# Software Deployment for Distributed Embedded Real-Time Systems of Automotive Applications

Florian Pölzlbauer, Iain Bate, Eugen Brenner

**Abstract** Automotive applications can be described as distributed embedded software which perform real-time computation on top of a heterogeneous hardware platform. One key phase in designing distributed software systems is *software deployment*. Therein it is decided how software components are deployed over the hardware platform, and how the communication between software components is performed. These decisions significantly determine the system performance. This chapter tackles the software deployment problem, tailored to the needs of the automotive domain. Thereby, the focus is on two issues: the *configuration of the communication infrastructure* and how to *handle design constraints*. It is shown, how state-of-the-art approaches have to be extended in order to tackle these issues, and how the overall process can be performed efficiently, by utilizing search methodologies.

## 1 Introduction

In the past, automotive electronics and avionics systems were designed in a federated manner. Most functionality was implemented by special-purpose hardware and hardware-tailored software. One control unit performed only one or at most a limited number of individual functions, and functions had their own dedicated hard-

Florian Pölzlbauer
Virtual Vehicle, Graz, Austria
e-mail: florian.poelzlbauer@v2c2.at

Iain Bate
University of York, Department of Computer Science, York, United Kingdom
e-mail: iain.bate@cs.york.ac.uk

Eugen Brenner
Graz University of Technology, Institute for Technical Informatics, Graz, Austria
e-mail: brenner@tugraz.at

ware. 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. The design of electronics has moved from a hardware-oriented to a software/function-oriented approach. This means that functionality is mainly based on software which is executed on general-purpose hardware. In order to enable this trend an interface layer (AUTOSAR [2]) was introduced which separates the application software from the underlying hardware. At the same time, software development steadily moves from hand-coded to model-driven. In model driven development, system synthesis is an important design step to give a partitioning/allocation. The synthesis transforms the Platform Independent Model (PIM) of the system, held in views such as UML's class and sequence diagram, into a Platform Specific Model (PSM), held in views such as UML's deployment diagrams. Design-languages which support model-driven development (such as UML, EAST-ADL, MARTE, etc.) provide dedicated diagrams (e.g.: component, deployment, communication, timing).

In order to deploy the application software onto the execution platform, several configuration steps need to be performed. In the literature this is often referred to as the Task Allocation Problem (TAP). TAP is one of the classically studied problems in systems and software development. It basically involves two parts. Firstly allocating tasks and messages to the resources, i.e. the processors and networks respectively. Secondly assigning attributes to the tasks and messages. Tasks represent software component, and are described by their timing and resource demand (e.g. memory). Messages represent communication between tasks, and are described by their data size and timing. Processors represent the computational units which execute tasks, and are described by their processing power and memory. Networks enable cross-processor communication, and are described by their bandwidth and protocol-specific attributes. In its simplistic form it is an example of the Bin Packing Problem (BPP) where the challenge is to avoid any resource becoming overloading [11]. This "standard" version of the problem is recognised as being NP-hard. Solutions normally involve three components: a means of describing the problem, a fitness function that indicates how close a solution is to solving the problem, and a search algorithm. A wide range of techniques have been used for searching for solutions with heuristic search algorithms, branch and bound, and Integer Linear Programming being the main ones. The problem was later expanded to cover:

1. hard real-time systems where schedulability analysis is used to ensure that the system's timing requirements are met as failing to meet them could lead to a catastrophic effect [3]
2. reducing the energy used [1]
3. making them extensible [19], i.e. so that task's execution times can be increased while maintaining schedulability
4. handling change by reducing the likelihood of change and the size of the changes when this is no longer possible [8]
5. supporting mode changes with the number of transitions minimized [6] and fault tolerance [7]

### Open Issues of State-of-the-Art

The list above represents an impressive subset of the problems that need to be solved, however it still does not meet all needs of modern hard real-time systems. There are (at least) two important problems not covered.

- Firstly there are often constraints over the solution, e.g. replicas of tasks cannot reside on the same processor, or that tasks should reside on the same processor near a certain physical device.
- Secondly, communication demand of applications is steadily increasing. Thus, due to limited bandwidth, bus systems are becoming a bottleneck. State-of-the-art bus configuration approaches do not tackle this problem, since they use the bandwidth in an inefficient way. This is due to the fact that frames are hardly utilized, and thus too much overhead data is generated. This not only leads to poor bandwidth usage, but may also lead to unschedulable bus systems. The Frame Packing Problem (FPP) deals with this issue, by packing several messages into a single bus frame, thus improving bandwidth usage and reducing the likelihood of an unschedulable solution when in fact a schedulable one is feasible. In the coming years the communications bus is likely to become a greater bottleneck.

Both these problems have been studied to a limited extent, however this chapter shows how they can be solved more effectively and efficiently (including the scalability to larger more complex systems) using automotive systems as an example.

### Outline

The structure of this chapter is as follows. The software deployment problem is outlined in section 2 starting with an explanation of the standard TAP before explaining the needs of hard real-time systems using the domain of automotive systems as an example. Next, a solution to the FPP is presented and demonstrated compared to the previous state-of-the-art approaches. Then, the standard TAP (including the FPP) is extended to deal with the constraints from the automotive industry, before finally directions for future work are outlined.

## 2 The Software Deployment Problem

The purpose of this section is to explain the **standard TAP**, and outline how it might be solved. The standard TAP refers to the allocation of tasks and messages to processors and networks respectively. Originally it did not consider constraints over and dependencies between tasks.

