_p = turn_p + slot_{p;i}$;
5: **end for**
6: choose pair random $r \in [2, \ldots, k]$;
7: choose r distinct integers $q_0, \ldots, q_{r-1} \in [0, k-1]$;
8: **for** $(i = 0;\ i <= r-1;\ i = i+2)$ **do**
9: $\quad slot_{disposable} = slot_{p;q_i} - load_{\max;q_i} * turn_p$;
10: $\quad$ choose random double $d_{applied} \in]0, d_{\max}]$;
11: $\quad slot_{o;q_i} = slot_{o;q_i} - d_{applied} * slot_{disposable}$;
12: $\quad slot_{o;q_{i+1}} = slot_{o;q_{i+1}} + d_{applied} * slot_{disposable}$;
13: **end for**

Algorithm 5 Mutate turn

Input : time slots of parent p: $[slot_{p;0}, \ldots, slot_{p;k-1}]$
max. % by which turn is cut or extended: $d_{\max}$
Output: time slots of offspring o: $[slot_{o;0}, \ldots, slot_{o;k-1}]$

1: $turn_p = 0$;
2: **for** $(i = 0;\ i <= k-1;\ i = i+1)$ **do**
3: $\quad turn_p = turn_p + slot_{p;i}$;
4: **end for**
5: choose random boolean b;
6: choose random double $d_{applied} \in]0, d_{\max}]$;
7: **if** $(b = true)$ **then**
8: $\quad turn_{new} = turn_p + d_{applied} * turn_p$;
9: **else**
10: $\quad turn_{new} = turn_p - d_{applied} * turn_p$;
11: **end if**
12: **for** $(i = 0;\ i <= k-1;\ i = i+1)$ **do**
13: $\quad slot_{o;i} = slot_{p;i} / turn_p * turn_{new}$;
14: **end for**

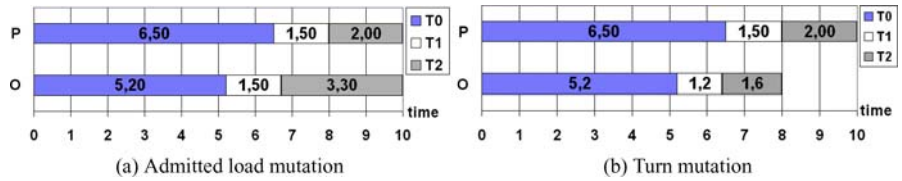


Fig. 11 Mutation operators

5.1.3. Traffic shaping

A traffic shaping chromosome represents a d^--EAFs performing traffic shaping on an event stream connecting two functionally dependent components. In our framework it is realized using a real number representation of the minimum distance d^- and arithmetic real-coded variation operators. Its search range can be bounded by a search interval and the arithmetic precision. For convenience reasons, the bounds of the search interval can be configured by specifying the allowed buffering delays produced by the traffic shaper (more comprehensible for timing constrained systems) or the allowed buffer sizes needed for the traffic shaper (more comprehensible for buffer constrained systems). Internally, these specifications are translated into corresponding d^- values for the search interval (see Section 2.4).

The search strategy of the variation operators is similar to that of the TDMA time slot chromosome, with the difference that only one problem dimension needs to be considered. Starting from random d^- values within the search interval, the search is conducted by one crossover and one mutation operator. The crossover operator takes as input two parent configurations and creates two offsprings. The d^- values of the first and the second offspring are placed at $\frac{1}{3}$ and $\frac{2}{3}$, respectively, in the d^- interval defined by the two parents. This strategy leads to the convergence of the obtained d^- values towards optimal solutions lying between the configurations considered by the evolutionary algorithm. The purpose of the mutation operator is to prevent the search getting stuck in local minima and to reach parts of the search space, which are inaccessible by only using the crossover operator. It creates one offspring by increasing or decreasing the d^- value of the parent configuration by a random percentage in the interval $]0, d_{\max}]$, where $d_{\max}$ is configurable.

5.2. Optimization objectives

The SymTA/S exploration framework is capable to perform a multi-objective optimization of several concurrent optimization objectives, leading usually to the discovery of several pareto-optima.

Optimization objectives can be any kind of metric defined on desired or undesired properties of the considered system. Note that some metrics only make sense in combination with constraints. Each system configuration considered during exploration is associated with a fitness vector containing one entry for every concurrent optimization objective.

5.2.1. Example metrics

In the following some example optimization objectives used in the SymTA/S exploration framework will be introduced using the following notation:

R	maximum response time of a task or maximum end-to-end latency along a path
D	deadline (task or end-to-end)
ω	constant weight > 0
k	number of tasks or number of constrained tasks/paths in the system

Note that we exemplify using timing properties of distributed systems. Corresponding metrics can easily be derived to optimize jitter and local or global buffer requirements.

A basic global metric for expressing the timing qualities of a given system configuration is the weighted sum of completion times:

$$\sum_{i=1}^{k} \omega_i * R_i$$

Even though this metric can be used to minimize response times of tasks or end-to-end latencies, its practical relevance is limited if we consider systems with timing constraints. For such systems metrics taking deadlines into account are much more appropriate.

