# Model-Based Resource Analysis and Synthesis of Service-Oriented Automotive Software Architectures

**Conference Paper** · September 2019

DOI: 10.1109/MODELS.2019.000-8

**3 authors:**

Philipp Obergfell
Karlsruhe Institute of Technology
**17** PUBLICATIONS **59** CITATIONS
SEE PROFILE

Stefan Kugele
Technische Hochschule Ingolstadt
**64** PUBLICATIONS **301** CITATIONS
SEE PROFILE

Eric Sax
Karlsruhe Institute of Technology
**198** PUBLICATIONS **663** CITATIONS
SEE PROFILE

# Model-Based Resource Analysis and Synthesis of Service-Oriented Automotive Software Architectures

Philipp Obergfell*
*BMW Group Research, Technologies, and Innovation*
Munich, Germany
philipp.obergfell@bmw.de

Stefan Kugele*
*Technical University of Munich*
Munich, Germany
stefan.kugele@tum.de

Eric Sax
*Karlsruhe Institute of Technology*
Karlsruhe, Germany
eric.sax@kit.edu

***Abstract*—*Context*: Automotive software architectures describe distributed functionality through an interplay of software components. One drawback of today's architectures is their strong integration into the onboard communication network based on predefined dependencies at design-time. To foster independence, the idea of service-oriented architecture (SOA) provides a suitable prospect as network communication is established dynamically at run-time. *Aim*: We target to provide a model-based design methodology for analysing and synthesising hardware resources of automotive service-oriented architectures. *Approach*: For the approach, we apply the concepts of design space exploration and simulation to analyse and synthesise deployment configurations at an early stage of development. *Result*: We present an architecture candidate for an example function from the domain of automated driving. Based on corresponding simulation results, we gained insights about the feasibility to implement this candidate within our currently considered next E/E architecture generation. *Conclusion*: The introduction of service-oriented architectures strictly requires early run-time assessments. In order to get there, the usage of models and model transformations depict reasonable ways by additionally accounting quality and development speed.**

***Index Terms*—Service-oriented architecture, real-time behaviour, model-based design, automotive architectures**

## I. Introduction

During the last decades, thousands of mostly software-controlled functions were included in modern cars, which are executed on a large number of electronic control units (ECU). Their particular characteristics reach from non-safety-critical to safety-critical and real-time-critical functions. Driving forces for this development were: (i) safety requirements, (ii) customer demands for more comfort and the newest infotainment systems, and (iii) advanced driver assistance systems allowing to reach higher levels of driving automation (cf. [1]).

*Status Quo:* These driving forces led to the current electric/electronic (E/E) architectures that are best characterised as historically grown, mostly federated, partly integrated architectures with often pragmatic, cost-efficient, and ad-hoc solutions. More than 100 purpose-built ECUs realise in an interplay of timed signals sent via heterogeneous bus systems (e. g. CAN, LIN, FlexRay, or Ethernet) with gateway structures the functions' behaviours. Current automotive software architectures mainly follow the idea of static communication paths in line with the AUTOSAR Classic Platform [2]. Here, distributed software applications are strictly linked to the exchange of messages between ECUs at design-time which in turn harms the separation of developing both parts independently. This ECU or message-centric focus is hardly capable of coping with future challenges in automotive software and systems engineering. To foster a quicker roll-out of innovative functions and services, more *agile* development methodologies need to be installed into established processes.

*The first two authors have contributed equally and are co-first authors.

TABLE I
FUTURE METHODOLOGIES AND ADDRESSED CHALLENGES

| **Aspect** | **Current** | **Future** | **Challenge** |
|---|---|---|---|
| *Methodology and process* | | | |
| ↪ Process model | heavyweight | agile | **C1, C2, C3** |
| ↪ Modelling artefact | signal, PDU, message | service | **C2, C3** |
| ↪ Development | ECU | service | **C2, C3** |
| *Architecture* | | | |
| ↪ Paradigm | federated | centralised, zonal | **C1, C2** |
| ↪ Deployment | static | dynamic | **C2** |
| *Communication* | | | |
| ↪ Paths | static | static, dynamic | **C2, C3** |
| ↪ Technology | heterogeneous | homogeneous | **C1, C2, C3** |
| *Deployment* | | | |
| ↪ Delivery | once | continuous | **C1, C2, C3** |
| ↪ Update | — | over-the-air | **C1, C2, C3** |

A lot of research effort is currently put in developing all-new E/E architectures that will be even better equipped for future trends, innovations, and new technologies [3], [4].

We see three fundamental changes in the automotive industry requiring new approaches for development at *design-time* as well as for operation during *run-time*:
**C1 Automation**: Driving at higher levels of driving automation (i. e., levels 3-5 according to SAE J3016 [1]) require a particular focus on *functional safety* and *security*. Liabilities switch from the driver to the car vendors in the case of automated driving beginning with level 3. At conditional automation such as highway pilots, the car fully controls in particular operational situation the vehicle (longitudinal and lateral movement) without the need to be monitored by the human driver. Regulatory authorities will by law require to improve those safety-critical functions, i. e., fix possible errors and update crucial components to the state-of-the-art. Necessary machine learning models are trained in the backend of OEMs and need to be *delivered continuously*.

**C2 Intelligence**: Machine learning is an emerging and cross-disciplinary area, which is advancing with great strides also for in-vehicle functions. Sophisticated *smart capabilities* and *highly personalised* functions are no longer a future vision but can already be found today. Machine learning requires a lot of data to be useful. E/E architectures need to be able to transmit, store, and process this data which poses requirements on bandwidth, computing power, and communication topology.
**C3 Connectivity**: Experts are sure that reaching level 5 of driving automation can only be achieved through car-to-x (car, infrastructure, backend) communication. Vehicles are in use for more than 15 years on average. During that time-span, information technology improves rapidly: E. g. an encryption algorithm used for backend or car-to-car communication can become outdated and not be considered secure anymore and has to be updated after almost a decade. Hence, possible attack surfaces need to be mitigated continuously.

Table I summarises essential aspects that have already partially changed today, but most will change—that is our firm belief—in the future.

*Future Perspective:* To cope with these challenges, besides a scalable and performant computing platform, *software architecture* plays a critical role. When designing a software architecture of a system, the primary goals are to develop flexible, adaptive, and maintainable systems. *Service-oriented architectures* (SOA) are known for supporting the design of flexible systems. In contrast to existing CAN-based approaches combined with AUTOSAR Classic, service-oriented approaches allow for *late binding* at *run-time* facilitating the integration of new functionalities and services at run-time.

Automotive OEMs are on the way to steadily substitute legacy communication by including more and more IP-based communication technologies into their vehicles, which in turn support a better separation of software and hardware. At the very core of the presented approach is the principle of *service-orientation*. The combination of IP-based communication and SOA allows to *add*, *update*, *remove*, or *start* and *stop* services at run-time. Moreover, the workload for network engineers is reduced to a minimum, since they do not need to do all the way from signals down to messages manually, but only define the service provision and consumption relationship (cf. Fig. 1) together with their *interaction pattern* (cf. Fig. 2).

