Towards Self-Adaptation in Real-time, Networked Systems: Efficient Solving of System Constraints for Automotive Embedded Systems

Marc Zeller, Christian Prehofer, Gereon Weiss, Dirk Eilers, Rudi Knorr
Fraunhofer Institute for Communication Systems ESK
Munich, Germany
{marc.zeller, christian.prehofer, gereon.weiss, dirk.eilers, rudi.knorr}@esk.fraunhofer.de

Abstract—While there has been considerable work on self-adaptive systems, applying these techniques to networked, embedded systems poses several new problems due to the requirements of embedded real-time systems. Among others, we have to consider memory and hardware limitations, as well as task schedulability and timing dependencies. The goal of this paper is to find a correct placement of software components efficiently, even though most of these individual constraints are highly intractable (NP-complete). This is a prerequisite for runtime adaptation in such domains and can be used for system optimization, extension or failure handling.

We introduce an integrated model of system constraints for efficient computation of software component allocation, focusing on automotive embedded systems. For solving these, we have developed and compared two techniques based on SAT solving and Simulated Annealing, which enforce placement constraints efficiently. This reduces the size of the constraints significantly, but still leads to 2 million variables and more than 126 thousand equations in our case study with realistic automotive system settings. We show that both approaches provide solutions in several seconds on current commodity hardware, and show that SAT solving is more efficient for larger sets of equations.

Keywords—runtime adaptation; networked embedded systems; automotive; constraints; allocation

I. INTRODUCTION

There has been considerable work on self-adaptive systems, which can reconfigure their software configuration at runtime [1], [2], [3]. However, applying these techniques to networked, embedded systems poses several new problems due to limitations and reliability requirements of embedded systems [4]. This means that resource constraints of the hardware platforms and networks have to be considered. In particular, we focus on automotive embedded systems, where the main constraints are

- limited memory resources
- task schedulability
- timing dependencies between software components
- heterogeneous hardware platforms
- different sub-networks connected by a gateway

The goal of this paper is to find a practical model of the above constraints which delivers sound solutions in reasonable time.

These requirements are well studied in embedded systems research and there exist specific and sophisticated mathematical models of the constraints (e.g. [5], [6]). Typically, the main goal is to optimize resource usage and to find optimal solutions. Some approaches also consider constraints which cover design time (e.g. wiring constraints) and runtime (e.g. [7], [8]). In turn, most of these individual formalisms are computationally highly demanding and do not scale well. For instance most of the above problems are \( NP \)-complete (e.g. schedulability analysis of tasks).

Here, our goal is to find valid solutions for runtime reconfigurations, considering all of the above constraints. This means, we focus on the efficiency and scalability of constraint solving and consider this more important than optimal solutions. Furthermore, we have to consider all the above constraints at the same time. The motivation for this comes from the fact that adaptation of an embedded system may have to be done during runtime. For instance, in case of failures finding a sound solution quickly and within a known time period is more important than unrestricted search for optimal solutions.

Automobiles are a prominent example for a complex networked, embedded system. Modern automobiles consist of an increasing number of interconnected electronic devices - so-called Electronic Control Units (ECUs). Innovations within the automotive domain are mainly introduced by software, e.g. driver assistance features. Thus, today we find...
about 2000 software functions distributed over up to 100 ECUs connected via multiple networks in modern vehicles [9]. Fig. 1 shows the in-vehicle network of a typical upper class automobile.

Enhancing nowadays automotive embedded systems with self-management capabilities [10] provides a promising solution for the current challenges in automotive embedded systems. Self-* properties like self-configuration, self-healing, self-optimization or self-protection [11] improve the scalability, robustness and flexibility of the system [12]. Since not all possible situations which lead to a reconfiguration of the system can be foreseen during design, the adaptation of the system may have to be calculated during runtime by solving the previously mentioned constraints. Otherwise, systems use very conservative fall-back solutions which often means that all affected features have to be disabled.

We introduce an integrated model of system constraints for quick computation of software component allocation, focusing on automotive embedded systems. Even though the model is simpler than others which focus on specific aspects, its application to realistic automotive system settings leads to more than 2 million variables and more than 126 thousand equations. Secondly, we show that such systems can be solved efficiently in several seconds on current PC-like hardware. For this, we have compared optimized techniques based on SAT solving and Simulated Annealing, where SAT solving scales better for our optimized set of equations.

The remainder of this paper is organized as follows: First, we will present a formal system model for automotive embedded systems. Section III will give a brief definition of terms and explain the principles of runtime adaptation in automotive embedded systems. In Sec. IV we present an overview of methods to solve this set of constraints. Afterwards, we illustrate the benefits of our approach in an experimental evaluation. In Sec. VII related work is discussed. Finally, the paper is concluded in Sec. VIII.