The lateness of a task or a path is defined as the amount of time by which it misses its deadline. Consequently, a negative value denotes that the task (the path) completes before the expiration of its deadline. In the case of constrained systems the lateness can be used to define expressive global metrics for the timing properties of a given system configuration. Following example metric can be used to minimize the (weighted) average lateness for a given system:

$$\sum_{i=1}^{k} \omega_i * (R_i - D_i)$$

The given metric expresses the average timing bahavior of a system configuration with regard to its timing constraints. It might mislead an evolutionary optimizer and prevent him from finding system configurations fulfilling all timing constraints, since met deadlines compensate linearly for missed deadlines. For systems with hard real-time constraints, metrics with higher penalties for missed deadline and less rewards for met deadlines can be more appropriate, since they lead to a more likely rejection of system configurations violating hard deadline constraints. Following example metric penalizes violated deadlines in an exponential way and can be used to optimize the timing properties of a system with hard real-time constraints:

$$\sum_{i=0}^{k} c_i^{R_i - D_i}, \; c_i > 1 \text{ constant}$$

Another refinement of the global timing metric can be achieved by not using absolute values for the lateness of a task (a path), but expressing it relatively to the constraint. Such a metric is much more appropriate for systems containing timing constraints with different orders of magnitude, as it does not discriminate small constraints in comparison to larger ones. In the case of significant differences in the order of magnitude, small constraints would simply be neglected using the prior metric. Following example metric penalizes relative

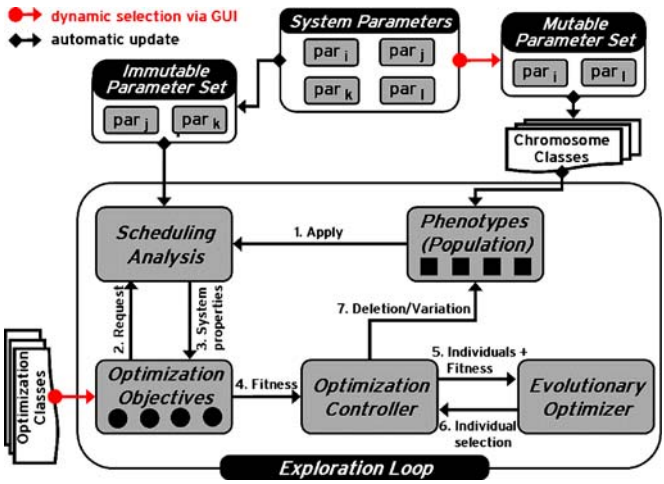


Fig. 12 Design space exploration loop

deadline misses in an exponential way:

$$\sum_{i=0}^{k} c_i^{\frac{R_i - D_i}{D_i}}, \quad c_i > 1 \text{ constant}$$

The above presented metrics integrate all timing properties of a given system configuration into a single value. Using this metrics is making sense to quickly find good functioning system configurations. However, they do not provide information about possible design trade-offs with regard to multiple timing properties.

For this purpose the SymTA/S exploration framework allows to use the above introduced metrics for a subset of tasks or paths (including single ones). Together with the supported pareto-optimization, this allows the designer to focus the optimization process on a set of specific timing properties of the given system, and thus enables her to identify interesting design trade-offs among the obtained solution set.

5.3. Design space exploration loop

Figure 12 shows the design space exploration loop performed in our framework (Hamann et al., 2004). The *Optimization Controller* is the central element. It is connected to SymTA/S and to an *Evolutionary Optimizer*. SymTA/S checks the validity of a given system configuration and provides performance data necessary to calculate the performance metrics, which are subject to optimization. The *Evolutionary Optimizer* is responsible for the problem-independent part of the optimization problem, i.e. elimination of system configurations and selection of interesting system configurations for variation. Currently, we use SPEA2 (Strength Pareto Evolutionary Algorithm 2) (Zitzler et al., 2001) and FEMO (Fair Evolutionary Multiobjective Optimizer) (Laumanns et al., 2002) for this part. They are coupled via PISA (Platform and Programming Language Independent Interface for Search Algorithms) (Bleuler et al., http://www.tik.ee.ethz.ch/pisa/).

Note that the selection and elimination strategy depends on the used multi-objective optimizer. FEMO, for instance, eliminates all dominated system configurations in every iteration

and pursuits a fair sampling strategy, i.e. each parent configuration participates in the creation of the same number of offspring configurations. This leads to a uniform search in the neighborhood of elitist individuals.

The problem-specific part of the optimization problem is coded in the chromosomes and their variation operators, i.e. crossover and mutation.

Before exploration can be started the designer has to select the desired search space (see Sections 4.1 and 5.1) and the optimization objectives (see Section 5.2) she wants to optimize. The chromosomes representing the search space are included in the evolutionary exploration, while all other system parameters remain immutable. After the designer has selected the search space and the optimization task, SymTA/S is initialized with the immutable part of the system, and the selected chromosomes are used as blueprints to create the initial population. Note that each chromosome is responsible for creating initial values for the system parameters it represents. Usually, chromosomes create random initial parameters in order to achieve a uniform distribution of the initial set of system configurations in the search space. However, sometimes it may be favorable to initialize chromosomes with heuristically determined parameters.

