Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2012, Article ID 971430, 15 pages
doi:10.1155/2012/971430

Research Article

A Multi-Layered Control Approach for Self-Adaptation in Automotive Embedded Systems

Marc Zeller and Christian Prehofer

Fraunhofer Institute for Communication Systems ESK, Hansastraße 32, 80686 Munich, Germany

Correspondence should be addressed to Marc Zeller, marc.zeller@esk.fraunhofer.de

Received 7 June 2012; Accepted 27 August 2012

Academic Editor: Phillip A. Laplante

We present an approach for self-adaptation in automotive embedded systems using a hierarchical, multi-layered control approach. We model automotive systems as a set of constraints and define a hierarchy of control loops based on different criteria. Adaptations are performed at first locally on a lower layer of the architecture. If this fails due to the restricted scope of the control cycle, the next higher layer is in charge of finding a suitable adaptation. We compare different options regarding responsibility split in multi-layered control in a self-healing scenario with a setup adopted from automotive in-vehicle networks. We show that a multi-layer control approach has clear performance benefits over a central control, even though all layers work on the same set of constraints. Furthermore, we show that a responsibility split with respect to network topology is preferable over a functional split.

1. Introduction

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

(i) limited memory resources,

(ii) heterogeneous hardware platforms,

(iii) different subnetworks connected by a gateway,

(iv) various requirements of different functionalities, and

(v) high demand on safety and reliability.

The focus of this paper is on self-adaptation in automotive embedded systems using a hierarchical, multi-layered control approach implemented by concrete instances of a control architecture.

Today's automobiles consist of an increasing number of interconnected electronic devices—so-called electronic control units (ECUs)—which realize most functionalities of the car by software. This networked embedded system keeps the vehicle running by controlling the engine and the breaks, provides active safety features (e.g., antilock breaking system), makes driving more convenient, and entertains the passengers with a large number of information and comfort services (e.g., air conditioning, audio player). Especially modern driver assistance systems, which distribute their functionality over several components, increase the complexity of today's vehicular embedded systems enormously. Managing nowadays vehicle software systems means managing over 2,000 software components, running on up to 100 ECUs [5].

Enhancing automotive embedded systems with self-adaptation provides a promising solution for the current challenges in automotive embedded systems [6]. So-called self-* properties, like self-configuration, self-healing, self-optimization or self-protection [7] improve the scalability, robustness, and flexibility of the system [8]. With the size of automotive systems, it becomes difficult to calculate all configurations and all failure cases in advance. Hence, the adaptation of the system may have to be calculated during runtime. In this work, we focus on the adaptation control of

the system after the breakdown of a hardware platform. The actual reconfiguration of the system itself is not the focus here and can, for instance, be performed during a (partial) restart of the system.

For realizing systems with these self-managing capabilities, a control component is needed [9]. This external component supervises the system and initiates the adaptation during runtime using closed feedback-loops (so-called control loops) [10]. In the autonomic computing *(AC)* paradigm, the elements of the system are managed by control loops based on the so-called *MAPE-K cycle* [11] which optimizes the operation of the supervised elements and enables the realization of self-* properties. Such a control loop continuously monitors and analyzes the system and its environment. Based on this information it plans the next steps and executes the planned actions. The different phases have access to a common knowledge base which provides information about the supervised elements or system.

Especially for automotive systems with various requirements and constraints, enabling self-adaptation and building a control architecture are a challenging task. Different aspects of the automotive system like safety issues must be considered by the control components appropriately. Furthermore, the control architecture has to react quickly to changing conditions, either from inside the system (e.g., hardware or software failures) or from outside the system (e.g., changing environmental conditions).

The control component of a self-adaptive system can be realized in different ways. Either a single-centralized control entity may realize the adaptation of a software system, or multiple control components may realize the adaptation of composite of software systems in a decentralized manner [12]. While centralized control may not be efficient for realizing adaptation in large, complex, and heterogeneous systems (e.g., automotive embedded systems), fully decentralized approaches may lead to a coordination overhead within the system.

An alternative is hierarchical multi-layered control architectures, as discussed in [11], where multiple control loops cooperate to achieve adaptation. In this work, we model such automotive systems using a set of constraints as introduced in [13]. Based on this, we can define the operation and responsibility of each local control cycle in the hierarchy based on different functional criteria. Thus, resulting in a control architecture with a certain number of control loops arranged in a specific way using multiple layers. Even though all control cycles work on the same set of constrains, local responsibility means that only some variables can be controlled and the others are fixed. This also means that some constraints are out of scope for some local control cycles. We compare different options, regarding different splits of responsibility as well as multi-layered control versus centralized control.

In summary, the main contributions of this paper are as follows.

(i) We introduce different instances of the multi-layered control approach for automotive embedded systems which hierarchically enforce system requirements on several layers. Therefore, the software components of the system are clustered based on different functional criteria realizing a control architecture with different numbers of layers.

(ii) We compare these concrete instances of the multi-layered approach in a self-healing scenario with realistic setups of up to 100 ECUs. It is shown that local repair based on a layered control approach in such a system is more efficient than a pure central approach. Secondly, it shows that a responsibility split based on locality w.r.t. network topology performs better than a split regarding functional areas.

The structure of this paper is as follows. In Section 2 a brief introduction to automotive embedded systems is given. Section 3 describes our approach for hierarchical, multi-layered control. Afterwards, we propose concrete instances of our multi-layered control approach for self-adaptive automotive software system. In Section 5, we outline how to realize self-adaptation during runtime using our multi-layered control approach. Moreover, we introduce a coordination mechanism for the different control loops within the multi-layered control architecture. Section 6 presents the results of the experiments which we performed to compare our concrete instances of the multi-layered approach with a centralized control approach. Section 7 discusses related work.

2. Automotive Embedded Systems

An automotive embedded system is a distributed real-time system with heterogeneous hardware platforms (ECUs) interconnected by different network buses [5]. Moreover, the automotive embedded system consists of various functionalities which are implemented in software and which must satisfy different requirements.

Thereby, an automotive embedded system A consists 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 components SW, where each feature f_i is implemented by a set of software components SW_{f_i} with $\mathrm{SW}_{f_i} \subset \mathrm{SW}$. Software components, 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 components, 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 so-called system configuration c describes the allocation of the software components ($\mathrm{SW} = \{s_1, \ldots, s_n\}$) to the available ECUs ($P = \{p_1, \ldots, p_m\}$) at time t

$$c(t) : \mathrm{SW} \longrightarrow P = \{0,1\}^{n \times m} \tag{1}$$

with $n \cdot m$ variables $x_{i,j}$. If the software component s_i is assigned to ECU p_j, then $x_{i,j} = 1$, else it is 0. The allocation

