Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Timo Häckel

Automobile Kommunikationsarchitekturen zur Unterstützung von Dienstgütevereinbarungen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Timo Häckel

Automobile Kommunikationsarchitekturen zur Unterstützung von Dienstgütevereinbarungen

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 07. Juni 2018

Timo Häckel

Thema der Arbeit
Automobile Kommunikationsarchitekturen zur Unterstützung von Dienstgütevereinbarungen

Stichworte
Kommunikationsarchitekturen, Fahrzeug, Bordnetz, Dienstgütevereinbarungen, dienstorientierte Architektur, SOA, Middleware, Echtzeit, Ethernet

Kurzzusammenfassung
Die Einführung neuer Funktionen im Auto, von Fahrassistenzsystemen über Connected Cars bis hin zum autonomen Fahren bringt Herausforderungen für die Kommunikationsarchitektur im Auto mit sich. Dazu gehören steigender Bandbreitenbedarf, größere Vernetzung von Komponenten und die Öffnung des Autonetzes zum Internet of Things. Diese können durch die Einführung einer neuen zentralisierten dienstorientierten Kommunikationsarchitektur gelöst werden. Da im Kommunikationsnetz des Autos Dienste mit verschiedensten Anforderungen an die Kommunikationsarchitekturen existieren, ist die Vereinbarung von Dienstgüte ein zentraler Aspekt. In dieser Arbeit werden die verschiedenen Aspekte einer Kommunikationsarchitektur zur Unterstützung von Dienstgüte analysiert. Auf dieser Basis wird ein Konzept für eine Middleware zur dienstorientierten Kommunikation im Auto entworfen und in der Simulation mit Beispielszenarien evaluiert.

Timo Häckel

Title of the paper

Automotive Communication Architectures Supporting Quality-of-Service Agreements

Keywords

Communication Architectures, Car, On-Board Network, Quality-of-Service Negotiation, Service-Oriented Architecture, SOA, Middleware, Real-Time, Ethernet

Abstract

The introduction of new features like Driver Assistance, Connected Cars and Autonomous Driving pose many challenges such as increasing demand in bandwidth, increasing interconnectivity of components and opening the vehicular network to the Internet of Things. To overcome them, a novel centralised service-oriented communication architecture is introduced. As services in the vehicular network have different requirements, Quality-of-Service agreements are a central aspect in the communication of services. This thesis aims to examine the various aspects of automotive communication architectures supporting quality of service agreements. Furthermore, a concept for a service-oriented communication architecture will be designed and evaluated with example scenarios in a simulationenvironment.

Inhaltsverzeichnis

1 Einleitung

Die Information and Communication Technology (ICT) moderner Fahrzeuge ist schon jetzt einer der wichtigsten Innovationsmotoren der Automobilindustrie. Wie Broy (2006) bereits feststellte, sind hochwertige Softwarekomponenten essentiell für die Wettbewerbsfähigkeit im Automobilmarkt. Abbildung 1.1 zeigt die Entwicklung der Bedeutung von Software im Fahrzeug, sowohl in der Softwarekomplexität, als auch im Anteil am Gesamtwert des Automobils. Die Aufgaben der entwickelten Software reichen von Verbesserungen im Komfort, über Steigerung der Performance, bis zur Verstärkung der Sicherheit. Die Abbildung zeigt sehr deutlich, wie stark sich die Automobilindustrie in der Zukunft auf Software fokussieren wird, um dem Käufer neue Funktionen zu bieten.

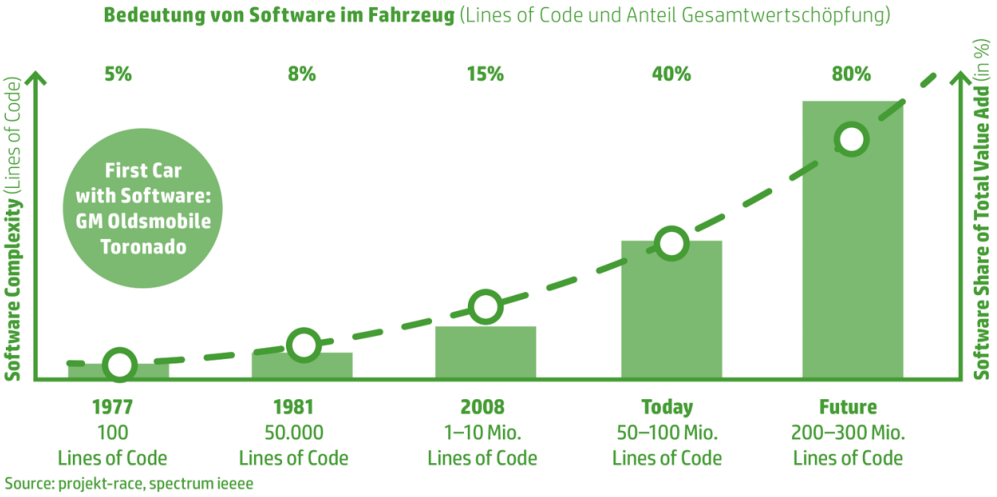


Abbildung 1.1: Entwicklung der Bedeutung von Software im Anteil des Gesamtwertes moderner Fahrzeuge und in der Komplexität der Komponenten. (Quelle: PlattformAuto2018)

Für die Umsetzung dieser Softwarefunktionen werde im Auto eine Vielzahl von Sensoren, Aktoren und anderen Steuergeräten verwendet. Diese sogenannten Electronic Control Unit (Electronic Control Units (ECUs)) sind über das gesamte Fahrzeug verteilt und miteinander verbunden (siehe Abbildung 1.2. Einzelne ECUs erfüllen spezifische Teilaufgaben und

kommunizieren mit anderen, um Funktionen zu realisieren. Dies führt zu komplexen Kommunikationsarchitekturen. Um die unterschiedlichen Anforderungen der Softwarekomponenten an die Kommunikationskanäle zu gewährleisten, werden im Auto verschiedene Bustechnologien. Dazu zählen Controller Area Network (CAN), FlexRay, Media Oriented System Transport (MOST), Local Interconnect Network (LIN) und in den aktuellen Modellen auch schon Ethernet. Mit den jüngsten Fortschritten hin zu komplexen Fahrassistenzsystemen und im autonomen

Abbildung 1.2: Aufbau der ICT mit verteilten und vernetzten ECUs in einem modernen Fahrzeug. (Quelle: http://autoservice-haensel.de/elektrik-elektronik.html)

Fahren, entwickelt sich das Auto zu einem kommunikationshungrigen System. Durch die rasant wachsende Zahl an Steuergeräten, nimmt die Komplexität des Kommunikationsnetzes deutlich zu und wird schwerer zu handhaben. Um neue hoch komplexe Funktionen wie Fahrassistenzsysteme zu realisieren, werden die Subsysteme des Autonetzes immer stärker miteinander vernetzt und der Bedarf an Bandbreite steigt rapide an. So ist die Integration neuer Komponenten immer schwerer da die Abhängigkeiten und Wechselwirkungen schwerer vorherzusagen sind (vgl. Buckl u. a. (2012)). Neben den Anforderungen an die Bandbreite bringen neue Trends wie Connected Cars noch weitere Herausforderungen mit sich, wie z.B. die Öffnung des Autonetzes zum Internet of Things (IoT).

