On the Conceptual Design of a Dynamic Component Model for Reconfigurable AUTOSAR Systems

Jakob Axelsson
Swedish Institute of Computer Science (SICS)
Kista, Sweden
jakob.axelsson@sics.se

Avenir Kobetski
Swedish Institute of Computer Science (SICS)
Kista, Sweden
avenir.kobetski@sics.se

***Abstract*—The automotive industry has recently developed the embedded software standard AUTOSAR, which is now being introduced widely in production vehicles. The standard structures the application into reusable components that can be deployed in a specific vehicle using a configuration scheme. However, this configuration takes place at design time, with no provision for dynamically installing components to reconfigure the system. In this paper, we present the conceptual design of a dynamic component model that extends an AUTOSAR based control unit with the possibility to add plug-in components that execute on a virtual machine. This concept is intended to give benefits in terms of much shorter deployment time for new functions, even into vehicles that have already been produced. Further, it creates opportunities for vehicles to take part in federated embedded systems together with other products. It also opens up a market for third-party developers, and fosters open innovation in an ecosystem around the automotive software business.**

Keywords—component-based software; federated embedded systems; automotive systems; AUTOSAR; reconfigurable software

I. INTRODUCTION

Over the last few decades, automotive embedded systems have expanded rapidly in functionality and complexity. Today, a car typically contains several dozen electronic control units (ECUs) that are connected through communication networks. Each ECU runs control functionality using sensors and actuators, but there are also control functions that are distributed over several ECUs. It is common in the vehicular industry to rely on external suppliers for the development of both ECU hardware and software, and with rising complexity, it has become increasingly costly to integrate these ECUs into a functioning system.

To cope with the increasing complexity of in-vehicle software, the automotive industry has for the last decade been developing the standard Automotive Open System Architecture (AUTOSAR) [2]. It decouples the basic software that needs to exist in all ECUs and can be standardized, from the application software. It also provides a component model that eases reuse of parts of the application software, and allows it to be redistributed if the underlying distributed hardware architecture is changed, thereby improving flexibility and scalability. It is, according to the AUTOSAR consortium, already in use in 25 million electronic control units (ECUs) in 2011, a figure expected to rise to 300 million in 2016.

The automotive industry is very cost sensitive, and ECU hardware resources are traditionally kept to a minimum. Therefore, AUTOSAR has been designed to execute with limited resources and hence configuration of the system is done at design time with no structural dynamics during execution. However, more powerful hardware can be expected to be used in the future, since there are limits to how many ECUs are sensible to put in a vehicle. This would open up for adding sufficient resources to introduce also the possibility for dynamic reconfiguration of parts of the system.

The purpose of this paper is to describe the conceptual design of a component model that makes it possible to dynamically reconfigure parts of an AUTOSAR system by allowing for plug-in components on top of the statically configured software. Such a dynamic model would have several benefits. Firstly, it would drastically decrease the time to market since software can be added or modified very late in the development process. Secondly, in combination with external wireless communication, it gives the possibility for creating federated embedded systems [5], i.e., embedded systems in different products that cooperate with each other. Thirdly, it would create a foundation for open innovation where an ecosystem of third party developers can develop new services that add to the value of the products. In the next section, a brief overview of AUTOSAR will be given, followed by a description of the dynamic component concept which is the key contribution of this paper. In Section IV, related work is presented, and the paper finishes with conclusions and directions for future research.

II. OVERVIEW OF AUTOSAR

AUTOSAR is structured around a layered software architecture that contains three levels: the basic software (BSW), a middleware called the runtime environment (RTE), and the application software (ASW).

The BSW consists of an operating system that has evolved from the OSEK standard; system services for, e.g., memory management; communication concepts; ECU and microcontroller hardware abstractions; and complex device drivers for direct access to hardware.

On top of the BSW is the RTE. It manages all communication between ASW components, as well as their access to the lower layers. To make the ASW components independent of their physical allocation to different nodes, a

concept called the virtual functional bus (VFB) is used, which allows ASW components to communicate between each other as if they were all allocated to the same ECU. If they are in fact on different ECUs in a particular implementation, the communication between them has to be mapped to network messages, and this is taken care of by the VFB. The actual implementation of the RTE is done by generating ECU specific software from a description of how the constituent ASW components are allocated to ECUs and what links between the components exist. The result is thus a C program that provides an API to the ASW, and that in turn calls the API of the BSW. Apart from communication, the RTE also handles other functionality, such as events, critical sections, etc.

