Efficient Constraint Handling during Designing Reliable Automotive Real-Time Systems

Florian Pölzlbauer[1], Iain Bate[2], and Eugen Brenner[3]

[1] Virtual Vehicle, Graz, Austria
[2] University of York, Department of Computer Science, York, United Kingdom
[3] Graz University of Technology, Institute for Technical Informatics, Graz, Austria

Abstract. In modern embedded systems, e.g. avionics and automotive, it is not unusual for there to be between 40 and 100 processors with a great deal of the software having hard real-time requirements and constraints over how, when and where they execute. The requirements and constraints are essential to the overall systems dependability and safety (e.g. to ensure replicas execute on different hardware). This leads to a complex design space exploration (DSE) problem which cannot be practically solved manually especially if the schedule is to be maintained.

In this paper it is shown that dealing with the constraints using a conventional state of the art "System Configuration Algorithm" is less efficient, less effective and does not scale well. This issue can be improved by performing constraint pre-processing as well as constraint encoding. It is shown that our approach can handle typical industrial requirements that come from the automotive industry's AUTOSAR standard in an efficient way.

Keywords: design constraints, system configuration, task allocation, efficient design space exploration, real-time systems.

1 Introduction

In the past, automotive electronics were designed in a federated manner. Most functionality was implemented by special-purpose hardware and software. Therefore one control unit performed only one or at most a limited number of individual functions, and functions had their own dedicated hardware. As the functionality steadily increased, the number of control units has also increased. Nowadays cars contain up to 80 control units.

During the last several years, a paradigm shift has occurred in the automotive domain. The design of electronics has moved from a hardware-oriented to a software/function-oriented approach. This means that the functionality is mainly based on software that is executed on general-purpose hardware. In order to enable this trend a middleware (AUTOSAR [1]) was introduced, which separates the application software from the underlying hardware.

In order to develop such reliable and safety-relevant software-based systems, several engineering steps have to be performed. Besides developing the systems functionality, designing the software architecture and implementing software components, several configuration steps have to be performed. These are essential if systems are to meet their reliability requirements (e.g. in the form of timing requirements being met), availability requirements (e.g. where task replicas have to be placed on different processors), and safety requirements (e.g. where tasks have to be allocated and executed in a particular way).

- task allocation: local assignment of tasks to processors
- data routing: finding a route (via buses and gateways) for data transmission between tasks that reside on different processors
- frame packing: packing application messages into bus frames
- scheduling: planning of the temporal attributes of the system (e.g. priority assignment to tasks and frames)
- system performance evaluation: schedulability, resource utilization, etc.

Due to the increasing system complexity, high number of design constraints and safety as well as reliability demands, finding feasible system configurations is a challenging and error-prone task, if performed manually. In order to disburden engineers from these tasks, automized system configuration generation is needed. This will enable the engineers to spend more time on actually designing the systems functionality.

1.1 Related Works

In the literature, the task of finding a system configuration is often referred to as the task allocation problem (TAP). The TAP consists of two principal parts: allocating tasks to processors and messages to communication buses, and then assigning attributes such as the priority to tasks and messages. The TAP should be solved such that all essential requirements (or constraints) are met (e.g. task deadlines) and that the objectives are optimized (e.g. the minimum number of processors are used).

The TAP has been the subject of a great deal of research over the last couple of decades ranging from the early works that dealt with independent tasks to later work that handles more complex requirements, e.g. dependent tasks (or transactions) in [4, 6]. More recent work has considered other constraints, such as tasks having to reside on a particular processor [11].

In [16] the aspect of *extensibility* is tackled. Thereby system configurations are optimized to tolerate increasing worst-case execution times (WCET). This metric can be used to estimate how many tasks and messages could be included in future systems. In [9] even more attributes (e.g. periods) are subject to variation, and multi-dimensional robustness analysis is performed.

