A Formal Approach to Multi-Dimensional Sensitivity Analysis of Embedded Real-Time Systems

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

Abstract

System robustness is a major concern in the design of efficient and reliable state-of-the-art heterogenous embedded real-time systems. Due to complex component interactions, resource sharing and functional dependencies, one-dimensional sensitivity analysis cannot 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 heuristic and a stochastic approach suited for the multi-dimensional sensitivity analysis of large heterogenous embedded systems with complex timing constraints.

1 Introduction

The robustness of an architecture to changes is a major concern in embedded system design. Robustness is important in early design stages 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 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.

First sensitivity analysis approaches known from literature are restricted to very 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, this was the level of technology in the area of real-time systems. 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, the formal methods presented in [7, 3] could not be adopted for such systems.

Instead, new approaches were proposed, based on different performance analysis methods aware of system level dependencies and requirements [5, 6]. In spite of their high feasibility and adaptability, these approaches can only consider variations of *only one* system property at a time. Therefore, they have applicability only for totally independent system components, as an aid to reduce the design space, or when investigating components leading to performance bottlenecks.

In this paper we describe two alternative approaches for the multi-dimensional sensitivity analysis of different parameters of large distributed heterogeneous real-time systems. First, we motivate the importance of the multi-dimensional sensitivity analysis for the design process of state-of-the-art real-time systems. Afterwards, in Sections 3.1 and 3.2 we describe two methods for multi-dimensional sensitivity analysis. In Section 4 we present a set of experiments carried out on an embedded real-time system example. Finally, we compare the two methods and we draw the conclusions.

2 System design challenges

Due to complex system dependencies and global timing constraints, design sensitivity is quite unpredictable, 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.

The system presented in Figure 1 contains two independent subsystems integrated via a network (BUS). The signal processor (DSP) periodically executes a filter task (T_3) that sends data at completion over channel C_3 on the bus to the IP_1 component. The sensor Sens sporadically communicates via the same bus with the control task (T_1) mapped on CPU. On CPU is additionally executed a periodic task

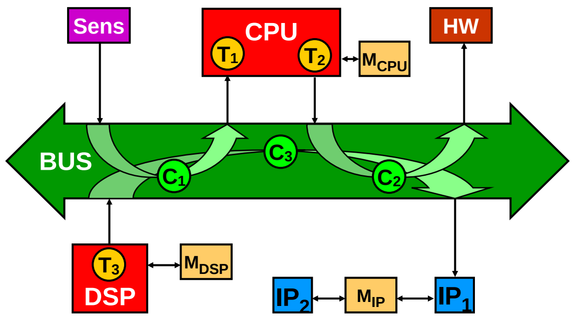


Figure 1. Multi-processor embedded system

(T_2) that sends data over channel C_2 to the hardware component HW. On both, CPU and BUS 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 C_1, C_2, C_3 on BUS 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.

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.

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.

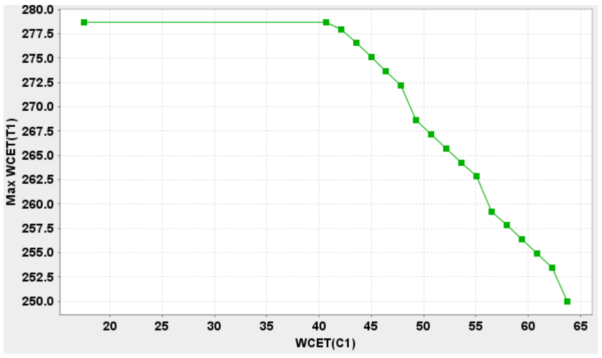
In this context, some issues need to be investigated in order to guarantee high system flexibility. At first, one must identify the system properties that may change during later design steps. These are usually task execution times, the parameters characterizing the activation models, communication volumes, the operational speed of the processing elements, the bandwidth of the communication resources, etc. For these parameters it is necessary to determine the maximum variation of their values that is permitted by the set of constraints. The *sensitivity* of a system property is inversely proportional to the available slack of that property, the smaller the available slack, the higher the sensitivity. The *flexibility* of a property is defined as the inverse function of its sensitivity. In [5] we presented a sensitivity analysis framework for different system parameters assuming different performance metrics.

In that paper we introduced an analysis approach that combines a set of formal equations with a binary search technique, in order to determine the variation limits of different system properties. We described the algorithms for task execution times, processor speeds, bus throughputs and input data rates. However, the proposed technique can only consider variations of only one parameter at a time. Such information is of major importance when looking for the performance bottlenecks of the system, or when dealing with totally independent system parameters, like activation models at system inputs, isolated tasks, or protected IP components.

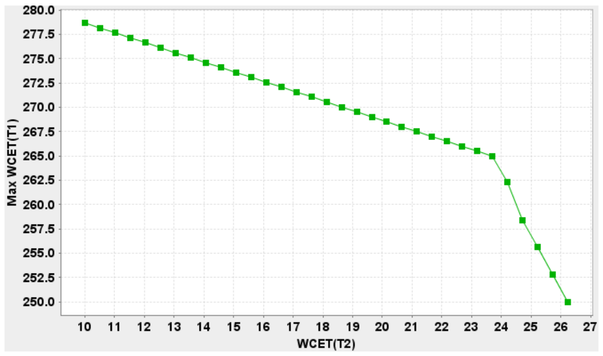
However, in most cases, the system parameters share common properties such that modifications of one parameter typically imply variations of other parameters, as well. For example, consider the system presented in Figure 1: assume that the amount of data sent by sensor Sens increases. This leads to a larger communication time of channel C_1, but, as a consequence results in a larger processing time of task T_1. Therefore, considering only the variation of the communication time of C_1, without taking into account the resulting variation of the execution time of T_1, may lead to constraint violations, and thus, to faulty system configurations. Similar examples are the dependencies between the computation tasks T_2 and T_3, and the communication channels C_2 and C_3, respectively.

