On Practical Service-Based Computing in Distributed Embedded Automotive Systems

Hesham Shokry and Mike Hinchey
Lero—the Irish Software Engineering Research Centre
University of Limerick
Ireland
{hesham.shokry, mike.hinchey}@lero.ie

Abstract

The automotive industry is, like several other domains, a source of both challenging problems and innovative technologies of various kinds. One interesting phenomenon in this domain is the extensive interdependencies between the constituent nodes, or ECUs, of its networked embedded system. This in turn leads to extensive interactions between ECUs over the network infrastructure of an automobile.

We survey two significant approaches for applying Service-Based Computing (SBC) in automotive embedded systems and build on our experience in this domain to synthesis and identify potential gaps between the state-of-the-art and the actual state-of-practice in industry.

We argue that standardization of the automotive system-level services is the gateway to practical application of SBC in this domain. Also we describe potential solutions for closing these gaps through a description of our ongoing work for practical SBC in automotive embedded system environment.

1. Introduction

Automotive systems differ from other Distributed Real-time Embedded (DRE) Systems, having both unique characteristics and development challenges. Automotive manufacturers and software vendors face problems which include:

- high demand for customization, driven by market competition and end-user preferences;
- subsystems, supplied by multiple vendors, are developed using different processes and different engineering technologies;
- software applications running on these individual subsystems are inherently incompatible, due to the diversity of vendors' development cultures, and
- unreliability of these software applications, due to the high pressure on vendors, for fast time-to-market delivery, in turn resulting in high costs incurred by manufacturers for vehicle recalls and maintenance.

Moreover, automotive systems typically operate in highly dynamic environments characterized by diverse change to components or requirements; these include changes to runtime resources due to failures of, 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 to telematic services as cars travel from one location to another.

Automotive DRE systems are a networked set of computing nodes connected typically by a shared data medium. The main objective of introducing a data network infrastructure into the vehicular system is to disseminate information from different ECUs (such as acquired sensor data) through point-to-point communication links. This infrastructure has proven to be inefficient, with a huge, and increasing, number of data signals to be broadcast to an ever increasing number of ECUs. For example, the vehicle speed estimated by the engine controller or by wheel rotation sensors has to be known in order to adapt the steering effort, to control the suspension, or simply to choose the right speed for wipers. In today's luxury cars, up to 2500 signals (i.e., elementary information such as the speed of the vehicle) are exchanged by up to 70 ECUs [1]. This has motivated the use of multiplexed networks, such as the high speed CAN

network [3] and the more predictable and fault tolerant Flexray standard [4].

Alongside this evolution in the complexity of the network infrastructure, the features in modern cars are becoming highly interdependent and require interactions and exchange of functionality across ECUs in the same domain. Sometimes these interactions span subdomain boundaries. This evolution can have severe implications for car manufacturers mainly because of the integration problems that arise from incompatibility between ECUs developed by different vendors and third-tier automotive suppliers. The AUTOSAR consortium is trying to address these issues via a component-based approach whereby interfaces are standardized and communications between distributed components are modeled (statically) through the Virtual Function Bus, or VFB. The VFB is used to check consistency of interfaces at design-time, and once the application is compiled, the interactions between components are wired for the whole lifecycle of the system. Obviously such a mechanism is incapable of handling the inherit characteristics of the interaction scenarios such as concurrency, overlap, and hierarchical composition, to name but a few.

The complexity and interdependencies of automotive systems have attracted the attention of software engineering researchers in the past few years as a challenging domain for application of their techniques and research. In particular, researchers in the area of service-based or service-oriented computing have proposed approaches that abstract a meaningful set of component interactions as a *service* and model it, typically, using extensions of the UML notation.

We discuss and synthesize two approaches that we believe are promising for dealing with this complexity. However, several issues critical to this domain are only implicitly considered by these approaches, and others are completely unaddressed. We will consider these issues and discuss how the approaches can be refined to be more practically applicable in this important domain.

2. Pervasive Automotive Systems

A modern automotive DRE system exhibits a wide set of characteristics that renders the design task more challenging than for embedded systems in other domains, such as industrial process control and consumer electronics. We will address these characteristics here.