In [8] the issue of *system configuration upgrade* is tackled. Starting from a given initial system configuration, the search algorithm (based on simulated annealing (SA)) searches for an improved/optimized system configuration. Thereby minimal changes between the initial and the optimized system configuration are preferred.

Although the TAP has been addressed from different perspectives, some common issues can be identified: Firstly, almost all works focus on creating a new system configuration "from scratch". Although the configuration may be designed with respect to extensibility [9,15,16], the necessary design steps to actually perform a system configuration upgrade are not shown. Only [8] tackles this issue. Secondly, the works only deal with simple design constraints that are insufficient for the needs of many critical systems including those based on the AUTOSAR standard. However, when designing dependable, real-life, reliable, long-life software-based systems, more sophisticated design constraints need to be taken into account (for details see table 1).

1.2 Contribution and Outline

The contributions of this work are:

- provide an overview of industrial relevant design constraints
- provide methods that satisfy these constraints in an efficient way
- show how these methods can be incorporated into *system configuration optimization frameworks*
- present experimental results that evidence the efficiency of the approach

The work is structured as follows: In section 2 we present an overview of design constraints that typically are present in industrial system configuration problems. Then we present *when* each of these constraints can be satisfied (= constraint satisfaction time). Later we show *how* each of these constraints can be satisfied. In section 3 we present an *optimization framework* for finding near-optimal system configurations, which incorporates the methods for satisfying the design constraints. In section 4 we present experimental results that evidence the efficiency of the proposed methods. Finally we draw our conclusions and provide an outlook on future research directions.

The following symbols and abbreviations are used in this work.

Symbol	Description	Symbol	Description	Symbol	Description
t	task	B	set of bus systems	adm	admissible
c	task cluster	P	set of processors	ded	dedicated
m	message	dyn	dynamic	ex	excluded

2 Design Constraints

Design constraints may have a wide variety of sources. Most relevant are:

- safety considerations: If safety analysis of the entire system has been performed (e.g. hazard and risk analysis, in accordance with ISO 26262 [2]), safety requirements can be derived. These impose constraints on design decisions.

- compatibility to legacy systems: Automotive systems are usually designed in an evolutionary fashion. A previous version of the system is taken as a starting point and is extended with additional features, in order to satisfy current demands/requirements. Thus, legacy components may impose constraints on design decisions.
- engineer's experience: Engineers who have been designing similar systems typically have figured out "best practices". These may exclude certain design decisions, thus imposing additional constraints.

Within the automotive domain, the AUTOSAR standard [1] has positioned itself as a leading standard. Within the AUTOSAR standard, design constraints which might occur have been specified in the *AUTOSAR system template*. Therein, a variety of constraint-types can be found. However, these constraints are not only relevant for automotive systems, and could easily be applied to other domains (e.g. rail, aerospace, automation, ...). Table 1 provides a summary of the constraint-types. They can be categorized within 6 classes.

Table 1. Constraint-Types specified within AUTOSAR System Template

Constraint-class	**Constraint-type**	**Literature**
A: limited resources	A-1: processor CPU speed	yes
	A-2: processor memory	yes
	A-3: bus bandwidth	yes
B: real-time behaviour	B-1: task deadline	yes
	B-2: communication deadline	yes
	B-3: end-to-end deadline	yes
C: allocation (task to processor)	C-1: dedicated processors	yes
	C-2: excluded processors	yes
	C-3: fixed allocation	yes
D: dependencies (task to task)	D-1: grouping	no*
	D-2: separation	yes
E: data routing (data to bus)	E-1: processor-internal only	no*
	E-2: dedicated buses	no
	E-3: excluded buses	no
	E-4: same bus	no
	E-5: separated buses	no
F: frame packing (data to frame)	F-1: dedicated frame	no
	F-2: same frame	no
	F-3: separated frames	no