II. FORMAL SYSTEM MODEL

In the following, we introduce a formal definition of automotive embedded systems.

An automotive embedded system $\mathcal{A}$ is a heterogeneous distributed real-time system consisting of a set of inputs $I$ (sensors), a set of functionalities $F = \{f_1, \ldots, f_n\}$ (features), and a set of outputs $O$ (actuators). The set of functionalities is realized by a set of software functions $SW$, where each feature $f_i$ is implemented by a set of software functions $SW_{f_i}$ and $SW_{f_i} \subseteq SW$. Software functions, sensors and actuators are connected to each other in a specified way. This so-called function network can be represented by a directed graph $G_f(V_f, E_f)$. The vertices $V_f$ represent software functions, actuators or sensors. The directed edges $e_f$ indicate a data flow from one vertex to another by sending messages. Each vertex may have multiple incoming edges and multiple outgoing edges.

The physical resources of the automotive in-vehicle network are modeled by an undirected graph $G_r(V_r, E_r)$. The vertices $V_r$ represent the vehicles’ ECUs. The edges $e_r$ correspond to available communication links.

The so-called system configuration $c$ describes the allocation of the software functions $(S_1, \ldots, S_n)$ to the available ECUs $(E_1, \ldots, E_m)$ at time $t$:

$$c(t) : V_f \rightarrow V_r = \{0, 1\}^{n \times m}$$

with $n \cdot m$ variables $x_{i,j}$. If the software function $i$ is assigned to ECU $j$, then $x_{i,j} = 1$, else it is 0. The general problem of finding a mapping for a set of tasks on a set of resources is known to be $NP$-hard [13].

Moreover, $\mathcal{A}$ consists of a set of constraints $\Psi = \{\psi_1, \ldots, \psi_p\}$ which enable the definition of valid allocations. A linear constraint $\psi \in \Psi$ is defined as a Boolean formula of the following form:

$$\psi = \left( \sum_{i=1}^{n} \sum_{j=1}^{m} a_{i,j} x_{i,j} \right) \circ b = \{true, false\}$$

with the Boolean literals $x_{i,j} \in \{0, 1\}$ which represent the allocation of the software function $S_i$ to the ECU $E_j$, the coefficients $a_{i,j}, b \in \mathbb{R}$, and the operator $\circ \in \{\leq, \leq, =, \geq, \geq\}$

With this formal system model it is possible to describe the runtime adaptation of a self-adaptive automotive embedded system.

III. RUNTIME ADAPTATION

Self-adaptive automotive software systems can be realized by adapting the structure of the system at runtime in response to changes in the environment or within the system itself (Structural Adaptation) [14]. In the context of automotive embedded systems, structural adaptation means to find a new mapping of the function network $G_f$ to the existing physical in-vehicle network $G_r$. To adapt an automotive system in a flexible and adequate way, new system configurations are determined during runtime in order to consider the current system conditions.

Since automotive embedded systems provide safety-relevant applications, these systems must preserve the predefined requirements to guarantee the proper system behavior at all time. Thus, the system configuration $c$ of a self-adaptive automotive system is called valid at time $t$, if all constraints $\Psi$ are satisfied:

$$\bigwedge_{\psi \in \Psi} \psi_f(t) = true \iff c(t) \text{ is valid}$$

The aim of the runtime adaptation is to provide a valid system configuration at all times.
During runtime, each constraint of the system can be monitored by a control instance. If one of the defined constraints is not met anymore, the structure of the system can be adapted in order to meet the constraints.

Describing a self-adapting system by a set of constraints allows the structural adaptation of the system only within predefined boundaries [4]. Thereby, the system is prevented from emerging invalid allocations and uncontrolled behavior. Since only a small part of all possible system configurations are valid, it is difficult to determine a new, valid system configuration during runtime - especially for an embedded system with restricted resources. Therefore, it is important to find valid allocations (configurations) in reasonable time.

In the next section, we present an integrated and correct set of constraints to define valid allocations in self-adaptive automotive embedded systems which enables the efficient solution of the set of equations with respect to performance.

IV. MODEL OF SYSTEM CONSTRAINTS

The allocation of software functions in automotive embedded systems must fulfill certain predefined requirements to be valid. Therefore, we present the following set of equations or in-equations. This set is optimized to be solved during runtime in order to enable the computation of valid system configurations in a self-adaptive automotive embedded system in reasonable time.

All constraints deal with worst case assumptions. This may lead to the exclusion of valid allocations in our model, but guarantees that the solutions of the set of (in-)equations are valid allocations.

