A Middleware Solution for Open and Dynamic ICT Architectures in Future Cars

Timo Häckel
Hamburg University of Applied Sciences, Dept. Computer Science,
Berliner Tor 7
20099 Hamburg, Germany
Email: timo.haeckel@haw-hamburg.de

***Abstract*—The Information and Communication Technology (ICT) of today's vehicles is currently experiencing a major revision. With the introduction of a centralised communication medium and the Service-Oriented-Architecture (SOA) paradigm, the automotive industry adopts the challenges posed by connected cars and autonomous driving.**

In novel SOA based ICT architectures the communication middleware for services plays an important role, as it enables services to exchange messages. This paper names the key aspects of such a middleware and specifies our solution. We design a Service-Oriented Architecture based middleware that supports Quality-of-Service (QoS) policies to define the requirements on a connection. Finally, we describe our realisation in the simulation environment OMNeT++.

I. Introduction

Information and Communication Technology (ICT) is already a major driver in the automotive industry. Thus, high quality software components are essential for the competitiveness in the automotive market. Already in 2007 M. Broy determined that ICT contributes up to 50% to the total value of a car and nearly 80% of innovations in the automotive sector were a direct product of the technology transfer from the domain of computer systems [1]. With the recent development and advances in *autonomous driving* the car becomes more and more software dependent and has evolved into a communication hungry system.

With the increasing number and complexity of software components (see figure 1), the demand on processing power and data rate increases. At the same time, the integration effort for new components increases as well, as the interactions of components become harder to predict [2].

Over the last decades the ICT in modern vehicles has developed in an evolutionary way. As support for *legacy systems* has a pivotal role in the automotive domain, well tested and proven components continued to be used in the next generation of cars [3]. Now, features like *drive assistance* and *autonomous driving* as well as new technologies pose new demands on the ICT in modern cars. For example: Dynamically add and find components in the network; Opening the vehicular communication to the Internet; And a much higher bandwidth for communication. These needs cannot be fulfilled by the current ICT architecture, without increasing the complexity and creating security issues [2]. Thus, they lead to the introduction of new ICT architectures. To address the issue of legacy systems, the ICT architecture must be revised so far-sightedly, that they can perform their indispensable role in future cars [2].

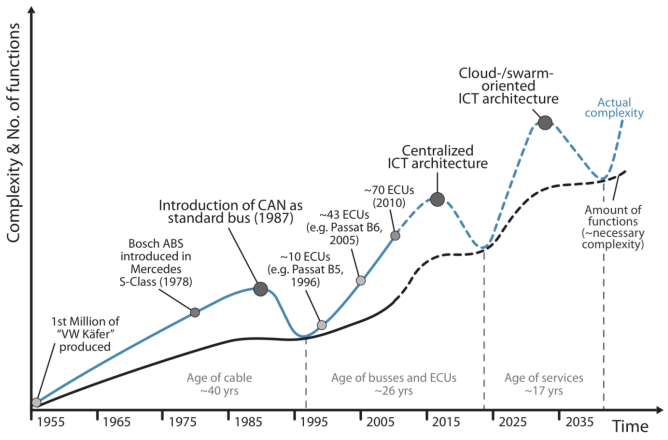

Fig. 1. Evolution of complexity in automotive ICT architectures. [2]

To solve the problems of today's ICT architecture, industry and research approaches contain the following key aspects:

- The introduction of a centralized communication medium with high bandwidth.
- The introduction of dynamic software components with the Service-Oriented Architecture (SOA) paradigm.

This paper aims to describe our middleware solution for open and dynamic ICT architectures in future cars, in methodology, design and realisation.

Therefore, we reflect the most important revisions for a novel ICT architecture by providing some background knowledge about the state of the art in industry and research (section II). We name the challenges for the current ICT architecture and the required transformation. Section III describes the requirements for an automotive middleware solution by analysing domain specific communication characteristics. Based on these requirements, we design features and components, as well as describe used technologies, paradigms and the protocol stack of our middleware solution (section IV). Afterwards, section V describes the realisation process in the simulation environment of OMNeT++. Finally, section VI concludes the paper and gives an overview on future research.

