The software development challenges for today’s vehicles are enormous. Besides the complexity in automobile manufacturing, which has been increasing for years, the introduction of SOA (pronounced “so-ah”) also leads to completely new development paradigms and processes. Conway already postulated in 1968:

> “organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations.” [1]

Thus it is no wonder that the software architecture of today’s vehicles mostly has a silo structure and reproduces traditional organizational structures according to domains such as powertrain, body and chassis. However, effective introduction of a SOA requires completely different cross-domain coordination and communication. Here responsibilities move beyond the previous departmental and organizational boundaries; cooperation and collaboration in a company become implicit preconditions for determining and further developing a SOA and its interfaces. The organizational change also affects the personnel domain. New roles and job requirements arise in the company; the necessary specialists must be trained or recruited.

On the technical side, through service-based communication the requirements intensify the trend towards enormous growth in bandwidth demand in onboard networking. In this regard vehicle manufacturers are relying increasingly on Ethernet technology, which has proven successful in the IT industry. This allows implementation of appropriate bandwidths for applications and communication within the vehicle, as well as between the vehicle and its environment or the Internet. Therefore, tools that take these requirements into account during design and maintenance of the systems are necessary in order to develop future-proof electrical/electronic architectures.

The Advantages of Service Orientation
With a SOA, a service landscape arises that is characterized by clearly defined interfaces. Ideally, these interfaces are described formally and unambiguously, both syntactically and semantically. Clear definition makes it possible to structure services. For example, layered architectures are
generally used which make convey the dependencies on services and represent clear design rules. Furthermore, in a SOA the components are loosely coupled to each other with a service bus as the middleware. The middleware is the intermediary between the service provider and the service consumer; it controls the communication between the service provider and service consumer at system start and usually also defines the data serialization that is managed by the Basic Software (‘BSW’) or Operating System (‘OS’) of the controller (Figure 1).

This precise calculation rule, called a “serializer”, describes the data transfer in the physical bus system. The serializer determines how the data is transformed into a serial bit stream and how it is deserialized on the receiving side.

The use of middleware creates the connection between the service provider and service consumer at runtime – and not at the system design time as previously. This addresses various problems of previous software development. For one thing, partial updates of the system can be performed: if backwards compatibility is ensured, the change to a service does not necessarily have an effect in a software adaptation for all consumers of a service. “Soft” migration scenarios are also possible through use of version management and revision control of service interfaces; this allows all parts of the system to use an older version of the service while another part is already using a new aspect of the service interface. Furthermore, failure safety can be introduced in the design: in the event of failure of one service instance, another equivalent, or at least compatible, service instance can take its place. In distributed systems, that means that not all Electronic Control Units (‘ECUs’) involved in a service necessarily need to be updated.

Last but not least, a consistent service design also allows more powerful design patterns in the implementation. For example, Remote Procedure Calls (‘RPCs’) are common practice in a service-based architecture. RPCs were previously only used within an ECU but now lead to use of client-server paradigms in onboard networking as well. Consequently, the signal orientation in the communication design of vehicle, which had previously been so pronounced, is suppressed.

The SOA Method – A Top-down Approach
If automobile manufacturers and suppliers want to demonstrate the advantages of a SOA, they need an appropriate method that supports the design of new systems with all necessary development steps. The goal here is to manage the complexity while simultaneously making quality-assured designs possible. The method described below explains the development process from the requirements, through the service architecture, to the communication design.
The goal is an AUTOSAR Classic compliant system description for a vehicle or vehicle subsystem. However, the method can also be applied in the context of AUTOSAR Adaptive or to systems outside of AUTOSAR.

The development process starts with the idea of a new functionality whose features are specified. Nowadays this is usually structured according to use cases. The notation of use case diagrams from the Unified Modelling Language (‘UML’) standard has proven successful and established itself for this purpose. With use cases, services can be derived from a catalog of features. Various views are need in order to structure services and illustrate dependencies.

One obvious and already-established view is oriented towards the information flow and thus illustrates the dependencies of functions. The so-called “logical architecture” has established itself for this purpose in the system tool PREEvision: here the logical architecture can be represented in the form of a block diagram and thus describes the functional connection of individual service functions independently of the implementation in hardware or software through event chains (Figure 2).