During the development process, the designer may define a so-called Allocation Set (AS) for each software function $S_i \in SW$. An AS defines a set of ECUs on which the software function can be allocated:

$$\forall S_i \in SW : AS_{S_i} = \{E_x | S_i \text{ can be assigned to } E_x \}$$

with $1 \leq x \leq m$ and $1 \leq |AS_{S_i}| \leq m$.

Furthermore, an AS can be defined for each ECU representing the set of software functions which can be allocated on the ECU:

$$\forall E_j \in V_e : AS_{E_j} = \{S_y | S_y \text{ can be assigned to } E_j \}$$

with $1 \leq y \leq n$ and $1 \leq |AS_{E_j}| \leq n$.

By setting up Allocation Sets during the design, developers have the opportunity to restrict the number of allocation choices for each software function individually. Thereby, dependencies of software functions to certain hardware platforms or network systems, as well as safety requirements (e.g. the airbag function is not allowed to be migrated) can be resolved during the design. This reduces the set of (in-)equations which must be resolved during runtime to determine a new valid allocation.

We illustrate the system constraints in this section with a small example within the vehicle’s comfort domain. Our example represents the Keyless Entry feature of a car which is composed of the software functions $S_1, \ldots, S_5$. A driver can approach his car with the key in his pocket and the doors will unlock automatically. Antenna components detect the key in the surrounding and inform the central door locking function, which in turn unlocks the doors. Moreover, the door locking control interacts with the exterior light feature of the vehicle which gives feedback via the direction indication lights. In Fig. 2 all software functions, sensors and actuators used in our example as well as the communication dependencies between them are illustrated. In addition, a timing requirement for the Keyless Entry feature in our example is defined in form of an end-to-end timing chain as follows:

$$TC_1 : Sen_{1,2,3} \rightarrow S_1 \rightarrow S_2 \rightarrow Act_{1,\ldots,4} \leq 1000$$

Fig. 3 shows a sub-system of an automotive in-vehicle network, consisting of four ECUs on which the five software functions can be assigned to. These ECUs are interconnected by two low speed Controller Area Network (CAN) buses [15] with a bandwidth of 125 kbit/s each and a worst case transmission delay of 2.16 ms for a CAN message [7]. The network buses CAN 1 and CAN 2 are interconnected by a gateway with a worst case throughput of 250 kbit/s.
For our example, we set up the following Allocation Sets:

\[ AS_{S_1} = \{ E_1 \} \]
\[ AS_{S_2} = \{ E_2, E_3 \} \]
\[ AS_{S_3} = \{ E_1, E_2, E_3, E_4 \} \]
\[ AS_{S_4} = \{ E_1, E_2, E_4 \} \]
\[ AS_{S_5} = \{ E_1, E_2, E_3, E_4 \} \]

A. Assignment

Automotive in-vehicle networks consist of heterogeneous hardware platforms (ECUs). Thus, a software function can only be executed on a subset of all available ECUs. Moreover, each software function must only be assigned to exactly one of these ECUs. This requirement is defined as follows:

\[ \forall S_i \in SW : \sum_{E_j \in AS_{S_i}} |AS_{E_j}| x_{i,j} = 1 \]

Example: For our example the following assignment constraints are defined based on the predefined Allocation Sets:

\[ S_1 : x_{1,1} = 1 \]
\[ S_2 : x_{2,2} + x_{2,3} = 1 \]
\[ S_3 : x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} = 1 \]
\[ S_4 : x_{4,1} + x_{4,2} + x_{4,3} + x_{4,4} = 1 \]
\[ S_5 : x_{5,1} + x_{5,2} + x_{5,3} + x_{5,4} = 1 \]

B. Resources

Within the automotive domain, hardware resources are strictly limited. Resource constraints define the capability of an ECU to execute a software function:

\[ \forall \rho, E_j \in V_r : \sum_{S_i \in AS_{E_j}} |AS_{S_i}| a_{i,\rho} x_{i,j} \leq b_{j,\rho} \]

where \( a_{i,\rho} \) is the amount of resource \( \rho \) required by the software function \( S_i \), and \( b_{j,\rho} \) is the maximum amount of resource \( \rho \) available on \( E_j \). Typical resources in automotive control units are volatile memory (RAM) and non-volatile memory (ROM/Flash). Apart from memory requirements, the constraints for other resources can be modeled similarly.

