On Service-Orientation for Automotive Software

Stefan Kugele*, Philipp Obergfell†, Manfred Broy*‡, Oliver Creighton†, Matthias Traub†, and Wolfgang Hopfensitz†
*Department of Informatics, Technical University of Munich, 85748 Garching, Germany
Email: stefan.kugele@tum.de, broy@in.tum.de
†BMW Group, Munich, Germany
Email: {Philipp.Obergfell, Oliver.Creighton, Matthias.Traub, Wolfgang.Hopensitz}@bmw.de
‡Zentrum Digitalisierung.Bayern, 85748 Garching, Germany
Email: Manfred.Broy@zd-b.de

***Abstract*—*Background:* During the last decades, the functional power and complexity of automotive E/E architectures grew radically and is going to grow further in the future. For highly and fully automated driving, functions with the highest safety integrity level need to be realized, requiring new development methodologies and a new level of formal rigor. *Aim:* We investigate to what extent SOA concepts are applicable for safety-critical embedded automotive software systems and whether this concept is appealing to E/E architects. *Method:* We conducted a survey research by interviewing system architects at our industrial partner, then we applied the grounded theory method in order to derive a theory and a set of requirements for an automotive SOA approach. Additionally, we illustrate the approach using a function needed in a highly automated driving scenario. *Results:* We present a formal service model and an automotive SOA framework. Both aspects, i. e., architecture *structuring* and *formal* service description resulted from the analyzed interview data. *Limitation:* This approach has not been evaluated extensively, yet. *Conclusion:* Our first results suggest that SOA concepts are indeed successfully applicable in (continuous) automotive software engineering and are a means to cope with complexity and safety requirements.**

I. Introduction

During the last decades, more and more software-controlled functions were introduced in embedded systems. Many of them are characterized as safety-critical such as those found in the avionic, automotive, railway, and industry automation domains. In the future, the number of those functions [1] as well as the complexity of involved hardware topologies will grow further—especially in emerging cyber-physical systems.

The trend towards digitisation will likely continue in the future. It is accompanied by more and more systems having partial or full autonomous features. Since decades, the autopilot in airplanes is a well-established mechatronic software-controlled system. Shipping and aerospace are two more sectors where autopilot features are established. We investigate the use of service-oriented architecture (SOA) in the automotive domain for functions with a high or even full degree of automation, i. e., SAE levels 4 and 5 according to [2].

Today, E/E architectures (electric/electronic) are best characterized as historically grown, federated architectures with oftentimes pragmatic, cost-efficient, and ad-hoc solutions. This might have been sufficient for today's advanced driver assistance systems (ADAS) or even for highly automated driving, where already existing and less-critical ADAS functions are extended in form of new software, sensors, actuators, and computing units. However, for fully automated vehicles the assumption that a driver can always assume control in case of a system malfunction or if the system cannot handle a specific traffic situation or road condition, is not true anymore.

In order to reach the highest demanded level of safety for these cyber-physical systems, new methodologies to develop automotive software are required. The current version of ISO 26262 [3] only *recommends* (for $\geq$ ASIL B) and *highly recommends* (for $\geq$ ASIL A) the use of a formal and semi-formal notation for the software architecture design. According to ISO 26262, a formal notation is a "description technique that has both its syntax and semantics completely defined" and a semi-formal notation is a "description technique that has its syntax completely defined, but its semantics definition may be incomplete". Given examples are graphical modeling approaches such as UML use case diagrams or block diagrams. However, the creators of the standard did not have autonomous vehicles in mind. Therefore, in order to be able to apply formal verification methods "to prove the correctness of a system against the formal specification of its required behavior", we investigate formal modeling notations to build "trust" (cf. Wagner and Koopman [4]).

Besides the benefits with regard to the quality of architectural specifications using formal methods, our approach also presents a framework for an automotive service-oriented architecture. We believe that a shift from traditional ECU-based (electronic control unit) development towards service-oriented development helps to address the high complexity of the development process better.

The ECU-based development approach is characterized by the notion of having "one ECU per function". Nowadays, several functions are running on a single ECU and are interacting freely across ECU boundaries. By just addressing ECUs, seen as black boxes that send and receive signals from other ECUs, it is often not clear which function on an ECU is addressed by a specific signal. Even more, in the course of evolution of functionality signals are used by a number of functions without clear specifications. This leads to a strong focus on technical details and a lack of clarity in higher-level, emergent properties of the whole system. We use the term of an *architectural*

978-1-5090-5729-0/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSA.2017.20

IEEE computer society

Authorized licensed use limited to: Malardalen University. Downloaded on January 04,2022 at 04:38:56 UTC from IEEE Xplore. Restrictions apply.

smell. According to Garcia et al. [5] an *architectural smell* "is a commonly (although not always intentionally) used architectural decision that negatively impacts system quality". This has a negative impact on system lifecycle properties, such as understandability, extensibility, maintainability, and reusability in particular with regard to continuous software and systems engineering. This circumstance occurs, for example, if the architecture follows the organizational structure rather than the technical structure in line with Conway's Law [6]:

> "Organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations."

Traditionally, car manufacturer considered systems as a set of subcomponents that are assembled by OEMs (cf. [7]). Thus, also the evolution of organizational structures tended to develop in line yielding rather independent business units. In the course of introducing new, innovative, highly-connected customer features, such as autonomous driving, OEMs are faced with both, technical and organizational challenges.

We ask ourselves whether the introduction of an automotive service-oriented architecture helps to address these challenges and which (i) stakeholders, (ii) tools, and (iii) processes have to be involved. This leads to the following research questions, which we pursue in this paper:

Research Questions:

RQ1 What are the main drivers for complexity in E/E architecture?

RQ2 Which interface specifications are required by architects?

RQ3 What are the expected benefits of introducing SOA with respect to architectural goals?

Contributions: This paper provides contributions to the following topics:

(i) Empirical evidence (from an interview study) that SOA is a promising methodology to meet the new challenge of highly interconnected, innovative automotive features;
(ii) A formal notion of services in architecture specifications;
(iii) A framework to design automotive functions using service composition in a layered architecture.

Outline: The remainder of this paper is structured as follows. We compare our approach to related work in Section II, then we describe the interview study design and results in Section III. Next, necessary background information is provided in Section IV before explaining the main contribution in Section V. As an example in Section VI, we sketch a part of the advanced driver assistance system (ADAS) of highly automated driving, namely the *takeover request* that requests the human operator to assume control of the vehicle. Section VII discusses the presented αSOA approach. Finally, Section VIII concludes the presented work.

II. Related Work

This work is related to approaches from different areas of the computer science community: this includes formal specification of systems and architectures, embedded systems, in particular SOA for embedded systems. Thus, we provide related work from these fields (partially overlapping).

Broy et al. [8] introduce a formal model for the specification of services. The formal notation in this work follows their notation. Malkis and Marmsoler [9] provide a theoretical approach towards services. Marmsoler et al. [10] present as a follow up work a formalization of the layered architecture style. Damm et al. [11] investigate the use and reuse of rich components in automotive embedded applications.