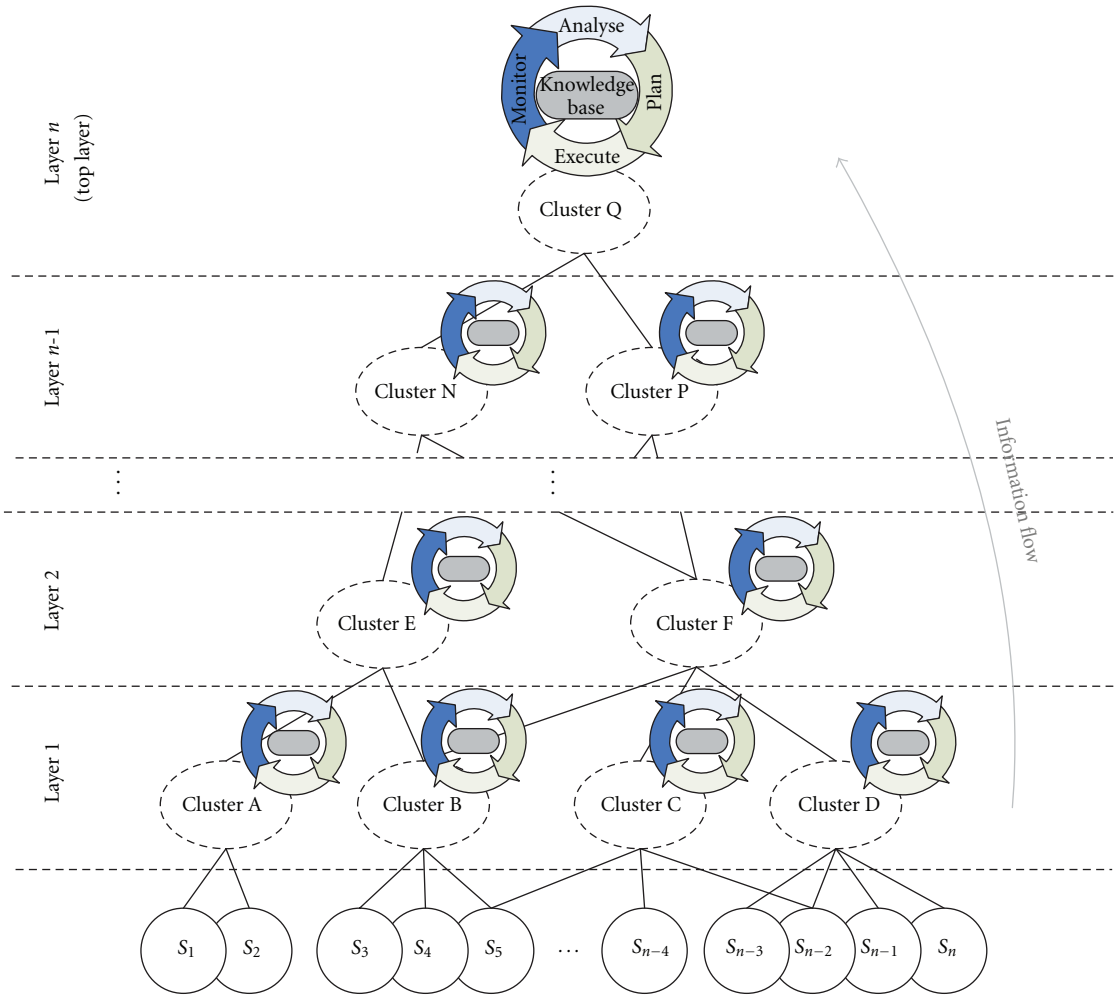


Figure 1: Hierarchical, multi-layered control approach.

of a specific software component $s_i \in \mathrm{SW}$ is represented by φ:

$$\varphi(s_i) = p_j, \quad p_j \in P. \tag{2}$$

Moreover, A consists of a set of constraints $\Psi = \{\psi_1, \ldots, \psi_p\}$ which represent the requirements of the system during runtime and enable the definition of valid allocations. Abstractly, a linear constraint $\psi_k \in \Psi$ is defined as a Boolean formula:

$$\psi_k = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} a_{i,j} x_{i,j} \right) \circ b \longrightarrow \{\text{true}, \text{false}\} \tag{3}$$

with the Boolean literals $x_{i,j} \in \{0, 1\}$ which represent the allocation of the software component s_i to the ECU p_j, the coefficients $a_{i,j}, b \in \mathbb{R}$, and the operator $\circ \in \{<, \leq, =, \geq, >\}$.

Typical constraints in terms of automotive systems concern hardware platform resources (e.g., volatile, non-volatile memory), network resources (e.g., bandwidth), task dependencies, task schedulability, timing, or the network topology. A detailed description of these equations, called system constraints, which define valid system configurations under real-time constraints, can be found in [13]. This set of equations is optimized to be solved efficiently during runtime in order to enable the computation of valid system configurations in an adaptive system in reasonable time.

3. Hierarchical, Multi-Layered Control Approach

To cope with the complexity of modern automotive embedded systems, we propose a hierarchical, multi-layered control approach (see Figure 1). Resources for enforcing adaptations at runtime are scarce but absolutely inevitable for realizing self-* properties. Therefore, a divide-and-conquer strategy can be applied during design which partitions the automotive embedded system into smaller entities—so-called clusters.

A cluster cl is defined by a set of r software components $\{s_1, \ldots, s_r\} = \mathrm{SW}_{\mathrm{cl}} \subseteq \mathrm{SW}$. The current cluster configuration $c_{\mathrm{cl}}(t)$ describes the current allocation φ_{cl} of the software components $\mathrm{SW}_{\mathrm{cl}}$ to the available ECUs at the time t according to (1).

Repeated partitioning of the automotive embedded system results in a hierarchy of clusters, representing the entire system. Thereby, each cluster has at least one parent cluster

and any number of child clusters. The top level of this hierarchy consists of exactly one cluster. Each cluster within the hierarchy is controlled by its own control loop, thus building a multi-layered control architecture. A control loop is an external component which is not included within the cluster itself. It is supervising and controlling (adapting) the clustered elements during runtime, so that the constraints Ψ are satisfied during runtime (see Section 5).

Due to the repeated segmentation of the automotive embedded system, the clusters on the lowest layers in this control architecture are relatively small. They consist of only a few software components. Due to the individual implementation of the control loops at the lowest layers, this hierarchical, multi-layered control approach can be tailored individually for the needs of specific functionalities or subsystems of the automotive embedded system (e.g., safety-critical X-by-wire systems). As a drawback, the clusters on the lower layers have a restricted local scope and are not always capable of determining a new, valid cluster configuration which satisfies all given requirements.

At the higher layers, the number of software components included in a cluster, which must be controlled, is growing. Thus, the possibilities of determining a valid cluster configuration increase but also the complexity of determining a valid configuration is rising with the number of software components which are considered.

At the top layer, a single root cluster represents the top element in the hierarchy. It has a global view of the system and supervises the entire automotive embedded system during runtime with all its predefined requirements.

During runtime, the automotive embedded system is supervised by a hierarchy of multiple MAPE-K cycles which adapt the system in response to changes within the system or in the system's environment. Based on an individual knowledge base, each MAPE-K cycle manages its cluster in four stages.

Monitor. Each control component within the control architecture collects, aggregates, and filters specific parameters of their managed elements (software components within the cluster). The parameters which need to be monitored are given by the requirements which can be supervised by the scope of the control loop.

Analysis. During this stage the monitored data are analyzed and compared to the available data in the knowledge base of the control loop. Each control instance evaluates if the predefined requirements, which it supervises, are currently satisfied.

Plan. The plan stage provides mechanisms that create or compose a set of actions to adapt the supervised part of the system. In the context of automotive embedded systems, the planning stage determines a new cluster configuration which fulfills all predefined requirements.

Execute. During this stage the planned actions are executed. In our case, each control loop adapts the allocation of the software components which are included in its cluster in the planned way.

Based on the hierarchy of MAPE-K cycles within our multi-layered control approach, it is possible to enhance the automotive embedded system by adaptation and self-* properties.

For building concrete instances of the hierarchical, multi-layered control approach to realize self-adaption within automotive embedded systems, different design options need to be addressed:

(i) number of layers within the control architecture,

(ii) number and scope of the different clusters within the hierarchy (resulting from the segmentation of the system),

(iii) realization of the coordination between multiple control loops.

In the next section, we address these issues and provide three different instances of the hierarchical, multi-layered control approach to realize self-adaptation in automotive embedded systems.

4. Instances of the Multi-Layered Control Approach for Automotive Embedded Systems

In this section, we present three concrete instances of the hierarchical, multi-layered control approach for automotive embedded systems.

The most intuitive approach is a topology-oriented structuring according to the network topology of today's in-vehicle networks (see Figure 2). For this purpose, the automotive embedded system—vehicle cluster on the top layer of the hierarchy—is divided into different clusters according to the physical sub-networks within the in-vehicle network which result from the segmentation of the car's functionalities into functional domains (e.g., power train, infotainment, chassis, comfort etc.). Thereby, five so-called Network Clusters are created which include all software components defined for a specific sub-network (domain). At the next layer of the hierarchy, each network cluster is split up according to the physical hardware platforms connected to the subnetwork. The resulting platform clusters represent the software components which are initially allocated to a specific ECU and control them by its own MAPE-K cycle. Since this control loop must be able to detect if the specific ECU is faulty and must adapt the allocation of the ECU's software components, it is implemented on one of the other ECUs within the sub-network.

Due to the increasing interconnection of the different vehicle domains (e.g., by advanced driver assistance systems), this kind of partitioning of automotive embedded systems may become less suitable for the realization of a multi-layered control architecture for automotive embedded systems. In other words, a local control loop responsible for a sub-network may not be able to find a local solution due to dependencies with other domains.