The ASW consists of a number of software *components* (SW-C) which are in many ways similar to established component models like Koala [6]. Each component declares a number of *ports*, which can be either *required* ports (where the component is expecting input) or *provided* ports (that the component uses for its output). The ports can implement different interaction schemes, including sender-receiver or client-server. The internal functionality, or the *runnable*, of the component only accesses its ports, and not any other components. The runnables are mapped to OS tasks. SW-Cs can also be composite, i.e., containing other SW-Cs inside.

In addition to technical concepts, AUTOSAR also provides an outline of a development methodology which heavily relies on different tools for software configuration. Since the intended use is software for resource-constrained embedded systems, the approach is to do all configuring statically at design time instead of dynamically at run-time. This is achieved through a number of description files (using primarily XML format) that are processed by different tools. These description files include, among many other things, information about how the ports of different SW-Cs are connected to each other to form a system, and how SW-Cs are allocated to ECUs. From these description files, executable software is generated that implement the BSW, RTE, and ASW for a particular ECU.

Although AUTOSAR provides a lot of flexibility in reconfiguring a system, it is important to notice that it occurs at design time. It does not offer any possibility to make dynamic additions, but any changes require the software to be rebuilt and the ECU to be reprogrammed. The key contribution of this paper is thus to describe a reconfigurable software concept that extends AUTOSAR.

III. A Dynamic Component Concept

We will now describe a concept for allowing reconfiguration of AUTOSAR based embedded systems. The intended application area is automotive control systems, and these are usually not only resource-constrained but also safety critical. Since we are striving for a concept where other parties than the OEM can develop additional software, there is no way to avoid that software of different criticality is mixed. Therefore, we find it wise to separate the built-in software from the additional software, and execute the additional software as plug-ins in a virtual machine (VM). The key idea is thus to encapsulate a VM in an AUTOSAR SW-C, and by installing that SW-C into an ECU, it becomes plug-in enabled. In that way, no changes are needed to the AUTOSAR standard, and it becomes a very easy process to make a system plug-in enabled.

Figure 1 gives an overview of how the plug-in concept relates to the underlying AUTOSAR based software. In the figure, dotted lines are used to show the plug-ins and their connections, whereas solid lines are used for the AUTOSAR SW-Cs and their links. In the following subsections, different parts of the concept will be detailed.

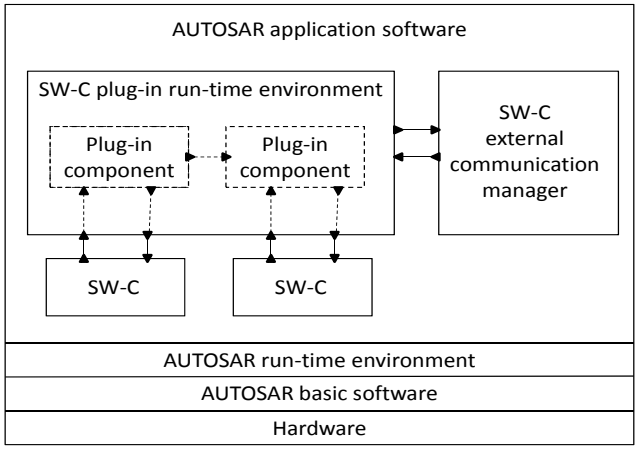

Fig. 1. The structure of an AUTOSAR based ECU with the plug-in concept.

A. Plug-In Component Concept

To make the transition between AUTOSAR ASW and plug-ins as transparent as possible, a similar component model is used for plug-ins, and most of the concepts available to SW-Cs are also present in plug-in software. However, the full range of services available to plug-in components depend on what ports the developers of the underlying system choose to make available to the plug-ins, which is a decision that is still made during design of the ECU.

A difference between ordinary AUTOSAR components and plug-in components is also that whereas the typical language used for ASW is C, Java will primarily be used for the plug-ins since the execution environment will be based on the Java VM. (Technically, other VM technology could also be used, but we have chosen Java due to its wide availability.) All ports available to plug-ins will therefore have their direct correspondence in a Java API that is available to plug-in developers.

In AUTOSAR, allocation of SW-Cs to ECUs, and connection between ports of SW-Cs, are described in configuration files separate from the source code. However, for plug-ins, this has to be handled dynamically on-line, and therefore allocation and connection are included in an initialization method of the plug-in components.

The concept also allows a plug-in to define ports that other plug-ins can connect to. This means that the full range of services available to a new plug-in is those of the underlying ECU plus those of all other plug-ins already installed.

B. Plug-In Run-Time Environment

To execute the Java plug-ins, a Java VM has to be installed in the ECU. In our concept, this VM is packaged in a generic AUTOSAR SW-C called the plug-in run-time environment (PIRTE). It connects to the ASW through a set of ports which have to be defined by the user at design time. Those ports are then replicated in a Java library which is available to the plug-in components. Apart from the port configuration, the PIRTE is general and can be installed into any ECU with sufficient resources, to make it plug-in enabled.

