

Low Latency Communication in Service Oriented Networks

Kay Werthschulte

ESG Elektroniksystem- und Logistik-GmbH

Copyright © 2009 SAE International

ABSTRACT

Safety-critical systems often lack scalability and exchangeability between OEMs and different suppliers. Although standards like AUTOSAR ease the development process, data exchanged on the network is subject to misinterpretation and programming faults during the development phase. Therefore this paper describes a way to abstract the needs of several communication networks used (e.g. Ethernet, FlexRay, CAN) from the software development process.

Especially the integration of advanced driver assistance systems will enforce distribution of functional components and the exchange of sensor data necessary to track the status of the vehicle.

INTRODUCTION

Today the data being exchanged is mapped to each communication system into single messages. Existing standards parameterise low-level communication stacks to map the data to locations inside telegrams. The paper will present a means to make all data available to every function and node without losing information encapsulation.

By introducing a service oriented approach with additional software layers the mapping of data will be transparent to higher software layers and ease the development process.

The basic concept decomposes functions into atomic entities. These entities can be categorised into service provider and service consumer. The services are described by their behaviour and their interface. Service consumers can access the functionality through a well-defined shared interface. The link between a service provider and a consumer is established at run time. To provide fast startup times the service provider/consumer relations can be cached for future startups. Each service registers itself at a central catalogue called a service manager. The service manager stores information about service characteristics like execution position (node), interface and a brief description. To use a service the consumer retrieves the needed information from the service manager. Following this step all communication is done directly between the service and the consumer.

The development process is simplified and prevents inconsistencies by assuring that the interfaces could not be mismatched. After defining and implementing the services and service connections, system integration

will be done independent of node position. A repartitioning does not affect the functionality because the linking between service provider and consumer will be done dynamically at run time. It is not necessary to change software/configuration files or to rerun a code generation process.

FUNCTION DECOMPOSITION

Modern EE architectures are often oriented to ECU-centric architectures. Most functions are located on single ECUs lacking composability of software structures. Software written today will often be reinvented due to a missing decomposability and reusability. A shift towards a new approach in software development started with the introduction of AUTOSAR. AUTOSAR introduces the definition of standard software units common to most automotive applications [2]. Even if AUTOSAR does not define the requirements for hardware components, it does provide means to standardise the access to common MCU resources. This also includes the exchange of data inside an ECU and data sent to other nodes on the network. The pairing of nodes sending and receiving data is done statically. All communication links are set up during the early development phase. Repartitioning after this phase has been finished, e.g. during series production, will often impair other ECUs. Changing the software partitioning results in updating the ECU software.

Service-oriented networks work in a different way: The network itself provides a variety of services, which can be accessed by any member. The connections between services and consumer are set up during run time. This concept is well known and has also been proven to work well in widely distributed, embedded systems [1]. Being highly scalable is only one of the advantages. In contrast to statically defined structures, these networks support a high degree of reusability. The main logical structure of a service-oriented network can be seen in Figure 1.

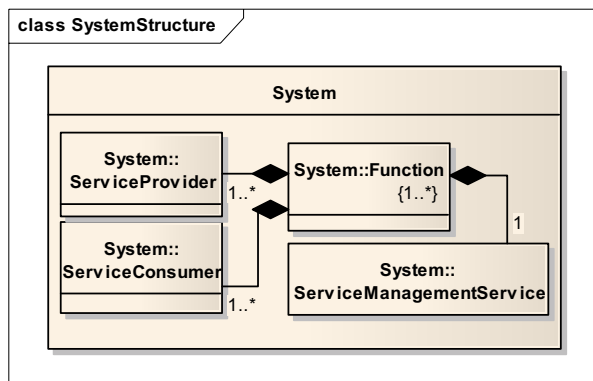


Figure 1: Service oriented system structure

Functions are defined in the sense of a composition of service provider and service consumer. System services are structural elements encapsulating the interfaces and also behaviour being shared across the network. Consumers are using these services as a part of the system function. The *ServiceManagementService* is a specialized and predefined system service immediately available at system start-up. This special service holds all necessary data about the available services in the system, a short description and information about the physical location in the network. The services are only addressed by using a unique network-wide service identifier. The service identifiers are tightly coupled with the software development process. They characterize the unity between service functionality and the exposed interfaces. Any change in the functionality or in the interface declaration must be followed by providing a new service identifier.

The service provider can also be composed of an arbitrary number of service consumers. As this cascading might add execution time, a trade-off has to be found between software reuse and staggered service calls.