## 2.1 The Standard Problem

Assuming a given software application (consisting of several communicating tasks) and a given execution platform (consisting of several processors, connected via networks). Software deployment (or task allocation) deals with the question, how the software application should be allocated onto the execution platform. Thereby, objectives need to be optimized and constraints need to be satisfied. Applied to hard real-time systems, the process consists of the following design decisions:

- task allocation: local allocation of tasks onto processors
- message routing: routing data from source processor to destination processor via bus-systems and gateway-nodes
- frame packing: packing of application messages into bus frames
- scheduling: temporal planning of the system execution (task execution on processors, frame transmission on bus-systems)

These steps are followed by system performance and timing analysis in order to guarantee real-time behaviour. Due to the different design decisions involved, the terms *software deployment* or *task allocation* seem inappropriate to describe the overall process. The term **system configuration** seems more adequate. This is why it will be used throughout this chapter, from this point on.

## 2.2 Solving the Standard Problem

For several years, system configurations have be designed manually by engineers. To a large degree, this approach is still performed today. Due to the steadily increasing system complexity, the manual approach reaches several limitations:

- It is hard for engineers to keep in mind all direct and indirect interactions (e.g. direct precedence relationships between tasks or indirect relationships such as preemption and blocking) within the system. Design mistakes may occur.
- Generating a system configuration is time consuming. Due to time constraints, only a small number of system configurations are generated and evaluated. Thus it is unlikely that the generated system configuration is close to optimal.

In order to overcome these issues, **design automation** can be applied. Therein, design decisions are performed by algorithms. Thanks to high computation power available, a very high number of system configurations can be generated and evaluated within a reasonable time frame (e.g. several hundred thousand configuration can be evaluated within several hours). This automated approach significantly increases the coverage of the design space, and thus increases the probability of finding near-optimal solutions. Consequently it increases the confidence into the solution. However, due to the enormous design space, complete coverage is impossible. Experience has shown that approaches have to be efficient, in order to be applicable to the typical size and complexity of real world systems [18, 19].

**Algorithm 1:** System Configuration Optimization algorithm (based on Simulated Annealing search algorithm)

```
   Input: config.init /* initial configuration */
   Data: t /* temperature */
   Data: iter.max /* max. iterations */
   Data: iter.at.t.max /* max. iterations at constant t */
 1 begin SystemConfigurationSimulatedAnnealing
 2     /* initialize */;
 3     cost.init = cost(config.init);
 4     config.current = config.init /* start at initial configuration */;
 5     cost.current = cost.init;
 6     config.best = config.init;
 7     cost.best = cost.init;
 8     /* search, until stopping-criteria is reached */;
 9     while stop() = false do
10         while iter.at.t < iter.at.t.max do
11             /* propose new configuration */;
12             config.new = neighbour(config.current);
13             cost.new = cost(config.new);
14             /* accept move? */;
15             if acceptMove() = true then
16                 /* improvement of best configuration? */;
17                 if cost.new < cost.best then
18                     /* remember best */;
19                     config.best = config.new;
20                     cost.best = cost.new;
21                 end
22             end
23             iter.at.t++;
24             /* next iteration at constant t */;
25         end
26         cool(t);
27         iter.at.t = 0;
28         /* resume search at lower t */;
29     end
30 end
   Output: config.best /* best configuration found */
```

In order to perform system configuration Design Space Exploration (DSE), the meta-heuristic search algorithm *Simulated Annealing (SA)* can be utilized. It is 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. It has proven to be very robust, and can be tailored to specific problems with low effort. However, the main reason for using SA is that it is shown in [5] how SA can be tailored to address *system configuration upgrade* scenarios. This aspect will be re-visited in section 7.

In order to apply SA to a specific problem (here: system configuration), the following 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.

Algorithm 1 shows the overall procedure. The search starts from an initial configuration, which is either randomly generated or given by the user. By applying the neighbour function, a new configuration is generated. By analysing its cost, the SA determines whether or not to accept it. After a certain number of iterations, the value of parameter $t$ (which represents the temperature in the annealing process) is decreased, which impacts the acceptance rate of *worse* configurations. The overall search stops, due to a defined exit criteria (usually a defined number of iterations or a defined cost-limit).

The following optimization objectives are encoded into the cost function.

- number of needed processors $\rightarrow$ min
- bus utilization $\rightarrow$ min
- processor CPU utilization $\rightarrow$ max & balanced

The individual terms are combined into a single value, using a scaled weighted sum. Determining adequate weights is challenging, and should be done systematically [15].

$$cost = \frac{\sum w_i \cdot cost_i}{\sum w_i} \tag{1}$$

In order to get a modified system configuration, different neighbour moves can be performed. Most commonly used are:

- re-allocate a task to another processor
- swap the allocation of two tasks (which reside on different processors)
- change scheduling attributes: For priority-based scheduling, changing the priority of a task. For time-triggered scheduling, change the time-slot assignment.