In addition the UML profile “SoaML”, which is specialized for the description of service architectures, defines logical nodes called “participants”. Participants arise as both providers and consumers of services – in both cases, the exchange takes place via so-called “SOA ports”. These make dependencies between services visible – for example if a participant is the consumer of one or more (basic) services and offers high-quality services in turn. This design step is called “orchestration” and is similar to the widespread event chain view for control functions.

However, in fully dynamic systems in which multiple instances of a service exist within the system as different versions, this representation of the allocation of services to participants is a scenario that can only arise at runtime. Since in the automotive sector services are currently primarily assigned statically to dedicated hardware, this representation can also be understood as the allocation of a service role (provider/consumer) to an ECU. Besides that, participants also make various constellations that arise at runtime visible. Thus the representation of logical service nodes as participants can also be seen as a helpful analysis tool. Last but not least, being familiar with and documenting these constellations is an important foundation for testing and safeguarding such systems (Figure 3).

The SOA ports addressed above are typed by a service interface that is described unambiguously by multiple aspects. It is first necessary to clarify what role a port plays – does it provide a service, or is it a consumer of one? The properties of the interface are described here syntactically: generally valid, technology-independent properties such as available methods, fields, and events – but also data types that are already known at this point and are supported by the middleware, for example the parameters of a method. This completely describes the static behavior of a service.

The semantics of the service contract between the SOA ports are frequently described in text form. This specifies time sequences between service roles and represents protocols, among other things. This design step is also called “service choreography”. The collaboration or message sequence charts (‘MSCs’) proposed by the UML standard are usually used for visualizing sequences.
The formal description of a service, which is initially technology-independent, is typically transported via an interface description (Interface Description Language, ‘IDL’). In modern system design tools like PREEvision, the external structure of the implementation can already be derived from this, for example for the application in the context of AUTOSAR Classic.

This creates the implementation bodies that are necessary for providing further detail about the design and implementing it. Furthermore, the role of the service and the description of the service interface are taken over. The advantage of this consistent approach: changes to the role of a service or the description of a service interface can be carried over to the technology-specific implementation automatically through synchronization methods. Software design interfaces are then derived from the IDL descriptions and represented in the AUTOSAR Classic platform as sender-receiver and client-server interfaces. If file types appropriate for the technology have already been selected, they can be taken over automatically.

The introduction of a special “service interface” is being discussed for the AUTOSAR Adaptive platform. This is intended to contain the properties described above directly and take full advantage of a technology-independent description of services. The goal is to derive implementation bodies for various technologies from the same service definition with the push of a button. The middleware-specific startup behavior should be specified and parameterized after this step. This process is called “service discovery”.

Communication Design

During the further procedure, the data supply and structures of the communication description depend on the communication bus technology and middleware used. AUTOSAR uses Service Oriented Middleware over Internet Protocol (‘SOME/IP’) by default as the transformer for service discovery and data serialization.

However, AUTOSAR also provides for the use of other transformers and middleware. If Ethernet and Internet Protocol (‘IP’) are used as the communication channel, an essential work step is defining sockets in terms of their IP address, transport protocol, and port. And while a description of socket addresses suffices for AUTOSAR Adaptive, for AUTOSAR Classic the signal level must also be specified – since the AUTOSAR Classic basic software stack is designed for signal-based communication using, for example, FlexRay or the more widely-used Controller Area Network (‘CAN’) bus and Local Interconnect Network (‘LIN’) sub-bus. Here integrated system design tools offer a
crucial advantage: they make it possible to derive the signal communication, based on the work products of the previous steps. The service description and information about where in the system a service is to be provided – i.e., which specific ECU implements the provider and which one the consumer – is necessary for this purpose. At this point PREEvision also transfers further properties from the service description into the communication description, for example service and method IDs (Figure 4).

**Summary**
The development tool PREEvision supports the methodical, consistent design of a service-oriented architecture. The user is guided from the definition of the service interfaces, through specification of the interaction of services, to an AUTOSAR compliant Ethernet design through an integrated workflow. If other bus technologies such as CAN, LIN, or FlexRay are also to be used in addition to Ethernet, mixed topologies can also be designed. Thus PREEvision accompanies the system designer through the challenging task of combining classic embedded design with modern service orientation and the necessary back-end communication – and thereby supports the transformation of the automobile into a data center on wheels.

**Figure 4: From service design to communication design**

**Literature References**


**Authors**

Dipl.-Ing. (FH) Markus Helmling
Vector Informatik GmbH
Product Management Engineer PREEvision

Originally published in “Elektronik automotive” magazine
Special issue “Automotive Ethernet” – March 2017