Die evolutionär gewachsene ICT Architektur moderner Fahrzeuge ist so komplex geworden, dass sie Innovationen eher aufhält statt sie zu fördern (vgl. Buckl u. a. (2012)). In Zukunftsszenarien führt sie zu Sicherheitsproblemen, wie z.B. in dokumentierten Hacks (siehe all-electronics.de). Die aktuelle ICT Architektur ist den Herausforderungen der Zukunft somit nicht mehr gewachsen (vgl. Buckl u. a. (2012)). Laut einer Studie der fortiss GmbH (2011),

können diese Probleme nur mit Hilfe einer neuen ICT Architektur bewältigt werden. Diese muss mit genügend Weitsicht gestaltet sein, um ihre unverzichtbare Aufgabe im Automobil auch in Zukunft erfüllen zu können. Hierfür muss ein zentrales Kommunikationsmedium eingeführt werden in dem Softwarekomponenten dienstorientiert kommunizieren können.

In dieser Arbeit soll auf Basis der Forschung im Bereich zentralisierte Netzwerke mit Ethernet Echtzeiterweiterungen (vlg. Steinbach u. a. (2014) und Bello (2011)) eine dienstorientierte Kommunikationsarchitektur entwickelt werden, die auf die Anforderungen des Autos zugeschnitten ist. Hierfür wird eine Middleware-Konzept entwickelt, das in der Lage ist die heterogenen Anforderungen mit einem variablen Protokollstack zu bedienen und den Diensten erlaubt diese Anforderungen in Dienstgütevereinbarungen auszuhandeln. Anschließend wird dieses Konzept in einer Machbarkeitsstudie überprüft. Für diese Realisierung gibt es zwei Möglichkeiten: Entwicklung eines Demonstrators in Hardware oder Entwicklung einer Netzwerksimulation zur Evaluation in einem Szenario. Da die Auswahl der Hardwarekomponenten und die Umsetzung des Protokollstacks auf den jeweiligen Systemen sehr Zeitaufwändig ist und der Fokus dieser Arbeit auf dem Entwurf eine Konzeptes liegt, wurde die Simulation gewählt. In der verwendeten Simulationsumgebung sind die Protokollimplementierungen für Internettechnologien, Komponenten des Autonetzes und Echtzeiterweiterungen für Ethernet bereits vorhanden. Somit kann der Fokus auf der Evaluation des Konzeptes in verschiedenen Szenarien liegen.

Die Arbeit ist wie folgt gegliedert: In Kapitel 2 werden die Grundlagen zum Verständnis dieser Arbeit vermittelt. Dazu gehören Echtzeiterweiterungen für Ethernet, die Erklärung der Konzepte Middleware, Service-Oriented Architecture, Cloud Computing und Connected Cars, sowie die Beschreibung der Simulationsumgebung. Anschließend werden in Kapitel 3 Automobile Kommunikationsarchitekturen analysiert und Ansätze für die Bewältigung der Aufgaben der Zukunft vorgestellt. Kapitel 4 stellt die verwendeten Konzepte zur Klassifizierung von Diensten vor. Außerdem wird ein Vergleich existierender Service-Orientierter Middleware-Lösungen gegeben und eine Strategie zur Verbreitung und Migration entwickelt. Darauf aufbauend wird im Kapitel 5 die Middleware-Lösungen entworfen und die Entwicklungsdetails beschrieben. Dieser Entwurf wird in der Simulationsumgebung umgesetzt und in Kapitel 6 mit Hilfe verschiedener Szenarien ausgewertet. Abschließend fasst Kapitel 7 die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick für darauf aufbauende Projekte.

2 Grundlagen

Um eine Wissensbasis für diese Arbeit zu schaffen, werden in diesem Kapitel Grundlagen aus verschiedenen Bereichen vermittelt. Es wird jedoch Grundwissen über weit verbreitete Standards, wie z.B. Internetprotokolle vorausgesetzt. Im Folgenden werden die verschiedenen in dieser Arbeit verwendeten Echtzeiterweiterungen für Ethernet vorgestellt. Anschließend werden die wichtigsten Grundlagen und Definitionen aus den Bereichen Middleware, Service-Oriented Architecture (SOA) und Cloud-Computing aufgezeigt und die Grundbegriffe der Connected Cars erläutert. Abschließend wird die verwendete Simulationsumgebung für Fahrzeugnetzwerke mit ihren Erweiterungen beschrieben.

2.1 Echtzeiterweiterungen für Ethernet

Da in dieser Arbeit ein zentrales Ethernet Kommunikationsnetz für alle Softwarekomponenten im Auto angestrebt wird, ist es wichtig die verschiedenen Zeitanforderungen der Anwendungen an die Kommunikation zu garantieren. Beispielsweise ist ein verzögertes Bremsen als Folge einer Verzögerung in der Kommunikation nicht hinnehmbar.

Da Standard Ethernet nach 802.3 des Institute of Electrical and Electronics Engineers (2008), keinerlei Garantien für die Qualität der Kommunikation (Quality-of-Service (QoS)) gibt, wurden verschiedene Echtzeiterweiterungen umgesetzt. Diese befinden sich auf der zweiten Ebene im Open Systems Interconnection (OSI) Model, somit auf dem *Data Link Layer*. Ihre Aufgabe ist es, eine zuverlässige, das heißt weitgehend fehlerfreie Übertragung zu gewährleisten und den Zugriff auf das Übertragungsmedium zu regeln. Nur an dieser Stelle können Zeitgarantien gegeben werden. Hierfür werden bei den meisten Erweiterungen neue Nachrichtenklassen eingeführt, welche unterschiedlich starke Zeitgarantien erlauben. Der Standard Ethernet Verkehr wird dann unter dem Namen Best-Effort (BE) beibehalten. Dem OSI Modell entsprechend können in den darüber liegenden Schichten weitere Protokolle eingesetzt werden.