* not stated as a constraint, but used as means to reduce bus utilization

Some of these constraints are relevant for a wide range of software systems. E.g. all embedded software must content itself with limited resources. Thus these constraints are well studied in the literature. However, for reliable systems, additional constraints need to be taken into account. Most safety-related systems must guarantee real-time behaviour, especially if human life is at risk (e.g. drive-by-wire application in a car). Hence, safety analyses are performed in order to identify potential risks, and derive adequate safety goals how to address these risks. A strategy (safety concept) is derived, how to achieve these safety goals. The safety concept must be satisfied/implemented by the architecture. This imposes additional design constraints on the architecture. E.g. a typical safety concept is to use redundancy and replication [7]. If a triple modular redundant approach is used to improve availability and reliability then different physical processors and communication paths need to be enforced in order to avoid common mode failures or byzantine effects. Therefore replicated tasks must not reside on the same processor (task separation), certain processors are inadequate for handling certain tasks (excluded processors), and data must be transferred via separated buses, probably even within separated bus frames.

Concluding: In order to satisfy the safety goals and thus comply to safety standards [2] and legal regulations, a set of highly heterogeneous design constraints need to be handled. This imposes significant effort on engineers. Therefore methods for efficient constraint handling are needed.

It is interesting to note, that several constraint-types are not addressed in the literature. Especially constraints that focus on the configuration of the communication infrastructure have not been tackled. This can be explained, because most works on system configuration (e.g. task allocation) use simplified models for cross-processor communication. These models do not cover all relevant details of the communication infrastructure, and thus the use of detailed constraints seems obsolete. In real dependable systems though, these constraints are of high importance.

2.1 How to Satisfy Constraints in an Efficient Way

During designing safety-relevant distributed real-time systems, that are mainly empowered by the use of software architectures (like the AUTOSAR middleware), the question arises: How can the set of highly heterogeneous constraints be handled and satisfied in an efficient way? By analysing the constraints, we can identify two concepts to satisfy these constraints. Each concept can be applied to a sub-set of the constraints. By combining both concepts, all relevant constraints can be handled and satisfied.

1. pre-processing: Certain constraints can be resolved before the DSE. This way, the constraints are always satisfied during the DSE. Figuratively speaking: The constraints are removed from the design space, thus reducing the design space by removing infeasible regions.
2. encoding: Constraints that cannot be resolved have to be encoded into the search algorithm, e.g. into the objective-function. This way the search is guided towards configurations that satisfy the constraints.

Basically, each constraint-type could be addressed by "encoding". However, this option is not very efficient, since no guarantee of constraint satisfaction can be given. "Resolving" a constraint-type can give that guarantee. Thus the goal is to resolve as many constraints as possible.

Resolve Constraints before Design Space Exploration

Table 2 shows, which constraints can be resolved, which constraints cannot be resolved, and why that is the case. In order to resolve constraints before performing the DSE, the following rules have to be applied:

Table 2. Constraint Satisfaction Time: "before" or "during" Design Space Exploration

Type	before	during	Rationale
A-1		x	CPU utilization can only be checked after task allocation
A-2		x	memory utilization can only be checked after task allocation
A-3		x	bus utilization can only be checked after message routing and frame packing
B-1		x	can only be checked after scheduling
B-2		x	can only be checked after scheduling
B-3		x	can only be checked after scheduling
C-1	x		a set of admissible processors can be calculated
C-2	x		a set of admissible processors can be calculated
C-3	x		allocation algorithm does not modify the allocation
D-1	x		tasks can be grouped (forming a task cluster); task clusters are handled as "single elements" by task allocation
D-2		x	a set of excluded processors can be derived dynamically
E-1	x		sender- and receiver-tasks can be grouped
E-2	x		a set of admissible buses can be calculated
E-3	x		a set of admissible buses can be calculated
E-4	x		group sender-tasks; group receiver-tasks
E-5		x	message routing results from task allocation
F-1	x		only the dedicated frame will be used
F-2	x		demand E-4; perform frame packing in two phases
F-3	x		perform frame packing in two phases