II. Background & Related Work

This section presents related work and background knowledge regarding novel ICT architectures in future cars.

A. Today's and Future ICT Architecture

As previously stated, the ICT Architecture of today's cars was developed in an evolutionary way. Based on the study [4] conducted in Germany in 2011, the authors of [2] discuss the problems in today's ICT architecture and the challenges of a future ICT architecture.

They depict the following problems as crucial regarding today's architecture: Hardware overhead in ECUs due to different manufacturers leads to a waste in resources of micro controllers and networks; Heterogeneous networks with different demands (e.g. time-triggered, priority based, etc.); Increasing demand for interconnectivity and bandwidth, due to functions using *data-fusion* approaches to generate the state of the environment; Complex system verification, due to heterogeneous networks and black-box ECU's; And limited flexibility due to static vehicle configurations.

As figure 1 shows, there is an evident trend for architectures to become more complex than required, considering the evolutionary development of vehicle architectures and the complexity growth over time. According to the previously mentioned study [2], only a revision of the architecture and the use of new technology can bring the complexity down. This could results in much smaller integration costs and an increasing innovation curve. This process has already been observed in the past. For example in 90's the rise of complexity lead to the introduction of micro controllers and new bus systems like CAN-Bus.

For future ICT architectures the authors of [2] name the following key concepts to be implemented, some of them already in use in recent cars. On the one hand side, they depict a list of hardware concepts, such as: A centralised computer architecuter with scalable computing units; The use of smart sensor and actor components; And a standardised communication backbone to replace the heterogeneous networks. On the other hand side, they describe concepts for the software platform used in a novel ICT architecture, such as: A data-centric approach; Meta-data support for extra-functional properties (e.g. timings or fault tolerance); And Plug&Play capabilities for hardware and software components added after sale. The main concept they suggest based on the results of the study is a Service-Oriented communication approach for software components.

Connected vehicles are another big research field in the automotive domain. Already, cars connected to the Internet and exchanging data with smartphones is state of the art. Future cars will be connected to almost everything: Smart homes, Cloud Services, roadside infrastructure and other vehicles around them [5]. As a conclusion cars become part of the Internet of Things (IoT). Thus they function as driving sensor nodes providing data to the Internet [6]. On the other hand they will consume services, e.g. roadside infrastructure. To enable these communications with external devices, the introduction of Web Service standards to the car is another key feature of novel ICT Architectures.

B. Centralised Communication Medium

The most important foundation for a novel ICT architecture is a centralised and open communication medium [3]. It needs to resolve the previously mentioned issues of heterogeneous networks, hardware overhead, increasing demand for interconnectivity and bandwidth, as well as supporting the connection to the IoT.

One technology which is expected to replace all of the currently used In-Car networks is Ethernet. The advantages of ethernet are it's high bandwidth, the vast distribution, low mass production cost and it's wide standardisation. To deploy such a technology to the car there are some important features to be added: Reliability and safeguarding against failure; Security features to ensure robustness against attackers; And Real-Time capabilities.

Many variants of *Automotive Ethernet* are developed in different research groups all over the world. One example is a prototype developed at the CORE research group[1] [7] using a switched *Real-Time Ethernet* backbone. Although there are still some problems regarding Real-Time Ethernet (e.g. background cross-traffic bursts [8]), it is expected to be the main in-car communication medium of the future.

To create a standard for Real-Time capable Ethernet solutions, the IEEE founded the *Time-Sensitive Networking Task Group* to combine and refine already existing standards as well as create new ones 'to provide deterministic services through IEEE 802 networks, i.e., guaranteed packet transport with bounded latency, low packet delay variation, and low packet loss' [9].

C. Service-Oriented Architecture

The Service-Oriented-Architecture Paradigm is a well known and widely used software design method to separate components of a *system* into *services* which interact over a network. One or more services support or automate a business function and are realized with software and hardware [10]. Although SOA originated from web technologies and has many different implementations, it is not about the implementation but the design paradigm. The key aspects of a SOA, *reusability* and *decoupled components*, make it a fitting paradigm for the automotive software development. SOA based ICT architectures generally have the advantage of a low level of complexity and integration effort, as well as the ability to dynamically add and find components in the network, which leeds to a high flexibility and can reduce the number of ECU's as services are no longer bound to there location in the network.