Als Bewertungskriterien für Echtzeitkommunikation werden insbesondere die Latenz und der Jitter verwendet. Die Latenz beschreibt die Verzögerung bei der Übertragung einer Nachricht vom Sender zum Empfänger. Der Jitter gibt die Differenz der maximalen und der minimalen Latenz einer Menge von Nachrichten an.

In dieser Arbeit wird eine Kombination aus Echtzeit-Ethernet Protokollen und Internetstandards als Grundlage für die Kommunikationsarchitektur verwendet. Daher werden in diesem Kapitel die wichtigsten Grundlagen zu den Ethernet Erweiterungen Time-Triggered Ethernet (TTE), Audio Video Bridging (AVB) und Time-Sensitive Networking (TSN) beschrieben. Hiermit soll lediglich ein grobes Verständnis der Protokolle gegeben werden. Für detailliertere Beschreibungen wird auf die Standards verwiesen.

2.1.1 Time-Triggered Ethernet

Time-Triggered Ethernet ist ein auf TDMA[1]-Technologien basierendes Echtzeit-Ethernet-Protokoll und ist von der Society of Automotive Engineers - AS-2D Time Triggered Systems and Architecture Committee (2011) standardisiert in AS6802. Neben BE werden die folgenden zwei neuen Nachrichtenklassen definiert, welche priorisiert behandelt werden.

- *Time-Triggered (TT)*-Verkehr ist die am höchsten priorisierte Nachrichtenklasse, deren Paket auf offline definierten Routen weitergeleitet werden. In der Konfiguration werden die Zeitpunkte für das Senden und Empfangen der Pakete bei allen Knotenpunkten festgelegt. Pakete, für die keine Konfiguration existiert, oder die ihre Konfiguration nicht einhalten, werden verworfen. Da in der statischen Konfiguration auch die Latenz schon definiert ist, liegen die Jitter im Nanosekunden Bereich.
- *Rate-Constrained (RC)*-Verkehr hat die nächst niedrigere Priorität. Die Routen werden ebenfalls in einer offline Konfiguration definiert. Allerdings werden keine Konkreten Zeitpunkte für das Senden und Empfangen festgelegt, sondern ein relativer Mindestabstand. Innerhalb dieser Klasse sind zusätzlich acht numerische Prioritäten definiert.

2.1.2 Audio Video Bridging

Audio Video Bridging wurde vom Institute of Electrical and Electronics Engineers (2011) in 802.1BA-2011 standardisiert. Pakete werden in definierten Nachrichtenströmen von einem sogenannten Talker[2] zu (mehreren) Listeners[3] versendet. Hierfür müssen alle an der Kommunikation beteiligten Komponenten das AVB Protokoll verstehen. Alle Knotenpunkte bilden gemeinsam eine sogenannte AVB Wolke, innerhalb der über eine gewissen Anzahl an Hops,

[1] TDMA ist ein Zeitmultiplexverfahren bei dem Daten von unterschiedlichen Sendern zu jeweils bestimmten Zeitabschnitten versendet werden.
[2] Talker (deutsch: Sender, Sprecher) ist die Quelle eines AVB Nachrichtenstroms.
[3] Listener (deutsch: Empfänger, Zuhörer) ist die Senke eines AVB Nachrichtenstroms.

Zeitgarantien gegeben werden können. Das Protokoll basiert auf 802.1Q (vgl. Institute of Electrical and Electronics Engineers (2014)), weshalb 8 Prioritäten für die Pakete zur Verfügung stehen. Die beiden höchsten werden in AVB jedoch als gesonderte Klassen interpretiert. Diesen wird ein Class Measurement Interval (CMI) zugeordnet, innerhalb dessen eine Bandbreite reserviert werden kann. So kann trotz dynamischer Routen, ein Zeitverhalten garantiert werden. Die Nachrichtenklassen werden wie folgt definiert:

- Klasse-*A*-Verkehr hat die höchste Priorität. Das CMI entspricht $125\,\mu s$. Die maximale Latenz ist mit $2\,ms$ über 7 Hops angegeben.
- Klasse-*B*-Verkehr beschreibt die zweite Prioritätsstufe. Das CMI entspricht $250\,\mu s$. Hier ist die maximale Latenz mit $50\,ms$ über 7 Hops definiert.
- *BE*-Verkehr bildet auch hier die niedrigste Priorität ab, wobei dieser über die verbleibenden 6 Institute of Electrical and Electronics Engineers (IEEE) 802.1Q Prioritäten verteilt werden kann.

Mithilfe des Stream Reservation Protocol (SRP) (vgl. Institute of Electrical and Electronics Engineers (2010) 802.1qat) können zur Laufzeit dynamisch Verbindungen vom Talker zum Listener aufgebaut werden. Der Talker legt dabei die Eigenschaften seines Streams fest (Klasse, Paketperiode, Paketgröße) und macht diesen im Netzwerk bekannt. Ein Listener kann nun eine Verbindung mit dem Stream beantragen. Jeder Knoten auf dem Weg zum Talker reserviert die erforderliche Bandbreite. Wenn der Talker auf diesem Weg erfolgreich erreicht wird, stellt er den Stream bereit. Mit Hilfe eines Shapers wird sichergestellt, dass die Bandbreite nicht überzogen und das keine Priorität ausgehungert wird.

2.1.3 Time-Sensitive Networking

Das Time-Sensitive Networking Echtzeit-Ethernet-Protokoll ist eine Sammlung von Standards die speziell für den Einsatz in industriellen Kontrolleinrichtungen und Kommunikationsnetzwerken innerhalb von Fahrzeugen ausgelegt sind. Die Standardisierung wird in der IEEE 802.1 TSN Task Group durchgeführt.

Im Protokoll werden die Paketauswahlmethoden von AVB und TTE ähnlichem TDMA-Verkehr kombiniert. Hierfür werden, wie in AVB, acht Prioritätsstufen von 0-7 definiert. Mit einem *Transmission Selection Algorithm* ähnlich dem Shaper aus AVB wird ausgewählt, welches Paket wann übertragen wird. Diese sind im Standard IEEE 802.1Qbv definiert (vgl. Institute of Electrical and Electronics Engineers (2016)). Anschließend entscheidet ein sogenanntes *Transmission Gate*. Dieses hat einen von zwei möglichen Zuständen: *OPEN* und *CLOSED*.

Pakete werden nur im Zustand *OPEN* durchgelassen. Damit kann zum Beispiel ein TDMA-Verkehr umgesetzt werden, in dem man in einem bestimmten Zeitfenster nur einzelne Kanäle über das *Transmission Gate* freischaltet.