To ensure that the temporal attributes of the system meet the requirements, two analyses need to be performed: Before starting the DSE, worst-case execution time (WCET) analysis must be performed for each task. During the DSE, schedulability analysis must be performed. Therefore, worst-case response time (WCRT) analysis can be applied.

$$r_i = J_i + B_i + C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i + J_j}{T_j} \right\rceil C_j \tag{2}$$

## 3 The Needs of the Automotive Industry

Electronics and embedded software was introduced into automotive systems in 1958. The first application was the electronically controlled ignition system for com-

bustion engines. It was introduced in order to meet exhaust emission limits. Over the years, a wide range of functionality of a car was implemented by electronics and embedded software. Thereby, the embedded real-time software was developed in a federated manner. Software was tailored to special-purpose hardware, and almost each software-component was executed on a dedicated hardware-unit. As a consequence, the software-components were quite hardware-dependent, which made it hard to migrate software-components to new hardware-units. Also, the number of hardware-units dramatically increased, which increased system cost and system weight.

To overcome these issues, a paradigm shift has occurred during the last several years. Software is executed on general-purpose hardware. Processing power of this hardware steadily increases, thus allowing to execute several software-components on a single hardware-unit. In order to make it easier to migrate software-components to new hardware-units, software-components are separated from the underlying hardware. This is enabled by the introduction of an interface layer, which abstracts hardware-specific details, and provides standardized interfaces to the application-layer software-components. In the automotive domain, the AUTOSAR standard [2] has positioned itself as the leading interface-standard. The main part of AUTOSAR with respect to TAP is the Virtual Function Bus (VFB). This allows two software-components to communicate with each other without explicit naming and location awareness. Basically the source would send a system-signal to the destination via a VFB call. The actual details of where and how to send the signal is decided by the Runtime Environment (RTE), which instantiates the VFB calls on each processor. Taking this approach restricts changes to the RTE when a new task allocation is generated. In other domains, similar trends can be observed. In the avionics domain, the Integrated Modular Avionics (IMA) standard implements a very similar concept referred to as Virtual Channels (instead of VFB calls) and System Blueprints (containing the lookup tables).

## *Challenges*

In order to solve the automotive system configuration problem, the general system configuration approaches need to be tailored to automotive-specific demands. Thereby, the following issues are the most challenging ones. To a large degree, these issues are not sufficiently covered by the literature.

- The configuration of the communication infrastructure needs to tackle details of automotive bus protocols. Thereby, a special focus needs to be set on frame packing.
- Automotive system configuration is subject to a set of heterogeneous design constraints. Only a subset of them is covered in the literature. Efficient methods for ensuring constraint satisfaction are needed.
- Automotive systems are rarely designed “from scratch”. Instead, existing systems (or parts of thereof) are taken as an initial solution. These are extended in

order to meet current requirements. In addition, system components may be used within several system variants. Consequently, several legacy design decisions may be in place, which must be taken into account during system configuration. This imposes additional constraints.
- The search algorithms need to scale to realistic sizes and complexities of systems. A significance influence on scalability is how hard for the system is it to synthesise a schedulable solution. Broadly speaking the closer the resources are to their utilisation limits the more difficult the challenge.

## *3.1 Frame Packing*

A large number of automotive systems perform control-related tasks. Physical data is sensed by sensors, read by IOs and processed by application software-components. The output data is then fed back into the physical world via IOs and actuators. Due to cost constraints, the accuracy of sensors, IOs and actuators is limited to 8 to 12 bits. As a consequence, most data is encoded by 1 to 16 bits variables.

Most automotive bus-systems transmit bus frames which can contain up to 8 byte payload data (LIN, CAN). Emerging bus-systems like FlexRay and automotive Ethernet can transmit more payload data (e.g. 254 byte for FlexRay). However, in addition to the payload data, protocol-related data (header, checksum, ...) must be transmitted. For LIN and CAN these protocol data consume 64 bits per frame. Based on the packing density of a frame, the bandwidth efficiency is between 1,5% (1 : 1+64) and 50% (64 : 64+64). Over the years, the communication demand of automotive systems has steadily increased. However, the available bus bandwidth is a limited resource. Due to physical constraints, the bandwidth of a shared bus cannot easily be increased. As a consequence, bandwidth-efficient frame packing is needed. This is why frame packing is heavily used in the automotive domain by engineers. Typically, frames contain between 48 and 64 bits payload. However, almost no work in the literature on system configuration considers this issue. Instead, almost all works consider only the following options:

- If a message fits into a frame (message.size $\leq$ frame.payload.max), each message is packed into a separate frame.
- If a message does not fit into a frame (message.size $>$ frame.payload.max), the message is split into several parts, each part is packed into a separate frame, and the parts are re-joined at the receiving processor.

These simplistic assumptions lead to several negative attributes of the communication configuration: First, it is a non-realistic approach, which is not accepted by industry. Second, the approach leads to a high number of frames. The frame-set has poor bandwidth usage, since too much bandwidth is consumed by the protocol-overhead data. The high number of frames also leads to increased interference between frames, thus leading to increased response times (and decreased schedulability). In order to overcome these issues, realistic frame packing must be performed by

system configuration approaches. Therefore, several messages must be packed into a single bus frame. Packing objective should be to minimise the bandwidth demand of the resulting frame-set.

The Frame Packing Problem (FPP) is defined as follows: A set of messages $M = \{m_1, m_2, \ldots, m_n\}$ must be packed into a set of bus frames $F = \{f_1, f_2, \ldots, f_k\}$, subject to the constraint that the set of messages in any frame fits that frame's maximum payload. Usually, the FPP is stated as an optimization problem. The most common optimization objectives are: 1) minimize the number of needed frames; or 2) maximize the schedulability of the resulting frame-set. A message is defined by $m_i = [s_i, T_i, D_i]$. A frame is define by $f_j = [pm_j, M_j, T_j, D_j]$. In general each frame may have its individual max. payload (depending on the bus protocol). However, usually all frames on the same bus have the same max. payload.