The idea to refactor automotive architectures is investigated by Vogelsang et al. [12] by identifying implicit communal components. Their work suggests to consider refactoring methods in order to make implicit communal components explicit by moving them to a dedicated layer which they call "platform component layer".

Bocchi et al. [13] provide an approach for service-oriented computing in the automotive domain. The authors use the domain-specific language SRML developed within the SENSORIA [14] project for service specification. Similar to our approach, they specify services at a high level, however their approach abstracts away service behavior.

AUTOSAR (AUTomotive Open System ARchitecture) [15] together with SOME/IP (Scalable service-Oriented MiddlewarE over IP) [16] is one attempt to introduce the notion of a service into automotive software engineering. However, this happens at a very technical implementation level rather than being usable for function or software developers. Moreover, SOME/IP is currently only available for Ethernet. Approaches known from business informatics—predominant web services using WSDL, SOAP, DPWS, and other technical solutions—are not capable to support specific automotive requirements like real-time, predictability, and safety just to mention the most important ones. Projects like SIRENA [17] and SOCRADES [18] worked on that topic, however it seems questionable whether heavy-weight web-services are appropriate in small devices and ECUs. This drawback is addressed by ϵSOA by Scholz et al. [19]. They provide a middleware platform including generation of stubs and service composition. However, the behavior of services is not addressed. Moreover, they lack a formal definition of their approach.

III. Interview Study

First, we describe the research method, then the grounded theory approach. Finally, the results are presented.

A. Research Method

In order to address the research questions and to propose an automotive SOA approach, we conducted 12 half-hour semi-structured interviews over a five-week period in October/November 2016 with (senior) system architects using an interview guide[1]. We had a mixture of closed (mostly on a 5-point Likert scale) as well as open questions. Table I lists the interviewees ($n = 12$) by area, with their relevant years of work experience.

[1] Interview guide available at http://www4.in.tum.de/~kugele/asoa/guide.pdf

Authorized licensed use limited to: Malardalen University. Downloaded on January 04,2022 at 04:38:56 UTC from IEEE Xplore. Restrictions apply.

TABLE I
INTERVIEWEES—THEIR DOMAINS AND YEARS OF EXPERIENCE.

Name	Domain	Experience [yrs.]
Harald	Drive architecture	> 8
Christian	Drive architecture	> 8
Marinus	Drive architecture	$3 - 5$
Rudolf	Infotainment architecture	> 8
Andreas[1]	Driver assistance architecture	$3 - 5$
Andreas[2]	Driver assistance architecture	$3 - 5$
Bernd[1]	Driver assistance architecture	$6 - 8$
Judith	Driver model architecture	$0 - 2$
Maik	Body & Comfort architecture	$0 - 2$
Bernd[2]	Body & Comfort architecture	$6 - 8$
Rainer	Body & Comfort, backend architecture	$3 - 5$
Manuel	Electrical energy network architecture	$0 - 2$

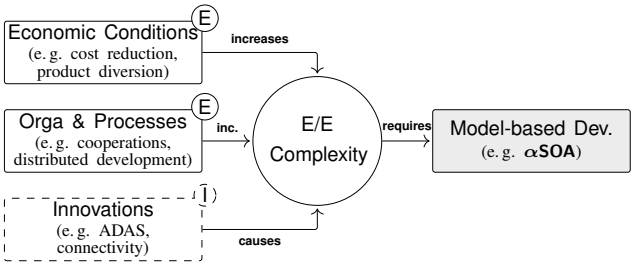


Fig. 1. Derived categories of the theory.

We had the permission to record 10 interviews, which we transcribed. For the remaining two, we took detailed notes. We followed the *grounded theory* approach in which we analyzed the interviews in order to derive a *theory* forming the basis for the presented automotive SOA (αSOA) approach. Also our experience and knowledge fed into the presented theory.

B. Data Analysis and Theory Building

We *coded* the interview transcripts to form a *grounded theory*. Therefore, we added *codes* to smaller transcript parts (sentences or paragraphs) (i. e., *open coding*). Next, we organized the found codes into *categories*. Codes and categories were then used to identify relationships among them (i. e., *axial coding*). These relationships were annotated with their explanations. The more transcribed interviews we analyzed, the more saturated our relationship model became. Finally, we built our theory by identifying one abstract core category which is related to all other categories (i. e., *selected coding*). As evaluation, we presented our results to the interviewees in order to receive feedback for further refinement. Figure 1 depicts the final, revised version of the derived theory.[2]

The core category is *E/E Complexity* which addresses the E/E architecture development complexity. This category is influenced by reasons, which stem from the three categories: (i) *economic* conditions (2 subcategories, 20 codes), (ii) *organizational* and *process*-related aspects (4 subcategories, 30 codes), and (iii) *innovations* (2 subcategories, 22 codes). As shown, we can distinguish between the two main categories of complexity drivers in the development of automotive E/E systems: (i) *intrinsic* (I) and (ii) *extrinsic* (E). On the one hand, intrinsic complexity of E/E architectures is *caused* by the nature of *innovations* as discussed in the introduction. On the other hand, a second source of complexity are *extrinsic* reasons imposed by the development framework. These unnecessarily *increase* complexity.

We observe that architects are less decisive than product line managers and purchasing departments within most OEMs. Both, the economic market and competitive pressure, as well as the organizational structure and de facto processes at automotive OEMs negatively influence the architectural quality:

1) Economic conditions: This category refers to *cost pressure* influencing the development of E/E architectures. This economic framework results in rigorously calculated target costs for hardware components. Particularly, an architectural design with scarce resources hampers *continuous architecture evolution* as repartitioning might occur more frequently. One example is partitioning of weak cohesive functions onto a single ECU. On the one hand, this efficiently uses resources, but, on the other hand, in the long run reduces architectural quality attributes such as maintainability and extensibility.

2) Organization and process: The next category does justice to the fact that besides cost pressure, there is also *time to market* and *innovation pressure*. Car companies react to that by allowing late changes in requirements. Thus, decisions are sometimes made less thoroughly, yielding to possible negative long-term effects, compromising architecture quality. This is substantiated by an interviewee's quote:

> "Thinking in balconies[3] rather than thinking in structures."

3) Innovations: This category of intrinsic drivers of complexity can hardly be avoided. New, innovative systems comprising e. g. connectivity and ADAS features make architectures inherently complex.

4) Model-based Development: The last identified category identifies an inadequacy of models that document the system under development. This refers to the fact that commonly, there is no single vehicle-wide *functional model* reflecting *all* functional interactions. Different department-specific tools do not share a common semantics hindering reasonable tool couplings or model transformations among them. As a consequence, system architects try to understand functions and their interrelation by interviewing function-specific experts and by accessing hardware-oriented models such as signal databases. However, they only reflect the interaction patterns between ECUs rather than functions running on them. Moreover, the interviewed architects expressed the wish to also have behavioral models to better understand automotive features. To address this situation, we derive the requirement for a *model-based development* approach. The presented automotive SOA – αSOA– is one way to facilitate this.