In the following we refer to a specific system configuration consisting of several specific chromosomes (phenotypes) as *individual*.

For each individual in the population the following is done:

- Step 1: The chromosomes of the considered individual are applied to the SymTA/S engine. This completes the system and it can be analyzed.
- Step 2 + 3: Each optimization objective requests the necessary system properties of the analyzed system to calculate its fitness value.
- Step 4: The fitness values are communicated to the *Optimization Controller*.

Once these 4 steps are performed for each individual inside the population, the *Optimization Controller* sends a list of all individuals and their fitness values to the *Evolutionary Optimizer* (step 5). Based on the fitness values the *Evolutionary Optimizer* creates two lists, the list of individuals to be deleted and the list of individuals selected for variation, and sends them back to the *Optimization Controller* (step 6). Based on the two lists the *Optimization Controller* then manipulates the population, i.e. it deletes the according individuals, creates new offsprings based on the individuals selected for variation, and adds them to the population (step 7). Note that the variation operators, i.e. mutation and crossover, utilized to create new individuals for the population are chromosome specific.

This completes the processing of one generation. The whole loop begins again with the new created population.

At each moment during exploration the designer can choose to modify the search space. This consists, like explained in Section 4.3, in adding/removing chromosomes to/from the search space. The necessary adaptations and the reevaluation of the fitness values are performed automatically by the framework and the next exploration iteration is then started.

6. System on chip example

The system in Fig. 13 represents a SoC consisting of a micro-controller (*uC*), a digital signal processor (*DSP*) and dedicated hardware (*HW*), all connected via an on-chip bus (*BUS*). The *HW* acts as an interface to a physical system. It runs one task (*sys_if*) which issues actuator commands to the physical system and collects routine sensor readings. *sys_if* is controlled by controller task *ctrl*, which evaluates the sensor data and calculates the necessary actuator

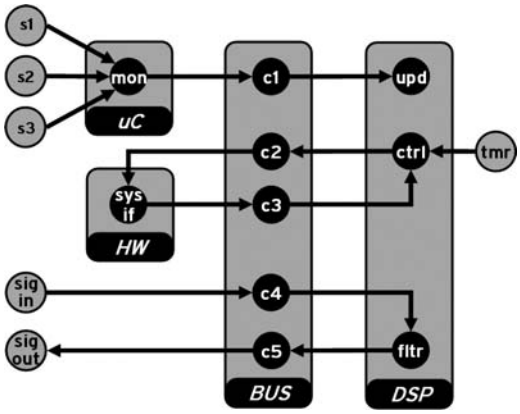


Fig. 13 System on chip example

commands. *ctrl* is activated by a periodic timer (*tmr*) and by the arrival of new sensor data (AND-activation in a cycle).

The physical system is additionally monitored by 3 smart sensors ($s_1 - s_3$), which produce data sporadically as a reaction to irregular system events. This data is registered by an OR-activated monitor task (*mon*) on the *uC*, which decides how to update the control algorithm. This information is sent to task *upd* on the *DSP*, which writes the updated controller parameters into shared memory.

The *DSP* additionally executes a signal-processing task (*fltr*), which filters a stream of data arriving at input *sig_in*, and sends the processed data via output *sig_out*. All communication (with the exception of shared-memory on the *DSP*) is carried out by communication tasks *c1 – c5* over the on-chip *BUS*.

Computation and communication tasks shall have the core execution times (i.e. assuming no interrupts) listed in Table 3. We assume the event models at system inputs specified in Table 4. In order to function correctly, the system has to satisfy the path latency constraints and the maximum jitter constraint at sig_out listed in Tables 5(a) and (b). In the following we assume that the *DSP* as well as the *BUS* are scheduled according to a static priority preemptive policy.

Table 3 Core execution times

Computation task	Core execution time
mon	[10, 12]
sys_if	[15, 15]
fltr	[12, 15]
upd	[5, 5]
ctrl	[20, 23]
Communication task	**Core communication time**
c1	[4, 4]
c2	[4, 4]
c3	[4, 4]
c4	[8, 8]
c5	[4, 4]

Table 4 Input event models

Input	Event model
s_1	sporadic, $\mathcal{P}_{s_1} = 1000$
s_2	sporadic, $\mathcal{P}_{s_2} = 750$
s_3	sporadic, $\mathcal{P}_{s_3} = 600$
sig_in	periodic, $\mathcal{P}_{in} = 60$
tmr	periodic, $\mathcal{P}_{tmr} = 70$

Table 5 Constraints

(a) Path latency constraints

Constraint #	Path	Maximum latency
1	$s_i \rightarrow upd$	70
2	$sig\_in \rightarrow sig\_out$	60
3	cycle (e.g. $ctrl \rightarrow ctrl$)	140

(b) Maximum jitter constraint

Constraint #	Output	Event model jitter
4	*sig_out*	$\mathcal{J}_{sig\_out,max} = 22$

7. Exploring the example system

In this section, we explore the given SoC example. We will do this in several steps, modifying the search space gradually. First we will perform a local optimization on the *BUS* altering only the priorities of the communication channels. Afterwards, we will extend the search space by allowing d^--EAFs at reasonable positions. During these two steps we will assume the following priority assignment on the *DSP*: $upd > fltr > ctrl$. Finally we will optimize the system globally, i.e. the priority assignment on the *BUS* and the *DSP* as well as traffic shaping. Optimization objectives during all these exploration steps are the minimization of the path latencies ($s_i \rightarrow upd$ and $sig\_in \rightarrow sig\_out$), the minimization of the cycle latency ($ctrl \rightarrow ctrl$) and the minimization of the jitter at output sig_out ($\mathcal{J}_{sig\_out}$).