Another approach for a control architecture for automotive embedded systems is based on a function-oriented

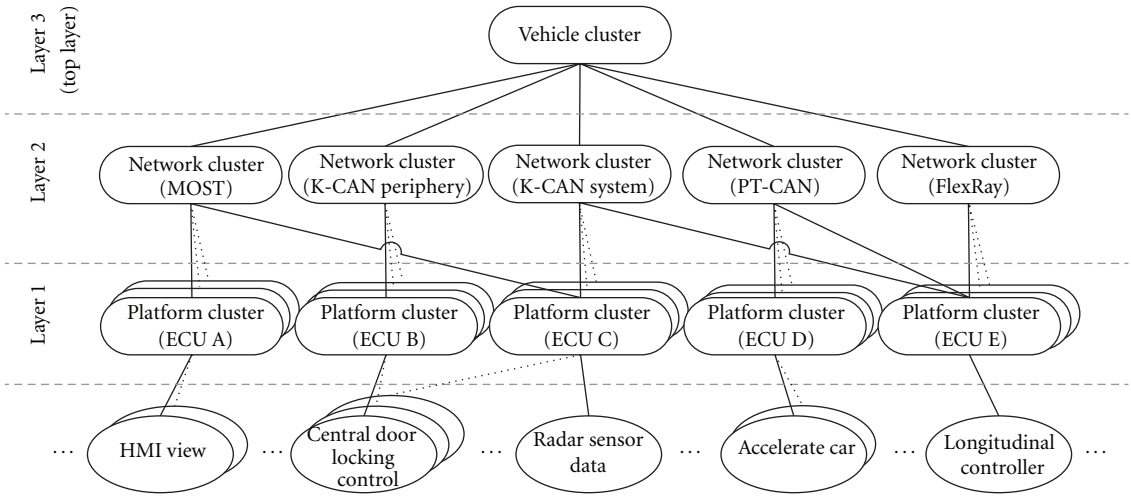


Figure 2: Multi-layered control architecture for automotive embedded systems (variant 1, topology oriented).

structuring of the in-vehicle network (cf. [14]). Thereby, the automotive embedded system is divided into five so-called safety clusters according to the safety integrity levels *(SIL) 0–4* defined by [15] or by the automotive safety integrity levels *(ASIL) A–D* as well as quality management *(QM)* defined by [16] which are used to classify the safety requirements of the car's functionalities (see Figure 3). Thus, functionalities with different safety requirements are monitored and controlled by specifically adjusted mechanisms. Safety-critical functionalities are reconfigured using adequate techniques which consider the functions' safety requirements, while other functionalities are adapted during runtime by other, more appropriate techniques. At the next layer of the hierarchy, each safety cluster is split up according to the functionalities included into the vehicle's domain. Each of the resulting feature clusters represents a functionality of the automobile which is realized by software and controls the functionalities' software components by its own MAPE-K cycle. This represents the logical structure of the automotive embedded system. Hence, interdependencies between different clusters due to interconnected sub-networks may be avoided.

A more fine-granular segmentation of the system is achieved by combing the two previous approaches (topology oriented and function oriented) for the realization of a concrete instance of the multi-layered control approach for automotive embedded systems (see Figure 4). Therefore, the automotive embedded system is firstly decomposed according to the different safety requirements of the car's functionalities. Hence, functionalities which are not safety-critical may be deactivated easily in certain situations (e.g., in case failures within the system during runtime). At the next layer, the system is partitioned according to the physical sub-networks within the in-vehicle network. Thereby, adaptations within a specific sub-network are realized by one specific MAPE-K cycle. The next layer is formed by feature clusters which represent each of the car's software-based functionalities. Since many functionalities of today's automobiles, like advanced driver assistance systems, realize their functionality using the same sensor input data or the same actuators, certain software functions which are used by different functionalities are clustered as so-called services (cf. [14]). Thus, in this variant of a multi-layered control architecture each functionality is decomposed into one or more services which are organized in so-called service clusters. The lowest layer of the hierarchy is composed of these clusters. Since these services are used by more than one of the car's functionalities, a service can be part of more than one Feature Cluster. For instance, the adaptive cruise control *(ACC)* feature, which can automatically adjust the car's speed to maintain a safe distance to the vehicle in front, can be decomposed into the radar sensor service, the longitudinal controller service, the engine service to accelerate, and the braking system service to decelerate.

In all these approaches, the so-called vehicle cluster forms the top layer of the hierarchical, multi-layered control architecture. This cluster includes all software components of the automotive embedded system and its MAPE-K cycle has a global view of the supervised system.

By using one of these instances of the hierarchical, multi-layered control approach, self-adaptation and different self-* properties can be realized while considering the specific nature of today's automotive embedded systems. Thereby, the various requirements of these systems are monitored by several control components (MAPE-K cycles) which are organized hierarchically. Each MAPE-K cycle adapts the specific part of the networked embedded system which it supervises.

In the following section, we describe how self-adaptation is realized in automotive embedded systems using the multi-layered control approach.

5. Self-Adaptation with Multi-Layered Control

In the following, we discuss the operation of multi-layered control for our setting with a global set of system constraints. The aim of our hierarchical, multi-layered control approach is to preserve all predefined requirements of the automotive embedded system during runtime. Thereby, the proper

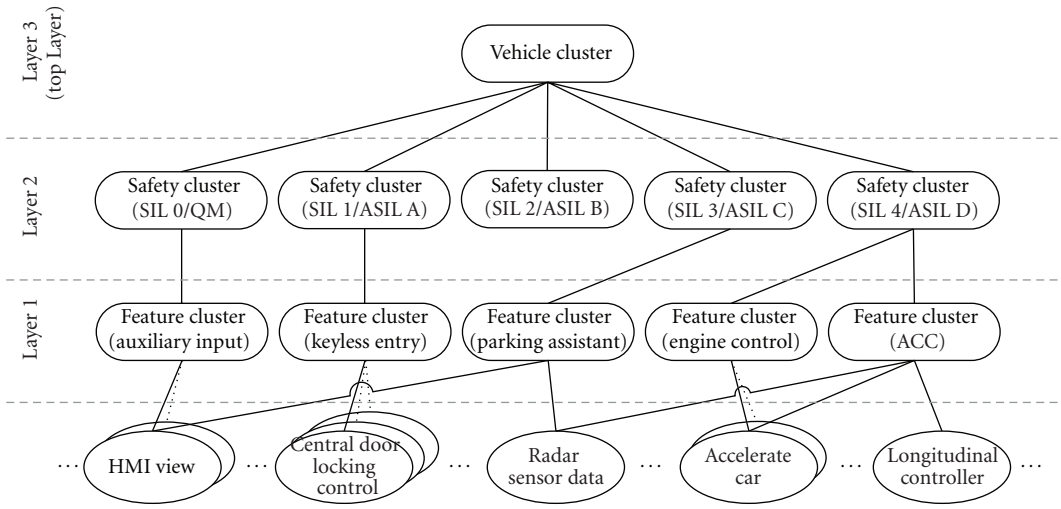


Figure 3: Multi-layered control architecture for automotive embedded systems (variant 2, function oriented).

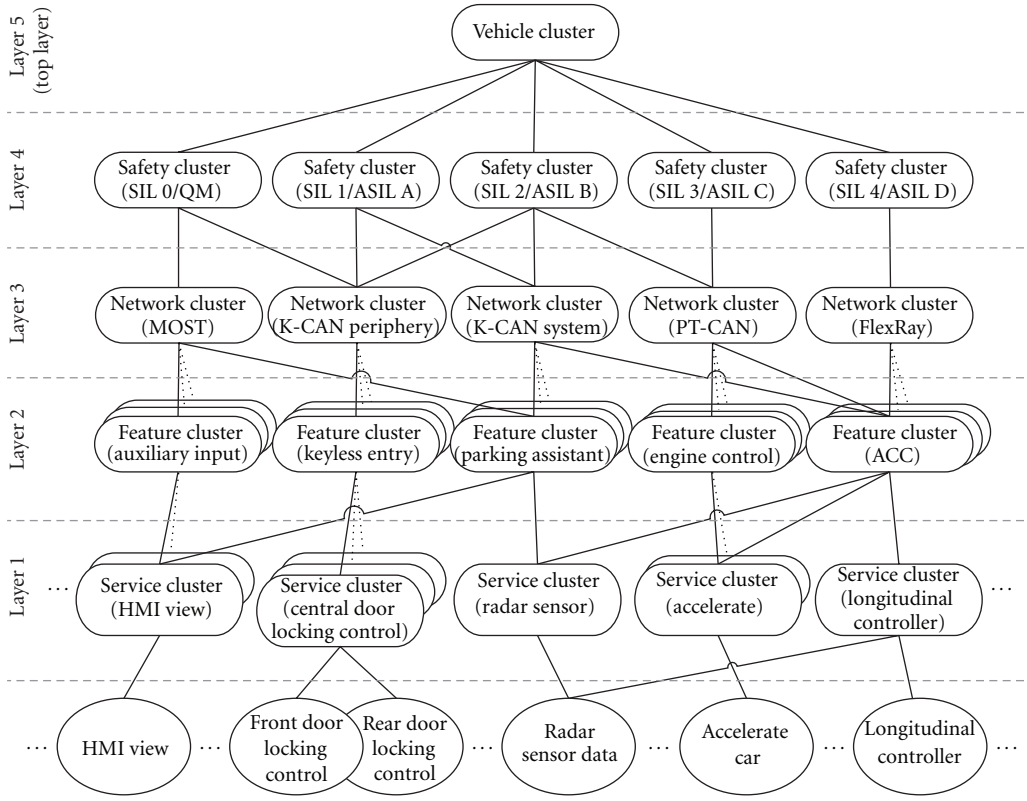


Figure 4: Multi-layered control architecture for automotive embedded systems (variant 3, combined approach).

system behavior of the automotive embedded system is guaranteed (see [4]). In our approach, each MAPE-K cycle within the hierarchy enforces all the requirements specified in the design, which are represented in form of linear, Boolean (in-) equations. If any of the predefined requirements are not satisfied anymore during runtime (e.g., caused by the breakdown of a hardware platform), the configuration of the system c is adapted to meet these constraints, if possible.

