Dynamic Software Product Line Architectures Using Service-Based Computing for Automotive Systems

Hesham Shokry, M. Ali Babar
Lero, University of Limerick, Ireland
{hesham.shokry, malibaba}@lero.ie

Abstract

Our research is aimed at applying the notion of dynamic runtime variability of software product lines in the embedded automotive software systems to create adaptable and reconfigurable software architectures. We argue that Service-Oriented Architecture (SOA) can be used to describe and compose the software architecture of an in-vehicle Distributed Real-time Embedded System (DRES) software application. This paper describes how the SOA is, in general, an appropriate architectural style for automotive DRES and in particular it has the potential to help achieve the run time variability in product line architectures. The architecture of such a product line is composed of a set of interacting services. These services are "adaptively" connected together with adaptive connectors. The variability of the product line architecture lies and managed within these connectors.

1. Introduction

Automotive systems differ from other Distributed Real-time Embedded Systems (DRES) by having unique characteristics and development challenges. OEMs[1] face problems such as:

- high demand of customization driven by market competition and end-user preferences,
- multiple vendors are supplying individual subsystems which are being developed by different processes and engineering technologies,
- inherently incompatible software applications running on these individual subsystems due to the diversity of those vendors' development cultures,
- and unreliability of these software applications, due to the high pressure on vendors for fast time-to-market delivery, which resulted in high cost incurred by OEMs for vehicles recalls and maintenance.

Moreover, the automotive systems usually run in highly dynamic environments characterized by several kinds of changes such as runtime resources changes due to failures or changes in hardware/software components, ambient environment changes (due to system deployment in different geographic locations), changes in regulations and laws, and even changes in the telematics services as cars travel from one location to another.

In order to overcome these challenges, sophisticated and rigorous engineering technologies for developing these systems have to be developed and standardized. Such technologies are expected to enable the creation of compatible systems which are easy to integrate with each other. That is why standardization efforts like AUTOSAR [1] are being promoted as an important step toward achieving these goals. Moreover, these systems need to possess some autonomy to evolve and adapt to changes in their environments.

Software Architecture is an effective technology to model software systems at a high level of abstraction that enable early reasoning about functional and nonfunctional requirements of a system. Software architectures built from a set of *principal* architectural components that can be connected together via *connectors* with a suitable *architectural style* [6].

When it comes to designing adaptable systems, architectural style and connectors are important players in introducing adaptability. Service-Oriented Architecture (SOA) is considered an effective way of designing adaptable systems that can be (self-) reconfigurable for adapting to contextual changes.

The term "service-oriented" has existed for some time [2]. SOA approach is becoming increasingly popular to integrate highly heterogeneous systems. It defines a distinct approach of coordination for integrating components (services) in unstable and evolving context [14]

In this position paper, we introduce SOA as an effective architecture style that can support the creation of dynamically reconfigurable product line architectures.

In addition to the architecture style, adaptable software systems development also means to realize runtime re-configurability of an application. There is a need to model such re-configurability in the software architecture.

Dynamic Software Product Lines (DSPL) appear to be an appropriate way of achieving the re-configurability in which variability in software features is modeled in the

[1] Original Equipment Manufacturer

architecture at design time, and bound at runtime. *This means that a dynamic product line architecture that has its variability points bound at runtime can be considered as an adaptive architecture*. A more sophisticated approach is to enable runtime discovery and binding of variability during online operations and the reasoning logic is included on-board (self-adaptable) [3]. However, we assert that such approach is not currently practical for a safety-critical and resource-constrained automotive system. In this paper, we describe how adaptability that is an integral characteristic of DRES can help address the challenges in designing highly complex and dynamic solutions for such systems. We proposed a solution based on the idea of developing dynamic product line architectures for automotive systems using service-oriented approach. We also demonstrate the proposed approach by providing a practical example of applying it in the automotive domain.

2. Background and Motivation

Mission-critical DRES such as in-vehicle electronics system have evolved from vertically-integrated to be horizontally-integrated systems in which a system's main building blocks (subsystems) are supplied by multiple vendors based on de-facto standard and specifications set by industry leaders. A related example of such standards in the automotive domain is AUTOSAR [1] initiative.

From technological point of view, system architecture has changed from being federated to being integrated; and recently it has evolved into what is called System of Systems (SoS) [4]. Federated system architectures are usually characterized by:

- Set of subsystems, each of which is dedicated to a specific purpose.
- Loose coupling between these subsystems.