2.1 Automotive DRE system architecture

In general, the vehicular DRE system is a networked set of Electronic Control Units (ECU) each of which is *embedded* near a mechatronic part of the vehicle (sensors, actuators, and mechanical systems) that to be managed and controlled by this ECU through input/output signaling. The ECUs are arranged in clusters each of which manages a major set of related functionalities in the vehicle such as Chassis applications, Power-train management, and Body electronics [2]. We briefly discuss the structural, functional and performance distinctive characteristics of the vehicular DRE system.

Structural Characteristics. An automotive DRE system is a set of networked clusters, called subdomains, each of which involves a set of nodes called ECUs. The ECUs in a subdomain cluster are typically dedicated to controlling all of the relevant functions of this subdomain. For example, Figure 1 shows a schematic of the Chassis subdomain where ECUs (represented as boxes) are connected together via a shared bus over which they exchange messages. The Adaptive Cruise Control ECU in the figure exchanges I/O signals to manage the mechatronic system under control. The subdomain functions are composed of specialized functions such as the Adaptive Cruise Control, Steering Control, etc. (cf. Figure 1) and each specialized function is hosted by an ECU.

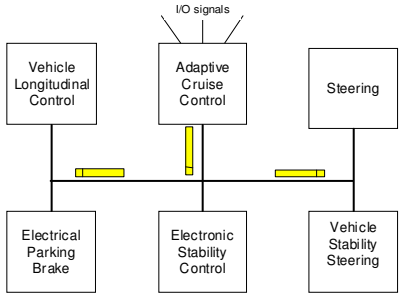

Figure 1: Chassis subdomain: the cruise control ECU communicates with other ECUs via a shared bus

An ECU is a self-contained computing unit running a set of application-specific tasks such as a signal-processing algorithm or an interrupt handler. Each ECU exhibits self-management in the sense that it manages task-scheduling and resource-access priorities. The entire set of ECUs (as black-boxes) in a subdomain, however, share *common responsibility*

in the decision-making process and execution of subdomain functions. This *federated* architecture is mimicked at the subdomain level, where data and requests are passed through gateways to provide global status and synchronization between different tasks in different subdomains. A common example is the data collected from different subdomains to be presented on the dashboard, which is part of the Body electronics subdomain.

There are well-established technologies for designing and analyzing the software components or tasks governed by a single ECU (e.g., schedulability, inter-task communications, time predictability, etc.). However, technology for modeling and analyzing functions at the higher structural-level, involving tasks from two or more ECUs possibly belonging to different subdomains, is less mature.

Service-Based Computing (SBC) is an attractive concept for modeling these higher-level interactions. There are, however, many issues to be addressed because, unfortunately, we do not (currently) have sufficient infrastructure for development support (e.g., distributed operating systems, global resource sharing, global timing, etc.) as we do, for example, in the case of computing that respects ECU boundaries.

Functional Characteristics. Communication in the vehicular system is classified into two areas: Intra-vehicle and Inter-vehicle. In the latter, the whole vehicle is considered as a large mobile system that may have *opportunistic* communications with other nearby vehicles and/or telematics infrastructure. These characteristics dominate others and open the door for concepts such as Enterprise Computing and Mobile Computing, which lies outside the focus of our discussion.

An ECU typically manages the following functions:

1. It controls and abstracts the corresponding vehicle physical mechatronic system.
2. It executes the algorithms specific to the system under control. For example, a vision-based driver-assistance system executes sophisticated image processing algorithms, while a gateway ECU performs simple formatting and filtering of network messages.
3. It communicates (periodically and sporadically) sensor-data and status information with other ECUs. For example, it can periodically disseminate current vehicle-speed values or broadcast the "wipers parked" event.

Although such functionality is typical of an embedded system, it is insufficient to realize higher-level tasks or functions which involve *coordinated* sets of contributions from many software components in different ECUs, which involves far more than simple dissemination of data or status. This demand for coordination requires careful modeling of higher-level processes with the associated logic to track context and handle any possible exceptions or failures in one of the contributing components. A service-based coordination model can help to realize that.

