

Playground for Early Automotive Service Architecture Design and Evaluation

Vadim Cebotari¹ and Stefan Kugele²

Abstract—Context: We consider the structure of service-oriented architectures in vehicular software. **Aim:** We aim at evaluating the structure and grouping of service architectures. **Method:** We propose and discuss architectural metrics tailored towards automotive service-oriented architectures. We apply the metrics on an adaptive cruise control case example extracted from the AUTOSAR standard. **Results:** The application of the proposed metrics to two different service groupings for ACC points clearly to the same service grouping that we consider, after a thorough analysis, to be better with respect to coupling and cohesion attributes. **Conclusion:** We demonstrate the usefulness of proposed service group metrics in early design phases of the development process and validate the metrics on the case example of an adaptive cruise control function.

I. INTRODUCTION

During the last decades, more and more software-controlled functions were included in vehicles. The ever-growing availability of computing resources, memory, and newest technologies allows for new levels of *automated* and *intelligent* systems. Current trends in the automotive industry are introducing new, increasingly complex software functions into vehicles [1]. Automated, connected, electric, and shared vehicles are the drivers and push technological developments. The functional scope spreads from infotainment and comfort functions over functions increasing the vehicle's safety to advanced driver assistance systems (ADAS). The latter mentioned category has extended its capabilities more and more over time. The ultimate goal is to reach the highest level 5 (according to SAE J3016 [2]) of driving automation, i.e., robot taxi. Therefore, a multitude of new sensors, such as HD cameras and Lidar scanners need to be integrated into the technical vehicle architecture. These new sensors, as well as the already installed sensors, produce several gigabytes of raw data per hour. A significant hub of on-board signals accompanies the growth of the data to be transmitted and the bandwidth required. For the upcoming level-3 vehicles, one expects to have more than 30.000 signals, which corresponds to almost a doubling of current values. Current signal-based development paradigms that focus on ECUs (electronic control units) rather than on services, however, cannot manage the immense functional stroke in a competitive, economic, and safe way anymore.

Data is—partly redundantly—distributed among the whole in-vehicle network consisting of a complex gateway architecture interconnected with heterogeneous bus systems. Data

about the vehicle itself, its passengers, and its environment is mandatory for (i) well-being functions, (ii) intelligent chauffeur, (iii) predictive maintenance, (iv) holistic passenger and environmental model required for automated driving. As complex functions and their produced data are distributed and not solely deployed onto a distinctive ECU, there is currently no unified way to access data.

We argue that a *service-oriented architecture* (SOA) eases the development of those functions and facilitates the introduction of dynamicity at runtime (i.e. dynamic discovery of a backup service in a failover scenario when the main service fails). By composing complex features using modular services, reuse is fostered. Moreover, modular design allows for continuously improving the system even after production in the field. Continuous service integration and delivery helps to improve the quality and speed at which automotive software-based innovations are delivered to the customers' vehicles—going along with the DevOps way of thinking.

With the introduction of a service-oriented architecture in automotive E/E systems, amongst others, four main benefits can be realised: (i) First, a hierarchical in-vehicle function-/software architecture is realised; (ii) Second, the complex interplay of services can be understood, allowing for better analysability, impact analysis, reusability, and extensibility; (iii) Third, a *unified* way for data access is supported by exposed service interfaces; (iv) Fourth, innovations are delivered continuously at a higher frequency to customers.

Service-oriented architectures consist of a set of services that communicate with each other through the provided service interfaces and are deployed onto distributed computing nodes. For business information systems, this architectural runtime model might be appropriate, but for safety-critical cyber-physical systems, it poses many challenges due to security, safety, and real-time requirements. Challenging questions are: (i) Which service is allowed to subscribe to other services? (ii) Are there any restrictions that require services to be deployed on the same node (e.g. real-time requirements, latency, bus load), or on different nodes (e.g. safety requirements such as dislocality)? (iii) Are services running within the vehicle or remote, subsuming the infrastructure, other vehicles, fog/edge devices, or backend facilities? (iv) Are SOA design principles such as *loose-coupling*, *high cohesion*, and a *well-chosen service granularity* met?

In [3], we tackled these questions at design time by building service groups. To select the most appropriate service grouping (w.r.t. the challenges) we need effective means to evaluate the quality of the derived service groupings.

In this paper, we propose a *metrics-based* approach to

The authors contributed equally to this work.

¹Vadim Cebotari is with the Department of Informatics, Technical University of Munich, Germany vadim.cebotari@tum.de

²Stefan Kugele is with the Department of Informatics, Technical University of Munich, Germany stefan.kugele@tum.de

perform analyses of early architectural design candidates. In this context, we use the term “*playground*” to reflect the idea of early experimentation. This helps in the dimensioning of architectures and supports design decisions.

A. Research Objective

- RQ1** How can the analysis and metric-based evaluation of derived service groups benefit the development of qualitative architectural designs for distributed service-oriented automotive E/E systems?
- RQ2** Which set of service group metrics can be applied to measure the coupling and cohesion quality attributes of architectural designs for a distributed SOA?

B. Contribution

This paper provides contributions to the following topics:

- (i) we motivate the creation, analysis and metric-based evaluation of service groupings in order to facilitate the development of loosely coupled and highly cohesive distributed service-oriented systems;
- (ii) we propose a set of metrics to measure the coupling and cohesion quality attributes of derived service groups.

C. Outline

The remainder of this paper is structured as follows. Section II relates the presented approach with related work. Next, in Section III, we sketch necessary preliminaries followed by the main contribution in Section IV. Section V introduces our case example, a service architecture for a realistic adaptive cruise control (ACC) automotive function. The gained results are presented and discussed in Sections V-B and V-C. Finally, we conclude in Section VI.