Example: The maximum available volatile memory on ECU \( E_1 \) in our example is 128 kByte. Software function \( S_1 \) and \( S_3 \) require 64 kByte each in worst case; \( S_1 \) and \( S_5 \) require up to 32 kByte each. With these information the resource constraint for \( E_1 \) is defined as follows:

\[ E_1 : 64 + 64x_{3,1} + 32x_{4,1} + 32x_{5,1} \leq 128 \]

C. Scheduling

In our model, we consider the scheduling of real-time tasks with fixed and dynamic priorities.

Present automotive embedded systems are mostly using a task scheduling mechanism based on fixed priorities. For example, the AUTOSAR [16] runtime environment is a standardization software architecture for automotive systems which uses the fixed priority preemptive scheduling strategy. To check if all tasks running on the vehicles’ ECUs can fulfill their deadlines, the so-called Response Time Analysis (RTA) [17] can be applied. The RTA is a sufficient and necessary condition for schedulability for preemptive task sets with arbitrary fixed priorities. By using the upper bound for response times introduced in [18], the following linear constraints are defined for schedulability analysis:

\[ \forall E_j \in V_r \forall S_i \in AS_{E_j} : WCET_{i,j} x_{i,j} + \sum_{k=1, \forall \rho \in AS_{E_j}, \text{prio}_k > \text{prio}_i} |AS_{E_j}| \left( WCET_{k,j} \left( 1 - \frac{WCET_{k,j}}{T_k} \right) + \frac{WCET_{k,j}}{T_k} D_i \right) \alpha_{i,k,j} \leq D_i \]

with

\[ \forall i, k \in \{ 1, ..., |AS_{E_j}| \}, j \in \{ 1, ..., m \} : \]
\[ x_{i,j} + x_{k,j} - 2\alpha_{i,k,j} \geq 0 \]
\[ \forall i, k \in \{ 1, ..., |AS_{E_j}| \}, j \in \{ 1, ..., m \} : \]
\[ \alpha_{i,k,j} - x_{i,j} - x_{k,j} \geq -1 \]

where \( WCET_{i,j} \) is the Worst Case Execution Time of the software function \( S_i \) on ECU \( E_j \), \( T_i \) is the period, \( \text{prio}_j \) is the priority and \( D_i \) is the deadline of the software function \( S_i \). \( \alpha_{i,k,j} \) is a helper variable which are needed to keep the in-equation linearly (see transformations in [19]).

Other common approaches for scheduling real-time tasks are based on dynamic priorities. The Earliest Deadline First (EDF) scheduling mechanism [20] is such an approach where the priority of a task is calculated dynamically according to its relative deadline. For the schedulability analysis the following constraint based on the utilization-based schedulability analysis for EDF [20] is defined as follows:

\[ \forall E_j \in V_r : \sum_{S_i \in AS_{E_j}} \frac{WCET_{i,j}}{\min (D_i, T_i)} x_{i,j} + \leq 1 \]

This formula is a sufficient condition for schedulability for task sets scheduled according to EDF under the assumption \( D_i \leq T_i \) and all tasks \( S_i \) in \( SW \) are periodic [21].

Example: For the ECU \( E_4 \) in our Keyless Entry example, the schedulability constraint based on RTA is defined as
follows (assuming that the priority of the software function equals its number and \( D_i = T_i \)):

\[
WCET_{3,4}x_{3,4} + \left( 2 \cdot WCET_{4,4} - \frac{WCET_{4,4}^2}{10} \right) \alpha_{3,4,4} + \\
\left( 2 \cdot WCET_{5,4} - \frac{WCET_{5,4}^2}{10} \right) \alpha_{3,4,5} \leq 25
\]

\[
WCET_{4,4}x_{4,4} + \\
\left( 2 \cdot WCET_{5,4} - \frac{WCET_{5,4}^2}{10} \right) \alpha_{4,4,5} \leq 10
\]

\[
WCET_{5,4}x_{5,4} \leq 10
\]

\[
x_{3,4} + x_{4,4} - 2\alpha_{3,4,4} \geq 0
\]

\[
x_{3,4} + x_{5,4} - 2\alpha_{3,4,5} \geq 0
\]

\[
x_{4,4} + x_{5,4} - 2\alpha_{4,4,5} \geq 0
\]

\[
\alpha_{3,4,4} - x_{3,4} - x_{4,4} \geq -1
\]

\[
\alpha_{3,4,5} - x_{3,4} - x_{5,4} \geq -1
\]

\[
\alpha_{4,4,5} - x_{4,4} - x_{5,4} \geq -1
\]

The schedulability constraint for \( E_4 \) using EDF scheduling is defined as follows:

\[
\frac{WCET_{3,4}}{25}x_{3,4} + \frac{WCET_{1,4}}{10}x_{4,4} + \frac{WCET_{5,4}}{10}x_{5,4} \leq 1
\]