Performance Characteristics. An automotive DRE system is characterized by a (possibly contradicting) set of performance requirements, often necessitating careful compromise. In the same vehicle, for example, one can find subdomains with soft real-time constraints (such as the in-vehicle entertainment system) and others with hard (strict) ones (such as power-train).

Safety-criticality and real-time-sensitivity are two parallel properties in the vehicular DRE system. The former can be guaranteed (in part) by reliability analysis and the latter by worst/average executions analysis. Orthogonal to these properties is the resource-scarceness (CPU, memory, power, comm. bandwidth) which is a common denominator in most embedded applications.

Applying SBC here can be more problematic than first meets the eye. What are the analysis techniques that can guarantee a certain level of quality-of-service (reliability and timeliness)? What are the implementation techniques that maintain resource efficiency? A practical approach to services in automotive DREs must address these questions.

Product development. The development process of the vehicular DRE system is worth noting here to complete the picture.

As in many other domains, high levels of competition between suppliers of more innovative products has led to strict confidentiality requirements for both software and the associated interfaces, while realization of more innovative functions requires the coalition of more than one product (ECU) as we discussed before. Consequently, in this competitive market, it is becoming more-and-more difficult for products to be able to communicate with each other. One solution to this is to have the high-level executable specifications (services specification) developed by the vehicle manufacturers and with which all relevant suppliers must comply. An alternative solution is to have a careful coordination

model that decouples the behavior of different products participating in a service. We will see examples of these solutions later.

Another important issue for the development process is maintainability of the system. It is crucial for the successful application of services technology here to provide systematic diagnosis and maintainability support.

2.2 SBC in Automotive DREs

Although the classification of characteristics in Section 2.1 is not the only possible one, it is useful for the purposes of identifying the challenges facing the application of the notion of a "service" in this context.

Our main objective in this paper is to assess the feasibility of applying such a notion of "service" as a means of solving the problem of automotive DRE systems complexity. To this end, we have provided a discussion of automotive DRE system characteristics, and in the light of this view, we synthesize the two proposals that claim to address this problem. We conclude with a section that describes our work on an approach that builds on the merits of these proposals and mitigates their impractical aspects.

3. Approaches to SBC

We survey approaches to SBC that have been argued to be applicable in the automotive domain. Here we focus on approaches that address the *in-vehicle* embedded system and rule out other approaches for inter-vehicle computing as it is less related to the embedded systems paradigm (the issues addressed there are more related to those of telematics and enterprise computing).

There are other approaches that have proposed service-based frameworks for embedded systems in general, or for other vertical domains in particular, such as mobile computing and industrial production systems. In-vehicle automotive systems are quite different from other vertical domains.

For example, in-vehicle embedded systems do not need to address the mobility factor as in mobile systems (excluding inter-vehicle computing), because it is a wired network of ECUs (in-car wireless devices have negligible mobility effect). Moreover, in comparison to industrial production systems, the computing processes of automotive systems are, by nature, highly dynamic (because of the dynamic environment changes), interactive (highly dependent on user and sensors input) and distributed (the large number of ECUs embedded everywhere in the car [1]), whereas industrial production processes do not possess such characteristics and they are defined *a priori* (even reconfigurable production systems are also changed offline).

On the other hand, in-vehicle embedded systems share issues—such as safety-critical constraints and timeliness—with domains such as medical device systems and avionics. Therefore, the approaches we discuss here are also applicable to these domains.

There are two main approaches that have been proposed to address SBC issues for in-vehicle embedded systems. The first is that of Baresi *et al.* [5, 6, 7]. This approach employs an event-based flavor of the well-known publish/subscribe component-model to mediate the cooperation between automotive-functionality *components* that execute automotive-specific tasks or algorithms. In what follows, we will use the term *component* to refer to these functionality components and will identify any other auxiliary components explicitly. The second approach proposed by Kruger ([12]-[14]) uses a formal notion of service and maps the semantics of this model to Message Sequence Charts. These components communicate via messages exchanged over (logical) channels. As we shall see, both approaches consider services as interactions between components.