Enabled by the use of a centralised communication medium (see II-B), it is now possible to introduce SOA to the car. Already, the main organization for standardisation of the automotive industry, AUTOSAR adopted the challenge and

[1] CORE research group at the Hamburg University of Applied Science, more information at https://core.informatik.haw-hamburg.de

successfully introduced SOA to there platform standards [11]. Over the past decade AUTOSAR has been established as the organization and main driver for the standardisation of software platforms in the automotive industry. As a reaction on upcoming demands and new functionalities, they restructured their portfolio [12]. Besides enhancing the AUTOSAR Classic Platform with Ethernet support and Service-Oriented communication, they recently introduced the AUTOSAR Adaptive Platform [5]. Although this is a first step towards centralised SOA based communications, AUTOSAR's approach is still very conservative as they only use Service-Orientation and Ethernet for software components without hard Real-Time and security requirements.

D. Service-Oriented Middleware

To enable applications to communicate with each other with the SOA paradigm, a Service-Oreciented communication middleware is needed. Obviously there are already hundreds of middleware solutions out there each with its own application. But as most of the times the requirements of the automotive domain are a bit different and are not fulfilled appropriately by the existing middlewares [13]. As we want to design a fitting middleware for the automotive communication, we looked at middleware solutions contributing something new for our project.

With the previously mentioned changes to the AUTOSAR Platform they introduced SOME/IP [13], a Service-Oriented communication middleware which enables *TCP-/UDP-IP* based service communication. SOME/IP successfully introduces SOA to the car and newly-creates IP-based services, but this solution is not real-time capable.

In the industrial automation big sensor and actor networks communicate in real-time. Thus, the research in industrial automation is a nice inspiration for the car. With the Internet of things they introduced service based communication with an open infrastructure for web services [14]. As a base concept they introduced the service classes *Web Services* and *Real-Time Services* and a QoS based protocol stack which we will try to apply to the car.

Another well structured middleware are *Data Distribution services* (DDS) [15] designed by the Object Management Group. They support service endpoints for the publish/subscribe mechanism based on quality of service parameters, which is also a fitting concept for in-car communication. The purpose of the DDS specification can be summarized as enabling the 'Efficient and Robust Delivery of the Right Information to the Right Place at the Right Time'. By their definition QoS (Quality of Service) is a general concept that is used to specify the behavior of a service. Programming service behavior by means of QoS settings offers the advantage that the application developer only indicates 'what' is wanted rather than 'how' this QoS should be achieved. Generally speaking, QoS is comprised of several QoS policies. Each QoS policy is then an independent description that associates a name with a value. Describing QoS by means of a list of independent QoS policies gives rise to more flexibility. DDS provide a very fitting communication pattern and a well structured architecture.

All these solutions provide fitting communication patterns, concepts for QoS based communication and a well structured architecture. But they do not allow the usage of a different protocol stack which is eminent for the heterogeneous requirements of in-car communication.

III. Methodology

As modern cars incorporate software components from very different domains, automotive software has very heterogeneous requirements on communication. On the one hand side, *Multimedia*, *Passenger/Comfort*, *Safety Electronics*, *Engine/Drive-train* or *Diagnostic* components have very different definitions of important communication requirements such as *Real-Time* or *Data Complexity*. On the other hand side, they rely on different *communication patterns*. Thus, a centralised and open communication middleware needs to fulfil the requirements of all automotive software domains. Our approach is designed to enable the following key concepts of communication:

A. Service Orientation

The communication in Service-Oriented Architectures involves three important building blocks:

1) *Service Provider*, creates a service and provides its information to the service registry.
2) *Service Registry*, makes services available for potential consumers.
3) *Service Consumer*, locates a service provider in the service registry and invokes one of its services.

The middleware has to provide functionality to create service providers and consumers, as well as provide a service registry to find services in the network.

B. Protocol Abstraction

Services should not need to know anything about underlying protocols and technologies. The middleware has to provide a protocol abstraction threw meta data attached to the service consumers and providers (section III-D).

