Towards Runtime Adaptation in AUTOSAR

Marc Zeller, Christian Prehofer, Daniel Krefft, Gereon Weiss
Fraunhofer ESK, Munich, Germany
{marc.zeller, christian.prehofer, daniel.krefft, gereon.weiss}@esk.fraunhofer.de

***Abstract*—In many industrial application domains networked embedded systems realize safety-critical applications. In such systems, adapting the software distribution at runtime can be used to optimize system configurations, to add new features or to handle failure cases. The main objective of this paper is to devise a flexible and efficient solution for runtime adaptation in AUTOSAR, which requires minimal changes to the current architecture. We elaborate the main challenges for extending AUTOSAR and argue that small changes in the architecture and design process are feasible and effective for this purpose. Our work is validated by a proof of concept implementation.**

I. Introduction

In many industrial application domains (e.g. railway, automotive, avionic, etc.), networked embedded systems realize safety-critical applications which have high demands on dependability. In such systems, adapting the software distribution at runtime can be used to optimize system configurations, to add new features or to handle failure cases [1].

There has been considerable work on self-adaptive systems, which can modify their software configuration at runtime [2], [3]. However, applying these techniques to networked, embedded systems poses several new problems due to the limitations and reliability requirements of embedded systems [4]. In particular, we focus on automotive embedded systems consisting of multiple Electronic Control Units (ECUs). These are interconnected locally by different bus systems and system-wide via gateways. In automotive embedded systems, runtime adaptation can be used, for instance, to increase the availability and reliability of software-based applications without additional hardware costs (e.g. in the area of fully electronically implemented functions such as steer-by-wire).

Modern runtime environments for automotive software, such as the *Automotive Open System Architecture (AUTOSAR)* [5] initiative, provide a standardized software architecture for cars and hide the heterogeneity of the underlying hardware from the application software. Current AUTOSAR systems consist of a layered architecture. This is shown in Fig. 1.[1] Each AUTOSAR based ECU implements the *AUTOSAR Basic Software* which consists of a set of the so-called *Basic Software (BSW) Modules*. These standardized software components provide an abstraction from the hardware as well as the *AUTOSAR Operating System (OS)* and a services for memory use and communication. The AUTOSAR Basic Software, especially the hardware abstraction part, is individually implemented for each ECU platform. Based on the basic software the so-called *AUTOSAR Runtime Environment (RTE)* acts as a static middleware for the *AUTOSAR Software Components (SW-Cs)*. The RTE is generated for a concrete deployment of software components to ECUs during the design. Moreover, the scheduling of the tasks by the AUTOSAR OS is defined by *Scheduling Tables* which are also generated during the development process.

With the AUTOSAR Mode Management [6], there exists a basic concept to change between predefined configurations of the RTE, the basic software components and the network communication at runtime. Responsible for switching modes is the so-called *Mode Manager*, which is either a SW-C or a BSW Module. The so-called *Mode Users* are informed of *Mode Switches* performed by a Mode Manager and may read the currently active mode. The main limitation is that all modes must be pre-configured in advance and changing the deployment of a component is not considered. These two restrictions significantly limit the adaptation to pre-planned changes.

AUTOSAR is configured statically during design time, including the mode concept. 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 supported. Hence, specific extensions to the present AUTOSAR standard are needed to enable runtime adaptation by changing the software component allocation dynamically.

There have been different approaches for enhancing AUTOSAR with reconfiguration capabilities. In [7] runtime adaptation is realized by implementing a middleware service which is implemented as AUTOSAR applications and results in a significant resource overhead. [8], [9] are using the concepts provided by AUTOSAR to enable runtime adaptation, but all possible adaptation must be designed in advanced and cannot be changed while the AUTOSAR system is running. An approach for the reconfiguration of one single-threaded AUTOSAR SW-C is presented in [10]. Thereby, an additional reconfiguration layer is added between the basic software and the application layer of the AUTOSAR architecture. This poses