[2]Coding details available at http://www4.in.tum.de/~kugele/asoa/gt.pdf

[3]"Balcony" refers to a feature that was initially not planned and integrated after the architecture has been decided.

Authorized licensed use limited to: Malardalen University. Downloaded on January 04,2022 at 04:38:56 UTC from IEEE Xplore. Restrictions apply.

C. Study Results

The theory explains the complexity of developing modern E/E architectures as a combination of *extrinsic* (e. g. economic aspects) and *intrinsic* (e. g. innovations) factors. This answers research question **RQ1**. Both kinds can interact: E. g. an innovative market and tough competition can lead to a process that allows the approval of a late inclusion of innovative features. System architects expect huge benefits from introducing a vehicle-wide model of interacting functions using services (with a formal behavioral description), which answers **RQ2**: Architects require syntactic as well as semantic interface specifications. However, they pointed out that the introduction of a service-oriented architecture is no panacea for all E/E development problems. They proposed specific measures: (i) Introduce a *single*, well-defined framework for service design, (ii) adapt the current development process in order to better balance between pros and cons of late architecture changes with regard to long-term effects, (iii) give architects enough (i. e., more) flexibility in defining *their* architectural goals, in order to protect architectures from unsuitable changes, and (iv) establish a process that allows the transition from abstract service designs to implementation artifacts. Finally, SOA is considered as a means to support reuse of functionality as services need to be designed for multiple purposes. One direct quote describes the design of elements for multiple purposes in the following way:

> "Create decoupled, self-sufficient architectural elements that can be combined into bigger complex functions."

Besides open questions, we also had four closed questions. The questions refer to the following topics:

(i) Importance of models for operative work,
(ii) assessment of available methodologies and tools,
(iii) importance of hardware-independent services, and
(iv) years until penetration of SOA within the organization.

The results are discussed below and shown in Figure 2.

1) Models: The vast majority (92%) of the interviewees strongly agreed that the models being used are important. They are used to model (i) *effect chains*, i. e., the function's data-flow, (ii) *software architectures*, and (iii) *hardware topologies*.

2) Methodologies and Tools: However, more than 83% think that the methods and tools currently being used do not sufficiently satisfy the needs. In particular, they asked for a system-wide functional view and seamless tool integration.

3) Hardware-independence: Again, a vast majority (91%) agreed that services should be designed independently from a target hardware platform.

4) SOA-Thinking: The interviewed architects believe that fully thinking in services rather than ECUs can be established in the medium (3-8 yrs., 67%) to long run (> 8 yrs., 33%).

IV. Background, Terms, and Definitions

First the notion of a service-oriented architecture (SOA) is given, and then we provide a formal model of services closely following the FOCUS [20] approach as one way to model a multi-functional system using a formal notation. A concrete example is given in Section VI.

A. Service-Oriented Architecture

The notion of a *service-oriented architecture* is traditionally linked with application fields such as enterprise service architectures and business information systems in general. The main aim is to cope with complexity in big IT systems, which help implementing business processes. The idea of SOA in, for instance, enterprise services is to execute business processes as services offered by the technical infrastructure of companies. The following three steps are involved:

1. Step: Representing business processes as services.
2. Step: Model service architectures using modeling techniques such as UML or SoaML.
3. Step: Bringing services into technical systems by developing service interfaces with service description languages such as WSDL.

The promising aspects of service-orientation for development of vehicles is fueled by the idea of providing a range of services that can be used by several vehicle functions. Service-orientation concentrates on shared use of services with the idea of flexible interfaces that are not restricted to a specific functional context.

One example for a service which supports a vehicle function is given by a detection mechanism for the driver's presence, which is used by a wide range of vehicle functions. The functionality which starts the motor or driver assistance functions do not need a detailed description of how the driver's presence is detected. They only need the resulting information itself, which could be seen as a service that is integratable in any other functional context.

In order to introduce SOA in the automotive domain, we consider three steps:

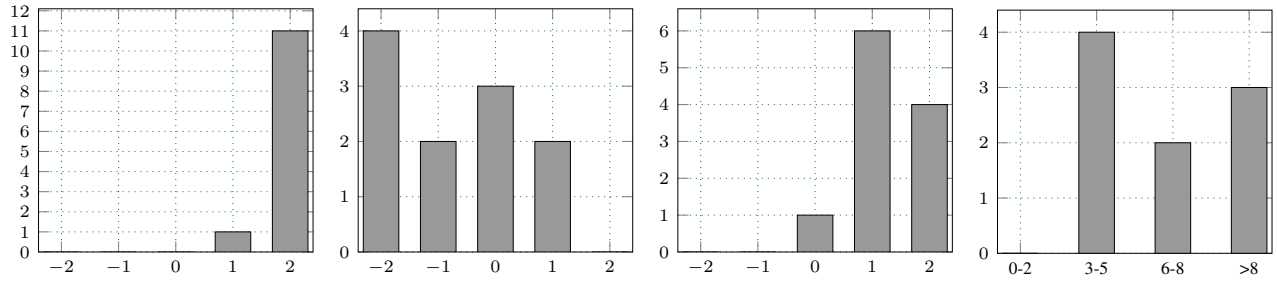
1. Step: Representing vehicle functions as services.
2. Step: Model service architectures on a syntactic and semantic level.
3. Step: Technically realize service execution on platforms like AUTOSAR.

This paper addresses the first two steps. Consequently, we concentrate on abstract levels rather than focusing on implementation details, i. e., we do not consider aspects of a particular middleware in the context of platforms like AUTOSAR, for instance.

B. Formal Specification

SoaML intends to model an architecture of services using logical components with syntactic interfaces that refer to method signatures on the basis of UML. For the field of business information systems this way of semi-formal modeling seems to be applicable as we do not deal with hard real-time systems. However, in the context of the automotive domain a lack in having a formal description is a disadvantage as we deal with safety-critical systems. We provide a formal service model closely following the FOCUS approach.

Authorized licensed use limited to: Malardalen University. Downloaded on January 04,2022 at 04:38:56 UTC from IEEE Xplore. Restrictions apply.



(a) Importance of models (b) Suitability of tools (c) Hardware independence (d) Years until SOA penetration

Fig. 2. (a) Models are important for operative work, (b) available methods and tools satisfy all demanded requirements, (c), a hardware independent service model is important, and (d) time duration for penetration of SOA. Note: -2 (strongly disagree), -1 (disagree), 0 (neutral), 1 (agree), 2 (strongly agree). (c) and (d) each with one abstention.

In the following, we refer to the product under development as *system*. This might be a complete car, a feature (i. e., function), or a part or it. The basic building blocks of our architectural model are that of *services* and *(sub)systems*. Systems may *provide* and *consume* services. The *provision/consumption relation* defines the interconnection between subsystems. Each service $\mathfrak{s} \in \texttt{SERVICE}$ has a *syntactic* and a *semantic interface*. The syntactic interface consists of typed ports that allow interaction with the environment. An *architecture* is defined by the interconnection of *subsystems*. Let PORT denote the set of all ports and TYPE the set of all types. The function type assigns a type to each port: $\mathrm{type}\colon \texttt{PORT} \to \texttt{TYPE}$.