In the following subsection we will describe and analyze each of these approaches and then we will synthesis their merits and limitations from a practical point-of-view.

3.1. Event-Based Interactions Approach

The approach outlined in [6] employs the publish/subscribe component-model [8], a coordination protocol used to share information among a set of loosely-coupled components, as a coordination infrastructure between components.

The use of the publish/subscribe model to describe and validate component coordination was originally described in [5], and is used in [6] to demonstrate its application in coordinating components in automotive systems. Essentially, components publish *events* they generate and subscribe to other events they are interested to be notified about. The behavior of these coordinated components is modeled in Statecharts [9] where the transition between states describes the reaction of the components to events notified to it by the dispatcher. For example, in Figure 2 (adapted from [6]), the climate control component subscribes to the event *warm* (among other events) and the

component's machine is initially in the state *cold*. When someone publishes the event *warm*, the dispatcher notifies all subscribers of the occurrence of the event *warm* and the climate control component consumes this notification by advancing its state-machine to *warming* and, in turn, publishes an event *warming* [6].

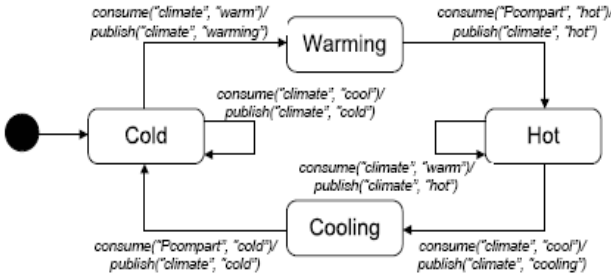

Figure 2: Statechart of climate control software component.

Before making observations on this approach, we first refer to the survey of publish/subscribe variants in [8]. In this survey, three dimensions of decoupling between components have been identified, namely, time, space and synchronization. Space-decoupling means that the communicating components need not to be aware of each others' identity and location. For example, subscriber components do not keep references to publishers and vice versa. Also a publisher of an event does not know how many subscribers are being notified by this event. This fact is truly significant and favorable in automotive embedded system as it directly supports the distribution of components over separate ECUs. Moreover, as pointed out in [6], this blind-like communication between components facilitates the dynamic addition and replacement of software components in case of maintenance or in the case of a removable node being plugged into the network (GPS for example).

Synchronization is another dimension of decoupling between subscribers; it is particularly important if the service involves two or more simultaneous actions. Here we raise a practical question: how can this approach handle the case of simultaneous reactions of two subscribers (components) to the same event when the two reactions are *logically related*? If the two subscribers belong to the same ECU, one may argue that we can schedule them based on the desired sequence of the service. But this is not possible because the scheduler is not aware of the service sequencing, and the best it can do is schedule them based on (typically predetermined) task priority, which has nothing to do with the service sequence. On the other hand, if the subscribers belong to two different ECUs, then the situation is even worse as we do not have in this approach any support for synchronizing their reactions in this approach. A central state-machine maintained by the dispatcher would help to synchronize the simultaneous reactions (parallel states in Statecharts).

The third dimension is decoupling of *timing*. Obviously, in automotive DRE systems, the reaction of individual components must meet deadlines (in most cases). Some components may be waiting for these events (synchronous interaction) to continue execution and others may execute in parallel (asynchronous interaction). (Note that we talk about synchronization here from a timing stand-point, not from a concurrency one.) In the case of synchronous interaction, the situation can be rectified by using a timeout and taking corrective action in the case that the task does not meet its deadline (completion of the event has not been notified in time). However, in the asynchronous case, the component will not be aware if the task has completed on time or not. The component will receive the event anyway (assuming the task will complete in finite time) but it would not know then that the task has completed late. Timestamps or other mechanisms may be used, and the approach should take that into consideration.

