21

Towards Automotive Embedded Systems with Self-X Properties

Gereon Weiss, Marc Zeller and Dirk Eilers
Fraunhofer Institute for Communication Systems ESK
Germany

1. Introduction

Since the first pieces of software have been introduced into automobiles in 1976, the complexity of automotive software systems is growing rapidly. Today automotive software is widely installed for diverse applications ranging from the infotainment domain (e.g. entertainment, navigation, etc.) with typically no real-time requirements to safety-critical control software (e.g. engine control, safety functionalities, etc.) with hard real-time requirements. In addition, many comfort functionalities of automobiles are realized by software nowadays (e.g. the control of the air condition system, electronic window regulator, etc.). Up to 90% of today's innovations in the automotive industry are realized by hard- and software (Pretschner et al., 2007). This results in up to 2,500 "atomic" functions realized in software on up to 67 electronic control units (ECUs) in modern high-end cars (Fürst, 2010).

For the future development of automobile electronics, there are two major trends: A growing number of functionalities and through this a growing importance of software in the car (Hardung et al., 2004). Future generations of cars will be equipped with many new, complex *features* (Czarnecki & Eisenecker, 2000). For example, functionalities to support active driving safety (e.g. driver assistance systems), features which enable new innovative driving concepts (e.g. engine control for hybrid vehicles), or new functionalities in the comfort domain (e.g. new infotainment features). Most of these functionalities will be realized in software, which increases the amount and importance of software within the automotive domain necessarily. But these new features will also increase the complexity of future vehicular system architectures. For instance, driver assistance systems increase the complexity because they interact with several in-vehicle domains, e.g. the power-train and infotainment domain. In future, the trend of establishing more and more interactions between software components will continue, e.g. through x-by-wire features, where mechanical transmission is replaced by electrical signals. This results in a growing interdependency of separated software domains and in an increased need for interconnection. Another important aspect is the continuously growing number of functional variants caused by customer-specific equipment options or country-specific regulations. At the same time, the demand on the software quality within the automotive domain is very high at all times. These requirements must be satisfied in the future, despite the increasing complexity of automotive software architectures. Even today it is a great challenge to manage these systems from the outside.

In recent years, a lot of research has been done, trying to explore new methods for the management of general complex software systems. Within the research area of *Self-adaptation* (McKinley et al., 2004) and *Self-organization* (Serugendo et al., 2004) new paradigms for the management of complex systems have been introduced. Both approaches utilize control-loops for feedback-based control of the system. Self-adaptive systems realize the adaptation of the system in a top-down manner by setting global goals which are enforced hierarchically. On the contrary, self-organizing systems follow a bottom-up approach in which the local interaction of elements results in the intended global behavior. These paradigms for the development of general systems which are capable of adaptation also constitute a promising solution to master the complexity within automotive embedded systems (Weiss et al., 2009). Thereby, vehicular software systems can be enhanced with self-management capabilities. These so-called *self-x properties* (like self-configuration, self-healing, self-optimization or self-protection (Kephart & Chess, 2003)) improve the scalability, robustness and flexibility of the system.

In 2001 IBM introduced the *Autonomic Computing (AC)* paradigm (Horn, 2001). The main idea is the adaptation of the behavior of the central nervous system which interacts autonomously. As basic principle the management of *autonomic elements* is realized by a reconfiguration-cycle where each autonomic element monitors and analyzes the environment, plans its next steps and executes the resulting actions. Originally, the focus lies on the management of large-scale computer networks. With *Organic Computing (OC)* (Schmeck, 2005) a novel principle for self-organizing systems is given by imitating adaptive, life-like behavior in the nature. Self-organization is realized on different abstraction levels with observer/controller models utilizing control-loops. No particular field of application is addressed and interdisciplinary research is covered. With the *Self-adaptive Software Program* (Robertson et al., 2001) a very ambitious research field is addressed where software evaluates and changes its own behavior at runtime. Therefore descriptions of intentions and alternative behavior need to be added in the deployed software.

In the automotive sector several initiatives have already focused on evaluating self-x techniques for vehicles. A high-demanding goal for the future of transportation are autonomous cars which can adapt even in high complex scenarios as in urban traffic (Urmson & Whittaker, 2008). As promising as early results are, many - not only technical - problems are not solved yet and thus the practical appliance of autonomous driving is still not foreseeable yet.

For the in-vehicle information and entertainment functionalities the *Media Oriented Systems Transport (MOST)* bus (MOST Cooperation, 2008) is a widespread established standard. It facilitates functional composition with a powerful API and already features very limited self-x properties with its configuration management. The *Automotive Open System Architecture (AUTOSAR)* (AUTOSAR Consortium, 2010) initiative is a consortium with the goal of an open standard for the automotive software architecture. Through a component-based architecture the reuse and scalability of future automotive software is pursued. By a virtual integration of software components (*Virtual Function Bus*) the allocation of functions to ECUs can be assembled at design time. Even though this approach facilitates a more liberal way of the allocation, it does not support any dynamic allocation at runtime. Hence, self-adaptation techniques that rely on reallocation of functions cannot be applied. In (Trumler et al., 2007) self-healing and self-configuration is evaluated in a component-based automotive architecture which indicates the potentials arising with these techniques. Dinkel (Dinkel, 2008) focuses on the development and simulation of a completely new IT-

architecture for future cars with self-x capabilities. It utilizes Java and OSGi for simulation purposes and is not applied in the field. The DySCAS project (Anthony et al., 2006) focuses on developing a middleware enabling dynamic self-configuration in today's cars. For the reconfiguration of the system a policy-based mechanism is utilized. Another approach was proposed by DaimlerChrysler (Hofmann & Leboch, 2005). The *EvoArch* project tries to put more value on the autonomy of the different parts of the automobile enhancing the automobile with self-x properties. Within the research project ReCoNets (Teich et al., 2006) fault-tolerance is addressed by bringing Hardware/Software- Reconfiguration into the automobile. Although, reallocation of both hardware and software is a consequent progression of the currently advancing adaptivity and decomposability, it is not aligned with present automotive development method (e.g. FPGA reconfiguration).
As briefly described before, different approaches are in progress enabling self-x properties in future cars with various degrees of a possible adaptation. Many open challenges need to be researched for meeting the domain-specific requirements of automotive electronic systems (e.g. the verification of adaptation). But no project focuses on the embedding of techniques in present automotive electronic systems allowing a transition to self-adaptive systems.

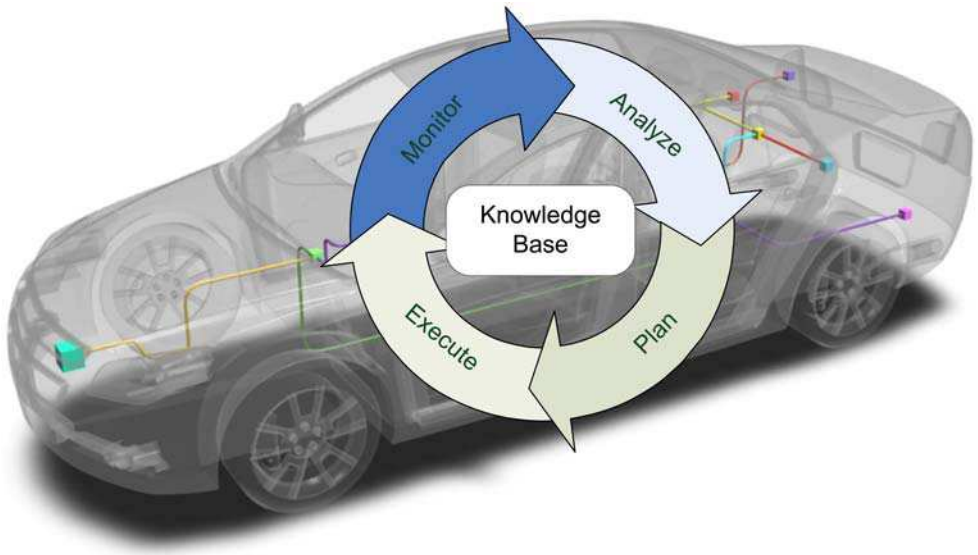


Fig. 1. Self-adaptation in the context of automotive embedded systems

The characteristics of *self-adaptive* or *self-organized systems* might provide a solution for the growing complexity in the automotive domain. Within this chapter we present an approach for enhancing automotive embedded software systems with self-x properties which increases the scalability, robustness and flexibility of vehicular software systems (cp. Figure 1).
The structure of this paper is organized as follows: In Section 3 we present the challenges in the realization of self-x properties in automotive embedded systems. Afterwards, we illustrate the advantages of self-adapting automotive software systems by presenting concrete use cases. Our model-based approach to design self-adaptive automotive software systems is outlined in Section 5. In Section 6 we introduce an approach to realize self-x capabilities during runtime. Finally, we conclude the chapter in Section 7.