II. RELATED WORK

The ideas to structure and modularise systems go back to seminal works by Dijkstra [4] and Parnas [5] in 1972. Decoupling, structuring, and hierarchisation are some examples of simplifications in system design.

The concept of building service groups is not new. A formal model for service grouping is introduced in [3]. The AUTOSAR Adaptive Platform uses extensively the concept of grouping meta-classes in the application design [6], i.e., applications are built as hierarchical compositions of software components or application errors are arranged in error groups to ease their reusability.

In this paper, we provide a set of metrics to evaluate the quality of the generated service groups. To the best of our knowledge, we are not aware of targeted work on metrics for service groups in service-oriented architectures. Still, there has been done much work on defining service level metrics. However, automotive specific challenges (e.g. deployment and safety constraints) are not well-addressed in existing work due to the lack of specification of service groups.

In the following, we review the most relevant work on service level metrics. Hirzalla et al. [7] propose a set of metrics to measure the complexity, flexibility, and agility of SOA solutions. The calculation of metrics is based on

SOA design (e.g. number of services and service interfaces, services composition, and service binding) and infrastructure (ESB, support for transactions) information. Liu and Traore [8] define a metric to measure the complexity of composite services. The metric is based on the information about inner dependency relationships that exist within the composite service.

Several researchers propose metrics to evaluate the quality attribute of *coupling* in service-oriented systems. Qian et al. [9] introduce a set of metrics to measure the coupling between services in a service composition based on the service stateness (stateless/stateful), persistent data usage, required service interfaces, and invocation types (synchronous/asynchronous). Pereplechikov et al. [10] focus on metrics that measure the number and type of connections between various design elements (service implementation elements/interfaces) and services. Sindhgatta et al. [11] define a set of coupling metrics based on information extracted from service interfaces and the underlying domain data model.

Cohesion is another SOA quality attribute for which several metrics were proposed [11], [12]. The defined cohesion metrics are based on the usage of the same input parameter types and implementation elements of service operations, as well as service interface usage patterns (number and order of invoked operations) of service consumers.

Further, research to derive metrics for evaluation of reusability, composability, and granularity of services in SOA solutions has been done [11], [13], [14].

III. PRELIMINARIES

A. Service Modelling

The object of investigation of this work is the interplay of provided and consumed services of an automotive service-oriented architecture. Within this architecture, sets of services are clustered together into *service groups*. Let SERVICE denote the set of all services. A service $s \in \text{SERVICE}$ provides a set of *service interfaces* $s^\bullet \subseteq \text{INTERFACE}$ where INTERFACE denotes the set of all provided interfaces. Furthermore, we denote with s_ℓ^\bullet *local service interfaces*, i.e., service interfaces that are only used within the same service group.

Moreover, let GROUP be the set of all groups $g \in \text{GROUP}$ and GROUP be a partitioning of the set SERVICE , i.e., SERVICE is a disjoint union of the subsets (groups g of GROUP). Sometimes, it is necessary to talk about provided service interfaces except those from a considered service group (the local ones). The group will always be clear from the context. So, let s_ℓ^\bullet be this set: $s_\ell^\bullet = s^\bullet \setminus s_\ell^\bullet$.

Figure 1 depicts two groups with three services. Each service provides a service interface i_1 , i_2 , and i_3 .

Within this paper, we specify that we only count service interfaces if at least one other service uses them. This is only true for service interfaces i_1 and i_2 .

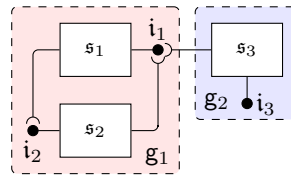


Fig. 1: Example grouping

B. Properties for Well-defined Metrics

In the next section, we introduce a set of metrics to measure the quality of generated service groups. Our primary focus in this paper is on coupling (X) and cohesion (C) of service groups, since these two quality attributes are decisive in determining which service grouping is optimal with respect to the bus load, extensibility, and service reallocability of an automotive service-oriented architecture.

Briand et al. [15] define for different quality attributes sets of properties that metrics have to fulfil in order to be considered *rigorous* and *well-founded*. For coupling metrics, the following properties are defined: (X.1) non-negativity, (X.2) null value, (X.3) monotonicity, (X.4) merging of service groups, and (X.5) disjoint service group additivity. For cohesion metrics, Briand et al. define (C.1) non-negativity and normalisation, (C.2) null value, (C.3) monotonicity, and (C.4) merging of service groups. We specify for each metric, which class of metrics (cohesion or coupling) it belongs to, and verify appropriate properties, as described above.

Most of the introduced metrics satisfy all above-mentioned properties. The coupling metrics introduced in Section IV-A.7 and Section IV-A.8 do not satisfy properties (X.4) and (X.5). In Section IV-B, we explain these two metrics and in Section V we show their usefulness on the case example of an adaptive cruise control function.

The *normalisation* property (C.1) is required to ensure the measure is independent of the size of service groups. The *null-value* property requires that the metric is null when there are no relationships to other groups (X.2) or between services inside the group (C.2). The *monotonicity* property states that by adding relationships to other service groups (X.3) or inside the service group (C.3), the coupling resp. cohesion can only increase. The *merging of service groups* property (X.4 and C.4) requires that the coupling resp. cohesion of the service group obtained by merging two service groups is not greater than the sum of coupling resp. cohesion values of the two original service groups. The *disjoint service group additivity* property (X.5) is a special case of the property (X.4), when no relationships between merged groups exist.