Definition 1 (Syntactic Interface)**.** Let I be a set of typed input ports and O be the set of typed output ports. The pair (I, O) characterizes the syntactic interface of a (sub)system or service. The syntactic interface is denoted by $(I \triangleright O)$.

Definition 2 (Semantic Interface)**.** The logical and temporal behavior, i. e., the *semantic interface* of a service $\mathfrak{s}$ with syntactic interface $(I \triangleright O)$ is given by the function $f\colon \vec{I} \to \wp(\vec{O})$.

Definition 3 (System)**.** A *system* $s \in \texttt{SYSTEM}$ is defined by (i) its syntactic interface $(I \triangleright O)$ and (ii) an *architecture specification* $\mathcal{A}$ (cf. Definition 6), i. e., $s = ((I \triangleright O), \mathcal{A}) \in \texttt{SYSTEM}$.

Definition 4 (Service)**.** A *service* $\mathfrak{s}$ is characterized by its (i) syntactic interface $(I^{\mathfrak{s}} \triangleright O^{\mathfrak{s}})$ and its (ii) semantic interface given by the behavior function f, with $I^{\mathfrak{s}} \subseteq I$ and $O^{\mathfrak{s}} \subseteq O$ (written $(I^{\mathfrak{s}} \triangleright O^{\mathfrak{s}}) \sqsubseteq (I \triangleright O)$), where $(I \triangleright O)$ is the syntactic interface of the system s providing the service $\mathfrak{s}$.

Services are used to produce, consume, manipulate, or work with data, events, and physical values. Oftentimes we are only interested in which services are provided and consumed by a system. Then, we refer to the *service interface* instead of the syntactic interface.

Definition 5 (Service Interface)**.** The *service interface* of a (sub)system $s \in \texttt{SYSTEM}$ is denoted by $(P, C)^{\#}_{s}$, where P denotes the set of *provided* and C the set of *consumed* services. The service interface is an *abstraction* from the syntactic interface.

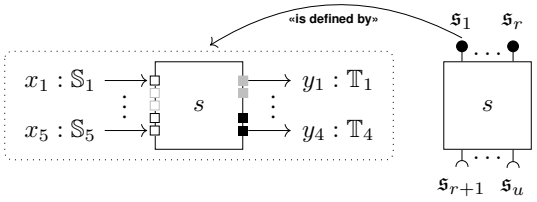


Fig. 3. System s with *provided* ($\mathfrak{s}_1$ to $\mathfrak{s}_r$) and *consumed* ($\mathfrak{s}_{r+1}$ to $\mathfrak{s}_u$) services. Black input (□) and output (■) ports are relevant for the service provision.

Definition 6 (Architecture Specification)**.** An *architecture specification* $\mathcal{A}$ for a system $s \in \texttt{SYSTEM}$ is a triple $(S, \sim, \mathbb{A})$ where $S \subseteq \texttt{SYSTEM}$ is the set of contained subsystems, $\sim \subseteq (S \times S)$ is the set of all *interconnected* subsystems, and $\mathbb{A}$ is the set of *interface assertions*. The following constraints hold: (i) *Acyclic subsystems*: $\forall s \in S$ with $(P, C)^{\#}_{s} : P \cap C = \emptyset$. (ii) *Unique provided services*: $\forall s, s' \in S, s \neq s'$ with $(P, C)^{\#}_{s}$ and $(P', C')^{\#}_{s'} : P \cap P' = \emptyset$.

Note, ISO 42010 [21] additionally describes architectures using *rationals* and *stakeholders*, which we omit in our model. Moreover, (sub)systems can again be further refined by providing an architecture specification for them. If a system $s \in S$ can *rely* on a successful provision of the consumed services, then it also *guarantees* a successful provision of the provided services. Figure 3 depicts a system s with its syntactic interface consisting of the input ports $x_1, \ldots, x_5$ of types $\mathbb{S}_1, \ldots, \mathbb{S}_5 \in \texttt{TYPE}$ and the output ports $y_1, \ldots, y_4$ of types $\mathbb{T}_1, \ldots, \mathbb{T}_4 \in \texttt{TYPE}$. The service $\mathfrak{s}_1$ is defined by a projection on the system's interface: $\left(\{x_1, x_4, x_5\}^{\mathfrak{s}_1} \triangleright \{y_3, y_4\}^{\mathfrak{s}_1}\right) \sqsubseteq (\{x_1, \ldots, x_5\} \triangleright \{y_1, \ldots, y_4\})$.

The given definition of an architectural specification uses a very general notion of interdependencies between subsystems. We can be more concrete and distinguish between different types of interrelation: *directed* ($\rightarrow$) and *mutual cooperating* ($\leftrightarrow$). Both are refinements of $\sim$.

Definition 7 (Directed Connection)**.** By convention, we write $s \rightarrow s'$ for $(s, s') \in \rightarrow$. Then, two systems $s, s' \in S, s \neq s'$ with $(P, C)^{\#}_{s}$ and $(P', C')^{\#}_{s'}$ are *directly connected*, i. e., $s \rightarrow$

Authorized licensed use limited to: Malardalen University. Downloaded on January 04,2022 at 04:38:56 UTC from IEEE Xplore. Restrictions apply.

s', iff (i) $C' \subseteq P$ and (ii) $P' \cap C = \emptyset$. In this case, s' *depends on* s. The relation $\rightarrow$ is *transitive*, *acyclic*, and *antisymmetric*. We denote by $\rightarrow^+$ the *transitive closure* of $\rightarrow$.

Definition 8 (Mutual Cooperation)**.** By convention, we write $s \leftrightarrow s'$ for $(s, s') \in \leftrightarrow$. Then, two systems $s, s' \in S, s \neq s'$ with $(P, C)^{\#}_{s}$ and $(P', C')^{\#}_{s'}$ are *mutually cooperating*, i. e., $s \leftrightarrow s'$, iff (i) $\exists \mathfrak{s} \in P \colon \mathfrak{s} \in C'$ and (ii) $\exists \mathfrak{s} \in P' \colon \mathfrak{s} \in C$.

Using those relations, any type of architecture can be described, e. g. a *layered architecture* or the *pipes and filters* pattern. For layered architectures, it is important to know which system is placed on which layer. Therefore, we introduce a function $\ell \colon S \rightarrow \mathbb{L}$, where $\mathbb{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$, which assigns a ℓayer to each system, $n \in \mathbb{N}^+$. Furthermore, let $\preceq$ be a *total order* defined on $\mathbb{L}$.

Definition 9 (Layered Architecture)**.** An architecture specification describes a *layered* architecture, iff (i) the architecture specification only contains *directed connections*, (ii) $\forall (s, s') \in \rightarrow \colon \nexists t \in S \colon (s' \rightarrow^+ t) \wedge (t \rightarrow^+ s)$, and (iii) $\forall (s, s') \in \rightarrow \colon \ell(s) \preceq \ell(s')$.