2. Self-adaptation in the context of automotive embedded systems

Under the umbrella term self-adaptation a set of terms is defined, e.g. self-x properties. In this section we will explain relevant terms in the domain-specific context of automotive embedded systems.

Self-adaptive software systems must be able to adapt the behavior (*Behavioral Adaptation*) or/and the structure (*Structural Adaptation*) of the system to changes in the environment or within the system itself (Zadeh, 1963)(McKinley et al., 2004). To adapt itself autonomously, the system must be able to detect and to evaluate its own context. Therefore, a model of the system and its feasible states is needed. The comparison of the currently detected context and the reference situation of the system model enables the evaluation of the current system state. This so-called *Self-Awareness* is the basis for the adaptation of the system or a sub-system.

During runtime self-awareness is enabled by the *Self-Description* of each component within the system. The language and the scope of this description must be as small as possible to fulfill their purpose. Furthermore, the description must be processable by an embedded system with limited resources.

Today, there are already examples for adaptive behavior in modern automobiles. For example, the engine control adapts the fuel injection according to the current road behavior. But this kind of adaptivity is limited to control applications and allows adaptation only in predetermined variants. To fully exploit the potentials of adaptivity, it is not practical to limit the variability by calculating all possible system configurations in advance (during design time). Due to the enormous amount of possible variability in today's and future automotive software systems, it is necessary to adapt the system dynamically at runtime (*Dynamic Reconfiguration*) (Geihs, 2008).

With respect to (Hofmeister, 1993) three different kinds of dynamic reconfiguration can be differentiated:

1. The implementation of a component is replaced by another one (Behavioral Adaptation).
2. The relation between components of the system is modified. New components and features can be added or removed (Behavioral Adaptation).
3. The allocation of the software components is changed without the modification of the logical structure. Therefore, components are migrated from one hardware platform to another (Structural Adaptation).

In the context of automotive embedded systems behavioral adaptation is achieved by the dynamic activation or deactivation of specific software-based features during runtime. Structural adaptation is realized by the dynamic reallocation of software components onto the available control units during runtime.

To apply dynamic reconfiguration successfully in the context of automotive embedded systems, we have to deal with so-called *emergent behavior*. Emergence is defined as a property of a total system which cannot be derived from the simple summation of properties of its constituent sub-systems (Müller-Schloer, 2004). It is a result of self-adaptive or self-organizing processes and leads to a system behavior which is not explicitly defined (Wolf & Holvoet, 2004). This may lead to unwanted or uncontrolled behavior - so-called *emergent misbehavior* (Mogul, 2005). Because automotive embedded systems provide safety-relevant applications (e.g. airbag), it is very important that the predefined requirements and constraints of the system are preserved during runtime. Therefore, emergent behavior is not tolerated in adaptive automotive software systems.

Instead, the aim of self-adaptation in the context of vehicles is to improve to the system with different self-x properties:

Self-Management: The system must be able to manage its own functionalities without actions from outside the system. The complexity of the system management task can be decreased by increasing the management capability of single components. For example, by adding a self-description to each element within the system the current status of self-aware elements does not need to be supervised continuously. Thereby, a divide-and-conquer strategy is applied. The more complex the management of individual elements is, the less complex is the management of the overall system. In the context of automotive embedded systems, a trade-off is needed between the self-management of the overall system and the management of individual parts of the system.

Self-Configuration: Today, the configuration of complex systems (e.g. vehicular software systems) is performed by experts. By enhancing a system with self-configuration capabilities, it is possible to find a feasible configuration in a distributed and autonomous way. Thus, the manual and error-prone configuration process can be omitted. Furthermore, self-configuration enables the dynamic integration of new components and features during the runtime of the system. For example, in today's cars the autonomous configuration is already supported by the infotainment system MOST in which a central instance - the so-called *NetworkMaster* - enables the configuration of features (*MOST FBlocks*) independent from their position.

Self-Healing: The autonomous diagnosis of the current system state enables the detection of invalid system states. Afterwards, a valid system state is restored by means of self-healing. The self-healing process is supported by the self-configuration capabilities of the system. To achieve the complete "healing" of the system a certain degree of redundancy is assumed. The ability to heal itself is growing with the size of the overall system. Thus, self-healing is especially interesting in the field of infotainment and telematics applications. Delays due to the process of self-healing must be considered additionally during system design.

Self-Protection: Self-protection of specific elements is necessary if the system is operating in a dynamic environment. For automobiles which are divided into different separated domains of automotive software, self-protection is an additional overhead which is not justifiable in the context of present automotive embedded systems. But the protection against critical system states and the prediction of problematic conditions is an option to prevent the system from failures and to satisfy the safety requirements within the automotive domain. Furthermore, by opening the in-vehicle communication to the outside world (e.g. car-2-x communication (CAR 2 CAR Communication Consortium, 2010), the importance of self-protection will increase.

Self-Optimization: The proactive search of a specific element for new opportunities to optimize its own behavior helps to reach the optimal system state. But to achieve such an optimization, resources are continuously needed. In the context of automobiles, it is necessary to evaluate carefully if this effort for self-optimization is justifiable. Context-based self-optimization in terms of different predefined scenarios may be a potential trade-off for automobiles.

To use the full potential of the previously described self-x properties in automotive embedded systems, certain challenges must be met. In the next Section we will describe these challenges in detail.

3. Challenges in realizing self-x properties in automotive embedded systems

For realizing self-adaptive software systems for automobiles which enhance the system with self-x properties, several challenges have to be addressed which we describe in the following. Today, the software-based features of modern vehicles are statically assigned to specific ECUs. Since the number of control units cannot be expanded arbitrarily for the integration of new features, new concepts for the dynamic allocation of features to ECUs are needed. The Automotive Open System Architecture (AUTOSAR) (AUTOSAR Consortium, 2010) initiative aims to establish a standardized software architecture for cars since 2002. By using a component-based approach, the reusability and the scalability of automotive software is increased. The so-called *Virtual Function Bus (VFB)* enables the virtual integration of software components by allocating these components to ECUs during design time. Thereby, the flexibility of designing automotive embedded systems is increased. However, with a more modular approach like AUTOSAR there is the need to decompose features into services and services into atomic functions. This approach enables the reuse of functionalities and reduces the overhead by eliminating redundant implementations within the software system. Furthermore, more freedom for the runtime adaptation is achieved by a more fine-granular decomposition of features.