IV. APPROACH

In the future, intelligent vehicles will be integrated into external backend and infrastructure systems. Due to improved functionality and connection to evolving external systems, the deployment of innovative, new functions and updates of existing functions during vehicle lifetime will be necessary. Vehicle functions will become more complex and compute-intensive. At the same time, both on-board and off-board functions used in the vehicle must fulfil strict integrity, security, and safety requirements [16]. These requirements lead to the necessity to manage (1) memory allocation to sets of processes, (2) allocation of sets of processes to CPU cores, and (3) access control to service interfaces. These aspects are easy to manage, if the vehicle software does not change significantly during vehicle lifetime. However, it becomes much more complicated when the vehicle software is subject to frequent changes (e. g. instantiation

TABLE I: Benefits of service groups at different stages of the system life-cycle.

Design Time	Aspects
The hierarchical design allows to inherit properties from (the) parent group(s).	👤 ⌚
Groups can be interpreted as <i>virtual</i> ECUs, SWCs, or processes depending on the level within the hierarchy: • top level: virtual ECU (i. e., container for SWCs); • mid level: virtual SWC (i. e., container for processes); • bottom level: virtual process (i. e., container for threads);	🏠 📊 🔍
Runtime	
Flexible memory allocation to sets of processes.	🔍 ☑
Flexible allocation of sets of processes to CPU cores.	🔍 ☑
Flexible configuration of access control to service interfaces.	🔍 ☑
👤 Usability ⌚ Development time 🏠 Playground 📊 metrics 🔍 Analysability ☑ Testability	

of new services or updates of existing services). Improved or extended vehicle software may pose new requirements regarding aspects (1)–(3), i. e., a new version of the service requires, for instance, more computational power or memory. Service groups facilitate the management at runtime. The memory is then allocated not to single POSIX¹ processes (runtime representation of a service), but sets of processes (runtime representation of service groups). Those can be Adaptive AUTOSAR partitions or resource groups which are assigned a maximum quota of memory/CPU time. Regardless of the changes inside the service groups (instantiation, update, deactivation of services), the allocated memory per service group remains unchanged. Specification of system configuration parameters, such as memory allocation to processes, allocation of processes to CPU cores, and access control to service interfaces, on the level of service groups, eases the deployment of new services and the update of existing services considerably. By assigning services to service groups, services inherit automatically service group properties. System integrators are exempted from the task to manually configure each service, which leads to faster and more secure service deployments. Analysability and testability of applications are also improved, since service groups are fixed, and there is no need to perform new tests each time a new service is deployed. A well-chosen cut into groups brings clear benefits during both design-time and runtime. Some important ones are listed in Tab. I. In the next section, we will elaborate on different metrics for service group evaluation.

A. Service Group Metrics

In the following, we introduce metrics to evaluate automotive service architectures early in the design phase. To do so, we discuss the properties introduced in Section III-B for each metric and give an example. Please note that the introduced metrics are either single-digit or double-digit functions. We, therefore, use the appropriate function notation.

¹We use the terms of the AUTOSAR Adaptive Platform as reference.

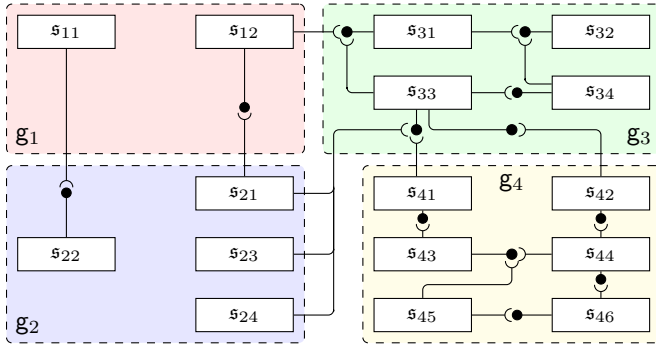


Fig. 2: Example Grouping

1) *Service Group Interface Count (ic)*: This metric (derived from [7]) counts for a given service group g the number of provided service interfaces and is defined by the function $ic: \text{GROUP} \rightarrow \mathbb{N}_0$,

$$ic(g) = \sum_{s \in g} |s^\bullet|. \quad (1)$$

The metric belongs to the class of coupling metrics.

Metric Properties: The property (X.1) is evident. Property (X.2) is satisfied, since the metric is null when the services contained in the service group do not provide any interfaces. The property (X.3) is also satisfied, since adding new interfaces to services contained in the service group can only increase the number of provided service interfaces. Property (X.4) states that the number of provided service interfaces of a service group g obtained by merging the service groups g_1 and g_2 is not greater than the sum of provided service interfaces of original service groups g_1 and g_2 : $ic(g) \leq ic(g_1) + ic(g_2)$. This is true. Actually, we have a stronger relationship: $ic(g) = ic(g_1) + ic(g_2)$. Thus, both properties (X.4) and (X.5) are satisfied. The total number of provided service interfaces does not change by merging two service groups. The only thing that changes, if there are “requires”-relationships between service groups to be merged, is that some relationships change from intergroup to intragroup “requires”-relationships.

Example: In Fig. 2, a SOA built of 16 services that are subdivided into four service groups g_1 , g_2 , g_3 , and g_4 is depicted. The service s_{33} has two service interfaces; all other services have at most one service interface. The metric ic has following values for the service groups illustrated in Fig. 2: $ic(g_1) = 1$, $ic(g_2) = 1$, $ic(g_3) = 5$ and $ic(g_4) = 5$.

2) *Service Group Local Interface Count (lic)*: The metric lic (derived from [10]) denotes the number of local service interfaces of a group. The metric is defined by the function $lic: \text{GROUP} \rightarrow \mathbb{N}_0$,

$$lic(g) = \sum_{s \in g} |s^\bullet|. \quad (2)$$

The metric lic is a cohesion metric.