Having an explicit central control of service execution is essential to make full use of this approach. The most important feature with that is ability to support *concurrency* of services (interaction scenarios). A component may engage in more than one scenario simultaneously without being aware of that. The component is reacting to events based on its internal state machine but it is not aware if this reaction is part of, possibly, more than one scenario. The state machine is designed (by definition) based on what the component is responsible for. It is easy to recognize that for every step in a scenario, there may be more than one component (subscribers to the event which led to that step) reacting simultaneously, and consequently contributing to perform this. This step itself can be imagined as a synchronization point between *concurrent* and *overlapping* scenarios executing simultaneously. In general, this overlap of services is natural and favorable in modeling automotive embedded systems with, however, careful consideration of concurrency.

Another important point is the *hierarchical composition of services*. It is essential for an approach to SBC in automotive systems to provide modeling

mechanisms for *composing* services into higher level ones that are too complex to be defined as elementary interactions between individual components. The approach of [6] uses Live Sequence Charts [10], LCS, to model interaction scenarios that realize services. LSC has novel mechanisms for modeling complex interactions (such as loops) similar to the ones proposed in the current version of UML [11]. Despite that, LSC are not able to model explicit hierarchical service composition (though proprietary extensions to LSC may be helpful in that regard!).

In conclusion: this event-driven interaction model is promising for modeling services in automotive embedded systems and considers inherent properties in the domains such as concurrency and distribution of computation. However, as discussed above, the approach needs to consider other characteristics such as timing, concurrency and composition of services.

3.2. Message-Based Data-Flow approach

In [12], Kruger bases his approach to SBC on a model of precise specification of services.

In the description of the approach given here, we are interested in clarifying the method of specification of services and system architecture, rather than the particulars of the underlying formalisms. Many sound formalisms would suffice for early analysis of the system architecture. Our concern is a discussion of the ability of the approach to represent the notion of service, while maintaining the essential characteristics of an automotive DRE system as outlined in Section 2.1.

In Kruger's approach, components are modeled as dataflow nodes connected together with *logical* input/output channels over which the components exchange messages. Channels are *typed* in the sense that each channel carries a certain type (or aggregated type) of message. This is much similar to using the concept of typed-ports (that accept/deliver a certain type of data or request message) in modeling component interfaces in a component-based architecture.

A channel also has a communication *history*, and this history is modeled as a *stream* (the concept of *stream* here is based on the work in [18]). A channel history refers to the *message-sequence* communicated over the channel during some period of time. Note, the notion of time here is simply used to refer to the fact that messages are ordered with possible time delay, and does not relate to the temporal behavior of components (we will come to this later). To this end, a component is considered as a dataflow node that maps input *streams* of messages to output streams. See [16] for a complete discussion of this model and its associated formalism.

Assume that a component has only one input channel and one output channel. The component specification (functional behavior) is determined by *two* sets of message-sequences. The first is a set of input messages-sequences referred to as *assumptions* in [12], i.e. the assumption a component makes about the message-sequences at its input channel. The second set is the group of message-sequences that represent components' reactions to the input sequences or histories. This model can be thought of as being very similar to a discrete event-based system with message-sequences representing the event strings that drive the component's automaton to generate a specific output of event-strings.

To this end, a component is described in terms of a set of interfaces, and each interface has syntax and semantics as follows:

- The *interface syntax*: determined by the set of input/output types of messages communicated over the channels of this interface.
- The *interface semantics*: determined by the input/output histories (message-sequences) prescribed for this interface.

In this way, the interfaces are describing the *component's behavior*. This behavior is identified in [17] as *total behavior* of a component; this contrasts with the concept of *partial behavior* of the component that describes the *component's contribution* to a service. From the component's point of view, a service is regarded as a partial behavior over the component's total behavior. Kruger's approach provides for a supporting formalism and associated tool (FOCUS [17]) for these concepts of *totality* and *partiality* of behavior.

To enable usability of this model, Kruger uses an extended version of MSCs [13, 15] as a graphical means of describing services as interaction patterns between components, and then maps the semantics of MSC to his formal model. We will not discuss the particulars of this model here, but we will take important notes about it as necessary in the discussion. However, we refer the interested readers to [12]-[14].