In TSN kann des Weiteren eine entsprechende Eingangskontrolle an den Ports verwendet werden. Diese ist in dem Standard IEEE 802.1Qci definiert (vgl. Institute of Electrical and Electronics Engineers (2017)). Hier muss eine Nachricht drei Ebenen durchlaufen um nicht verworfen zu werden: *Stream Filters*, *Stream Gates* und *Flow Meters*. *Stream Filters* weisen, auf Basis vorheriger Konfiguration, ein konkretes Gate und Meter zu. Die *Stream Gates* haben, wie *Transmission Gates*, einen der zwei Zustände. Auch hier kann der Zustandswechsel Zeitgesteuert über eine vorherige offline Konfiguration durchgeführt werden. Im Zustand *CLOSED* wird die Nachricht verworfen. *Flow Meters* können dann, ähnlich der *Transmission Selection*, unterschiedliche Verfahren zum Filtern von Nachrichten umsetzten. Wenn eine Nachricht alle drei Stufen passiert, wird sie im Gerät weiterverarbeitet.

2.2 Middleware

Eine Middleware (deutsch Zwischenanwendung) ist eine anwendungsneutrale Software, welche zwischen Anwendungen vermittelt, um die Komplexität der Anwendung und der Infrastruktur zu verbergen. Sie stellt eine Plattform in einem komplexen Softwaresystem dar, die als „Dienstleister“ anderen ansonsten entkoppelten Softwarekomponenten den Datenaustausch ermöglicht.

Meist werden Middleware-Lösungen in einer Schichtenarchitektur aufgebaut. Auf oberster Ebene stehen die Anwendungen, welche über eine Schnittstelle auf die Middleware zugreifen können. Unter der Middleware liegen die verschiedenen Kommunikationsprotokolle und Betriebssystemfunktionen, welche vor der Anwendung versteckt bleiben. Auf unterster Ebene liegt schließlich die Hardware.

Die Aufgabe eine Middleware ist es das Problem der M2M[4] Kommunikation zu lösen. Dieses geschieht auf unterschiedlichen Wegen, welche in drei grobe Kategorien gegliedert werden können:

- *Anwendungsorientiert:* Der Fokus liegt auf der Realisierung verteilter Anwendungen. Hierbei werden RPCs[5] realisiert, bei denen beide Anwendungen so arbeiten können, als

[4] Machine-to-Machine Kommunikation (häufig mit M2M abgekürzt), bezeichnet den automatisierten Informationsaustausch zwischen Maschinen .

[5] Remote Procedure Call (RPC) (deutsch Entfernter Funktionsaufruf), erlaubt Funktionsaufrufe von im Netzwerk verteilten Objekten.

würden sie eine lokale Methode aufrufen. Der wohl bekannteste Vertreter anwendungsorientierter Middleware-Lösungen ist Common Object Request Broker Architecture (CORBA) (vgl. Object Management Group (2012)).

- *Kommunikationsorientiert:* Der Fokus liegt darauf, die Netzwerkprogrammierung zu abstrahieren. Auf diesem Wege können RPC ausgeführt werden. Bekannte Vertreter sind Jave Remote Method Invocation (Java RMI) oder allgemein Web Services.

- *Nachrichtenorientiert:* Sogenannte Message Oriented Middleware (MOMs) fokussieren sich darauf, Nachrichten synchron oder asynchron von einer Anwendung zur nächsten zu übertragen. Dies geschieht mittels *Channels*, die die Anwendungen verbinden. Diese können zum Beispiel für das Puffern eintreffende Nachrichten oder für ausgehende Nachrichten an mehrere Teilnehmer, als Publish/Subscribe Channel, spezialisiert werden. Bekannte Vertreter sind Java Message Service (JMS) oder in modernen Microservice Architekturen RabbitMQ.

2.3 Service-Oriented Architecture

Auch wenn Service-Oriented Architecture weit verbreitet sind, gibt es keine einheitliche Definition. Viele verwechseln das SOA Prinzip mit Web Services, die allerdings nur die am weitesten verbreitete Umsetzung sind. Allgemein wird eine SOA aus Diensten (englisch Services) und Konsumenten (englisch Consumers) zusammengesetzt, die mit einander Kommunizieren und ein gesamt System bilden. Die Kernprinzipien einer SOA nach Bean (2010) sind:

- *Loose coupling (deutsch Lose Kopplung):* Beschreibt die Minimierung von Abhängigkeiten zwischen Diensten und Konsumenten, sowohl in Hardware, als auch in Software. Die Unabhängigkeit in Hardware wird meist durch die Abstraktion der Kommunikation mittels einer Middleware gelöst. Auf der Softwareebene erfolgt dies durch definierte Schnittstellen, kann allerdings lediglich durch ein gutes Design gewährleistet werden.

- *Interoperability (deutsch Kompatibilität):* Beschreibt die Minimierung von technologiespezifischen Einschränkungen, wie z.B. Programmiersprachen, Datenbanksystemen oder Betriebssystemen. Hierfür werden häufig plattformübergreifende Middleware-Lösungen eingesetzt. Außerdem werden Standards zur Kodierung von Nachrichten und Daten festgelegt, wie z.B. die XML[6].

[6] Extensible Markup Language (XML) (deutsch Erweiterbare Auszeichnungssprache) ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten im Format einer Textdatei.

- *Reusability (deutsch Wiederverwendbarkeit):* Ist ein viel genutzter Begriff, der häufig verschiedene Bedeutungen hat. Ein Dienst der nach dem Prinzip der losen Kopplung und Kompatibilität entworfen wurden, ist generell ein guter Kandidat für Wiederverwendbarkeit. Das bedeutet, dass ursprünglich geplante Konsumenten, genau wie neu entwickelte Konsumenten, die Funktionalität des Dienstes ausnutzen können. Eine Voraussetzung um einen Dienst wiederverwenden zu können, ist allerdings die im Folgenden beschriebene Auffindbarkeit.

- *Discoverability (deutsch Auffindbarkeit):* Beschreibt die Funktion, Dienste im Gesamtsystem auffinden zu können. Dies wird meist mit einer *Service Registry* (Fowler, 2002, Seite 480) gelöst. Eine Service Registry ist ein Katalog von bekannten Diensten, der häufig auch nähere Informationen zu den Diensten enthält und Funktionalität zum Suchen und Finden bietet. Voraussetzung dafür ist jedoch, dass ein Anbieter seinen Dienst im Netzwerk bekannt macht.

