Real-Time Syst (2008) 39: 31–72
DOI 10.1007/s11241-007-9039-9

Sensitivity analysis of complex embedded real-time systems

Razvan Racu · Arne Hamann · Rolf Ernst

Published online: 22 November 2007

Abstract The robustness of an architecture to changes is a major concern in the design of efficient and reliable state-of-the-art embedded real-time systems. Robustness is important during design process to identify if and in how far a system can accommodate later changes or updates, or whether it can be reused in a next generation product. In the product life-cycle, robustness helps the designer to perform changes as a result of product updates, integration of new components and subsystems, or modifications of the environment. In this paper we determine robustness as a *performance reserve*, the slack in performance before a system fails to meet timing requirements. This is measured as *design sensitivity*. Due to complex component interactions, resource sharing and functional dependencies, one-dimensional sensitivity analysis might not cover all effects that modifications of one system property may have on system performance. One reason is that the variation of one property can also affect the values of other system properties requiring new approaches to keep track of simultaneous parameter changes. In this paper we present a framework for one-dimensional and multi-dimensional sensitivity analysis of real-time systems. The framework is based on compositional analysis that is scalable to large systems. The one-dimensional sensitivity analysis combines a binary search technique with a set of formal equations derived from the real-time scheduling theory. The multi-dimensional sensitivity analysis engine consists of an exact algorithm that extends the one-dimensional approach, and a stochastic algorithm based on evolutionary search techniques.

R. Racu (✉) · A. Hamann · R. Ernst
Institute of Computer and Communication Network Engineering, Technical University of Braunschweig, 38106 Braunschweig, Germany
e-mail: racu@ida.ing.tu-bs.de

A. Hamann
e-mail: hamann@ida.ing.tu-bs.de

R. Ernst
e-mail: ernst@ida.ing.tu-bs.de

Keywords Real-time · Embedded · Distributed systems · System-on-chip · Performance verification · Scheduling analysis · Compositional · Sensitivity analysis · Robustness · Slack · System properties · Binary search

1 Introduction

Due to complex system dependencies and global timing constraints, design sensitivity is hard to determine, especially in an environment with suppliers and integrators, where design data are often not fully available. The system designer can easily miss the system bottlenecks, and finding conforming system configurations might be a hard-to-accomplish task.

Starting from an architecture description, the application is mapped to components and communication links, thereby deriving first system timing properties. These include task execution times, communication times, models describing the execution requests of the single tasks, etc. These parameters together with other asserted specifications as for resource sharing, memory management, communication strategies, are used to build up the first performance model of the proposed system.

Figure 1 depicts a simplified automotive platform. In the figure, two main design layers can be identified: 1. the *application layer* shows the application elements (tasks) and the corresponding inter-dependencies; 2. the *architecture layer* contains the description of the hardware resources and the mapping of the application elements. The system in Fig. 1 consists of a set of electronic control units (ECU) that are connected via a distributed network containing several buses linked by one bridge. On the application level, a task graph describes data and timing dependencies between tasks. The tasks are characterized by a set of timing properties, like worst-case/best-case execution demands, activation rate, access times on shared resources, memory

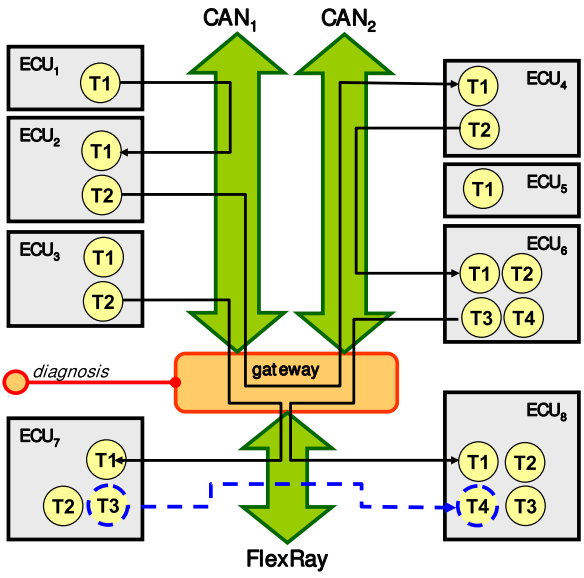


Fig. 1 Automotive platform

request times, etc. Additionally, the set of real-time constraints is explicitly formulated.

The heterogeneity of the scheduling environments, the complex hardware and software interactions, the large set of constraints, made system-level performance verification to become one of the major issues in the design of embedded systems. Since we assumed that the initial system configuration is not entirely or finally specified, variations of system properties may occur at any step during the design process. Therefore, the designer must be supplied with additional information concerning the robustness of different system configurations.

In this context, some issues need to be investigated in order to guarantee high system flexibility. At first, one must identify the system properties that may change during later design steps. These are usually task execution times, the parameters characterizing the activation models, communication volumes, the operational speed of the processing elements, the bandwidth of the communication resources, etc. For these properties it is necessary to determine the maximum variation of their values that is permitted by the set of constraints. We call *feasibility slack* the maximum variation of a system property value without violating the system feasibility. The *sensitivity* of a system property is inversely proportional to its feasibility slack, the smaller the available slack, the higher the sensitivity.

First sensitivity analysis approaches known from literature are restricted to simple example systems, like single-processor systems with purely periodic tasks and deadlines smaller than period. This is not a limitation of these approaches, but, at the time they came out, they perfectly fit in the real-time system requirements existing at that time. Meanwhile, the uni-processor system architectures were replaced by heterogeneous multi-processor platforms with large numbers of parameters and properties, and very complex dependencies and timing constraints. Therefore, most formal methods developed to measure the sensitivity of the system properties could not be adopted for such systems.

In this context, current sensitivity analysis methods must be able to cover systems with heterogeneous components in terms of scheduling, complex timing properties at system inputs and complex interactions between task, distributed timing constraints, etc. One should also require scalability, that is best achieved with novel compositional analysis approaches that will be discussed later.

Moreover, assuming only variations of one system property at a time might be insufficient. In most cases, the system parameters share common properties such that modifications of one parameter typically imply variations of other parameters, as well. An example is the dependency between the execution and communication tasks belonging to the same application, or the dependency between the execution demands of all tasks mapped on a resource and the speed of the resource. For the system depicted in Fig. 1, assume that the amount of data sent by task $\mathsf{T2}$ on ECU_4 increases. This leads to a larger communication time of the corresponding channel on CAN_2, but, due to data dependencies, results in a larger processing time of task $\mathsf{T1}$ on ECU_6. Therefore, considering only the variation of execution time of $\mathsf{T2}$ without accounting for variations of the communication time or variations of the execution time of the data-dependent task may lead to constraint violations, and thus, to faulty system configurations. Another example is the variation of the available clock rate of

ECU_6. Obviously, the execution times of the applications mapped on that resource are scaled consequently. We call the sensitivity analysis of such properties, *pseudo multi-dimensional sensitivity analysis*. Luckily, such scenarios can be formally reduced to the one-dimensional case and solved accordingly.

In a more general case, the dependency is not explicitly formulated or not known in detail, but it is still very helpful to see the influence of one system parameter on the sensitivity of another. Such an example is presented in Figure 1, by mapping additional applications on ECU_7 and ECU_8 and the corresponding inter-tasks communication on FlexRay network. Obviously, the variation of the available slacks of the three resources must be considered simultaneously.

This paper describes a framework for multi-dimensional sensitivity analysis of complex embedded systems with hard real-time requirements. Some parts of this paper extend the work presented in Racu et al. (2005) and Racu et al. (2006). Section 2 presents the existing work in the area of sensitivity analysis of real-time systems. Section 3 introduces the SymTA/S analysis model, used as a basis for system analysis and validation by the sensitivity analysis algorithms. The main part of this work describes the sensitivity analysis framework. Section 4 gives a structural view of the framework and the integration with the SymTA/S analysis engine. Section 5 presents the algorithms to compute the one-dimensional sensitivity of different system properties. Afterwards, in Sect. 6 we describe an exact and a stochastic algorithm for multi-dimensional sensitivity analysis. In Sect. 7 we carry out a set of experiments to show the applicability of the method and to compare the exact and stochastic methods for multi-dimensional sensitivity analysis. Finally, we draw some conclusions.

2 Related work

There is a large amount of work done in the area of sensitivity analysis. Event though the first publications give only a *Yes/No* answer regarding the schedulability of a particular system configuration, they derive the first equations to determine the border between feasible and infeasible system configurations. In fact, sensitivity analysis extends the former schedulability tests to determine all feasible configurations of particular system properties.

Liu and Layland (1973) defined the *utilization bound* on a resource that guarantees the schedulability of a task set under Rate Monotonic Scheduling (RMS) policy. The task model assumed contains periodically activated tasks with deadlines equal to period. The worst-case execution times of the tasks are known. The tasks do not share common resources, i.e. they do not block each other, and no synchronization between tasks is allowed. The authors proved that if all tasks are released at the same time—*the critical instant*—then each task experiences its worst-case response time. The schedulability test for the rate-monotonic algorithm is based on the critical instant concept. The schedulability condition is sufficient, but not necessary. The authors define the maximum utilization that guarantees the schedulability of a set of tasks under EDF. Since the utilization bound is 100%, the condition is not only sufficient, but necessary, as well.

Later on, Lehoczky et al. (1989) computed the *critical scaling factor* Δ^* as the largest possible scaling factor of the execution times of a given task set, that guarantees the schedulability of the task set under rate monotonic scheduling. The processor utilization associated with the critical scaling factor represents the *breakdown utilization* of the given task set. The approach is still restricted to periodically activated tasks with deadlines equal to period. In (Lehoczky 1990) he extends this approach for task sets with deadlines defined as the same constant fraction or multiple of task periods. He showed that tasks with deadlines larger than periods have higher utilization bounds than task sets with deadlines equal to periods.

The breakdown utilization is refined later by Katcher et al. (1993), considering the overhead of the scheduling kernel. He analyzed the costs of the scheduler in fixed priority scheduling algorithms for both, event- and timer-driven scheduling implementations. For the latter he defined the *optimal timer rate* as the rate that maximizes the breakdown utilization of a task set. The model used is still limited to periodically activated tasks with $D = T$.

Vestal formulated the first questions regarding the sensitivity of the schedulability analysis with respect to changes in the characteristics of the task set. In Vestal (1994), the author considered fixed priority periodic task sets with deadlines equal to period. He introduced *slack variables* into the inequalities proposed by Lehoczky et al. (1989) in order to transform them into equalities, and to obtain the maximum bound of each task compute time. To determine the slack of a task compute time, an equation is derived for each scheduling point within the response time of the task. A similar algorithm is derived to compute the slack of the execution times of a module, assuming that the tasks are defined as a linear combination of such application modules. Similar equations calculate bounds of task blocking times. Based on the obtained slack variables, a critical scaling factor is computed for the execution demands of a task set.

Yerraballi et al. (1993) used sensitivity analysis to capture three important design problems: (1) scalability—how much can a task execution time be increased maintaining its schedulability; (2) estimation—what is the maximum tolerable variance in task execution time; (3) portability—is schedulability analysis still valid if the target processor changes. He proposed a method to determine the minimum common scaling factor *sf* by which the task execution times can be scaled without invalidating the schedulability of the task set. The model used assumes periodic tasks with deadlines less or equal to the period. The algorithm determines for each task τ_i a scaling factor of its execution time and the execution times of the higher priority tasks, such that task τ_i completes its execution exactly at deadline. However, the computed scaling factor does not guarantee that all higher priority tasks meet their deadlines, as well. Therefore, the common scaling factor that guarantees the schedulability of the entire task set is the minimum of the scaling factors computed for each task.

Cottet and Babau (1996) provided a graphical approach to determine the system sensitivity to variations of task activation periods, task deadlines and release times. The analyses still assume task sets with deadlines less than periods.

The analysis given by Punnekkat et al. (1997) uses a combination of a binary search algorithm and a modified version of the response time equations proposed by Joseph and Pandya (1986) and Audsley et al. (1993). The proposed sensitivity metrics are build upon variation of task execution times, execution time scaling factors,

task periods and deadlines. Similar algorithms are derived for fault-tolerant real-time systems. The mathematical proof to demonstrate the applicability of the binary search on the set of analyzed parameters, and the evaluation of the algorithm complexity are not specified by the authors. In essence, the method is limited to components due to the underlying analysis, but the idea of a binary search can be adapted, as we will show later.

Regehr (2002) introduced a binary search algorithm in order to determine systems that, on one hand, offer a high flexibility with respect to task execution times and, on the other hand, guarantee a minimum number of threads required to run a given task set.

Bini et al. (2006) simplified the analysis proposed by Vestal (1994) and extended it to task periods. He used the feasibility equations introduced in (Lehoczky et al. 1989) and (Seto et al. 1998) to determine the system feasibility regions in the space of execution times and activation periods, respectively. The robustness of a system configuration with respect to a particular parameter is measured as the distance between the point representing the value of that parameter in the corresponding feasibility space and the feasibility bound. However, even if the proposed approach simplifies the equations proposed by Vestal (1994), the model used is still restricted to periodic task sets with deadlines equal to periods.

The MAST tool suite (Harbour et al. 2001) has implemented a sensitivity analysis engine to compute the execution-time slacks at resource level, transaction level and system level. MAST uses holistic analysis that may also cover different scheduling strategies in a system. However, we could not find any technical details that allows a direct comparison with the approach presented in this paper.

ETAS and Live Devices developed the *RTA-OSEK* (Live Devices, http://en.etasgroup.com/products/rta) product that has a built-in sensitivity analysis engine for embedded systems based on OSEK-OS.

The fact that most previous algorithms to sensitivity analysis consider only very restricted task models, with the task deadline less than the period, represents the main limitation of their applicability. One reason is that most real-time systems have defined global timing constraints, like end-to-end deadlines, rather than local constraints, like task deadlines, and the algorithms to derive local deadlines from global deadlines are not always optimal, and need to consider slack distribution over different components. A second reason is the huge complexity or sometimes the impossibility to adapt the algorithms for tasks with arbitrary deadlines. A third reason, at least as important as the previous two, is the limited scalability of the underlying analyses that only supports a fixed combination of scheduling strategies.