C. External Communication Manager

In addition to PIRTE, one more generic SW-C is provided in the concept, which is the external communication manager (ECM). It should be installed on an ECU which has a communication link to the outside (usually a wireless link to the Internet). It contains the mechanisms needed to download plug-ins, but also serves as a gateway for plug-ins to communicate externally. The latter is important, since many plug-ins are expected to be used for transferring information to and from off-board services, and for building federated embedded systems.

D. Plug-In Installation

When a new plug-in application has been downloaded into the vehicle, different parts of the plug-in must be distributed to the correct ECUs, and connections between its ports and the required and provided ports of the application software have to be put in place.

To handle this, each plug-in application is at the top level divided into a set of plug-in software components, in such a way that there is (at most) one component to be installed in each ECU. In most cases, those components will in practice be composite components, containing other plug-in software components inside.

The following steps are performed to install and configure a new plug-in application:

1. The user instructs the vehicle to install a certain plug-in application in the vehicle.
2. The vehicle downloads the plug-in application over the external connection from a pre-defined trusted server.
3. The ECM executes the allocation instructions of the plug-in application. This will have the effect that the plug-in packages are transferred to the ECUs where they should be installed, and stored in FLASH memory.
4. Each ECU that has received a package reconfigures its PIRTE by executing the connection instructions related to that package. This will have the effect that all ports are connected to other ports in the PIRTE, or in other plug-in applications. This part has to be executed every time the application is starting, including when the ECU is restarted.
5. When installation has completed, an acknowledgement is sent back to the trusted server. In this way, the site can keep track of the status of installed plug-ins, making it possible to restore the contents of an ECU in case of hardware replacement, etc.

E. Internal Communication

Since most automotive embedded systems are distributed, it will often be necessary to have several ECUs plug-in enabled, simply because the data the plug-ins need is only available in those ECUs. This also entails that there is a need for communicating plug-in data between ECUs. To handle this, generic messaging ports are provided in all PIRTEs, which allow them to connect to each other. Note that the communication details are transparent to the plug-ins, allowing to easily re-allocate them by simply reconfiguring appropriate PIRTEs. Since the PIRTEs are allocated on different ECUs, the AUTOSAR run-time environment needs to be set up to provide appropriate network messages to implement the communication between those ports. In a similar way, the ECM is connected to all PIRTEs, to provide channels for downloading plug-ins and for external data communication.

F. Safety and Security

A key issue when allowing reconfiguration of embedded systems by installing external software is how to ensure safety and security. One important concept is the use of a VM embedded in an SW-C for executing the plug-ins. This means that plug-ins can only access the underlying software and hardware through the ports of that SW-C. It is up to the ECU developers to decide which ports should be connected, and in the case of provided ports of the SW-C, how data received from the plug-in should be handled. If that data is used to control the underlying system, it is important that (non-reconfigurable) fallback mechanisms, that monitor safety risks and have the authority to override the plug-in actions, are included in the functionality of the ASW.

The PIRTE executes in its own thread and with its own memory areas and network messages. This means that CPU, memory, and bandwidth is pre-allocated to plug-ins, and there will not be a competition for resources with the built-in functionality. Plug-ins are thus executed under a best effort scheme, whereas built-in software has predictable behavior.

A potential security threat is the installation of plug-ins. In this concept, it is only allowed to install plug-ins from a trusted server at a pre-defined address. In this way, much of the firewall issues are moved from the resource-constrained embedded system to a server, where all appropriate mechanisms can be implemented. To change the trusted server address requires reprogramming of the ECU's built-in software, which has its own security mechanisms.

G. Tools for Configuration

To enable plug-ins in an ECU is a step which can to a large extent be automated through the use of tools. Given an ECU with an AUTOSAR application consisting of a set of SW-C's and ports, the AUTOSAR configuration files contain, in XML format, information about these ports. That configuration file can be read into a new tool, the Plug-in configurator, which is part of our concept. In this tool, the user manually selects which of all the available ports should be visible for plug-in applications on that ECU. The information entered into the tool

is stored in a separate configuration file to be able to repeat the process. Based on the AUTOSAR configuration files and the information about visible ports, the following files are generated:

1. An AUTOSAR configuration file describing the ports of the PIRTE SW-C.
2. An AUTOSAR configuration file describing how the ports of the PIRTE SW-C are connected to the ports of other SW-C's in that ECU.
3. A Java source code file which contains the visible ports described as Java objects, where the internals of those ports are calls through the Java Native Interface to the AUTOSAR RTE of that ECU. The source file can be used by developers to define how their plug-ins connect to the visible ports. A compiled version of the source file is pre-installed in the PIRTE.