These characteristics lead to the partitioning of the requirements of and constraints on the main embodied system (the whole vehicle system) into a set of isolated subsystems with little or no coupling between them. This kind of architecture no longer supports the complexity and resources constraints requirements.

Integrated system architectures are considered the next step toward addressing these challenges. The integrated system architectures are usually characterized by:

- Extensive resource sharing, ranging from sensors to processors and communication channels.
- Reuse of data and functionality, instead of implementing redundant and duplicate computations in different subsystems.

However, the current architecture design technologies and tools employed in developing these systems are still not able to address these challenges in safety-critical and resource-constrained environments. For example, In-car electronics systems are getting highly complex as customers demand more features, which result in more and more software-based functions to be integrated in a car. Subsequently, this results in strong interdependencies between various subsystems in a car such as the Locking Control, Airbag Control, Climate Control, Cruise Control, and Braking Control. Such interdependency may come in the form that for one subsystem to complete its mission, it needs to get data from a sensor which is connected to another subsystem; hence it needs to request the required data from that subsystem. An example of such a case is the Climate Control subsystem, where it needs a car's speed information in order to adjust the airflow inside the car (and hence adjusting the in-car temperature).

Another more safety related example is that the Cruise Control subsystem, in case it requires accelerating or decelerating the car, it needs to access the fuel injection-rate facility (which is part of the Engine Management subsystem) in order to control the car's speed and acceleration. In such a case, the subsystem not only needs information, but also requires an action to be taken. This scenario highlights the aspect of QoS which is very important for architecture analyses. Hence, it appears that every subsystem requires a sporadic or periodic access to other functions and data, which reside on other subsystems, in order to perform its functionality. Figure 1 shows how an emerging industry standard, AUTOSAR [1], is expected to provide interoperability between various subsystems in automotive domain.

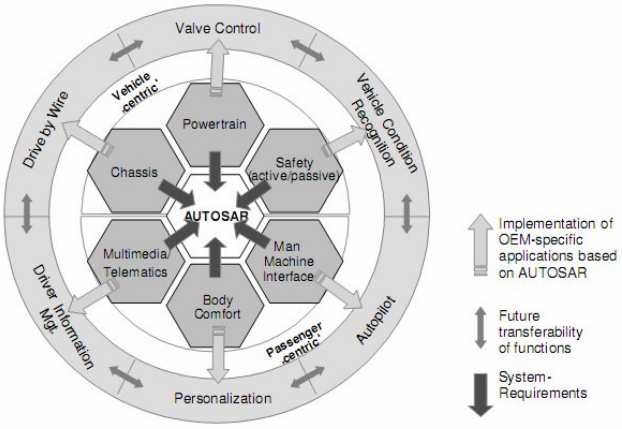

Figure 1: Services transferability in AUTOSAR [1].

An undesirable solution to avoid the interdependencies is to centralize the in-car control system in a single computing node that has access to all sensors/actuators. The shortcomings of such an approach are well known.

A sophisticated approach is to keep the essential distributed nature of a system and deal with these interdependencies by novel interactions mechanisms. That is where the service-based computing comes into play.

Service-Oriented Architectures (SOA) provides a promising solution to orchestrate the interactions between these subsystems. We assert that SOA is the future for automotive software systems and will provide several advantages such as:

1- Effective System Integration — Architecting in-car DRES as a set of end-to-end services with formally described interfaces will provide an opportunity for robust and straightforward system integration, rather than designing a set of isolated computing islands and spending tremendous effort to connect them together in ad-hoc way.

2- Effective Customer Specifications — automakers will be able to express the Non-Functional Requirements (NFR) for end-to-end system services spanning multiple subsystems and provide these system-level specifications to their tier-1 suppliers. This scenario is better than specifying NFRs for individual subsystems, and then, possibly, getting surprised later with unacceptable behavior of the overall system after integration

Having the automakers focus on the specifications of individual subsystems results in frequent change of their requirements to the suppliers, and endless loop of design modifications.

3- Effective Architecture Description — Having services as first-class citizens at the architectural level allows early separation of concerns at the highest level of abstraction, and naturally expressing QoS for individual services. This will identify the critical services and allow investing more on them such as assigning them the right QoS, giving them higher priorities in resources allocations and scheduling, and using more rigorous verification approaches during testing and verification phases, while relaxing these properties for less-critical services.

4- Removing Redundancy and enabling reuse — By centralized management of the whole system architecture knowledge, coordination between subsystems can be modeled early in the design, and hence reusable services are identified and used across subsystems. This removes any duplication or redundancies.