In Fig. 1a, engineers need to manually map signals into protocol data units (PDUs), which in turn are mapped to CAN messages. This is a time-consuming and error-prone process also hindering dynamicity and thus reconfigurations of the network topology at run-time. The resulting communication infrastructure is then *static* over the system's lifetime.

In Fig. 1, two applications $\mathsf{App}_1$ and $\mathsf{App}_2$ are logically interacting: in the first case by exchanging signals and in the second case by establishing a provision/usage connection. The latter mentioned usage relation only needs to be refined by defining their pattern of interaction. We distinguish between two principle communication patterns (cf. Fig. 2): (1) *Request/Response*, i. e., a *method* is called and a result is returned and (2) *Publish/Subscribe*, i. e., a server first publishes a service

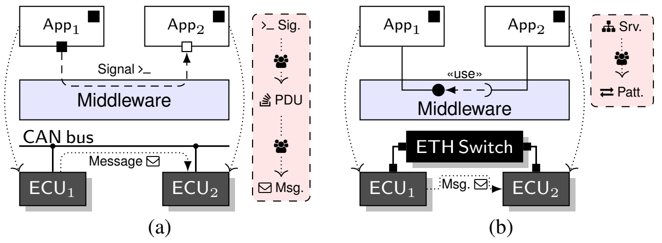


Fig. 1. (a) Classic signal to message mapping: Logical signals are grouped into PDUs (protocol data units), which in turn are mapped to CAN messages. (b) In contrast to service binding in a SOA approach. Here, only the interaction patterns are defined. The rest is done automatically.

and a client subscribes to it. Then, either periodically or at each change event at server-side, the client gets notified.

In a service-oriented architecture, *services* provided by a *server* can be looked-up in a *service registry* and used by *clients* (cf. Fig. 2). In a request/response-pattern, the client sends a request to the server, which in turn responds to the client. A `method`-call is realised in this way. In the publish/subscribe-pattern, the server first `publishes` a service which can be `found` by the client. The client then `subscribes` to the service and gets either *periodically* `notified` or by a change at server-side. Technically, this functionality is achieved using, for instance, the SOME/IP [5] or DDS [6] middleware in combination with the AUTOSAR Adaptive Platform standard for Ethernet-based topologies.

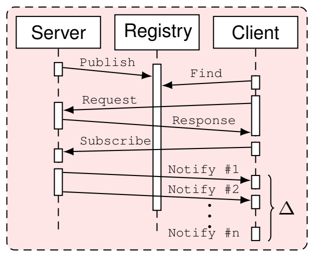


Fig. 2. Interaction patterns.

*Model-based Design:* SOA paradigms help to decouple software and hardware development to a large extent, facilitating fast and light-weight software updates in a fast-moving agile development process based on the idea of *continuous software engineering* [7]. Services will replace the former centricity on signals and ECUs in a strongly *model-driven* and *model-focused* approach shortly. We use models on the one hand early in the development process for verification, simulation, and analysis purposes, and, on the other hand as a basis for the generation of production code in a model-to-model transformation fashion. As a scope for this paper, we aim to solve two challenges associated with service-orientation by model-based techniques: (1) The deployment problem for a service-oriented architecture onto an overall future E/E architecture, and (2) the assessment of timing properties with emphasis on the idea of run-time reconfiguration.

*Research Questions:* For this work, we pose the following guiding research questions:

**RQ1** How does an optimised allocation of computing resources for service-based software applications from different automotive domains within a future E/E architecture look like?

**RQ2** How can timing properties of a service-oriented architecture that already allocated computing resources be assessed in a model-based manner?

*Contributions:* This paper provides the following contributions:

(i) Stepwise procedure for modelling and deploying a service-oriented architecture;
(ii) Implementation of the stepwise procedure within a model-based toolchain;
(iii) Evaluation of the procedure through an example from the domain of automated driving

*Outline:* The remainder of this paper is structured as follows. Section II summarises related work followed by the main approach in Section III. In Section IV, we present the conducted evaluation of our approach. Finally, we discuss the results and conclude in Sections V and VI.

## II. Related Work

Related to our work are the fields of model-based automotive architecture design and service-orientation for automotive system architectures.

### A. Model-based Automotive Architecture Design

Model-based architecture design in the automotive sector can be divided into two main parts: (1) Descriptive approaches, i. e., development and usage of architecture description languages (ADLs) and (2) architecture analysis approaches for early system verification and validation.

*1) Architecture Description:* The concept of architecture description is standardised by the ISO 42010 [8] and introduces the notion of architecture viewpoints in order to support stakeholders in respect of individual concerns. Over time, concrete instances of this idea have been introduced that all distinguish between functional and technical, i. e., implementation-related descriptions of a system architecture. Broy et al. [9] and Dajusren [10] developed approaches for architecture frameworks that facilitate to derive viewpoints with a focus on the distinction between static and dynamic aspects of an architecture. Recently, Pelliccione et al. [11] introduced an update on existing viewpoints that rather focuses on topics related to development collaboration like continuous integration and deployment as well as the idea of software ecosystems. For the more general domain of embedded systems, Pohl et al. [12] introduce the SPES_XT (Software Plattform Embedded Systems) methodology. In addition, several Architecture Descriptions Languages (ADLs) and reference architectures made their way into practice (cf. [13] and [14]). Here, especially the idea of separating different abstraction levels of an architecture is represented by the Electronics Architecture and Software Technology—Architecture Description Language (EAST-ADL [15] and the Electric Electronic Architecture—Analysis Design Language (EEA-ADL) [16] that rely on the MOF [17] approach.

*2) Model-based Architecture Analysis:* Besides the description of an architecture in corresponding languages, model-based approaches provide the application of various analysis techniques. Here, main objectives are to derive whether an architecture concept can meet requirements on real-time capabilities or functional correctness that are highly demanded for the development of safety-critical architectures. Possible approaches are given by model-based timing analysis, simulation techniques, as well as model checking.

For model-based timing analysis, the Real-Time Calculus [18] and the SymTA/S (Symbolic Timing Analysis for Systems) approach [19] are widely known and implemented in corresponding tools (cf. [20]). Recently, the idea of the logical execution time paradigm [21] made its way into the field of real-time assessment for automotive systems.

For simulation approaches, the C++ library SystemC [22], as well as the Functional Mock-up Interface (FMI) standard [23], provide capabilities for the test of system-level behaviour.

In addition, formal techniques like model checking are widely used in academia but also become more and more important for practitioners. The automotive functional safety standard ISO 26262 [24] *highly recommends* formal techniques at a certain degree of required integrity. AutoFOCUS [25] is a research prototype for model-driven development supporting formal verification capabilities.

### B. Automotive Service-Orientation

Kugele et al. [26] introduced the idea of $\alpha$SOA by formalising a framework for automotive services–oriented architectures that emphasises different service domains. Already before that, Broy and Krüger [27] as well as Malkis and Marmsoler [28] introduced formalisations of service-oriented architectures. Also Becker et al. [29] and Bocchi et al. [30] provide formalisations but with a focus on aspects such as service reconfiguration and orchestration. On the level of technologies, middleware concepts facilitate to indeed develop automotive service-oriented software architectures. Here, the scalable service-oriented middleware over IP (SOME/IP) and the Data Distribution Service (DDS) approach are present.