Metric Properties: The properties (X.2)–(X.4) can be shown in the same way as for the metric ic . *Non-negativity* is evident. *Normalisation* is satisfied, since the values of the

metric lic are bounded above by the total number of local service interfaces of all service groups.

Example: For the service groups g_1 , g_2 , g_3 , and g_4 from Fig. 2, the metric lic has following values: $lic(g_1) = 0$, $lic(g_2) = 0$, $lic(g_3) = 2$ and $lic(g_4) = 5$.

3) *Service Group Interface Exposure Degree (ied)*: The metric ied reflects the fraction of service interfaces that are *not local* in relation to all provided service interfaces. The metric is defined by the function $ied: \text{GROUP} \rightarrow [0, 1]$,

$$ied(g) = \begin{cases} \sum_{s \in g} |s^\bullet| / \sum_{s \in g} |s^\bullet|, & \text{if } \exists s \in g: s^\bullet \neq \emptyset \\ 0, & \text{otherwise.} \end{cases} \quad (3a)$$

$$= \begin{cases} \frac{ic(g) - lic(g)}{ic(g)} = 1 - \frac{lic(g)}{ic(g)}, & \text{if } ic(g) > 0 \\ 0, & \text{otherwise.} \end{cases} \quad (3b)$$

As depicted in (3b), the metric ied can be computed using the metrics ic and lic . The metric ied can be used to measure both coupling and cohesion of a service group.

Metric Properties: *Non-negativity* and *normalisation* properties are satisfied, since by definition $ied(g) \in [0, 1]$ for all groups. The exposure degree of a service group g is null when the services in g do not provide service interfaces or all provided service interfaces in g are local service interfaces. Thus, the *null value* property also holds. The *monotonicity* property is satisfied, since the exposure degree of a group can only increase by adding a new provided service interface to the group. Concerning the union of service groups, it is sufficient to verify the *merging of service groups* property (X.4 and X.4), since the metric ied describes both coupling and cohesion quality attributes. The *merging of service groups* property states that the exposure degree of a group g obtained by merging the groups g_1 and g_2 is not greater than the sum of exposure degrees of the two original groups g_1 and g_2 . The property is evident for $ied(g_1) = 0$ or/and $ied(g_2) = 0$. We prove the *merging of service groups* property for the case $ied(g_1) > 0$ and $ied(g_2) > 0$ by applying this property to the metrics ic and lic in (3b)

$$\begin{aligned} ied(g) &= ied(g_1 \cup g_2) \\ &\stackrel{(3b)}{=} \frac{ic(g_1 \cup g_2) - lic(g_1 \cup g_2)}{ic(g_1 \cup g_2)} \\ &= \frac{ic(g_1) - lic(g_1)}{ic(g_1) + ic(g_2)} + \frac{ic(g_2) - lic(g_2)}{ic(g_1) + ic(g_2)} \\ &\leq \frac{ic(g_1) - lic(g_1)}{ic(g_1)} + \frac{ic(g_2) - lic(g_2)}{ic(g_2)} \\ &= ied(g_1) + ied(g_2) \end{aligned} \quad (4)$$

Example: In the following, we compute exemplary the exposure degree of service group g_1 from Fig. 2:

$$ied(g_1) = \frac{ic(g_1) - lic(g_1)}{ic(g_1)} = \frac{1 - 0}{1} = 1$$

4) *Service Group Exposure Count (ec)*: This metric counts for a given service group g the number of other service groups that contain services with “requires”-dependencies to the service interfaces of the service group

g. The metric is defined by the function $ec: \text{GROUP} \rightarrow \mathbb{N}_0$,

$$ec(g) = |\{g' \in \text{GROUP} \setminus \{g\} : \exists s' \in g', s \in g \text{ such that } s' \text{ requires } i \in s^\bullet\}|. \quad (5)$$

The metric ec belongs to the class of coupling metrics.

Metric Properties: Properties (X.1) and (X.2) are evident. *Monotonicity* is satisfied, since the value of the metric ec for the group g increases when adding a service group that has services with “requires”-relationship to service interfaces of g . When merging two service groups g_1 and g_2 to a new group g , the metric $ec(g)$ will not be greater than the sum of $ec(g_1)$ and $ec(g_2)$. The metric $ec(g)$ can only get smaller, if there are service groups with services that have “requires”-relationships to service interfaces of both original service groups g_1 and g_2 . Otherwise, the equality will hold. Thus, properties (X.4) and (X.5) are also satisfied.

Example: Using the example from Fig. 2, we compute the exposure count for the service groups g_1 , g_2 , g_3 , and g_4 : $ec(g_1) = 1$, $ec(g_2) = 1$, $ec(g_3) = 3$, $ec(g_4) = 0$.

5) **Service Group Required Interfaces Count (ric):** The metric ric denotes for a given service group g the number of required service interfaces that are provided by services located in other service groups. It is defined by the function $ric: \text{GROUP} \rightarrow \mathbb{N}_0$,

$$ric(g) = |\{i \in \text{INTERFACE} : \exists s \in g, g' \in \text{GROUP} \setminus \{g\}, s' \in g' \text{ such that } i \in s'^\bullet \wedge s \text{ requires } i\}|. \quad (6)$$

The metric ric measures the coupling quality attribute.