7.1. Optimizing the BUS

The first step in our design space exploration is local optimization of the *BUS*. Although there are only five communication channels on the *BUS*, it is not intuitive for the designer which priority assignments lead to systems that meet all constraints. Local exploration of the *BUS* will give us a first feeling about the systems behavior, and thus a deeper understanding of its performance dependencies. Table 6 shows the obtained solutions.

Table 6 Pareto-optimal solutions: local optimization on the BUS

#	*BUS* tasks	*DSP* tasks	$s_i \rightarrow upd$	$sig\_in \rightarrow sig\_out$	$ctrl \rightarrow ctrl$	$\mathcal{J}_{sig\_out}$
1	*c5*, *c4*, *c1*, *c2*, *c3*	*upd*, *fltr*, *ctrl*	55	42	120	18
2	*c5*, *c4*, *c2*, *c1*, *c3*	*upd*, *fltr*, *ctrl*	59	42	112	18
3	*c5*, *c2*, *c4*, *c1*, *c3*	*upd*, *fltr*, *ctrl*	59	46	108	22
4	*c4*, *c5*, *c2*, *c3*, *c1*	*upd*, *fltr*, *ctrl*	63	42	96	18
5	*c5*, *c2*, *c4*, *c3*, *c1*	*upd*, *fltr*, *ctrl*	63	46	92	22

As we can see there are five priority assignments for the communication channels on the *BUS* leading to functioning systems. These solutions are pareto-optimal, which means that they represent a certain trade-off between multiple objectives, leaving it to the designer to decide which solution to adopt.

We observe that channels $c4$ and $c5$ have high priorities in all obtained solutions, whereas channel $c3$ has throughout the lowest or second lowest priority.

7.2. Traffic shaping

In the second step we want to evaluate the optimization potential of selective traffic shaping (see Section 2.4) for the given architecture. We extend our search-space by using d^--EAFs at the output of task *mon*. It is making sense to perform traffic shaping at this location, because the OR-activation of *mon* can lead in the worst-case scenario to bursts at its output. That is, if all three sensors trigger at the same time, *mon* will send three packets over the *BUS* with a minimum distance of 10 time units, which is its minimum core execution time. This transient load peak affects the overall system performance in a negative way. A d^--EAF is able to increase this minimum distance in order to weaken the global impact of the worst-case burst. Exploration over the minimum distance d^- of successive packets enforced by the inserted shaper is subject of this exploration step.

We observed in the previous experiment that communication channel $c3$ was always assigned the lowest or second lowest priority. Even in the lowest case, the cycle constraint ($ctrl \rightarrow ctrl$) was easily met. Therefore, we will fix channel $c3$ to the lowest priority on the bus. This narrows the search space considerably, the number of possible priority assignments on the bus is reduced from $5! = 120$ to $4! = 24$.

Table 7 shows the additional pareto-optimal solutions found using d^--EAFs at the output of *mon* extending the minimum distance of successive events to integer values between 11 and 20 time units. Solutions which are dominated by the results obtained in the previous section are not listed.

We observe that performing traffic shaping at the output of *mon* leads to several new interesting solutions. We found new priority assignments on the *BUS* which, combined with a certain shaper, result in better values for the path constraints $s_i \rightarrow upd$, $sig\_in \rightarrow sig\_out$ and the jitter constraint $\mathcal{J}_{sig\_out}$. Solely the previously obtained values for the cycle constraint $ctrl \rightarrow ctrl$ are not reached, but the constraint remains fulfilled by a large margin.

Only two different priority assignments, $c5 > c1 > c4 > c2 > c3$ and $c1 > c5 > c4 > c2 > c3$, occur in the solutions listed in Table 7. Let us take a closer look on the global impact of traffic shaping at the output of *mon*.