## III. Approach

Our approach targets on an early hardware resource analysis and synthesis of SOA-based automotive software architectures. The approach consists of the following parts: (1) The first part describes a meta-model used to formalise automotive services as the main building blocks of a service-oriented architecture. (2) The second part discusses an automated design space exploration (DSE) for the allocation of computing resources by services from different automotive domains. (3) Finally, the generation of source code artefacts for testing timing properties of an automotive service-oriented architecture is described. The application of our approach is outlined in Fig. 3. After modelling the service-oriented architecture and the network topology, SMT-based (SAT Modulo Theories) design space exploration is started. Next, candidate solutions are verified with regard to timing requirements considering network communication.

### A. Modelling Automotive Services

Automotive services provide functionality that is encapsulated by their service interfaces. Those are exposed to client

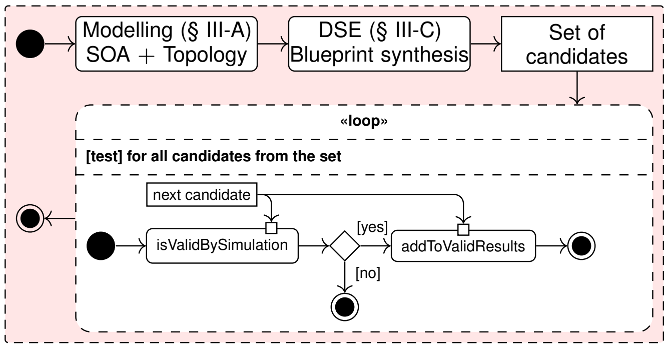


Fig. 3. Modelling, design space exploration, and timing analysis process.

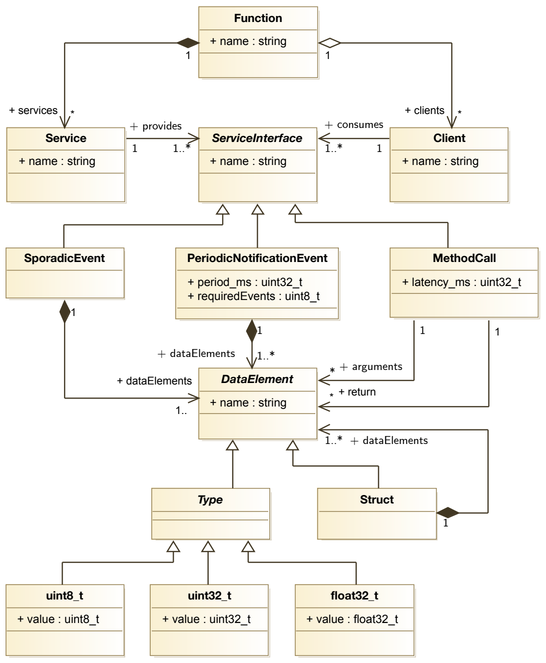


Fig. 4. Modelling automotive services.

applications for usage, i. e., service consumption. The service interface can be instantiated in different ways (cf. Fig. 4): (1) As sporadic event, (2) as periodic notification event, or as (3) method call.

*1) Timing Properties:* Service interfaces may feature also particular timing properties. For a periodic notification event, timing is considered by a time period in which the client needs to retrieve a minimum amount of events. For a *request-driven* behaviour, the timing property describes the maximally allowed latency between a request/response pair.

*2) Data Properties:* We introduce a generic structure consisting of nested structs to model data relevant for a service interface. For sporadic and periodic notification events, a nested struct describes the data that is provided to the client. For methods, two nested structs represent the input for a method and the output of a method, respectively.

In our running example, we consider the service interface of the vehicle's *environmental model* instantiated as periodic notification event. It is essential for every automated driving system to perceive and evaluate its environment. The gathered data originating from different sensor sources (see Section IV-A) are *fused* to form a comprehensive *environmental model*. In the example given, the environmental model returns a list of obstacles represented as 3D bounding boxes (blue cuboid in Fig. 6) positioned relative to the ego vehicle. An obstacle is thus described in the 3-dimensional space by the tuple $(x, y, z, w, l, h, \theta, \mathtt{t})$, where $(x, y, z)^\intercal$ gives the *location* of the box centroid, $(w, l, h)$ defines its *size*, $\theta$ defines its *yaw angle*, and $\mathtt{t}$ specifies the obstacle's type. Types can be cars, pedestrians, and cyclists: $\mathtt{t} \in T = \{\mathtt{c}, \mathtt{p}, \mathtt{y}\}$. Fig. 5b illustrates an example of a service interface that contains 8 data elements encapsulated in a `struct`. The timing property of the service interface describes that the client should receive at least n=3 events within a period of $\Delta = 30$ ms (cf. Fig. 2). The obstacle types are also encoded using unsigned integers since enumerations are not considered in SOME/IP as one dominant middleware approach in the field of SOA.[1]

## B. Automotive Framework Conditions

Unlike for instance consumer electronics (CE) software and systems, automotive E/E architectures including software pose particular requirements specific and unique in their combination for the automotive domain.

In the past, state-of-the-art semiconductor technologies known from CE devices found their way into the automotive domain around 7-10 years after their initial introduction in the CE area. Today, however, compute hungry computer vision and machine learning algorithms are fundamental for reaching higher levels of driving automation and require cutting-edge technologies both from software as well as from silicon side.

From that perspective, we can roughly name three categories of requirements: (i) *Function*, (ii) *Safety&Security*, and (iii) *Technology* in their interaction depicted in Fig. 5.

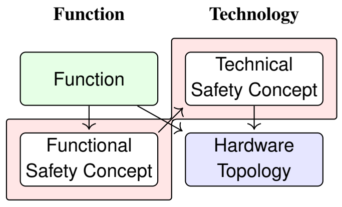
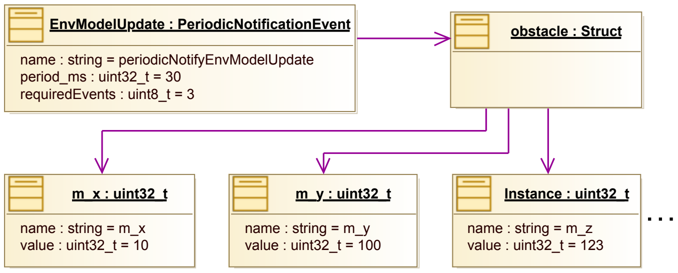


Fig. 5. Interaction among categories.

Right at the core is the *function* to be developed. After hazard analysis and risk assessment (HARA), safety goals are defined and a functional, as well as a technical safety concept, is derived. These analyses include the elicitation of safety requirements and the classification into automotive safety integrity levels (ASILs). Based on these assessments, technical measures such as diverse redundancy or safety patterns, in general, are taken. Moreover, the use of hypervisors (cf. [32]) allows to deploy services with different criticality levels onto the same ECU by supporting *time* and *space partitioning*. This also facilitates *security* properties, by restricting and monitoring information flow between partitions. Besides the safety- and security-related aspects, also functions pose particular requirements concerning hardware.