5- Enabling Inter-vehicle communications — Once services are standardized across models of the same OEM or may be across OEMs' families of cars, higher-level services can be exposed by a vehicle system to other nearby vehicles or to the telematics infrastructure. A simple example of such application can be a special-purpose vehicle like an ambulance, which could automatically alert all other vehicles in a crowded street to make the way for it by propagating a signal across cars.

In the software product line context, software products are developed in a two-stage process, i.e. domain engineering and a concurrent application engineering process, as depicted in Figure 2.

The *domain engineering* process is aimed at establishing the reusable platform. The platform consists of artifacts such as the requirements specification, architecture documentation, design specification, implementation, and test cases. Domain engineering defines the commonality and variability between members of a product line. It involves, among other things, the development of appropriate product line architecture and a set of reusable software components such that the commonalities can be exploited economically while retaining the ability to vary the products [5].

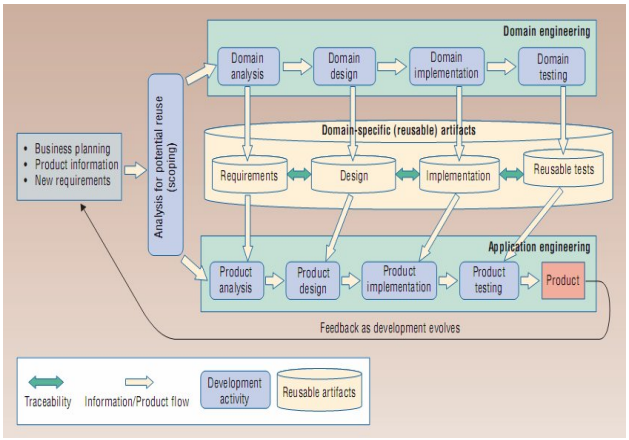

Figure 2: PLE process model [3]

Considering the description of the Dynamic SPLs As described in [3], "*Because it is impossible to foresee all the functionality or variability an SPL requires, there's a need for dynamic SPLs that produce software capable of adapting to fluctuations in user needs and evolving resource constraints. DSPLs bind variation points at runtime, initially when software is launched to adapt to the current environment, as well as during operation to adapt to changes in the environment.*", we assert that dynamic product line architecture is appropriate for realizing the re-configurability in software architecture.

3. Component Based Vs Service-Based Software Engineering

There is no clear distinction between a Service-Based Architecture (SBA) and Component-Based Architecture (CBA). Sometimes both terms, component and service, are used interchangeably in the literature. Differences such as fine-grained vs. coarse-grained or high-level vs. low-level are probably good observations, but the main point lies elsewhere. Indeed, CBA and SBA are different as they address very different issues; composing services into higher-level processes is totally different from integrating some components together into an application. SBA is architecture of an environment of running services. These services may be implemented as a set of (even distributed) components or objects.

W3C [13] defines a *component* as a software object, meant to interact with other components, encapsulating certain functionality or a set of functionalities. A component has a clearly defined interface and conforms to a prescribed behavior common to all components within an architecture. The goal of component based software engineering is to increase productivity and quality in software development.

SBA is different in that its constituent elements (services) have a loose-coupling with strong possibility of dynamic auto-engagement at runtime, instead of the prescribed connectivity in CBA. It defines a flexible coordination paradigm for integrating components in an evolving context.

The dynamic nature of automotive DRES precludes a-priori identification of the components that can define a system, and demands for runtime discovery and composition of such services [14].

However, in a resource-constraint application like automotive DRES, the supporting mechanisms used in SBA (like service description information and service discovery logic) are still too expensive to be adopted on-board in such a cost-competitive market. Moreover, using SBA in such safety-critical systems domain can introduce uncertainties in a system behavior like availability of services.

These limitations call for a back-to-the-basics initiative to put foundation for a lightweight, reliable and predictable infrastructure that enables the adoption of SBA in these DRES. We believe that this can bring the DRES software architecture to the next level, instead of squeezing the big elephant of web-services into these resources-constraint environments.

In the following section, we show how the abovementioned views about SBA can be used to describe an adaptive architecture to support dynamic variability in software product lines.

.

4. Adaptive Service-Based PLA

Software architecture is characterized by *Architectural Components*, *Connectors* and a *Configuration* that weaves these *Components* in a meaningful application [6]. Based on the discussion in the previous section, software services in SBA are the placeholders for these architectural *Components,* and *Connectors* will provide the weaving material for the architecture *Configuration.* In the next section, we will refer to these *Components* elements as services.