To enable a fast adaptation of the system during runtime, adaptations are performed on the lowest layer of the hierarchy by the cluster which detects the cause for the adaptation. Since all system constraints are considered by each control loop within the hierarchical, multi-layered control architecture, each MAPE-K cycle—even on the lowest layer of the hierarchy—is able to determine a new, valid system configuration. However, the different control

cycles have different scope—hence the set of free variables is restricted to the local scope. In other words, the local control cycles are in charge of the placement of the software components of their cluster.

Only if a cluster was not able to reach a new valid cluster configuration, its parent cluster takes over the adaptation process. The root cluster (Vehicle Cluster in Section 4) is only involved in the adaptation process as the last instance. Since the root cluster has the global view and knowledge of the entire system, it is always capable of reaching a new valid system configuration—if one exits.

In the best case, the adaptation of the system is performed by the control loop of one single cluster. In the worst case, up to n control loops are involved in the process of adaptation within an n-layered control architecture. Thereby, constraint checks may be performed multiple times until a new, valid system configuration is found. This results in unnecessary computation overhead—unless a solution is found on the lower layers. On the other hand, on the top level, new placements for all components are to be determined, which is much more complex than solving equations on a lower layer of the hierarchy. For the local scope, only a limited, local number of placements are to be determined. Furthermore, on the local scope, equations may be trivially true for a specific configuration, if they are not affected by the free variables of this control cycle.

In the following, we aim to compare the different instances of the hierarchical control approach with a central control solution. Thus, we have a tradeoff between finding solutions locally with lower cost, but possibly with repeated attempts on different layers, versus a single control loop with a higher cost. Clearly, if the hierarchical solution has to resort to the top layer frequently, the centralized solution will be more efficient.

Control loops at the same layer of the control architecture shall not interfere with each other in order to avoid oscillation during adaptation—hence only one of them can be active at a time. In other words, they must be coordinated by a dedicated mechanism to avoid that several control components adapt the same part of the system simultaneously (see [17]). This needs to be ensured, typically by the next layer (see Section 5.1).

5.1. Coordination within Multi-Layered Control. The coordination of different control loops within our approach is based on the design pattern "hierarchical control" described in [17, 18], which is adopted for our specific application scenario.

To avoid that multiple control loops adapt the system simultaneously, an MAPE-K cycle notifies the control component of its parent cluster that it will adapt the system. If none of the other control loops in the same scope of the cluster is already adapting the automotive embedded systems, the request is permitted (see Figure 5). Otherwise the request is rejected. While the system is reconfigured, further requests for adaption are declined by the control component which executes the adaption or by the control components which has allowed the adaptation. Therefore, it is guaranteed that the system configuration of the automotive embedded systems (or a specific subsystem) is not adapted simultaneously by multiple control components within the control architecture. Hence, oscillations during adaptation are avoided by this coordination approach. Nevertheless, different subsystems can be adapted in parallel (e.g., adaptations within different physical sub-networks of the networked embedded system).

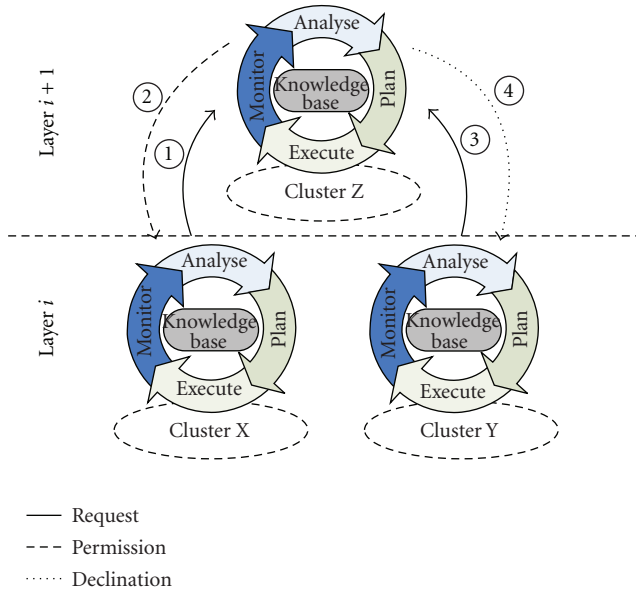


Figure 5: Coordination of control loops within the hierarchical, multi-layered control approach.

This approach allows the coordination of multiple MAPE-K cycles within the hierarchical, multi-layered control architecture during runtime adaption with minimal communication overhead. Only two messages are exchanged to coordinate two control loops in case of runtime adaptation. During the "normal" operation of the system (monitoring and analysis of the correctly working networked embedded system), no coordination is required and hence no additional communication between different control loops within the multi-layered control architecture is performed.

6. Evaluation

In this section, we illustrate the potential benefits of our approach of hierarchical, multi-layered control for realizing self-adaptation in automotive embedded systems (see Section 4) w.r.t. efficiency of determining a new, valid system configuration in case of an ECU breakdown (self-healing scenario). At first, an initial assignment of software components to hardware is determined. After this, the adaptation of the system is triggered by simulating the breakdown of a randomly selected ECU. Moreover, we simulate the simultaneous breakdown of two randomly selected ECUs in a second self-healing scenario.

6.1. Evaluation Setup. In our experiments, we simulate the typical embedded systems of modern automobiles (see [5]). Therefore, we use various setups representing different sizes

Table 1: Experimental setups.

Setup	ECUs	Functions	Sensors/actuators	Subnetworks
1.1	20	500	40/40	4
1.2	30	750	50/50	4
1.3	40	1000	60/60	5
1.4	50	1250	70/70	5
1.5	60	1500	80/80	8
1.6	70	1750	90/90	8
1.7	80	2000	100/100	8
1.8	90	2250	110/110	8
1.9	100	2500	120/120	8
2.1	40	200	60/60	5
2.2	40	400	60/60	5
2.3	40	600	60/60	5
2.4	40	800	60/60	5
2.5	40	1000	60/60	5
2.6	40	1200	60/60	5
2.7	40	1400	60/60	5
2.8	40	1600	60/60	5
2.9	40	1800	60/60	5
2.10	40	2000	60/60	5

Table 2: Composition of network buses for different setups of automotive in-vehicle networks.

Number of networks	Composition
4	2x low speed CAN, 2x high speed CAN
5	3x low speed CAN, 2x high speed CAN
8	3x low speed CAN, 3x high speed CAN, 1x MOST, 1x FlexRay

and variants of automotive in-vehicle networks (see Table 1). While the ratio between the number of software components and the number of ECUs are fixed in the first setups (setup 1.1–1.9), we also perform experiments with setups where this ratio is variable (setup 2.1–2.10).

In all our experiments, the in-vehicle network consists of a central gateway (maximum throughput = 250 Kbyte/s, maximum delay = 100 ms) which interconnects all network buses. The network buses are either low speed or high speed CAN, MOST, or FlexRay systems. The parameters of these networks (bandwidth, maximum transmission delay, maximum message size) are introduced in [19]. The different configurations of the network buses in our experiments are listed in Table 2. An ECU is always connected to exactly one network bus. The number of ECUs is equally spread over the available network buses. Moreover, sensors and actuators are also directly connected to one of the network buses (cf. [20]). The connection of an ECU, a sensor, or an actuator to a certain network bus is done for each setup randomly.

An ECU is defined by volatile memory (RAM) with 32, 64, 128, 256, or 512 Kbyte as well as nonvolatile memory (ROM, Flash) of 64, 128, 256, 512, or 1024 Kbyte. Moreover, a certain CPU frequency is assigned to each ECU, which is between 50 MHz and 1 GHz (in steps of 50 MHz). The value Δ is used to specify the ratio of the CPU frequency to a reference frequency of 100 MHz (Δ = reference frequency/CPU frequency).