[1]See §4.1.4.6 of the SOME/IP protocol specification: https://www.autosar.org/fileadmin/user_upload/standards/foundation/1-0/AUTOSAR_PRS_SOMEIPProtocol.pdf

(a)



(b)

```
// Header file for struct
struct data_structure
{
    std::uint32_t m_x; std::uint32_t m_w; std::uint32_t m_theta;
    std::uint32_t m_y; std::uint32_t m_l; std::uint32_t m_type;
    std::uint32_t m_z; std::uint32_t m_h;
};
```

(c)

Fig. 6. (a) Picture #007007 taken from the KITTI benchmark [31]. (b) An excerpt of an object diagram of the environmental model is given respecting the characteristics of obstacles. (c) Definition of the data structure (`struct`) that models obstacles.

These requirements comprise computing requirements demanding for certain types of processors or application-specific integrated circuits (ASICs) in combination with their performance, e. g. frequency and memory range. A function that provides computer vision services, for instance, benefits from being deployed onto a graphics processing unit (GPU). Services relying on frameworks such as TensorFlow[2] benefit from being run on a tensor processing unit (TPU). For distributed functions, additionally, the network topology with specific bandwidth and protocol properties is relevant in order to meet latency constraints. In the future, a network topology such as those exemplified in Fig. 7 will be of importance. The main communication will be based on switched Ethernet in a *centralised computing platform* equipped with several ECUs containing several cores, GPUs, and TPUs. The idea behind this selective concentration of computational resources (cf. [33]) is the attempt to gradually transform from a highly federated and distributed architecture into a centralised one. Still today, ECUs and their included software functions belong to logical domains such as infotainment, powertrain, chassis, and comfort. In the course of centralisation, those—from their real-time, functional safety, or used technology—considerably different functions, will be migrated to the centralised platform.

Sensory data and relevant information such as high definition maps provided by other ECUs are processed within the centralised computing platform. Actuation is either performed again using actuators directly connected to the platform or via a gateway connected sub-networks. Please note that in this simplified topology, only relevant redundantly used sensors are included necessary for the environmental model. Of course, for level 4 driving also redundant actuators are present. The automated driving function will be running on the centralised computing platform. A trajectory is planned taking the environmental model into account. Necessary control actions such as longitudinal and lateral movement are then sent via the Ethernet switch and a gateway to the driving dynamics ECU which then triggers the actuators accordingly.

[2]https://www.tensorflow.org/

The mentioned redundancy is necessary since at level 4, the primary driving system is monitored by a system and fail-operational behaviour in the unlikely event of a system malfunction needs to be provided. Fail-operational means in comparison to fail-safe that the system continues to operate when one of its control systems fails. This means for our use case that the environmental model is deployed twice, one for the nominal behaviour and one for the fail-operational behaviour. Both operate on their individual set of sensors also depicted in the topology figure. Lastly, services and the providing software functions inherently run in parallel on different cores or in a distributed E/E architecture. Thus, this concurrent nature of automotive software needs to efficiently harness the full computing power available in multi-core/manycore processors respecting the theoretically limited speedup according to Amdahl's law [34]. The availability of high-performance cores qualified according to the highest safety level is still limited.

## C. Deployment Problem

The availability of multi-core SoCs provides greater scope for architecture implementation. The formerly predominant 1-to-1 function to ECU mapping does not hold anymore in times of dynamic updatable E/E architectures. The drawback of this freedom is an exploded design space that needs to be *explored* to find *"optimal"* architecture blueprints. Those blueprints represent solution candidates of the deployment problem, namely the mapping of functions providing or consuming services to processor cores.

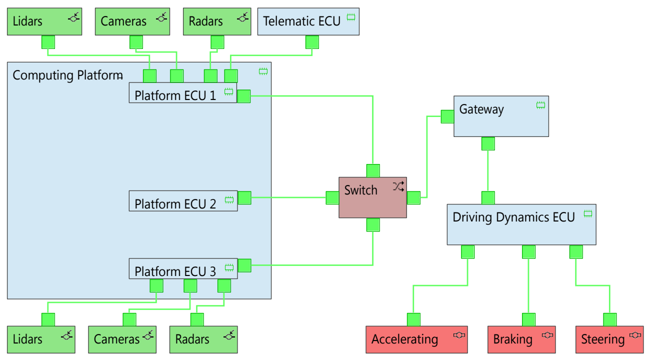


Fig. 7. Subset of the hardware topology used for the automated driving.

We apply *design space exploration* techniques to tackle this challenge (**RQ1**) in a similar way as the authors of [35], [36]. We do not only aim at deriving feasible, i. e., valid solutions, but also *optimal* solutions. Therefore, the notion of optimality needs to be clarified for this work.

It is in the nature of innovative and future-proof automotive architectures, that not a single optimisation goal describes best the intent of its engineers; However, several objectives need to be optimised (minimises or maximised) simultaneously yielding a *multi-objective optimisation problem*. In such a setting, there is not "the" best solution but a set of *Pareto-optimal* solutions describing compromises between possibly contradicting goals.

Besides optimisation objectives also *constraints* need to be considered. Necessary constraints prevent solutions from overloading processors, exceeding memory capacities, or invalidating safety requirements. The multi-objective optimisation problem is mathematically given as follows:

$$\begin{aligned} \text{minimise } & f_1(\boldsymbol{x}), \dots, f_m(\boldsymbol{x}), \boldsymbol{x} \in \mathcal{X} \\ \text{subject to } & g_i(\boldsymbol{x}) \le 0, i = 1, \dots, m \\ & h_j(\boldsymbol{x}) = 0, j = 1, \dots, p \end{aligned}$$

where $f_i$, $1 \le i \le m$ $(m > 1)$, are the *objectives* with $f_i \colon \mathcal{X} \to \mathbb{R}$ and $\mathcal{X}$ is the decision space containing the decision variables: $\boldsymbol{x} = (x_1, x_2, \dots, x_k)^\intercal$ encoding the DSE problem. Moreover, $g_i$ are *inequality* and $h_j$ are *equality* constraints. Note that $m$ and $p$ can be equal 0.

*1) Optimality:* For the case example described in Section IV we consider the two optimisation criteria:

$f_1$ *minimise* the number of used cores,

$f_2$ *minimise* the use of cores that are qualified according to high ASILs for applications that do not require the highest levels.

For the second optimisation function, we detect when a software function is deployed on an overqualified processor core and capture the violation by a penalty value: For example, a software function having a specified safety requirement of ASIL B is deployed on a processor core qualified up to ASIL D would yield a penalty value of 2. The maximal penalty is given by a non-safety-critical software function (having no ASIL requirement) which is deployed on an ASIL D processor core. This would yield a penalty value of 4.

*2) Constraints:* Performance and resource indicators derived from the design space exploration allow for early verification of hardware capabilities such as performance, memory consumption, and bandwidth requirements. The presented work considers (i) processor utilisation, (ii) volatile memory (RAM), (iii) flash memory, and (iv) ASIL compatibility for the DSE and deployment *synthesis*.