Tables 8(a) and (b) show the performance of the system with growing minimum distance of events at the output of *mon* for these two priority assignments. Rows containing pareto optimal solutions are emphasized. Note that in this example a shaper extending the minimum

Table 7 Additional pareto-optimal solutions: traffic shaping at mon output

#	*BUS* tasks	*DSP* tasks	d^-	$s_i \rightarrow upd$	$sig\_in \rightarrow sig\_out$	$ctrl \rightarrow ctrl$	$\mathcal{J}_{sig\_out}$
6	*c1*, *c5*, *c4*, *c2*, *c3*	*upd*, *fltr*, *ctrl*	13	51	45	120	21
7	*c1*, *c5*, *c4*, *c2*, *c3*	*upd*, *fltr*, *ctrl*	14	53	45	116	21
8	*c1*, *c5*, *c4*, *c2*, *c3*	*upd*, *fltr*, *ctrl*	16	57	45	112	21
9	*c5*, *c1*, *c4*, *c2*, *c3*	*upd*, *fltr*, *ctrl*	17	63	41	112	17
10	*c1*, *c5*, *c4*, *c2*, *c3*	*upd*, *fltr*, *ctrl*	20	65	40	104	16

Table 8 System performance with traffic shaping at mon output

d^-	$s_i \rightarrow upd$	$sig\_in \rightarrow sig\_out$	$ctrl \rightarrow ctrl$	$\mathcal{J}_{sig\_out}$
10	49	50	162	26
11	51	50	162	26
12	53	46	120	22
13	55	46	120	22
14	57	46	116	22
15	59	46	116	22
16	61	46	112	22
17	**63**	**41**	**112**	**17**
18	65	41	112	17
19	67	41	112	17
20	69	41	104	17

(a) BUS priorities: $c5 > c1 > c4 > c2 > c3$

d^-	$s_i \rightarrow upd$	$sig\_in \rightarrow sig\_out$	$ctrl \rightarrow ctrl$	$\mathcal{J}_{sig\_out}$
10	45	54	162	30
11	47	54	162	30
12	49	50	120	26
13	**51**	**45**	**120**	**21**
14	**53**	**45**	**116**	**21**
15	55	45	116	21
16	**57**	**45**	**112**	**21**
17	59	45	112	21
18	61	45	112	21
19	63	45	112	21
20	**65**	**40**	**104**	**16**
21	67	40	104	16
22	69	40	104	16

(b) BUS priorities: $c1 > c5 > c4 > c2 > c3$

distance to a value between 11 and 20 time units needs to store at most one packet at a time. To achieve larger minimum distances, two packets need to be stored in the worst-case.

We see that the value for the path constraint $s_i \rightarrow upd$ is climbing with growing minimum distance. This is not surprising because the inserted shaper is creating additional latency in the worst-case, depending on the desired minimum distance. The longest minimum distance that does not lead to violation of the path constraint $s_i \rightarrow upd$ for the priority assignments $c5 > c1 > c4 > c2 > c3$ and $c1 > c5 > c4 > c2 > c3$ are 20 and 22 respectively. However, while the shaper leads to increased values for the path constraint $s_i \rightarrow upd$, the rest of the system is profiting from the weakend burst.

7.3. Including the DSP

So far, we obtained ten solutions representing different trade-offs for our example SoC by using local exploration techniques. Now, we extend our search space by the priority assignment on the *DSP*. Since we already observed that the cycle constraint is uncritical, we will

Table 9 Additional pareto-optimal solutions: global optimization

#	*BUS* tasks	*DSP* tasks	d^-	$s_i \rightarrow upd$	$sig\_in \rightarrow sig\_out$	$ctrl \rightarrow ctrl$	$\mathcal{J}_{sig\_out}$
11	*c5*, *c4*, *c1*, *c2*, *c3*	*fltr*, *upd*, *ctrl*	10	70	27	120	3
12	*c1*, *c5*, *c4*, *c2*, *c3*	*fltr*, *upd*, *ctrl*	12	64	35	120	11
13	*c5*, *c1*, *c4*, *c2*, *c3*	*fltr*, *upd*, *ctrl*	12	68	31	120	7
14	*c1*, *c5*, *c4*, *c2*, *c3*	*fltr*, *upd*, *ctrl*	14	68	35	116	11

fix the priorities of communication channels $c2$ and $c3$ to the second lowest and lowest on the *BUS* respectively. Additionally, we fix the priority of task $ctrl$ to the lowest on the *DSP*.

In Table 9 we see the new system configurations found. Solutions which are dominated by the results obtained in the previous sections are not listed.

The obtained solutions represent new interesting trade-offs because they lead to a low jitter at *sig_out*. This quality did not exist in any of the previously obtained system configurations. However, the low jitter at *sig_out* is bought with high values for the constraint $s_i \rightarrow upd$.

7.4. Summary of results

Table 10 gives an overview about all pareto-optimal system configurations found so far. The best reached values for each objective are emphasized. For example an attractive solution might be one where all constraints are fulfilled with a healthy margin to the respective maximum values. This is the case for solutions 4, 10, and 12 for instance.

For the given example system an exploration loop of 15 iterations with a population size of 50 individuals found all pareto-optimal solutions for the experiments in the Sections 7.1, 7.2 and 7.3 in almost every run. This exploration takes approximately 20 seconds. on a Pentium 4 at 2400 MHz.