**D. In-vehicle Networks**

Modern automotive in-vehicle networks consist of different sub-networks (see Fig. 1), where each network poses a communication resource itself. Network constraints define if a network bus \( \nu \) has enough bandwidth available, so that all software functions attached to the network bus are able to communicate properly:

\[
\forall \nu : \sum_{i=1}^{n} a_i \cdot \sum_{j=1, E_j \in AS_{S_i}}^{\nu} x_{i,j} \leq b_{\nu}
\]

where \( a_i \) is the maximum required bandwidth of the software functions \( S_i \) and \( b_{\nu} \) is the maximum bandwidth of the network \( \nu \). This constraint is independent from the network system or protocol used. Considering the properties of specific automotive network buses with more sophisticated constraints for event-triggered or time-triggered messages (see [5]) would lead to more accurate solutions. But the resulting set of equations would be much larger. Therefore, we use the constraint above to analyze the in-vehicle networks in our model.

Example: The constraint for CAN 2 in our Keyless Entry example is:

\[
50 + 10 (x_{2,2} + x_{2,3}) + 160 (x_{3,1} + x_{3,2} + x_{3,3}) + \\
100 (x_{4,1} + x_{4,2} + x_{4,3}) + \\
100 (x_{5,1} + x_{5,2} + x_{5,3}) \leq 125000
\]

**E. Topology**

Modern automotive in-vehicle networks consist of multiple sub-networks which are all interconnected by a central gateway (see Fig. 1). The network traffic resulting from the allocation of software functions to ECUs must not exceed the throughput of this gateway. Thereby, the linear topology constraint for the central gateway topology is defined as follows:

\[
\sum_{i=1}^{n} \sum_{j=1, E_j \in AS_{S_i}}^{\nu} a_i \cdot k \delta_{i,k,j} \leq b_{gw}
\]

where \( a_{i,k} \) is the maximum required bandwidth of the communication between the software functions \( S_i \) and \( S_k \). \( b_{gw} \) is the maximum throughput of the central gateway. \( E_j \) means that there is a directed edge between \( S_i \) and \( S_k \) in the functional network \( G_f \). \( \nu_i \neq \nu_j \) means that \( E_j \) is not connected to the same sub-network \( \nu \) as ECU \( j \). \( \delta_{i,k,j} \) is a helper variable (see transformations in [19]).

Example: For the running example, the constraint for the central gateway topology is defined as follows:

\[
50 \delta_{1,2,1} + 10 \delta_{2,3,2} + 10 \delta_{3,3,3} + 160 \delta_{3,4,1} + 160 \delta_{3,4,2} + 160 \delta_{3,5,2} + 160 \delta_{3,4,3} + 160 \delta_{3,5,3} + \\
160 \delta_{3,4,4} + 160 \delta_{3,5,4} \leq 250
\]

\[
1 - \delta_{1,2,1} \geq 0
\]

\[
x_{2,2} + x_{3,4} - 2 \delta_{2,3,2} \geq 0
\]

\[
x_{2,3} + x_{3,4} - 2 \delta_{2,3,3} \geq 0
\]

\[
x_{3,1} + x_{4,4} - 2 \delta_{3,4,1} \geq 0
\]

\[
x_{3,2} + x_{4,4} - 2 \delta_{3,4,2} \geq 0
\]

\[
x_{3,3} + x_{4,4} - 2 \delta_{3,4,3} \geq 0
\]

\[
x_{3,2} + x_{4,1} + x_{4,2} + x_{4,3} - 4 \delta_{3,4,4} \geq 0
\]

\[
x_{3,1} + x_{5,4} - 2 \delta_{3,5,1} \geq 0
\]
functions without loops and branches. Existing branches timing chain is always a linear linked list of software within the timing chain (see Fig. 4). Therefore, sink of the data which are processed by AUTOSAR tasks set of sensors and actuators, which define the source and the chain \(S\). The Worst Case Transmission Delay (WCTD) of the software function \(S\) is the end-to-end deadline of the timing chain \((TC)\). \(WCTD\) is the Worst Case Response Time (WCRT) \(k\) of \(S\). In the automotive domain, a timing chain also consists of a end-to-end timing constraint within the AUTOSAR system is defined as follows:

\[
\forall TC : \sum_{S_i \in TC} (WCRT_i + WCTD_i) + WCRT_k \leq D_{TC}
\]

where \(WCRT_i\) is the Worst Case Response Time (WCRT) of the software function \(S_i\), \(WCTD_i\) is the Worst Case Transmission Delay (WCTD) of the message \(m_i\) send from \(S_i\) to \(S_i+1\), and \(D_{TC}\) is the end-to-end deadline of the timing chain \(TC\).