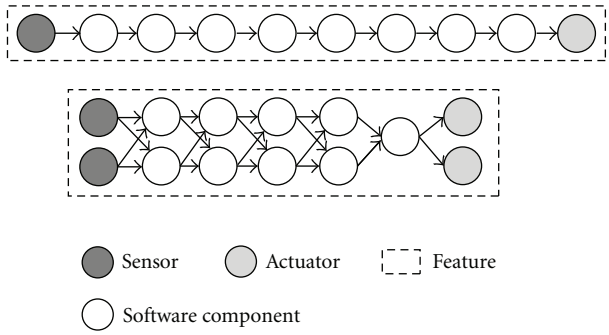


Figure 6: Example of a feature which is generated for our experiments.

The sensors in our generated in-vehicle networks provide data always periodically. These periods can vary for each sensor in a range between 20 ms and 200 ms (in steps of 5 ms). Actuators react directly on incoming events or messages and hence are aperiodic.

All parameters of the physical resources of the automotive in-vehicle network for our experiments are generated randomly and are equally distributed in the given ranges.

The function network $G_f(V_f, E_f)$ of the generated automotive embedded systems consists of a certain number of features. In all setups listed in Table 1, we assume that each feature of the vehicle consists of 8-9 software functions, as well as one or two sensors and actuators. Each of the software functions interacts with either one or two of the other functions which implement the same feature. We assume that there are no cycles within the function network. Figure 6 illustrates how a generated feature may look alike. Between a sensor and a software function as well as between a software function and an actuator, data in a range between 1 and 10 Bit are exchanged. Between different software functions, which implement a certain feature of the automotive system, data in a range between 2 and 20 Bit are exchanged each. Thereby, software functions, sensors, and actuators can be part of any number of features, but are at least part of one feature. All features are generated automatically in our evaluations, where all parameters are chosen randomly and equally distributed.

Each software function needs a certain amount of volatile and nonvolatile memory in order to be executed on a hardware platform. These values are in a range between total amount of the available memory of all ECUs/(number of software functions $*$ 2) and total amount of the available memory of all ECUs/number of software functions in all experiments. All software functions are periodic tasks. The period of a software function is between 20 ms and 200 ms (in steps of 5 ms). The deadline of a function is equal to its period. Furthermore, the worst case execution time (WCET) of a software function is calculated according to the *UUniFast algorithm* [21]. The value of the WCET refers to an ECU with the reference frequency of 100 MHz. Moreover, a priority is defined for each software function, which is

Table 3: Size of the set of system constraints.

Setup	No. of constraints	No. of literals
1.1	7,599	60,877
1.2	20,047	197,385
1.3	33,040	397,584
1.4	52,658	781,281
1.5	65,773	1,154,150
1.6	85,081	1,689,251
1.7	126,877	2,771,502
1.8	138,641	3,393,508
1.9	174,434	4,580,388
2.1	6,746	77,396
2.2	12,801	156,798
2.3	17,220	217,867
2.4	27,899	338,935
2.5	33,040	397,584
2.6	35,516	435,196
2.7	45,021	539,373
2.8	54,239	659,452
2.9	60,376	770,882
2.10	69,218	877,051

between 1 and number of software functions/number of ECUs. The parameters of each software function are also generated randomly and equally distributed in the given range of values.

Moreover, timing requirements in form of an end-to-end timing chain are defined consisting of one sensor input, a sequence of software functions, which are part of a certain feature, and one actuator. In the following, we assume that half of the features have a timing chain defined.

In all our experiments, the automotive embedded system consists of three different kinds of hardware platforms (ECUs). Each software function can only be executed on one kind of the hardware platforms. The definition on which kind of hardware platform a certain software function can be executed is done randomly. Thus, the allocation set for each software function/ECU contains about 33% of the vehicles' ECUs/software functions in all our evaluation setups. Other limitations of the automotive in-vehicle network, which must be considered during runtime, are expressed by the system constraints, as introduced in [13].

For reference, we list the number of these constraints and literals needed to solve the SAT problem for the given setups in Table 3.

For our experiments, the concrete control architecture variants for automotive embedded systems outlined in Section 4 are implemented. Furthermore, a centralized control architecture (similar to [22]) is implemented to compare our hierarchical, multi-layered approach with this state-of-the-art approach. According to [13] the SAT-solver SAT4J Version 2.2 [23] is used in all our experiments to determine a valid allocation of software components to ECUs within each MAPE-K cycle.

For each setup in Table 1, we perform tests with each variant for the control architecture 10 times and calculate the average values as well as the 95% confidence interval of the average values. All experiments are performed on an embedded platform with an Intel Atom Processor at 1.6 GHz and 1.5 Gbyte RAM (reference platform for the next generation in-vehicle infotainment systems).

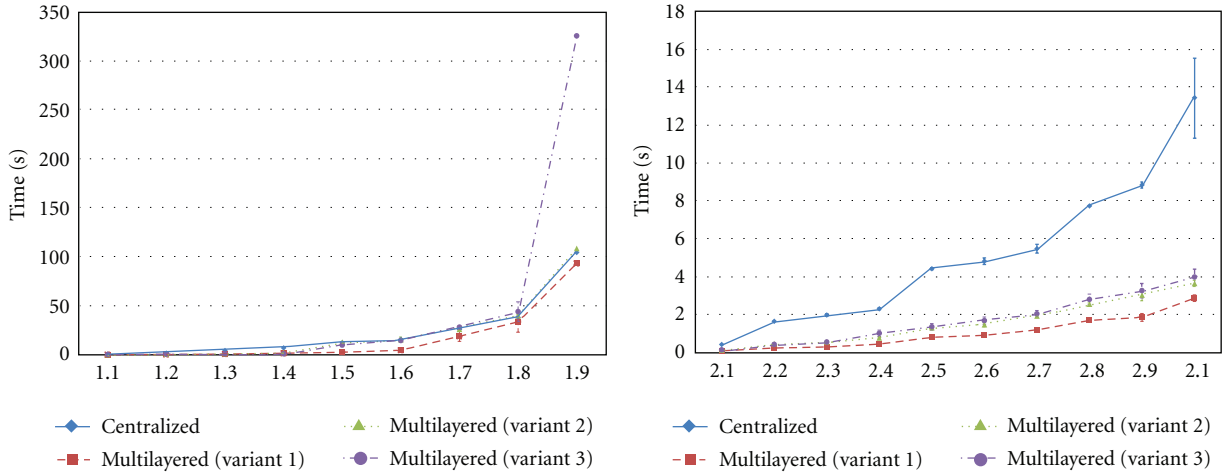
6.2. Breakdown of an ECU. Figures 7(a) and 7(b) show the results of the experiments which are performed with the previously described variants of automotive embedded systems (see Table 1). In all experiments, we have measured the time needed to determine a new, valid allocation of software components after the breakdown of a randomly selected ECU occurred (self-healing scenario).

This result shows that when using one of the instances of the hierarchical, multi-layered control approach described in Section 4, less computation time is needed until a new valid system configuration is found, than using a centralized control approach with one single control loop. For the setups 1.*x*, the difference between the centralized approach and the variant 2 (function oriented) of our approach (see Figure 3) is quite small, especially for larger automotive in-vehicle networks. But using the variant 1 (topology oriented) of the hierarchical, multi-layered control architecture (see Figure 2), a significant performance optimization is reached. Moreover, using the combination of variant 2 and 3 (variant 3, see Figure 4), a new valid system configurations can be determined for small automotive embedded systems (setup 1.1–1.4) in less time than using a centralized control approach. However, with the growing size of the allocation problem which needs to be solved (size of the automotive embedded system), this variant of a hierarchical, multi-layered control architecture needs approximately the same time to determine a new valid system configuration. For large automotive embedded systems (setup 1.7–1.9) this variant needs even more time. For the setups 2.*x*, significant optimization of the time needed to determine a new, valid system configuration is reached by using our hierarchical, multi-layered approach for controlling the automotive embedded systems. The difference between the three instances of the multi-layered control approach is quite small. Yet the topology-oriented variant is performing better in all our experiments, although adaptation is performed for setup 2.*x* on the lowest layer of the hierarchy in all different instances of our multi-layered control approach.

The results of our experiments clearly show that the complexity of realizing self-adaptation in automotive embedded systems can be reduced significantly by using our hierarchical, multi-layered control approach. Furthermore, we also see that the topology-oriented variant is significantly better in most cases than variant 2, which clusters along the functional entities, or variant 3. This means in case of an ECU failure, the variant which considers solutions based on the local network has better chances to determine a new allocation on the lowest layer of the hierarchy than the second variant, which considers solutions based on functionality. While solutions based on functionality are more natural regarding the functional dependencies, there is the risk that a CPU

Table 4: Successful determination of valid configurations within the multi-layered control architectures (in %).