Modern runtime environments for automotive software, like *Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen (OSEK)* (OSEK VDX Portal, n.d.) or AUTOSAR, are only configured statically during design time. Within statically designed systems most of the available resources are assigned permanently. Dynamic changes of this configuration (e.g. creating a new task) during runtime are not allowed. As runtime adaptation is needed to control the growing complexity, a runtime resource and conflict management is inevitable for the dynamic reconfiguration of the system (e.g. instead of a statically resolved virtual function bus with fixed port assignments in AUTOSAR, a real communication bus with a dynamic scheduling is needed). Therefore, the resources of each ECU - like CPU, memory, etc. – must be managed dynamically.

Although sensors and actuators are separated from the computation, there is still the necessity for locality of the software functions to access the sensor/actuator data in today's automotive embedded systems. Caused by the growing cross-linking of different functionalities – even inter-domain (e.g. caused by driver assistance features) - sensor and actuator data must be accessible by all features. Techniques like publish/subscribe and distributed data access might ease this problem. By the complete separation of sensors or actuators from the computation (control unit), their data can be accessed throughout the whole in-vehicle network. In case of an ECU breakdown the data of sensors or actuators will still be available. Thus, a more flexible distribution of software components is enabled which is mandatory to tap the full optimization potential of self-adaptation.

Another challenge for the realization of self-x properties poses the heterogeneity of today's vehicle electrical system architecture where diverse technologies are incorporated. The various hardware platforms and the different interconnection systems make it difficult to reallocate software components to different ECUs during runtime. For the migration to a different ECU, software components must be recompiled which increases the latency to adapt the system enormously or the program code of each component must be pre-compiled for the corresponding hardware platform and stored within the in-vehicle network. But for the various hardware platforms in today's automobiles the memory capacity must be increased significantly which is not cost-effective. Only an abstraction from the underlying

technology (e.g. via a runtime environment or middleware) will allow the interaction of the components and thus the efficient self-adaptation of the overall system.
In the automotive domain several applications with divergent safety and real-time requirements (specified as *Safety Integrity Level, SIL* (International Electrotechnical Commission (IEC), 1998)) are composed to one system. Presently, the requirements are met by a separation into domains (infotainment, power-train, comfort and chassis). Thus, a major challenge is to guarantee and meet the safety requirements of automotive systems even in adaptive systems (for example the ability to satisfy hard timing constraints). This results in an implied limitation of possible configurations of the system. The mandatory system constraints must be extracted during the design process and enforced during runtime. Thereby, the reconfiguration process of the system must not influence the behavior of safety-critical features. For this reason, the constraints and the effects of the adaptation must be considered in safety-relevant systems - like automobiles.
To realize self-adaptive or self-organizing (technical) systems a control instance is needed which collects information about the system, analyses these information and decides how to adapt the system to reach the predefined objectives (Mühl et al., 2007). Such a control instance must ensure that the system is in a correct state at any time. Present automotive systems have no capabilities to describe their properties and requirements at runtime so that a controller instance could not obtain enough information about the current systems state, only deduced information. Accordingly, a description of the components has to be made available at runtime. For component-based approaches a self-description (for hardware and software components) generated out of the design seems promising. But a trade-off between the expressiveness with more potential for self-adaptation and the overhead of a higher complexity for analysis algorithms has to be done.
To address these challenges in realizing self-adaptive automotive embedded systems with self-x properties, a design process is necessary which allows the modeling and the verification of adaptivity while considering the domain-specific requirements. Furthermore, we need a runtime environment which monitors the requirements and constraints specified during design and which enables the dynamic reconfiguration of the system. Before we introduce these concepts in Section 5 and 6, use cases which exploit self-x properties of automobiles are presented in the next section.

4. Use cases for the application of self-x properties in automobiles

By enhancing the automotive software system with the self-x properties as described in Section 2, significant improvements beyond today's state of the art may be realized.

4.1 Resource optimization

A car operates in a continuously changing environment. On the freeway, features like the cruise control system, the lane departure warning system or the adaptive driving speed control are used. While driving in the city, other or modified driver assistance features are needed (e.g. the parking assistant system). The night view assistant, adaptive headlights or the high-beam assistant are only used by night or in cases of restricted view. To reach the optimal utilization of the available resources, individual features have to be used situation-based. Thereby, the required hardware resources are reduced by mutually exclusive features. The situation-based deactivation of unnecessary but simultaneously possible features saves resources (e.g. energy, computing time, etc.) during runtime.

4.2 Fault tolerance

Due to reasons of cost and efficiency, there is almost no redundancy in today's automobiles. The failure of software-based functionalities must be repaired normally by a specialized car repair. In some cases, the failure of an electronic component may lead to the total breakdown of the car. These failures within the car's electric/electronic are very negative experiences for the customers and in worst case may possibly threaten the life of the driver. By enhancing automotive embedded systems with self-healing capabilities, the fault tolerance and the availability of the systems is increased by software without costs for additional hardware resources. For example, the failure of a control unit can be compensated by the dynamic adaptation of the system's structure. Thus, a temporary *emergency operation* of the automotive embedded system is enabled by equipping the system with self-x properties. Life-threatening situations for the driver can be avoided and the satisfaction of the customers can be increased.

4.3 Third party consumer device integration

Today, the replacement of vehicle components, the upgrade of new components (after-market products) within the car repair or the update of the vehicular software may lead to problems because the software versions of specific components may not be compatible with the shipped vehicle software. Furthermore, the user demand for integrating modern consumer devices (e.g. mobile phones, smart phones, PDAs, etc.) into the vehicle is very high. The short lifecycles (especially in comparison to the life-cycles of automobiles) and the diversity of these devices have led to proprietary solutions for connecting consumer devices to the vehicle infotainment system. Enhancing the automotive embedded system by self-configuration enables the seamless, flexible and scalable integration of new software-based features, new hardware components and consumer devices. Thereby, failures due to software versions which are incompatible are eliminated. Based on the autonomous allocation of software components to ECUs, self-configuration reduces the complexity for the system integrator and the effort during the production of the automobile. The error-prone manual assignment of features to hardware platforms and the time-consuming flashing (software deployment) of the ECUs during the end-of-line production can be omitted.

4.4 Partial in-vehicle network operation

Another use case which can be enabled by self-adaptive automotive systems with self-x properties is the partial in-vehicle network operation. In this use case certain parts of the in-vehicle network or single ECUs can be shut down to save resources (e.g. energy) during runtime. This can be done in certain contexts (situations) when all features located in a distinct area of the network are not required or can be substituted by functions running on other platforms. These functions might be started dynamically or for simplicity run as shadow tasks in the background all the time. The potential benefits of a partial in-vehicle network operation of course strongly depend on the mapping of software components to the ECUs. For an optimal allocation, with respect to the partial in-vehicle network operation, the distribution should cluster functionality which is and is not used in the same context. Self-x properties may improve the partial in-vehicle network operation by dynamically reallocating software components to shut down even more parts of the network.

5. Designing automotive embedded systems with self-x properties

Nowadays software development in the automobile area has been dominated by its traditional development of mechanical components, as it has been practiced for the last decades. With the growing number of automobile features realized mainly in software, the design process is becoming more and more challenging. For managing the complexity of distributed embedded systems like automotive electronic systems a specialized software development process is necessary which allows the abstraction and realization of single system components and the whole system. Therefore, the description and the description language are a critical factor how well - in terms of how close to the reality - the system can be modeled on different layers of abstraction. In a distributed adaptive system with self-x properties - beneath the static description - the dynamic description in particular is of great importance.

An architecture of a software system is generally described by an *Architecture Description Language*. In the automobile domain several efforts for the system modeling are undertaken. EAST-ADL (Electronics Architecture and Software Technology - Architecture Description Language) (Cuenot et al., 2008) as a domain-specific architecture description language is a promising standard for the seamless automotive architecture design. On the Implementation Level it also targets the component-based architecture AUTOSAR (AUTOSAR Consortium, 2010) allowing an integration with this standard. In our approach we foresee to utilize EAST-ADL as basis for describing an automotive electronic system enhanced with self-x properties. EAST-ADL allows the design of static automotive systems based on UML (EAST-ADL2, 2010), but for adaptive systems with self-x properties it has to be enhanced, e.g. by considering dynamic behavior at runtime. Additionally defined attributes are modeled to specify the runtime variability of the EAST-ADL system components. Thereby, for example self-configuration and self-healing can be supported by annotating components to be reconfigurable at runtime. Thus, they can be instantiated in a self-configuration or self-healing process.