In the automotive domain, a timing chain also consists of a set of sensors and actuators, which define the source and the sink of the data which are processed by AUTOSAR tasks (see Fig. 4). Therefore, \(k+1\) messages are interchanged within the timing chain \((m_0, ..., m_{k+1})\). In our model, a timing chain is always a linear linked list of software functions without loops and branches. Existing branches in the function network \(G_f\) must be eliminated during the design to determine the required set of timing chains. Thereby, the number of timing chains may be increased, but the system’s functionality is not affected by the elimination of branches and loops. According to [23], an end-to-end timing constraint for each \(TC\) is connected to.

\[
\forall i \in \{1, ..., k-1\}, j \in \{1, ..., |AS_{S_i}|\}:
- x_{i,j} + x_{(i+1),j} + 2\alpha_{i,i+1,j} \leq 1
\]

\[
\forall i \in \{1, ..., k-1\}, j \in \{1, ..., |AS_{S_i}|\}:
- x_{i,j} - x_{(i+1),j} - \alpha_{i,i+1,j} \leq 0
\]

\[
\sum_{m \in SW} Q_{m_{0,q}} + W_{m_{0,q}} \geq \alpha_{1,q} + 1
\]

\[
\sum_{E_j \in AS_{S_i}} (T_i + WCET_{i,j}) x_{i,j} + (Q_{m_{i,j}} + W_{m_{i,j}}) \alpha_{i,i+1,j} + 1
\]

\[
\sum_{E_j \in AS_{S_k}} (T_k + WCET_{k,j}) x_{k,j} + (Q_{m_{k,r}} + W_{m_{k,r}}) \alpha_{k,r}
\]

with

\[
Q_{m_{i,j}}\text{ represents the maximum queuing delay for the network protocol used to transmit } m_{i,j}\text{, and } W_{m_{i,j}} \text{ is the worst case transmission time of the message } m_{i,j}\text{ on the network bus where ECU } E_j \text{ is connected to. } T_i \text{ represents the worst case delay on the receiver side (period of task } S_i\text{). } m_{0,q} \text{ is the input of the timing chain } TC \text{ and } q \text{ is the index of the ECU where the input sensor is attached to. Since the output actuator of the timing chain can be triggered directly on receiving the message } m_{k,r}, \text{ there is no additional delay of the receiver side. } r \text{ is the index of the ECU where the actuator is attached to.}
\]

Depending on the automotive network system used to transmit the message \(m_{i,j}\), \(Q_{m_{i,j}}\) and \(W_{m_{i,j}}\) are defined as follows:

\[
\sum_{m \in SW} WCTD_i + WCRT_i \leq D_{TC}
\]
• FlexRay [24] (static segment): $Q_{m,i,j} = T_{m,i,j} \cdot \gamma$ and $W_{m,i,j} = v$ ($T_{m,i,j}$ is the period of $m_{i,j}$, $\gamma$ is the duration of the FlexRay communication cycle and $v$ is the duration of one static slot within the FlexRay cycle).

• CAN [15]: $Q_{m,i,j} = w_{m,i,j}$ is the worst case waiting time and $W_{m,i,j}$ is the worst case transmission time of $m_{i,j}$. These values can be computed using the formula provided in [25].

With these constraints worst-case timing requirements of end-to-end timing chains in automotive systems can be defined.

Example: The constraint for the timing chain $TC_1$ in our example is defined as follows:

$$(T_1 + WCET_1,1) + 2.16 + (T_2 + WCET_2,2)x_{2,2} + (T_2 + WCET_2,3)x_{2,3} + 2 \cdot 2.16 \cdot x_{2,2} + 2 \cdot 2.16 \cdot x_{2,3} \leq 1000$$

V. SOLVING THE SYSTEM CONSTRAINTS

With the equations and in-equations presented in Sec. IV a valid allocation of software functions to ECUs can be described considering the specific requirements of automotive embedded systems. Thereby, all system constraints are modeled as linear (in-)equations with Boolean literals - so-called Boolean propositional formulas. Thus, the allocation problem can be formulated as a propositional SATisfiability (SAT) problem. Solving the SAT problem determines any allocation of software functions to ECUs which satisfies the given constraints.