## *3.2 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 [10]), 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.
- legal requirements: Certain design solutions may not be allowed, in order to comply to legal regulations.

Within the AUTOSAR standard, design constraints that 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.

Since all embedded software must content itself with limited resources, these constraints are well studied in the literature. Automotive systems must be reliable, and thus have to satisfy additional constraints. Most safety-related functions must guarantee real-time behaviour, especially if human life is at risk (e.g. drive-by-wire application in a car). If the function is high-critical, it may be needed to apply redundancy. Therefore replicated tasks must not reside on the same processor (task separation), certain processors are inadequate for handling certain tasks (excluded

**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

processors), and data must be transferred via separated buses, probably even within separated bus frames.

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 automotive systems though, these constraints are of high importance.

## 4 Solving the Frame Packing Problem

The FPP can be seen as a special case of the Bin Packing Problem (BPP), which is known to be a NP-hard optimization problem. In the literature there are several heuristics for the BPP [4]. Well known on-line heuristics are: next fit, first fit, best fit, etc. Off-line heuristics extend these approaches by applying initial sorting, resulting in: next fit decreasing, best fit decreasing, etc. In general, off-line approaches outperform on-line approaches, since off-line approaches can exploit global knowledge, whereas on-line approaches have to take decisions step-by-step, and decisions cannot be undone.

Inspired by the main concepts of BPP heuristics, heuristics for the FPP have been developed. It is interesting to note that there are only a few works in the literature addressing the FPP, although the FPP has significant impact on the performance of the system. Most FPP algorithms mimic some BPP heuristic. [17] mimics *next fit decreasing*, where messages are sorted by their deadline. [16] mimics *best fit decreasing*, where messages are sorted by their periods. In addition, the sorted message-list is processed alternately from the beginning and the end. In [14] messages are sorted by their offsets. [16, 14] combine the FPP with the scheduling problem. [18, 19] include the FPP into the TAP. Thereby FPP and TAP are formulated as a Mixed Integer Linear Problem (MILP), and solved sequentially.

**Table 2** Symbols used for Frame Packing

| Symbol | Description |
|---|---|
| $m$ | message |
| $f$ | frame |
| $M$ | set of messages |
| $F$ | set of frames |
| $D$ | deadline |
| $T$ | period |
| $T_m$ | period of message |
| $T_f$ | period of frame |
| $s_m$ | data size of message |
| $pay_f$ | payload of frame |
| $pm$ | max. payload of frame |
| $oh_f$ | overhead of frame |
| $br$ | baudrate of bus |
| $bw$ | bandwidth demand |

## *4.1 Insufficiencies of state-of-the-art Approaches*

Besides these differences, all state-of-the-art FPP algorithms share one common issue: The packing decision is made based on one condition only:

$$\text{message.size} \leq \text{frame.payload.left} \tag{3}$$

Due to limited bus bandwidth, frame packing should be bandwidth demand minimizing. The bandwidth demand of a frame is determined by two factors: data (payload and overhead) and period.

$$bw_f = \frac{pay_f + oh_f}{T_f} \tag{4}$$

The payload contains all packed-in messages. The overhead contains all protocol-specific data. Since a frame may have several messages packed-in, the frame must be transmitted at a rate which satisfies the rate of all packed-in messages. Thus the lowest message period determines the frame period.

$$T_f = \min_{i=m \in f} \{T_i\} \tag{5}$$

In order to achieve minimal bandwidth demand, the following aspects must be tackled:

1. The number of frames must be minimized, in order to minimize the overhead data.
2. Messages which are packed into the same frame should have *similar* periods, in order to avoid sending a message more frequently than needed.

Some state-of-the-art FPP approaches try to tackle these.

1. Messages are packed into frames, for as long as there is space left. This reduces the number of needed frames, and thus the bandwidth consumption by the overhead data.
2. Messages may be sorted by their period, before performing the packing. This way, the period variation is reduced. Thus, messages with *similar* periods are packed into the same frame.

Although both strategies (dense packing and initial sorting) may help reducing bandwidth demand, they cannot guarantee minimal bandwidth demand. Here is the **flaw**: During the packing procedure, only the necessary condition (see equation 3) is considered. Instead, each packing step must be subject to optimality considerations.

## *4.2 Optimality Criteria for Frame Packing*

Sophisticated frame packing should be bandwidth demand minimizing. The optimal/ideal situation is to have fully utilized frames and all messages inside a frame having the same (or very similar) periods. In general (since both the data size as well as the period of messages varies) the real situation represents a trade-off: If messages with different periods are packed into a frame, the frame must be sent at the lowest period, and thus some of the messages are sent more frequently than needed. This increases the bandwidth consumption. On the other hand, the more messages are packed into a frame, the less frames are needed. Thus less overhead data is sent. This reduces the bandwidth consumption. The optimal trade-off can be accomplished as follows:

Assume the following minimal example: There exists a frame that already has some messages packed-in. Another message needs to be packed-in and it can fit the existing frame. The question is: Should the message be packed into the existing frame (thus extending it), or should the message be packed into a new frame? This

decision can be taken in an optimized way, by analysing the bandwidth demand of the two alternatives (left and right side of equation):

$$\underbrace{\frac{pay_f + s_m + oh_f}{T_f^{'}}}_{\text{extended frame}} = \underbrace{\frac{pay_f + oh_f}{T_f}}_{\text{existing frame}} + \underbrace{\frac{s_m + oh_f}{T_m}}_{\text{new frame}} \tag{6}$$

Note that the period of a frame is determined by the message with the lowest period inside the frame. By adding a message, the period of the extended frame $T_f^{'}$ may change. Originally it is $T_f$.

$$T_f^{'} = \min\left\{T_f \cup T_m\right\} \tag{7}$$

Depending on the relation between $T_m$ and $T_f$, there are 3 cases for this packing situation. For each of them, an optimal decision can be made.