Metric Properties: Properties (X.1) and (X.2) are obvious. By adding a new “requires”-relationship from a service in the service group g to a service interface provided by a service located in another service group, the total number of required service interfaces of g will also increase by one. Thus, *monotonicity* is satisfied. The properties (X.4) and (X.5) are also satisfied, since the metric ric of the service group g obtained by merging the service groups g_1 and g_2 is equal to the sum $ric(g_1) + ric(g_2)$.

Example: For the four service groups g_1 , g_2 , g_3 , and g_4 from Fig. 2 we obtain following values for the metric ric : $ric(g_1) = 2$, $ric(g_2) = 2$, $ric(g_3) = 0$, $ric(g_4) = 2$.

6) **Service Group Required Groups Count (rgc):** The metric rgc counts for a given service group g the number of service groups $g' \in \text{GROUP} \setminus \{g\}$ that contain service interfaces required by the service group g . The metric is defined by the function $rgc: \text{GROUP} \rightarrow \mathbb{N}_0$,

$$rgc(g) = |\{g' \in (\text{GROUP} \setminus \{g\}) : \exists s \in g, s' \in g' \text{ such that } s \text{ requires } i \in \text{INTERFACE} \wedge i \in s'^\bullet\}|. \quad (7)$$

This metric belongs to the class of coupling metrics.

Metric Properties: Properties (X.1) and (X.2) are obvious. Properties (X.3)–(X.5) can be shown in a similar way as for the metric ric .

Example: We use again the service groups illustrated in Fig. 2 to compute the metric rgc : $rgc(g_1) = 1$, $rgc(g_2) = 2$, $rgc(g_3) = 1$, $rgc(g_4) = 1$.

7) **Service Groups Dependency Intensity (di):** The metric di measures the strength of dependency interconnections between two service groups. The metric is defined by the function $di: \text{GROUP} \times \text{GROUP} \rightarrow \mathbb{R}_0^+$,

$$di(g_1, g_2) = \begin{cases} \frac{s_1}{|g_1|} \cdot ric(g_1)_{|g_2} + \frac{s_2}{|g_2|} \cdot ric(g_2)_{|g_1}, & \text{if } g_1 \neq g_2 \\ 0, & \text{otherwise} \end{cases} \quad (8)$$

where

$$s_1 = |\{s \in g_1 : \exists s' \in g_2 \wedge i \in s'^\bullet \text{ such that } s \text{ requires } i\}|, \\ s_2 = |\{s \in g_2 : \exists s' \in g_1 \wedge i \in s'^\bullet \text{ such that } s \text{ requires } i\}|,$$

and $ric(g)_{|g'}$ denote the number of required service interfaces in g that are provided by services from g' .

The metric di is a relative measure for the dependency intensity between two service groups. For the interpretation of this measure, we need to derive a threshold value that will provide the basis for determining whether we have high or low dependency intensity between two service groups. The metric belongs to the class of coupling metrics.

Metric Properties: Properties (X.1)–(X.2) follow immediately from (8). We assume w.l.o.g. that a new dependency in form of a required service interface from a service in g_1 to a service in g_2 is added. In (8), the factor s_1 will either not change or increase, and $ric(g_1)_{|g_2}$ will increase because of property (X.3) shown for the metric ric . Thus, *monotonicity* is satisfied. Properties (X.4) and (X.5) are not satisfied, i.e. the metric is not linear and the metric value of the group g obtained by merging two service groups g_1 and g_2 can be higher than the sum of the metric values of the two original service groups g_1 and g_2 . Still, we consider the metric well-grounded, since the expressiveness of the metric is much higher when we take into consideration the relation between the number of services that have “requires”-relationships to the other group and the size of the group.

Example: As an example, we compute the dependency intensity between service groups g_1 and g_3 , as well as g_2 and g_3 from Fig. 2: $di(g_1, g_3) = \frac{1}{2} \cdot 1 + \frac{0}{4} \cdot 0 = 0.5$, $di(g_2, g_3) = \frac{3}{4} \cdot 1 + \frac{0}{4} \cdot 0 = 0.75$.

8) **Service Group Reallocation Capacity (rc):** The metric rc describes the ease of reallocating a service group e.g. from one ECU to another ECU. The metric is defined by the function $rc: \text{GROUP} \rightarrow \mathbb{R}_+$,

$$rc(g) = \begin{cases} \frac{1}{|\text{GROUP}|-1} \cdot \sum_{\substack{g' \in \text{GROUP} \\ g' \neq g}} di(g, g'), & \text{if } |\text{GROUP}| > 1 \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

The metric rc belongs to the class of coupling metrics.

Metric Properties: Properties (X.1) and (X.2) follow immediately from (9). Property (X.3) follows from the respective property demonstrated for the metric di .

Example: As an example, we compute the reallocation capacity of the service group g_1 from Fig. 2. First, we compute the dependency intensities of the group g_1 to all other groups: $di(g_1, g_2) = 0.75$, $di(g_1, g_3) = 0.5$, $di(g_1, g_4) = 0$. The reallocation capacity of the group g_1 can be computed as follows: $rc(g_1) = \frac{1}{4-1} \cdot (0.75 + 0.5 + 0) = 0.42$.

B. Metric Interpretation

In this section, we consider some combinations of metrics that give us useful indications about the quality of service groups. For the sake of simplicity, we limit the range of examined values for metrics to high (▲) and low (▼). Table II contains a summary of relevant combinations of discussed metrics, their values and possible interpretations.

1) *Metrics ic, lic, ied, and ec*: High values of the metrics *ic* and *ied* for a service group *g* are an indicator of strong coupling of this group to other service groups. If the value of the metric *ec* is also high, then the group is characterised through high reusability of services. If, additionally, the metric *lic* displays a high value, then the services in the group are highly cohesive. Thus, the service group, in this case, encapsulates a well-defined small set of highly reusable services. If the value of the metric *lic* is small, then the services in the group are not cohesive. This can point to a problematic group (e.g. provision of disparate services or too coarse-grained groups).