Since the dependency is often not explicitly formulated or not known in detail, it is helpful to see the influence of one system parameter on the sensitivity of another, possibly dependent parameter. Figure 2(a) shows the variation of the available slack of the execution demand of T_1 (Y-axis) under different discrete values of the worst-case communication time of C_1 (X-axis). We observe that for values of $WCET_{C_1}$ (worst-case execution/communication time) between 17.5 and 40.0, the available slack of the execution time of T_1 is constant. For values of $WCET_{C_1}$ larger than 40, the flexibility of $WCET_{C_1}$ decreases piecewise linearly with the slope $s_{C_1-T_1} \approx -1.2$.

A second type of dependencies are between system components sharing common resources, like tasks T_1 and T_2 on CPU, or channels C_1, C_2 and C_3 on BUS. Obviously, between the execution or communication times of these tasks, there exist a load dependency as a result of the shared utilization of the hardware resources. Figure 2(b) shows the relation between the worst-case execution demand of task T_2 and the available slack of the execution time of T_1. As it can be observed, for values of $WCET_{T_2}$ between 10.0 and 24.0, the flexibility of $WCET_{T_1}$ linearly decreases with the slope



(a) $WCET_{\mathsf{C}_1} - WCET_{\mathsf{T}_1}$



(b) $WCET_{\mathsf{T}_2} - WCET_{\mathsf{T}_1}$

Figure 2. System parameters with functional and non-functional dependencies

$s_{\mathsf{T}_2-\mathsf{T}_1} \approx -1.0$. This effect is explained by the limited service capability of CPU. If $WCET_{\mathsf{T}_2}$ exceeds 24.0, then the flexibility of $WCET_{\mathsf{T}_1}$ drastically decreases with the slope $s'_{\mathsf{T}_2-\mathsf{T}_1} \approx -5.0$. This characteristic is given by a predefined constrained delay along the path $\mathsf{T}_2 - \mathsf{C}_2 - \mathsf{HW}$.

The information provided by these charts help the designer to keep the system within the required performance bounds, while modifying the initial specification. In this paper we focus on two algorithms to compute the flexibility of various system parameters assuming the simultaneous variation of other system parameters.

3 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. Section 3.1 describes a heuristic algorithm that combines a user-controled search algorithm with a binary search technique. The method is optimized for the analysis of simultaneous variations of two system parameters. For the analysis of simultaneous variations of more than two system parameters, we present a stochastic approach based on evolutionary algorithms.

3.1 Search-based analysis

The main idea of Algorithm 1 is a systematic selection of a set of values for the base parameter, Px. For each selected value of Px is calculated the available slack of the target parameter Py, using the algorithms presented in [5]. The values of the base parameter ($\mathcal{V}_{\mathsf{Px}}$) are searched within a closed interval defined by its initial value ($\mathcal{V}_{\mathsf{Px}}^{init}$) and the maximum/minimum value allowed for that parameter ($\mathcal{V}_{\mathsf{Px}}^{ext}$). The latter is determined by applying the one-dimensional sensitivity analysis (line 2)[5]. Lines 3 and 4 return the available slacks of the target parameter corresponding to the extreme values of the base parameter. The bounding pairs and all other pairs of type ($\mathcal{V}_{\mathsf{Px}}, \mathcal{S}_{\mathsf{Py}}$) computed during analysis, are added to map $\mathcal{M}_{\mathsf{Px,Py}}$.

Function $computeSlack(\mathsf{Py}, \mathcal{V}_{\mathsf{Px}})$ calculates the flexibility of Py considering the system configuration with $\mathcal{V}_{\mathsf{Px}}$ defined as value for Px. The main function of the algorithm is $depthSearch$ called at line 7, explained in detail in the next section.

Algorithm 1 $analyzeParameterPair$

```
INPUT: (Px, Py), the parameters to be analyzed.
OUTPUT: $\mathcal{M}_{\mathsf{Px,Py}}$, a set of points of type $(\mathcal{V}_{\mathsf{Px}}, \mathcal{S}_{\mathsf{Py}})$.
1: get $\mathcal{V}_{\mathsf{Px}}^{init}$, the current value of Px;
2: compute $\mathcal{V}_{\mathsf{Px}}^{ext}$, the largest variation of Px value;
3: $\mathcal{S}_{\mathsf{Py}}^{init} = computeSlack(\mathsf{Py}, \mathcal{V}_{\mathsf{Px}}^{init})$;
4: $\mathcal{S}_{\mathsf{Py}}^{ext} = computeSlack(\mathsf{Py}, \mathcal{V}_{\mathsf{Px}}^{ext})$;
5: add $(\mathcal{V}_{\mathsf{Px}}^{init}, \mathcal{S}_{\mathsf{Py}}^{init})$ to $\mathcal{M}_{\mathsf{Px,Py}}$;
6: add $(\mathcal{V}_{\mathsf{Px}}^{ext}, \mathcal{S}_{\mathsf{Py}}^{ext})$ to $\mathcal{M}_{\mathsf{Px,Py}}$;
7: $depthSearch(1, \mathcal{V}_{\mathsf{Px}}^{init}, \mathcal{V}_{\mathsf{Px}}^{ext}, \mathcal{M}_{\mathsf{Px,Py}})$;
8: return $\mathcal{M}_{\mathsf{Px,Py}}$;
```