3 The SymTA/S approach

The sensitivity analysis method explained in this paper uses the performance analysis engine of SymTA/S (Project 2003). SymTA/S is a formal system-level timing analysis tool for heterogeneous system architectures. The application model of SymTA/S is described in Sect. 3.1. The core of SymTA/S is a technique to couple local scheduling analysis algorithms using event streams (Henia et al. 2005). 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 Sect. 3.2. The analysis composition using event streams is described in Sect. 3.3.

3.1 Application model

An application is modeled by a set of computation and communication tasks (application entities). The tasks are mapped to and executed on a set of processing and communication elements, representing the system architecture. Each task is characterized by its execution time interval, defined as the minimum and maximum time the task requires for a complete execution on the corresponding resource, assuming that no blocking occurs during execution.

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. 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 also assumes that input data is removed from the input FIFO at the end of one execution.

All activating events of a task are captured by an *event stream*. The behavior of the event streams is described using *event models*. The event models used in SymTA/S, called *standard event models*, are characterized by a common set of timing parameters. A detailed presentation of the standard event models is given in Sect. 3.2. Any timing dependency between the activations of two tasks is represented by an event stream emerging from one task and entering the dependent task. The real-time behavior is described by a set of timing constraints, like task deadlines, end-to-end deadlines, maximum jitter at system output, etc.

When multiple tasks share the same resource, two or more tasks may request the resource at the same time. To arbitrate request conflicts, each resource is associated with a scheduler which selects a task to execute out of the set of active tasks according to some scheduling policy. The scheduling analysis calculates worst-case and best-case task response times, i.e. the maximum and minimum times between task activation and task completion taking into account the effects of scheduling. Scheduling analysis guarantees that all observable response times will fall into the calculated [best-case, worst-case] interval. One may check the feasibility of the system by comparing the results of the scheduling analysis against the set of timing constraints. We say that a system is *feasible* if all constraints are fulfilled. Vice-versa, a system is *infeasible* if at least one constraint is violated.

Figure 2 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 scheduled according to a static priority preemptive policy. Src1 and Src2 are the sources of the external activating events at system inputs.

3.2 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 peri-*

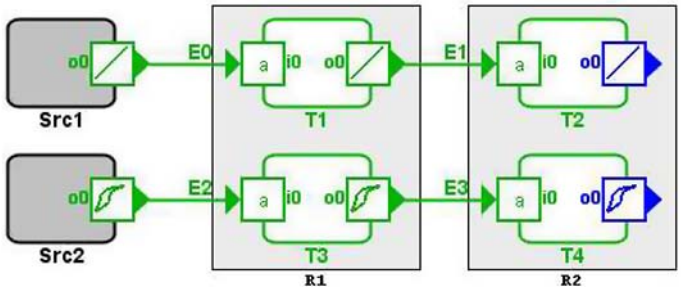


Fig. 2 System modeled with SymTA/S

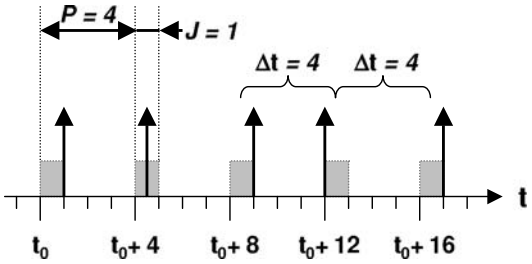


Fig. 3 Example of an event stream that satisfies the event model ($\mathcal{P} = 4,\ \mathcal{J} = 1$)

odic event model has one parameter $\mathcal{P}$, and states that each event 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 Fig. 3. 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. 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 two consecutive events within a burst.

Additionally, *sporadic* events are also common. 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. (2003).

3.3 Analysis composition

SymTA/S analysis engine is based on the compositional performance analysis methodology proposed by Richter (2004) and Chakraborty et al. (2003). The compositional performance analysis extends the basic concept of the purely sequential holistic analysis proposed by Tindell and Clark (1994), by iteratively performing task response time analysis and propagating the analysis jitter of a task to the input of the connected tasks. In contrast to the holistic analysis, the compositional analysis uses a common interface between components to systematically compose local scheduling analysis into system level analysis.

The system analysis alternates local scheduling analysis on each component with propagation of the timing information between components using the common interface. The common interface allows, contrary to holistic analysis, to easily apply the compositional analysis on arbitrarily large systems with complex dependencies between tasks and heterogeneous scheduling algorithms.

The composition is realized in SymTA/S by connecting the components using the standard event models presented in the previous section. The common interface is easily extensible and captures, besides the standard timing parameters, like period, jitter or minimum inter-arrival distance, additional timing information, like activation offsets between tasks, execution modes, etc. Recent work (Perathoner 2006) shows that the compositional analysis technique performs at least as well as the holistic analysis technique, but is at the same time scalable.

Similar to SymTA/S, the RTC toolbox (Wandeler and Thiele 2006) uses Real-Time Calculus (Thiele et al. 2000) as method for the common interface between components.

The holistic analysis has been later extended and improved by Palencia and Harbour (1998) and Redell and Trngren (2002) to consider more complex task sets with static and dynamic offsets and application cycles. Later work in the MAST framework also used iteration, but without formally defining interface models that provide the formal power of the compositional approach, and enables modularity, composition and scalability.

3.3.1 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 worst-case and best-case task response times, represents the component internal jitter, also called response time jitter. The internal jitter is added to the jitter of activating event model, yielding the jitter of the output event model (Henia et al. 2006):

$$\mathcal{J}_i^{\text{out}} = \max_{\forall k}(\mathcal{J}_{i,k}^{\text{in}} + R_{i,k}^{w}) - R_i^{b} \tag{1}$$

where $\mathcal{J}_{i,k}^{\text{in}}$ and $R_{i,k}^{w}$ represent the input jitter delay and the worst-case response time of the k-th job of task τ_i. R_i^b represents the best-case response time of task τ_i (Redell and Sanfridson 2002).

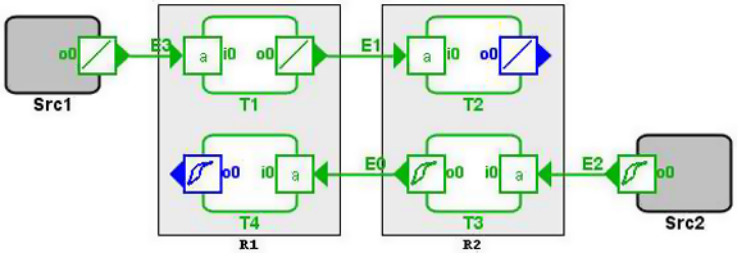


Fig. 4 Example of a system with cyclic scheduling dependency

Note that, if the calculated output event model has a jitter larger 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. The order in which the events arrive at input is certainly preserved at output due to the FIFO mechanism presented in Sect. 3.1. In reality, two consecutive output events cannot follow closer than the best-case response time of the producer task. This is captured by the value of the *minimum distance* parameter d^-.

3.3.2 Analysis composition using standard event models

In the following, the compositional analysis approach is explained using the system example in Fig. 2. 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 (Sect. 3.3.1). In the second phase, all output event models are propagated. The output event models become the activating event models for T2 and T4. Now, the scheduling analysis of R2 can be performed since all activating event models are known.

However, sometimes might be impossible to directly perform the system level analysis as explained above. Such a complex analysis scenario occurs for the system presented in Fig. 4. The system consists of 2 resources, R1 and R2, each containing 2 tasks. Initially, only the activating event models of T1 and T3 are known. At this point the system cannot be analyzed, because the activating event models of one task on each resource are missing. This means, the response times on R1 need to be calculated to be able to analyze R2. On the other hand, R1 cannot be analyzed before analyzing R2. We call this problem a *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. The global analysis loop is depicted in Fig. 5. A global analysis step consists of two main phases (Richter et al. 2003). In the first phase local scheduling analysis is performed for each resource and the output event models are calculated (Sect. 3.3.1).

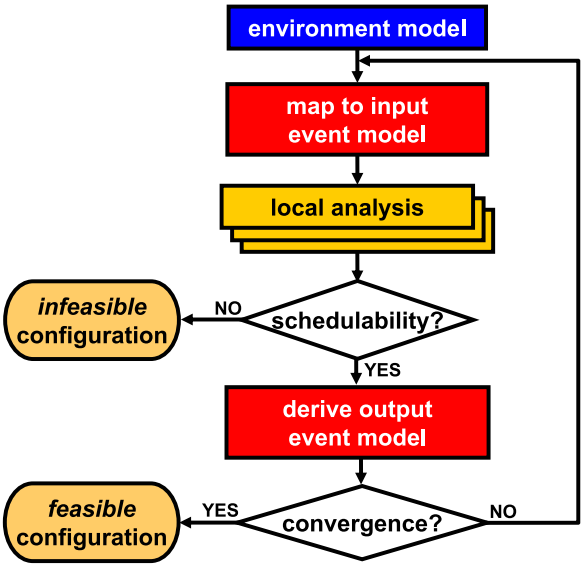


Fig. 5 System analysis loop in SymTA/S

During the second phase all output event models are propagated. Then, it is checked if the first phase has to be repeated because of obsolete parameters of the activating event models. This happens when newly propagated output event models are different from the output event models propagated in the previous analysis step. Analysis completes either when no event model changes change during propagation , or when at least one abort condition, e. g. the violation of a timing constraint, has been reached.

4 Sensitivity analysis in SymTA/S

In this section we introduce the sensitivity analysis framework implemented in SymTA/S. Section 4.1 presents the sensitivity analysis flow and the analysis modules. Thereafter, in Sect. 4.2, are described the sensitivity analysis parameters. Finally, Sect. 4.3 explains the binary search algorithm and formulates the assumptions required for its proper applicability on different system properties.

4.1 Sensitivity analysis loop

The diagram presented in Fig. 6 shows the components of the sensitivity analysis framework and the corresponding analysis flow. The user selects (Step 1) from the pool of system properties, a set of *sensitivity tuples* on which the sensitivity analysis is performed.

Definition 1 A property tuple is represented by a vector $\Gamma = (p_1, \ldots, p_n)$, where $n \geq 1$ and $p_i, 1 \leq i \leq n$ is a system property. A property tuple may contain system properties of different types.

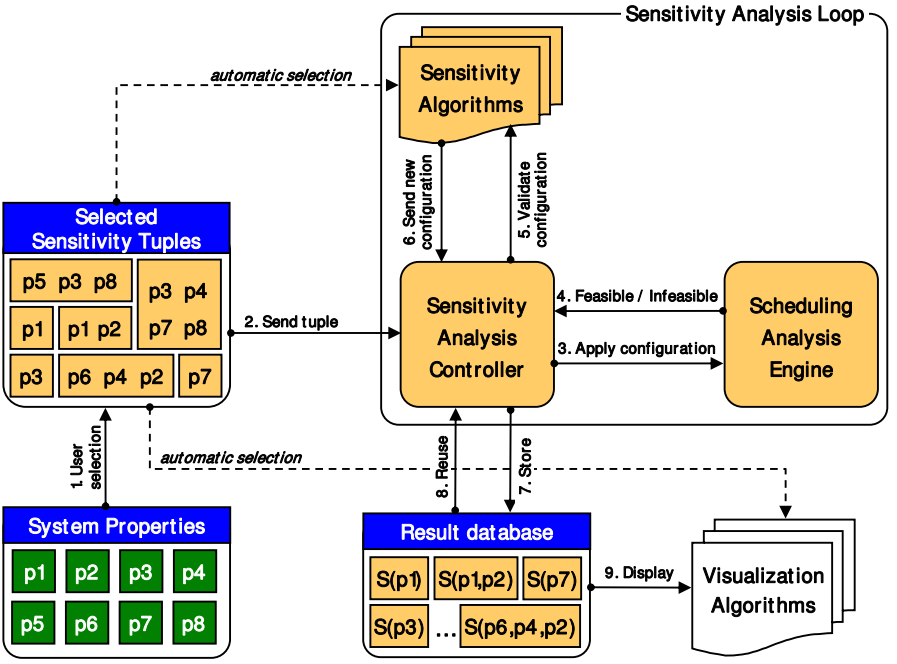


Fig. 6 Sensitivity analysis framework

The type of a system property is also referred as sensitivity analysis parameter (see Sect. 4.2). The length of a property tuple, i.e the value of n in Definition 1, determines the number of dimensions of the search space investigated by the sensitivity analysis (see Sects. 5 and 6).

Definition 2 A sensitivity tuple is represented by a data object $\mathcal{T} = (\Gamma, \mathcal{M})$, where Γ is a property tuple and $\mathcal{M}$ represents a set of sensitivity objectives defined for Γ.

The sensitivity tuples are sent one by one to the sensitivity analysis controller (Step 2). The *sensitivity analysis controller* is responsible for directing the flow of information between modules. The *sensitivity analysis loop* contains the sensitivity analysis controller, the scheduling analysis engine and the sensitivity algorithms. At first iteration, the sensitivity analysis controller transmits (Step 3) the property values of the selected sensitivity tuple to the scheduling analysis engine. It receives back (Step 4) the status of the analyzed system: *feasible* or *infeasible*. Based on the received status, the search space is defined and sent (Step 5) to the analysis algorithm. The algorithm selects a value in the search space and sends it back to analysis controller (Step 6). The steps 3 to 6 are repeated until the algorithm finds the desired value. Note that, the search space is defined in the first iteration by the sensitivity controller and adapted during each iteration by the analysis algorithm. When the analysis completes, the results are stored (Step 7) in the *result database*. The results

are displayed (Step 9) by demand using visualization algorithms specific to each sensitivity tuple.

The sensitivity algorithms and the visualization algorithms are automatically selected, depending on the sensitivity tuple which is currently analyzed. Before first analysis iteration or sometimes during iterations, the sensitivity analysis controller might access existing information in the result database (Step 8).

The scheduling analysis is performed using the compositional system level analysis introduced in Sect. 3. The communication between the scheduling analysis engine and the sensitivity analysis controller is implemented using a client/server interface. The communication between the sensitivity analysis controller and the other modules is realized via function calls.