Table 10 All pareto-optimal solutions

#	*BUS* tasks	*DSP* tasks	d^-	$s_i \rightarrow upd$	$sig\_in \rightarrow sig\_out$	$ctrl \rightarrow ctrl$	$\mathcal{J}_{sig\_out}$
1	*c5*, *c4*, *c1*, *c2*, *c3*	*upd*, *fltr*, *ctrl*	10	55	42	120	18
2	*c5*, *c4*, *c2*, *c1*, *c3*	*upd*, *fltr*, *ctrl*	10	59	42	112	18
3	*c5*, *c2*, *c4*, *c1*, *c3*	*upd*, *fltr*, *ctrl*	10	59	46	108	22
4	*c4*, *c5*, *c2*, *c3*, *c1*	*upd*, *fltr*, *ctrl*	10	63	42	96	18
5	*c5*, *c2*, *c4*, *c3*, *c1*	*upd*, *fltr*, *ctrl*	10	63	46	**92**	22
6	*c1*, *c5*, *c4*, *c2*, *c3*	*upd*, *fltr*, *ctrl*	13	**51**	45	120	21
7	*c1*, *c5*, *c4*, *c2*, *c3*	*upd*, *fltr*, *ctrl*	14	53	45	116	21
8	*c1*, *c5*, *c4*, *c2*, *c3*	*upd*, *fltr*, *ctrl*	16	57	45	112	21
9	*c5*, *c1*, *c4*, *c2*, *c3*	*upd*, *fltr*, *ctrl*	17	63	41	112	17
10	*c1*, *c5*, *c4*, *c2*, *c3*	*upd*, *fltr*, *ctrl*	20	65	40	104	16
11	*c5*, *c4*, *c1*, *c2*, *c3*	*fltr*, *upd*, *ctrl*	10	70	**27**	120	**3**
12	*c1*, *c5*, *c4*, *c2*, *c3*	*fltr*, *upd*, *ctrl*	12	64	35	120	11
13	*c5*, *c1*, *c4*, *c2*, *c3*	*fltr*, *upd*, *ctrl*	12	68	31	120	7
14	*c1*, *c5*, *c4*, *c2*, *c3*	*fltr*, *upd*, *ctrl*	14	68	35	116	11

8. Evaluating the exploration framework

In this section we conduct experiments to evaluate our exploration framework and its user-controlled exploration strategy regarding exploration efficiency (Section 8.1) and ability to scale to systems which are difficult to optimize (Section 8.2).

8.1. Experiment 1: Exploration efficiency

In the first experiment we compare different exploration strategies with respect to their efficiency, i.e. the average number of evaluated configurations to find the first working system configuration.

Fully automated closed exploration over all free system parameters, a widely pursued approach for the exploration of heterogeneous embedded systems, is taken as baseline for comparing the efficiency of the introduced exploration capabilities of our framework, i.e. optimization through traffic shaping and user-controlled exploration.

Accordingly, we evaluate three different exploration strategies:

1. closed exploration, i.e. including the scheduling parameters on all resources of the explored system into the search space,
2. closed exploration extending the search space with traffic shaping on two event streams in the system, and
3. user-controlled exploration.

Note, that all three exploration strategies are evaluated with our framework using the according search space composition.

For evaluating the efficiency of the different exploration strategies we utilize randomly generated systems with the following properties:

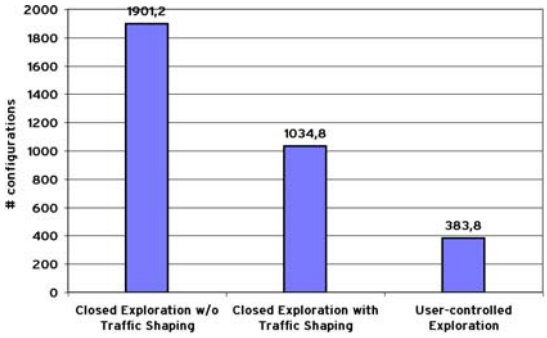
- 30 tasks with complex application structures, such as multiple input/output ports and functional cycles
- 6 static priority preemptive or TDMA scheduled resources
- task best-case and worst-case execution times between 5 and 100 time units
- periods at the system inputs between 200 and 600 time units
- jitter at the system inputs between 0 and 1000 time units
- multiple end-to-end deadlines

During exploration the last presented metric in Section 5.2 is used as optimization objective to quickly find working system configurations satisfying all end-to-end deadlines in the system.

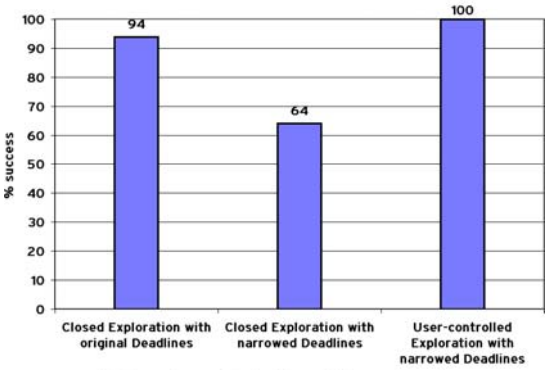
Figure 14(a) shows the averaged results obtained by exploring 50 systems with the three different strategies.

We observe that the closed exploration strategy needed to evaluate approximately 1900 configurations to find a working system in the average case. Compared to that, extending the search space with traffic shaping on two event streams increased exploration efficiency by almost factor 2. However, the best results were obtained by guiding the exploration manually. For the considered systems, the user-controlled exploration lead to working system configuration nearly 5 times faster than the closed exploration and more than 2.5 times faster than the closed exploration with traffic shaping.