C. Communication Patterns

Due to the heterogeneous domain specific requirements of automotive software, the middleware has to enable different communication patterns:

1) *Publish/Subscribe*, to replace broadcast based bus systems and dynamically demand updates on certain information. A publishing service provides an interface to register new subscribing services and delivers information to all subscribed services.
2) *Streams*, to enable sequential delivery of data elements over time (Multimedia).
3) *Messages*, to enable components to deliver information as a message asynchronously to a service in the network.
4) *Remote Procedure Calls* (RPC), allow a service to remotely execute a subroutine of a different service in the network.

D. Quality-of-Service Policies

Quality-of-Service (QoS) policies add meta data to the connection between services and enable services to specify certain properties of the communication. Important policies are e.g.: Throughput, transmission delay, availability, jitter. The middleware needs to provide meta data support threw QoS policies to enable the application to choose which qualities of the connection need to be fulfilled. Besides, the middleware needs to monitor if all policies are maintained.

E. Openness

As the car opens up to the IoT, the middleware needs to provide state of the art Internet technologies such as support for the *Hypertext Transfer Protocol* (HTTP) and *REpresentational State Transfer* (REST) Web Services. It is important that services do not need to know whether a corresponding service is located in- or outside of the vehicle.

F. Lightweight

To support small ECU's and legacy systems, it's important for the middleware to be lightweight and keep a low level of complexity.

IV. Design

As previously argued in section II-D, there are no suitable middleware solutions for the heterogeneous requirements of automotive communication. Therefore, we designed a middleware solution using the key concept mentioned above.

During the design process we focused on four key concepts: Service orientation, openness, protocol abstraction and QoS policies for connections.

Allthough it is important to have a migration and deployment strategy for novel ICT Architectures as described in [3], we do not focus on the lightweights and legacy support, as the goal of this proof of concept is to show how different communication requirements can be combined in one uniform middleware solution. Likewise we do not focus on security aspects, although we recognize the danger in centralising the internal vehicular communication. We believe that security concepts, e.g. encryption and encapsulation, can be added comparable to existing Internet standards.

The following sections will describe the choices we made during the design of our middleware solution regarding: The protocol stack to support web services, IP-based communication and real-time communication; The QoS negotiation process to decide e.g. which protocol to use; Endpoint design for specific communication patterns; And the component design to realise the functionality.

A. Service Classes

The heterogeneous communication requirements of automotive services make it hard to find one unified catalogue of requirements. Therefore, we decided to split them into different categories, which all group services sharing the same profile.

We extracted three different categories of services from the wide range of automotive applications.

1) *Real-Time Services* (RTS): Real-Time timing guarantees, usually small amounts of data in a fairly high frequency.
2) *Standard IP-based Services* (STDS): openness, No hard deadlines, varying amounts of data with varying frequency, low serialisation overhead to support small ECUs.
3) *Web Services* (WS): Web Service Standard support leads to openness, varying amounts of data with varying frequency not knowing where the responding service is located

B. Protocol Stack

The two key features of our middleware solution are the protocol abstraction and the QoS based decision on communication policies. Both of them highly depend on the underlying protocol stack. Based on the layered architecture of the OSI-model we designed our protocol layers to be easily replaceable and reusable.

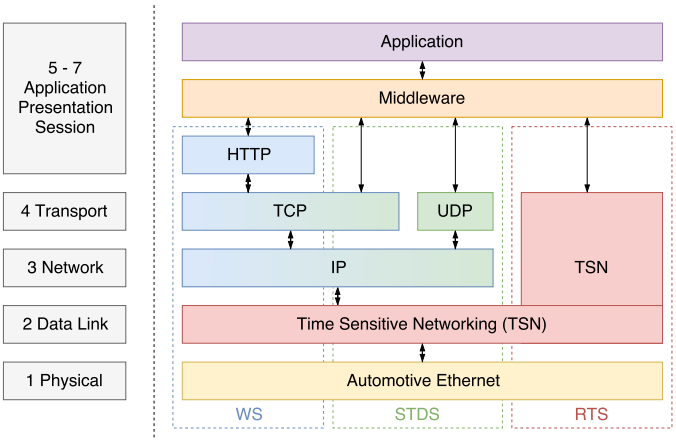

Fig. 2. Middleware protocol stack according to the OSI reference model.