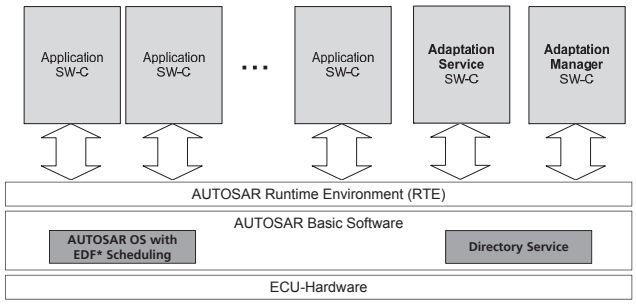

Fig. 1. AUTOSAR layered architecture with proposed extensions (shown in bold)

[1]Note that Fig. 1 also shows new components of our proposal which are discussed in Sec. III.

a certain overhead during design and runtime. In [1] a concept for enabling reallocation of software components in automotive infotainment systems with runtime adaptation is presented. Although this approach is strictly limited to the in-vehicle infotainment, it serves as a basis for our approach to enhance AUTOSAR systems by runtime adaptation.

The main objective of this paper is to devise a flexible and efficient solution for runtime adaptation in AUTOSAR, which requires minimal changes to the current architecture. In contrast to prior work, we argue that small changes in the architecture as well as design process are feasible and effective for this purpose. We present the challenges for enhancing AUTOSAR systems with runtime adaptation in Sec. II. Afterwards we present our proposed new concept and discuss the key design decisions in Sec. III. Our concept includes an Adaptation Service and an Adaptation Manager as well as changes in the AUTOSAR basic Software, shown in Fig. 1 and discussed in Sec. III.

II. Challenges of Runtime Adaptation in AUTOSAR

In the following, we give an overview of the main challenges of runtime adaptation in AUTOSAR systems. These will be addressed further in the next section.

The AUTOSAR development methodology is shown in Fig. 2. For developing an AUTOSAR-based system, the software of each ECU is generated based on information about the software component (*SW Component Description* and *SW Component Code*) as well as information about the ECUs within the in-vehicle network (*ECU Resource Description*) and the *System Constraint Description*. Using this information, a deployment model of each ECU is generated (*ECU Configuration*) which includes the mapping of the SW-Cs. Based on the description of an ECU, the source code of the RTE and the BSW Modules as well as the configuration of the AUTOSAR OS are generated.

To realize runtime adaptation in AUTOSAR systems we require a reallocation of software components at runtime and the computation of adaptations during runtime. In particular, the following specific challenges must be addressed for runtime adaptation:

Runtime Control: To apply adaptation in networked systems, a runtime control entity must supervise the system and both decide and implement the reallocation of the software components if necessary. Thus, a protocol is needed to coordinate the adaptation throughout the networked embedded system.

Real-time Requirements: The real-time system requirements of the automotive embedded system must be guaranteed during runtime. These constraints must be satisfied for the new allocations [11] and during the adaptation process itself [12].

Self-describing Components: Information about non-functional properties of the software components and the available hardware resources must be made available during runtime, see e.g. [13]).

Scheduling: Real-time capable task scheduling with dynamic priorities must be provided in order to enable efficient planning of new allocations during runtime [11].

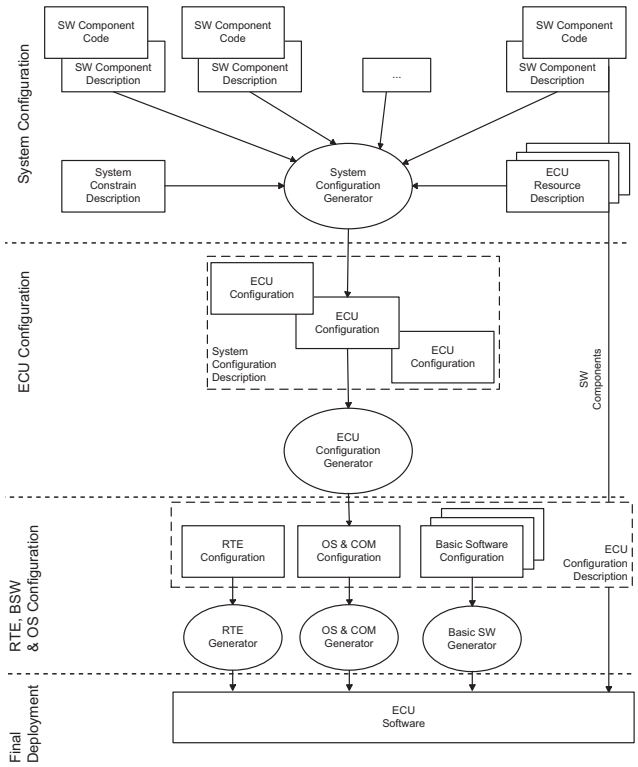

Fig. 2. AUTOSAR development methodology

ECU-independent Addressing: Since the allocation of software components is changed during runtime, software components must be addressed independently from their physical location within the system's communication network.