The user-controlled exploration runs were performed modifying the search space two times in the average case. First, exploration was started including all scheduling parameters and traffic shaping at two selected positions. After reaching sufficiently low values for the



(a) Experiment 1: Exploration efficiency



(b) Experiment 2: Scaling of the exploration

Fig. 14 Evaluation of the exploration framework

utilized timing metric, the exploration was paused to modify the search space. At this point, some end-to-end deadlines were already fulfilled, and the search space was restricted to scheduling parameters on resources running tasks, which heavily contributed to the remaining deadline violations. Afterwards, exploration was resumed. For the case that no working system configuration was found after a while, exploration was paused again to perform a second search space modification. After the second exploration step, rarely more than one or two end-to-end deadlines remained violated, and it was not difficult to restrict the search space to one resource or one position in the system for traffic modulation to obtain a working system configuration in the third exploration step.

Note that the necessary decisions to modify the search space within the user-controlled exploration runs were always taken within one minute. The evaluation of one system configuration with the given size takes between 1 and 1.5 second on a Pentium 4 at 2400 MHz. Consequently, the overhead for one search space modification in the performed experiments corresponds approximately to the evaluation of 40 to 60 system configurations.

8.2. Experiment 2: Scaling of the exploration

In the second experiment we evaluate how the user-controlled exploration strategy scales in comparison to the closed exploration strategy. Scaling in this experiment does not mean increasing the size of the explored systems, but increasing the difficulty to find working configurations for them.

More precisely, for a given set of constrained systems, generated in the same way as in the previous experiment but containing 60 tasks and 15 resources, and a maximum bound of 5000 evaluated system configurations per exploration run, we are interested in how successful user-controlled exploration performs in finding working system configurations compared to fully automated closed exploration.

In order to compare scaling we perform two such experiments. The first experiment is performed using the original end-to-end deadlines of the generated systems. For the second experiment the end-to-end deadlines are narrowed by 5%, increasing the difficulty to find working system configurations.

Figure 14(b) shows the averaged results obtained by exploring 50 systems with both exploration strategies. Note that all exploration runs were conducted without including traffic shaping into the search space.

In the first experiment, the closed exploration succeeded for 94% of the considered systems to find a working system within the limit of 5000 evaluated configurations. However, for the second experiment with the narrowed deadlines the success rate dropped to 64%. This indicates that closed exploration has difficulties to scale to systems with narrow timing constraints.

Compared to that, we had no problems finding working configurations for the given systems by manually guiding the exploration. Note that the search space modifications during the user-controlled exploration runs were performed with the same strategy as in the experiment described in the previous section.

9. Conclusion

In this paper we presented a framework for flexible design space exploration and system optimization for heterogeneous SoC and distributed systems using SymTA/S and evolutionary optimization techniques.

The ambition of our framework is not to perform a global black-box optimization, but to give the designer control over the exploration process. To that end, the framework allows the dynamic (re)configuration of the search space during exploration without losing already obtained results. This feature enables the designer to perform multiple exploration steps, adjusting the search space as his understanding of the systems performance dependencies grows, in order to guide the search process towards interesting design sub-spaces containing good solutions. Thereby, the framework allows, at any time, a transparent comparison of so far obtained design alternatives with respect to multiple optimization objectives. Experiments using synthetical complex systems have shown that the user-controlled exploration approach increases exploration efficiency in comparison to closed automated exploration approaches.

In addition to classical optimization parameters, like e.g. scheduling, our framework supports system optimization by means of traffic shaping. The optimization potential through traffic shaping in complex heterogeneous SoC and distributed systems is very high. We saw in the discussed SoC example, that inserting a traffic shaper to reduce a transient load peak considerably improved system behavior, and consequently lead to the discovery of interesting design alternatives, which are not possible without traffic modulation. Additionally, experiments with synthetical systems indicate, that traffic shaping can broaden considerably the solution space, and thus can decrease exploration time to find working system configurations.

Acknowledgment We would like to thank Lothar Thiele and his group from the ETH Zürich for providing us with the PISA interface and the corresponding algorithms for multi-objective optimization.

References

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

Davis L (1985) Applying adaptive algorithms to epistatic domains. In Proc. of the 9th International Joint Conference on Artificial Intelligence (IJCAI) Los Angeles (CA), USA, pp. 162–164.

Deb K (2001) Multi-objective optimization using evolutionary algorithms. John Wiley, Chichester.

Dick RP, Jha NK (1998) MOGAC: A multiobjective genetic algorithm for hardware-software co-synthesis of hierarchical heterogeneous distributed embedded systems. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 17(10):920–935.

Erbas C, Erbas SC, Pimentel AD (2003) A multiobjective optimization model for exploring multiprocessor mappings of process networks. In Proc. of the IEEE/ACM/IFIP International Conference on HW/SW Codesign and System Synthesis (CODES-ISSS) Newport Beach, USA.