Finally, we retrieve a set of synthesised deployment candidates that form a good working basis for engineers in their daily architectural work.

## D. Network Deployment and Test of Timing Properties

Each of the synthesised deployment *candidates* is further analysed using simulation-based testing (**RQ2**) to assess whether required timing behaviours of service interfaces hold.

To do so, we generate a test setup for each service interface.

*1) Generation of Source Code:* The source code generation provides basic elements of a test setup. These comprise source code stubs for test stimuli and test oracles containing timing conditions to be checked.

*a) Test Stimuli:* As we are concerned with the assessment of timing properties and not the particular algorithmic implementation of services, our test stimuli contain only source code stubs. Based on a generated source code stub, a runnable service application is implemented and then included into a test scenario.

*b) Test Oracles:* For the test oracles, we transform the timing properties of a service interface into run-time conditions that must hold. For a periodic notification event, it is checked whether the freshness of received data is given. For a method, it is checked whether a latency for receiving the response on a send request is not violated. The run-time conditions are used within the architecture design in order to assess the timing behaviour of an architecture candidate. For the subsequent integration on a physical target device, the run-time conditions are used to monitor the execution of a service from the client's perspective. A significant benefit of service-oriented architectures is the exception handling in the case of violated conditions that, by default, triggers a reconfiguration based on IP-based protocols such as UDP or TCP.

*c) Template-based Code Generator:* We generate source code with a template-based approach using the OMG model to text transformation language [37]. Listings 1 to 3 demonstrate the C/C++ code generation based on the SOA model in Fig. 4. This source code is used for timing simulation later on. For periodic notification events, we generate a *stub (service task)* and a corresponding *monitor (client task)* encapsulating the timing condition specified in the corresponding service interface. Note, the event counter variable used in the condition is incremented by a standard function responsible for message handling that we include into our test scenarios. In addition, the source code for the client reflects the run-time mitigation strategy if the condition is violated. In such a scenario, the system is reconfigured by calling `reconfiguration_procedure()`. The

```
[template interface_generator(c : Class)]
#include <cstdint>
#include <cfloat>
[guard (c.oclsTypeOf(ServiceInterface)]
[for (as:Association | c.Associations())]
  [let asEnd : c1 = as.ownedEnd->select
    (type<>class)->asSequence()->first()]
// Include generated structs
#include <[asEnd.name]>
  [/let]
[/for]
void run_service_task()
{
    // Source code stub
}
[/guard]
[/template]
```

Listing 1. Template to generate stub for service task.

```
[template struct_generator(c : Class)]
[guard (c.oclsTypeOf(Struct)]
// Header file data_structure.h
struct [c.name]
{
  [for (a : Attribute) | c.attribute)];
    [a.type.name/][a.name/];
  [/for]
  [for (as : Association | c.Associations())]
    [let asEnd : c1 = as.ownedEnd->select
      (type<>class)->asSequence()->first()]
      [struct_generator(asEnd)/]
    [/let]
  [/for]
};
[guard]
[/template]
```

Listing 2. Template to generate data structure included by service task.

`reconfiguration_procedure()` targets on subscribing to a second available service instance.

*2) Environmental Model:* For the introduced environmental model service interface in Fig. 6, the application of the templates yields the source code in Listings 4 and 5. The included struct in the source code of the service task is given in Fig. 5c.

## IV. Evaluation

In the following, we evaluate our approach by means of the already introduced *environmental model* function as part of an automated driving architecture. The evaluation has two main objectives: (1) The synthesis of *function to core mappings* that we call *deployment blueprints* (or deployment candidates). These blueprints are the result of a *cross-domain deployment optimisation* (**RQ1**) describing a centralised consolidation of functions from several automotive domains on a centralised computing platform and topology. (2) Verification of real-time requirements of the deployed environmental model (**RQ2**). We use the well-established modelling tool PREEvision [38] for architectural modelling. We extended this MDD-tool with DSE functionalities to provide the best possible user experience in a seamless modelling approach.

```
[template monitor_generator(c : Class)]
#include <cstdint>
#include <stdlib>
[guard (c.oclsTypeOf(ServiceInterface)]
// eventCounter
static std::uint8_t eventCounter;
void generate_client_monitor_task()
{
 // initialise eventCounter
 eventCounter = 0;
 // identify period and required amount of events
 [for (a : Attribute) | c.attribute)]
   [guard (a.name = "period_ms")]
     // implement period
     sleep([a.period_ms]);
   [/guard]
   // implement condition for required events
   [guard (a.name = "requiredEvents")]
     if (eventCounter < [a.requiredEvents])
     {
         reconfiguration_procedure();
     }
     else
     {
         do_nothing();
     }
   [/guard]
 [/for]
}
[/guard]
[/template]
```

Listing 3. Template to generate monitor of client task.

```
#include <cstdint>
#include <cfloat>
#include <obstacle.h>
void run_service_task()
{
    // Source code stub
}
```

Listing 4. Example task for service using header file.

### A. Logical Service Architecture

Fig. 8 depicts a simplified logical architecture for automated driving. It consists of several components each providing/consuming data. Functions are structured in a layered architecture as proposed by Kugele et al. [26]. The lowest layer consists of functions that *sense* the environment using sensors like lidar, radar, or cameras and furthermore affect the vehicle's movement using actuators for steering and (de-)acceleration. Sensory information is the data basis for the vehicle's environmental model located on the second layer and responsible for perception, sensor fusion, and scene understanding. As we can see, the driving dynamics function as the actuator coordination unit is also located on that layer. The environmental model function provides data to the *trajectory planning function* on the top-most layer. On this basis, the trajectory planning function calculates control requests that are finally transformed by the driving dynamics functions to lateral and longitudinal control actions, i. e., accelerating, braking and steering. The architecture concept reflects automated driving at level 4 according to the SAE J3016 [1] classification.

```
#include <cstdint>
#include <stdlib>
// event counter variable
static std::uint8_t eventCounter;
void run_client_monitor_task()
{
    // initialise eventCounter
    eventCounter = 0;
    // implemented period
    sleep(30);
    // implemented condition for required events
    if (eventCounter < 3)
    {
        reconfiguration_procedure();
    }
    else
    {
        do_nothing();
    }
}
```

Listing 5. Example monitor task.

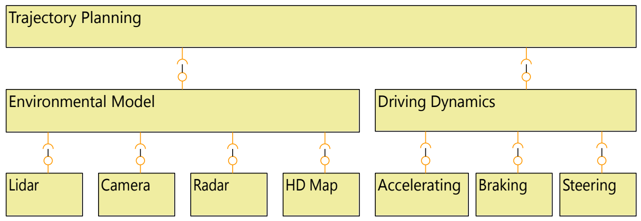


Fig. 8. Logical service architecture concept.

*Cross-domain centralisation and consolidation:* Besides the mentioned functions responsible for automated driving, we also aim at centralised consolidating other functions onto the new platform computers, too. As those functions originate from other domains, we refer to this step as *cross-domain optimisation*. In total, we consider 25 software functions from different domains: infotainment (6 functions), comfort (15 functions), and assisted/automated driving (4 functions).