4.2 Parameters and objectives

In the context of this paper we call *system property* any system characteristic dynamically defined by the specification of the applications or the organization of the system architecture. In other words, a system property is usually not directly defined or controlled by system designer. Modification of the application specification and/or system architecture structure during design may lead to unexpected changes of the system properties. Following system properties are likely to be changed during design:

1. Task execution time: best-case, worst-case
2. Resource speed factor
3. Activation period ($\mathcal{P}$)
4. Activation jitter ($\mathcal{J}$)
5. Minimum inter-arrival distance (d^-)
6. Volume of communication
7. Traffic shaping parameters

These parameters have associated following sensitivity analysis objectives:

1. Minimize best-case execution time, maximize worst-case execution time
2. Minimize the resource speed factor
3. Minimize the activation period
4. Maximize activation jitter
5. Reduce the minimum inter-arrival distance
6. Maximize the communication volume
7. Minimize the buffer size required by a traffic shaper

The difference between the initial value of a property and the value obtained by the sensitivity analysis represents the *feasibility slack* of that property. If the initial system configuration is feasible, the feasibility slack of a parameter is a non-negative number. Otherwise, the feasibility slack is a negative number.

4.3 Binary search technique

As mentioned in Sect. 2, different approaches were proposed for the sensitivity analysis of different system parameters. However, most of them are applicable only on single resources, as they are limited by local timing constraints, like tasks deadlines.

Due to complex inter-resource dependencies and global timing requirements, the sensitivity analysis techniques used at resource level can not be straight-forward adapted at system level, as this implies huge effort and less flexibility.

An alternative is a search algorithm that can be easily applied on various search spaces. *Binary search* is a fast and simple search technique used to determine a specific value within an *ordered* set of data. Since the timing metrics used by the feasibility analysis monotonically change with the variation of the system properties subject to sensitivity analysis, the binary search algorithm can be safely applied in order to determine the feasibility slack of these properties. A proof of correctness is provided in Sect. 5.

Since most system properties subject to changes can be varied within a continuous range of values, appropriate conditions must be defined to limit the number of iterations of the binary search algorithm. The search is aborted when the size of the search interval becomes smaller than the search resolution, denoted ϵ. The value of ϵ is defined depending on the definition range of the sensitivity parameter.

In order to apply the binary search technique, the search space must be bounded. One bound is determined by the initial value of the investigated system property. Depending on the feasibility of the initial configuration of the system, the second bound is calculated differently. For the *feasible* case, the second bound can be defined using additional information about the constraints imposed for a particular system. For example, when investigating the execution time of a task, the maximum bound of the search space is defined by the value of the execution time corresponding to 100% resource utilization. Any value larger than this value definitely leads to an infeasible configuration. For the *infeasible* case, the second bound is defined by the value of the system property which generally minimizes the system load. Hence, in case of execution time sensitivity analysis, this bound is set to 0. Notice that, depending on the architecture configuration, the scheduling techniques used and the set of timing constraints, there is no guarantee that a feasible configuration exist, even when the parameter is set to its minimum value.

The complexity of the algorithm can be calculated using the value of the search resolution and the size of the search space. If we denote by L the size of the search space, then the total number of discrete points within this space is $\lceil \frac{L}{\epsilon} \rceil$. Since binary search is a logarithmic algorithm with the execution complexity $O(\log n)$, the total number of iterations N required to find the desired value is

$$N = 1 + \log_2 \left\lceil \frac{L}{\epsilon} \right\rceil. \tag{2}$$

Algorithm 1 shows the implementation of the binary search technique to determine the feasibility slack of a system property. The initial property value is $\mathcal{V}^{\text{init}}$. The search interval is defined depending on the feasibility status of the initial system configuration. Hence, if the initial system configuration is feasible, the search interval is determined by the initial property value and high_feasible (lines 1–3). Otherwise, the search interval is determined by the initial property value and low_infeasible (lines 4–6).

At each iteration, the value of the analyzed property is set to the middle value of the search interval (lines 9–10), and the search interval is adjusted accordingly

Algorithm 1 Binary search

INPUT: initial value: $\mathcal{V}^{\text{init}}$

search resolution: ϵ

OUTPUT: feasibility slack: $\Delta\mathcal{S}^{\max}$

1: **if** (*isFeasible*()) **then**
2: $low = \mathcal{V}^{\text{init}}$
3: $high =$ high_feasible
4: **else**
5: $low =$ low_infeasible
6: $high = \mathcal{V}^{\text{init}}$
7: **end if**
8: **while** $|high - low| > \epsilon$ **do**
9: $middle = (high + low)/2$
10: set $\mathcal{V} = middle$
11: **if** (*isFeasible*()) **then**
12: $low = middle$
13: **else**
14: $high = middle$
15: **end if**
16: **end while**
17: set $\mathcal{V} = \mathcal{V}_{\text{init}}$
18: $\Delta\mathcal{S}^{\max} = low - \mathcal{V}_{\text{init}}$

(lines 11–15). Procedure *isFeasible*() performs a global system analysis and return the system status.

The algorithm terminates when the size of the search interval becomes smaller than the predefined search resolution, ϵ. The feasibility slack, $\Delta\mathcal{S}^{\max}$ is defined as the *maximum positive variation* of the property value, in case of *feasible* initial configuration, and as *minimum negative variation* of the property value, in case of *infeasible* initial configuration.

5 One-dimensional sensitivity analysis

5.1 Task execution time

In this section we present details of the binary search algorithm to determine the feasibility slack of the worst-case execution time of a task.

Proof of correctness To be able to perform a binary search we need to prove that the system performance metrics (response times, end-to-end latencies, jitters) monotonically change with the variation of the execution time of the analyzed task.

Assuming static priority preemptive scheduling, the worst-case response time R_i^w of a task τ_i can be determined using the response time analysis presented in Audsley et al. (1993)

$$R_i^{(n+1)} = C_i + \sum_{\forall j \in HP(i)} \left\lceil \frac{R_i^{(n)}}{\mathcal{P}_j} \right\rceil \cdot C_j; \qquad R_i^{(0)} = C_i \tag{3}$$

where $HP(i)$ contains the indices of all tasks with priorities higher than τ_i. The worst-case response time R_i^w, is iteratively calculated using (3), until $R_i^{(n+1)} = R_i^{(n)}$.

From (3) we observe that R_i^w depends on the worst-case execution time and the period of τ_i and of all higher priority tasks. Since the activation periods are assumed fixed, we can express R_i^w as follows

$$R_i^w = f(C_k), \quad \forall k \geq i. \tag{4}$$

Obviously, f is monotonic. Similar observations can be made for the task response times functions corresponding to other arbitration policies.

Equation (3) can be replaced by more complex equations (Tindell and Clark 1994; Palencia and Harbour 1998) to capture blocking times between tasks, activation jitters or activation offsets. However, the monotonic behavior of the response time with respect to execution time is still preserved when using these equations.

Furthermore, since the activation jitter $\mathcal{J}_i^{\text{in}}$ and the best-case response time R_i^b are not functions of C_i, it follows from (1) that the output jitter $\mathcal{J}_i^{\text{out}}$ is also a monotonic function of C_i. An increasing jitter may lead to an increasing response time of the connected task.

Since the maximum latency of a path strongly depends on the worst-case response times of all tasks along that path, the path latency is a monotonic function of the execution times of all tasks within the path.

Algorithm implementation If the initial system configuration is *feasible*, then the search interval is determined by the current worst-case execution time and the value of the execution time corresponding to the maximum utilization of the resource.

The current utilization of the resource is (Liu and Layland 1973):

$$U_{\text{initial}} = \frac{C_1}{\mathcal{P}_1} + \frac{C_2}{\mathcal{P}_2} + \cdots + \frac{C_i}{\mathcal{P}_i} + \cdots + \frac{C_n}{\mathcal{P}_n}. \tag{5}$$

If we increase the execution time of task τ_i by ΔC_i, then the new utilization

$$U_{\text{new}} = \frac{C_1}{\mathcal{P}_1} + \cdots + \frac{C_i + \Delta C_i}{\mathcal{P}_i} + \cdots + \frac{C_n}{\mathcal{P}_n} \tag{6}$$

which must be smaller than the maximum allowed utilization, $U_{\max}$

$$U_{\text{new}} \leq U_{\max}. \tag{7}$$

If not defined by user, $U_{\max}$ is set by default to 1. Any utilization above 100% means that the resource is overloaded and the selected configuration is infeasible.

The maximum variation of C_i is then

$$\Delta C_i^{\max} = \mathcal{P}_i \cdot (U_{\max} - U_{\text{initial}}). \tag{8}$$

Therefore, the value of high_feasible is equal to

$$\text{high\_feasible} = C_i + \Delta C_i^{\max} \tag{9}$$

and the range investigated by the binary search algorithm is $[C_i; C_i + \Delta C_i^{\max}]$.

If the initial system configuration is *infeasible*, the search interval is determined by the initial execution time and

$$\text{low\_infeasible} = 0. \tag{10}$$

There is no guarantee that a value leading to a feasible configuration is found. Therefore, to speed up the search, the algorithm firstly investigates the value 0. If the configuration corresponding to this value is infeasible, then the search is aborted. Otherwise, the binary search is performed.

5.2 Resource speed factor

This section gives details on the binary search algorithm to determine the minimum speed factor of a resource. The execution times of all tasks mapped on a resource are inverse ratios of resource speed, i.e. reducing the resource speed by a factor sf will increase the execution demands of all tasks mapped on that resource by the same factor.

Proof of correctness Since the variation of the resource speed factor results in the variation of the execution times of the tasks mapped on that resource, and the monotonic behavior of the system timing metrics with respect to task execution times has been already proved in Sect. 5.1, a similar proof can be derived for a common scaling factor of a set of execution times.

Algorithm implementation If we scale the resource speed by sf, (6) becomes:

$$U_{\text{new}} = \frac{1}{\text{sf}} \sum_{i=1}^{n} \frac{C_i}{\mathcal{P}_i}. \tag{11}$$

The new resource utilization obviously must be smaller than the maximum utilization allowed for that resource. Hence, substituting (5) and (11) in (7) we obtain:

$$\text{sf}_{\min} \geq \frac{U_{\text{initial}}}{U_{\max}}. \tag{12}$$

If the initial system configuration is *feasible*, the search space is determined by the initial speed factor (usually 1) and the value of $\text{sf}_{\min}$ in the above equation. Hence,

$$\text{high\_feasible} = \text{sf}_{\min}. \tag{13}$$

Similarly, one can define a minimum resource utilization and the appropriate value for the speed factor of that resource. This factor corresponds to the maximum clock frequency at which the resource can operate. The maximum speed factor is computed as follows:

$$\mathsf{sf}_{\max} \leq \frac{U_{\text{initial}}}{U_{\min}}. \tag{14}$$

If the initial system configuration is *infeasible*, the search space is determined by the current speed factor value (usually 1) and the value of $\mathsf{sf}_{\max}$. Hence,

$$\mathsf{low\_infeasible} = \mathsf{sf}_{\max}. \tag{15}$$

5.3 Activation period

This section presents details of the binary search algorithm to determine the slack of a task activation period. Our application model assumes that the tasks have *non-conditional output*, such that the data rate at task input is preserved at task output, i.e. the period of the completion events is equal to the period of the activation events. Therefore, all tasks traversed by the same application path have the same activation period. The period of an application path is modeled by a source task that contains the timing specifications either at system input or internally generated by the system. The analysis determines the variation of the period specified by a source task.

Proof of correctness According to (3), if we assume that the task execution demands are constant, the worst-case response time of the task monotonically increases with the decrease of the activation period of any task interfering with the analyzed task (including the task itself). Again, the increasing worst-case response time leads to a higher output jitter and, consequently, to a higher worst-case response time of the connected tasks. Thus, the latencies of the application paths within the system increase, as well. Hence, the system timing metrics are monotonic functions of the activation periods of all tasks in the system.

Algorithm implementation First, we need to determine the bounds of the search space. If we consider the variation of a source task period we have to check the variation of the load on all resources traversed by the application path connected to that source task. Let $\mathcal{A}$ be the application path and $\mathcal{R}$ the set of hardware resources traversed by $\mathcal{A}$. The utilization of any resource R in $\mathcal{R}$ can be calculated using (5). Assuming that the period of $\mathcal{A}$ changes, the load on R can be divided into two parts: one is static and the other one changes dynamically with the period:

$$U^R_{\text{initial}} = U^R_{\text{static}} + U^R_{\text{dyn}} = \sum_i \frac{C_i}{\mathcal{P}_i} + \sum_j \frac{C_j}{\mathcal{P}_{\mathcal{A}}}, \quad \tau_i \notin \mathcal{A},\ \tau_j \in \mathcal{A},\ R \in \mathcal{R}. \tag{16}$$

$\mathcal{P}_{\mathcal{A}}$ represent the period at the input of path $\mathcal{A}$. The tasks τ_j are mapped on R and contained in path $\mathcal{A}$. Obviously, a path may contain more than one task mapped on the same resource.

If we decrease the activation period $\mathcal{P}_\mathcal{A}$ by $\Delta\mathcal{P}_\mathcal{A}^R$, the new utilization of R becomes

$$U_{\text{new}}^R = \sum_i \frac{C_i}{\mathcal{P}_i} + \frac{\sum_j C_j}{\mathcal{P}_\mathcal{A} - \Delta\mathcal{P}_\mathcal{A}^R}. \tag{17}$$

If we substitute (16) and (17) in (7) we obtain:

$$\mathcal{P}_\mathcal{A}^{R,\min} = \frac{\sum_j C_j}{U_{\max} - U_{\text{initial}} + \frac{\sum_j C_j}{\mathcal{P}_\mathcal{A}}}. \tag{18}$$

For each resource R in $\mathcal{R}$ (18) calculates the minimum of $\mathcal{P}_\mathcal{A}$ leading to a resource utilization equal to $U_{\max}$. In order to keep the load on all resources below or equal to $U_{\max}$, the absolute minimum is determined by:

$$\mathcal{P}_\mathcal{A}^{\min} = \max_k(\mathcal{P}_\mathcal{A}^{R,\min}), \quad \forall R_k. \in \mathcal{R}. \tag{19}$$

If the initial system configuration is *feasible*, the value of high_feasible is equal to the value of $\mathcal{P}_\mathcal{A}^{\min}$ in (19).

If the initial system configuration is *infeasible*, the search space is bounded by the initial period and the value corresponding to a minimum user-defined load generated by the tasks on the resources in $\mathcal{R}$ (see (20)).

$$\mathcal{P}_\mathcal{A}^{\max} = \frac{\max_k(\sum_j C_j)}{U_{\min}}, \quad \forall R_k \in \mathcal{R}. \tag{20}$$

Hence, low_infeasible is equal to the value of $\mathcal{P}_\mathcal{A}^{\max}$ in (20).