Within an architectural layer, we also allow *mutually cooperating* subsystems $s, s' \in S$ (see Figure 4), iff they are encapsulated into a new system s'', which provides services to systems $V \subseteq S$ on higher layers, i. e., $\forall v \in V \colon \ell(s'') \preceq \ell(v)$ and only consumes services from systems $U \subseteq S$ on lower layers, i. e., $\forall u \in U \colon \ell(u) \preceq \ell(s'')$.

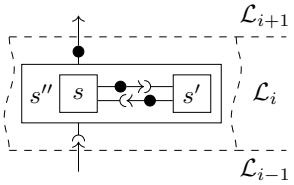


Fig. 4. Mutual cooperation in s'' within an architectural layer $\mathcal{L}_i$.

Using layered architectures to specify automotive systems has the advantage of having (i) *clear responsibilities*, (ii) the architecture specification follows *clear rules*, (iii) architectures are *better comprehensible*, and thus (iv) *complexity is reduced*.

Definition 10 (Strictness, Looseness)**.** A layered architecture—specified by $\mathcal{A} = (S, \rightarrow, \mathbb{A})$—is *strict*, iff $\forall (s, s') \in \rightarrow \colon \ell(s') - \ell(s) = 1$, otherwise it is *loose*.

V. Automotive SOA – αSOA

Next, the three steps of Section IV-A are explained.

A. Conceptual Design for αSOA

The conceptual design addresses the three steps of the original SOA idea as they help to distinguish between services, service models, and service implementations.

1) Representing Vehicle Functions as Services: As we deal with vehicle functions instead of business processes, we derive services from a functional perspective. Functional decomposition is an applicable methodology for this. Broy et al. [22] suggest an approach for functional decomposition of automotive systems. Moreover, also a use case-driven approach during object-oriented analysis is conceivable. In Figure 5 an example for the derivation of services from a high-level vehicle function *highly automated driving* (HAD) is given. Dependencies are stated following the feature diagram notation [23].

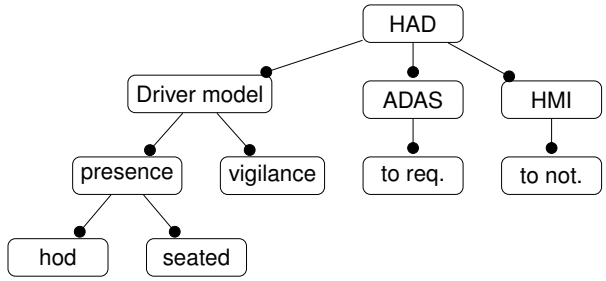


Fig. 5. Decomposition of the highly automated driving (HAD) function into services in FODA notation [23]. (hod: hands-off detection; to req: takeover request; to not: takeover notification)

2) Model Structure and Behavior of αSOAs: We distinguish between the structure, i. e., architecture, and the behavior of automotive systems that, as defined above, provide and consume services. On the one hand, we propose a generic framework that instantiates a layered architecture (cf. Section V-B) and, on the other hand, we use a formal modeling notation to describe the behavior of automotive services (see also Section VI).

We derive the importance of proposing an appropriate modeling approach from Section IV in which we stated that a simple adoption of UML or particularly SoaML as ways to model services is not satisfactory for automotive application due to the fact that they lack an adequately formal description.

3) Bringing Services into the Vehicle: This paper does not propose another middleware or other technical realization of the approach, but instead refers to existing approaches. The AUTOSAR platform in combination with the middleware SOME/IP provides SOA concepts. With the next generation of AUTOSAR—Adaptive AUTOSAR—the relevant concepts to describe SOA will be addressed. However, the suitability of this middleware in order to execute service design on a high level is rather poor (lower levels are supported well). One reason for this is the technical orientation of the protocol.

As Definition 9 suggests, there is a well-defined *design flow* of service provision and service consumption in *layered architectures*. For a more concrete architecture realization towards an implementation, the directed $s \rightarrow s'$ relations of an architecture specification have to be refined. Interactions between directed connections are modeled using *interaction patterns*. These patterns can be *instantiated* and further refined. Common interaction patterns are e. g. (i) send, (ii) receive, (iii) send/receive (request/response), and (iv) a mixture of data-driven protocols.

B. Service Framework and Service Kit

The *service framework* defines the general setting how automotive services are categorized, sorted, and how they are connected. It defines an instantiation of a layered architecture (cf. Definition 9) and facilitates the orchestration of multi-sensor fusion, processing, and multi-actuator coordination. Automotive functions, e. g. *highly automated driving*, are realized

Authorized licensed use limited to: Malardalen University. Downloaded on January 04,2022 at 04:38:56 UTC from IEEE Xplore. Restrictions apply.

by connecting systems via providing and consuming services from different *service kits* (see Definition 12).

Definition 11 (Service framework)**.** A *service framework* can formally be defined as a *≥3-tier architecture* building the implementation frame for an architecture specification (cf. Definition 6) of interconnected systems providing services. On the lowest layer, i. e., *Sense and Act* ($\mathcal{L}_1$), subsystems are modeled responsible for providing sensor and actuator interface services. The layer above, i. e., *Fusion and Coordination* ($\mathcal{L}_2$) models subsystems providing data fusion and data coordination services. Layers $\mathcal{L}_i, i \geq 3$ are used to model higher-level features that make use of lower-layer services.

Layers describe the degree of service abstraction which helps to distinguish between hardware-dependent services on the lowest layer and hardware- and technology-agnostic services on higher layers. Commonly shared services provide sensor and actuator interaction, data fusion and coordination, as well as arbitrary computations. Low-layered services are modeled with the expectation of rather low change rates, whereas higher-layer services are subject to more frequent changes.

Fusion and *coordination* systems are conceptually the opposite of each other. Fusion systems merge the consumed services and provide higher-value services. In contrast, coordination systems distribute (i. e. split) higher-value service behavior into (typically several) lower-layer, more concrete services. In Section VI we demonstrate how a service interaction can be realized in an abstract implementation, i. e., in an *effect chain*.

It is recommended to design *strict* layered architectures. However, sometimes it is necessary—especially when accessing sensor and actuator services—to relax this recommendation. That is, services provided on $\mathcal{L}_1$ are consumed from subsystems on all layers $\mathcal{L}_i, i > 1$ due to performance reasons for example. These cases, and especially their rationale, have to be documented very well in order to stay maintainable and extensible in the future.

Next, we introduce the notion of a *service kit*. The idea behind service kits is to *decouple* higher-level services from their technical and physical realization. This fosters *reuse* of services and allows to replace sensors, actuators, or technical components in general. Services on layer $\mathcal{L}_i, i \geq 2$ are in charge of guaranteeing and continuing service fulfillment and have to adapt if requirements change. Moreover, service kits break with the historically grown paradigm to structure vehicles according to domains that follow a mechanical engineering perspective e. g., *engine*, *body*, and *chassis*. For innovative functions like highly or fully automated driving, effect chains (i. e., causal effects at an abstract implementation level due to service consumptions) spread over all "classical" domains. Therefore, an assignment of sensors and actuators to those domains makes no sense anymore for highly-connected and distributed customer features. Instead, it is necessary to assign responsibilities rather than "domains" to technical components.