F-1: Dedicated frame packing is typically used, because the same *frame catalog* is used within different cars. To satisfy this constraint, messages that have this constraint associated, will only be packed into the dedicated frame.
E-1: By grouping the sender- and the receiver-task (forming a task-cluster), we can make sure that the task allocation algorithm will allocate both tasks to the same processor. Thus, the communication between these tasks is always performed processor-internal.
D-1: Similar to E-1, this constraint can be resolved by grouping the associated tasks (forming a task-cluster).
E-2 & E-3: Based on these sets, a set of admissible buses can be calculated for each message.

$$B_{adm} = \begin{cases} B \setminus B_{ex} & \text{if} \quad B_{ded} = \{\} \\ B_{ded} \setminus B_{ex} & \text{otherwise} \end{cases} \tag{1}$$

This admissible message-routing implies a set of admissible processors X for the sender- and receiver-task of this message. Only processors connected to the admissible buses of the message are potential candidates for hosting the sender- and receiver-task.

$$P_{adm}^{(t \to m \to t)} = P \text{ connected to } B_{adm} \tag{2}$$

Since a task may send and receive several messages, only the intersected set X is a potentially admissible processor for each task.

$$X = \bigcap P_{adm}^{(t \to m \to t)} \tag{3}$$

C-1 & C-2: Based on these sets, a set of admissible processors can be calculated for each task. Thereby, the set of admissible buses (derived from E-2 & E-3) of the sent/received messages has also to be taken into account.

$$P_{adm} = \begin{cases} (P \cap X) \setminus P_{ex} & \text{if} \qquad P_{ded} = \{\} \\ (P_{ded} \cap X) \setminus P_{ex} & \text{otherwise} \end{cases} \tag{4}$$

If tasks are grouped (forming a task cluster), the set of admissible processors for a task cluster c is:

$$P_{adm}^{(c)} = \bigcap_{t \in c} P_{adm} \tag{5}$$

C-3: If an allocation is fixed, the task allocation algorithm will not modify that allocation.
E-4: Two messages can only be routed via the same bus, if their sender-tasks reside on the same processor and also their receiver-tasks reside on the same processor. Thus, E-4 can be satisfied by two D-1 constraints.
F-2: Two messages can only be packed into the same frame, if both messages are sent from the same processor and routed via the same bus. This can be stated by E-4. In addition, frame packing is performed in two phases. In phase 1, messages that must be packed into the same frame are packed into the same frame. In phase 2, all remaining messages that have not been packed yet are packed into frames (either into new frames or adding them into existing frames).
F-3: Frame packing is performed in two phases. In phase 1, messages that must be packed into separated frames are each packed into a separate frame. In phase 2, all remaining messages that have not been packed yet are packed into frames (either into new frames or adding them into existing frames).

Tackle Constraints during Design Space Exploration
Some of the constraints that cannot be resolved before performing the DSE, can be addressed during the DSE, by applying the following rules:
D-2: The set of admissible processors can be updated dynamically (during the DSE).

$$P_{adm.dyn} = P_{adm} \setminus P_{ex.dyn} \tag{6}$$

$$P_{ex.dyn} = P \text{ of tasks that the current task must be separated from} \tag{7}$$

Concluding: For a set of constraints (A-1, A-2, A-3, B-1, B-2, B-3, E-5) no rules how to satisfy them, could be derived. Consequently, these constraints must be tackled somehow else. An elegant way to do this, is to encode them into the search algorithm. Thereby they can either be represented as a *mandatory* or as *desired*. However, the following implications should be taken into account, when deciding between these options:

– mandatory: If a mandatory constraint is violated, the configuration is treated as being *infeasible*. Thus it will be rejected. Consequently, the configuration is not considered as the starting point for generating new configurations.
– desired: A configurations that does not satisfy a desired constraint is not rejected. Instead it is punished by a high *cost value*. However, the configuration can still be picked as the starting point for subsequent exploration steps.

The difference may sound minor, but actually has significant impact on the DSE. Using *desired constraints* enables the search to gradually traverse through infeasible regions. However, even configurations with "moderate" cost may be infeasible. Using *mandatory constraints* ensures that all constraints are satisfied for feasible configurations.