3.1.1 Depth search

The function $depthSearch$ described in Algorithm 2 divides the search interval in half, and computes the flexibility of the target parameter for the system configuration assuming the middle value of the interval assigned to the base parameter. The pair determined by this value and the corresponding slack of the target parameter is added to $\mathcal{M}_{\mathsf{Px,Py}}$. Function $depthSearch$ is recursively applied to the left-half and right-half intervals. The recursion terminates if at least one abort condition defined in function $abortCheck()$ holds. The abort conditions are thoroughly presented in Section 3.1.2.

Algorithm 2 *depthSearch*

```
INPUT: depth, the current search depth; $\mathcal{V}_{\mathsf{Px}}^{left}$ and $\mathcal{V}_{\mathsf{Px}}^{right}$, the bounds of the search interval; $\mathcal{M}_{\mathsf{Px,Py}}$ the set of analyzed points.
1: if $abortCheck(\mathsf{depth}, \mathcal{V}_{\mathsf{Px}}^{left}, \mathcal{V}_{\mathsf{Px}}^{right}, \mathcal{S}_{\mathsf{Py}}^{left}, \mathcal{S}_{\mathsf{Py}}^{right})$ then
2:     stop search;
3: $\mathcal{V}_{\mathsf{Px}}^{mid} = (\mathcal{V}_{\mathsf{Px}}^{left} + \mathcal{V}_{\mathsf{Px}}^{right})/2$;
4: $\mathcal{S}_{\mathsf{Py}}^{mid} = computeSlack(\mathsf{Py}, \mathcal{V}_{\mathsf{Px}}^{mid})$;
5: add $(\mathcal{V}_{\mathsf{Px}}^{mid}, \mathcal{S}_{\mathsf{Py}}^{mid})$ to $\mathcal{M}_{\mathsf{Px,Py}}$;
6: $depthSearch(\mathsf{depth}+1, \mathcal{V}_{\mathsf{Px}}^{left}, \mathcal{V}_{\mathsf{Px}}^{mid}, \mathcal{M}_{\mathsf{Px,Py}})$;
7: $depthSearch(\mathsf{depth}+1, \mathcal{V}_{\mathsf{Px}}^{mid}, \mathcal{V}_{\mathsf{Px}}^{right}, \mathcal{M}_{\mathsf{Px,Py}})$;
```

3.1.2 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 ($\mathsf{depth}_{\mathsf{max}}$) that represents the largest number of halving-levels performed by the $depthSearch$ function. If $\mathsf{depth}_{\mathsf{max}}$ is met, the search is aborted (lines 1 and 2 in Algorithm 3). For a depth equal to $\mathsf{depth}_{\mathsf{max}}$ the maximum number of analyzed points is $2^{\mathsf{depth}_{\mathsf{max}}}$.

Algorithm 3 *abortCheck*

```
INPUT: depth, the current search depth; $\mathsf{depth}_{\mathsf{max}}$, the maximum depth; $\mathcal{V}_{\mathsf{Px}}^{left}$ and $\mathcal{V}_{\mathsf{Px}}^{right}$, the bounds of the search interval; $\mathcal{S}_{\mathsf{Py}}^{left}$ and $\mathcal{S}_{\mathsf{Py}}^{right}$, the available slack of Py corresponding to $\mathcal{V}_{\mathsf{Px}}^{left}$ and $\mathcal{V}_{\mathsf{Px}}^{right}$; resolution, the minimum size of the search interval.
OUTPUT: a boolean value
1: if $(\mathsf{depth} > \mathsf{depth}_{\mathsf{max}})$ then
2:     return $true$;
3: else if $\left|\mathcal{V}_{\mathsf{Px}}^{left} - \mathcal{V}_{\mathsf{Px}}^{right}\right| < \mathsf{resolution}$ then
4:     return $true$;
5: else if $(\mathcal{S}_{\mathsf{Py}}^{left} = \mathcal{S}_{\mathsf{Py}}^{right})$ then
6:     return $true$;
7: else
8:     return $false$;
```

A second parameter controlling the search algorithm is the minimum size of the search interval. If the resolution of the search domain becomes smaller than resolution, no further points are investigated (lines 3 and 4 in Algorithm 3). 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 [5, 4], we proved the monotonic relation between the execution time interval of a task and the available system slack.