Definition 12 (Service Kit)**.** Let $\mathcal{K}$ be the set of service kits. A *service kit* $K \in \mathcal{K}$ is a *logical* collection of *technically* or *conceptually* related services, i. e., $K \subseteq \texttt{SERVICE}$. Service kits are *disjoint*, i. e., for all $K, K' \in \mathcal{K}$ it holds that $K \cap K' = \emptyset$.

For instance, the service kit *driver model* comprises all services needed to gather information about the driver, for instance his or her presence, his or her vigilance, or his or her fatigue. Other service kits might be that of the *environment*, that of *ADAS*, or that of *HMI*. Moreover, the hardware-dependent services for hardware interaction ($\mathcal{L}_1$) are in a distinct service kit for reuse. Figure 6 depicts the architecture of our running example.

Finding an appropriate granularity and thus functional scope of services arranged in service kits might be difficult and has to be considered on a case-by-case basis. Whether there is a universal rule is still an open research question. Metrics like *functional cohesion* (cf. [24]) may be applicable in this context and especially provide a good answer for the original idea of SOA in which the derivation of services from business processes is used as a principle (cf. Section IV-A). When transforming this principle to the automotive domain, we substitute processes by vehicle functions and may get a justification for functional cohesion as a metric.

In today's automotive architectures, equivalent or at least similar functions are often realized in different software components executed on ECUs. This might have several reasons: (i) engineers were not aware of already existing functionalities, (ii) redundancy has been introduced because of safety reasons (as a safety pattern, also together with dissimilarity), or (iii) because of performance requirements, such as low-latency communication via bus to the original software component. The first aspect can be solved with the service framework concept linking all vehicle functions to their respective services. Repositories can be searched for functions and services. The second aspect is system-intrinsic, i. e., redundancy is necessary for safety-critical applications and thus cannot be omitted in any case. Also, a dissimilar technical realization of a service might be necessary. In such a case, a single service is present in the service kit, but several *dissimilar* implementations are linked to it and deployed if necessary. Moreover, with regard to *safety* and *fail-operational* requirements, also *degraded* services have to be provided. The third aspect illustrates that also technical aspects have to be considered—not for service modeling, but for service deployment. Technical circumstances may yield in a redundant *instantiation* of a service in order to reduce, for example, bus load or to meet timing requirements.

VI. Example: Takeover Request Service

In this section, we demonstrate the modeling of a composed automotive service and its embedding in the service framework of the αSOA approach. We use services necessary to realize the functionality *highly automated driving* (HAD). In highly automated driving scenarios, it must always be possible that, in special situations, vehicle control is handed back over to

Authorized licensed use limited to: Malardalen University. Downloaded on January 04,2022 at 04:38:56 UTC from IEEE Xplore. Restrictions apply.

the human driver. This might be the case due to a system malfunction, but also—which is the much more likely case—if, for example, a highway ends and the next street segment is not approved for highly automated driving. In the following, we model this *takeover request service*, which is responsible for coordinating the scenario in which the driver takes over the control of the vehicle. If the driver ignores acoustic, visual, and tactile notifications the system changes into a *fail-safe* mode. In this mode, the vehicle's velocity is reduced and it is parked safely.

A. Used Service Kits

Besides knowledge about the vehicle's environment (stored in the environmental model service kit), information about the driver is of importance to avoid hazardous traffic situations. This information is kept in the so-called driver model, in our case in the *driver model service kit* (K_{DM}). As a driver assistance service, the **t**ake**o**ver **r**equest service ($\mathfrak{s}_{\mathrm{tor}}$) is located in the ADAS service kit (K_{ADAS}). The driver has to be informed about the vehicle's intention to release control. This notification service (**t**ake**o**ver **n**otification, $\mathfrak{s}_{\mathrm{ton}} \in K_{\mathrm{HMI}}$) is part of the HMI service kit. In the following, the contained services and service kits are explained in more detail.

1) Driver Model Service Kit: The driver model is an example for the aggregation of services that are linked with the same intention; more specifically, services for collecting information about the driver. For instance, the service to detect whether a driver is present or not ($\mathfrak{s}_{\mathrm{presence}} \in K_{\mathrm{DM}}$) is important for several systems like the engine control units or the gearbox control modules. These examples demonstrate that not only newly designed services need to know whether the driver is present, but also existing functions. With the advent of more and more ADAS functions which intend to make highly automated driving and autonomous driving possible, further aspects of the driver condition need to be gathered during runtime. Therefore, a model that has several views on the driver is currently being introduced. The main request for these views come from functions that support the driver while driving and therefore need to know the driver's constitution (e. g., presence, fatigue, and vigilance). All these services are provided by the driver model service kit. One could argue that, for example, a vehicle interior camera detecting the driver should also be mapped to this service kit. However, the camera is also used by an intrusion detection and alarm system. Therefore, all sensor and actuator services are part of a shared service kit.

2) ADAS Service Kit: This service kit contains services such as lane detection, adaptive cruise control, or remote parking. Moreover, the takeover request service ($\mathfrak{s}_{\mathrm{tor}} \in K_{\mathrm{ADAS}}$), which we are modeling in this example, is included in this service kit. The intention of this service is to notify the driver that a street segment approved for highly automated driving ends soon and that the driver has to take back control. A notification might also be necessary in case of a system malfunction. For this simplified example, we assume that the driver is informed using *acoustic* (gong), *visual* (signs and text at both the combi and head unit displays), and *tactile* (bucking e. g. by intermittent braking) signals. Note, for reasons of clarity, the realization of *bucking* is not given in detail. Basically, a combination of breaking and acceleration using the dynamic stability control yields this behavior.

3) HMI and Hardware Interaction Service Kit: **H**ard**w**are **i**nteraction services are located in a respective service kit (K_{HWI}). The example uses a sensor to detect whether the driver keeps both hands on the steering wheel (encapsulated in the **h**ands **o**ff **d**etection service $\mathfrak{s}_{\mathrm{hod}} \in K_{\mathrm{HWI}}$). Additionally, a **s**itting **m**at sensor with respective **d**etection service ($\mathfrak{s}_{\mathrm{smd}} \in K_{\mathrm{HWI}}$) in the driver's seat determines whether or not the driver is sitting at the driving position. If the driver pushes an **ack**nowledgement button at the steering wheel, an acknowledgement event is fired by service $\mathfrak{s}_{\mathrm{ack}}$, also in K_{HWI}. As we can see, this architecture is *not strict*. The appropriate control of the notification is done using the $\mathfrak{s}_{\mathrm{ton}}$ service from the human-machine-interface service kit. It is responsible for actuator interaction using the respective services $\mathfrak{s}_{\mathrm{an}}$ (**a**coustic **n**otification) and $\mathfrak{s}_{\mathrm{vn}}$ (**v**isual **n**otification) from K_{HWI}. Figure 6 depicts the architecture of the takeover request service.