2) *Metrics ric and rgc*: The examination of the two metrics *ric* and *rgc* in conjunction gives us a first good indicator of the dependency degree of services in a service group to other service groups. A low value of *ric* for the service group *g* is an indication of loose coupling between the services in *g* and the services in other service groups. Inversely, a high value of *ric* points out a strong dependency of services in *g* to services in other service groups. The metric *rgc* describes the type of service group dependency. A high value of the metric *rgc* points to dependencies to different service groups, while a low value of *rgc* indicated dependencies to a restricted set of service groups. The combination of high value for *ric* and low value for *rgc* can be interpreted as an indication to investigate if the respective service groups could be merged or should be deployed on the same control unit.

A further aspect to be considered in the context of an examination of the metrics *ric* and *rgc* is the *data volume* exchanged between service groups. The exchanged data volume may not be of great relevance when considering single services. However, for service groups that are interpreted as abstractions of ECUs, SWCs, processes or process threads, the exchanged data is of great importance, since it impacts directly e.g. the bus load which is an important factor in time- and safety-critical embedded systems.

We define the exchanged data volume between two service groups as the number of exchanged data frames per time unit multiplied by the data frame size. High data volume exchanged between service groups in conjunction with high *ric* and low *rgc* is a clear sign for the necessity of further investigations of respective service groups.

3) *Metric di*: The metric *di* measures the strength of the dependency interconnections between two service groups. This metric takes into consideration besides the existing dependencies between two service groups, also the size of these groups. This allows better comparability between the values of the metric *di* for different service groups. High values of the metric *di* indicate strong dependency

TABLE II: Metric Interpretation

Metrics				Interpretation
ic	lic	ied	ec	
▲	▼	▲	▲	Strong coupling to other service groups, high reusability of services of the service group, low cohesion of services of the service group;
▲	▲	▲	▲	Strong coupling to other service groups, high reusability of services of the service group, high cohesion of services of the service group;
▲	▼	▲	▼	Strong coupling to a small set of service groups, low cohesion of services in the service group;
▲	▲	▲	▼	Strong coupling to a small set of service groups, high cohesion of services in the service group;
▲	▲	▼	▼	Weak coupling to other service groups, high cohesion of services in the service group;
Other metrics				No clear conclusion about the service group, further analysis necessary;
ric				rgc
▲	▲			high dependency to a large set of service groups
▲	▼			high dependency to a restricted set of service groups
▼	▲			weak dependency to other service groups
▼	▼			weak dependency to a restricted set of service groups

▲denotes high values, ▼denotes low values

relationship between two service groups, and low values show weak dependency between two service groups.

4) *Metric rc*: The metric *rc* calculates a mean value for the dependency strength of a service group with respect to all other groups. Low values of the metric can be interpreted as better possibilities for service group reallocation (e.g. from one ECU to another ECU or from one SWC to another SWC) because of weaker dependencies to other service groups. High values of the metric *rc* indicate on average stronger dependency relationship to other service groups. That makes the reallocation of service groups more complicated.

V. CASE EXAMPLE









In this section, we introduce the service architecture of the ACC example and report on the results of the conducted evaluation.

A. Preparatory Steps: Data Extraction and Modelling

We have intensively studied the AUTOSAR Classic Platform standard documents (version 4.4.0) and extracted from the stated examples a logical model for the Cruise Control and Adaptive Cruise Control (CrsCtrlAndAcc) function. Its core functionality is located in the chassis domain. We analysed the interconnections of this function to other vehicle domains and extended our logical model to cover the whole data flow from sensors to actuators.

Next, we decomposed the function into incorporating services. The services and their architecture are based on data-flow between software component decompositions of the presented example. Table III lists the precise source and reference from the AUTOSAR specification that we used. Besides the extracted services, we extended our logical model by three off-board services running in the OEM backend as one service group *g₃*: (1) TrafficSrv, (2) WeatherSrv, and (3) CrsCtrlAndAccDiagnostics (function diagnosis).

TABLE III: Sources of services

Services and Source
 AUTOSAR_EXP_AIChassis, \$4.1: Camera, Lidar, AccSnsrDataFusion, AccObjTarSeln, CrsCtrlFree, FolwCtrlAndAArbn, CrsCtrlAndAccStCtrl
 AUTOSAR_EXP_AIChassis, \$4.2: Esc
 AUTOSAR_EXP_AIChassis, \$4.3: Ssm
 AUTOSAR_EXP_AIChassis, \$4.4: Epb
 AUTOSAR_EXP_AIChassis, \$4.5: Vlc
 AUTOSAR_EXP_AIPowertrain, \$4: PTTorqueCrd, PTSpeedCrd, PTOpMode, EngTorqueModeMngm, AirSystem, FuelSystem, IgnitionSystem
 AUTOSAR_EXP_AIHMMIMultimediaAndTelematics, \$5: CrsCtrlAndAccController, CrsCtrlAndAccStatusDisplay, CrsCtrlAndAccActivationPanel
 AUTOSAR_EXP_AIBodyAndComfort, \$4.6: BrkLiMgr, ExtrLiAdprFrnt, ExtrLiAdprRe, ExtrLiActrFrnt, ExtrLiActrRe

Next, we discussed in a 2-hours workshop the gained model with experts from an international premium-class car manufacturer. First, the derived service-oriented architecture was confirmed as realistic and plausible. Secondly, we identified two possible architecture options to combine services to software components, as depicted in Fig. 3. Most of the services can be assigned in an unambiguous way to one of the defined service groups g_1 to g_6 , based on the vehicle domains. The two services AccObjTarSeln and CrsCtrl&AccStCtrl can be assigned to different service groups. In one scenario, illustrated in Fig. 3(a), we mapped them to the same service group g_2 . In the second scenario, depicted in Fig. 3(b), we placed them in different service groups, g_1 resp. g_4 .