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

$$\mathcal{S}_{\mathsf{Py}}^{left} = \mathcal{S}_{\mathsf{Py}}^{right} \tag{1}$$

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

$$\mathcal{S}_{\mathsf{Py}}^{in} = \mathcal{S}_{\mathsf{Py}}^{left} = \mathcal{S}_{\mathsf{Py}}^{right} \tag{2}$$

Proof Assume that

$$\mathcal{S}_{\mathsf{Py}}^{in} > \mathcal{S}_{\mathsf{Py}}^{left} \quad \text{and} \quad \mathcal{S}_{\mathsf{Py}}^{in} > \mathcal{S}_{\mathsf{Py}}^{right} \tag{3}$$

Since the algorithm presented in Section 3.1 investigates only the upper bound of the execution time interval of Px, i.e. the $WCET_{\mathsf{Px}}$ values, by computing the slack $\mathcal{S}_{\mathsf{Py}}^{in}$ corresponding to $\mathcal{V}_{\mathsf{Px}}^{in}$, it is guaranteed that $\mathcal{S}_{\mathsf{Py}}^{in}$ is valid for all values $\mathcal{V}_{\mathsf{Px}}$ within the interval $[BCET_{\mathsf{Px}}; \mathcal{V}_{\mathsf{Px}}^{in}]$.

Because $[BCET_{\mathsf{Px}}; \mathcal{V}_{\mathsf{Px}}^{left}] \subset [BCET_{\mathsf{Px}}; \mathcal{V}_{\mathsf{Px}}^{in}]$, Equation 3 says that there is at least one value in interval $[BCET_{\mathsf{Px}}; \mathcal{V}_{\mathsf{Px}}^{in}]$ that determines a slack $\mathcal{S}_{\mathsf{Py}}$ smaller than the minimum slack computed for that interval. Hence, Equation 3 is never valid.

A similar proof can be carried out for the case

$$\mathcal{S}_{\mathsf{Py}}^{in} < \mathcal{S}_{\mathsf{Py}}^{left} \quad \text{and} \quad \mathcal{S}_{\mathsf{Py}}^{in} < \mathcal{S}_{\mathsf{Py}}^{right} \tag{4}$$

Since $[BCET_{\mathsf{Px}}; \mathcal{V}_{\mathsf{Px}}^{in}] \subset [BCET_{Px}; \mathcal{V}_{\mathsf{Px}}^{right}]$, results that there is at least one value in $[BCET_{\mathsf{Px}}; \mathcal{V}_{\mathsf{Px}}^{right}]$ that leads to a slack $\mathcal{S}_{\mathsf{Py}}$ smaller than the lowest slack calculated for that interval. Therefore, Equation 4 could not be valid.

Hence, if Equation 1 is valid, all values between $\mathcal{V}_{\mathsf{Px}}^{left}$ and $\mathcal{V}_{\mathsf{Px}}^{right}$ determine a slack for Py equal to $\mathcal{S}_{\mathsf{Py}}^{left}$ and $\mathcal{S}_{\mathsf{Py}}^{right}$.

Thus, if Equation 1 holds for two values of the base parameter, then no further search is required within the interval determined by these values. Such a scenario can be observed in Figure 2(a) for values of $WCET_{C_1}$ between 17.5 and 40.5.

However, not all analyzed parameters have a monotonic relation. For instance, changing processor speeds or bus throughputs may lead to anomalous behavior. Such a scenario can be observed in Figure 3, showing the two-dimensional sensitivity analysis of CPU's speed and the throughput of BUS. In that case all intermediate points have to be considered, even though there exists two values for the base parameter for which the corresponding slacks of the target parameter satisfy Equation 1.

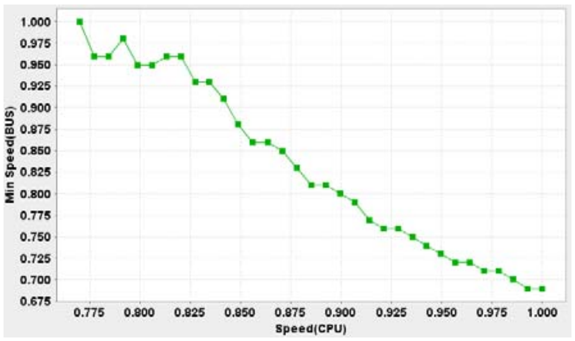


Figure 3. Resource speed anomaly

3.2 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 [2], which uses multi-dimensional evolutionary search techniques [1, 8].

We first give a short description of how we use the exploration framework to perform multi-dimensional sensitivity analysis (Section 3.2.1). We then give details on several important aspects: search space encoding (Section 3.2.2), the creation of an initial population used as starting point for the exploration (Section 3.2.3), and the variation operators used to guide the exploration (Sections 3.2.4 and 3.2.5).

3.2.1 Analysis idea

Classical applications of exploration frameworks for complex distributed systems assume a 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. the configuration results in a feasible solution.

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 maximization of the latter.

Note that our exploration framework performs a 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.

3.2.2 Search space encoding

A system property value combination considered during the stochastic multi-dimensional sensitivity analysis is directly 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. $A = (a_1, a_2, a_3)$.

3.2.3 Initial population

Algorithm 4 describes the creation of the initial population. In the first part (lines 1 to 3) it uses one-dimensional sensitivity analysis [5] to calculate the available slack for each considered system property. These information is used to generate individuals for the initial population representing extreme points of the sought-after sensitivity front.

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 determined by the initial and the extreme property values.

3.2.4 Crossover operator

The crossover operator described in Algorithm 5 implements a heuristic strategy to converge towards the sensitivity front, i.e. the boundary between working and non-working systems. It takes as input two parent individuals A and B and generates two offsprings C and D by using the generalized mean function described in Definition 1.