Generell besteht ein Dienst aus Hardware und Software. Einer oder mehrere Dienste automatisieren oder unterstützen eine Betriebsfunktion. Laut Barry und Dick (2013) gibt es zwei verschiedene Arten von Diensten: *Atomic* und *Composite*. Atomic Services sind wohldefinierte abgeschlossene Funktionen, die nicht von anderen Diensten abhängig sind. Composite Services sind eine Gruppe von Atomic Services oder anderer Composite Services, die eine (neue) Aufgabe umsetzen. Ein Dienst innerhalb eines Composite Service darf Abhängigkeiten zu anderen Diensten innerhalb derselben Gruppe haben.

Zu den bekanntesten SOA Standards zählen:

- Simple Object Access Protocol (SOAP) (vgl. W3C (2007)) erlaubt Nachrichtenaustausch und RPCs in verteilten Systemen. Es basiert auf XML zur Datenrepräsentation und Hypertext Transfer Protocol (HTTP) und Transmission Control Protocol (TCP) zur Kommunikation. SOAP kann in Kombination mit verschiedenen Middlewares wie zum Beispiel Java Messaging Services verwendet werden.

- Web Service Description Language (WSDL) (vgl. W3C (2006)) ist eine Beschreibungssprache für Dienste im Netzwerk, die unabhängig von Protokollen, Plattformen und Programmiersprachen ist. Sie basiert auf dem Austausch von XML Nachrichten. Mit ihr wird das Application Programming Interface (API) und Zugangsprotokolle beschrieben.

- Devices Profile for Web Services (DPWS) (vgl. OASIS (2009)) soll Dienst basierte Kommunikation auf eingebetteten Systemen ermöglichen. Es bietet die üblichen Funktionen einer SOA, erweitert diese jedoch noch um ereignisgesteuerte Datenverteilung.

- Representational State Transfer (ReST) (vgl. Richardson und Ruby (2007)) ist ein Paradigma um M2M Kommunikation im Semantic Web zu optimieren. Es basiert auf dem Angebot von zustandslosen Web Services als Ressourcen im Netzwerk. Die Umsetzungen bestehen aus den Methoden des HTTP.

2.4 Cloud-Computing

Das Buch *Cloud-Computing Patterns* von Fehling u. a. (2014) beschreibt Cloud-Computing[7] als die logische Folge der Arbeitsteilung. Die meisten Unternehmen haben mit dem Outsourcing ihrer IT begonnen. Somit passt sich die IT dem Wirtschaftssystem an. Allgemein bezeichnet Cloud-Computing also das Ausführen von Softwarekomponenten auf Großrechnern in (fremden) Rechenzentren.

Cloud-Computing Anbieter bieten sogenannte *Offerings* in einer Cloud Environment an. Diese reichen von Datenspeicherung und Verbreitung, über das Anbieten von Rechenleistung, bis hin zum Bereitstellen ganzer Umgebungen mit einem Anwendungsmanagement und Kommunikationsmechanismen. Die besondere Kernkompetenz von in der Cloud ausgeführten Softwaresystemen ist die *Skalierbarkeit*. In der Laufzeitumgebung einer Cloud ist es möglich, Anwendungen mehr Speicher oder Rechenzeit zuzuweisen. Außerdem können bei einer großen Zahl von Anfragen, mehrere Instanzen der gleichen Anwendung angeboten werden.

Das Konzept solcher Laufzeitumgebungen wird als *Platform as a Service* bezeichnet. Dieses Konzept ist für diese Arbeit besonders interessant, da so meist eine *Execution Environment* für Dienste und eine *Message-Oriented Middleware* bereitgestellt wird. Auch im Auto werden bereits vergleichbare Plattformen eingesetzt (vgl. AUTOSAR beschrieben von Fürst und Bechter (2016)). Außerdem werden verschiedene Cloud-Dienste benutzt, zum Beispiel Kartendienste und Navigationssysteme. Besonders in der Zukunft von *Connected Cars* (siehe Kapitel 2.5) wird Cloud-Computing auch im Auto eine große Rolle spielen, wie Iwai und Aoyama (2011) beschreiben.

2.5 Connected Cars

Connected Cars beschreiben ein Zukunftsszenario, in dem das Auto mit den Gegenständen seiner Umgebung im Internet oft Things verbunden ist. Dies wird of auch als *Car2X* (oder Car2Environment) Kommunikation bezeichnet. Auch Fürst und Bechter (2016) beschreiben

[7] Cloud-Computing (deutsch Rechnerwolke) beschreibt die Bereitstellung von IT-Infrastruktur wie beispielsweise Speicherplatz, Rechenleistung oder Anwendungssoftware als Dienstleistung über das Internet.

das zukünftige Auto, was mit fast allem verbunden ist: Smart Homes, Road Side Infrastructure und Autos in der Umgebung. Datta u. a. (2016) gehen noch einen Schritt weiter und integrieren das Auto in das IoT Ökosystem von Smart Cities. Aus ihrer Sicht sind die Autos der Zukunft in erster Linie fahrende Sensorknoten.

2.6 Simulationsumgebung

Die Realisierung des in dieser Arbeit entwickelten Middleware Konzeptes erfolgt als Machbarkeitsstudie in einer Simulationsumgebung. Im Folgenden werden die Umgebung OMNET++, sowie die verwendeten Frameworks beschrieben.

OMNeT++ ist ein C++ Simulationsbibliothek mit Fokus auf die Simulation von Kommunikationsnetzwerken. Es ist modular und erweiterbar. Erweitert wird diese Basis mit weiteren Simulationsframeworks. Diese Simulationsumgebung ist in Abbildung 2.1 dargestellt und die einzelnen Komponenten im Folgenden beschrieben:

- *INET* (vgl. OpenSim Ltd.) ist eine Framework für OMNeT++, dass Standard Ethernet- und Internettechnologien umsetzt.
- *CoRE4INET* (vgl. CoRE Working Group (b)) ist ein Framework, dass in der Communication over Realtime Ethernet (CoRE) Arbeitsgruppe (vgl. CoRE Working Group (a)) entstanden ist und auf dem INET-Framework aufbaut. Es enthält Implementierungen von AVB und TTE mit dessen Hilfe in der Vergangenheit schon TSN-Technologien simuliert wurden.
- *FiCo4OMNeT* (vgl. CoRE Working Group (b)) ist ein Framework, dass in der CoRE Arbeitsgruppe (vgl. CoRE Working Group (a)) entstanden ist. Es realisiert Feldbus Kommunikation für CAN und FlexRay.
- *SignalsAndGateways* (vgl. CoRE Working Group (b)) ist ein Framework, dass in der CoRE Arbeitsgruppe (vgl. CoRE Working Group (a)) entstanden ist und auf CoRE4INET und FiCo4OMNeT aufbaut. Es ermöglich die Simulation eines heterogenen Netzwerks mit Hilfe von Gateways[8].