B. Results

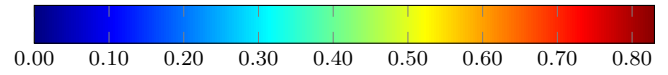
We applied the set of metrics introduced in Section IV to evaluate the quality of service groups that describe two possible composition variants of services into software components, as illustrated in Fig. 3. The software components are abstracted by the service groups g_1 to g_6 . The computation results of metrics are depicted in Tabs. IV and V. Table V illustrates the dependency intensity between service groups. The calculated values of the metric indicate better coupling and cohesion properties of the service groups in the first architectural variant. We observe that in the second architectural variant, only the dependency intensity between the groups g_2 and g_3 is better than in the first architectural variant. This is due to the fact that the service CrsCtrl&AccStCtrl requires three service interfaces provided by services in the service group g_3 . By moving the service CrsCtrl&AccStCtrl from service group g_2 to g_4 in variant 2, we improve considerably the coupling between service groups g_2 and g_3 (Variant 1: $di(g_2, g_3) = 0.60$, Variant 2: $di(g_2, g_3) = 0.0$). The dependency intensity values between the service group g_2 and all other groups except g_3 are lower

TABLE IV: Metrics for each group of the two variants.

	Variant 1						Variant 2					
Metric	g_1	g_2	g_3	g_4	g_5	g_6	g_1	g_2	g_3	g_4	g_5	g_6
ic	3	7	3	3	10	7	4	4	3	5	10	7
lic	2	6	0	2	9	6	3	2	0	4	9	6
ied	0.33	0.14	1	0.33	0.1	0.14	0.25	0.5	1	0.2	0.1	0.14
ec	1	1	1	1	1	1	1	1	1	1	1	1
ric	0	7	0	1	0	0	0	4	0	5	0	0
rgc	0	5	0	1	0	0	0	4	0	2	0	0
rc	0.04	0.346	0.12	0.106	0.04	0.04	0.066	0.364	0.15	0.316	0.066	0.066

TABLE V: Dependency Intensity Metric

	Variant 1						Variant 2					
Gr.	g_1	g_2	g_3	g_4	g_5	g_6	g_1	g_2	g_3	g_4	g_5	g_6
g_1	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.33	0.00	0.00	0.00	0.00
g_2	0.20	0.00	0.60	0.53	0.20	0.20	0.33	0.00	0.00	0.83	0.33	0.33
g_3	0.00	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.75	0.00	0.00
g_4	0.00	0.53	0.00	0.00	0.00	0.00	0.00	0.83	0.75	0.00	0.00	0.00
g_5	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.33	0.00	0.00	0.00	0.00
g_6	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.33	0.00	0.00	0.00	0.00



in the first variant, which demonstrates weaker dependency between service groups (e.g. Variant 1: $di(g_2, g_1) = 0.20$, Variant 2: $di(g_2, g_1) = 0.33$).

Next important metric for the measurement of coupling and cohesion attributes of service groups is the reallocation capacity. The calculated values for the reallocation capacity from Tab. IV clearly shows that the first architectural variant is preferable to the second. E.g. the reallocation capacity for the service group g_1 has the value of 0.04 in the first variant, and 0.066 in the second architectural variant, which demonstrates better reallocation possibilities for g_1 in the first variant.

C. Discussion

Although both architectural variants illustrated in Fig. 3 are realistic and plausible, the application of service group metrics introduced in Section IV shows a clear preference for the *Variant 1*. The service groups contained in *Variant 1* are characterised by weaker dependency on each other and support easier reallocation of services in the case of ECU failures or outages. This outcome is not apparent at first glance and could be obtained, even for a small number of services, by applying metrics. Thus, the employment of well-defined and adequate service group metrics contributes substantially to the identification of optimal service compositions in a service-oriented architecture in order to improve the quality attributes of the software system. The application of service group metrics, especially in the early software design phases, helps gain valuable insights into the quality of different service groupings and lays a good foundation for the development of loosely coupled and highly cohesive service-oriented architectures.