6 Multi-dimensional sensitivity analysis

In this section we introduce two multi-dimensional analysis techniques to compute the system robustness assuming simultaneous variations of a set of system properties. Sect. 6.2 describes an exact algorithm that combines a user-controlled search algorithm with a binary search technique. The method is optimized for the analysis of simultaneous variations of two system properties. For the analysis of simultaneous variations of more than two system properties, we present a stochastic approach based on evolutionary algorithms.

6.1 Definitions and notations

Definition 3 A vector $(\mathcal{V}_{p_1}, \ldots, \mathcal{V}_{p_n})$ represents a *sensitivity point* if any value of p_i other than $\mathcal{V}_{p_i}$, defined according to the sensitivity objective of p_i, leads to an infeasible system configuration.

Definition 4 The *sensitivity front* $\mathcal{F}_{(p_1,\ldots,p_n)}$ is represented by all sensitivity points $(\mathcal{V}_{p_1}, \ldots, \mathcal{V}_{p_n})$, where $\mathcal{V}_{p_i} \in [\mathcal{V}_{p_i}^{\text{init}}; \mathcal{V}_{p_i}^{\text{ext}}]$. Thereby, $\mathcal{V}_{p_i}^{\text{init}}$ represents the value of p_i for the initial system configuration, and $\mathcal{V}_{p_i}^{\text{ext}}$ represents the extreme value of p_i determined using one-dimensional sensitivity analysis.

Algorithm 2 *analyzePropertyTuple*

INPUT: (p_b, p_t), the property tuple to be analyzed.

OUTPUT: $\mathcal{M}_{(p_b,p_t)}$, a finite subset of the sensitivity front.

1: get $\mathcal{V}_{p_b}^{\text{init}}$, the initial value of p_b;
2: compute $\mathcal{V}_{p_b}^{\text{ext}}$, the largest variation of p_b's value;
3: $\mathcal{S}_{p_t}^{\text{init}} = computeSlack(p_t, \mathcal{V}_{p_b}^{\text{init}})$;
4: $\mathcal{S}_{p_t}^{\text{ext}} = computeSlack(p_t, \mathcal{V}_{p_t}^{\text{ext}})$;
5: insert $(\mathcal{V}_{p_b}^{\text{init}}, \mathcal{S}_{p_t}^{\text{init}})$ into $\mathcal{M}_{(p_b,p_t)}$;
6: insert $(\mathcal{V}_{p_b}^{\text{ext}}, \mathcal{S}_{p_t}^{\text{ext}})$ into $\mathcal{M}_{(p_b,p_t)}$;
7: $depthSearch(1, \mathcal{V}_{p_b}^{\text{init}}, \mathcal{V}_{p_b}^{\text{ext}}, \mathcal{M}_{(p_b,p_t)})$;
8: return $\mathcal{M}_{(p_b,p_t)}$;

6.2 Search-based analysis

This section presents a search-based algorithm for the two-dimensional sensitivity analysis. The algorithm can be also extended for n-dimensions, however the complexity exponentially increases with the number of dimensions.

The main idea of Algorithm 2 is a systematic selection of the values for the base parameter, p_b. For each selected value of p_b is calculated the feasibility slack of the target parameter p_t, using the one-dimensional sensitivity analysis algorithms presented in Sect. 5. The values of the base parameter ($\mathcal{V}_{p_b}$) are searched within a closed interval defined by its initial value ($\mathcal{V}_{p_b}^{\text{init}}$) and the extreme value allowed for that parameter ($\mathcal{V}_{p_b}^{\text{ext}}$). The latter is determined by applying the one-dimensional sensitivity analysis (line 2) (Racu et al. 2005). Lines 3 and 4 return the available slacks of the target parameter corresponding to the extreme values of the base parameter. The sensitivity point $(\mathcal{V}_{p_b}, \mathcal{S}_{p_t})$ is then added to $\mathcal{M}_{(p_b,p_t)}$. Obviously, $\mathcal{M}_{(p_b,p_t)}$ is a finite subset of the sensitivity front $\mathcal{F}_{(p_b,p_t)}$.

The function $computeSlack(p_t, \mathcal{V}_{p_b})$ calculates the feasibility slack of p_t considering the system configuration with $\mathcal{V}_{p_b}$ defined as value for property p_b. The function *depthSearch* at line 7 is explained in detail in next section.

6.2.1 Depth-search

The function *depthSearch* described in Algorithm 3 divides the search interval in half, and computes the feasibility slack of p_t for the system configuration assuming the value of p_b equal to the middle value of the search interval. The sensitivity point determined by this value and the corresponding slack of p_t is inserted into $\mathcal{M}_{(p_b,p_t)}$. Function *depthSearch* is recursively applied to the left-half and the right-half intervals. The recursion terminates if at least one abort condition defined in function *abortCheck*() is satisfied. The abort conditions are thoroughly presented in Sect. 6.2.2.

Algorithm 3 *depthSearch*

INPUT: depth, the current search depth;
$\mathcal{V}_{p_b}^{\text{left}}$ and $\mathcal{V}_{p_b}^{\text{right}}$, the bounds of the search interval;
$\mathcal{M}_{(p_b,p_t)}$ the set of analyzed points.

1: **if** *abortCheck*(depth, $\mathcal{V}_{p_b}^{\text{left}}$, $\mathcal{V}_{p_b}^{\text{right}}$, $\mathcal{S}_{p_t}^{\text{left}}$, $\mathcal{S}_{p_t}^{\text{right}}$) **then**
2: stop search;
3: **end if**
4: $\mathcal{V}_{p_b}^{\text{mid}} = (\mathcal{V}_{p_b}^{\text{left}} + \mathcal{V}_{p_b}^{\text{right}})/2$;
5: $\mathcal{S}_{p_t}^{\text{mid}} = computeSlack(p_t, \mathcal{V}_{p_b}^{\text{mid}})$;
6: insert $(\mathcal{V}_{p_b}^{\text{mid}}, \mathcal{S}_{p_t}^{\text{mid}})$ into $\mathcal{M}_{(p_b,p_t)}$;
7: *depthSearch*(depth + 1, $\mathcal{V}_{p_b}^{\text{left}}$, $\mathcal{V}_{p_b}^{\text{mid}}$, $\mathcal{M}_{(p_b,p_t)}$);
8: *depthSearch*(depth + 1, $\mathcal{V}_{p_b}^{\text{mid}}$, $\mathcal{V}_{p_b}^{\text{right}}$, $\mathcal{M}_{(p_b,p_t)}$);

6.2.2 Smart step

The abort conditions consist of two user controlled tests and one abort test determined by the type of the analyzed parameters and their behavior. The user can define the maximum depth of the search algorithm ($\text{depth}_{\max}$) that represents the largest number of halving-levels performed by the *depthSearch* function. If $\text{depth}_{\max}$ is met, the search is aborted (lines 1 and 2 in Algorithm 4). For a depth equal to $\text{depth}_{\max}$ the maximum number of analyzed points is $2^{\text{depth}_{\max}}$.

A second parameter controlling the search algorithm is the minimum size of the search interval. If the resolution of the search space becomes smaller than ρ, no further points are investigated (lines 3 and 4 in Algorithm 4). This test dominates the previous condition such that, for initially small search domains the number of investigated points can be significantly reduced.

The third abort condition is considered only if the relation between the analyzed parameters is monotonic in the complete analyzed domain. In such a case, if the slacks of the target parameter corresponding to the values of the investigated interval bounds are equal, no further search is required in that interval.

This condition can be safely applied for the sensitivity analysis of task execution times or channel communication times, as proved by Lemma 1. In Racu et al. (2005), we proved the monotonic relation between the execution time interval of a task and the available system slack.

Lemma 1 *Consider two system properties $p_{\mathsf{T_b}}$ and $p_{\mathsf{T_t}}$ representing the worst-case execution times of two tasks, as the base and the target parameters of the two-dimensional sensitivity analysis. The initial execution time of the base task is* $[BCET_{\mathsf{T_b}}; WCET_{\mathsf{T_b}}]$. *Assume that for two values of $WCET_{\mathsf{T_b}}$, $\mathcal{V}_{p_{\mathsf{T_b}}}^{\text{left}}$ and $\mathcal{V}_{p_{\mathsf{T_b}}}^{\text{right}}$, the target parameter has the values $\mathcal{S}_{p_{\mathsf{T_t}}}^{\text{left}}$ and $\mathcal{S}_{p_{\mathsf{T_t}}}^{\text{right}}$, such that*

$$\mathcal{S}_{p_{\mathsf{T_t}}}^{\text{left}} = \mathcal{S}_{p_{\mathsf{T_t}}}^{\text{right}}. \tag{21}$$

Algorithm 4 *abortCheck*

INPUT: depth, the current search depth;

$\mathsf{depth}_{\max}$, the maximum depth;

$\mathcal{V}_{p_b}^{\text{left}}$ and $\mathcal{V}_{p_b}^{\text{right}}$, the bounds of the search interval;

$\mathcal{S}_{p_t}^{\text{left}}$ and $\mathcal{S}_{p_t}^{\text{right}}$, the available slack of p_t corresponding to $\mathcal{V}_{p_b}^{\text{left}}$ and $\mathcal{V}_{p_b}^{\text{right}}$;

ρ, the minimum size of the search interval.

OUTPUT: a boolean value

1: **if** ($\mathsf{depth} > \mathsf{depth}_{\max}$) **then**
2: return *true*;
3: **else if** $|\mathcal{V}_{p_b}^{\text{left}} - \mathcal{V}_{p_b}^{\text{right}}| < \rho$ **then**
4: return *true*;
5: **else if** ($\mathcal{S}_{p_t}^{\text{left}} = \mathcal{S}_{p_t}^{\text{right}}$) **then**
6: return *true*;
7: **else**
8: return *false*;
9: **end if**

Then, for an arbitrary value $\mathcal{V}_{p_{\mathsf{T_b}}}^{\text{in}}$ *within the interval* $[\mathcal{V}_{p_{\mathsf{T_b}}}^{\text{left}}; \mathcal{V}_{p_{\mathsf{T_b}}}^{\text{right}}]$, *the following equation is valid*:

$$\mathcal{S}_{p_{\mathsf{T_t}}}^{\text{in}} = \mathcal{S}_{p_{\mathsf{T_t}}}^{\text{left}} = \mathcal{S}_{p_{\mathsf{T_t}}}^{\text{right}} \tag{22}$$

where $\mathcal{S}_{p_{\mathsf{T_t}}}^{\text{in}}$ *represents the slack of the worst-case execution time of* $\mathsf{T_t}$ *corresponding to the worst-case execution time of* $\mathsf{T_b}$ *equal to* $\mathcal{V}_{p_{\mathsf{T_b}}}^{\text{in}}$.

Proof Assume that

$$\mathcal{S}_{p_{\mathsf{T_t}}}^{\text{in}} > \mathcal{S}_{p_{\mathsf{T_t}}}^{\text{left}} \quad \text{and} \quad \mathcal{S}_{p_{\mathsf{T_t}}}^{\text{in}} > \mathcal{S}_{p_{\mathsf{T_t}}}^{\text{right}}. \tag{23}$$

Since the algorithm presented in Sect. 6.2 investigates only the upper bound of the execution time interval of $\mathsf{T_b}$, i.e. the $WCET_{\mathsf{T_b}}$ values, by computing the slack $\mathcal{S}_{p_{\mathsf{T_t}}}^{\text{in}}$ corresponding to $\mathcal{V}_{p_{\mathsf{T_b}}}^{\text{in}}$, it is guaranteed that $\mathcal{S}_{p_{\mathsf{T_t}}}^{\text{in}}$ is valid for all values $\mathcal{V}_{p_{\mathsf{T_b}}}$ within the interval $[BCET_{\mathsf{T_b}}; \mathcal{V}_{p_{\mathsf{T_b}}}^{\text{in}}]$.

Because $[BCET_{\mathsf{T_b}}; \mathcal{V}_{p_{\mathsf{T_b}}}^{\text{left}}] \subset [BCET_{\mathsf{T_b}}; \mathcal{V}_{p_{\mathsf{T_b}}}^{\text{in}}]$, (23) says that there is at least one value in interval $[BCET_{\mathsf{T_b}}; \mathcal{V}_{p_{\mathsf{T_b}}}^{\text{in}}]$ that determines a slack $\mathcal{S}_{p_{\mathsf{T_t}}}$ smaller than the minimum slack computed for that interval, that obviously can not be valid.

A similar proof can be carried out for the case

$$\mathcal{S}_{p_{\mathsf{T_t}}}^{\text{in}} < \mathcal{S}_{p_{\mathsf{T_t}}}^{\text{left}} \quad \text{and} \quad \mathcal{S}_{p_{\mathsf{T}-\mathsf{t}}}^{\text{in}} < \mathcal{S}_{p_{\mathsf{T_t}}}^{\text{right}}. \tag{24}$$

Since $[BCET_{\mathsf{T_b}}; \mathcal{V}^{\text{in}}_{p_{\mathsf{T_b}}}] \subset [BCET_{T_b}; \mathcal{V}^{\text{right}}_{p_{\mathsf{T_b}}}]$, results that there is at least one value in $[BCET_{\mathsf{T_b}}; \mathcal{V}^{\text{right}}_{p_{\mathsf{T_b}}}]$ that leads to a slack $\mathcal{S}_{p_{\mathsf{T_t}}}$ smaller than the lowest slack calculated for that interval. Therefore, (24) can not be valid.

Hence, if (21) is valid, all values between $\mathcal{V}^{\text{left}}_{p_{\mathsf{T_b}}}$ and $\mathcal{V}^{\text{right}}_{p_{\mathsf{T_b}}}$ determine a slack for $p_{\mathsf{T_t}}$ equal to $\mathcal{S}^{\text{left}}_{p_{\mathsf{T_t}}}$ and $\mathcal{S}^{\text{right}}_{p_{\mathsf{T_t}}}$. □

For any two values of the base parameter satisfying (21) no search is required within the interval determined by these values. The slack of the target parameter corresponding to any value of the base parameter located between these 2 values is always constant.

6.3 Stochastic analysis

In this section we present a stochastic approach for multi-dimensional sensitivity analysis. It is based on a previously published design space exploration framework (Hamann et al. 2006), which uses multi-dimensional evolutionary search techniques (Bleuler et al. 2003; Zitzler et al. 2002).