Es werden die Versionen OMNET++ 5.1.1, INET Framework 3.5, das CoRE4INET Framework in der Nightly vom 31. Mai 2017, das FiCo4OMNeT Framework in der Nightly vom 24 Mai 2017, sowie das SignalsAndGateways Framework in der Nightly vom 25. Mai 2017 verwendet.

[8] Gateway (deutsch Einfahrt und Ausfahrt) bezeichnet in der Informatik eine Komponente, welche zwischen zwei Systemen eine Verbindung herstellt.

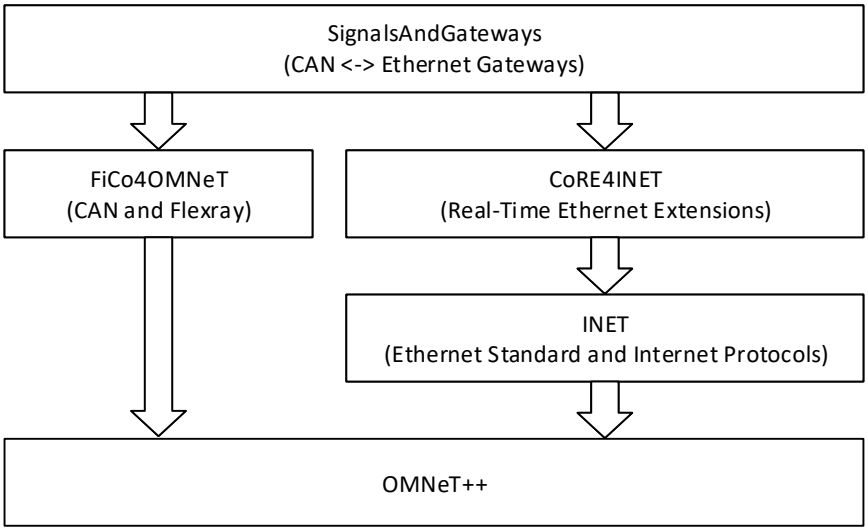


Abbildung 2.1: Erweiterte Simulationsumgebung OMNeT++ mit den Frameworks INET, CoRE4INET, FiCo4OMNeT und SignalsAndGateways.

3 Analyse

Broy (2006) fasst in seiner Arbeit die Geschichte der ICT im Auto wie folgt zusammen: Vor ca. 40 Jahren wurden die ersten Software Funktionen im Auto eingeführt. Diese Komponenten waren lokal isolierte Regelungssysteme ohne Verbindungen zu anderen Systemen. So wurden Standard Architekturen für abgeschlossene Steuergeräte (ECUs) für unterschiedliche Aufgaben geschaffen, die in vielen Generationen von Automobilen der verschiedensten Hersteller eingesetzt werden konnten. Später wurden dann Bussysteme eingeführt, welche ECUs und ihre Sensoren und Aktoren miteinander verbanden. Auf dieser Basis wurden bis heute Funktionen entwickelt, die als Verteiltes System, auf verschiedenen mit Bussystemen verbundenen ECUs ausgeführt wurden. Broy (2006) merkt an, dass dieses System aufgrund des historischen Wachstums aus vielen Einzelteilen zusammengesetzt ist und also nach einem *Bottom-Up* Ansatz von unten nach oben aufgebaut wurde. Er kritisiert, dass aus diesem Grund nie ein richtiger Entwurf des Gesamtsystems, nach einem *Top-Down* Ansatz, gemacht wurde und betont, dass wenn man das Gesamtsystem Auto heutzutage noch einmal entwerfen würde, es vermutlich sehr viel anders aussehen würde. Auch eine Studie der fortiss GmbH (2011) schätzt zwar den Wert der vielen Innovationen den die ICT ermöglichte, von Anti-Lock Braking Systems (1978) über Electionic Stability Control (1995) bis zu Emergency Brake Assist (2010). Sie kommt jedoch wie schon Broy (2006) zu dem Schluss, dass aufgrund der zunehmenden Wichtigkeit von Software Funktionen für den Marktwert des Automobils und der wachsenden Komplexität im Sinne der ausgeführten Funktionen, der genutzten Technologien und der Supplychain, ein kompletter Neuentwurf der ICT im Auto die einzige Chance ist, den Herausforderungen der Zukunft gewachsen zu sein. Die Studie der fortiss GmbH (2011) betont, dass die Einführung der Elektromobilität bereits sehr große Veränderungen über das Gesamtsystem des Autos bringt und dies somit der richtige Zeitpunkt ist, die ICT Architektur anzupassen.

In den folgenden Kapiteln werden die Aufgaben und Chancen einer neuen Kommunikationsarchitektur beschrieben. Außerdem wird dargelegt, welche der zukünftigen Anforderungen von der jetzigen Architektur nicht erfüllt werden und mithilfe welcher Ansätze, eine neue Architektur die Aufgaben lösen kann. Als Grundlage für diese Argumentation wird vor allem auf der Studie der fortiss GmbH (2011) aufgebaut, welche die wichtigsten Akteure der (deut-

schen) Automobilindustrie zu den Herausforderungen der ICT im *Software Car* der Zukunft (ca. 2030) befragt. Die Ergebnisse dieser Studie wurden bereits von Buckl u. a. (2012) diskutiert. Des Weiteren werden verschiedene Ansichten von Akteuren aus der Forschung Chakraborty u. a. (2012); Kugele u. a. (2017) und Vertretern der Industrie, wie Beispielsweise AUTOSAR[1] oder BMW präsentiert.

3.1 Aufgaben zukünftiger Kommunikationsarchitekturen in Automobilen

Damit eine neue Kommunikationsarchitektur in der Lage ist ihr unverzichtbare Rolle in der Weiterentwicklung des Automobils zu tragen, muss das Design mit genügend Weitsicht und Zukunftsorientierung durchgeführt werden. Hierbei spielen nicht nur technische, sondern vor allem auch wirtschaftliche und gesellschaftliche Trends eine Rolle.

Um neue Technologien tatsächlich in der Realität zu etablieren, ist es wichtig die Wirtschaftlichkeit zu betrachten. Im Folgenden werden die wichtigsten wirtschaftlichen Trends neuer Kommunikationsarchitekturen beschrieben:

- *Kostenreduktion Materialkosten:* Um Kosten einzusparen, schlägt Broy (2006) vor die Anzahl der Bussysteme und Leitungen zu reduzieren. Außerdem sollte die Anzahl der ECUs stark reduziert und nicht ausgelastete ECUs von anderen Softwarekomponenten mitbenutzt werden. Ergänzend schlägt die Studie der fortiss GmbH (2011) vor, mechanische und hydraulische Komponenten durch elektronische zu ersetzen (z. B. "X-by-Wire"), um teure Komponenten zu eliminieren.