Algorithm 4 Initial Population

```
INPUT:  System properties $\mathsf{P}_1, \ldots, \mathsf{P}_n$
        Initial property values $\mathcal{V}^{init}_{\mathsf{P}_1}, \ldots, \mathcal{V}^{init}_{\mathsf{P}_n}$
        Initial population size $\alpha > n$
OUTPUT: Initial population $\mathcal{I}$
1: for $(i = 1;\ i <= n;\ i = i + 1)$ do
2:     $\mathcal{V}^{ext}_{\mathsf{P}_i} = computeSlack(\mathsf{P}_i)$
3:     add vector $(\mathcal{V}^{init}_{\mathsf{P}_1}, \ldots, \mathcal{V}^{ext}_{\mathsf{P}_i}, \ldots, \mathcal{V}^{init}_{\mathsf{P}_n})$ to $\mathcal{I}$
4: while $(|\mathcal{I}| < \alpha)$ do
5:     for $(i = 1;\ i <= n;\ i = i + 1)$ do
6:         Choose random
           $\mathcal{V}^{rand}_{\mathsf{P}_i} \in \left[\min\left(\mathcal{V}^{init}_{\mathsf{P}_i}, \mathcal{V}^{ext}_{\mathsf{P}_i}\right), \max\left(\mathcal{V}^{init}_{\mathsf{P}_i}, \mathcal{V}^{ext}_{\mathsf{P}_i}\right)\right]$
7:     add vector $\left(\mathcal{V}^{rand}_{\mathsf{P}_1}, \ldots, \mathcal{V}^{rand}_{\mathsf{P}_n}\right)$ to $\mathcal{I}$
```

Definition 1 (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; $k \to \infty$: $\max(x_1, \ldots, x_n)$.

Figure 4 shows 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 Figure 2(b)). For 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.

Note that the crossover operator automatically ensures that the extreme values $\mathcal{V}^{init}_{\mathsf{P}_i}$ and $\mathcal{V}^{ext}_{\mathsf{P}_i}$ determined during the creation of the initial population (Section 3.2.3) for each considered system property P_i are not violated. This improves the exploration process since the number of generated invalid system property assignments, i.e. due to resource overload or constraint violation, is reduced.

3.2.5 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.

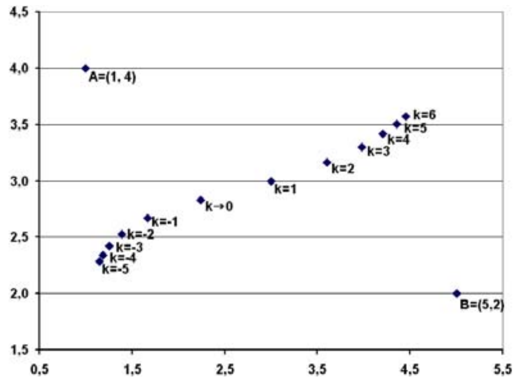


Figure 4. Generalized mean between A and B for different k

Algorithm 5 Crossover operator

```
INPUT:  $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$
        $k_{min}$ and $k_{max}$ with $k_{max} \geq k_{min}$
OUTPUT: $C = (c_1, \ldots, c_n)$ and $D = (d_1, \ldots, d_n)$
1: Choose random $k_1 \in [k_{min}, k_{max}]$
2: Choose random $k_2 \in [k_{min}, k_{max}]$
3: for $(i = 1;\ i <= n;\ i = i + 1)$ do
4:     if $(k_1 = 0)$ then
5:         $c_i = \sqrt{a_i \times b_i}$
6:     else
7:         $c_i = M_{k_1}(a_i, b_i)$
8:     if $(k_2 = 0)$ then
9:         $d_i = \sqrt{a_i \times b_i}$
10:    else
11:        $d_i = M_{k_2}(a_i, b_i)$
```

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 mutation operator is described by Algorithm 6. It takes as input one parent individual, from which it creates one offspring by randomly increasing or decreasing each property value by a bounded random percentage $percent_{max}$. 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 $\mathcal{V}^{init}_{\mathsf{P}_i}$ and $\mathcal{V}^{ext}_{\mathsf{P}_i}$ calculated during the creation of the initial population (Section 3.2.3).

Algorithm 6 Mutation operator

```
INPUT:  $A = (a_1, \dots, a_n)$
        minimum values $a_1^{min}, \dots, a_n^{min}$
        maximum values $a_1^{max}, \dots, a_n^{max}$
        $percent_{max} < 1$
OUTPUT: $B = (b_1, \dots, b_n)$
1: for $(i = 1;\ i <= n;\ i = i + 1)$ do
2:     Choose random percentage $p \in ]0, percent_{max}]$
3:     Choose random boolean $bool$
4:     if $(bool)$ then
5:         $b_i = \min\left(a_i \times (1 + p), a_i^{max}\right)$
6:     else
7:         $b_i = \max\left(a_i \times (1 - p), a_i^{min}\right)$
```

4 Example

We presented two algorithms for the multi-dimensional sensitivity analysis of different system parameters. In this section we introduce and discuss a set of experiments on the embedded real-time system presented in Figure 5. In Section 4.2 and 4.3 we show the results obtained using the heuristic and the stochastic methods, respectively. In Section 4.4 we compare the two approaches and we present a set of numbers showing the complexity of these algorithms.