4.1 Variability in product line architectures

In our context, service description involves a defined set of interacting ports and a set of meaningful usage scenarios. These usage scenarios describe the *variations* in using a service. Each of these scenarios is designed to make use of a service in a certain situation. When changing a usage scenario of a service, the interactions between the surrounding services are also changed accordingly. We believe that architectural connectors elements are vital in realizing the inter-service communication variability. They provide management mechanisms for re-routing the communication links between services adaptively. We treat the architectural connectors as the main loci of variability in such product line architectures.

4.2 Run-time variability as a basis for adaptive software architectures

Traditionally, architecture-level computing elements are composed together at design time so that they cooperate together through their interacting ports to achieve a higher level goal or mission. These interactions are decided at design time after which services are tied together to achieve the over all system's goal. This traditional way of engagement is enough for systems that run in stable environments and are easily accessible to a design team (example web-service applications running on servers hosted by a development organization and its runtime environment is a fixed hardware platform with no interaction with the physical environment).

However, DRES usually run in dynamically changing environments such as automotive or avionics systems. Hence, such systems require flexibility that enables them to adapt to different situations and contexts. We argue that modeling the software architecture variability as a re-configurable engagement between the architectural computing elements, realizing this variability in the architectural connecting elements (as mentioned above) and having this variability bound at run-time, is a powerful conceptual model for creating adaptable DRES.

5. Achieving Adaptability in a Resource-Constrained Environment

Adaptability incurs overhead in terms of system design requirements such as memory, CPU cycles, communication bandwidth, and power consumption. Consequently, adaptive systems must carefully be designed, analyzed and built to find the right tradeoff between too much and too little flexibility. The goals of adaptive software architectures are the properties such as optimizing the system to re-prioritize or remove un-continued feature usage, fail-safe recovery in case of hardware or software components failures, re-adjusting to resources changes.

5.1 On-board product lines

In the abovementioned model, the dimension of product line variability is not targeting individual members of a line; rather, the whole line is deployed on board without any duplication of functionality in form of different implementations, as in the case where the variability is represented as multiple variants of the same component. However, it is very challenging to design such computational elements and factor out their functionality in such a way to enable reconfiguration at runtime. We assert that service-based approach is a promising way to achieve this objective. But there is still the challenge of reasoning and deciding about the appropriate reconfiguration of the architecture in response to different contexts. A more challenging issue is to have suitable mechanism to make this reconfiguration to occur at run-time during online operation (For example, when a car is being driven) with the reconfiguration reasoning logic on-board (self-adaptable). However, the later issue is impractical from both safety point of view, as the system needs to have at least a separate stable state for reconfiguration and then coming back to its online state, and also from resource-constraints point of view as a huge memory and CPU resources are needed by the self-adaptation logic.

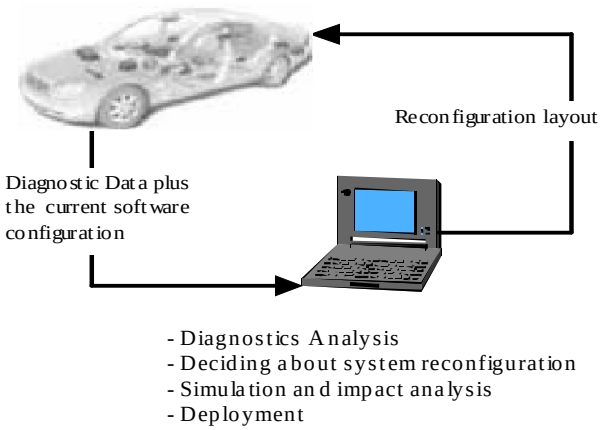

Figure 3: Reconfiguring the in-vehicle software at maintenance center

5.2 Off-board reconfiguration

We propose that a practical way to deal with the challenge described in the previous Section is to have the reconfiguration reasoning logic off-board and to have the DRES connected to this off-board system for deciding about the suitable reconfiguration (deriving and deploying another product). This approach can be implemented in the in-vehicle DRES by having the reasoning logic residing on a server at a maintenance center.

Figure 3 shows a car connected to an off-board computer in the maintenance center. The computer includes certified Architecture Reconfiguration-Engine application that is specific to the car's OEM, with a complete knowledge of the whole software product line. This feedback-based approach analyses the diagnostic information collected at runtime of the car's operation, synthesizes this information, and decides about the appropriate architecture reconfiguration based on certain goals such as resource optimization.