Setup	Variant 1			Variant 2			Variant 3				
	Layer 1	Layer 2	Layer 3	Layer 1	Layer 2	Layer 3	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5
1.1	100	0	0	100	0	0	100	0	0	0	0
1.2	100	0	0	100	0	0	100	0	0	0	0
1.3	100	0	0	100	0	0	100	0	0	0	0
1.4	100	0	0	100	0	0	100	0	0	0	0
1.5	0	100	0	0	100	0	10	0	90	0	0
1.6	0	88.9	11.1	10	90	0	0	0	100	0	0
1.7	0	30	70	0	100	0	0	0	100	0	0
1.8	0	28.6	71.4	0	100	0	0	0	90	10	0
1.9	0	0	100	0	100	0	0	0	100	0	0



(a) Setup 1.*x*

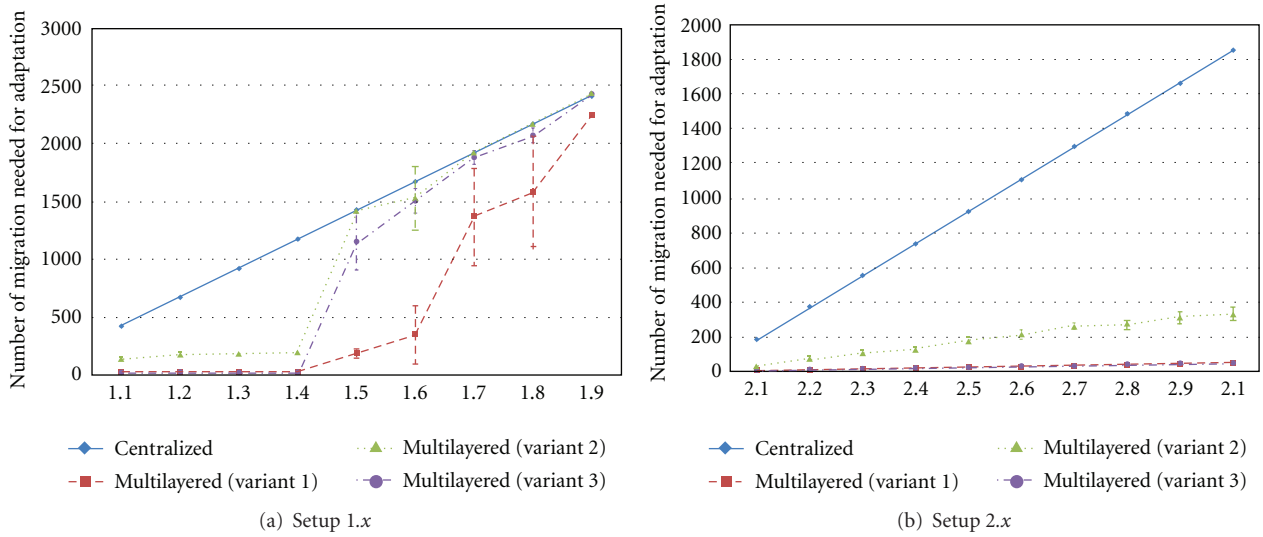
(b) Setup 2.*x*

Figure 7: Time needed to determine a new, valid system configuration.

failure may affect several functional categories—hence a local repair by one control cycle is not possible (cf. Table 4).

This is also supported by the different results for the setup 1.*x* and 2.*x*. Here the topology-oriented variant of the multi-layered control architecture is significantly better compared to function-oriented variant or the combination of variant 1 and 2 for setups 2.*x*, while all three are closer in most experiments in setup 1.*x* (except 1.5–1.7). Furthermore, all instances of our hierarchical, multi-layered control approach are significantly better than the centralized approach for setup 2.*x*. Our multi-layered approach performs also much better than the centralized one for smaller automotive embedded systems (setup 1.1–1.4). In case of larger and more complex allocation problems (setup 1.5–1.9), only variant 1 of a hierarchical, multi-layered control architecture enables the calculation of a valid allocation in shorter time. Variant 2 (function-oriented) as well as variant 3 (combination of variant 1 and 2) of the layered control approach do not perform better in average than the central control architecture for these setups. This is because, for large allocation problems with many constraints (cf. Table 3), it is very difficult to find a new valid allocation with the local scope of the control loops on the lower layer of a multi-layered control architecture (cf. Table 4). The chance to find a new, valid system configuration locally on the lowest layer of the hierarchically does not depend on the number of software components with are supervised by control loop (cf. results for setup 2.*x* in Figure 7(b)).

Furthermore, we measured the distance between the current and the newly determined system configuration (lower bound on the number of software component migrations needed to adapt the automotive embedded system) for rating the quality of the solutions. The results (Figures 8(a) and 8(b)) clearly show that using a hierarchical, multi-layered control architecture leads to solutions with a shorter distance to the current allocation when determining a new, valid allocation of software functions to ECUs using a SAT-solver-based approach. Especially, if it is possible to find a new, valid system configuration by a control loop on the lowest layer of the hierarchy (in our experiments for setup 1.1–1.4 and 2.*x*), a significant reduction of necessary migrations can be achieved. For larger automotive embedded systems (setup 1.5–1.9 in our experiments), a valid allocation of software functions to ECUs must often be determined by a control



(a) Setup 1.*x* (b) Setup 2.*x*

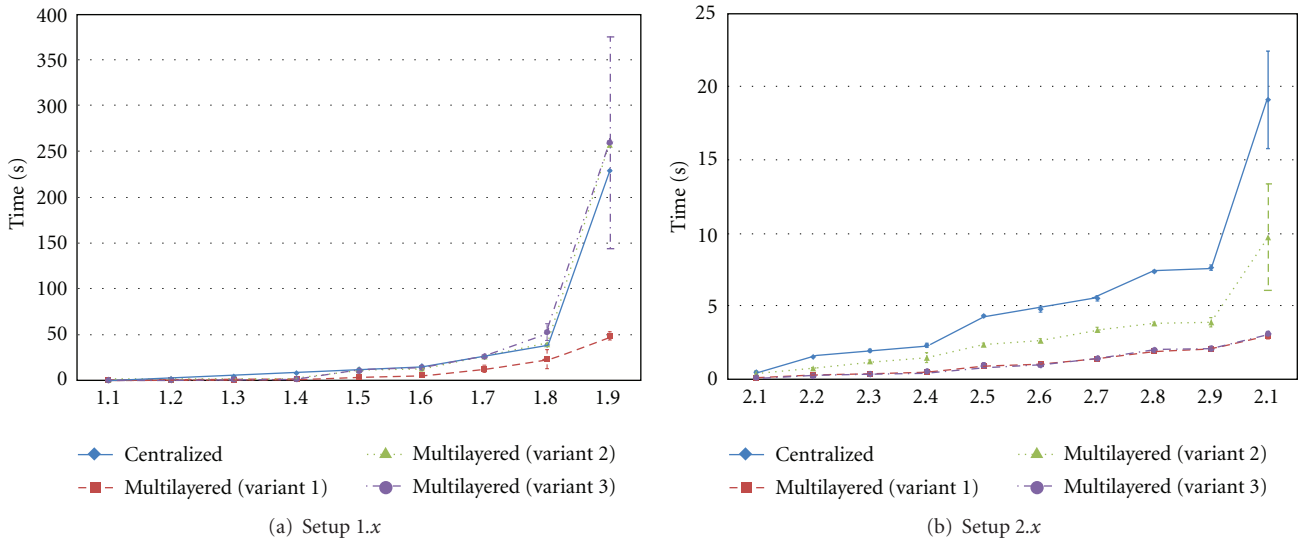
Figure 8: Distance between the current and the new system configuration.

loop on a higher layer of the hierarchy, because it is not possible to find a valid solution for the allocation problem within the local scope of one of the control instances at the lowest layer (cf. Table 4). In these tests, variant 1 of our hierarchical, multi-layered control approach performs much better than the other two variants described in Section 4 (see Figure 8(a)). Moreover, our experiments for setup 2.*x* show no significant difference in the results using variant 1 and variant 3 of our hierarchical, multi-layered control approach with respect to the quality of the newly determined system configuration. However, these two variants result in much better solutions for setup 2.*x* than the function-oriented variant of a hierarchical, multi-layered control architecture.

Thus, not only the needed time to determine a new, valid system configuration during the planning stage, but also time needed to execute the newly planned configuration is reduced significantly when using the hierarchical, multi-layered control approach in this self-healing scenario.

6.3. Simultaneous Breakdown of Two ECUs. In additional tests, we evaluate the different instances of the hierarchical, multi-layered control approach for self-adaptation in automotive embedded systems using a self-healing scenario, in which two ECUs break down simultaneously. Thereby, all experiments are performed with the previously described variants of automotive embedded systems (see Table 1). Again we measured the time needed to determine a new, valid allocation of software components after the breakdown occurred. The results of these experiments are presented in Figures 9(a) and 9(b).