Furthermore, it is possible to solve the allocation problem with an heuristic optimization algorithm, e.g. Simulated Annealing (see [26]). This algorithm starts with a randomly generated allocation and iteratively optimizes this solution until a valid allocation is found (see Fig. 5). Therefore, a utility function - the so-called energy function $\epsilon$ - is needed to evaluate an allocation. The energy function consists of different components representing the constraints defined in Sec. IV. If any of the constraints is violated by the allocation of software functions to ECUs, the corresponding component within the energy functions returns a negative value. The Simulated Annealing algorithm tries to maximize the energy function. New solutions are created by changing the allocation of a random software function in the current solution. If the energy function reaches a value of 0, the optimization satisfies all given constraints. The complete energy function is defined as follows:

$$\epsilon = \sum_{S \in SW} k_1 \epsilon_{het} + \sum_{E \in V_e} k_2 \epsilon_{mem_a} + \sum_{E \in V_c} k_2 \epsilon_{mem_a} + \sum_{E \in V_e} k_2 \epsilon_{sch} + \sum_{E \in V_c} k_2 \epsilon_{sch} + \sum_{n \in A} k_3 \epsilon_{bus} + k_2 \epsilon_{gw} + \sum_{TC} k_2 \epsilon_{time}$$

with the weightings $k_1 = 15777.3$, $k_2 = 117.4$ and $k_3 = 12.4$ which are defined in [26]. $\epsilon_{het}$ penalizes the allocation of a software function on a ECU where it cannot be executed. $\epsilon_{mem_a}$ and $\epsilon_{mem_s}$ represent the resource constraint for volatile and non-volatile memory of each ECU. The schedulability criteria for each ECU is included by the component $\epsilon_{sched}$. The in-vehicle network constraint for each network bus is expressed by the component $\epsilon_{bus}$. If the allocation exceeds the throughput of the central gateway, $\epsilon_{gw}$ returns a negative value. End-to-end timing constraints are represented by $\epsilon_{time}$.

1: Generate initial random solution
2: repeat
3: Compute neighbor to current solution
4: Compute energy $\epsilon_c$ of the current solution
5: Compute energy $\epsilon_n$ of the new solution
6: if $\epsilon_n \geq \epsilon_c$ or Random(0,1) < $e^{\epsilon_n - \epsilon_c}$ then
7: current solution := new solution
8: end if
9: until maximum number of iterations is reached

Figure 5. Simulated Annealing algorithm in pseudo code

The Allocation Sets defined for each software function are integrated into the Simulated Annealing heuristic by restricting the choices of computing new solutions. Thus, each software function can only be allocated on a control unit which is part of the function’s Allocation Set.

In the following experiments, we use a SAT-solver and the Simulated Annealing heuristic to determine a valid allocation for a set of typical automotive in-vehicle network settings.

VI. EXPERIMENTAL RESULTS

In this section, we examine the efficiency of solving our model of system constraints presented in Sec. IV in different experiments using the two different approaches outlined in Sec. V to determine valid allocations.

A. Evaluation Setup

In our experiments, we evaluate our approach with typical values representing networked, embedded systems of modern automobiles (see [27], [9], [28]). Therefore, we use various setups representing different sizes and variants of automotive in-vehicle networks (see Table I). While the ratio between the number of software functions and the number of ECUs is fixed in the first setups (1.1 - 1.9), we also perform experiments with setups where this ratio is variable (setup 2.1 - 2.10). For all experiments, we assume that each feature of the vehicle consists of about 8 software functions as well as any number of sensors and actuators. Moreover, for half of the features an end-to-end timing chain is defined consisting of one sensor input, up to 8 software functions, and one actuator. The volatile and the non-volatile memory
requirements of each is ECU is represented as resource constraints. In our experiments, the in-vehicle network consists of three different kinds of hardware platforms. Each software function can only be executed on one kind of hardware platform. Thus, the Allocation Set for each software function / ECU contains 33% of the vehicles’ ECUs / software functions.

The Simulated Annealing heuristic presented in Sec. V and the SAT-solver SAT4J [29] are used in the experiments to determine a valid allocation of software functions to ECUs. The so-called “temperature” of the Simulated Annealing algorithm in our experiments is starting with a value of 10.0 and is linearly decreasing with increasing iterations until reaching a value of 1.0. All experiments are performed on a standard PC with a Quad-Core AMD Opteron Processor @2.8 GHz and 8 GByte RAM.

**B. Results**

At first, we list the number of constraints and literals needed to solve the SAT problem for the given setups in table II using either fixed priorities or dynamic priorities for scheduling tasks on the ECUs (see Sec. IV-C). The results of our experiments show that the number of constraints and literals increases approximately linearly with the size of the automotive in-vehicle network when using the EDF strategy for task scheduling. When using scheduling based on fixed priorities, the number of constraints and literals in our model are increasing exponentially with the size of the problem. Moreover, it is only possible to build a set of system constraints for very small setups. For larger problems the number of constraints and literals lead to out of memory exceptions when setting up the SAT-solver.