**Case I:** $T_m = T_f$

If the periods of the frame and the message are equal, it is always beneficial to extend an existing frame. Creating a new frame is never beneficial, because of the additional overhead data.

$$\frac{pay_f + s_m + oh_f}{T} = \frac{pay_f + oh_f}{T} + \frac{s_m + oh_f}{T} \tag{8}$$

$$oh_f < 2\ oh_f \tag{9}$$

**Case II:** $T_m > T_f \Rightarrow T_f^{'} = T_f$

The trade-off is: By extending the frame, the message will be sent more frequent than needed, but no additional overhead is created. By creating a new frame, additional overhead is created, but the message will not be sent too frequent.

$$\frac{pay_f + s_m + oh_f}{T_f} = \frac{pay_f + oh_f}{T_f} + \frac{s_m + oh_f}{T_m} \tag{10}$$

$$\frac{s_m}{T_f} = \frac{s_m + oh_f}{T_m} \tag{11}$$

At the threshold period of the message, the two alternatives perform equally.

$$T_m^* = T_f \frac{s_m + oh_f}{s_m} \tag{12}$$

Thus, the optimal solution is:

- $T_m < T_m^* \Rightarrow$ extending the frame is beneficial

- $T_m > T_m^* \Rightarrow$ creating a new frame is beneficial

**Case III:** $T_m < T_f \Rightarrow T_f^{'} = T_m$

The trade-off is: By extending the frame, the frame will need to be sent more frequent, but no additional overhead is created. By creating a new frame, the original frame will not be sent more frequent, but additional overhead is created.

$$\frac{pay_f + s_m + oh_f}{T_m} = \frac{pay_f + oh_f}{T_f} + \frac{s_m + oh_f}{T_m} \tag{13}$$

$$\frac{pay_f}{T_m} = \frac{pay_f + oh_f}{T_f} \tag{14}$$

The threshold period is:

$$T_m^* = T_f \frac{pay_f}{pay_f + oh_f} \tag{15}$$

Thus, the optimal solution is:

- $T_m < T_m^* \Rightarrow$ creating a new frame is beneficial
- $T_m > T_m^* \Rightarrow$ extending the frame is beneficial

## *4.3 Improved Frame Packing Heuristic*

The main issue of state-of-the-art frame packing heuristics is: During packing only the necessary packing condition (see equation 3) is checked. In case the periods of messages vary significantly, the approaches perform poorly, even if messages are sorted by their periods.

To overcome this issue, the packing decision must be taken by also incorporating the trade-off optimality criteria, derived above. The proposed frame packing heuristic (see algorithm 2) incorporates these criteria. Its structure is inspired by the *Fixed Frame Size* approach of [17] which mimics *next fit decreasing*. However, messages are not sorted by their deadline. Instead messages are sorted by their period, inspired by [16]. However, the packing procedure is not done in a bi-directional way.

Within the *ExtendOrNew* method, the most beneficial decision is determined using the optimality criteria presented earlier. This way each packing step has minimal increase of bandwidth demand.

Due to the NP-hard nature of the FPP, the proposed approach cannot guarantee an optimal packing. However, experimental evaluation shows that it outperforms state-of-the-art approaches. On the one hand, the bandwidth demand of the resulting frame-set is significantly decreased. On the other hand, the schedulability of the frame-set is less sensitive against timing uncertainties.

**Algorithm 2:** Frame packing (based on optimal decisions)

```
   Input: messages
 1 sort(messages, T, increasing) /* sort by T [0..n] */;
 2 frame = new frame;
 3 foreach message do
 4 |   if frame.payload.left ≥ message.size then
 5 |   |   /* take most beneficial decision */;
 6 |   |   benefit = extendOrNew(message, frame);
 7 |   |   if benefit = extend then
 8 |   |   |   pack(message, frame);
 9 |   |   else if benefit = new then
10 |   |   |   pack(message, new frame);
11 |   |   end
12 |   else
13 |   |   pack(message, new frame);
14 |   end
15 end
   Output: frames
```

**Table 3** Improvement of Poelzlbauer et al. compared to state-of-the-art

| | improvement [%] | | | message size | improvement |
|---|---|---|---|---|---|
| # nodes | 125 k | 256 k | 500 k | [bit] | [%] |
| 1..3 | 0.0 | 0.0 | 0.0 | 1 .. 8 | 0 .. 18 |
| 5 | 5.9 | 2.3 | 0.0 | 1 .. 16 | 0 .. 18 |
| 10 | 14.4 | 6.2 | 3.3 | 1 .. 24 | 0 .. 19 |
| 15 | 13.8 | 15.0 | 2.4 | 1 .. 32 | 0 .. 16 |
| 20 | 17.6 | 16.2 | 6.2 | 1 .. 64 | 0 .. 6 |

Table 3 shows the improvements in bandwidth demand. On the left side, improvements are shown due to *number of sending processors* and *bus baudrate*. The main improvements can be seen for systems with higher number of sending processors. Such systems will be used in future automotive applications. On the right side, improvements are shown due to *message size*. An interesting finding is that the improvements are almost the same for a wide range of message sizes. Currently, physical data is mainly encoded in up to 16 bit variables. Future applications may need higher accuracy, thus 32 bit variables may be used. The proposed approach also handles these systems in an efficient way. More details on the evaluation can be found in [13].