*Fail-operational behaviour:* The environmental model is deployed twice, one *nominal* and one *fail-operational* (FO) instance (cf. Section III-B). Note, Fig. 9 also includes functions for sensing and acting. Since their allocation is fixed to the respective physical device, they are not part of the design space exploration process.

*Resource footprint:* Each function has a specific *resource footprint* with respect to (i) utilisation of the processor core, (ii) flash memory, (iii) volatile memory, and (iv) ASIL classification. Details are presented in Tab. III. For confidentiality reasons, we cannot give the precise ASIL classification per function, the processor core utilisation, and the memory consumptions, but we give their assignments to classes. Based on a single-core processor operating at a clock speed of 200 MHz as a baseline, the number of functions that require up to ten, up to 100, or more relative core utilisation is given. Similarly, for the memory consumption and the ASIL classification, three classes with the numbers of their containing

TABLE II
RESOURCE FOOTPRINT AND SAFETY REQUIREMENTS OF FUNCTIONS.

| | | | | |
|---|---|---|---|---|
| **Processor utilisation** | [%] | $[0, 10]$ | $(10 - 100]$ | $(100 - 1000]$ |
| | [#] | 3 | 21 | 1 |
| **Memory consumption** | [KiB] | $[0, 100]$ | $(100, 1000]$ | $(1000, 15000]$ |
| Flash | [#] | 1 | 15 | 9 |
| RAM | [#] | 8 | 17 | 0 |
| **ASIL classification** | | QM | B | D |
| | [#] | 14 | 8 | 3 |

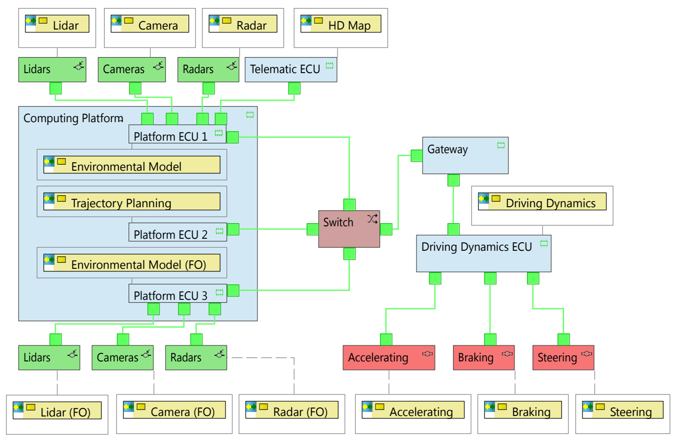


Fig. 9. Network topology

functions are shown. ASIL requirements are necessary for selecting appropriate processor cores that feature the specific ASIL qualifications. The corresponding integrity levels are derived from a thoroughly conducted assessment by safety experts of the company. The environmental model and the trajectory planning (both instances) functions require the highest level (ASIL D) since they significantly contribute to a level 4 automated driving architecture. From the remaining 22 functions, 8 functions have an assigned ASIL B requirement and 14 non-safety-critical functions have assigned QM (Quality Management) requirements.

## B. Computing Platform

Fig. 9 depicts the centralised computing platform for automated driving at level 4. It consists of three ECUs equipped with a different number of cores and hardware accelerators. Table III gives an overview of the specific characteristics of the ECUs. For confidentiality reasons, we cannot give the ASIL classification per core, but their absolute distribution among the ASIL levels: QM (3), A (0), B (15), C (0), and D (7). For the same reason, we cannot state the sizes of the predefined per core RAM partitions, but again, their rough distribution is mentioned in the table. Flash memory is shared among the cores of an ECU.

## C. Deployment

Some functions are statically deployed onto a fixed ECU. These are *Environmental Model* (ECU 1), *Environmental*

TABLE III
CHARACTERISTICS OF THE CENTRALISED COMPUTING PLATFORM.

| **ECU** | **Cores** | **Clock**[a] [GHz] | **Flash** [MiB] | **RAM**[b] [MiB] | **GPUs** |
|---|---|---|---|---|---|
| ECU 1 | 9 | $0.2-2$ | 512 | 2.506 | 1 |
| ECU 2 | 10 | $0.2-2$ | 256 | 4.002 | 0 |
| ECU 3 | 6 | $0.2-2$ | 512 | 8 | 1 |

[a] Clock speed distribution: 0.2 GHz (1), 0.3 GHz (12), and 2 GHz (12)
[b] RAM allocation: $<$ 1 MiB (2), $>=$ 1 MiB (10), $>$ 100 MiB (13)

*Model (FO)* (ECU 3), *Trajectory Planning* (ECU 2), *HD Map*[3] (Telematic ECU), *Driving Dynamics* (Driving Dynamics ECU), and all sensor and actuator functions to their respective devices. ECU 1 and its connected sensors are part of the *nominal* execution path, whereas ECU 3 and its connected fail-operational sensors form the *fail-operational* path.

*Optimisation:* The DSE process is automated by encoding the deployment problem into an SMT instance. For this purpose, performance, memory, and safety requirements are formulated as constraints, and the two optimisation objectives stated in Section III-C1 form the objectives that simultaneously need to be minimised: (1) First, we want to minimise the number of used cores; (2) Second, we want to prevent non-safety-critical software components to be deployed onto cores with a high ASIL qualification. Additional cores can be removed to save money, which is in many cases a crucial factor in the automotive domain. The second optimisation function ensures efficient utilisation of highly qualified processor cores. We achieve this by detecting software function using processor cores that overfulfil posed ASIL requirements as violations. The subsequent rating of violations is done by penalty values (cf. Section III-C1).

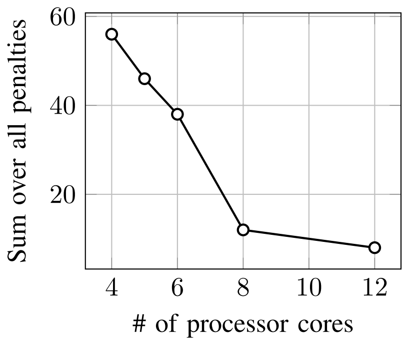


Fig. 10. Pareto-front of the candidates.

We use the Z3 SMT solver [39] supporting optimisation. Fig. 10 shows the Pareto-front with 5 candidate solutions. The $x$-axis shows the number of processor cores. The $y$-axis shows the sum of penalties. Recall that we wanted to *minimise* both values.

Based on the found Pareto-optimal deployment blueprints, network timing analyses are conducted in the next step. In the further course of this evaluation, we exemplify this for candidate with the *lowest* value of the penalty function. This candidate is of special interest because hardware components are expensive and in turn their efficient usage of great interest. This deployment candidate requires 12 processor cores and has a penalty of 8.

[3]The HD Map function is not part of the fail-operational driving mode.

## D. Timing Analysis

We conduct two simulation experiments: **(E1)** The first experiment only uses a single environmental model without the possibility to fail-over to the redundant instance. Whereas the second experiment **(E2)** also considers a reconfiguration at run-time to activate the fail-operational path.

We consider the already discussed topology and the selected architecture blueprint. Communication is based on switched Ethernet using the Audio Video Bridging (AVB) standard [40]. Traffic is shaped using the credit-based shaping algorithm [41].