III. A Concept for Enabling Runtime Adaptation in AUTOSAR

In the following, we present our concept for adaptation in AUTOSAR. The main new components are shown in Fig. 1. We explain our architecture concept by addressing the above challenges in the following subsections.

1) Runtime Control: In order to reallocate software components, we must be able to manage the start and stopping of AUTOSAR software components (SW-C) dynamically. For this purpose, we extend the system by a *runtime control* based on the so-called *MAPE cycle* [14]. This control loop consists of the four stages: Monitor, Analyze, Plan, and Execute. Moreover, the existing concept of the AUTOSAR Mode Management can be reused to implement runtime adaptation.

Fig. 1 illustrates our approach. Thereby, the runtime control is implemented by two components: the *Adaptation Service* and the *Adaptation Manager*. The Adaptation Service implements the monitoring and execution stage of the MAPE cycle (see Fig. 3). It supervises the current status of the system's environment by monitoring the equipped sensors as well as the status of the applications running on the ECU. Moreover, it implements the actual activation and deactivation of specific SW-Cs. This can be done by changing the scheduling table which is used by the AUTOSAR OS. In order to deactivate a software component, the corresponding entry is removed from

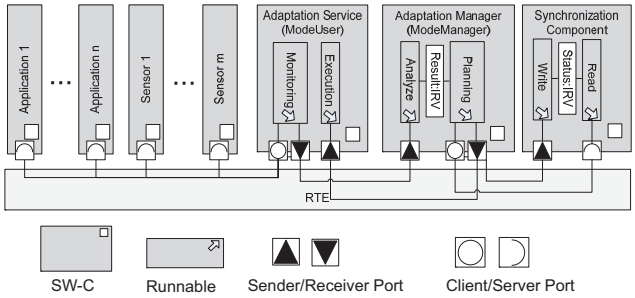

Fig. 3. Runtime control for reallocating AUTOSAR SW-Cs at runtime

the scheduling table. A software component is activated by adding a corresponding entry to the scheduling table. The Adaptation Service extends the Mode User concept of the AUTOSAR Mode Management and can either be implemented as a BSW module or as SW-C.

After the monitoring stage is completed, a *ModeChangeRequest* is sent to the Mode Manager. The Adaptation Manager implements the analyze and planning phase of the MAPE cycle (see Fig. 3). It encapsulates the computation intensive parts of the MAPE cycle. For our concept it would be sufficient to have one single Adaptation Manager within the whole automotive embedded systems. But in order to reuse the concepts provided by the AUTOSAR Mode Management for implementing the runtime control, there must be one Adaptation Manager implemented on each ECU. Thereby, the Adaptation Manager extends the Mode Manager concept and can either be realized as a BSW module or as SW-C. It analyses the current system state based on the data provided by the monitoring part of the Adaptation Service and decides whether the allocation of software components must be modified. Based on the decision taken in the analyze stage, the Adaptation Manager determines a new allocation of software components which satisfies the given system requirements. The newly determined allocation is then executed by activating or deactivating specific software components through the Adaptation Service. Therefore, a *ModeSwitchNotification* is sent to the Adaptation Service.

Since each ECU must provide a local Adaptation Manager, adaptation is managed locally and a coordination mechanism is needed. For this purpose, a new *Synchronization Component* is implemented on one single ECU within the in-vehicle network. After planning a novel allocation of software components, the Adaptation Manager requests the current allocation from the Synchronization Component. If the allocation stored in the Synchronization Component is not equal to the current allocation known to the Adaptation Manager, the Adaptation Manager executes the adaptation described by the Synchronization Component. If not, the Adaptation Manger communicates the newly planned allocation to the Synchronization Component and starts executing this configuration. In case the Synchronization Component receives a changed allocation of software components, it sends this new configuration to all Adaptation Managers within the in-vehicle network.