The design space of runtime adaptive systems with self-x properties increases exponentially in terms of possible runtime configurations. Thus, special emphasis has to be placed on the *validation of the dynamic behavior* in early design phases. This allows an iterative validation of the system and its adaptation behavior leading to find faults early in the development process. By this, the development costs can be decreased as late design changes typically result in drastically increased costs. The designed and validated system has to be executed by a tailored runtime environment as outlined in Section 6. The allowed degree of variability has to be defined in the design to comply with requirements on the system safety. An uncontrolled self-organization is not feasible in safety-related systems as certain requirements need to be met at any time. Especially, the abstraction of the definition of the adaptation is crucial. For the validation of the system and its behavior this should be defined on a rather high level of abstraction. Thereby, the allowed nominal behavior - including the adaptation behavior – can be constrained. On a high level of abstraction so-called features can be modeled representing user-visual functionality. Additional to static features which are present in a product, dynamic features can be defined. These represent adaptive functionality on an abstract level. They contain interdependencies and distinct selection criteria which define their selection at runtime. Derived contexts can be used to select a set of dynamic features with respect to the actual driving situation. Car manufacturers may specify distinct scenarios (e.g. driving situations) in which defined functions of the automobile are necessary.

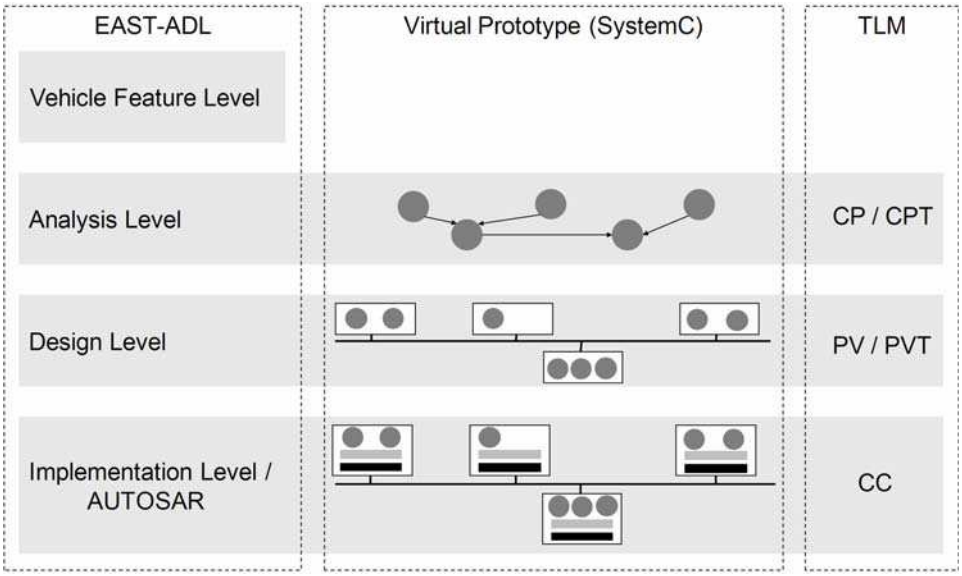


Fig. 2. Integration of SystemC validation on EAST-ADL layers of abstraction

For the iterative validation of the design we use SystemC simulations. SystemC is a standardized system modeling and simulation language which supports Hardware/Software-Co-Design and Co-Simulation. It is specified and promoted by the *Open SystemC Initiative (OSCI)* (Open SystemC Initiative (OSCI), 2010) and has been approved by the IEEE Standards Association as IEEE 1666-2005 (IEEE, 2005). Based on the wide-spread programming language C++, SystemC provides artifacts to simulate concurrent processes and an event-driven simulation kernel. It incorporates semantic constructs of hardware description languages (like VHDL and Verilog) and can be used to model the holistic system using plain C++. A stepwise refinement in a top-down design process is realized with the SystemC *Transaction-Level Modeling* (TLM) (Cai & Gajski, 2003) methodology. TLM is a methodology used for modeling digital systems which separates the details of communication among computational components from the details of the computational components. Details of communication or computation can be hidden in early stages of the design and added later. Since the application of SystemC for a simulation-based validation of automotive electronic systems is a promising approach for the design exploration and hardware sizing, it is integrated within our approach for adaptive automotive systems with self-x properties. Therefore, we adopt SystemC in the development process with architecture descriptions based on EAST-ADL. An automatic transformation on the layers of abstraction of EAST-ADL to the SystemC TLM levels is performed (see Figure 2) which enables a simulation-based validation. Thereby, architecture models can be iteratively refined and improved in the development process. Through this, adaptive automobile systems can be seamlessly developed and described. The design of such a system including the defined adaptivity has to be realized and enforced at runtime in the end, which is described in Section 6. In the next section we present an automotive example which has been designed with the above methodology and validated by a SystemC simulation.

5.1 Automotive example for validation

As outlined before, in the design of automotive systems with self-x properties the validation of such systems is increasingly challenging. Therefore, we transform EAST-ADL models to executable SystemC models in a prototypical tool-chain. For evaluation purposes an automotive case study (Hardung et al., 2004) has been modeled in EAST-ADL and transformed to SystemC simulations on different levels of abstraction.

The use case is located within the so-called body domain of an automobile and consists of the four features *exterior light*, *direction indication*, *central door locking* and *keyless door entry*. The exterior light feature allows controlling the front and rear lights of the vehicle. The lights can be switched on/off manually or automatically through darkness or rain detected by the rain/light sensor. These inputs are interpreted by the function exterior light control which controls the light units (front and rear). For the direction indication a direction indication switch can be used to signal the turning direction. With the hazard light switch, risky driving situations can be signaled to other road users. Therefore, the direction indication master control informs the direction indication front and rear controls about the designated status of the direction indication lights. These turn the direction indication lights on or off in the front and rear light units. Central door locking allows locking and unlocking all doors simultaneously by using the key in the lock or by radio transmission. A radio receiver signals the information to the central door locking control. This function flashes the direction indication lights for a feedback to the driver and controls the four door locks of the car. An additional feature to the un-/locking of an automobile is the keyless entry. A driver can approach his car with the key in his pocket and the doors will unlock automatically. It can be locked by simply pressing a button on the door handle. Antenna components detect the key in the surrounding and inform the central door locking function which in turn unlocks the doors. With respect to the interaction with exterior light (which gives feedback via the direction indication lights), it does not make any difference whether the doors have been unlocked in a standard way or via the keyless entry. At Analysis Level this use case is modelled in EAST-ADL by so-called *FunctionalDevices* components: *KeylessEntryController*, *CentralDoorLockingController*, *DirectionIndicationMasterController*, *DirectionIndicationFront-Controller*, *DirectionIndicationRearController* and *ExteriorLightController* as is depicted in Figure 3. The behavior of these functionalities is described as UML opaque behavior of the components (C++ source code). Additionally, behavior can also be modeled with UML Statecharts as a UML based behavior specification. Communication is designed as data flow between the components represented by `FunctionFlowPorts` and `FunctionConnectors`. A SystemC simulation generated from this level includes modules interconnected for each of the above mentioned FunctionalDevices. They implement the respective behavior of these modeled components in a thread of the module. A simulation based on the abstract EAST-ADL Analysis Level of the use case was realized. Thus, the interaction of the abstract modeled functionalities can be validated with a simulation-based analysis.