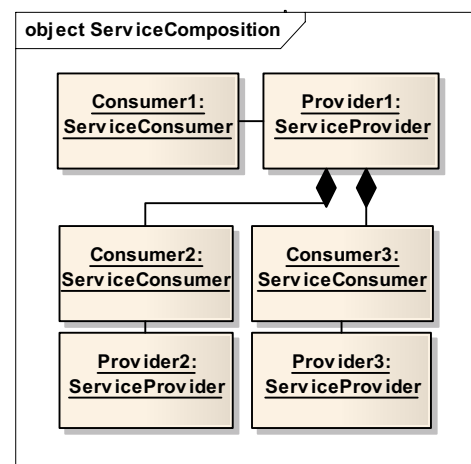


Figure 2: Cascaded service composition

Figure 2 shows an example of interleaved service usage. The *Consumer1* accesses the *Provider1* service interface. The functionality of *Provider1* is composed of using other system services as well. This level of indirection centralizes access to system services and can help keep the overall complexity and the number of interaction interfaces low.

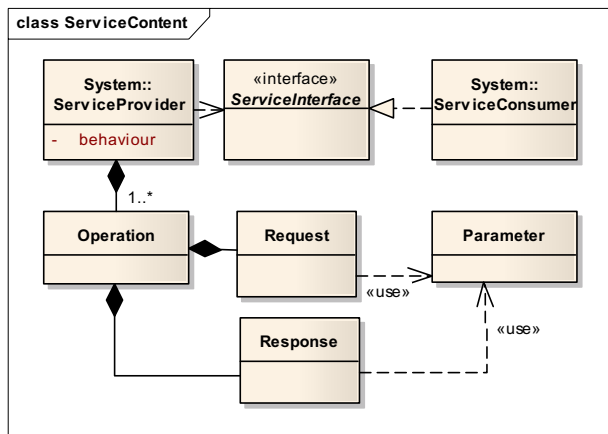


Figure 3: Service structure and interface

Each service consists of a set of operations with input and output parameters defined by the *ServiceInterface*. The *ServiceInterface* is shared between the consumer and the provider. Communication between service consumer and service provider is based on standard requests. These standard requests are comparable to function calls in software. Each call can be an action with or without impact on the internal state of the service.

COMMUNICATION-INDEPENDENT APPROACH

Connecting a service provider and consumer is done during the development phase. After decomposing atomic functions in the network into provider and consumer, the logical connections can be established by linking service identifiers (interfaces) together. This logical linking can be started in an early development phase without needing the actual implementation in software. This procedure enables simulations of the functionality without subcontracting the software development to suppliers. This procedure can be extended by reusing existing services and performing software-in-the-loop tests. Besides enforcing software reuse, the service oriented approach ignores implementation details concerning communication systems. The communication itself, i.e., exchanging data, is reduced to service requests and responses.

The solution is scalable, in order to take the different requirements of communication systems into account. FlexRay, for example, being a time-triggered bus system, provides time-triggered (and also event-triggered) communication by default. Addressing mainly control tasks for safety-critical systems may be added by software stacks with only a little overhead. In contrast to that, IEEE802.3 provides extensive means for transferring large amounts of data across the network.

Protocols like TCP/UDP-IP are available for almost every platform providing connection-oriented and connectionless communication links. Although laid out initially as a non-deterministic robust protocol, enhancements like IEEE1588-2008 allow time critical control applications.

The focus was set on an implementation on an embedded system with constraints on memory and execution time. The main requirements are

- Lean communication protocol
- Reduced memory allocation
- Reduced number of required parallel tasks
- High availability
- Low latency
- Scalability by parameterisation

SOFTWARE ARCHITECTURE

The main concept of a service-oriented architecture requires a layered software structure. The service provider and consumer exchange data according to Figure 3. All services are located in the service layer. The central element is a shared interface between consumer and provider. This interface is not dependent on whether requests are executed locally or remotely. The requests and responses are handled by the lower ComLayer.

The service layer in Figure 4 holds all service providers, consumers, a service directory (*ServiceManagement-Service*) and a *ResourceManagement*. The Resource-Management might not be needed explicitly. This strongly depends on the communication system used. When TCP/UDP-IP-based protocols are used an explicit resource management is not needed, because this function is already fulfilled implicitly by the operating system. As shown later, this is not true for FlexRay. FlexRay only provides communication services up to the Data Link Layer. Because both static and dynamic segments are based on a time division multiple access (TDMA) arbitration scheme, an additional software component is needed. This component will handle allocation and reallocation of slots in the static and dynamic segment. Another more static concept would require that a certain number of communication slots be allocated to each node in the network. In that case, every node could allocate the used slots internally without determining the required bandwidth at a central (software) component. Nevertheless this could be an option for a small amount of exchanged information and if there are only a few members in the system.