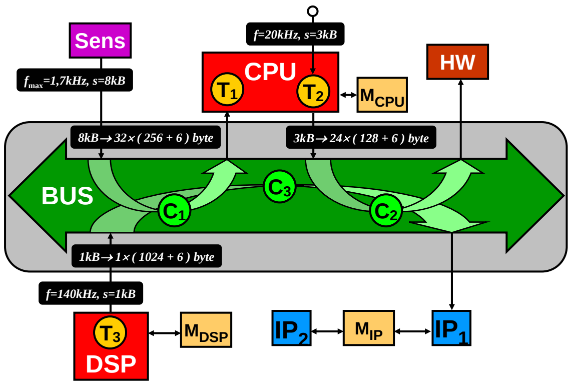


Figure 5. The initial system configuration

4.1 Set-Up

The actuators There are three actuators: the sensor sporadically sends data blocks of $8kB$ size to T_1, with a maximum sending frequency of $1.66kHz$, which corresponds to a *sporadic event model* with a minimum sporadic period of $1/1.66kHz = 600\mu s$. Process T_2 is periodically activated by the RTOS (real-time operating system) on the CPU with a period of $1/20kHz = 50\mu s$ and sends $3kB$ of data at completion. The signal processing application T_3 on DSP has a sampling frequency of $10kHz$, corresponding to a sampling period of $10\mu s$. T_3 sends $1kB$ of data at the end of each execution.

The network Instead of sending the complete data block, the data packets are fragmented to avoid too long blocking times. Each $8kB$ data block from Sens is split into 32 packets of $262byte$ each, $256bytes$ plus $6bytes$ protocol overhead–address, length, and CRC. The $3kB$ blocks from T_2 are split into 24 packets of $(128+6) = 134bytes$. 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 $6byte$ protocol information. The initial network speed is $480MB/s$.

The overall average network load L_{BUS} is:

$$\begin{aligned} L_{\mathsf{BUS}} &= L_{\mathsf{Sens-CPU}} + L_{\mathsf{CPU-HW}} + L_{\mathsf{DSP-IP_1}} \\ &= 14MB/s + 62.4MB/s + 103.2MB/s \\ &= 179.6MB/s \end{aligned}$$

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.

task	execution time	channel	communication time
T_1	250	C_1	17.50
T_2	10	C_2	6.50
T_3	[2;4]	C_3	2.15

Table 1. Execution and communication times

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 Figure 5. The constraints are listed in Table 2.

functional path	max delay
Sens – C_1 – T_1	$400\mu s$
T_2 – C_2 – HW	$300\mu s$
T_3 – C_3 – IP_1	$10\mu s$

Table 2. Real-time constraints

4.2 Search-based analysis

We grouped the task pairs in three groups depending on the dependencies between them. The first group are the tasks with functional dependencies, i.e, the tasks belonging to the same dependency path. The obtained results are shown in Figure 6. 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 $WCET_{\mathsf{T3}}$.

The second group of task pairs are the tasks with load dependencies, i.e. tasks or channels mapped on the same resource. Figure 7 shows the results obtained for different pairs of communication channels mapped on BUS. 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 are the tasks with no direct dependency, i.e tasks that are mapped on different resources and without any functional dependency between them. Figure 8 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 Figure 8(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 $\mathsf{T}_2 - \mathsf{C}_3$. 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 $WCET_{\mathsf{T}_2}$ lead to slight variations of the available slack of C_3.

The second type of investigated parameters are the speed of the processing and communication elements. Figure 9 shows the two-dimensional sensitivity analysis of the resource speed factors. Since the DSP and the CPU does not share common application paths, the flexibility of the DSP's speed is independent of the speed of CPU, as shown in Figure 9(a). Largely, are also the speeds of DSP and BUS (Figure 9(b)). A stronger correlation exists between the speed of CPU and the speed of BUS as shown in Figure 3 at page 5. This is due to the two application paths traversing these resources.

4.3 Stochastic analysis

Figures 10(a) and 10(b) show the three-dimensional sensitivity analysis results obtained using the stochastic approach presented in Section 3.2. Note that for both analyses our exploration framework generated 100 generations with each 200 individuals, taking approximately 5 minutes on a 2.00GHz 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 Figure 8(a) obtained using the heuristic approach.

4.4 Accuracy and complexity

Figure 11 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.

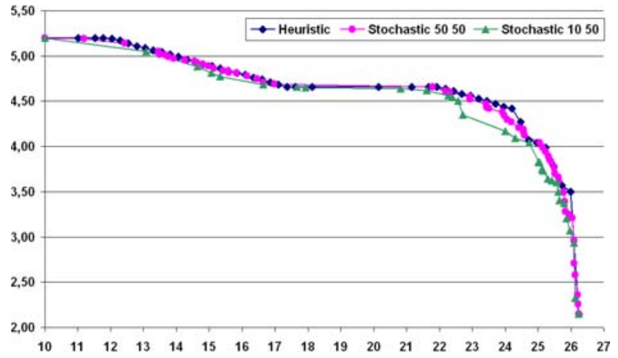


Figure 11. Comparison of the two approaches

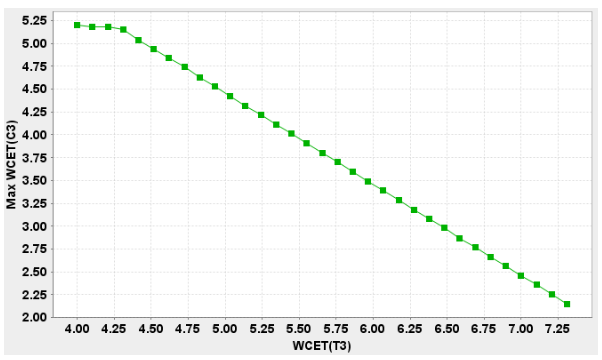
Table 3 shows the run-times (in ms) of the search-based and stochastic algorithms. The search-based algorithm was applied on two set of properties, one with monotonic relation (w/s_step) and the other one with non-monotonic relation (w/os_step). The experiments were carried out on a 2.00GHz Athlon64 standard PC. The complexity of the search-based algorithm is defined by the search-depth, while the complexity of the stochastic algorithm is determined by the number of generations and the number of individuals per generation.