At Design Level the use case is modeled in a Functional Design Architecture (FDA) representing the software parts and a Hardware Design Architecture (HDA) representing the hardware parts of the use case realization. The FDA includes `DesignFunctionTypes` for the software functionalities of the use case and `LocalDeviceManagers` representing the software access to the modeled sensors and actuators. Latter are designed in the HDA together with the hardware platforms (`Nodes`) and the interconnecting `LocalBus`. Components in the FDA are interconnected with `FunctionConnectors` and in the HDA

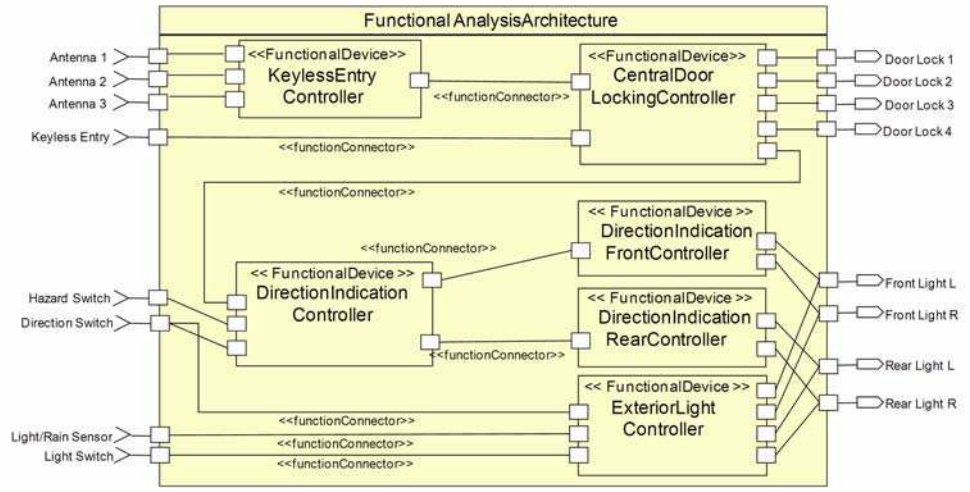


Fig. 3. Composite diagram of the use case at Analysis Level

with `HardwareConnectors`. Additionally, `LocalDeviceManagers` exist for each depicted *Sensor* and *Actuator* in the Functional Design Architecture which are not explicitly displayed in this figure. The generated SystemC implementation of the use case at Design Level - which models software and hardware explicitly - is depicted in Figure 4. It includes the use of a framework for automotive-specific modules. For example, ECUs and software functions can be included out of a library as specific *sc_module* implementations. As can be derived from Figure 4 the EAST-ADL Design Level components are generated as *sc_modules* representing software functions. These modules are included in another SystemC module which realizes a hardware platform with attached sensors and actuators in form of *sc_modules*. These hardware platforms are interconnected by a module implementation of the

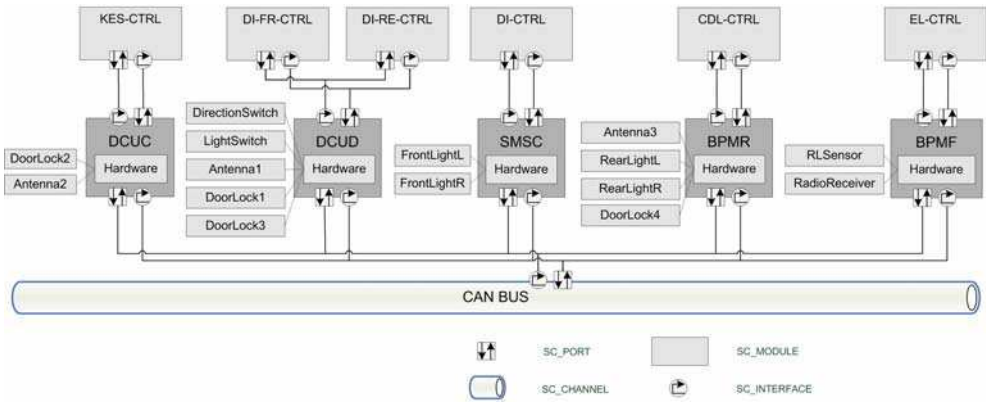


Fig. 4. Overview of the generated SystemC simulation at the Design Level

defined *LocalBus*. SystemC interfaces and channels realize the concrete interconnections of the modules. For example, a specialized type of *sc_interface* (*EcuSw_If*) realizes the communication between software functions and ECU modules.
The introduced transformation is realized in a prototypical *toolchain* which integrates into the Eclipse environment as a plug-in. By this, it can easily be used with EAST-ADL models based on UML in Eclipse (e.g. with the Papyrus UML modeling tool which supports EAST-ADL). The transformations itself are implemented as templates of the *Xpand* model-to-text transformation language. They use EAST-ADL models as input and generate the single SystemC files according to the mapping of the languages. Currently, simulations can be generated from the Analysis Level or Design Level. Simple checks allow to examine the conformity for a simulation. Because a generation of incomplete models in early design stages should be possible, the checks are only as strict as needed for generating correct SystemC simulations. This supports the iterative simulation of ADL models in the design process. For the simulation at Design Level we utilize a self-developed framework called *DynaSim* which allows the modeling of an automotive in-vehicle network in SystemC. The generated files refer to SystemC models in the DynaSim library (e.g. ECUs or software functions). By this, a simulation can be performed considering the automotive-specific system environment. We have briefly introduced our approach for the validation of self-x properties in adaptive automotive systems at different design stages on the basis of a case study. Since the designed properties of the models have to be ensured at runtime, the next section focuses on the runtime mechanisms.

6. Enforcing self-x properties during runtime

To enforce self-x properties in automotive embedded systems during runtime, an adopted runtime environment is needed. This must provide mechanisms to manage system resources dynamically and must enable the structural and behavioral adaptation of the automotive software system. Furthermore, it is essential to satisfy all mandatory requirements and constraints which are defined during the design process (see Section 5). Thereby, the correct system behavior can be guaranteed during runtime and unwanted or uncontrolled behavior can be avoided. In our approach this is realized by using a control loop based mechanism according to the AC paradigm. The automotive embedded system is monitored continuously, changes are analyzed and adaptations of the system are planed and executed.
Especially for automotive systems with various requirements and constraints, enabling self-x properties and building such a control loop is a difficult task. Not only functional but also non-functional requirements (e.g. timing, safety) have to be met during runtime. Generally, automotive software features are divided into so-called *Safety Integrity Level (SIL)* according to their safety relevance. Each SIL has different requirements which must be considered by the control loop accordingly. For example, a safety-critical feature (e.g. the airbag control) may not be affected at any time - even during reconfiguration. However, a feature from the infotainment domain (e.g. the hands-free kit) can be deactivated during reconfiguration without life-threatening consequences. The control architecture of the automotive embedded systems must take these requirements into account.

Furthermore, the control architecture of a self-adaptive system which provides safety-relevant applications has to be highly dependable and needs to provide the necessary degree of flexibility to react on changing conditions in an appropriate way. Therefore, managing the whole system by one single control loop is complex and results in a single-point-of-failure.

In order to cope with these requirements, a divide-and-conquer strategy can be applied which is partitioning the system into smaller entities - so-called *clusters*. A cluster is defined as a logic group of software components as well as a sub-set of requirements and system objectives which have to be met by all of software components within the cluster.

The partitioning of the system into different clusters can be based on different criteria:

- Functional dependencies
- Non-functional dependencies
- Physical location of the functions
- Requirements and system objectives

These criteria can be combined in any way in order to provide an optimal segmentation of the automotive software system. In this context, an optimal segmentation means that decisions can be made in a single cluster from a local point of view without interfering other clusters.