A practical application of this approach in the automotive domain has been the use of the approach to develop a RTCORBA-like executable specification of an automotive system-services model for the purpose of validation and integration with ECU suppliers [14]. In the context of this project, the

approach proposes a software development process and a supporting tool chain.

The process phase handles the description of possible usage of services through use case graphs to elicit and capture the relationships between high-level services (services involving interactions with the user, for example) and captures the interaction scenarios for every possible service using MSCs (as described in their formal model). *Roles* are used to define interacting entities in the MSC description instead of using named components. This greatly helps in the early description of the scenarios and later the mapping of these roles to components. See [14] for details of the process.

It is worth noting that this process is intended to generate an executable model (not the target code) of system services used as a reference model for ECU suppliers to validate their systems against.

In conclusion: the message-based data-flow approach is based on a decentralized computing model which is difficult to maintain. Also, if a component is absent, there is no disciplined way of providing exception handling (the service will be just not achievable and there is no other means of providing a degraded version of the service). Moreover, modeling of temporal behavior of components (that represent tasks in an ECU) is not specified and no means of analyzing the temporal behavior of the service (within the ECU boundaries, we can perform a temporal analysis because we have a specific RTOS scheduler and we know the capabilities of the underlying computing power). Hence, a useful approach necessitates having techniques for global analysis of a service spanning different ECUs.

Partial behavior specifications exist only at design time, and scatter over the participating components' specifications. At run time, we need a controller for the service itself to manage exception handling and to more easily maintain service logic.

4. An Extension to the Publish/Subscribe Approach

A practical step for introducing a new technology to the industry is to standardize the domain knowledge part of this technology. One factor of the success of the AUTOSAR reference architecture is the standardization of its Basic-Software components interfaces (the Basic Software is the infrastructure part of the embedded software that includes the hardware abstraction and network drivers).

Early versions of AUTOSAR had not considered the standardization of application level components because of its diversity and the different sophisticated implementation strategies of suppliers. However, as car functions and their integration are becoming more diverse and simultaneously we have in a need for a greater number of interactions; application level functionality has started to make its way into the standardization process. For example, the latest version of AUTOSAR prescribes the system-level functions of different subdomains.

We believe that SBC in the automotive industry will be boosted by standardizing the *possible* and *general* functions in every subdomain. Although the complexity of the vehicle system is quite prohibitive for such a task, the segmentation of the vehicle into a number of (standardized) subdomains facilitates the comprehension of the standard functions in each of these subdomains.

Additionally, this does not prevent the transferability of functions across subdomains, because gateways in the network can be used to overlay the requests/events from one subdomain with another.

Inspired by the notion of "service" as an interaction between components, we describe here our ongoing work on developing a practical approach to the coordination of distributed components in a subdomain. The approach is mainly based on the publish/subscribe coordination model described earlier. We will describe the main points of the approach and the rationale and implications of each.

Service Standardization and central control. Services are standardized (in every subdomain) and modeled as state-machines (Statecharts), whereby states simply represent the successive steps or states in which the service exists at a given point in time and should pass through before successfully completing; transitions are labeled by the events controlling the service execution flow. Maintaining a control model of the service offers some important benefits: First, it provides an *explicit* model of the service execution flow at *runtime*. Moreover, the service's temporal behavior is explicitly modeled and analyzable at design time. Second, it provides back-tracking capability in case of any discrepancies during the execution of the service. This rectifies the timing problem in previous approaches, where timing control of the service is scattered over the individual reactions of the contributing components. The implication of keeping a state-machine runtime model of the service is an overhead, but this is outbalanced