2) Guaranteeing Real-time Requirements: Since automotive embedded systems implement safety-critical applications, the expected system behavior must be guaranteed at any time. Especially the non-functional system requirements (like the real-time behavior) must be met in any system configuration. Thus, the mechanisms used to determine a new allocation in the planning stage of the runtime control must be based on a representation of the non-functional system requirements (e.g. [11]). Moreover, the correct system behavior must be guaranteed during the actual adaptation of the system. Hence, the sequence of activating or deactivating specific software components on the systems' ECUs must also be determined w.r.t. the components' timing requirements before executing the newly planned allocation [12].

3) Self-describing Components: In order to enable runtime adaptation, the control mechanisms must be aware of the system properties. Hence, the attributes of the software components and ECU platforms used to describe the system during the AUTOSAR development process (see Fig. 2) must be available for the different stages of the MAPE cycle during runtime. Especially the non-functional requirements of the software components which are needed to characterize to correct, predefined system behavior must be available for planning and executing changes of the software component allocation without influencing the actual applications negatively (cf. Sec. III-2). The attributes needed during runtime comprise of information from the AUTOSAR *SW Component Descriptions*, the *ECU Resource Descriptions* as well as the *System Constraint Descriptions* (e.g. timing requirements, such as end-to-end deadlines). This information must be available to the Adaptation Service and the Adaptation Manger which are implemented on each ECU within the in-vehicle network. Therefore, the data necessary to enable runtime adaption can either be stored on each ECU or in a central component located on one single ECU within the network. While storing the complete information on each ECU leads to a certain memory overheard, a centralized storage leads to additional network traffic and may result in a performance bottleneck or a single-point-of-failure. Thus, a trade-off is needed: The information which is needed frequently should be stored on each ECU, while other data may be stored in a centralized way or on an ECU within each sub-network of the automotive embedded system.

4) Dynamic Task Scheduling: In AUTOSAR, a real-time capable scheduling mechanism is needed to guarantee the timing requirements of each task. Currently, it is using static priorities, assigned to the tasks at design time. Changing the static scheduling would lead to an overhead in planning new allocations at runtime. This is due to the complex schedulability analysis needed to guarantee correct system behavior in new configurations [11]. Moreover, the scheduling table of the AUTOSAR OS must be modified in order to execute the planned adaptation. Thus, a real-time capable scheduling mechanism based on dynamic priorities needs to be used by the AUTOSAR OS to coordinate task execution during runtime. Thereby, the priority of each task is calculated dynamically during runtime more efficiently. In AUTOSAR such a mechanism must be able to schedule a set of preemptive (a-)periodic tasks with offsets. For this, the *Earliest Deadline First (EDF)** [15] algorithm can be applied to schedule tasks with real-time requirements in an AUTOSAR system based on dynamically calculated priorities. Although, calculating the task priorities during runtime leads to an overhead, planning and execution of a new allocation can be performed much more efficiently.

5) Locations-independent Addressing: Applying dynamic reallocation of software components to different ECUs in distributed systems means that SW-Cs change their physical location at runtime. In order to enable the communication between different software components, each software component must be addressed independently from their physical allocation within the system's communication network. Thus, each software component is addressed network-wide by a unique identifier. This identifier must be translated to the physical address of the hardware platform, where the SW-C is currently allocated. Thus, a so-called *Directory Service* can be used to map the location-independent identifier to a physical address. Such a Directory Service can be integrated into the AUTOSAR communication stack as a BSW module.

The actual mapping can be implemented efficiently using a look-up table. In order to transfer data correctly at any time, this look-up table must be modified according the adaption of the software component allocation. Therefore, all ECUs must be informed in case of an adaption. This is already done by the previously introduced Synchronization Component (see Sec. III-1). In case the Adaptation Manager receives a notification with a new allocation, the lookup-table within the Directory Service is adapted accordingly. Since no additional message is needed, this can be done very efficiently.