- *Reduktion der Komplexität:* Durch Strukturgebende Architekturen und Modellbildung können Entwicklungs- und Integrationskosten gespart werden. Broy (2006) wünscht sich für die Automobilindustrie den Trend, weg von Software Engineering einzelner Komponenten und hin zum Systems Engineering, da so die Komplexität stark verringert werden kann.

- *Verteilung der Arbeit:* Pretschner u. a. (2007) betont die Wichtigkeit der Supply Chain für die großen Automobilkonzerne. Die Fertigung moderner Automobile geschieht nicht allein in der Hand des Original Equipment Manufacturer (OEMs). Die Teilkomponenten

[1] AUTomotive Open System ARchitecture (AUTOSAR) ist eine Organisation mit dem Ziel, offene Standards für Steuergeräte und Softwarearchitekturen im Automobil zu schaffen und zu etablieren. Sie besteht aus den größten Partnern der Automobilindustrie .

werden von einer Vielzahl an Zulieferern entwickelt. Daher muss es auch in zukünftigen Architekturen eine Weg geben, Komponenten von Drittanbietern effizient einzubinden.

Neben den wirtschaftlichen Faktoren, haben auch die gesellschaftlichen Trends einen Einfluss auf die Funktionen des Autos. Diese Trends werden im Folgenden beschrieben:

- *Personalisierung:* Laut der Studie der fortiss GmbH (2011), verliert das Autofahren als Funktion an Bedeutung, während der eigentliche Mehrwert von der Anpassung eines Autos an einen größeren Kontext herrührt. Passagiere wollen ökonomisch reisen und gleichzeitig Komfort und Annehmlichkeiten wie eine Verbindung zum Internet genießen. Außerdem wollen sie das Auto mit von ihnen ausgewählten Diensten personalisieren und es mit ihren Geräten verbinden.

- *Unfallfreiheit:* Gefordert werden laut fortiss GmbH (2011) vor allem aktive Sicherheitsfunktionen, welche den Straßenverkehr sicherer machen und die Fahrzeuginsassen schützen.

- *Wirtschaftlichkeit:* Die fortiss GmbH (2011) betont, dass nicht nur das Preis-Leistungs-Verhältnis, sondern auch die Reichweite eine große Rolle beim Kauf eines Fahrzeugs spielen. Hier ist eine Reduktion des Fahrzeuggewichts hilfreich, z.B. durch austauschen mechanischer und hydraulischer Komponenten durch elektrische oder durch einsparen von Leitungen, etc..

Als ein Vertreter der Industrie malt AUTOSAR die in Abbildung 3.1 dargestellte Zukunftsvision. Fürst und Bechter (2016) beschreiben das Auto der Zukunft als ein mit allem verbundenes und autonomes Fahrzeug. Schon heute haben viele Automobile eine Internetverbindung oder können sich mit Smartphones verbinden. Bald wird das Auto allerdings mit seiner kompletten Umgebung interagieren, von Smart Homes über andere Fahrzeuge bis hin zu Road Side Infrastructure. In diesem Auto der Zukunft gibt es viele technologische Aufgaben und Herausforderungen für Kommunikationsarchitekturen. Diese werden im Folgenden beschrieben:

- *Anzahl an Funktionen:* Die Anzahl der in Software realisierten Funktionen steigt stetig an und wird sich laut fortiss GmbH (2011), mit der Einführung von Fahrassistenzsystemen und autonom fahrenden Autos noch einmal rasant erhöhen.

- *Heterogene Anforderungen:* Eine zukünftige Kommunikationsarchitektur muss die heterogenen Anforderungen von Automobilsoftware, von Multimedia bis hin zur Motorsteuerung, erfüllen. Dieses Problem wird derzeit durch verschiedene Bussysteme gelöst. (vgl. Pretschner u. a. (2007))

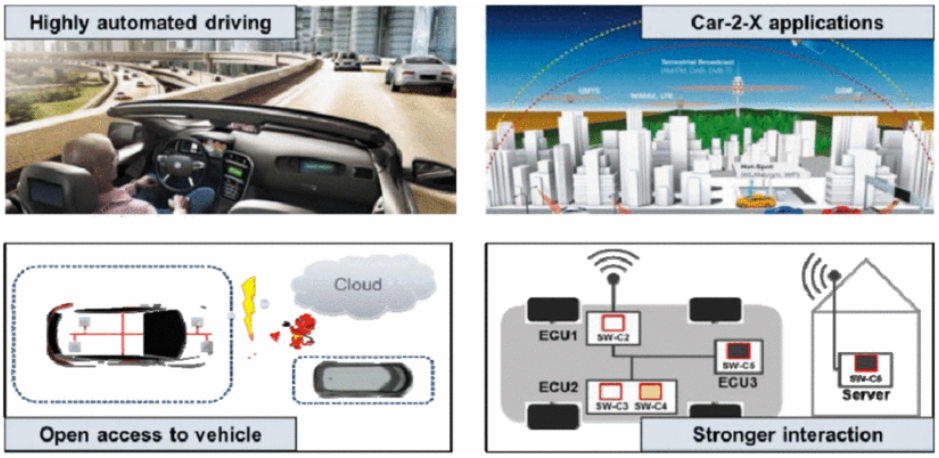


Abbildung 3.1: Herausforderungen im Auto der Zukunft nach Fürst und Bechter (2016).

- *Steigender Kommunikationsbedarf:* Der Bandbreitenbedarf der Softwarekomponenten im Auto steigt mit der Anzahl an Kameras und hochauflösenden Sensoren stark an. Gleichzeitig steigt der Grad der Vernetzung der Subsysteme durch Fahrassistenzfunktionen, die Zugriff auf die Sensoren und Aktoren aus verschiedenen Subsystemen brauchen.

- *Zentralisierung des Kommunikationsmediums:* Die Starke Verteilung der Software im Auto wird, kombiniert mit der starken Vernetzung durch hohe Abstraktionsebenen, mit der Zeit ein Problem für die Vielzahl an Bussystemen. Dies Betrifft laut Pretschner u. a. (2007) nicht nur die Kosten und den Integrationsaufwand, sondern auch die Zeitgarantien. Falls ein Sensor seine Daten an Komponenten in verschiedenen Bussystemen verteilen muss, können Zeitabweichungen nicht mehr in der jetzigen Form garantiert werden. Dieses Garantien können nur in einer zentralen Kommunikationsarchitektur gegeben werden, wo alle Empfänger der Daten in einem Netzwerk sind.