As described in the middleware discussion (section II-D), there are some projects trying to revision communication in industrial automation. One of them designed a protocol stack with different services classes working on different protocols (see [14]). Based on this approach we paired each service classes (section IV-A) with a unique and fitting protocol stack for the layers one to four/five where the middleware is the top level of protocol abstraction with a generalised application interface. This stack is shown in figure 2 and will be described below from bottom up.

The basis for all layers is the physical layer which uses *Automotive Ethernet* (source) technology and defines the electrical and physical data transmission details. As Automotive Ethernet is optimized for the automotive environment it is the obvious choice.

At the second/data link layer Time Sensitive Networking (TSN) Standard [9] is used, to realise Real-Time communication over Ethernet. This layer is responsible for the node-to-node transfer and the flow control. Thus, it is irreplaceable

for Real-Time communication, as only this layer can provide timing guarantees and priority based packet delivery.

On layer three, four and five in the OSI-Model each service class uses a specialised protocol stack by combining well-established protocols.

- *RTS:* Use TSN components for all layers as it provides its own routing and transport mechanism. If TSN can provide the timing guarantee the middleware can provide this guarantee as well.
- *STDS:* To realise basic IP-based services with a low communication overhead, we decided to use the well-established protocols TCP/UDP over IP. This way we can provide openness for STD services as well as use lightweight protocols. On the second layer this communication will be handled as Best-Effort traffic.
- *WS:* For WS it is important to use well-established and widely used standards. The current way to go for Web Services are ReST based Service interfaces using HTTP connections. Therefore, we decided to follow this route and use a HTTP - TCP - IP stack. On the second layer this communication will be handled as Best-Effort traffic.

C. Basic Middleware Architecture

The middleware is realised in four main modules: *Local Service Manager* (LSM), *Local Service Registry* (LSR), *Service Endpoint Factory* (SEF), *QoS Manager* (QoSM). Figure 3 illustrates the relationship between these modules.

The LSM is the entry point of the middleware and contains the main *Application Programming Interface* (API). It holds a reference to all service instances running on the local machine and applications can create and consume services with attached QoS policies. To separate the concerns it consists of three sub modules, each implementing one important part of the API.

To enable service oriented communication middleware clients need to be able to find running services in the network. The LSR provides a local instance of a service registry, by implementing the *Registry Pattern* [16, p. 480]. It holds information to all known services. Furthermore, it provides service discovery functionality to find unknown services in the local network via broadcast.

The SEF uses the *Factory Pattern* [17, p. 107] to provide factory methods for the different endpoints for each communication pattern (more detailed description in upcoming section IV-E). According to a parameter for a QoS Policy list, the factory decides which endpoint type it creates, e.g. a Real-Time capable subscriber.

The QoSM manages the QoS negotiation between the consumer and the provider. A more detailed description can be found in the upcoming section IV-D.

D. QoS Negotiation

As there are already many QoS based middleware approaches, we combined the solution of two projects. In [18], Menascé et al. describe an architecture that includes a QoS broker and service provider software components. In addition

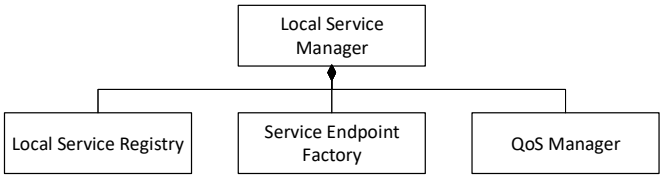

Fig. 3. Basic middleware architecture module overview.

they specify a QoS negotiation protocol with the support of a QoS broker. In the second project [19], Abdelzaher et al. describe their QoS based middleware approach to Real-Time Systems *RTPOOL* and check its application to automated flight control.