The results of these tests for setup 1.*x* show, that only using the topology-oriented variant of our hierarchical multi-layered control approach (variant 1) leads to a better performance for the self-adaptation in all experiments than the central control approach. Both the function-oriented approach and the combined approach based on variant 1 and variant 2 are able to determine a new valid system configuration for small automotive embedded systems (setup 1.1–1.5) more quickly than the centralized control approach, but in case of larger allocation problems (setup 1.7–1.9) these approaches need more time to determine a valid allocation of software components than the central control architecture. Reason for this is that several different functionalities are affected by the breakdown of two ECU when using the variant 2 or 3 of a multi-layered control architecture (cf. Section 4). In these cases a valid system configuration is normally found by one of the control loops on the higher layers of the hierarchy (cf. Table 5). Thereby, multiple MAPE-K cycles are involved in the adaptation process which includes the repeated execution of the necessary calculations (e.g., for determining a new, valid system configuration). Hence, using a hierarchical, multi-layered control architecture results in a more complex adaptation process compared to the centralized control approach, if multiple control loops on different layers of the multi-layered control architecture are triggered during the adaptation process. When using the topology-oriented approach of the hierarchical, multi-layered control approach, the top layer of the hierarchy is only involved in the adaptation process, if more than one subnetwork is affected by the breakdown of the two ECUs, which is not always the case in our experiments. Thus, the adaptation can be planned more quickly using the topology-oriented variant of our hierarchical, multi-layered control approach than using one of the other variants evaluated in our experiments (see Figure 9(a)). The results for setup 2.*x* show, that a significant optimization of the time needed to determine a new, valid system configuration is achieved by using our hierarchical, multi-layered control approach compared to a central control architecture (see Figure 9(b)). Moreover, these results illustrat, that variant 1 and variant 3 of a hierarchical, multi-layered control architecture for self-adaptation in automotive embedded systems (presented in Section 4) are more suitable than variant 2 in case that



(a) Setup 1.x (b) Setup 2.x

Figure 9: Time needed to determine a new, valid system configuration.

Table 5: Successful determination of valid configurations within the multi-layered control architectures (in %).

Setup	Variant 1			Variant 2			Variant 3				
	Layer 1	Layer 2	Layer 3	Layer 1	Layer 2	Layer 3	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5
1.1	100	0	0	100	0	0	100	0	0	0	0
1.2	100	0	0	100	0	0	100	0	0	0	0
1.3	100	0	0	100	0	0	100	0	0	0	0
1.4	100	0	0	100	0	0	100	0	0	0	0
1.5	10	80	10	20	80	0	0	20	80	0	0
1.6	20	70	10	30	70	0	10	0	90	0	0
1.7	0	70	30	0	100	0	0	0	100	0	0
1.8	0	66.6	33.3	0	100	0	0	0	100	0	0
1.9	0	80	20	0	100	0	0	0	100	0	0

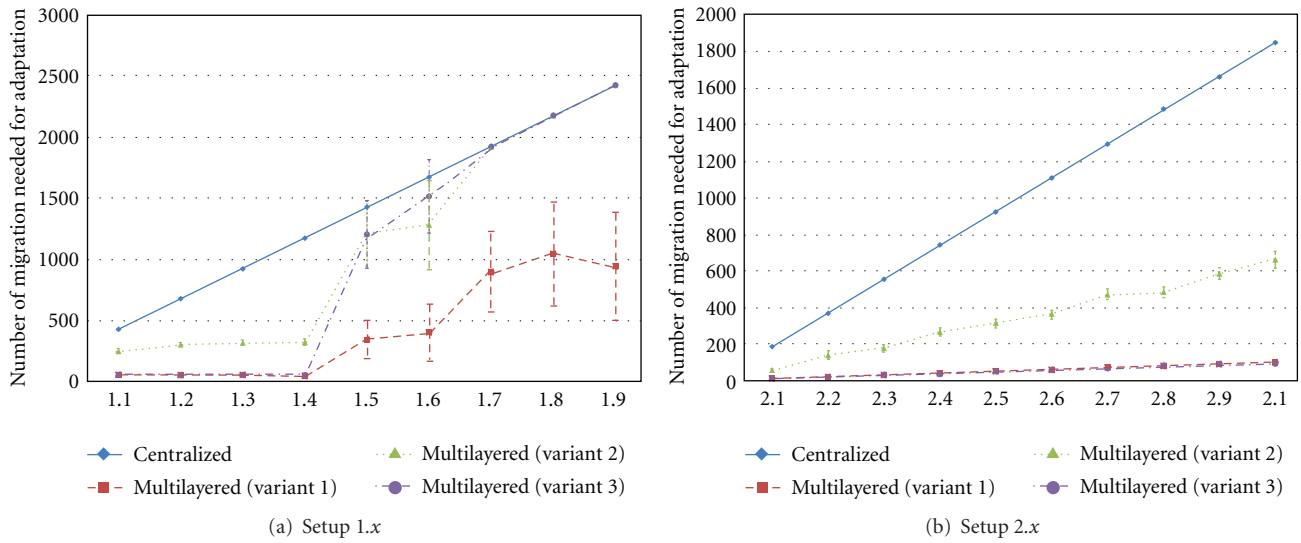
two control units break down simultaneously and all repair actions are performed locally on the lowest layer of the hierarchy.

Furthermore, the comparison of the quality of the solutions (see Figures 10(a) and 10(b)) shows that using our hierarchical, multi-layered control approach leads to solutions with a shorter distance to the current allocation when determining a new valid allocation of software functions to ECUs. Variant 2 and variant 3 of a hierarchical, multi-layered control architecture (presented in Section 4) lead to significant improvements w.r.t. the quality of the solution only for small automotive embedded systems (setup 1.1–1.7). However, the topology-oriented variant (variant 1) results in better solutions w.r.t. the distance between the current and the newly determined system configuration even for very large automotive embedded systems (see Figure 10(a)). Our experiments for setup 2.x also show, that the concrete instances of the hierarchical, multi-layered control approach in combination with a SAT-solver-based approach for the determination of a valid allocation for the systems' software components results in much better solutions compared to the centralized control approach with one single MAPE-K cycle. The best results for this scenario (in which two ECUs breakdown simultaneously) are achieved when using either variant 1 or variant 3 of a hierarchical, multi-layered control architecture (see Figure 10(b)).

7. Related Work

In the past, various approaches for realization of a multi-layered control architecture have been presented.

Kramer and Magee outline a three-layered architecture for self-managed systems in [24]. On the lowest layer (component layer), a control loop provides self-optimization algorithms as well as mechanisms for event and status reporting to higher layers and operations to support the modification of supervised components. On the next layer (change management layer), a set of predefined configurations is activated in response to state changes of the system below. Finally, on the goal management layer, time



(a) Setup 1.*x* (b) Setup 2.*x*

Figure 10: Distance between the current and the new system configuration.

consuming computations such as planning are performed to achieve the defined system goals. This three-layered control approach poses a generic reference architecture. The level of abstraction proposed by [24] reduces the complexity of self-management during runtime, but does not address the various functional and nonfunctional requirements of automotive embedded systems. To address this issue, we propose a divide-and-conquer strategy to split up the technical system into clusters according to different criteria.

In [25] a control architecture for embedded real-time systems is outlined. Simple module managers are running on each processing node of the system and one single global manager controls the whole system. The module managers collect information about the current condition of the system from their local point of view and transfer it to the global manager. The global manager analyzes the available information and decides which actions should be executed next. In this approach, the Module Managers do not need to implement the complete MAPE-K cycle and thus can run even on small processing nodes. The complex analysis and planning actions are performed by the Global Manager. Although this approach is well suited for the use within embedded systems, it consists of a relatively complex Global Manager component which still poses a single point of failure. Since the Global Manager is the only component to make decisions, its failure cannot be compensated. Within our hierarchical, multi-layered control approach, adaptations are performed by the control component which detects the cause for the adaptation. Only if it is not possible to determine a new, valid system configuration locally, a control loop at the next higher layer is involved in the adaptation process. Moreover, the multiple layers of our approach are able to consider the different aspects of automotive embedded system in contrast to the two-layered approach.

A first approach towards a control architecture for the realization of self-management in automotive embedded systems is presented in [22]. He describes a central control architecture for enabling autonomous management. The so-called system manager provides management functionalities on a system-wide basis. The Self-configurator uses these management functionalities to implement a control loop with a global knowledge base. The central control approach is not able to manage the growing complexity of today's automotive software systems in an adequate way. Our experiments show that our multi-layered approach is performing significantly better in determining a new, valid allocation of software components to ECUs (in case of a self-healing scenario) compared to the central approach (cf. Section 6). Moreover, the central control architecture results in a single point of failure which cannot be tolerated in safety-relevant systems like automobiles.