For the variant of our model using dynamic priorities for task scheduling (see Sec. IV-C), we measure the time needed to determine a valid allocation of software functions to ECUs using the Simulated Annealing heuristic and the SAT-solver. The results of these performance measurements for all setups are illustrated in Fig. 6 and Fig. 7. Our measurements show that the SAT-solver performs better than the Simulated Annealing heuristic. Especially with the growing size of the problem a performance optimization of up to 92% is reached. For very large problems (setup 1.8, 1.9 and 2.8, 2.9, 2.10) the Simulated Annealing algorithm is not able to find one of the few valid allocations in reasonable time. With the SAT-solver a solution for these setups is found in a few seconds. For very small automotive in-vehicle network settings the Simulated Annealing approach performs better than the SAT-Solver (see setup 1.1 and 2.1). For setups with many valid allocations the Simulated Annealing has a high chance to find a valid solution with its random initial solution.

The results of our experiments clearly illustrate that using a common SAT-solver and our model of systems constraints introduced in Sec. IV provides a scalable and efficient approach to determine a valid allocation of software functions to ECUs for a self-adaptive automotive embedded system. Due to the reduction of the number of constraints and literals by setting up Allocation Sets for each of the software functions and control units, it is possible to determine a valid allocation of software to ECUs efficiently even for automotive embedded systems with a large number of ECUs and software functions.
In today’s automotive embedded systems, the possible allocation choices of the software functions are much smaller than in our experiments, because of the heterogeneity of current control units and network buses. Thereby, the determination of a new valid allocation can be performed even faster than in our experiments. But in future automobiles - like fully electric vehicles - the number of allocation choices may be grow due to the current trend of equipping automotive embedded systems with more powerful and uniform control units.

VII. RELATED WORK

In this section, we discuss relevant related work. In [30] and [31] generic meta-models for self-managing systems are proposed, but no concrete constraints for preserving the systems’ requirements are presented. Other models only include different aspects of networked, embedded system. The model outlined by Zheng et al. in [32] solely covers timing constraints of periodic tasks and event-triggered messages. Feng et al. introduce a model for placing tasks to ECU on resource constraints (CPU limitation) and task dependencies [33]. In [34] a model for self-healing automotive systems is proposed which includes resource constraints (e.g. memory requirements) but which does not represent the interdependencies and the information flow between software components. In [7] and [8] detailed models are outlined which are tailored for the static placement of software to ECUs and the design space exploration during the design. These models focus on different kinds of parameters (e.g. costs of wiring ECUs) and the resulting sets of constraints are not applicable to determine a valid allocation during runtime adaptation. Compared to our model of placement constrains existing models, e.g. [5] and [6], are more sophisticated regarding scheduling of tasks and messages, but omit timing dependencies and network topology requirements, and still need several minutes/hours to be solve.

The present system models for automotive embedded systems focus either on the static placement of software on ECUs during the design and provide a model which is too complex to be resolved in reasonable time, or provide an abstract model which is not realistic enough to guarantee the predefined system behavior. We presented an integrated model of system constraints for automotive embedded systems which can be solved in a few seconds and scales.

VIII. CONCLUSION

This is the first paper to show an efficient computation of system constraints (in a few seconds) for realistic embedded, networked systems. Towards self-adaptive embedded-networked systems, our main contributions are as follows:

- We have defined an integrated and correct model of system constraints for realistic automotive embedded systems which can be solved efficiently, as needed for self-adaptive systems. Our model includes scheduling, network resources, and application timing dependencies to ensure correct behavior of time-critical software in such embedded systems.
- We have shown that our placement constraints can be solved in two ways, SAT-solving and Simulated Annealing, where SAT-solving scales better but is less efficient for small problems.
- Even though solving the above constraints is \( \mathcal{NP} \)-complete, we have shown several ways to simplify the constraints such that the number of constraints and literals grows linearly with the problem size. These simplifications on scheduling as well as on network resources and topology are essential to solve the placement constraints for realistic settings with thousands of software components.

Our main goal was to develop system constraints which can be solved efficiently with respect to performance. The main reason is that self-adaptation is often a time-critical process in itself and has to be performed efficiently. For instance, in case of a system component failure, a new, valid allocation of software components has to be found in very
short time. Furthermore, we have only covered the time for finding solutions, not for the actual system adaptation or the detection of failures, which takes further time. Thus, our goal was to find solutions in few seconds or less. Further system optimizations, possibly with more sophisticated methods, may be performed later once the system is back in operation.

The above simplifications mean that in some cases other, more complex models, may find solutions which our constraints do not cover. However, analyzing task scheduling with fixed priorities results in an exponential growth of the number of equations.

REFERENCES