Sensor fusion is done on layer $\mathcal{L}_2$, which provides the fused, more abstract information about the driver's presence ($\mathfrak{s}_{\mathrm{presence}}$).

B. Behavioral Specification

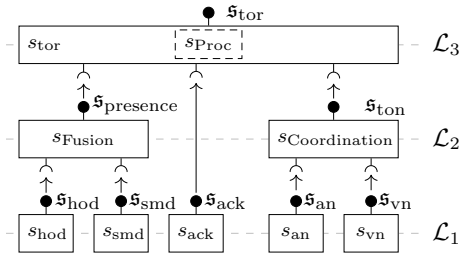
Formal behavioral specification is an additional pillar of this approach. We specify the takeover request using a state machine model. A three-stage driver alerting mechanism is used which is explained next.

- Stage 1 Prompting the driver with acoustic (a) and visual (v) signals to take over vehicle control. This alerting is performed in any case.
- Stage 2 If stage 1 is not successful, the vehicle starts bucking (b) (e. g. using break and motor).
- Stage 3 If stage 2 is not successful, the vehicle turns into a fail-safe mode, reduces its velocity and parks safely.

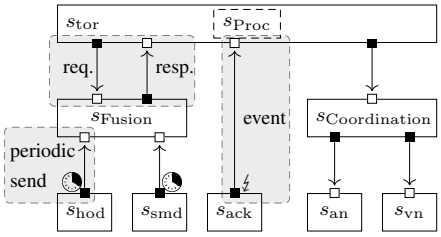
Depending on the driver model's state and in particular on the presence service, the time spans between the promoting stages differ. Figure 7 gives the detailed specification. The initial state req denotes the start of the takeover request, i. e., the time begins to count: $t := 0$. The other states have the following meaning:

- HS ... both hands are on the steering wheel driver is seated on the driver's seat
- $\bar{H}S$ ... both hands are **not** on the steering wheel driver is seated on the driver's seat
- $\bar{H}\bar{S}$ ... both hands are **not** on the steering wheel driver is **not** seated on the driver's seat
- ack ... the driver **ack**nowledges the takeover request
- $stop$... the driver does **not** acknowledge the takeover request within the time boundary

Authorized licensed use limited to: Malardalen University. Downloaded on January 04,2022 at 04:38:56 UTC from IEEE Xplore. Restrictions apply.



(a) Service interface view



(b) Interaction patterns in logical subsystems (gray)

Fig. 6. TOR architecture. *Legend*: **t**ake **o**ver **r**eqest, **t**ake **o**ver **n**otification, **h**ands **o**ff **d**etection, **s**eat **m**at **d**etection, **a**custic and **v**isual **n**otification.

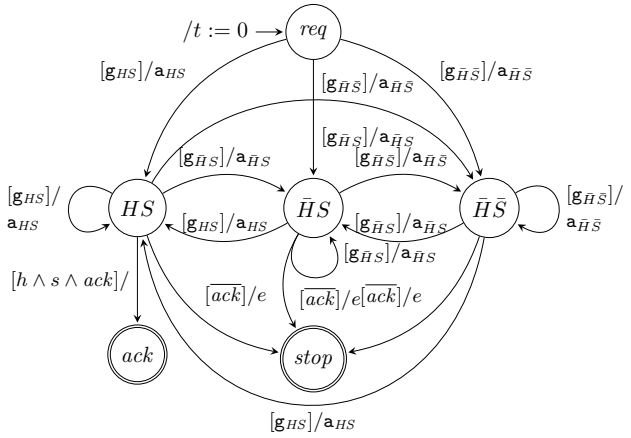


Fig. 7. Takeover request specification.
Legend: (h)ands on, (s)eated, (e)mergency stop, (*ack*)nowledge.

The three guard/action pairs $[\mathtt{g}_{HS}]/\mathtt{a}_{HS}$, $[\mathtt{g}_{\bar{H}S}]/\mathtt{a}_{\bar{H}S}$, and $[\mathtt{g}_{\bar{H}\bar{S}}]/\mathtt{a}_{\bar{H}\bar{S}}$ that define the promoting stages and time spans (in s) are defined as follows:

$$[\mathtt{g}_{HS}]/\mathtt{a}_{HS} \equiv \begin{cases} [h \wedge s \wedge (0 \leq t < 5)]/ & a, v \\ [h \wedge s \wedge (5 \leq t < 25)]/ & a, v, b \end{cases} \quad (1)$$

$$[\mathtt{g}_{\bar{H}S}]/\mathtt{a}_{\bar{H}S} \equiv \begin{cases} [\bar{h} \wedge s \wedge (0 \leq t < 5)]/ & a, v, \\ [\bar{h} \wedge s \wedge (5 \leq t < 20)]/ & a, v, b \end{cases} \quad (2)$$

$$[\mathtt{g}_{\bar{H}\bar{S}}]/\mathtt{a}_{\bar{H}\bar{S}} \equiv \begin{cases} [\bar{h} \wedge \bar{s} \wedge (0 \leq t < 5)]/ & a, v \\ [\bar{h} \wedge \bar{s} \wedge (5 \leq t < 15)]/ & a, v, b \end{cases} \quad (3)$$

Where ack denotes the predicate, that the driver acknowledges the takeover request in time, i. e., $t \leq 30$. The state ack can only be reached if the driver acknowledges the takeover request in time (by pushing a button on the steering wheel), is seated, and has his hands at the steering wheel. The emergency $stop$ state is always reached, whenever the driver does not acknowledge the takeover request in time.

The structure of the takeover request service is illustrated in Figure 6b. The logical core behavior is encapsulated in the subsystem s_{Proc}. The takeover request service is modeled using a state machine diagram. State machine diagrams respect an execution model that offers the option to describe functional behavior in a formal way. As other forms of formal modeling often lack in understandability and are therefore not widely used, state machine diagrams can be considered as compromise between specification quality and suitability with regard to a practical engineering context.

VII. Discussion

Internal Validity: A major threat to validity concerns the conduction of interviews that might be biased as questions may already suggest to favor SOA as a promising approach for the automotive domain. To prevent this, we tried to find out what are typical industry approaches for the design of system architecture at BMW without a direct link to the topic of SOA. Nevertheless, it was possible to derive a theory concerning the feasibility of an introduction of SOA in the automotive domain including possible benefits for the operational work of system architects. Another threat to validity addresses the selection of the interviewees that might be biased as our theory may just be derivable for our specific group of interviewees. In order to prevent this, we tried to choose a balanced selection of interviewees regarding their domain of expertise. Although we represent each domain at our industrial partner, except for the domain of the environmental model, by at least one system architect, we also concede that the domains of drive architecture, driver assistance, and body & comfort are represented disproportionately.

External Validity: A threat to the external validity is the chosen example. To prevent this, we tried to find out if we can apply our approach to already existing functional models at our industrial partner. Therefore, we investigated existing functional models from different departments in order to evaluate the suitability of our approach. With respect to the high degree of abstraction for our service model, which decouples high-level services from their technical or physical implementation, it is possible to apply our suggested approach. The reasons for that are linked with existing modeling approaches in all departments that model functions as effect chains based on data flow. This approach is closer to a technical or physical realization than our service model. Therefore, it is possible to abstract these existing models into our notion of services.