The concept of multiple control loops which cooperate to achieve a goal was already introduced by IBM [11]. To reduce the complexity of self-adaptation or self-management, multiple control loops must be decoupled with respect to time or respect to space and may be hierarchically organized [10], thereby, distinguishing between different time scales and different controlled variables (e.g., [26, 27]). Our hierarchical, multi-layered control approach is also based on the concepts of decoupling control loops by different controlled variables (software components) and organizing these control loops hierarchically. Thereby, it is possible to handle the complexity of self-adaptation in a networked embedded system which must fulfill various requirements with different scopes (e.g., an automotive in-vehicle network). Such a multi-layered control architecture for automotive embedded systems, where each control loop is responsible to supervise certain requirements and to adapt a certain part of the system, is introduced in [28]. Thereby, adaptation is performed locally on a lower layer first. If this fails, the next higher layer is in charge of finding a suitable adaptation. Yet in this approach no realistic set of system constraints, as done here, is used which enables the hierarchical enforcement of system requirements on several layers.

8. Conclusion

In this work, we have presented a hierarchical, multi-layered control approach for self-adaptation in automotive systems. We have introduced a multi-layer control approach which aims to cope with the rising complexity of today's automotive embedded systems by splitting up the system into a set of different clusters. By the segmentation of the system according to different aspects of the automotive system (locality or functionality), a hierarchy of these clusters is built. Based on such a hierarchical, multi-layered control architecture including multiple MAPE-K cycles, self-adaptation is realized during runtime by enforcing the systems' requirements.

We have compared different concrete instances of our hierarchical, multi-layered control approach in a self-healing scenario with realistic setups of up to 100 ECUs. This comparison shows that local repair based on a layered control architecture in such a system is more efficient than a pure centralized control approach. Not only w.r.t. the time needed to determine a new, valid system configuration after the breakdown of one or two ECUs, but also w.r.t. the quality of the newly determined solutions (the distance between the current and the newly determined system configuration). Moreover, it shows that a responsibility split based on locality with respect to network topology performs better than a split regarding functional areas in our self-healing scenario.

Thus, not only the needed time to determine a new, valid system configuration during the planning stage, but also the time needed to execute the newly planned configuration is reduced significantly when using our hierarchical, multi-layered control approach for self-adaptation in automotive embedded system in a self-healing scenario.

Furthermore, the multi-layered control approach presented in this work is adaptable to enhance other complex and networked embedded systems with real-time requirements (e.g., industrial plants, aircrafts or railways) with self-adaptation and self-* properties. Therefore, such networked embedded system must be split up into a set of different clusters based on aspects which are specific to the respective application domain. Moreover, realizing other self-* properties apart from self-healing (e.g., self-optimization) will be addressed in future work. In this context, a fully detailed discussion of local and global optima which are reached by using different control approaches for planning new system configurations in networked embedded systems as well as a comprehensive mathematical representation of the allocation problem, which needs to be solved to determine a new allocation of software components to hardware platforms, must be presented.

References

[1] J. Kramer and J. Magee, "Dynamic configuration for distributed systems," *IEEE Transactions on Software Engineering*, vol. 11, no. 4, pp. 424–436, 1985.

[2] P. Oreizy, N. Medvidovic, and R. N. Taylor, "Architecture-based runtime software evolution," in *Proceedings of the International Conference on Software Engineering*, pp. 177–186, April 1998.

[3] I. Georgiadis, J. Magee, and J. Kramer, "Self-Organising Software Architectures for Distributed Systems," in *Proceedings of the 1st Workshop on Self-Healing Systems (WOSS '02)*, pp. 33–38, November 2002.

[4] M. Zeller, G. Weiss, D. Eilers, and R. Knorr, "An approach for providing dependable self-adaptation in distributed embedded systems," in *Proceedings of the 26th Annual ACM Symposium on Applied Computing (SAC '11)*, pp. 236–237, March 2011.

[5] K. V. Prasad, M. Broy, and I. Krüger, "Scanning advances in aerospace automobile software technology," *Proceedings of the IEEE*, vol. 98, no. 4, pp. 510–514, 2010.

[6] G. Weiss, M. Zeller, D. Eilers, and R. Knorr, "Towards self-organization in automotive embedded systems," in *Proceedings of the 6th International Conference on Autonomic and Trusted Computing*, pp. 32–46, 2009.

[7] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[8] G. Weiss, M. Zeller, and D. Eilers, "Towards automotive embedded systems with self-x properties," in *New Trends and Developments in Automotive System Engineering*, InTech, 2011.

[9] G. Mühl, M. Werner, M. A. Jaeger, K. Herrmann, and H. Parzyjegla, "On the definitions of self-managing and self-organizing systems," in *Proceedings of the KiVS Workshopon Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme*, 2007.

[10] Y. Brun, G. Di Marzo Serugendo, C. Gacek et al., "Engineering self-adaptive systems through feedback loops," *Software Engineering for Self-Adaptive*, pp. 48–70, 2009.

[11] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," 2001.

[12] B. H. Cheng, R. Lemos, H. Giese et al., "Software engineering for self-adaptive systems: a research roadmap," in *Software Engineering for Self-Adaptive Systems*, B. H. Cheng, R. Lemos, h. Giese, P. Inverardi, and J. Magee, Eds., pp. 1–26, Springer, Berlin, Heidelberg, 2009.

[13] M. Zeller, C. Prehofer, G. Weiss, D. Eilers, and R. Knorr, "Towards self-adaptation in real-time, networked systems: efficient solving of system constraints for automotive embedded systems," in *Proceedings of the 5th IEEE Int. Conference on Self-Adaptive and Self-Organizing Systems*, pp. 79–88, 2011.

[14] M. Broy, G. Reichart, and L. Rothhardt, "Architekturen softwarebasierter Funktionen im Fahrzeug: von den Anforderungen zur Umsetzung," *Informatik- Spektrum*, vol. 34, pp. 42–59, 2011.

[15] ICE, "IEC/DIN EN 61508: Functional safety of Electrical/Electronic/Programmable Electronic (E/E/PE) safety related systems: overview," International Electrotechnical Commission (IEC), 1998.

[16] ISO, "ISO/WD 26262: Road Vehicles—Functional Safety," International Organization for Standardization (ISO), 2005.

[17] R. de Lemos, H. Giese, H. Müller et al., "Software engineering for self-adpaptive systems: a second research roadmap," in *Software Engineering for Self-Adaptive Systems*, vol. 10431 of *Dagstuhl Seminar Proceedings*, 2011.

[18] D. Weyns, B. Schmerl, V. Grassi et al., "On patterns for decentralized control in self-adaptive systems," in *Software Engineering for Self-Adaptive Systems 2*, Springer, Berlin, Heidelberg, 2012.

[19] B. Hardung, *Optimisation of the allocation of functions in vehicle networks [Ph.D. thesis]*, Universität Erlangen-Nürnberg, 2006.

[20] K. Klobedanz, G. B. Defo, W. Mueller, and T. Kerstan, "Distributed coordination of task migration for fault-tolerant

FlexRay networks," in *Proceedings of the 5th International Symposium on Industrial Embedded Systems (SIES '10)*, pp. 79–87, July 2010.

[21] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[22] M. Dinkel and U. Baumgarten, "Self-configuration of vehicle systems—algorithms and Simulation," in *Proceedings of the 4th International Workshop on Intelligent Transportation*, pp. 85–91, 2007.

[23] SAT4J, "A satisfiability library for Java," Version 2.2, 2009, http://www.sat4j.org/.

[24] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *Proceedings of the Future of Software Engineering*, pp. 259–268, May 2007.

[25] F. Kluge, S. Uhrig, J. Mische, and T. Ungerer, "A two-layered management architecture for building adaptive real-time systems," in *Proceedings of the 6th IFIP International Workshop on Software Technologies for Embedded and Ubiquitous Systems*, pp. 126–137, 2008.

[26] M. Litoiu, M. Woodside, and T. Zheng, "Hierarchical model-based autonomic control of software systems," *SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–7, 2005.

[27] A. Lapouchnian, S. Liaskos, J. Mylopoulos, and Y. Yu, "Towards requirements-driven autonomic systems design," in *Proceedings of the Workshop on Design and Evolution of Autonomic Application Software*, pp. 1–8, 2005.

[28] M. Zeller, G. Weiss, D. Eilers, and R. Knorr, "A multi-layered control architecture for self-management in adaptive automotive systems," in *Proceedings of the International Conference on Adaptive and Intelligent Systems (ICAIS '09)*, pp. 63–68, September 2009.