search-based			stochastic	
depth	$w/\ s\_step$	$w/o\ s\_step$	(#gen;#ind)	$WCET$
3	1469	1526	(10;50)	16806
5	4334	4580	(30;50)	50327
7	15906	17924	(50;50)	85141

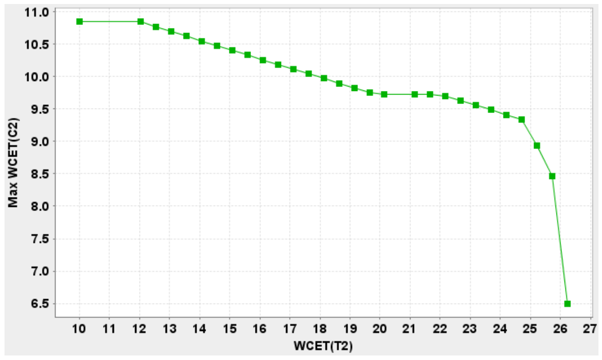
Table 3. The run-times of the algorithms (ms)

5 Conclusion

Design robustness, measured as sensitivity to design parameter variations, is a key concern in embedded system design. In this paper we presented a full-search and a stochastic approach for multi-dimensional sensitivity analysis. The full search approach provides better performance for 2 dimensions than the stochastic approach, but is limited to monotonous functions and cannot be easily extended to more than two dimensions.