5.3 Dynamic design space

The design space for possible configurations of the software architecture is explosive and hence, the domain engineering team is challenged by reaching at a set of meaningful configurations that serves as the basic options when deciding about architecture reconfiguration at runtime. Simulations approaches could be used during the product line domain engineering phase to explore, analyze and optimize the architecture configurations.

7. Related Work

There have been several attempts at applying service-oriented approaches for embedded systems in general and in automotive domain in particular. The most relevant work is proposed by Kruger [7, 8], who views service as a resultant function of interactions between software components. Another relevant work is that of Baresi et al. [14]. The common part of Kruger's notion of service and ours proposed approach is that services are recognized as a first-class architectural element, from which the whole DRES architecture can be described, however, we consider a service as an integral part of the application, and involves a *reusable* and *application-aware* computing logic, rather than a logical entity that results from interactions between components.

Other related work includes tool support (AUTOFOCUS) for service-based software development by Deubler [9], real-time service-based coordination between autonomous vehicles by Becker [10], Milanovic [11] has proposed to apply the web-services technology in embedded systems on a lightweight basis, and Hartmann [12] proposed a service-based specification language for requirements engineering of automotive software systems.

8. Conclusion and Future Work

The realization of adaptive DRES systems is a very challenging issue for designing and developing systems

that run in a highly dynamic environment such as automotive and avionics domains. In this paper, we have proposed that for these systems to evolve adaptively accordingly to the changing requirements of their execution environments, they need to be flexible and still reliable to meet the safety requirements. We have proposed that service–oriented approach can be exploited for these systems to support the flexibility of subsystems interactions. We used the Dynamic Product Line architecture as a mean to realize a runtime re-configurable architectures for automotive systems.

Our future work in this line of research includes:

- Identifying suitable notations and languages for describing a *service-based* and *adaptive* architecture with semantics to analyze them.
- Product line domain engineering techniques to design and analyze the target dynamic runtime environment and, accordingly, generating the reconfigurations that can be deployed at runtime.
- Suitable reasoning applications to perform the architectural reconfigurations.

Acknowledgement

Lero is funded by Science Foundation Ireland under grant number 03/CE2/I303-1.

9. References

[1] The AUTOSAR development partnership., www.autosar.org, Ed., Dec. 2004.

[2] Papazoglou, M. P., P. Traverso, S. Dustdar, and F. Leymann, Service-Oriented Computing: State of the Art and Research Challenges, Computer, vol. 40, pp. 38-45, 2007.

[3] Hallsteinsen, S., M. Hinchey, P. Sooyong, and K. Schmid, Dynamic Software Product Lines, Computer, vol. 41, pp. 93-95, 2008.

[4] Lui, S., Open challenges in real time embedded systems, SIGBED Rev., vol. 1, pp. 13-15, 2004.

[5] Thiel, S., L. O'Brien, M. A. Babar, and G. Botterweck, Software Product Lines in Automotive Systems Engineering, SAE World Congress Proceedings, paper 2008-01-1449, .SAE April 2008.

[6] Mary, S., D. Robert, V. K. Daniel, L. R. Theodore, M. Y. David, and Z. Gregory, Abstractions for Software Architecture and Tools to Support Them, IEEE Transactions on Software Engineering, vol. 21, pp. 314-335, 1995.

[7] Krueger, I., Service-Based Software Development for Automotive Applications, 2004.

[8] Kruger, I. H., Service-oriented software and systems engineering - a vision for the automotive domain, presented at Formal Methods and Models for Co-Design, 2005. MEMOCODE '05. Proceedings. Third ACM and IEEE International Conference on, 2005.

[9] Deubler, M., J. Grunbauer, G. Popp, G. Wimmel, and C. Salzmann, Tool supported development of service-based systems, presented at Software Engineering Conference, 2004. 11th Asia-Pacific, 2004.

[10] Becker, B. and H. Giese, On Safe Service-Oriented Real-Time Coordination for Autonomous Vehicles.

[11] Milanovic, N., J. Richling, and M. Malek, Lightweight services for embedded systems, presented at Software Technologies for Future Embedded and Ubiquitous Systems, 2004. Proceedings. Second IEEE Workshop on, 2004.

[12] Hartmann, J., S. Rittmann, D. Wild, and P. Scholz, Formal incremental requirements specification of service-oriented automotive software systems, presented at Service-Oriented System Engineering, 2006. SOSE '06. Second IEEE International Workshop, 2006.

[13] W3C, "W3C Working Group Note 11 February 2004," http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211

[14] Baresi, L., Guinea, S., An Introduction to Self-Healing Web Services, in Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems: IEEE Computer Society, 2005.