Figure 4 and 5 describe the QoS negotiation set-up at consumer and provider side. At first the service must be created. Therefore, the service provider creates the service X at the LSM, which creates an entry in the LSR. To connect to this service, the consumer issues a QoSReguest for service X at the LSM, which tries to find it at the LSR. If the LSR doesn't contain an entry for service X, service discovery will try to find it. If it can't be found, the consumer will be notified that the service doesn't exist, otherwise the QoS negotiation will be started by the QoSM which creates a broker for the clients request. When the request arrives at the machine of the service provider, the QoSM checks if the requested service X exists on this machine. If it doesn't the request will be dropped, otherwise a QoS Broker will be created to handle the request and ongoing negotiation. Both brokers now negotiate the terms of connection via a QoS negotiation Protocol.

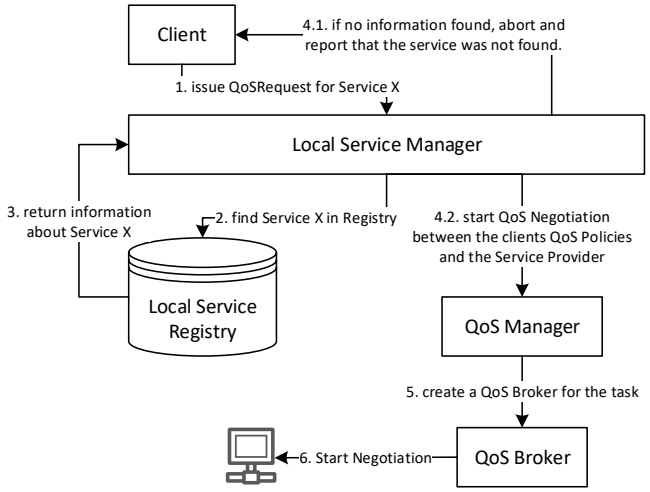

Fig. 4. QoS negotiation set-up at consumer side.

Figure 6 and 7 show the state machines that are implemented by the consumer and provider side QoS Brokers to enable the negotiation via the QoS negotiation Protocol. This Protocol contains a handshake to check if the service is reachable and is generally able to fulfil the requested QoS policies. In the second stage the connection details will be

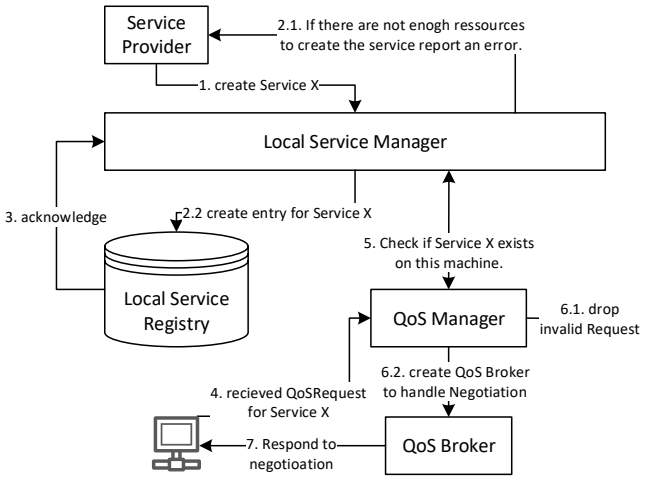

Fig. 5. QoS negotiation set-up at provider side.

exchanged if both brokers agree on the terms of connection.

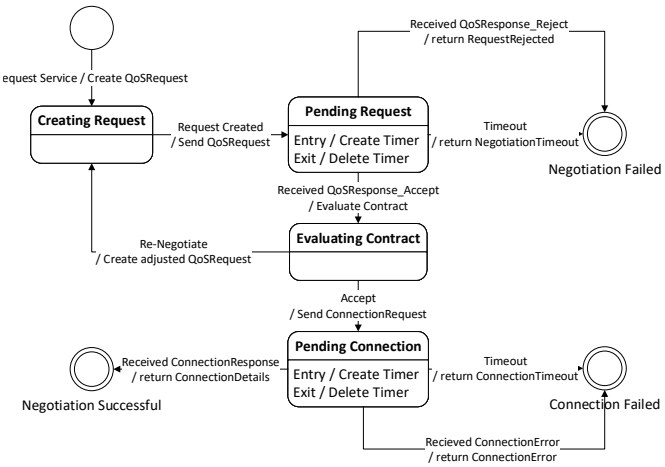

Fig. 6. Statemachine for the QoS Broker at consumer side.