Repeated partitioning of the system leads to a hierarchy of clusters, representing the entire automotive software system. Each cluster within this hierarchy is controlled by its own control loop resulting in a hierarchical multi-layered control architecture (cp. Figure 5). This control loop is an external component which is not included in the cluster itself. It is monitoring and controlling the current state of a cluster continuously, so that all requirements and system objectives are satisfied. If one of the defined requirements or system objectives is not met anymore, the affected cluster must be adapted in order to meet all requirements and system objectives again. This is either done by the reassignment of software components to different ECUs (structural adaptation) or by the activation/deactivation of specific software-based features (behavioral adaptation).

The clusters on the lower layers have a local scope with only a few requirements to be satisfied and software components to be controlled. Thereby, an individual implementation of the control loop and a fast reaction on changes is possible. Many clusters have only one system objective, so tailored methods and algorithms can be applied for the observation and control of the cluster. Due to different implementations of the control cycle, the control architecture can be customized individually for the different needs of the automotive software domains. As a drawback, the clusters on the lower layers have a restricted scope and may not be capable of finding a new valid assignment of software components to ECUs.

On higher layers, the number of software components managed by a cluster is increasing, as the number of requirements and system objectives, which have to be met. Thus, on the one hand the chance of finding a new allocation which satisfies all requirements is increased; one the other hand, it is more complex to find one at all.

The *Root Cluster* on the top layer represents the top element in the hierarchy and manages the entire automotive embedded system. But it is only involved in the self-adaptation process as a last instance. The Root Cluster is not aware of the decisions made on the lower clusters.

Within an *n*-layered control architecture up to *n* control loops are involved in the process of self-adaptation. In worst case, calculations of the control cycle are performed *n* times until a new valid allocation is found, resulting in a long response time to changes and a certain overhead. To reduce this overhead, partial solutions of calculations are passed to the next higher layer and will be reused there. But nevertheless, a trade-off is needed between the overhead provided by each new layer added to the control architecture and the advantages gained by it.

Each control loop within the hierarchical multi-layered control architecture consists of four stages, according to the AC paradigm:

Monitoring: Certain parameters of the system must be monitored continuously to detect changes quickly and dependably within the system's environment or within the system itself. To enhance the system with self-healing capabilities, malfunctions must be discovered autonomously. Traditionally, monitoring and fault detection recognize the malfunction of individual components. Thereby, the expected behavior is compared to the actual behavior of the component. If the actual behavior deviates from the expected behavior, a failure is likely. The representation of the expected behavior or the measurement of the actual behavior is very specific and tailored for a certain component. With growing complexity, interdependencies and distribution of the vehicles software features the following problems need to be solved:

- Monitoring the complete system behavior: Although each individual component is working correctly, the overall system exhibits incorrect behavior.
- Monitoring the dynamic system behavior: Adaptive systems may operate in different system configurations. Thus, it is difficult to predict all possible configurations (State Explosion) and to monitor the system with static monitoring techniques.
- Detection of unknown failures: Today's monitoring techniques have limited abilities to discover unknown failures during runtime. This is due to the use of error patterns to identify specific errors in most monitoring mechanisms. Errors which do not match the predefined patterns are not detected.

Analysis: During the analysis stage the present, the desired and the future state of the system must be detected and predicted. Thereby, the analysis stage is closely linked to the monitoring stage, because the observations from the monitoring of the system are directly passed to the analysis. In contrary to the monitoring stage, the analysis of the system uses additional information (e.g. current environmental conditions, predefined system objectives, etc.) for the evaluation of the actual system state. The so-called *Livingstone Model (Cimatti et al., 2003)* may be used for this purpose. It describes a model-based diagnostic mechanism for autonomous spacecrafts with self-configuration capabilities. Therefore, it compares the predicted behavior with the actual behavior and makes statements concerning the needed actions based on the model of the system. In diagnosis, information about which features are needed for the further operation of the system beyond the detection of error causes are made depending on the current environmental conditions of the system and the system's objective (Williams et al., 1996). With these information about the available resources and the features needed in future which are gained from the analysis stage, the next stage (planning) may find a new allocation of software components to ECUs.

Planning: The planning stage creates or composes a set of actions to modify the managed elements of the system. In the context of automotive embedded systems the planning stage determines a new set of features and a new allocation of software components and control units which fulfills all predefined requirements. This allocation problem can be either expressed as *Generalized Assignment Problem (GAP)* (Cattrysse & VanWassenhove, 1990) or as *Constraint Satisfaction Problem (CSP)* (Dinkel & Baumgarten, 2007). Since the allocation problem is a $\mathcal{NP}$-hard optimization problem, a heuristic approach is needed to solve this problem during runtime. The challenge of the planing stage is to find a trade-off between the computation time and the quality of the solution in order to satisfy the requirements of the automotive domain.

Execution: The execution stage of the control cycle provides mechanisms to execute the plan determined by the planning stage in order to adapt the system. Within the vehicular software system these changes refer to the activation or deactivation of features as well as the migration of software components to different ECUs. In the context of safety-relevant applications, it is important that the normal system behavior is not disturbed during the reconfiguration of the system. The migration of software components can be chosen whether the context of the software component (variables, program stack, etc.), the program code (binary or source code) or both is transfered to another ECU. Thus, for example, a safety-relevant feature may exist on several ECUs. In case of a migration only the current context of this feature must be transfered to another ECU. Other features may be recompiled for the target hardware platform in case of a migration and transmitted as binary code. According to the predefined requirements of a feature, specific techniques for the migration of software components may be used.

As pointed out before, a multi-layered control architecture provides the necessary performance and degree of flexibility to react on changes within the system's environment or within the system itself in an adequate way. Thus, it is possible to supervise these requirements predefined during the design and to adapt the system if one of the requirements is not satisfied anymore. Small clusters with individually tailored control loops can react quickly, while clusters on higher layers have a wider scope and more information to find the optimal configuration of the automotive embedded system. Thus, the chance of finding a new valid allocation of software components to ECUs is better on upper layers. Furthermore, a software component is always supervised by more than one control loop. This avoids single-point-of-failures and increases the dependability of the control architecture. In comparison to other control architectures the hierarchical multi-layered approach reduces the complexity of the self-adaptation process within automotive embedded systems (Zeller et al., 2009). Thus, the hierarchical multi-layered control architecture enables the extension of automotive embedded systems by self-x properties like self-configuration, self-healing and self-optimization.

6.1 Example control architecture for today's automotive embedded systems

Managing today's vehicle software systems, means managing about 270 features, running on nearly 70 different ECUs (Pretschner et al., 2007). These ECUs and various sensors and actuators are interconnected through different network buses.

Nowadays there are three major vehicle network systems (cp. Figure 6): The most common network technology used in vehicles is the *Controller Area Network (CAN)* bus (Robert Bosch GmbH, 1991). CAN is a multi-master broadcast bus for connecting ECUs without central control, providing real-time capable data transmission. FlexRay (FlexRay Consortium, 2005) is a fast, deterministic and fault-tolerant automotive network technology. It is designed to be faster and more reliable than CAN. Therefore, it is used in the field of safety-critical applications (e.g. active and passive safety systems). The *Media Oriented Systems Transport (MOST)* (MOST Cooperation, 2008) bus is used for interconnecting multimedia and infotainment components proving high data rates and synchronous channels for the transmission of audio and video data.