IV. Proof of Concept Implementation

In [16] the concepts described in Sec. III were implemented using *Arctic Core*, an open-source implementation of the AUTOSAR standard 3.1. A first evaluation was performed using two prototyping ECUs based on the *Freescale MPC5554* @135MHz which are interconnected using a CAN-bus. ECU 1 is hosting two applications: *Lane Departure Warning System* and *Parking Aid System*. To elaborate the implemented concepts, one of these applications is reallocated on the other ECU on which the Synchronization Component is implemented. The results of this evaluation shows that the runtime control described in Sec. III-1 can be implemented very efficiently w.r.t. to the memory overhead for the additional software components (4%-5% of the memory requirement by the overall system compared to the same AUTOSAR system without our extensions). Moreover, the evaluation shows that changing the set of SW-Cs running on an ECU can be performed very fast using the concepts described in this paper (about 23 μs to adapt one single ECU from the detection of a changed situation until the planed modification was executed). However, the overall time needed for the actual runtime adaptations mostly depends on the time-consuming data transfer in today's automotive networks. This is needed in order to synchronize all ECUs within the automotive embedded system during the adaptation process to guarantee a consistent system state at any time. Thus, the overall time required for the runtime adaptation of the automotive embedded system in our case study is measured as approximately 7.5 ms.

V. Conclusion and Future Work

We have identified the main challenges of enhancing AUTOSAR for runtime adaptation w.r.t. the placement of the software components, which is important for optimization and self-repair of networked embedded systems. Based on this, we have presented a flexible solution for runtime adaptation in AUTOSAR, which is also efficient and minimal changes to the current architecture. We show the needed AUTOSAR extensions and argue that small changes in the architecture as well as the design process are feasible and effective for this purpose. Our work is validated by a proof of concept implementation, which shows small overhead in our initial evaluation. Further work will encompass a more detailed evaluation of the new approach. Moreover, functional safety aspects of our approach to enable runtime adaptation in automotive systems will be investigated.

References

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

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

[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)*, 2002, pp. 33–38.

[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 2011 ACM Symposium on Applied Computing*, ser. SAC'11, 2011, pp. 236–237.

[5] AUTOSAR Consortium, "AUtomotive Open Sytem ARchitecture (AUTOSAR)," http://www.autosar.org/.

[6] AUTOSAR Consortium, "Guide to Modemanagement," 2011, http://www.autosar.org/.

[7] W. Trumler, M. Helbig, A. Pietzowski, B. Satzger, and T. Ungerer, "Self-Configuration and Self-Healing in AUTOSAR," in *Proceedings of the 14th Asia Pacific Automotive Engineering Conference (APAC-14)*. SAE International, 2007.

[8] B. Becker, H. Giese, S. Neumann, M. Schenck, and A. Treffer, "Model-based extension of autosar for architectural online reconfiguration," in *Proceedings of the 2nd International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES-MB 2009)*, 2009, pp. 123–137.

[9] J.-C. Fabre, M.-O. Killijian, and F. Taiani, "Robustness of automotive applications using reflective computing: lessons learnt," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, ser. SAC'11, 2011, pp. 230–235.

[10] C. Berger and M. Tichy, "Towards transactional self-adaption for autosar on the example of a collision detection system," in *Proceedings of the INFORMATIK 2012*, 2012, pp. 853–862.

[11] 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 (SASO)*, 2011, pp. 79–88.

[12] M. Zeller and C. Prehofer, "Timing constraints for runtime adaptation in real-time, networked embedded systems," in *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '12)*, 2012, pp. 73–82.

[13] G. Weiss, K. Becker, B. Kamphausen, A. Radermacher, and S. Gerard, "Model-driven development of self-describing components for self-adaptive distributed embedded systems," in *Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, 2011, pp. 477–484.

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

[15] H. Chetto, M. Silly, and T. Bouchentouf, "Dynamic scheduling of real-time tasks under precedence constraints," *Real-Time Systems*, vol. 2, pp. 181–194, 1990.

[16] D. Krefft, "Konzeption und Implementierung einer Laufzeitumgebung für Selbstadaption in automobilen E/E-Systemen," Master's thesis, University of Augsburg, 2012.