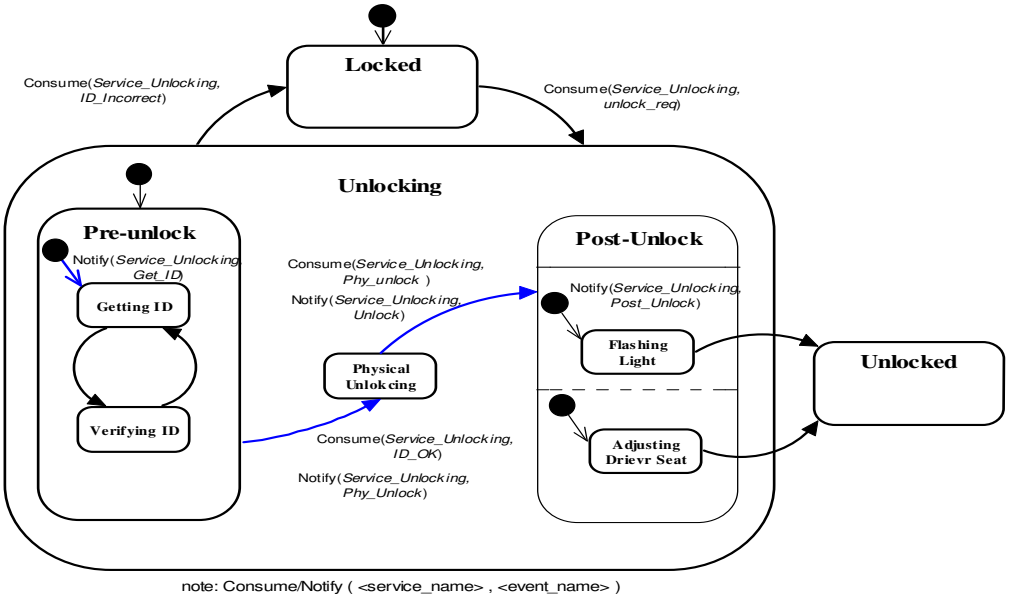

Figure-3: State-machine of the *Service_Unlocking* service

by the advantages we mention above. In Figure 3, a state machine of the unlocking service illustrates how consume and notify transition labels control the execution flow of the service (for space limitations, the figure does not show the labels of all transitions). Note that the Unlocking_Service is the service name that the dispatcher uses to distinguish services; however the Unlocking in Figure 3 is the name of a sub-state of the Statechart. Statecharts provide suitable notations to distinguish between events naming at different level of the machine hierarchy.

Component behavior. The behavior of the components contributing in the service is modeled inStatecharts, as in Baresi's approach described above. This retains the advantages of this approach, such as the possible use of model-checking for validation of the whole set of communicating-automata (both component automata and services automata). In other words, the service state-machine is, simply, acting as a higher-level controller of the individual reactions of components. The separation of this high-level control from individual contributions from components achieves the desired coordination which is the core of SBC that we target. With careful generalization of the service state-machine, the standardization of such services provides individual vendors a degree of freedom in their implementations of ECUs in the corresponding subdomain.

We are in favor of using the event-driven approach because of the inherent property of asynchronous service timing (rather than the close interaction between tasks in an ECU that are mostly synchronized for resource access, data sharing, etc.). Note that this encapsulation of events by the corresponding service enables components to contribute in more than one service and to identify which service has sent this event in the case that the event is used by more than one service. It is up to the component implementation to decide about its reaction to notifications of service-event combinations. For example, in Figure 3, the Flash-lighting system may react in the same way to the event post-unlock whatever the service executing happens to be.

The next steps in this approach are to establish formal specifications of the dispatcher and use the theory of communicating automata to validate the flow of individual services.

5. Conclusion

Embedded systems in general and automotive systems in particular, are continuously posing challenges for the Software Engineering community. The interactions between the (relatively) large number of connected ECUs in the modern car renders the development of its embedded software a complex task.

Service-Based Computing is an architectural concept capable of modeling the coordination between independent components physically distributed over the network. We have focused on the SBC approaches that

have demonstrated the ability to model interactions between the distributed components within the in-vehicle embedded system. More specifically, we have discussed two approaches, each of which employ different coordination mechanisms.

The first approach uses the well-known event-based publish/subscribe coordination model and the second approach uses a formal message-passing mechanism. We have discussed the merits and limitations of each.

A common denominator of the approaches discussed is the notion of a service as a sequence of interactions between a number of components contributing to this service. We see this as a realistic view of services and it directly rectifies the (unavoidable) decentralized nature of in-vehicle control systems. On the other hand these approaches show some limitations, particularly in timing and runtime control of the service execution flow.