Table 3. Constraint Encoding: as "mandatory" or as "desired"

Type	**mandatory**	**desired**	**Rationale**
A-1	x		utilization $\leq$ 100% required for schedulability
A-2		x	utilization $\leq$ 100% not required for schedulability
A-3	x		utilization $\leq$ 100% required for schedulability
B-1		x	guide search through un-schedulable regions
B-2		x	guide search through un-schedulable regions
B-3		x	guide search through un-schedulable regions
D-2	xx	x	depending on source of constraint (e.g. safety analysis)
E-5	xx	x	depending on source of constraint (e.g. safety analysis)
F-3	xx	x	depending on source of constraint (e.g. safety analysis)

Note: All other constraint-types can be resolved, thus are always satisfied, and don't need to be encoded. Options marked as "xx" are preferred by the authors.

Table 3 provides a proposal, in which way each constraint-type could be encoded. The proposal tries to tackle the nature of the constraint-types as well as efficiency considerations, in order to find the most appropriate encoding for each constraint-type. If a constraint-type is encoded as "desired", the following representation is proposed/advised:

$$cost_i = \frac{\text{\# of elements that violate a constraint-type}}{\text{\# of elements that have a constraint-type associated}} \rightarrow \min \tag{8}$$

This way, each cost term is scaled between 0 and 1, which makes it easier to incorporate the cost term into the *cost function*. The individual cost terms (for constraint encoding) can then be grouped into a single cost term *constraint violation*, using a scaled weighted sum, inspired by [10].

$$cost_{\text{constraint violations}} = \frac{\sum w_i \cdot cost_i}{\sum w_i} \rightarrow \min \tag{9}$$

This cost term can then be included into the cost function, wherein the optimization objectives (e.g. minimize bus utilization) are encoded. Finding adequate weights for the individual cost terms is a challenging task. It is almost impossible to find weights that perform well for all problems. Therefore weights should be assigned to problem-classes. By applying a systematic experimental approach [12], weights can be found that perform well for these problem-classes.

2.2 Implications on Design Space Exploration

Within the pre-processing phase, constraints are resolved. Therefore two methods are used:

1. grouping of tasks, forming task clusters
2. calculating a set of admissible processors for each task

In addition, a set of rules how to tackle certain constraints during the DSE were presented. These information is exploited during the DSE. As a consequence the following principles will be used during the DSE:

– Task clusters are treated as single elements during task allocation. Therefore, if a task cluster is re-allocated, all tasks inside that task cluster will be re-allocated to the same processor.
– When picking a “new” processor for a task / task cluster, only processors from the set of admissible processors are used as candidates.
– Rules for satisfying constraints during the DSE are applied to the commensurate design steps (e.g. frame packing)

As a consequence, a large number of infeasible configurations is avoided, since constraints are not violated. Thus, the efficiency of the DSE increases.

3 System Configuration – Optimization Framework

In order to actually perform the system configuration DSE, we are using a metaheuristic search algorithm called *simulated annealing (SA)*, a well known algorithm in the domain of artificial intelligence. Its name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material. The main reason for using SA is that it is shown in [8] how SA can be tailored to address system configuration upgrade scenarios. To ensure that the temporal attributes of the system meet the requirements, traditional methods are used: WCET-analysis and WCRT-analysis. In order to apply SA to a specific problem (here: system configuration), some methods have to be implemented:

– neighbour: Which modification can be applied to a system configuration, in order to get a new system configuration? These represent the modification an engineer would perform manually.
– energy (cost): How "good" is a system configuration? This represents the metrics that are used to evaluate a system configuration.

The following optimization objectives are encoded into the cost function. Again a scaled weighted sum is used.