Since the complexity of the exact approach presented in previous section grows exponentially with the number of search dimensions, we propose a stochastic algorithm which is suited for search spaces with more than two dimensions.

The presented stochastic multi-dimensional sensitivity analysis technique approximates the sought-after sensitivity front from two sides, i.e. coming from the space of working and from the space of non-working system property combinations. Therefore, the stochastic sensitivity analysis approach is perfectly suited to approximate system sensitivity by lower and upper bounds, and thus allows to quickly identify system configurations with high robustness potential. For this reason, the presented approach can be efficiently used for system sensitivity minimization.

The section is structured as follows: we first give a short description of how we use the exploration framework to perform multi-dimensional sensitivity analysis (Sect. 6.3.1). We then give details on the search space encoding (Sect. 6.3.2) and the creation of the initial population used as starting point for exploration (Sect. 6.3.3). Afterwards, we discuss the exploration strategy used during variation to bound the search space, and thus improving analysis speed and approximation quality (Sect. 6.4). Finally, we explain in detail the exploration strategy implemented by the variation operators (Sects. 6.4.1 and 6.4.2).

6.3.1 Analysis idea

Classical applications of exploration frameworks for complex distributed systems assume variation of system parameters like scheduling, mapping, etc. to optimize criteria including timing, power consumption and buffer sizes.

In this paper we utilize design space exploration in a different way in order to cover multi-dimensional sensitivity analysis. Instead of modifying the system parameter configuration during exploration, we modify system properties subject to sensitivity analysis, i.e. worst-case core execution times, CPU clock rates, input data rates, etc.

Thereby, the optimization objectives are, depending on the considered system properties, either the maximization or the minimization of the property values under the restriction that the system must stay functional, i.e. all system constraints, like end-to-end deadlines, maximum power consumption, maximum buffer sizes, etc., must be satisfied.

For instance, in the case of a three-dimensional *WCET* sensitivity analysis of three tasks, the search space consists of the *WCET* assignment for those tasks and the optimization objectives are the simultaneous maximization of the latter.

Note that our exploration framework (Hamann et al. 2006) performs Pareto-optimization, and that the obtained Pareto-front corresponds to the sought-after sensitivity front representing the boundary between feasible and non-feasible system configurations. Furthermore, the algorithms presented in the next sections are applicable to arbitrary system properties, including those which are subject to maximization (e.g. worst-case core execution times) and those which are subject to minimization (e.g. input data rates).

It is important to mention, that the sensitivity front coverage is controlled by problem independent selector algorithms (Bleuler et al. 2003). We currently use SPEA2 (Zitzler et al. 2002) as selector, which assures a diversified approximation of the Pareto-front through Pareto-dominance-based selection and density approximation.

6.3.2 Search space encoding

A system property value combination considered during the stochastic multi-dimensional sensitivity analysis is encoded as vector containing one real number entry for each considered property. In the following we refer to such a vector as *individual*.

For instance, in the case of a three-dimensional sensitivity analysis for the system properties p_1, p_2 and p_3, an individual A is represented as three dimensional vector, i.e. $A = (a_1, a_2, a_3)$.

6.3.3 Initial population

Algorithm 5 describes the creation of the initial population. In the first part (lines 1 to 3) it uses one-dimensional sensitivity analysis as described in Sect. 5 to calculate the available slack for each considered system property.

The one-dimensional slack values represent the extreme points of the sought-after sensitivity front, and thus describe the bounding hypercube containing all valid system property assignments. It is used throughout the whole exploration to considerably limit the generation of invalid individuals.

In the second part of the algorithm (lines 4 to 7) the rest of the initial population is randomly generated. Thereby, the individuals are uniformly distributed within the search space bounded by the hypercube calculated in the first part of the algorithm.

6.4 Bounding the search space

In this section we give a description of the technique used by our approach to efficiently bound the region containing the sought-after sensitivity front during explo-

Algorithm 5 Initial Population

INPUT: $(p_1, \ldots, p_n)$, the property tuple to be analyzed
$\mathcal{V}^{\text{init}}_{\mathsf{P}_1}, \ldots, \mathcal{V}^{\text{init}}_{\mathsf{P}_n}$, the initial property values
$\alpha > n$, the initial population size
OUTPUT: Initial population $\mathcal{I}$

1: **for** $(i = 1;\ i <= n;\ i = i + 1)$ **do**
2: $\quad \mathcal{V}^{\text{ext}}_{p_i} = computeSlack(p_i)$
3: $\quad$ add vector $(\mathcal{V}^{\text{init}}_{p_1}, \ldots, \mathcal{V}^{\text{ext}}_{p_i}, \ldots, \mathcal{V}^{\text{init}}_{p_n})$ to $\mathcal{I}$
4: **end for**
5: **while** $(|\mathcal{I}| < \alpha)$ **do**
6: $\quad$ **for** $(i = 1;\ i <= n;\ i = i + 1)$ **do**
7: $\quad\quad$ Choose random
$\quad\quad \mathcal{V}^{\text{rand}}_{p_i} \in [\min(\mathcal{V}^{\text{init}}_{p_i}, \mathcal{V}^{\text{ext}}_{p_i}), \max(\mathcal{V}^{\text{init}}_{p_i}, \mathcal{V}^{\text{ext}}_{p_i})]$
8: $\quad$ **end for**
9: $\quad$ add vector $(\mathcal{V}^{\text{rand}}_{p_1}, \ldots, \mathcal{V}^{\text{rand}}_{p_n})$ to $\mathcal{I}$
10: **end while**

ration. This information is used by the variation operators presented in the following sections to prevent exploration from generating and evaluating individuals not improving approximation quality.

The algorithm used to bound the search space as described above is based on the notion of Pareto-optimality.

Definition 5 (Pareto-optimal) Given a set V of n-dimensional vectors in $\mathbb{R}^n$, the vector $v = (v_1, \ldots, v_n) \in V$ dominates the vector $w = (w_1, \ldots, w_n) \in V$ iff for all elements $1 \leq i \leq n$ we have

1. minimization problem: $v_i \leq w_i$ and for at least one element l we have $v_l < w_l$.
2. maximization problem: $v_i \geq w_i$ and for at least one element l we have $v_l > w_l$.

A vector is called *Pareto-optimal* iff it is not dominated by any other vector in V.

Given Definition 5 a simple algorithm can be derived checking whether or not a given k-dimensional vector v is Pareto-optimal with respect to a collection of reference vectors V. In the following we refer to such an algorithm as $dominatedBy(v, V, max)$, where $max \in \{true, false\}$ indicates if Pareto-optimality is meant in the sense of a maximization or minimization problem.

In order to achieve an efficient bounding of the search space containing the sensitivity front, our analysis maintains two sets of individuals:

- the set of evaluated Pareto-optimal working individuals $\mathcal{F}^w$ called *bounding working Pareto-front*. Note that for system properties subject to maximization (minimization) Pareto-optimality is meant in the sense of a maximization (minimization) problem.

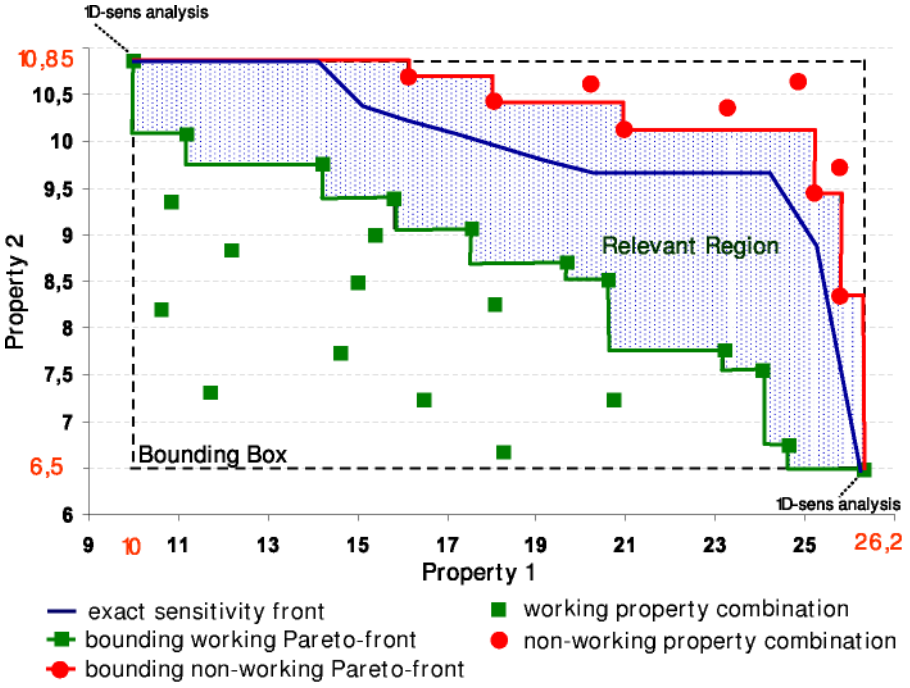


Fig. 7 Relevant region for two system properties subject to maximization

- the set of evaluated Pareto-optimal non-working individuals $\mathcal{F}^{nw}$ called *bounding non-working Pareto-front*. Note that for system properties subject to maximization (minimization) Pareto-optimality is meant in the sense of a minimization (maximization) problem.

In the following we refer to the region lying between the Pareto-sets $\mathcal{F}^{w}$ and $\mathcal{F}^{nw}$ as *relevant region*.

It is easy to understand, that the sought-after sensitivity front lies inside the relevant region. Consider for instance the relevant region determined for two system properties subject to maximization visualized in Fig. 7.

An individual lying below the bounding working Pareto-front cannot be part of the sensitivity front, since at least one individual of the bounding working Pareto-front has higher, and thus better, values for all considered system properties. Also, an individual lying above the bounding non-working Pareto-front cannot be part of the sensitivity front, since there exist at least one non-working individual on the bounding non-working Pareto-front, which has smaller values for all considered system properties. Consequently, the individual in question is non-working as well.

Algorithm 6 verifies if a given vector v lies in the relevant region. By configuring the input variable *max* the algorithm can be used for system properties subject to minimization ($max = false$) and maximization ($max = true$).

Note that the sets $\mathcal{F}^{w}$ and $\mathcal{F}^{nw}$ are updated during exploration directly after the evaluation of the offsprings generated during the processing of each generation. More precisely, for each working (non-working) individual i created during variation it is

Algorithm 6 isInRelevantRegion

INPUT: k-dimensional vector v, bounding working Pareto-front $\mathcal{F}^w$, bounding non-working Pareto-front $\mathcal{F}^{nw}$, type of considered properties $max \in \{true, false\}$

OUTPUT: *true*: if v lies in the relevant region, *false*: otherwise

1: $workingOK = !dominatedBy(v, \mathcal{F}^w, max)$

2: $nonWorkingOK = !dominatedBy(v, \mathcal{F}^{nw}, !max)$

3: **return** $workingOK$ && $nonWorkingOK$

checked whether i is Pareto-optimal with respect to the individuals contained in $\mathcal{F}^w$ ($\mathcal{F}^{nw}$). If this is the case, i is added to $\mathcal{F}^w$ ($\mathcal{F}^{nw}$) and all individuals dominated by i are removed.

It is not strictly necessary to remove dominated individuals from the corresponding sets for our approach to work. However, by doing so the maintained sets are kept small and the overall computational effort for checking whether or not a given individual is contained in the relevant region is reduced.

The variation operators guiding the exploration process, which are presented in the following sections, utilize Algorithm 6 to highly increase the generation of offsprings directly improving the approximation of the sought-after sensitivity front. This leads to decreased exploration time for the same quality of approximation.

6.4.1 Crossover operator

The crossover operator described in Algorithm 7 implements a heuristic strategy to converge towards the sensitivity front, i.e. the boundary between working and non-working systems. Its main function during exploration is to locally refine the approximation of the sensitivity front. It takes as input two parent individuals A_1 and A_2 and generates two offsprings B_1 and B_2 by using the generalized mean function (Definition 6), which maps well to the structure of the solution space.

Definition 6 (Generalized Mean) For positive numbers $x_1, \ldots, x_n$ the k-th mean is defined as follows:

$$M_k(x_1, \ldots, x_n) = \sqrt[k]{\frac{1}{n} \sum_{i=1}^{n} x_i^k}.$$

Special cases: $k \to -\infty$: $\min(x_1, \ldots, x_n)$; $k = -1$: harmonic mean; $k \to 0$: geometric mean; $k = 1$: arithmetic mean; $k = 2$: quadratic mean; $k \to \infty$: $\max(x_1, \ldots, x_n)$.

For crossover the generalized mean is applied property wise on the property vectors of the parent individuals (line 7). Figure 8 visualizes the behavior of the generalized mean function for the 2-dimensional case. A and B represent two points we want to "crossover". If $k = 1$ is chosen we obtain the arithmetic mean between A and B. This corresponds to a linear characteristic of the sensitivity front, which we observe for instance in the case of load dependent system properties (see Fig. 12). For

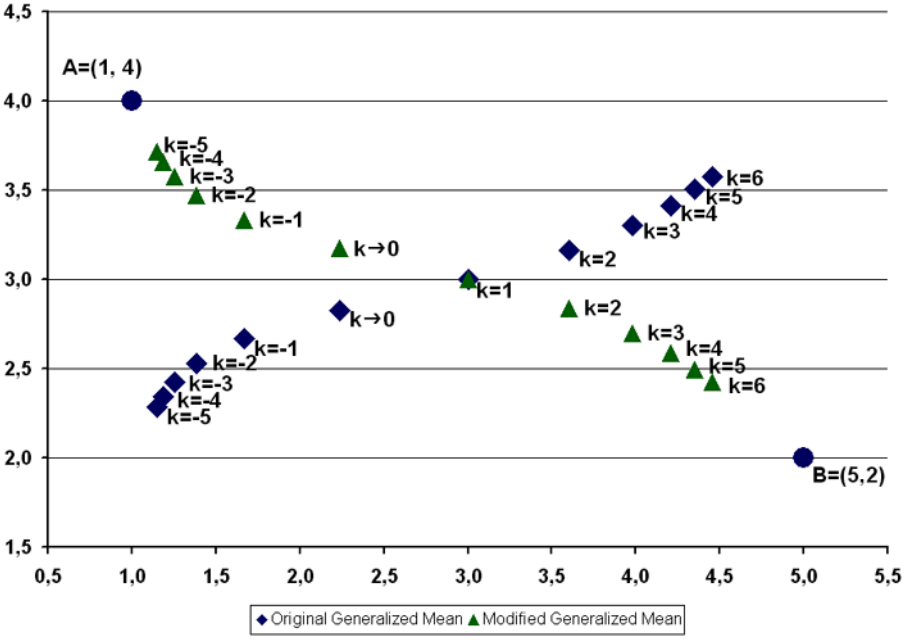


Fig. 8 Coordinate-wise generalized mean between A and B for different k

the case that the crossover operator chooses $k < 1$ a convex characteristic of the sensitivity front is approximated, whereas $k > 1$ leads to the approximation of a concave characteristic.