A two-steps process retrieves a simulation model for the selected configuration, which is discussed below.

**Step 1:** We use chronSIM [42], a tool for timing simulations. It operates on the network topology that has been exported and converted directly from PREEvision.

**Step 2:** The software for the simulation of the case example is implemented in C/C++. The basic concept is a middleware supporting the idea of a publish-subscribe pattern (cf. Fig. 2). Therefore, we generate two source code artefacts using a template-based code generator as plug-in for PREEvision:[4] (1) The source code stub for the environmental model as test stimuli and (2) the monitor source code for the trajectory planning function as test oracle (cf. Section III-D). The test oracle checks whether the specified timing property of the environmental model's service interface is met. In our test setup, the timing property of the environmental model service interface describes that within a period of 30 ms at least 3 updated list of obstacles (events) needs to be provided (cf. Fig. 5b). These lists are sent in single Ethernet frames with a size of 1.5 kB.

*1) Simulation Experiment (E1):* For the first experiment, we simulate the deployed architecture (cf. Fig. 9) without the option to reconfigure the system in case of a timing violation and to switch over to the fail-operational instance of the environmental model. Hence, we only focus on the communication path from ECU 1 to ECU 2 via the switch.

Since other functions also communicate over this path, we take into account additional load at the Ethernet ports of the ECUs. To do so, we *stress* the Ethernet port of ECU 2 by periodically sending 10 kB every 200 ms from the Driving Dynamics ECU (via the gateway). Additionally, ECU 3 sends periodically every 25 ms 25 kB via the switch to the port of ECU 2.[5] Both values are estimates by networking experts.

Variation is included in the simulation runs by using a jitter of 1 s for the start time of the publish/subscribe process (cf. Fig. 2) between the environmental model and trajectory planning function.

*Result:* For all 500 runs, the timing property of the environmental model function was not violated.

*2) Simulation Experiment (E2):* We now introduce the fail-operational instance of the environmental model. This means that we can reconfigure at run-time. The reconfiguration is described as follows: As soon as the monitor of the trajectory

[4]This generator is implemented in line with the concept from Section III-D1.
[5]Both data amounts are serialised in Ethernet frames of 1.5 kB.

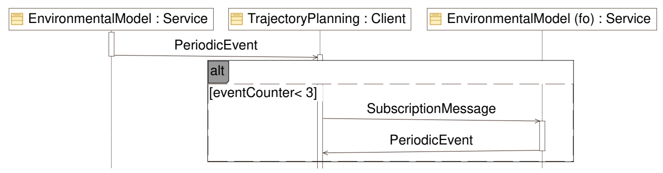

EnvironmentalModel : Service
TrajectoryPlanning : Client
EnvironmentalModel (fo) : Service
PeriodicEvent
alt
[eventCounter< 3]
SubscriptionMessage
PeriodicEvent

Fig. 11. Reconfiguration protocol

planning function detects a violation of the environmental model's timing requirement, the subscription to the functionally identical (i. e., providing the same service) fail-operational instance takes place.

Fig. 11 depicts a sequence chart of the run-time reconfiguration between the two service instances of the environmental model from the perspective of the trajectory planning as client application. However, this creates additional latency because of executing the publish/subscribe process again. We take this into account by defining a timing requirement on the *maximal reconfiguration time* with the help of safety experts which is 130 ms. Again, we introduced variation into the simulation runs by using a jitter of 1 s for the start time of the publish/subscribe process between the environmental model (nominal instance) and trajectory planning function. To enforce the reconfiguration process of the environmental model, we make sure that the nominal instance does not provide its service in time. We do this by assigning the lowest priority to the corresponding Ethernet frame, yielding a delay when entering the Ethernet switch. As a result, the timing requirement is violated and the reconfiguration process is triggered.

*Result:* For 500 simulation runs, we could show that our reconfiguration concept mets the timing requirement of 130 ms for 498 cases. From the successful simulation runs, 21 runs dropped into the defined interval of critical runs which spans from 117 ms up to 130 ms.

*3) Overall Simulation Results:* From the simulation results, we derive that the considered architecture candidate meets posed timing requirements in a test-based manner at least for the case without reconfiguration. For the second experiment, we have already identified by simulation that run-time reconfiguration is challenging when having hard real-time requirements. Here, the shared medium in the form of the Ethernet topology depicts the bottleneck and will be considered for improvement. Moreover, other candidate blueprints need to be analysed, too.

## V. Discussion

The evaluation demonstrates that the approach for analysing and synthesising resources of service-oriented automotive software architectures can be implemented with existing techniques and tools from the field of model-based system design.

### A. Assumptions

Our approach assumes a component-based model of a service-oriented architecture embedded into an overall automotive E/E architecture model respecting also hardware. In addition, we assume to have fine grained hardware resource requirements at an early stage of development. Both assumptions are not unrealistic but depend on a strong collaboration between E/E architects, software architects, and functions developers beyond the border of one company.

### B. Limitations

For the software deployment in line with a central computing platform, we considered 25 software functions from the automotive domains of infotainment, comfort, and assisted/automated driving. As a complement for future work, we aim to scale this amount on two ways:

(1) Extending the amount of software functions from already accounted domains, (2) and extending the amount of software functions from further domains, e. g. the domain of vehicle dynamics. Another limitation is the evaluation example. Here, we only considered a reconfiguration for the environmental model based on two service instances and in turn not an overall fail-operational concept. For achieving this, all components of an automated driving architecture need to have fail-over capabilities. Finally, the applied timing assessment method does not provide theoretical worst-case considerations.

## VI. Conclusion

Service-oriented architectures promote a better run-time decoupling between software and hardware than currently dominant signal-oriented automotive software architectures. Also, design-time activities are eased since communication dependencies between distributed software applications do not need to be predefined. Still challenging is the deployment of a service-oriented architecture on computing platforms as the implemented software functions pose diverse hardware requirements. For the future, it is likely that the resulting deployment problem focuses on a less distributed network than today. In pursuing this, the design of a central platform offering diverse computing capabilities remarks the key concept.

In this paper, we embedded the deployment problem of service-oriented architectures into such an idea. The central elements of the approach are: (1) Models of service-oriented automotive architectures fostering logical and deployment aspects, (2) synthesis of architecture candidates that are optimal according to efficient utilisation of computing resources, and (3) run-time assessments of architecture candidates in a simulation-based manner.

Besides the evaluation example, the roll-out of the approach on a larger scope of projects is currently done. Here, we already gained positive feedback from architects as we base on PREEvision as an established tool. In contrast to existing use cases for PREEvision at the BMW Group focusing on signal-oriented system design, we contribute by an approach for service-oriented systems. For the future, we will strengthen model-based techniques for all kinds of architectural innovations intending to assess their suitability at an early stage of development.

## References

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

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

[3] M. Traub, A. Maier, and K. L. Barbehön, "Future automotive architecture and the impact of IT trends," *IEEE Software*, vol. 34, no. 3, pp. 27–32, 2017.