– number of needed processors $\rightarrow$ min
– bus utilization $\rightarrow$ min
– processor CPU utilization $\rightarrow$ max & balanced
– end-to-end delay $\rightarrow$ min
– constraint violations $\rightarrow$ min

In order to get a modified system configuration, the following neighbour steps are applied: 1) A task / task cluster (whose allocation is not fixed) is randomly picked. 2) A "new" processor is determined by randomly picking out of the set of admissible processors for that task / task cluster.

When solving the underlying forward-problem for each allocation iteration, the following assumptions are made and the following methodologies are used:

– Tasks are activated by the arrival of a message. Therefore, the task scheduling problem is equivalent to finding adequate priorities for tasks. This problem can be addressed by simple, yet efficient heuristics: rate monotonic (RM), deadline monotonic (DM) or "deadline minus jitter" monotonic (D-JM) priority assignment [3].
– The bus protocol is CAN [5]. Therefore, the bus arbitration scheduling problem is equivalent to finding adequate priorities for bus frames. Thus, RM, DM or D-JM can be applied as well.
– Data that is exchanged between processors via data buses is packed into bus frames using a heuristic inspired by [13], but additionally incorporating the methods for satisfying packing constraints.
– Schedulability of the system is checked using [14].

The simplifications and assumptions that are made within this work are only introduced for the sake of simplicity. The proposed methods for satisfying the design constraints are independent of these assumptions. Thus more sophisticated methodologies (e.g. for priority assignment) can be applied, if desired/needed.

4 Experimental Results

In section 2 we have presented a set of 19 constraint-types that may be present in a system configuration problem. Later, we presented a set of rules that can be used to resolve 11 constraint-types. In addition, we have presented rules that can be used to dynamically satisfy 1 constraint-type. Assuming that each constraint-type is used equally often, we can make sure that about 63% of the constraint-types are satisfied during the DSE. Thus a high number of constraint-violations can be avoided, making the DSE more efficient.

approach	#	%
resolve	11	57.89
dyn. satisfy	1	5.26
no guarantee	7	36.84

In order to show the impact of the design constraints on DSE efficiency, we have performed several experiments. On the one hand, we tried to solve the system configuration problem by state-of-the-art approaches. On the other hand, we applied a pre-processing phase, during which constraints are resolved.

Due to the high number of different constraint-types, it is infeasible to demonstrate all combinations of constraint-types here. Therefore, let us focus on a problem instance of medium scale size

- hardware: 6 processors, 1 external data bus
- software: 30 tasks, 45 communication-links between tasks

and having the following constraints

- limited resources for all hardware elements (A-1 .. A-3)
- deadlines for all tasks and messages (B-1, B-2)
- 2 tasks already allocated (C-3)
- 4 task-groupings (D-1)
- 1 internal communication-link (E-1)

In order to reduce the uncertainties that are introduced by the used meta-heuristic (SA) which uses random numbers, several solving-runs are performed. The results of all these runs are shown in table 4. The most interesting results are highlighted in bold.

Table 4. Impact of Resolving Constraints on DSE Performance and DSE Results (min./median/max. of 10 runs per scenario)

criteria	**no pre-processing**	**with pre-processing**
iterations	10000	10000
unique allocations	9546 / 9588.5 / 9595	9423 / 9430.5 / 9469
feasible allocations	**0 / 1 / 1**	**2026 / 2169 / 2231**
infeasible, due to constr. D-1 & E-1	**9541 / 9578.5 / 9589**	**0 / 0 / 0**
infeasible, due to CPU overload	0 / 1.5 / 8	4050 / 4177 / 4335
infeasible, due to memory overload	1 / 1.5 / 7	2653 / 2693 / 2759
infeasible, due to deadline violation	0 / 2.5 / 6	385 / 406.5 / 421
used processors	5 / 6 / 6 (of 6)	**4 / 4 / 4** (of 6)
bus utilization [%]	12.66 / 13.73 / 14.61	7.52 / 8.23 / 9.87
CPU utilization [%] (average)	47.69 / 47.69 / 57.23	71.54 / 71.54 / 71.54
Δ CPU utilization [%] (average)	7.69 / 10.77 / 20.25	3.85 / 5.19 / 8.08

The results impressively show the negative effect that the set of design constraints has on the DSE performance, if state-of-the-art solving approaches are used (see: *no pre-processing*). Almost no feasible configuration can be found. The main reason is that some constraints are violated. This is even more impressive if we consider that only a small number of constraints is used.