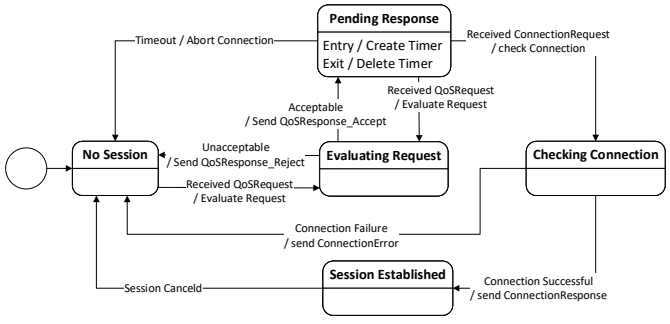

Fig. 7. Statemachine for the QoS Broker at provider side.

E. Endpoints

The middleware provides endpoints for the different communication patterns described in section III-C. As a reference we used the QoS based publish/subscribe middleware *Data Distribution Services* (DDS) [15] (described in section II-D) which delivers data to each subscribed service. We modified the structure to not only allow QoS policies to specify how the data is exchanged, but also manipulate the underlying protocol that is used.

Each endpoint type specifies a base class with the general interface and basic implementations. This base class is implemented by the three subclasses, one for each service class (see section IV-A). They provide adaptations that are generally needed for RTS, WS or STDS as well as the possibility to identify the type of endpoint for prioritisation. In addition, these three service classes again have subclasses, which implement the concrete protocol specific way of communication.

To provide a better understanding of the concept, lets give an example of the publish/subscribe mechanism. To realise this mechanism we need two base classes for the endpoints: *IPublisher* and *ISubscriber*. They describe the interface for all interactions with the subclasses, as the service should not know which specific protocol is used. On the next layer both interfaces have three different implementations, one for each service class: *IRTSPublisher*, *IWSPublisher* and *ISTDSPublisher*. These classes implement the general behavior of publishers in there service class and allow the LSM to identify the publishers service class. As these classes do not provide a concrete implementation for the registration of subscribers and the message delivery, we need another layer of subclasses for the specific protocol we want to use (e.g. TSN for RTS). This way we provide an easy way to add implementations for other protocols.

When an endpoint is created, the service specifies the QoS policies and the LSM chooses which endpoint type and concrete implementation are the best fitting.

V. Realisation

The middleware is implemented as a C++ library, as C is the preferred programming language in automotive embedded services. For the realisation and examination we had two different options:

- *Demonstrator*, an experimental set-up of in-car communications of different micro controller applications connected over RT-Ethernet.
- *OMNeT++ Simulation*, an environment to simulate the communication behaviour of many different nodes in a network over a simulated medium.

We decided to go with the simulation for the following reasons:

- Low level of complexity to start with a prototypical implementation for a proof of concept.
- Flexible creation of different models and scenarios to compare.
- There are existing libraries for all network protocols we want to use.
- Support for debugging and surveillance tools that can analyse the message flow.

As a simulation environment we use OMNeT++[2] (Version 5.1.1) in combination with the INET-Framework[3] (Version 3.5) and the CoRE4INET-Framework[4] (Version nightly/2017-05-31-00-00-06).

VI. Conclusion

This paper has provided an overview on the changes and paradigms hitting on the automotive domain. On the other hand side, we gave an overview on existing middleware solutions and presented our own approach on an automotive service-oriented QoS based middleware.

The presented middleware solution fulfils the specific requirements of the automotive domain by providing different QoS policies for the very heterogeneous communication requirements.

A. Results

As a result we can say that the QoS based approach lives up to the expectations. It allows applications to specify there communication requirements and to choose the QoS they need. On the other hand we noticed that the description of QoS needs to be abstracted from the underlying protocols. In the current state the QoS policies are closely interconnected to the protocol parameters which is suboptimal.

B. Future Research

One important part is refining the QoS policies as described above. In addition, we have not yet implemented a mechanism for the surveillance of QoS policies. Besides the middleware needs to be field tested on a hardware prototype which is the next step. Furthermore, after testing and evaluation of functionality, the security aspects will come to focus. When opening a previously encapsulated system to the Internet of Things it is important to do this safely and to secure the automotive network against a great many of attacks.