## 5 Handling Constraints

The task of finding a system configuration is challenging and time-consuming. By applying design automation, and thus using search algorithms, a large number of potential configurations can be evaluated within reasonable time. However, finding a system configuration for industrial applications is subject to a set of heterogeneous

constraints. Table 1 gives an overview. Thus the question arises: How can these constraints be handled and satisfied?

In order to determine, how many of the constraints are satisfied (and how many are violated) by a configuration, the *cost function* of the SA search framework needs to be extended. Therefore, the cost term *constraint violations* is added. It is stated as a minimization term. Configurations which violate constraints are punished. Ideally no constraints are violated. Due to the heterogeneous nature of the constraints, different constraint-types may be of different importance. This can be addressed by applying a weighted sums approach. Each constraint-type is evaluated as:

$$cost_i = \frac{\text{\# of elements that violate a constraint-type}}{\text{\# of elements that have a constraint-type associated}} \tag{16}$$

Basically, this extended SA search framework should be able to find system configurations which satisfy all constraints. Since constraint violations are punished, the search should be directed towards regions of the design space where all constraints are satisfied. However, experimental evaluation reveals some more diverse findings. For some constraint-types, this approach works. Configurations are modified, until the number of constraint violations becomes quite low. The approach works quite well for *E-1 processor-internal only*. This can be explained by the fact, that this constraint-type is in alignment with another optimization target (bus utilization $\rightarrow$ min). For some other constraint-types (e.g. *C-1 dedicated processor*) the approach is able to reduce the number of constraint violations, but cannot eliminate them. In addition, it takes a lot of search iterations, until the number of constraint violations drops. For other constraint-types (e.g. *F-2 same frame*) the approach fails entirely.

Concluding, this approach is inefficient and ineffective, and thus is not applicable to industrial system configuration instances. The question arises: How can the set of heterogeneous constraints be handled and satisfied in an efficient way?

## *5.1 Improving Efficiency*

In order to overcome these issues, and to handle constraints in an efficient way, two issues must be addresses:

1. neighbour: The neighbour-moves are not aware of the constraints. Thus infeasible configuration may be generated. However, neighbour-moves should be aware of the constraints, and only generate configuration within feasible boundaries.
2. pre-conditions: In order to be able to satisfy a constraint, a set of pre-conditions may need to be fulfilled (e.g. certain packing constraints need certain routing conditions). Thus it is highly important to fulfill these pre-conditions, else constraints can never be satisfied.

It is advised to split the entire system configuration problem into several sub-problems, and to handle design decisions and constraints within these sub-problems. Considering the various interactions, the following sub-problems seem appropriate:

- task allocation & message routing
- frame packing
- scheduling

### 5.1.1 Task Allocation & Message Routing

In order to increase the efficiency of the search, the neighbour-moves for task allocation are modified as follows: Each task has a set of *admissible processors* associated. Only processors out of these sets are candidates for allocation modifications.

The question is: How can the sets of admissible processors be derived, so that all allocation- and routing-constraints are satisfied? To achieve this, a set of rules are derived and applied. Most of these rules are applied before the search-run. Thus it is a one time effort.

E-1: By grouping the sender- and the receiver-task (forming a task-cluster), it can be ensured that the task allocation algorithm will allocate both tasks to the same processor. Thus, the communication between these tasks is always performed processor-internal.

E-2 & E-3: Based on these sets, a set of admissible buses can be calculated for each message.

$$B_{adm} = \begin{cases} B \setminus B_{ex} & \text{if} \quad B_{ded} = \{\} \\ B_{ded} \setminus B_{ex} & \text{otherwise} \end{cases} \tag{17}$$

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.

$$P_{adm}^{(t \to m \to t)} = P \text{ connected to } B_{adm} \tag{18}$$

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

$$X = \bigcap P_{adm}^{(t \to m \to t)} \tag{19}$$

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.

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.

$$P_{adm} = \begin{cases} (P \cap X) \setminus P_{ex} & \text{if} \quad P_{ded} = \{\} \\ (P_{ded} \cap X) \setminus P_{ex} & \text{otherwise} \end{cases} \tag{20}$$

D-1: Similar to E-1, this constraint can be resolved by grouping the associated tasks (forming a task-cluster). If tasks are grouped, the set of admissible processors for a task cluster $c$ is:

$$P_{adm}^{(c)} = \bigcap_{t \in c} P_{adm} \tag{21}$$

D-2: The set of admissible processors can be updated dynamically (during the design space exploration).

$$P_{adm.dyn} = P_{adm} \setminus P_{ex.dyn} \tag{22}$$

$$P_{ex.dyn} = P \text{ of tasks that the current task must be separated from} \tag{23}$$

C-3: If an allocation is fixed, the task allocation algorithm will not modify that allocation.

### 5.1.2 Frame Packing

In order to satisfy frame packing constraints, both aspects need to be tackled. On the one hand, the frame packing heuristic needs to be constraint-aware. On the other hand, the necessary pre-conditions (task allocation & message routing) have to be fulfilled. Within this context, pre-condition fulfillment is crucial:

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.