Based on the main concepts of these two approaches, we have described a conceptual model of a practical approach for modeling automotive services by explicitly modeling services using state-machines and maintaining this model at runtime to enable back tracking in case of failures. Moreover, this approach rectifies the timing problem in previous approaches, making it more practical to address the timeliness properties of embedded systems. Planned future work includes the development and validation of a prototype in conjunction with our industrial partners.

Acknowledgment

This work was supported in part by Science Foundation Ireland grant 03/CE2/I303_1 to Lero—the Irish Software Engineering Research Centre.

References

[1] A. Albert, Comparison of Event-Triggered and Time-Triggered Concepts with Regards to Distributed Control Systems. In *Proceedings of EmbeddedWorld 2004*, Nuremberg, Germany, February 2004.

[2] The AUTOSAR Consortium, Automotive Open System Architecture, www.autosar.org, Visited on 18 February 2009.

[3] International Organization for Standardization, ISO 11519-2, Road Vehicles—Low Speed Serial Data Communication—Part 2: Vehicle Area Network (VAN), ISO, 1994.

[4] FlexRay Consortium., FlexRay Communications System–Protocol Specification, Available at http://www.flexray.com, December 2005.

[5] L. Baresi, C. Ghezzi, and L. Zanolin, Modeling and Validation of Publish/Subscribe Architectures. In S. Beydeda and V. Gruhn (eds.) *Testing Commercial-Off The-Shelf Components And Systems*, pp. 273-292, Springer, 2005.

[6] L. Baresi and C. Ghezzi. Validation of Component and Service Federations in Automotive Software Applications. In *Proceedings of the First Automotive Software Workshop (ASWSD)*. San Diego, CA, USA, 2004, Lecture Notes in Computer Science 4147, pp. 57-73, 2006

[7] L. Baresi, C. Ghezzi, A. Miele, M. Miraz, A. Naggi, and F. Pacifici. Hybrid Service Oriented Architectures: A Case Study in the Automotive Domain. In *Proceedings of the International Workshop on Software Engineering and Middleware (SEM'05)*, co-located with ESEC 2005, Lisbon, Portugal, 2005, pp. 62-68.

[8] P. Th. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, The Many Faces of Publish/Subscribe, *ACM Computing Surveys*, 35(2):114-131, 2003.

[9] D. Harel and M. Politi, *Modeling Reactive Systems with Statecharts: The Statemate Approach*, McGraw-Hill, Inc., New York, NY, 1998

[10] W. Damm and D. Harel, LSCs: Breathing Life into Message Sequence Charts, *Formal Methods in System Design*, 19(1):45–80, 2001.

[11] UML 2.0, http://www.omg.org/uml

[12] I. Krüger, Specifying Services with UML and UML-RT, *Electronic Notes in Theoretical Computer Science*, 65(7), Elsevier, 2002.

[13] I. Krüger, Capturing Overlapping, Triggered, and Preemptive Collaboratons, In: M. Pezze (ed.) *Fundamental Approaches to Software Engineering*, 6th International Conference, FASE 2003, Lecture Notes in Computer Science 2621, Springer, 2003.

[14] I. H. Krüger, J. Ahluwalia, D. Gupta, R. Mathew, P. Moorthy, W. Phillips, and S. Rittmann. Towards a Process and Tool-Chain for Service-Oriented Automotive Software Engineering. In *Proceedings of the ICSE 2004 Workshop on Software Engineering for Automotive Systems (SEAS)*, 2004.

[15] ITU-T, Recommendation Z.120 – Message Sequence Chart (MSC 96) ,ITU, Geneva, 1996.

[16] M. Broy, I.H. Krüger, and M. Meisinger, A Formal Model of Services, *ACM Transactions on Software Engineering And Methodology*, 16(1), February 2006.

[17] M. Broy and K. Stølen, *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*, Springer-Verlag New York, Inc., Secaucus, NJ, 2001.

[18] R. Stephens. A Survey of Stream Processing. *Acta Informatica*, 34:491–541, 1997.