On the contrary, the results clearly evidence the positive effect of the pre-processing phase on the DSE performance (see: *with pre-processing*). If the pre-processing rules are applied, no more constraint violations stem from these constraint-types (here: C-3, D-1, E-1). As a consequence, more feasible configurations are generated and evaluated during the DSE. The median feasibility ratio is 0.217 (= feasible vs. all iterations). Overall, the DSE efficiency is significantly improved. In addition, the best obtained configuration is significantly improved, if pre-processing is applied: Fewer processors are needed, a lower bus utilization can be achieved, and processor utilization is more balanced.

However, it seems that pre-processing has a negative impact on resource utilization and real-time behaviour. Though, this is not the case, and can be explained: In these experiments we applied a hierarchical evaluation-schema. Constraints are checked in the following sequence: allocation, dependency, routing, packing, resource, timing. Thus, if allocation constraints are violated, other constraints (e.g. resource utilization or timing) are not checked/counted any more, since the configuration is already infeasible. This helps to speed up the entire evaluation process (since time-consuming schedulability-tests are performed only if all other constraints are satisfied). However, as a consequence resource overload and deadline violations are only counted if all the other constraints are satisfied. To get a more general comparison, the sum of all infeasible configurations can be taken as a metric.

Generality. In order to evaluate the generality of the proposed approach, several experiments have been performed. Therein, we varied both the size and structure of the problem instances, as well as the imposed design constraints. The problem sizes vary within the following ranges:

- 30..90 tasks / 45..135 messages
- 6..12 processors / system utilization: 0.331..0.493 per processor, if all processors would be used. Typically, a good configuration will not use all processors.

Besides limited resources and real-time behaviour, the following constraint-types are imposed on the systems:

1. scenario-type I: fault-tolerant system
   - excluded processors, for 15..25% of tasks (C-2)
   - task-separation, for 10..20% of tasks (D-2)
   - internal messages, for 5..10% of messages (E-1)
   - separated frames, for 10..15% of messages (F-3)
2. scenario-type II: system upgrade
   - fixed allocations, for 20..30% of tasks (C-3)
   - dedicated processors, for 10..20% of tasks (C-1)

– task-grouping, for 10..15% of tasks (D-1)
– dedicated frame, for 10..15% of messages (F-1)

10 examples of each category have been generated. For reason of space the full results cannot be presented. However the trends are similar to those in table 4. In summary: Without pre-processing almost no feasible configurations can be found, because of constraint violations. With pre-processing a significant larger number of feasible configurations can be found. The median feasibility ratio is 0.240, its min is 0.229, and its max is 0.547. So in general, the proposed method provides quite reproducible performance and results. This indicates the robustness of the method.

An even more general view can be derived from the results: All constraint-types can be categorized into 3 classes: (I) Those that can be resolved, (II) those that can be dynamically satisfied, and (III) those where no guarantee can be given. By applying the rules we have presented (I) and (II) will be satisfied. Thus only (III) are left. This means that any system configuration problem which contains the constraints presented in table 1 can be transformed into a problem where constraints of type (I) and (II) are no longer present. This transformation is performed by applying the presented resolving methods. Our experiments suggest that it is a robust method.

5 Conclusion

During the design of reliable software-based systems, a set of heterogeneous design constraints must be satisfied. These constraints stem from different sources. Most safety-relevant systems must guarantee real-time behaviour during operation, thus no deadline must be missed. Reliability is often enforced by applying replication. Consequently, replicated elements must be independent of each other. E.g. replicated tasks and messages must be assigned to separated resources. Manual configuration of such complex systems is time consuming and error-prone. Thus methods for automated system configuration are needed.