VIII. Conclusion

In this paper, we presented the result of an interview study conducted at BMW Group indicating the need for a seamless

Authorized licensed use limited to: Malardalen University. Downloaded on January 04,2022 at 04:38:56 UTC from IEEE Xplore. Restrictions apply.

integration of *processes*, *methods*, and *tools* supporting views on the complete E/E architecture. Additionally, we provided a formal definition for the notion of a service in automotive software architectures. A service-oriented approach shows promise for several salient goals in architectural work (cf. **RQ3**).

First, *uniformity*, e. g. in how to access any and all data on the one hand, or cause the remote execution of a certain functionality on the other hand, aims at establishing an intuitive way to think about interdependence. Minimizing unnecessary heterogeneity for wide-spread and similar needs allows architects and developers to "guess it right" when developing a new architectural dependence and thus aids in achieving a system-wide perspective.

Second, *separation of concerns* by "black-boxing" implementation details aims at decoupling development cycles and teams. A technical service contract is helpful for both, the service provider as well as the consumer, because it represents the culmination of an abstraction that both sides can fully understand and agree on.

Third, enable *computational techniques* for validation and conformity testing of different architectural alternatives: When the first two goals are sufficiently accomplished so that we can think of an architecture as a network of dependencies between consistent subsystems, it is possible to apply algorithms for evaluation and even optimization (see e. g. [25]) in order to validate systemic properties of a given architecture. Most importantly, such properties will be the overall behavior of the system, its performance, specific qualities, or adherence to constraints. Using the provided *interface assertions*, if a correct implementation is a refinement of the architecture specification, then properties and proofs on the architecture specification are also true on the implementation. We recently showed in [26] how to verify architectures.

With these goals the αSOA approach enables new possibilities of collaboration, OEM-internally as well as between OEM and Tier-1. A further potential is the faster integration of software-driven innovations, as the approach facilitates continuous service deployment.

Acknowledgment

This work was supported by the BMW Group. We thank our project partners for putting our research into an innovative practical context.

References

[1] M. Broy, S. Kirstan, H. Krcmar, and B. Schätz, "What is the benefit of a model-based design of embedded software systems in the car industry?" *Software Design and Development: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications*, p. 310, 2013.

[2] Society of Automotive Engineers, *Taxonomy and Definitions for Terms Related to On-road Motor Vehicle Automated Driving Systems, SAE Standard J3016*, 2014.

[3] ISO, "Road vehicles–Functional safety (ISO 26262)," 2011.

[4] M. Wagner and P. Koopman, *A Philosophy for Developing Trust in Self-driving Cars*. Springer International Publishing, 2015, pp. 163–171.

[5] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, "Identifying architectural bad smells," in *Proceedings of the 2009 European Conference on Software Maintenance and Reengineering*, ser. CSMR '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 255–258.

[6] M. E. Conway, "How Do Committees Invent?" *Datamation*, Apr. 1968.

[7] M. Broy, "Challenges in automotive software engineering," in *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*, L. J. Osterweil, H. D. Rombach, and M. L. Soffa, Eds. ACM, 2006, pp. 33–42.

[8] M. Broy, I. H. Krüger, and M. Meisinger, "A formal model of services," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 1, Feb. 2007.

[9] A. Malkis and D. Marmsoler, "A model of service-oriented architectures," in *2015 IX Brazilian Symposium on Components, Architectures and Reuse Software, SBCARS 2015, Belo Horizonte, Minas Gerais, Brazil, September 21-22, 2015*. IEEE Computer Society, 2015, pp. 110–119.

[10] D. Marmsoler, A. Malkis, and J. Eckhardt, "A model of layered architectures," in *Proceedings 12th International Workshop on Formal Engineering approaches to Software Components and Architectures, FESCA 2015, London, United Kingdom, April 12th, 2015.*, ser. EPTCS, B. Buhnova, L. Happe, and J. Kofron, Eds., vol. 178, 2015, pp. 47–61.

[11] W. Damm, A. Votintseva, A. Metzner, B. Josko, T. Peikenkamp, and E. Böde, "Boosting re–use of embedded automotive applications through rich components," in *FIT 2005 – Foundations of Interface Technologies*, Aug 2005.

[12] A. Vogelsang, H. Femmer, and M. Junker, "Characterizing implicit communal components as technical debt in automotive software systems," in *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, April 2016, pp. 31–40.

[13] L. Bocchi, J. L. Fiadeiro, and A. Lopes, "Service-oriented modelling of automotive systems," in *2008 32nd Annual IEEE International Computer Software and Applications Conference*, July 2008, pp. 1059–1064.

[14] "SENSORIA: A novel engineering approach to service-oriented computing." [Online]. Available: http://www.sensoria-ist.eu

[15] "AUTOSAR: Automotive Open System Architecture," http://www.autosar.org.

[16] L. Völker, "Scalable service-Oriented MiddlewarE over IP." [Online]. Available: http://some-ip.com/

[17] "SIRENA: Service Infrastructure for Real-time Embedded Networked Applications." [Online]. Available: https://itea3.org/project/sirena.html

[18] "SOCRADES: Service-Oriented Cross-layer infRAstructure for Distributed smart Embedded deviceS." [Online]. Available: http://www.socrades.net/

[19] A. Scholz, C. Buckl, S. Sommer, A. Kemper, A. Knoll, J. Heuer, and A. Schmitt, "ϵSOA - service oriented architectures adapted for embedded networks," in *Proceedings of the 7th International Conference on Industrial Informatics)*, 2009.

[20] M. Broy and K. Stølen, *Specification and development of interactive systems:* FOCUS *on streams, interfaces, and refinement*. New York: Springer-Verlag, 2001.

[21] "ISO/IEC/IEEE Systems and software engineering – Architecture description," *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pp. 1–46, Dec 2011.

[22] M. Broy, M. Feilkas, J. Grünbauer, A. Gruler, A. Harhurin, J. Hartmann, B. Penzenstadler, B. Schätz, and D. Wild, "Umfassendes Architekturmodell für das Engineering eingebetteter Software-intensiver Systeme," Jun. 2008.

[23] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-90-TR-021, 1990.

[24] E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, 1st ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1979.

[25] S. Kugele, G. Pucea, R. Popa, L. Dieudonné, and H. Eckardt, "On the deployment problem of embedded systems," in *13. ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2015, Austin, TX, USA, September 21-23, 2015*. IEEE, 2015, pp. 158–167.

[26] S. Kugele, D. Marmsoler, N. Mata, and K. Werther, "Verification of component architectures using mode-based contracts," in *2016 ACM/IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2016, Kanpur, India, November 18-20, 2016*. IEEE, 2016, pp. 133–142.

Authorized licensed use limited to: Malardalen University. Downloaded on January 04,2022 at 04:38:56 UTC from IEEE Xplore. Restrictions apply.