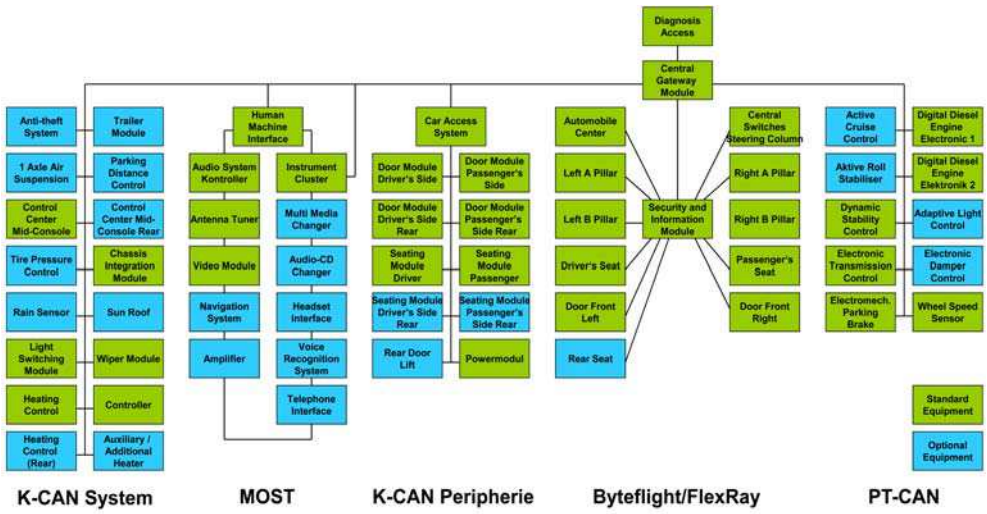


Fig. 6. In-vehicle network topology of a BMW 7-series (Source: BMW AG, 2005)

The vehicle features reach from infotainment functionalities without real-time requirements over features with soft real-time requirements in the comfort domain up to safety-critical features with hard real-time requirements in the chassis or power train domain. Therefore, various requirements and very diverse system objectives have to be satisfied during runtime.

By using a multi-layered control architecture it is possible to manage the complexity and heterogeneity of modern vehicle electronics and to enable adaptivity and self-x properties. To achieve a high degree of dependability and a quick reaction to changes, we use different criteria for partitioning the automotive embedded system into clusters (see Figure 7):

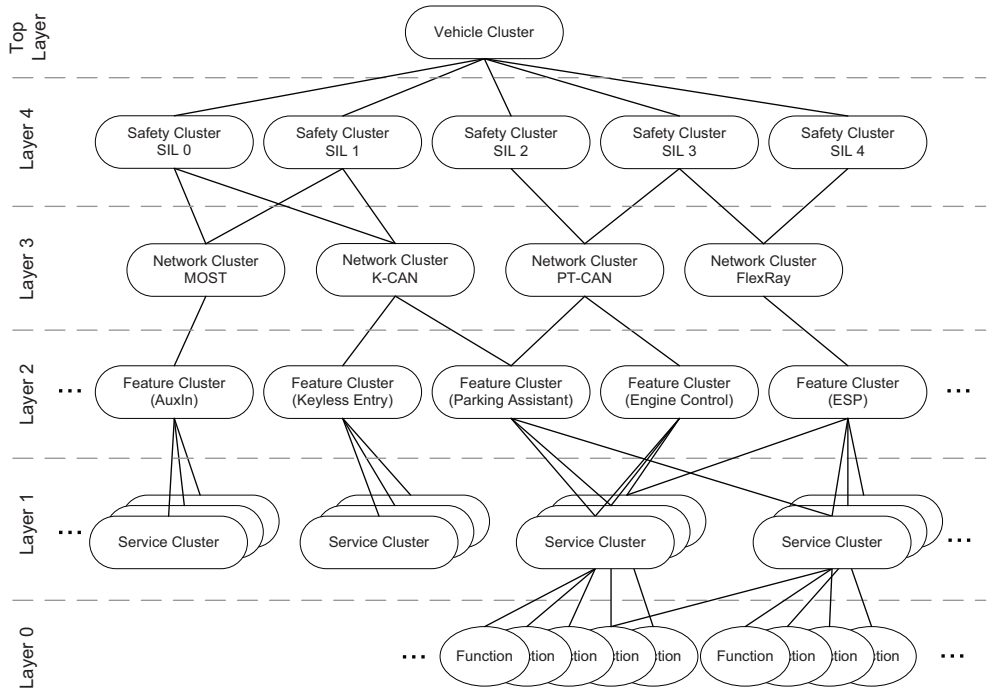


Fig. 7. Example of a hierarchical multi-layered architecture for today's automotive embedded systems

In a first step, the whole system (*Vehicle Cluster* on the top layer) is divided into the five *Safety Integrity Levels (SIL 0-4)* (International Electrotechnical Commission (IEC), 1998), because features with the same requirements on functional safety can be managed using the same algorithms and reconfiguration mechanisms. Nowadays, this classification is more appropriate than the traditional division into different automotive software domains because most new driver-assistance features do not fit into this domain-separated classification anymore.

In a second partitioning, the system is divided into the physical location of the vehicle's features according to the network bus the feature is designed for. This layer is added, so that all features with the same or similar communication requirements (e.g. required bandwidth) and real-time requirements can be controlled in the same way.

On the next layer, each *Network Cluster* is divided into the different features which are communicating using this vehicle network bus. Hence, each feature is controlled by its own control loop, managing its individual requirements and system objectives.

Most features within the automotive domain are composed of several software components as well as sensors and actuators. One example is the Adaptive Cruise Control (ACC) feature which can automatically adjust the car's speed to maintain a safe distance to the vehicle in front. This is achieved through a radar headway sensor to detect the position and the speed of the leading vehicle, a digital signal processor and a longitudinal controller for calculating

the car's deceleration. If the leading vehicle slows down or if another object is detected by the radar sensor, the system sends a signal to the braking system (the actuators) to decelerate. When the road is clear, the system will re-accelerate the vehicle back to the set speed of the cruise control.
On the bottom layer, each feature is decomposed into one or more *services* by which the feature is composed of. For example, the *ACC Feature Cluster* can be decomposed into the radar sensor service, the digital signal processing service, the longitudinal controller service and the engine and braking system services to decelerate. Each sensor, actuator and computation or controlling algorithm is a software-based function represented by a service. Each *Service Cluster* consists of either one or several software components (so-called *functions*) and represents the lowest control layer in our approach. Because services are often used by more than one feature, a service may be part of more than one Feature Cluster.
This hierarchical multi-layered control architecture provides a suitable mechanism for realizing adaptive automotive embedded systems. The requirements specified during design (see Section 5) can be enforced during runtime and self-x properties like self-healing, self-optimization or self-configuration are enabled by a control loop based approach.

7. Conclusion

The growing complexity of automotive software systems is getting more and more unmanageable. Enhancing these systems with self-x properties (e.g. self-healing or self-configuration) by self-adaptation or self-organization may overcome these problem. This increases the flexibility and efficiency of complex software systems at the same time. In this chapter, we described the domain-specific challenges in realizing self-adaptive automotive embedded systems which provide self-x capabilities. To cope with the safety and the real-time requirements of vehicular software systems, the degree of variability must be defined and uncontrolled behavior must be prevented.
This is pursued by an integrated development process which enables the verification and validation of the dynamic system behavior in iterative steps during the design. We presented our approach which incorporates the automotive domain-specific architecture description language EAST-ADL. The focus of the presented work is on enabling the validation through simulation. By this, the system and adaptation behavior realizing self-x properties can be validated. Since the designed requirements have to be met and the constraints have to be enforced at runtime, we introduced a cluster-based methodology for the runtime. For the reduction of the variability and to guarantee the predefined system behavior at runtime the satisfaction of the predefined requirements and constraints is supported by a hierarchical multi-layered control architecture. If any system requirement is not met anymore, the system is adapted to meet the constraints again. Thus, we have shown in this work the challenges of automotive embedded systems with self-x properties and presented our approach for the design and runtime.

8. References

Anthony, R., Ekelin, C., Chen, D., Törngren, M., de Boer, G., Jahnich, I. et al. (2006). A future dynamically reconfigurable automotive software system, *Proceedings of the "Elektronik im Kraftfahrzeug"*.
AUTOSAR Consortium (2010). AUtomotive Open Sytem ARchitecture (AUTOSAR).

https: //www.autosar.org.