[4] S. Kugele, V. Cebotari, M. Gleirscher, M. H. Farzaneh, C. Segler, S. Shafaei, H.-J. Vögel, F. Bauer, A. Knoll, D. Marmsoler, and H.-U. Michel, "Research challenges for a future-proof e/e architecture – a project statement," in *15. Workshop Automotive Software Engineering, Proceedings, Chemnitz, Germany*. LNI, Sep 2017.

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

[6] *Data Distribution Service (DDS)*, 1st ed., OMG, April 2015. [Online]. Available: http://www.omg.org/spec/DDS/1.4/

[7] J. Bosch, Ed., *Continuous Software Engineering*. Springer, 2014. [Online]. Available: https://doi.org/10.1007/978-3-319-11283-1

[8] "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.

[9] M. Broy, M. Gleirscher, P. Kluge, W. Krenzer, S. Merenda, and D. Wild, "Automotive architecture framework: Towards a holistic and standardised system architecture description," Technische Universität München, Tech. Rep., 2009. [Online]. Available: ftp://ftp.software.ibm.com/software/plm/resources/AAF_TUM_TRI0915.pdf

[10] Y. Dajsuren, "On the design of an architecture framework and quality evaluation for automotive software systems," Dissertation, Technische Universiteit Eindhoven, Eindhoven, 2015. [Online]. Available: https://pure.tue.nl/ws/files/15934981/20160307_Dajsuren.pdf

[11] P. Pelliccione, E. Knauss, R. Heldal, M. Ågren, P. Mallozzi, A. Alminger, and D. Borgentun, "Automotive architecture framework: The experience of Volvo Cars," *Journal of Systems Architecture*, vol. 77, 03 2017.

[12] K. Pohl, M. Broy, H. Daembkes, and H. Hnninger, *Advanced Model-Based Engineering of Embedded Systems: Extensions of the SPES 2020 Methodology*, 1st ed. Springer Publishing Company, Incorporated, 2016, ch. 3. SPES_XT Modeling Framework.

[13] Daniel Josef Gebauer, "Ein modellbasiertes, graphisch notiertes, integriertes Verfahren zur Bewertung und zum Vergleich von Elektrik/Elektronik-Architekturen," Dissertation, Karlsruher Insitut für Technologie, Karlsruhe, 2016. [Online]. Available: https://publikationen.bibliothek.kit.edu/1000062484

[14] T. N. Qureshi, M. Törngren, M. Pessson, D.-J. Chen, and C.-J. Sjöstedt, "Towards harmonizing multiplearchitecture description languages for real-time embedded systems," in *Real-Time in Sweden (RTiS)*, 2011, qC 20120214. [Online]. Available: http://www.mrtc.mdh.se/rtis2011/

[15] "EAST-ADL domain model specification," 2013. [Online]. Available: http://www.east-adl.info/Specification/V2.1.12/EAST-ADL-Specification_V2.1.12.pdf

[16] M. Jaensch, *Modulorientiertes Produktlinien Engineering für den modellbasierten Elektrik/Elektronik-Architekturentwurf.* Karlsruher Institut für Technologie, 2014. [Online]. Available: https://books.google.de/books?id=mhT1pN4Xg3UC

[17] OMG, "Meta object facility," 2016. [Online]. Available: http://www.omg.org/spec/MOF/

[18] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353)*, vol. 4, May 2000, pp. 101–104 vol.4.

[19] K. Richter, "Compositional scheduling analysis using standard event models," Ph.D. dissertation, Dec 2004. [Online]. Available: https://publikationsserver.tu-braunschweig.de/receive/dbbs_mods_00001765

[20] "chronval," https://www.inchron.com/tool-suite/chronval.html, accessed: 2019-01-16.

[21] R. Ernst, S. Kuntz, S. Quinton, and M. Simons, "The logical execution time paradigm: New perspectives for multicore systems (dagstuhl seminar 18092)," *Dagstuhl Reports*, vol. 8, pp. 122–149, 2018.

[22] *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005): IEEE Standard for Standard SystemC Language Reference Manual*. IEEE, 2012. [Online]. Available: https://books.google.de/books?id=shlZAQAACAAJ

[23] M. A. P. "FMI", "Functional mock-up interface for model exchange and co-simulation," 2014. [Online]. Available: https://fmi-standard.org/downloads/

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

[25] fortiss GmbH, "Welcome to the fortiss AutoFOCUS 3," September 2018. [Online]. Available: https://af3-developer.fortiss.org/

[26] S. Kugele, P. Obergfell, M. Broy, O. Creighton, M. Traub, and W. Hopfensitz, "On service-orientation for automotive software," in *2017 IEEE International Conference on Software Architecture, ICSA 2017, Gothenburg, Sweden, April 3-7, 2017*. IEEE, 2017, pp. 193–202. [Online]. Available: https://doi.org/10.1109/ICSA.2017.20

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

[28] 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.

[29] K. Becker, B. Schätz, M. Armbruster, and C. Buckl, "A formal model for constraint-based deployment calculation and analysis for fault-tolerant systems," in *Software Engineering and Formal Methods*, D. Giannakopoulou and G. Salaün, Eds. Cham: Springer International Publishing, 2014, pp. 205–219.

[30] 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.

[31] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[32] D. Reinhardt and G. Morgan, "An embedded hypervisor for safety-relevant automotive e/e-systems," in *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*, June 2014, pp. 189–198.

[33] S. Kugele, D. Hettler, and S. Shafaei, "Elastic service provision for intelligent vehicle functions," in *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*, W. Zhang, A. M. Bayen, J. J. S. Medina, and M. J. Barth, Eds. IEEE, 2018, pp. 3183–3190. [Online]. Available: https://doi.org/10.1109/ITSC.2018.8569374

[34] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*. Association for Computing Machinery, 1967, p. 483–485. [Online]. Available: https://doi.org/10.1145/1465482.1465560

[35] S. Kugele and G. Pucea, "Model-based optimization of automotive e/e-architectures," in *Proceedings of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis, CSTVA 2014, Hyderabad, India, May 31, 2014*, V. Ganesh and N. Williams, Eds. ACM, 2014, pp. 18–29. [Online]. Available: https://doi.org/10.1145/2593735.2593739

[36] 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.

[37] OMG, "MOF model to text transformation language, v1.0," 2008. [Online]. Available: https://www.omg.org/spec/MOFM2T/About-MOFM2T/

[38] "Preevision," https://vector.com/vi_preevision_de.html, accessed: 2019-01-16.

[39] N. Bjørner and A.-D. Phan, "$\nu$Z - Maximal Satisfaction with Z3," in *SCSS 2014*, ser. EPiC Series, T. Kutsia and A. Voronkov, Eds., vol. 30, 2014, pp. 1–9.

[40] "IEEE standard for local and metropolitan area networks–audio video bridging (AVB) systems," Tech. Rep., Sep. 2011.

[41] "IEEE standard for local and metropolitan area networks - virtual bridged local area networks amendment 12: Forwarding and queuing enhancements for time-sensitive streams," Tech. Rep., Jan 2010.

[42] Inchron, "chronSIM," https://www.inchron.com/tool-suite/chronsim.html, accessed: 2019-01-16.