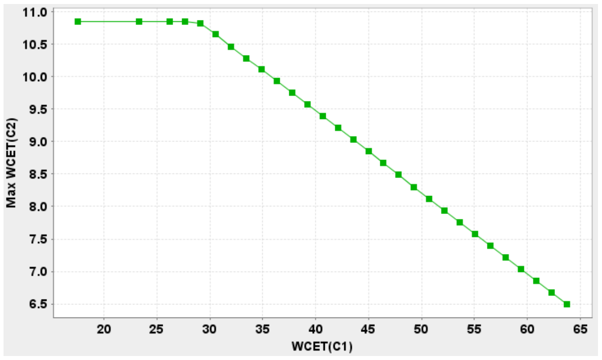


(a) $WCET_{\mathsf{T}_3} - WCET_{\mathsf{C}_3}$

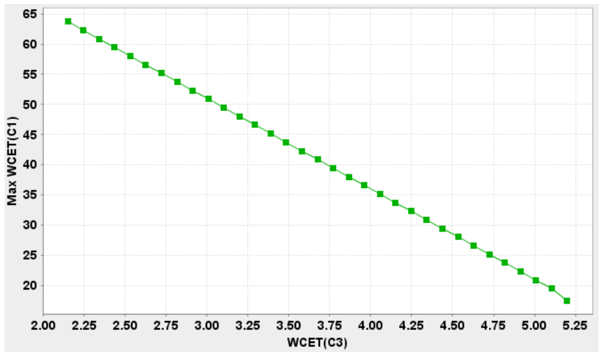


(b) $WCET_{\mathsf{T}_2} - WCET_{\mathsf{C}_2}$

Figure 6. Tasks with functional dependencies

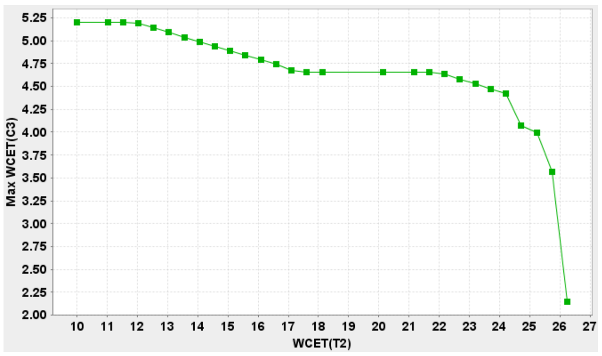


(a) $WCET_{\mathsf{C}_1} - WCET_{\mathsf{C}_2}$

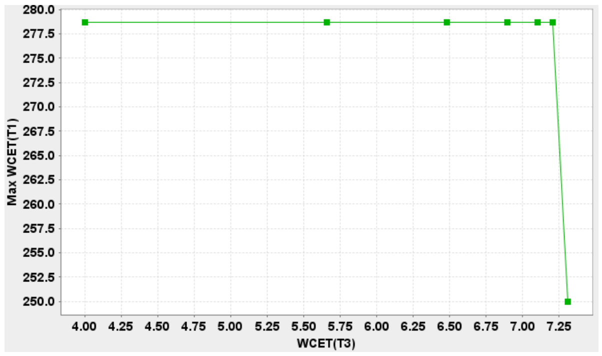


(b) $WCET_{\mathsf{C}_3} - WCET_{\mathsf{C}_1}$

Figure 7. Tasks with load dependencies

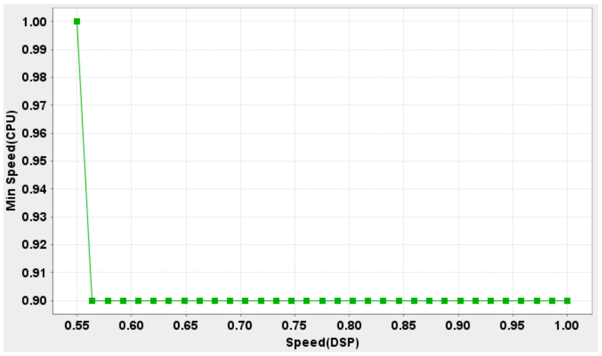


(a) $WCET_{\mathsf{T}_2} - WCET_{\mathsf{C}_3}$

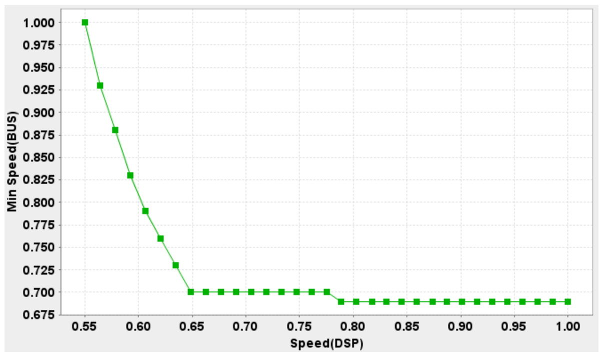


(b) $WCET_{\mathsf{T}_3} - WCET_{\mathsf{T}_1}$

Figure 8. Tasks with no direct dependency

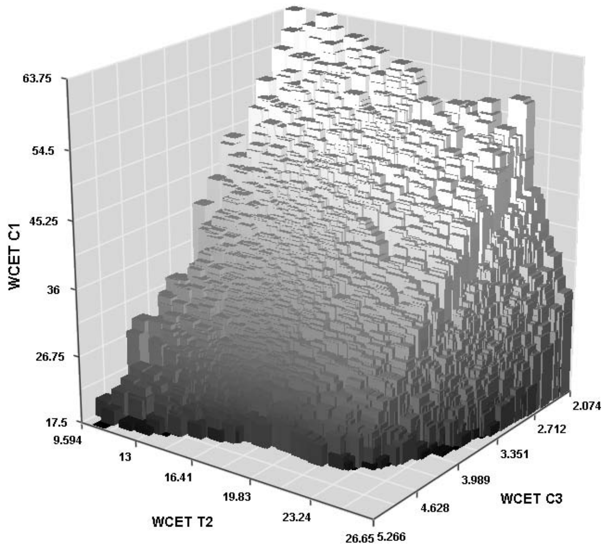


(a) $Speed_{\mathsf{DSP}} - Speed_{\mathsf{CPU}}$

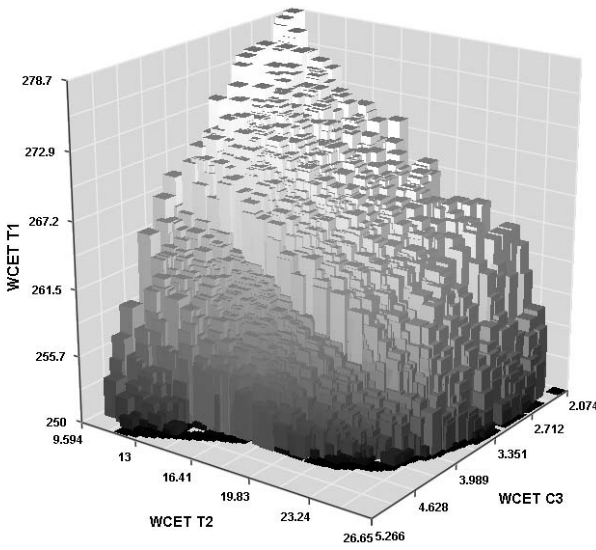


(b) $Speed_{\mathsf{DSP}} - Speed_{\mathsf{BUS}}$

Figure 9. Resource speed factors



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



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

Figure 10. Three-dimensional WCET sensitivity

References

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

[2] A. Hamann, M. Jersak, K. Richter, and R. Ernst. Design space exploration and system optimization with SymTA/S - Symbolic Timing Analysis for Systems. In *Proc. 25th International Real-Time Systems Symposium (RTSS'04)*, Lisbon, Portugal, Dec. 2004.

[3] S. Punnekkat, R. Davis, and A. Burns. Sensitivity analysis of real-time task sets. *ASIAN*, pages 72–82, 1997.

[4] R. Racu and R. Ernst. Scheduling anomaly detection and optimization for distributed systems with preemptive task-sets. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, San Jose, CA, USA, 2006. Accepted for publication.

[5] R. Racu, M. Jersak, and R. Ernst. 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, CA, USA, 2005.

[6] J. Regehr. Scheduling tasks with mixed preemption relations for robustness to timing faults. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS)*, Austin,Texas, December 2002.

[7] S. Vestal. Fixed-priority sensitivity analysis for linear compute time models. *IEEE Transactions on Software Engineering*, 20(4), april 1994.

[8] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for multiobjective optimization. In *Proc. Evolutionary Methods for Design, Optimisation, and Control*, pages 95–100, Barcelona, Spain, 2002.