For a better adaptation of the crossover operator to the solution space structure, we modify the values calculated according to the generalized mean formula slightly (lines 8–10). The aim of this modification is to obtain a n-dimensional curve connecting the considered crossover points. Such a connecting curve can be achieved, for instance, by mirroring the calculated general mean values at the bisector defined by the according parent property values in case that the first parent has a higher property value than the second parent. Figure 8 shows the curve we obtain by applying this modification.

The crossover operator utilizes Algorithm 6 presented in Sect. 6.4 to generate offsprings lying in the relevant region (line 13). However, in some cases the algorithm might fail to find such offsprings. Therefore, the maximum number of attempts is bounded by the constant $iter_{\max}$. One reason for the algorithm not being able to find offsprings lying in the relevant region might be that one of the selected parents for crossover dominates the other one. However, such cases rarely occur during exploration, and are therefore not distinguished in the crossover algorithm.

Note that the crossover operator automatically ensures, that all generated offsprings lie within the bounding hypercube given by $\mathcal{V}_{p_i}^{\text{init}}$ and $\mathcal{V}_{p_i}^{\text{ext}}$ determined during the creation of the initial population (Sect. 6.3.3) for all considered system properties p_i.

Algorithm 7 Crossover operator

INPUT: parents $A_1 = (a_{11}, \ldots, a_{1n})$ and $A_2 = (a_{21}, \ldots, a_{2n})$
$k_{\min}$ and $k_{\max}$ with $k_{\max} \geq k_{\min}$
type of considered properties $maximize$ (boolean)
bounding working Pareto-front $\mathcal{F}^w$
bounding non-working Pareto-front $\mathcal{F}^{nw}$
maximum number of attempts to reach relevant region $iter_{\max}$

OUTPUT: children $B_1 = (b_{11}, \ldots, b_{1n})$ and $B_2 = (b_{21}, \ldots, b_{2n})$

1: **for** $(i = 1;\ i <= 2;\ i = i + 1)$ **do**
2: Choose random $k \in [k_{\min}, k_{\max}]$
3: $iter = 0$
4: **repeat**
5: $iter = iter + 1$
6: **for** $(j = 1;\ j <= n;\ j = j + 1)$ **do**
7: $b_{ij} = M_k\,(a_{1i}, a_{2i})$
8: **if** $(a_{1i} > a_{2i})$ **then**
9: $temp = \min(a_{1i}, a_{2i}) + \frac{|a_{1i} - a_{2i}|}{2}$
10: $b_{ij} = temp - (b_{ij} - temp)$
11: **end if**
12: **end for**
13: **until** $(isInInterestingRegion(B_i, \mathcal{F}^w, \mathcal{F}^{nw}, maximize)\ ||\ iter > iter_{\max})$
14: **end for**

6.4.2 Mutation operator

The described crossover operator leads to the local convergence of the obtained property values towards the sought-after sensitivity front. In other words, it approximates the sensitivity front "between" individuals considered by the evolutionary algorithm.

Of course, it is possible that the variety of the initial population is insufficient to cover the whole sensitivity front by only using the crossover operator. Additionally, the exploration may get stuck in sub-regions of the front, without the possibility to reach other parts. Therefore, we introduce a mutation operator, enabling the evolutionary search to break out these sub-regions and to cover unexplored parts of the sensitivity front.

The proposed operator uses an adaptive mutation range which is based on the current approximation quality of the bounding Pareto-fronts $\mathcal{F}^w$ and $\mathcal{F}^{nw}$.

Definition 7 (Average and maximum front distances $\mathcal{D}_{\text{avg}}$ and $\mathcal{D}_{\max}$) Given the bounding working Pareto-front $\mathcal{F}^w$ and the bounding non-working Pareto-front $\mathcal{F}^{nw}$,

the average and maximum front distances $\mathcal{D}_{\text{avg}}$ and $\mathcal{D}_{\text{max}}$ are defined as follows:

$$\mathcal{D}_{\text{avg}} = \frac{\sum_{v \in \mathcal{F}^w} \min_{w \in \mathcal{F}^{nw}} \|v - w\|_2}{|\mathcal{F}^w|},$$

$$\mathcal{D}_{\text{max}} = \max_{v \in \mathcal{F}^w} \min_{w \in \mathcal{F}^{nw}} \|v - w\|_2.$$

Note that during exploration it is sufficient to calculate $\mathcal{D}_{\text{avg}}$ and $\mathcal{D}_{\text{max}}$ once for each processed generation before variation. By this means the computational effort for the adaptive mutation range is kept minimal.

The mutation operator is described in Algorithm 8. It takes as input one parent individual from which it creates one offspring by randomly increasing or decreasing each property value by a random value bounded by $\mathcal{D}_{\text{avg}}$ and $\mathcal{D}_{\text{max}}$. Note that compared to a static mutation range, this adaptive approach considerably increases the probability to create offsprings lying in the interesting region, and thus leads to an increased convergence speed of the bounding Pareto-fronts $\mathcal{F}^w$ and $\mathcal{F}^{nw}$.

Additionally, the mutation operator takes as input the minimum and the maximum allowed values for each considered system property which are respected during mutation. Note that these values correspond to $\mathcal{V}_{p_i}^{\text{init}}$ and $\mathcal{V}_{p_i}^{\text{ext}}$ calculated during the creation of the initial population (Sect. 6.3.3).

Like the crossover operator, the mutation operator utilizes Algorithm 6 described in Sect. 6.4 to generate offsprings lying in the relevant region (line 13).

7 Example

In this section we present a set of experiments performed on the automotive real-time system example presented in Fig. 9. Section 7.2 presents the results of one-dimensional sensitivity analysis of different system properties. Section 7.3 shows a comparison between the multi-dimensional sensitivity analysis results obtained using the exact and the stochastic algorithms.

The system presented in Fig. 9 contains two independent subsystems integrated via a control-area-network (CAN). The signal processor (DSP) on ECU_4 periodically executes a filter task (T_3) that sends data at completion over channel C_3 on the bus to the IP_1 component on ECU_5. The sensor Sens on ECU_1 sporadically communicates via the same network with the control task (T_1) mapped on CPU. On CPU is additionally executed a periodic task (T_2) that sends data over channel C_2 to the hardware component HW located on ECU_3. On both, CPU and CAN is assumed a static priority preemptive arbitration policy. For each functional path is constrained a maximum end-to-end delay that the system has to satisfy in order to assure the proper functionality.

Obviously, the interference of the logical communication channels $\text{C}_1, \text{C}_2, \text{C}_3$ on CAN and of the tasks T_1, T_2 on CPU leads to complex timing dependencies between the elements of different communication paths. These dependencies must be captured by the system-level analysis model.

Algorithm 8 Mutation operator

INPUT:
parent $A = (a_1, \ldots, a_n)$
minimum property values $a_1^{\min}, \ldots, a_n^{\min}$
maximum property values $a_1^{\max}, \ldots, a_n^{\max}$
average and maximum front distances $\mathcal{D}_{\text{avg}}$ and $\mathcal{D}_{\max}$
type of considered properties *maximize* (boolean)
bounding working Pareto-front $\mathcal{F}^w$
bounding non-working Pareto-front $\mathcal{F}^{nw}$
maximum number of attempts to reach relevant region $iter_{\max}$

OUTPUT: child $B = (b_1, \ldots, b_n)$

1: $iter = 0$
2: **repeat**
3: $iter = iter + 1$
4: **for** $(i = 1;\ i <= n;\ i = i + 1)$ **do**
5: Choose random $x \in [\frac{\mathcal{D}_{\text{avg}}}{2}, \mathcal{D}_{\max}]$
6: Choose random boolean *bool*
7: **if** (*bool*) **then**
8: $b_i = \min(a_i + x, a_i^{\max})$
9: **else**
10: $b_i = \max(a_i - x, a_i^{\min})$
11: **end if**
12: **end for**
13: **until** $(isInInterestingRegion(B, \mathcal{F}^w, \mathcal{F}^{nw}, maximize)\ ||\ iter > iter_{\max})$

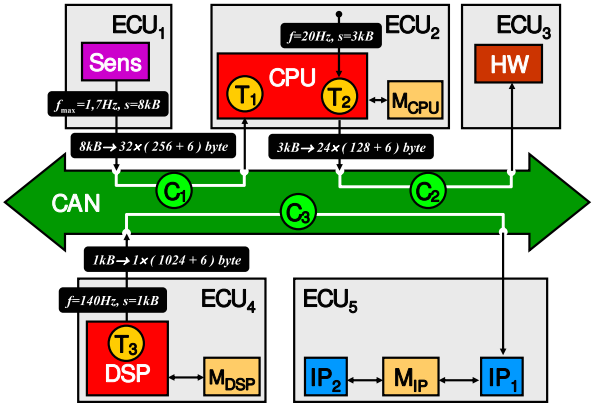


Fig. 9 The initial system configuration

Table 1 Execution and communication times

Task	Execution time	Channel	Communication time
T_1	250 ms	C_1	17.50 ms
T_2	10 ms	C_2	6.50 ms
T_3	[2;4] ms	C_3	2.15 ms

7.1 Set-up

The actuators There are three actuators: the sensor sporadically sends data blocks of 8 kB size to T_1, with a maximum sending frequency of 1.66 Hz, which corresponds to a *sporadic event model* with a minimum sporadic period of $1/1.66$ Hz $= 600$ ms. Process T_2 is periodically activated by the RTOS (real-time operating system) on the CPU with a period of $1/20$ Hz $= 50$ ms and sends 3 kB of data at completion. The signal processing application T_3 on DSP has a sampling frequency of 10 Hz, corresponding to a sampling period of 10 ms. T_3 sends 1 kB of data at the end of each execution. The three actuators are assigned to three generic event sources called S1, S2, S3. The one-to-one correspondence is shown in Table 2.

The network Instead of sending the complete data block, the data packets are fragmented to avoid too long blocking times. Each 8 kB data block from Sens is split into 32 packets of 262 byte each, 256 bytes plus 6 bytes protocol overhead–address, length, and CRC. The 3 kB blocks from T_2 are split into 24 packets of $(128+6) = 134$ bytes. Channel C_2 has a higher priority than channel C_1. The highest-priority channel C_3 does not split the DSP data packets, but only adds the 6 byte protocol information. The initial network speed is 460 kB/s.

The overall average network load L_{CAN} is:

$$\begin{aligned} L_{\mathsf{CAN}} &= L_{\mathsf{Sens-CPU}} + L_{\mathsf{CPU-HW}} + L_{\mathsf{DSP-IP}_1} \\ &= 13.3\ \text{kB/s} + 60\ \text{kB/s} + 100\ \text{kB/s} \\ &= 173.3\ \text{kB/s}. \end{aligned}$$

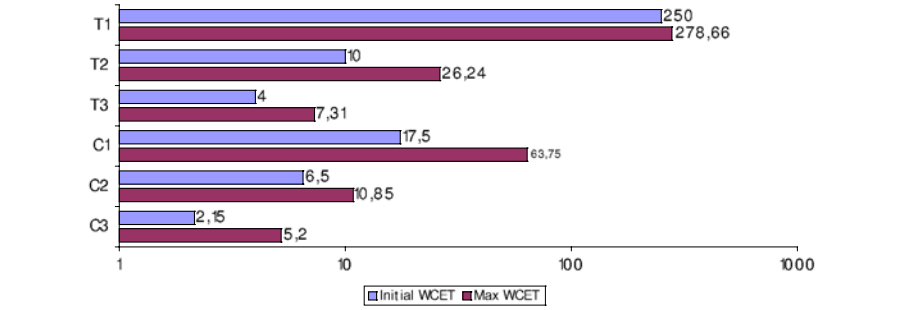
Execution and communication times The initial execution and communication times of the tasks on CPU and of the channels on BUS are assumed constant. The DSP application T_3 has a variable execution time depending on the data to be processed. The execution and communication times are listed in Table 1.

Resource scheduling The CPU and BUS are both scheduled according to a static priority preemptive policy. Channels C_3 and C_1 have on BUS the highest and the lowest priority, respectively. On CPU, task T_1 has a priority higher than T_2. Due to the non-preemptive packet communication on the network and the fragmentation of the data blocks, blocking times are calculated for the higher priority channels.

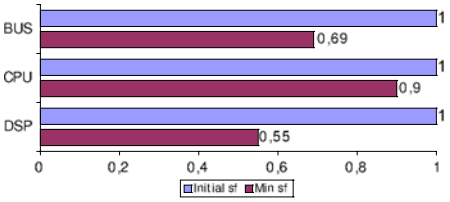
Real-time constraints The real-time behavior of the application is given by a set of hard real-time constraints defined for the three functional paths of the system presented in Fig. 9. The constraints are listed in Table 2.