References

[1] M. Broy, I. H. Kruger, A. Pretschner, and C. Salzmann, "Engineering automotive software," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 356–373, Feb 2007.

[2] C. Buckl, A. Camek, G. Kainz, C. Simon, L. Mercep, H. Stähle, and A. Knoll, "The software car: Building ict architectures for future electric vehicles," in *2012 IEEE International Electric Vehicle Conference*, March 2012, pp. 1–8.

[3] H. Stähle, L. Mercep, A. Knoll, and G. Spiegelberg, "Towards the deployment of a centralized ict architecture in the automotive domain," in *2013 2nd Mediterranean Conference on Embedded Computing (MECO)*, June 2013, pp. 66–69.

[4] "The software car: Information and communication technology (ict) as an engine for the electromobility of the future," ForTISS GmbH, Tech. Rep., mar 2011, summary of results of the "eCar ICT System Architecture for Electromobility" research project sponsored by the Federal Ministry of Economics and Technology.

[2]OMNeT++ is a simulation environment for network communication. More information at https://omnetpp.org/

[3]The INET Framework for OMNeT++ adds support for the most common protocols in the internet. More information at https://inet.omnetpp.org/

[4]The CoRE4INET Framework builds on to the INET Framework and adds support for simulation models for real-time networks. More information at http://core4inet.core-rg.de/trac/

[5] S. Fürst and M. Bechter, "Autosar for connected and autonomous vehicles: The autosar adaptive platform," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, June 2016, pp. 215–217.

[6] S. K. Datta, R. P. F. D. Costa, J. Härri, and C. Bonnet, "Integrating connected vehicles in internet of things ecosystems: Challenges and solutions," in *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, June 2016, pp. 1–6.

[7] T. Steinbach, K. Müller, F. Korf, and R. Röllig, "Real-time Ethernet In-Car Backbones: First Insights into an Automotive Prototype," in *2014 IEEE Vehicular Networking Conference (VNC)*. Piscataway, NJ, USA: IEEE Press, Dec. 2014, pp. 137–138.

[8] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz, "Beware of the Hidden! How Cross-traffic Affects Quality Assurances of Competing Real-time Ethernet Standards for In-Car Communication," in *2015 IEEE Conference on Local Computer Networks (LCN)*, oct 2015, pp. 1–9, lCN Best Paper Award.

[9] IEEE802.1WorkingGroup. (2018, feb) Time-sensitive networking (tsn). [Online]. Available: https://1.ieee802.org/tsn/

[10] D. K. Barry and D. Dick, *Web services, service-oriented architectures, and cloud computing: the savvy manager's guide*, second edition ed., ser. The Savvy manager's guides. San Francisco, Calif. : Oxford: Morgan Kaufmann ; Elsevier Science [distributor], 2013.

[11] G. L. Gopu, K. V. Kavitha, and J. Joy, "Service oriented architecture based connectivity of automotive ecus," in *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, March 2016, pp. 1–4.

[12] AUTOSAR. (2017, mar) Adaptive platform : Autosar. [Online]. Available: https://www.autosar.org/standards/adaptive-platform/

[13] D. L. Völker, "Some/ip – die middleware für ethernetbasierte kommunikation," *HANSER automotive Networks/2013*, pp. 17–19, 2013, german.

[14] T. Cucinotta, A. Mancina, G. F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, "A real-time service-oriented architecture for industrial automation," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 3, pp. 267–277, Aug 2009.

[15] *Data Distribution Service*, Online, Object Management Group Std. DDS™, Rev. 1.4, mar 2015, access: 2018-02-08. [Online]. Available: http://www.omg.org/spec/DDS/1.4

[16] M. Fowler, *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[17] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[18] D. Menascé, H. Ruan, and H. Gomaa, "Qos management in service-oriented architectures," vol. 64, pp. 646–663, 08 2007.

[19] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin, "Qos negotiation in real-time systems and its application to automated flight control," *IEEE Transactions on Computers*, vol. 49, no. 11, pp. 1170–1183, Nov 2000.