F-1: Similar to F-2, a message can only be packed into the dedicated frame, if they are sent from the same processor and routed via the same bus.

Assuming the pre-conditions are fulfilled, the frame packing constraints can be satisfied by the following constraint-aware packing heuristic. It consists of 4 phases, which are performed sequentially:

Phase 1: F-1: Dedicated frame packing is typically used, because the same *frame catalogue* is used within different cars. To satisfy this constraint, messages that have this constraint associated, will only be packed into the dedicated frame.

Phase 2: F-3: Messages which have this constraint associated are packed into separated frames each.

Phase 3: F-2: Messages which have this constraint associated are packed into the same frame.

Phase 4: All remaining messages (which have no constraint associated) can be packed according to the packing algorithm presented in section 4

## *5.2 Implications on Design Space Exploration*

Based on the considerations and rules presented earlier, design space exploration can be performed more efficiently. Exploration steps (performed via neighbour moves) are performed based on the following principles:

- 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.
- Frame packing is performed due to the constraint-aware packing heuristic.

As a consequence, a large number of infeasible configurations is avoided, since constraints are not violated. Thus, the efficiency of the search increases. In addition, constraint satisfaction can be guaranteed for certain constraint-types.

Unfortunately, not all constraint-types can be resolved. For a set of constraint-types (A-1, A-2, A-3, B-1, B-2, B-3, E-5) no rules how to satisfy them, could be derived. These constraints are addressed by the cost-term *constraint violations*. Thereby they can either be represented as a *mandatory* or as *desired*. 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 4 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.

By resolving constraints and using constraint-aware neighbour moves, the design space exploration can be performed more efficiently. Experimental evaluation shows that the improvements are two-fold: On the one hand, fewer constraints are violated during the search. On the other hand, the *best obtained solution* has improved attributes due to the optimization targets. Figure 1 shows the number of constraint violations during a search-run (for different constraint-types). It evidences the efficiency of the proposed approach of constraint resolving: Without resolving (just using the cost-term to punish a constraint violation) the number of violations is very high. With resolving, the number of violations is significantly lower, or in

**Table 4** 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 |
| E-5 | xx | x | depending on source of constraint (e.g. safety analysis) |

Note: All other constraint-types can be resolved. Thus they are always satisfied, and don't need to be encoded into the search algorithm any more. Options marks as "xx" represent the option preferred by the authors.

several cases even zero. More details (especially on the impact on the *best obtained solution*) can be found in [12].

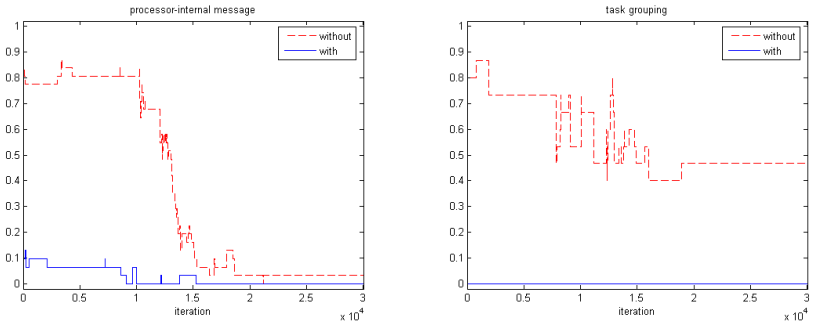


**Fig. 1** Constraint Satisfaction Efficiency without and with Constraint Resolving

## 6 Applicability to Real-World Systems

The presented methodologies in this chapter are described tool-independent. However, in order to be applicable for engineers, the methodologies need to be incorporated into state-of-the-art automotive engineering tools. Consequently, the system model (consisting of tasks, messages, processors and networks) needs to be represented in an appropriate format. This could either be a higher-level one (e.g. EAST-ADL) or a more detailed one (e.g. AUTOSAR). The generated outputs of the methodologies are again represented in such formats. Network-specific outputs, such as frame packing, could be encoded into the FIBEX format (which is the state-

of-the-art format for automotive networks). Software allocation and scheduling information could be represented in the AUTOSAR format.

The computational model assumes that data produced by a task may trigger the execution of a task which reads the data. If this was not being done, the generating task could produce another instance of the data, and overwrite the older ones. Thus the implementation of the system must ensure that these receiving tasks are executed at appropriate rates and in an appropriate order. Such implementation-specific details need to be addressed in the AUTOSAR RTE generation process (which is not covered in this chapter).

## 7 Future Research Directions

Automotive electronics and embedded software are developed in accordance to the following life cycle:

- Every few years, a new hardware-platform is developed. Here, new technology may be introduced, especially at the hardware-level. E.g. new bus protocols (like FlexRay or automotive Ethernet) and new micro-controllers (e.g. multi-core architectures). The goal of this phase is to find a platform which is extensible for future requirements. Within this phase, almost all design decisions may be modified. This phase may be called *finding a system configuration which maximizes extensibility*.
- During the following years, this system configuration is used as the basis. New technologies are rarely introduced. Modifications to the system configuration may mainly be due to 2 scenarios:
  - Minor modifications are applied to the system configuration. The goal is to improve the system. This may be called *system configuration improving*. Thereby, most design decision taken for the initial configuration must be treated as constraints.
  - Additional components (e.g. software) are added to the system, in order to meet new requirements. The goal is to find a system configuration for the extended system. This may be called *system configuration upgrade*. Thereby almost all design decisions from the initial system must be treated as constraints. In addition, the new configuration should be similar to the initial configuration, in order to reduce effort for re-verification.