Cai, L. & Gajski, D. (2003). Transaction level modeling: an overview, *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software Codesign and system synthesis (CODES+ISSS '03)*, pp. 19–24.

CAR 2 CAR Communication Consortium (2010). http://www.car-to-car.org.

Cattrysse, D. & Van Wassenhove, L. (1990). *A survey of algorithms for the generalized assignment problem*, Erasmus University, Econometric Institute.

Cimatti, A., Pecheur, C. & Cavada, R. (2003). Formal verification of diagnosability via symbolic model checking, *In Proceedings of the 18th International Joint Conference on Artificial Intelligence IJCAI03*, pp. 363–369.

Cuenot, P., Frey, P., Johansson, R., Lönn, H., Reiser, M., Servat, D., Koligari, R. & Chen, D. (2008). Developing Automotive Products Using the EASTADL2, an AUTOSAR Compliant Architecture Description Language, *Embedded Real-Time Software Conference, Toulouse, France*.

Czarnecki, K. & Eisenecker, U. (2000). *Generative programming: methods, tools, and applications*, Addison-Wesley.

Dinkel, M. (2008). *A Novel IT-Architecture for Self-Management in Distributed Embedded Systems,* PhD thesis, TU Munich.

Dinkel, M. & Baumgarten, U. (2007). Self-configuration of vehicle systems - algorithms and simulation, *WIT '07: Proceedings of the 4th International Workshop on Intelligent Transportation*, pp. 85–91.

EAST-ADL2 (2010). Profile Specification 2.1 RC3, http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_ 2010-06-02.pdf.

FlexRay Consortium (2005). The FlexRay Communications System Specifications Version 2.1. http://www.flexray.com/.

Fürst, S. (2010). Challenges in the design of automotive software, *Proceedings of Design, Automation, and Test in Europe (DATE 2010)*.

Geihs, K. (2008). Selbst-adaptive Software, *Informatik Spektrum* 31(2): 133–145.

Hardung, B., Kölzow, T. & Krüger, A. (2004). Reuse of software in distributed embedded automotive systems, *Proceedings of the 4th ACM international conference on Embedded software* pp. 203 – 210.

Hofmann, P. & Leboch, S. (2005). Evolutionäre Elektronikarchitektur für Kraftfahrzeuge (Evolutionary Electronic Systems for Automobiles), *it-Information Technology* 47(4/2005): 212–219.

Hofmeister, C. (1993). *Dynamic reconfiguration of distributed applications*, PhD thesis, University of Maryland, Computer Science Department.

Horn, P. (2001). Autonomic computing: IBM's perspective on the state of information technology, *IBM Corporation* 15.

IEEE (2005). *IEEE Standard 1666-2005 - System C Language Reference Manual*.

International Electrotechnical Commission (IEC) (1998). *IEC 61508: Functional safety of Electrical/ Electronic/Programmable Electronic (E/E/PE) safety related systems*.

Kephart, J. O. & Chess, D. M. (2003). The vision of autonomic computing, *Computer* 36(1): 41– 50.

McKinley, P. K., Sadjadi, S. M., Kasten, E. P. & Cheng, B. H. (2004). Composing adaptive software, *IEEE Computer* 37(7): 56–64.

Mogul, J. (2005). Emergent (Mis)behavior vs. Complex Software Systems, *Technical report*, HP Laboratories Palo Alto.

MOST Cooperation (2008). MOST Specification Rev. 3.0. http://www.mostcooperation. com/.

Mühl, G., Werner, M., Jaeger, M., Herrmann, K. & Parzyjegla, H. (2007). On the definitions of self-managing and self-organizing systems, *KiVS 2007 Workshop: Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS 2007)*.

Müller-Schloer, C. (2004). Organic computing: on the feasibility of controlled emergence, *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis,* ACM, pp. 2–5.

Open SystemC Initiative (OSCI) (2010). SystemC, http://www.systemc.org.

OSEK VDX Portal (n.d.). http://www.osek-vdx.org.

Pretschner, A., Broy, M., Kruger, I. & Stauner, T. (2007). Software engineering for automotive systems: A roadmap, *Future of Software Engineering (FOSE '07)* pp. 55–71.

Robert Bosch GmbH (1991). CAN Specification Version 2.0. http://www. semiconductors.bosch.de/pdf/can2spec.pdf.

Robertson, P., Laddaga, R. & Shrobe, H. (2001). Self-adaptive software, *Proceedings of the 1st international workshop on self-adaptive software*, Springer, pp. 1–10.

Schmeck, H. (2005). Organic computing - a new vision for distributed embedded systems, *ISORC '05: Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, IEEE Computer Society, pp. 201–203.

Serugendo, G., Foukia, N., Hassas, S., Karageorgos, A., Mostéfaoui, S., Rana, O., Ulieru, M., Valckenaers, P. & Aart, C. (2004). Self-organisation: Paradigms and Applications, *Engineering Self-Organising Systems* pp. 1–19.

Teich, J., Haubelt, C., Koch, D. & Streichert, T. (2006). Concepts for self-adaptive automotive control architectures, *Friday Workshop Future Trends in Automotive Electronicsand Tool Integration (DATE'06)*.

Trumler, W., Helbig, M., Pietzowski, A., Satzger, B. & Ungerer, T. (2007). Self-configuration and self-healing in autosar, *14th Asia Pacific Automotive Engineering Conference (APAC- 14)*.

Urmson, C. & Whittaker, W. R. (2008). Self-driving cars and the urban challenge, *IEEE Intelligent Systems* 23: 66–68.

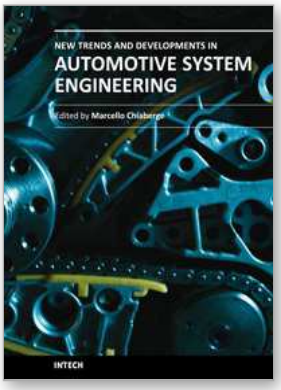
Weiss, G., Zeller, M., Eilers, D. & Knorr, R. (2009). Towards self-organization in automotive embedded systems, *ATC '09: Proceedings of the 6th International Conference on Autonomic and Trusted Computing*, Springer-Verlag, Berlin, Heidelberg, pp. 32–46.

Williams, B. C., Nayak, P. P. & Nayak, U. (1996). A model-based approach to reactive self-configuring systems, *In Proceedings of AAAI-96*, pp. 971–978.

Wolf, T. D. & Holvoet, T. (2004). Emergence and self-organisation: a statement of similarities and differences, *Lecture Notes in Artificial Intelligence*, Springer, pp. 96–110.

Zadeh, L. (1963). On the definition of adaptivity, *Proceedings of the IEEE* 51(3): 469–470.

Zeller, M., Weiss, G., Eilers, D. & Knorr, R. (2009). A multi-layered control architecture for self-management in adaptive automotive systems, *ICAIS '09: Proceedings of the 2009 International Conference on Adaptive and Intelligent Systems*, IEEE Computer Society, Washington, DC, USA, pp. 63–68.



New Trends and Developments in Automotive System Engineering

Edited by Prof. Marcello Chiaberge

ISBN 978-953-307-517-4
Hard cover, 664 pages
Publisher InTech
Published online 08, January, 2011
Published in print edition January, 2011

In the last few years the automobile design process is required to become more responsible and responsibly related to environmental needs. Basing the automotive design not only on the appearance, the visual appearance of the vehicle needs to be thought together and deeply integrated with the â€œpowerâ€ developed by the engine. The purpose of this book is to try to present the new technologies development scenario, and not to give any indication about the direction that should be given to the research in this complex and multi-disciplinary challenging field.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821