IV. Related Work

Several well-established component-based frameworks exist for embedded real-time systems, such as Koala [6] and SaveCCM [4]. They are both similar to the AUTOSAR model in principle, and thus only provide static configuration of the system.

Other researchers have investigated the use of Java in AUTOSAR based systems, such as the KESO compiler [9]. However, this solution generates native code for each ECU, and is thus less suitable for dynamic reconfiguration.

Dynamic reconfiguration in automotive systems has been studied in the DySCAS research project [1]. It defines a completely new architecture, with focus on mechanisms for self-reconfiguration. This however adds a lot of complexity to the architecture, which is avoided in our work with a more restricted and pragmatic approach based on plug-ins. However, it could still be viable in certain areas, such as infotainment.

Outside the automotive domain, several component based systems with dynamic reconfiguration mechanisms have been reported. In [3], an approach is described that focuses on state-preserving updates of resource-constrained nodes. However, it is not based on a VM, but requires node-specific binaries to be generated.

The problem of specifying the dynamic reconfiguration process is addressed in [7], where a script language is used for describing the steps to perform. Although it is applied to the reconfiguration of an OS kernel, similar ideas can be applied to also reconfigure the setup of plug-in installations.

A Java VM based approach is SEESCOA [8], which defines its own component model. It deals, among other things, with issues related to transferring the state of components from the old version to a new version during upgrades. This issue is dealt with more pragmatically in our approach, by mandating a plug-in to be stopped before being updated, and then restarted fresh.

This work differs from all the above in that it provides dynamic installation of software components in an AUTOSAR based control system, executing the plug-ins in a Java VM.

V. Conclusions and Future Work

In this paper, we have presented the conceptual design of a reconfiguration mechanism that makes it possible to dynamically add software plug-ins to an AUTOSAR based system. The solution is packaged into two generic SW-Cs that can very easily be included in a system to make it plug-in enabled. The plug-in concept will give many benefits to the development of automotive software, and make it possible to create federated embedded systems. Although the standard is from the automotive industry, the concepts are quite general and of value in other embedded system domains.

While in theory our design can be used for mixed-criticality applications, currently, our work is mainly focused on non safety-critical plug-in examples, running on dedicated resources in a best-effort way, while the safety-critical functionality of the underlying system is untouched. The concepts have been validated in a desktop simulation environment, whereas an implementation on embedded hardware is underway. The embedded implementation will provide additional insight into the performance of the solution, and allow further refinement of the concept with regards to issues like predictability of plug-ins. Also, developing more plug-in examples will allow a deeper analysis of what parts of Java are actually useful in this setting, possibly leading to VM optimization towards even less resource consumption. This will also allow studying how plug-ins can interact, and what further mechanisms are needed to make such interactions robust and predictable, opening up for safety-critical plug-in functionality.

Acknowledgements

This project is supported by Vinnova (grant no. 2012-02004), Volvo Cars, and the Volvo Group.

References

[1] R. Anthony, A. Rettberg, D. Chen, I. Jahnich, G. de Boer, and C. Ekelin, "Towards a dynamically reconfigurable automotive control system architecture," In Embedded System Design: Topics, Techniques and Trends, pp. 71-84, Springer 2007.

[2] AUTOSAR consortium, www.autosar.org.

[3] M. Felser, R. Kapitza, J. Kleinöder, and W. Schröder-Preikschat, "Dynamic software update of resource-constrained distributed embedded systems," In Embedded System Design: Topics, Techniques and Trends, pp. 387-400, Springer 2007.

[4] H. Hansson, M. Åkerholm, I. Crnkovic, and M. Törngren, "SaveCCM – a component model for safety-critical real-time systems," In Proc. Euromicro Conf., pp. 627-635, Aug. 2004.

[5] A. Kobetski and J. Axelsson, "Federated robust embedded systems: Concepts and challenges," SICS Tech. Report T2012:05, 2012.

[6] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee, "The Koala component model for consumer electronics software," IEEE Computer, March 2002.

[7] J. Polakovic, S. Mazaré, J-B. Stefani, P-C. David, "Experience with safe dynamic reconfigurations in component-based embedded systems," In Proc. 10th Intl. Symposium on Component-Based Software Engineering, pp. 242-257, Lecture Notes in Computer Science, Springer, 2007.

[8] Y. Vandewoude and Y. Berbers, "Run-time evolution for embedded component-oriented systems," In Proc. Intl. Conf. on Software Maintenance, pp. 242-245, 2002.

[9] C. Wawersich, I. Thomm, and M. Stilkerich, "The use of Java in the context of AUTOSAR 4.0," Embedded World, Nuremberg, Germany, March 1-3, 2011.