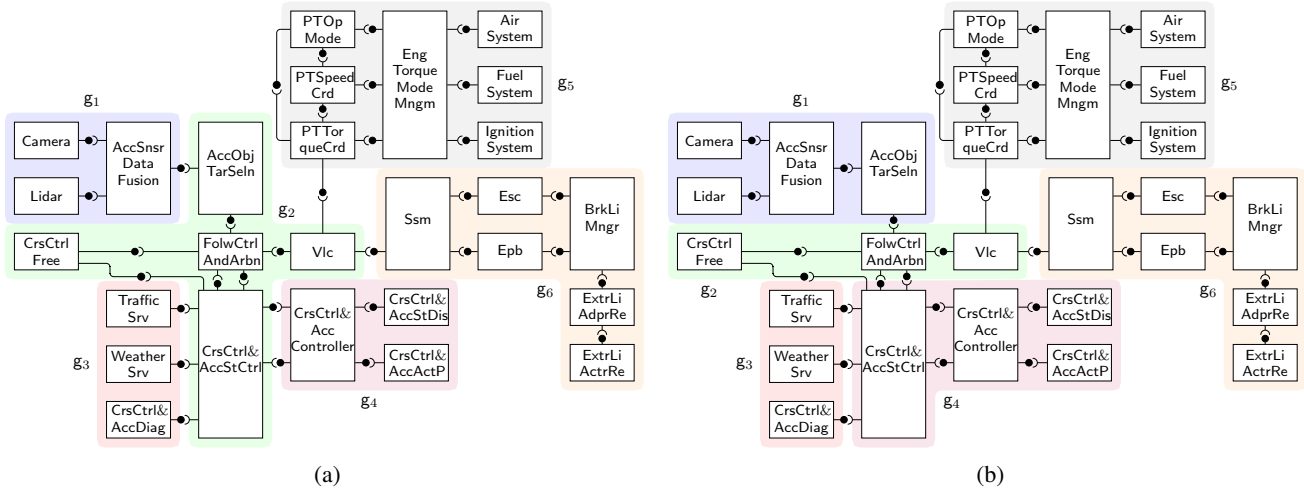


Fig. 3: Service Architecture for ACC Case Example. (a) Variant 1; (b) Variant 2.

VI. CONCLUSION

In this paper, we have motivated first the creation, analysis and evaluation of service groups in a distributed automotive service-oriented architecture. We then focused on the evaluation of service groups by proposing a set of metrics to measure the coupling and cohesion quality attributes of service groups. We showed that most proposed measures satisfy the set of properties defined in [15] and adapted to the context of service-oriented architectures. Although the measures in Sections IV-A.7 and IV-A.8 do not satisfy all properties defined in [15], we demonstrated their usefulness on a realistic case example from the automotive domain. We plan to research if all stated properties are strictly necessary for *rigorous* and *well-founded* SOA measures. We consider that the proposed metrics-based approach for the evaluation of service groups is especially applicable in the early design phases of the software development process. We coined the term “*playground*” to reflect the idea of early experimentation with different service groupings. By defining the abstract meaning of a service group (*virtual ECU*, *SWC*, or *process*), service groupings reflect different architectural design candidates, which can be evaluated with respect to different quality attributes. This paper is a first step in the direction of establishing a comprehensive set of metrics for the evaluation of architectural design candidates in distributed safety-critical service-oriented architectures. In our future work, we plan to extend the proposed metrics by further metrics to measure other quality attributes such as complexity. In addition, we plan to extend the proposed metrics to include in the measurement specific parameters (e. g. volume of exchanged data between service groups) that are important in designing and partitioning automotive E/E software systems.

REFERENCES

- [1] 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.
- [2] Society of Automotive Engineers, *Taxonomy and Definitions for Terms Related to On-road Motor Vehicle Automated Driving Systems*, SAE Standard J3016, 2014.
- [3] V. Cebotari and S. Kugele, “On the nature of automotive service architectures,” in *IEEE International Conference on Software Architecture Companion, ICSA Companion 2019*. IEEE, 2019, pp. 53–60.
- [4] E. W. Dijkstra, “Structured programming,” O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, Eds. London, UK, UK: Academic Press Ltd., 1972, ch. Chapter I: Notes on Structured Programming, pp. 1–82.
- [5] D. L. Parnas, “On the criteria to be used in decomposing systems into modules,” *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.
- [6] “AUTOSAR Adaptive Platform: Specification of Manifest,” <https://www.autosar.org/standards/adaptive-platform/adaptive-platform-1903/>.
- [7] M. Hirzalla, J. Cleland-Huang, and A. Arsanjani, “A metrics suite for evaluating flexibility and complexity in service oriented architectures,” in *Service-Oriented Computing - ICSOC 2008 Workshops*, ser. Lecture Notes in Computer Science, vol. 5472. Springer, 2008, pp. 41–52.
- [8] Y. Liu and I. Traore, “Complexity measures for secure service-oriented software architectures,” in *Third International Workshop on Predictor Models in Software Engineering (PROMISE’07: ICSE Workshops 2007)*. IEEE, 2007.
- [9] K. Qian, J. Liu, and F. Tsui, “Decoupling metrics for services composition,” in *5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSA’06)*. IEEE, 2006.
- [10] M. Pereplechikov, C. Ryan, K. Frampton, and Z. Tari, “Coupling metrics for predicting maintainability in service-oriented designs,” in *18th Australian Software Engineering Conference (ASWEC 2007)*. IEEE Computer Society, 2007, pp. 329–340.
- [11] R. Sindhgatta, B. Sengupta, and K. Ponnalagu, “Measuring the quality of service oriented design,” in *7th International Joint Conference, ICSOC-ServiceWave 2009*. Springer, Berlin, Heidelberg, 2009.
- [12] M. Pereplechikov, C. Ryan, and K. Frampton, “Cohesion metrics for predicting maintainability of service-oriented software,” in *Seventh International Conference on Quality Software (QSIC 2007)*. IEEE Computer Society, 2007, pp. 328–335.
- [13] X. Wang, “Metrics for evaluating coupling and service granularity in service oriented architecture,” in *2009 International Conference on Information Engineering and Computer Science*, Dec 2009, pp. 1–4.
- [14] S. W. Choi and S. D. Kim, “A quality model for evaluating reusability of services in soa,” in *10th IEEE Conference on E-Commerce Technology (CEC 2008) and 5th IEEE Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2008)*. IEEE, 2008.
- [15] L. C. Briand, S. Morasca, and V. R. Basili, “Property-based software engineering measurement,” *IEEE Trans. Software Eng.*, vol. 22, no. 1, pp. 68–86, 1996.
- [16] “AUTOSAR Adaptive Platform: Explanation of Adaptive Platform Design,” <https://www.autosar.org/standards/adaptive-platform/adaptive-platform-1903/>.