Consequently, future research activities have to be two-fold: On the one hand, emerging technologies such as multi-core architectures and automotive Ethernet have to be tackled. On the other hand, it must be investigated how these development-scenarios can be addressed. Concerning the latter, basically most ingredients are already at hand.

- In order to find a platform configuration, which is extensible for future modifications/extensions, the key issue is to have test-cases (i.e. software architectures)

which represent possible future requirements. This can be addressed by using *change scenarios* [8]. In addition, the configurations can be analysed with respect to *parameter uncertainties*. Well known approaches use sensitivity analysis for task WCET [19]. This can be extended towards other parameters (e.g. periods), thus resulting in multi-dimensions robustness analysis [9].

- The second issue is to actually perform a system configuration modification. A typical improvement scenario could be: reassign priorities to tasks on a certain processor, in order to fix timing issues. Therefore, state-of-the-art optimization approaches could be used, e.g. SA. Of course, the neighbour-moves must be constraint-aware.
- Within a system configuration upgrade, both the software-architecture as well as the hardware-architecture may be subject to changes. Typically new additional software-components (and communication between software-components) are added. In order to provide sufficient execution resources, additional processors may be needed. These scenarios can be addressed as follows: In [6] it is shown, how a system configuration can be found for multi-mode systems. Thereby, the goal is to have minimal changes between modes, thus enabling efficient mode-switches. If the different versions of the system (initial system, extended system) are treated as *modes*, similar methods can be used. However, there is one significant difference: [6] assumed that the hardware-platform is the same for all modes. In a system configuration upgrade scenario, this assumption is no longer valid.
- Thus, when performing system configuration modifications and system configuration extensions, the key issue is to deal with legacy decisions. These must be treated as constraints. Therefore, constraint handling is needed. This can be addressed by the methodology presented in section 5.

In order to address and solve system configuration upgrade scenarios, the following next steps have to be performed: First, a metric for determining *changes* has to be derived, and tailored to automotive needs. Second, the approach in [6] needs to be extended, so that it can handle changes in the hardware-platform. Third, the *constraint handling* approach needs to be incorporated with the part that will deal with changes.

**Acknowledgements** 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. T. AlEnawy and H. Aydin. Energy-aware task allocation for rate monotonic scheduling. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 213–

223, 2005.
2. AUTOSAR (automotive open system architecture). http://www.autosar.org. Revision 4.0, accessed 9 Sept. 2012.
3. A. Burns, M. Nicholson, K. Tindell, and N. Zhang. Allocating and scheduling hard real-time tasks on a point-to-point distributed system. In *Workshop on Parallel and Distributed Real-Time Systems*, pages 11–20, 1993.
4. E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard problems*, chapter 2, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1996.
5. P. Emberson. *Searching For Flexible Solutions To Task Allocation Problems*. PhD thesis, University of York, Department of Computer Science, 2009.
6. P. Emberson and I. Bate. Minimising task migration and priority changes in mode transitions. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 158–167, 2007.
7. P. Emberson and I. Bate. Extending a task allocation algorithm for graceful degradation of real-time distributed embedded systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 270–279, 2008.
8. P. Emberson and I. Bate. Stressing search with scenarios for flexible solutions to real-time task allocation problems. *IEEE Transactions on Software Engineering*, 36(5):704–718, 2010.
9. A. Hamann, R. Racu, and R. Ernst. Multi-dimensional robustness optimization in heterogeneous distributed embedded systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 269–280, 2007.
10. ISO 26262: Road vehicles – functional safety. Revision 1.0.
11. D. S. Johnson and M. R. Garey. A 71/60 theorem for bin packing. *Journal of Complexity*, 1(1):65–106, 1985.
12. F. Pölzlbauer, I. Bate, and E. Brenner. Efficient constraint handling during designing reliable automotive real-time systems. In *International Conference on Reliable Software Technologies (Ada-Europe)*, pages 207–220, 2012.
13. F. Pölzlbauer, I. Bate, and E. Brenner. Optimized frame packing for embedded systems. *IEEE Embedded Systems Letters*, 4(3):65–68, 2012.
14. P. Pop, P. Eles, and Z. Peng. Schedulability-driven frame packing for multicluster distributed embedded systems. *ACM Transactions on Embedded Computing Systems*, 4(1):112–140, 2005.
15. S. Poulding, P. Emberson, I. Bate, and J. Clark. An efficient experimental methodology for configuring search-based design algorithms. In *IEEE High Assurance Systems Engineering Symposium (HASE)*, pages 53–62, 2007.
16. R. Saket and N. Navet. Frame packing algorithms for automotive applications. *Embedded Computing*, 2(1):93–102, 2006.
17. K. Sandström, C. Norström, and M. Ahlmark. Frame packing in real-time communication. In *International Conference on Real-Time Computing Systems and Applications (RTCSA)*, pages 399–403, 2000.
18. W. Zheng, Q. Zhu, M. Di Natale, and A. Sangiovanni-Vincentelli. Definition of task allocation and priority assignment in hard real-time distributed systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 161–170, 2007.
19. Q. Zhu, Y. Yang, E. Scholte, M. Di Natale, and A. Sangiovanni-Vincentelli. Optimizing extensibility in hard real-time distributed systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 275–284, 2009.