Garcia JJG, Harbour MG (1995) Optimized priority assignment for tasks and messages in distributed real-time systems. In Proc. of the IEEE/ACM Workshop on Parallel and Distributed Real-Time Systems Santa Barbara (CA), USA.

Givargis T, Vahid F (2002) Platune: A tuning framework for system-on-a-chip platforms. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 21(11):1317–1327.

Hamann A, Ernst R (2005) TDMA time slot and turn optimization with evolutionary search techniques. In Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE) Munich, Germany.

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

Hamann A, Jersak M, Richter K, Ernst R (2004) 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.

Laumanns M, Thiele L, Zitzler E, Welzl E, Deb K (2002) Running time analysis of multi-objective evolutionary algorithms on a simple discrete optimization problem. In Proc. of the Parallel Problem Solving From Nature Conference (PPSN) Granada, Spain.

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM 20(1):46–61.

Maxiaguine A, Künzli S, Chakraborty S, Thiele L (2004) Rate analysis for streaming applications with on-chip buffer constraints. In Proc. of the IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC) Yokohama, Japan, pp. 131–136.

Pimentel AD, Lieverse P, van der Wolf P, Hertzberger LO, Deprettere EF (2001) Exploring embedded-systems architectures with Artemis. In IEEE Computer.

Pop P, Eles P, Peng Z, Izosimov V, Hellring M, Bridal O (2004) Design optimization of multi-cluster embedded systems for real-time applications. In Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE) Paris, France.

Richter K (2004) Compositional performance analysis. PhD thesis, Technical University of Braunschweig.

Richter K (2004) On the characterization of communication traffic and task load models in performance verification and architecture evaluation. Technical Report TR-SPI-04-01, Institut für Datentechnik und Kommunikationsnetze, Technische Universität Braunschweig.

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

Richter K, Jersak M, Ernst R (2003) A formal approach to MpSoC performance verification. IEEE Computer 36(4).

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

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

Snider G (2001) Automated design space exploration for embedded computer systems. Technical Report HPL–2001–220, Hewlett-Packard Laboratories.

Syswerda G (1990) Schedule optimization using genetic algorithms. In Handbook of Genetic Algorithms New York, Van Nostrand Reinhold.

Thiele L, Chakraborty S, Gries M, Künzli S (2002) A framework for evaluating design tradeoffs in packet processing architectures. In Proc. of the 39th IEEE/ACM Design Automation Conference (DAC) New Orleans, USA, pp. 880–885.

Thiele L, Chakraborty S, Gries M, Maxiaguine A, Greutert J (2001) Embedded software in network processors - models and algorithms. In Proc. of the ACM Workshop on Embedded Software (EMSOFT) Lake Tahoe (CA), USA.

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

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

Tindell K, Kopetz H, Wolf F, Ernst R (2003) Safe automotive software development. In Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE) Munich, Germany, pp. 616–612.

Whitley DL, Yoo N (1995) Modeling simple genetic algorithms for permutation problems. In Foundations of Genetic Algorithms III San Francisco, CA, Morgan Kaufmann, pp. 163–184.

Zitzler E, Laumanns M, Thiele L (2001) SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland.

Arne Hamann received his Maîtrise degree in Computer Science from the University of Bordeaux 1, France, in 2001, and his Diploma degree in Computer Science from the Technical University of Braunschweig, Germany, in 2003. He is currently working as research scientist in the Embedded System Design Automation Group of Professor Ernst. His research interests include formal timing analysis and optimization of heterogeneous distributed real-time systems.

Dr. Kai Richter received a Diploma and a doctoral degree "summa cum laude" in Electrical Engineering from the Technical University of Braunschweig, Germany in 1998 and 2004. He authored more than 40 papers in internationally recognised journals and conferences. His research interests include timing and performance analysis of distributed embedded systems and embedded system architectures. Since 2005, he is co-founder and Chief Technical Officer of Symtavision that offers unique solutions and analysis tools, including SymTA/S for system-level real-time scheduling analysis.

Dr. Marek Jersak received his Diploma degree in Electrical Engineering from Aachen University of Technology, Germany in 1997 and his doctoral degree with honours from the Technical University of Braunschweig, Germany in 2004. Between 1997 and 1999 he worked as a Design Engineer for Conexant Systems, Newport Beach, California, on DSP compiler optimization and processor/compiler co-design. Since 2005 he is CEO of Symtavision, a spin-off from the Technical University of Braunschweig focusing on timing analysis and optimization for complex embedded real-time systems.

Rolf Ernst received a Diploma in Computer Science and a Ph.D. in Electrical Engineering from the University of Erlangen-Nuremberg, Germany, in 81 and 87. From 88 to 89, he was a Member of Technical Staff in the Computer Aided Design & Test Laboratory at Bell Laboratories, Allentown, PA. Since 90, he has been a professor of Electrical Engineering at the Technical University of Braunschweig, Germany, where he heads the Institute of Computer and Communication Network Engineering. His current research interests include embedded architectures, hardware-/software co-design, real-time systems, and embedded systems engineering. Rolf Ernst is an IEEE Fellow and served as an ACM-SIGDA Distinguished Lecturer.