We have presented methods, how each of the relevant constraint-types can be addressed and satisfied. Several constraints can be resolved before the DSE, some can be dynamically satisfied, and for some no guarantee can be given. Experimental results evidence that by applying a pre-processing phase, during which constraints are resolved, significantly improves DSE performance. In addition, the quality of the best obtained configuration is improved.

In future research, the proposed methods will be incorporated into the DSE for *system configuration upgrade* (such as [8]) in order to make this DSE more efficient. Also, there are some indications that the presented approach can also be applied to (distributed) multi-core systems.

Acknowledgment. The authors would like to acknowledge the financial support of the "COMET K2 - Competence Centres for Excellent Technologies Programme" of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG). We also thank our supporting industrial (AVL List) and scientific (Graz University of Technology) project partners.

References

1. AUTOSAR (automotive open system architecture), `http://www.autosar.org`
2. ISO 26262: Road vehicles – functional safety
3. Audsley, N., Burns, A., Richardson, M.F., Wellings, A.J.: Hard real-time scheduling: The deadline-monotonic approach. In: IEEE Workshop on Real-Time Operating Systems and Software, pp. 133–137 (1991)
4. Chu, W.W., Holloway, L.J., Min-Tsung, L., Efe, K.: Task allocation in distributed data processing. Computer 13(11), 57–69 (1980)
5. Davis, R., Burns, A., Bril, R., Lukkien, J.: Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. Real-Time Systems 35(3), 239–272 (2007)
6. Efe, K.: Heuristic models of task assignment scheduling in distributed systems. Computer 15(6), 50–56 (1982)
7. Emberson, P., Bate, I.: Extending a task allocation algorithm for graceful degradation of real-time distributed embedded systems. In: IEEE Real-Time Systems Symposium (RTSS), pp. 270–279 (2008)
8. Emberson, P., Bate, I.: Stressing search with scenarios for flexible solutions to real-time task allocation problems. IEEE Transactions on Software Engineering 36(5), 704–718 (2010)
9. Hamann, A., Racu, R., Ernst, R.: Multi-dimensional robustness optimization in heterogeneous distributed embedded systems. In: IEEE Real Time and Embedded Technology and Applications Symposium (RTAS), pp. 269–280 (2007)
10. Pölzlbauer, F., Brenner, E., Magele, C.: A transparent target function and evaluation strategy for complex multi-objective optimization problems. In: IEEE Real-Time Systems Symposium (RTSS) – Work-in-Progress, pp. 77–80 (2009)
11. Pop, P., Eles, P., Peng, Z., Pop, T.: Analysis and Optimization of Distributed Real-Time Embedded Systems. ACM Transactions on Design Automation of Electronic Systems 11(3), 593–625 (2006)
12. Poulding, S., Emberson, P., Bate, I., Clark, J.: An efficient experimental methodology for configuring search-based design algorithms. In: IEEE High Assurance Systems Engineering Symposium (HASE), pp. 53–62 (2007)
13. Sandström, K., Norström, C., Ahlmark, M.: Frame packing in real-time communication. In: International Conference on Real-Time Computing Systems and Applications (RTCSA), pp. 399–403 (2000)
14. Tindell, K., Clark, J.: Holistic schedulability analysis for distributed hard real-time systems. Microprocessing and Microprogramming – Parallel Processing in Embedded Real-Time Systems 40(2-3) (1994)
15. Zheng, W., Zhu, Q., Di Natale, M., Sangiovanni-Vincentelli, A.: Definition of task allocation and priority assignment in hard real-time distributed systems. In: IEEE International Real-Time Systems Symposium (RTSS), pp. 161–170 (2007)
16. Zhu, Q., Yang, Y., Scholte, E., Di Natale, M., Sangiovanni-Vincentelli, A.: Optimizing extensibility in hard real-time distributed systems. In: IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 275–284 (2009)