Table 2 Real-time constraints

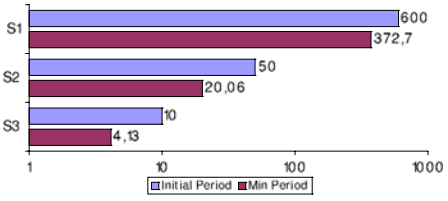
Source	Functional path	Max delay (ms)
S1	$\text{Sens} - C_1 - T_1$	400
S2	$T_2 - C_2 - \text{HW}$	300
S3	$T_3 - C_3 - IP_1$	10



(a) WCET feasibility slack



(b) Resource speed feasibility slack



(c) Period feasibility slack

Fig. 10 One-dimensional sensitivity analysis

7.2 One-dimensional sensitivity analysis

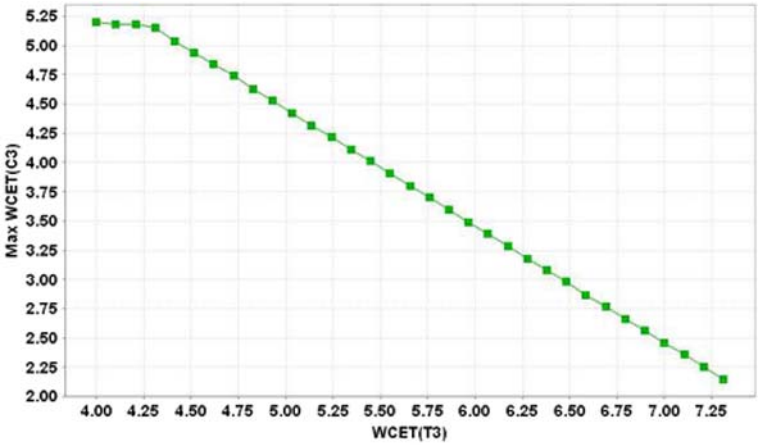
Figure 10 shows the initial values of the task execution times, the initial scaling factors of the resource speeds and the initial activation periods at system input, and their corresponding feasibility slacks. The values in Figs. 10(a) and (c) are displayed in logarithmic range.

7.3 Multi-dimensional sensitivity analysis

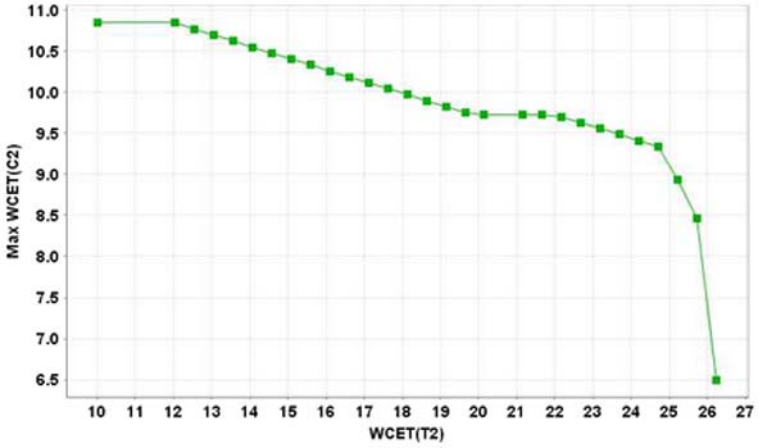
In Sects. 7.3.1 and 7.3.2 we show the results obtained using the exact and the stochastic methods, respectively. In Sect. 7.3.3 we compare the two approaches and we present a set of numbers showing the algorithm complexity.

7.3.1 Search-based analysis

We grouped the task pairs in three groups depending on the dependencies between them. The first group consists of tasks with functional dependencies, i.e., the tasks



(a) $WCET_{T_3} - WCET_{C_3}$



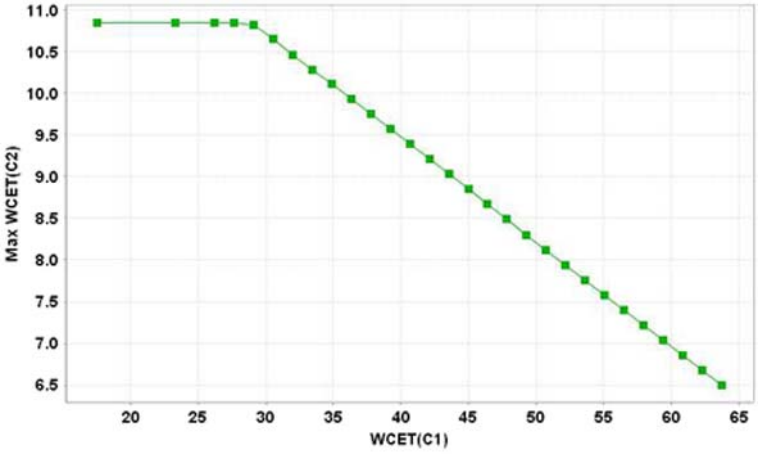
(b) $WCET_{T_2} - WCET_{C_2}$

Fig. 11 Tasks with functional dependencies

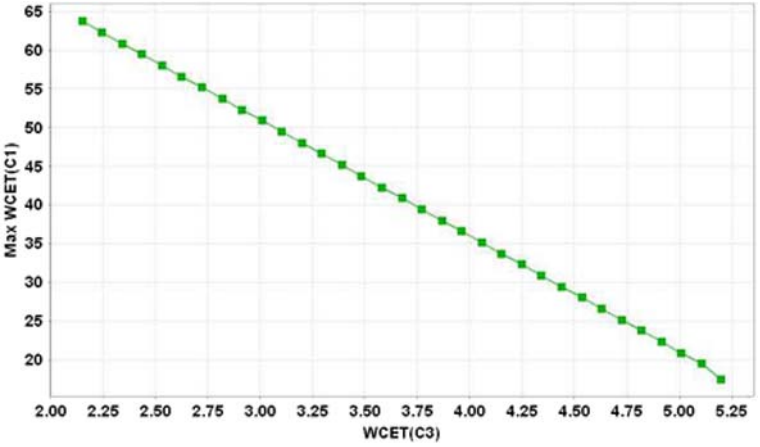
belonging to the same dependency path. The obtained results are shown in Fig. 11. The increase of the worst-case execution time of C_3 leads to a higher jitter at the input of C_3 that may generate transient overload on the communication resource. Therefore, the communication time of C_3 must be adapted accordingly. That is why the slack of C_3 decreases linearly with the increase of $WCET_{T3}$.

The second group contains tasks with load dependencies, i.e. tasks mapped on the same resource. Figure 12 shows the results obtained for different pairs of channels mapped on CAN. Obviously, when increasing the communication volume of one channel, the load on the network increases too. Thus, the available slack of all other channels is reduced.

The last group is formed of tasks with no direct dependency, i.e. tasks that are mapped on different resources and without any functional dependency between them.



(a) $WCET_{C_1} - WCET_{C_2}$

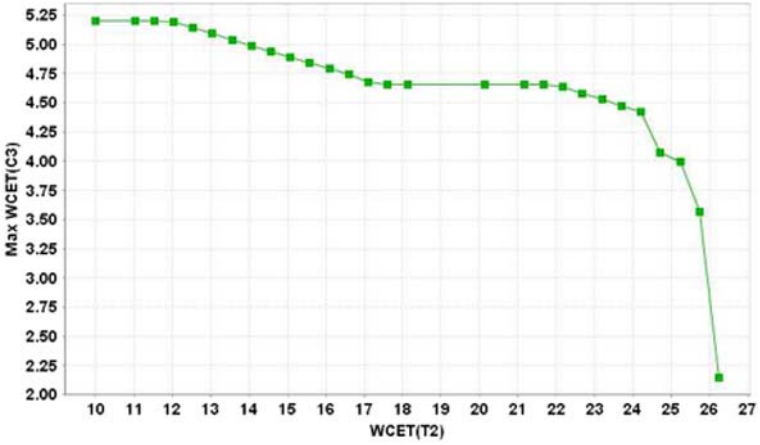


(b) $WCET_{C_3} - WCET_{C_1}$

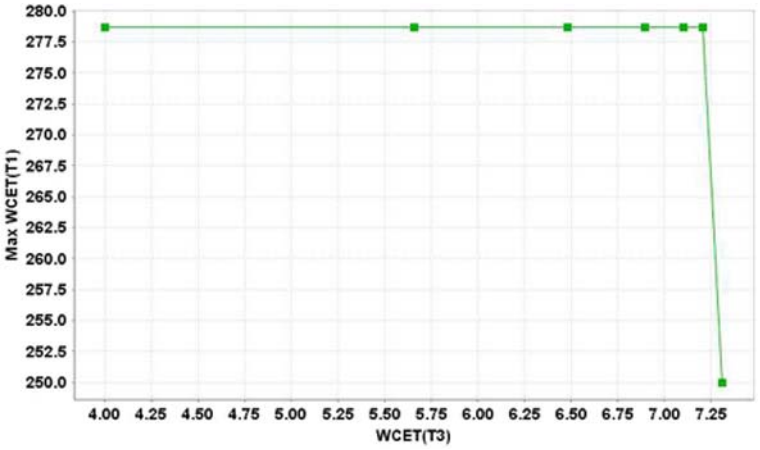
Fig. 12 Tasks with load dependencies

Figure 13 shows the two-dimensional sensitivity results obtained for the worst-case execution times of different pairs of independent tasks. As it can be observed in Fig. 13(b), since tasks T_3 and T_1 are totally independent, variations of the worst-case execution time of T_3 does not affect the available slack of T_1. Quite similar are the results obtained for the task-pair $T_2 - C_3$. However, since both tasks T_2 and C_3 have a functional and a load dependency with C_2, they are not totally independent. Therefore, variations of $WCET_{T_2}$ lead to slight variations of the available slack of C_3.

The second investigated system property is the speed of the processing and communication elements. Figure 14 shows the two-dimensional sensitivity analysis of the resource speed factors. Since the DSP and the CPU does not share common application paths, the slack of the DSP's speed is independent of the speed of CPU, as shown



(a) $WCET_{\mathsf{T}_2} - WCET_{\mathsf{C}_3}$

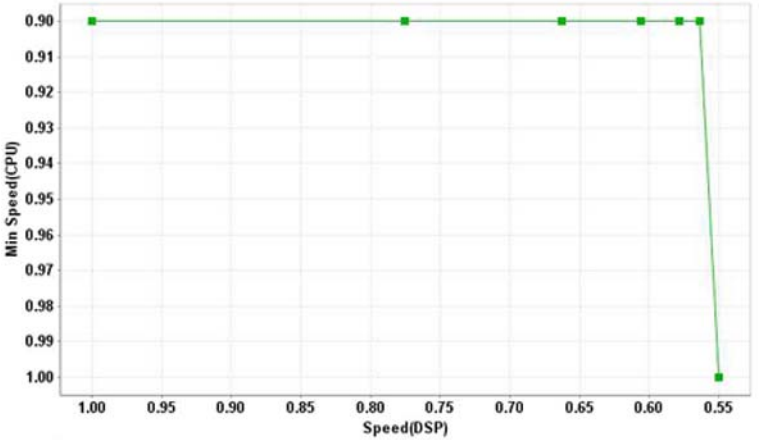


$WCET_{\mathsf{T}_3} - WCET_{\mathsf{T}_1}$

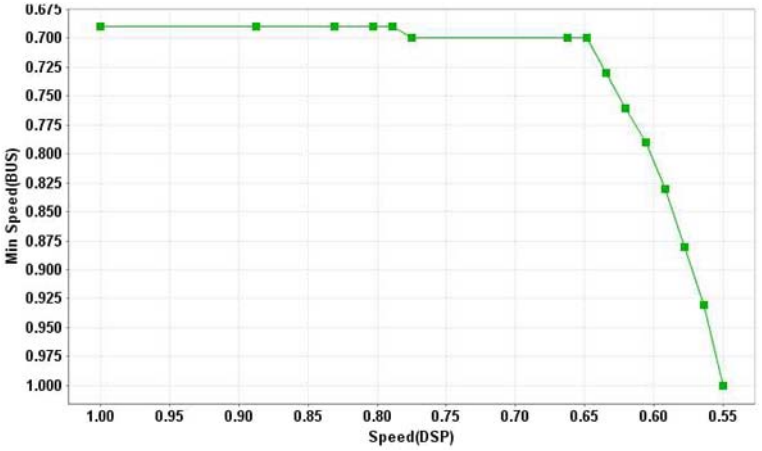
Fig. 13 Tasks with no direct dependency

in Fig. 14(a). Largely, are also the speeds of DSP and CAN (Fig. 14(b)). To keep the feasible configurations below the sensitivity front, the axes have decreasing ranges.

The last investigated property is the task activation period. Figure 15(a) shows the dependency between the activation periods of the application paths $\mathsf{Sens} - \mathsf{T}_1$ and $\mathsf{T}_3 - \mathsf{IP}_1$. We observe that for values of the former above 430, the feasibility slack of the latter is constant. A further decrease of the period of $\mathsf{Sens} - \mathsf{T}_1$ above 430 will increase the load on CAN over the admitted limit. Therefore, the feasibility slack of the period of $\mathsf{T}_3 - \mathsf{IP}_1$ is reduced accordingly. A more linear dependency is observed in Fig. 15(b) between the periods of the application paths $\mathsf{T}_2 - \mathsf{HW}$ and $\mathsf{T}_3 - \mathsf{IP}_1$. This is due to tighter latency constraints of these paths. Again, the range of the axes are inverted to display the feasible region below the sensitivity front.



(a) $sf_{DSP} - sf_{CPU}$



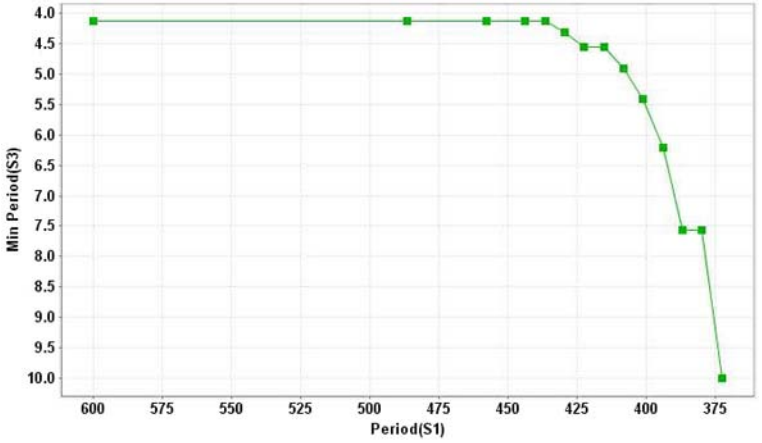
(b) $sf_{DSP} - sf_{CAN}$

Fig. 14 Resource speed factors

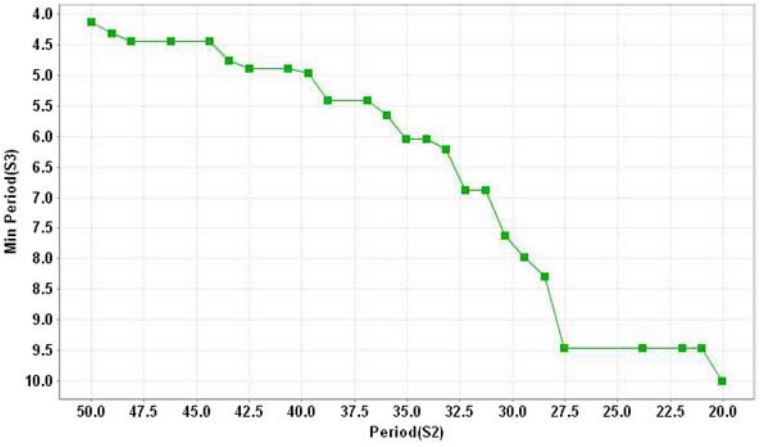
7.3.2 Stochastic analysis

Figures 16(a) and (b) show the three-dimensional sensitivity analysis results obtained using the stochastic approach presented in Sect. 6.3. Note that for both analyses our exploration framework generated 100 generations with each 200 individuals, taking approximately 5 minutes on a 2.00 GHz Athlon64 standard PC.

In both figures we observe that the two-dimensional projection of the sensitivity front on the $T2 - C3$ plan accurately approximates the curve in Fig. 13(a) obtained using the exact approach.



(a) $\mathcal{P}_{S1} - \mathcal{P}_{S3}$

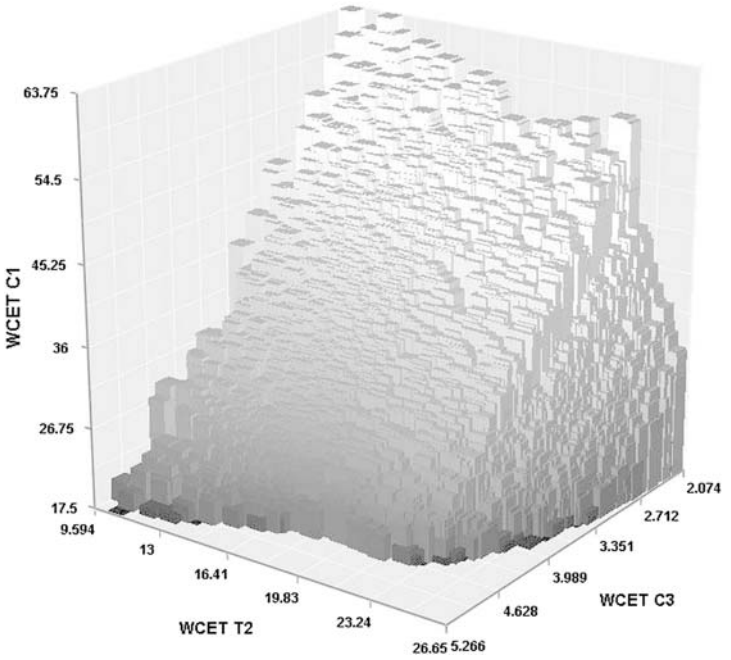


(b) $\mathcal{P}_{S2} - \mathcal{P}_{S3}$

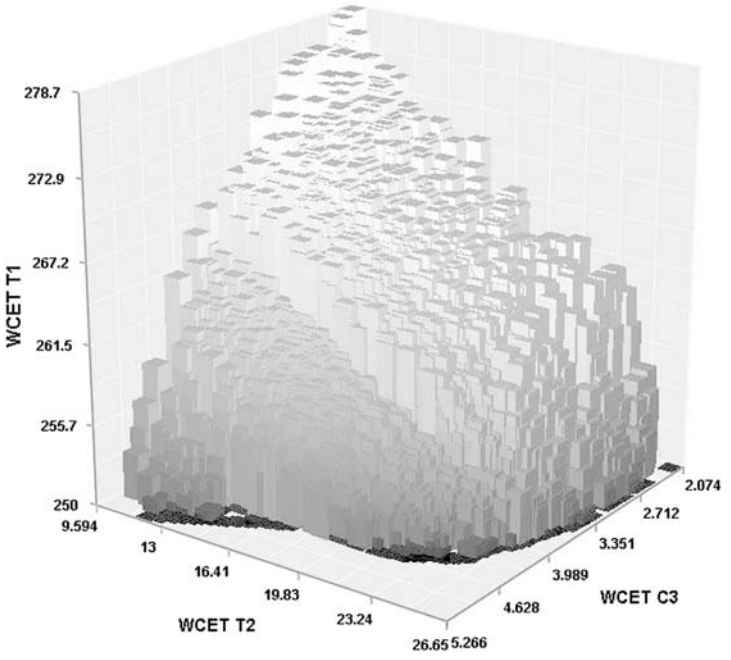
Fig. 15 Activation periods

7.3.3 Accuracy and complexity

Figure 17 shows a comparison of the results obtained using the proposed approaches. In the legend, the numbers assigned to the stochastic algorithm are the number of generations and the number of considered individuals per generation. For the search-based algorithm we used a search depth equal to 5. Table 3 shows the run-times (in ms) of the search-based and stochastic algorithms. The search-based algorithm was applied on two set of properties, one with monotonic relation (w/s_step) and the other one with non-monotonic relation (w/os_step). The experiments were carried out on a 2.00 GHz Athlon64 standard PC. The complexity of the search-based algorithm is defined by the search-depth, while the complexity of the stochastic algorithm



(a) $WCET_{C3} - WCET_{T2} - WCET_{C1}$



(b) $WCET_{C3} - WCET_{T2} - WCET_{T1}$

Fig. 16 Three-dimensional WCET sensitivity

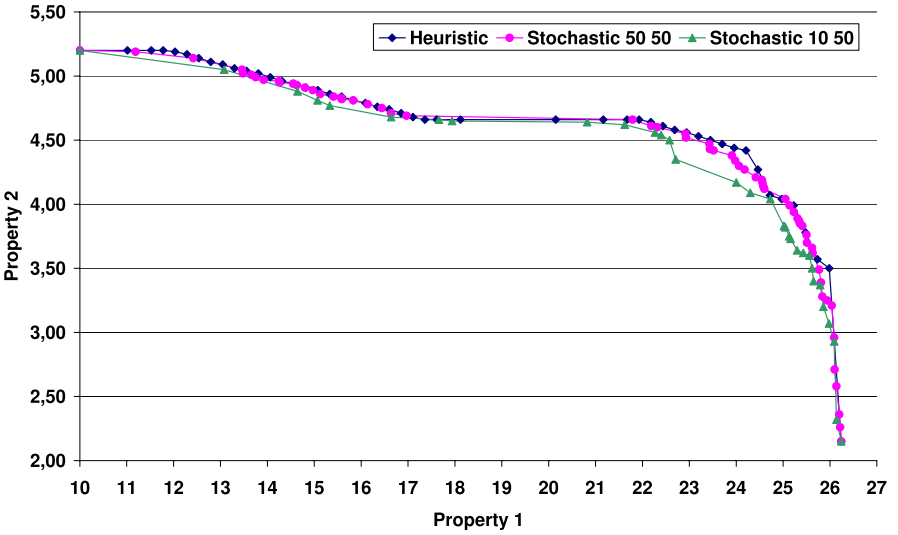


Fig. 17 Comparison of the two approaches

Table 3 The run-times of the algorithms (ms)

Search-based			Stochastic	
Depth	*w/ s_step*	*w/os_step*	(#gen;#ind)	*WCET*
3	1469	1526	(10;50)	16806
5	4334	4580	(30;50)	50327
7	15906	17924	(50;50)	85141

is determined by the number of generations and the number of individuals per generation.

8 Conclusion

Design robustness, measured as sensitivity to design parameter variations, is a key concern in embedded system design. In this paper we presented a sensitivity analysis framework for heterogeneous systems with complex timing dependencies and requirements. The sensitivity analysis uses as basis the SymTA/S compositional analysis methodology, which offers high scalability and adaptability. We presented a binary search algorithm used for the one-dimensional sensitivity analysis of different system parameters. Moreover, we described an exact and a stochastic approach for multi-dimensional sensitivity analysis. The exact approach performs for two-dimensional spaces better than the stochastic approach, but is limited to monotonous functions and its complexity grows exponentially with the number of dimensions of the investigated search space.

References

Audsley NC, Burns A, Richardson MF, Tindell K, Wellings AJ (1993) Applying new scheduling theory to static priority preemptive scheduling. J Real-Time Syst 8(5):284–292

Bini E, Natale MD Buttazzo G (2006) Sensitivity analysis for fixed-priority real-time systems. In: 18th Euromicro conference on real-time systems (ECRTS), Dresden, Germany, pp 13–22

Bleuler S, Laumanns M, Thiele L, Zitzler E (2003) PISA—a platform and programming language independent interface for search algorithms

Chakraborty S, Knzli S, Thiele L (2003) A general framework for analysing system properties in platform-based embedded system designs. In: Proceedings of design, automation and test in Europe (DATE), Munich, Germany, March 2003

Cottet F, Babau JP (1996) An iterative method of task temporal parameter adjustment in hard real-time systems. In: Proceedings of the 2nd IEEE international conference on engineering of complex computer systems (ICECCS), Montreal, Canada, October 1996

Hamann A, Jersak M, Richter K, Ernst R (2006) A framework for modular analysis and exploration of heterogeneous embedded systems. Real-Time Syst J 33(1-3):101–137

Harbour MG, Garcia JG, Gutierrez JP, Moyano JD (2001) MAST: modeling and analysis suite for real time applications. In: Euromicro conference on real-time systems (ECRTS)

Henia R, Racu R, Ernst R (2006) Improved output jitter calculation for compositional performance analysis of distributed systems. In: 15th international workshop on parallel and distributed real-time systems (WPDRTS), Long Beach, CA

Henia R, Hamann A, Jersak M, Racu R, Richter K, Ernst R (2005) System level performance analysis—the SymTA/S approach. IEE Proc Comput Digit Tech 152(2):148–166

Joseph M, Pandya P (1986) Finding response times in a real-time system. Comput J 29(5):390–395

Katcher DI, Arakawa H, Strosnider JK (1993) Engineering and analysis of fixed priority schedulers. Softw Eng 19(9):920–934

Lehoczky J (1990) Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: Proceedings of the real-time systems symposium, pp 201–209

Lehoczky J, Sha L, Ding Y (1989) The rate monotonic scheduling algorithm: exact characterization and average case behavior. In: Proceedings of the real-time systems symposium (RTSS). IEEE Computer Society Press, Los Alamitos, pp 166–171

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

Live Devices, ETAS Group, RTA-OSEK in detail. http://en.etasgroup.com/products/rta

Palencia JC, Harbour MG (1998) Schedulability analysis for tasks with static and dynamic offsets. In: Proceedings of 19th IEEE real-time systems symposium (RTSS), Madrid, Spain

Perathoner S (2006) Evaluation and comparison of performance analysis methods for distributed embedded systems. Master's thesis, Swiss Federal Institute of Technology, Zürich

Project S (2003) Institute of Computer and Communication Network Engineering, Technical University of Braunschweig, Germany. http://www.symta.org

Punnekkat S, Davis R, Burns A (1997) Sensitivity analysis of real-time task sets. In: ASIAN, pp 72–82

Racu R, Jersak M, Ernst R (2005) Applying sensitivity analysis in real-time distributed systems. In: Proceedings of the 11th IEEE real-time and embedded technology and applications symposium (RTAS), San Francisco, USA

Racu R, Hamann A, Ernst R (2006) A formal approach to multi-dimensional sensitivity analysis of embedded real-time systems. In: 18th Euromicro conference on real-time systems (ECRTS). Germany, Dresden, pp 3–12

Redell O, Sanfridson M (2002) Exact best-case response time analysis of fixed priority scheduled tasks. In: Proceedings of the Euromicro conference on real-time systems (ECRTS), pp 165–172

Redell O, Trngren M (2002) Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter. In: Proceedings of the eighth IEEE real-time and embedded technology and applications symposium (RTAS), Washington, DC, USA, pp 164–172

Regehr J (2002) Scheduling tasks with mixed preemption relations for robustness to timing faults. In: Proceedings of the 23rd IEEE real-time systems symposium (RTSS), Austin, TX, December 2002

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

Richter K, Racu R, Ernst R (2003) Scheduling analysis integration for heterogeneous multiprocessor SoC. In: Proceedings 24th international real-time systems symposium (RTSS'03), Cancun, Mexico, December 2003

Seto D, Lehoczky JP, Sha L (1998) Task period selection and schedulability in real-time systems. In: Proceedings of the IEEE real-time systems symposium (RTSS), Madrid, Spain, pp 188–198

Thiele L, Chakraborty S, Naedele M (2000) Real-time calculus for scheduling hard real-time systems. In: Proceedings of the international symposium on circuits and systems (ISCAS), Geneva, Switzerland

Tindell K, Clark J (1994) Holistic schedulability analysis for distributed real-time systems. Microprocess Microprogram Euromicro J (Special issue on parallel embedded real-time systems) 40:117–134

Vestal S (1994) Fixed-priority sensitivity analysis for linear compute time models. IEEE Trans Softw Eng 20(4)

Wandeler E, Thiele L (2006) Real-time calculus (RTC) toolbox

Yerraballi R, Mukkamala R, Maly K, Wahab HA (1993) Issues in schedulability analysis of real-time systems. In: Proceedings of 7th Euromicro workshop on real time systems (EUROMICRO-RTS), June 1993, pp 87–92

Zitzler E, Laumanns M, Thiele L (2002) SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In: Proceedings of evolutionary methods for design, optimisation, and control, Barcelona, Spain, pp 95–100

Razvan Racu Razvan Racu received a Diploma in Computer Engineering from the University of Craiova, Romania, in 2002. He is working since 2002 as research assistant in the Embedded System Design Automation Group of Professor Ernst, at the Institute of Computer and Communication Network Engineering, Technical University of Braunschweig, Germany. His research activities include real-time scheduling, formal timing analysis, perfomance characterization and sensitivity analysis of embedded systems.

Arne Hamann 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 optimisation of heterogeneous distributed real-time systems.

Rolf Ernst Rolf Ernst received a Diploma in Computer Science and a Ph.D. in Electrical Engineering from the University of Erlangen-Nuremberg, Germany, in 1981 and 1987. From 1988 to 1989, he was a Member of Technical Staff in the Computer Aided Design & Test Laboratory at Bell Laboratories, Allentown, PA. Since 1990, 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.