- *Plug & Play und Wiederverwendbarkeit:* Um in Zukunft Softwarekomponenten auf Nachfrage hinzufügen oder aktualisieren zu können, muss ein gewisses Maß an *Plug & Play* verfügbar sein. Hierfür müssen sowohl Hardware Komponenten als auch die Laufzeitplattformen und Kommunikationsmedien standardisiert und abstrahiert sein. Dies führt auch dazu, dass Komponenten häufig und lange wiederverwendet werden können.

- *Varianten und Konfigurationen:* Für den Weltmarkt müssen Automodelle in verschiedenen Varianten und Konfigurationen an die verschiedenen Anforderungen der Kunden angepasst werden können. Genauso muss es möglich sein, die Software im Auto mit Updates zu versorgen, um Fehler auszumerzen oder Funktionen hinzuzufügen. So kann

es sein das “alte” Softwarekomponenten mit “neuen” zusammenarbeiten müssen. Diese Flexibilität, Anpassungsfähigkeit und Vielfalt ist eine große Herausforderung und erschwert den Nachweis der korrekten Funktionsfähigkeit durch Tests. Bei einem Update einer Komponente muss sichergestellt werden, dass alle anderen Komponenten im Auto damit kompatibel sind. (vgl. Pretschner u. a. (2007))

- *IoT:* Um die von Fürst und Bechter (2016) beschriebenen Zukunftsszenarien von Connected Cars (siehe Kapitel 2.5) bis hin zu autonomen Fahren zu realisieren, wird es erforderlich, das Auto mit dem IoT zu verbinden. Schon heute ermöglichen viele Autohersteller die Fernsteuerung verschiedener Funktionen, wie dem Auffinden des Autos oder dem Auf- und Zuschließen. Außerdem werden laut Datta u. a. (2016) Autos in Smart City Anwendung als Teil des IoT zu fahrenden Sensorknoten. So wird eine zukünftige Kommunikationsarchitektur in Automobilen laut fortiss GmbH (2011) eine Verzahnung von Konzepten aus sicherheitskritischen eingebetteten Systemen und Internet-Technologie realisieren. Sie muss für den rasanten Fortschritt in den Gebieten Cloud-Computing und für neue Netzwerkstandards wie 5G Mobilfunknetze gewappnet sein (vgl. Gupta und Jha (2015)).

- *Sicherheit:* Die Öffnung des Autonetzwerks erfordert verschiedene Sicherheitsmaßnamen um Angreifer fernzuhalten, die bisher im abgeschlossenen System des Autos nicht in dem Umfang nötig waren.

3.2 Probleme der heutigen Kommunikationsarchitektur in Automobilen

Laut Pretschner u. a. (2007) ist das größte Problem der heutigen ICT Architekturen, dass sie sehr stark spezialisiert ist. Dies begründet er mit der Kostenoptimierung für Hardware, die immer zu spezialisierten Systemen führt, die nicht für andere Komponenten wiederverwendet werden können. Ein solches System ist schwer wartbar und noch schwerer zu aktualisieren, da es keine Kontrolle gibt, mit welcher Version einer Komponente kommuniziert wird.

Die Studie der fortiss GmbH (2011) betont, dass die jetzige Architektur aus historischen Gründen (siehe oben) so komplex geworden ist, dass sie die Innovationen eher aufhält, als nach vorne zu treiben. Eine neue Architektur würde neue Ansätze und Funktionen ermöglichen und fördern. Die Integration neuer Funktionen, wie aktiver Sicherheitsfunktionen und Steer-by-Wire wird aufgrund der hohen Anforderungen an die ICT sehr vorsichtig behandelt. Auch

im Infotainment Bereich kann die derzeitige Technologie nicht mit den Standards außerhalb des Autos mithalten. Dies begründet die Studie mit den folgenden Aspekten:

- Vielzahl an Steuergeräte, die viel zu Spezialisiert sind.
- Hochgradig komplexer Kabelbaum mit zu vielen verschiedenen Bussystemen und Kommunikationsprotokollen (zum Beispiel für Motorraum, Fahrgestell, Fahrgastraum und Infotainment durch LIN, CAN, FlexRay und MOST).
- Unterstützt keinerlei Dynamik und Konfiguration nach dem Verkauf.
- Architektur wird durch Autohersteller und Zulieferer geformt, nicht durch die Funktion, die sie erfüllen soll.

Die Vielzahl an Jahren, in denen niemand eine Änderung an der ICT durchführen wollten haben laut fortiss GmbH (2011) zu vielen unschönen Effekten geführt und dafür gesorgt, dass sie den Herausforderungen der Zukunft nicht mehr gewachsen ist. Abbildung 3.2 zeigt, Architekturen werden komplexer als nötig und verhindern Innovation. Dies kann nur durch eine neue Architektur gelöst werden. Wie auch schon bei der Einführung des CAN Busses 1980 durch den es aufgrund der Virtualisierung der Verbindung zwischen Geräten über ein Kabel einfacher wurde neue Funktionen hinzuzufügen. Heutige Architekturen haben ein ähnliches Problem: Die große Anzahl stark vernetzter Steuergeräte.

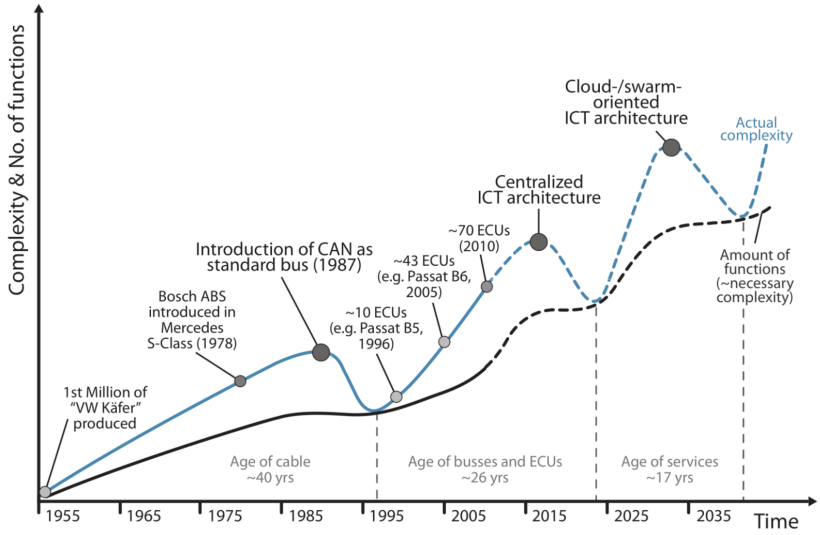


Abbildung 3.2: Entwicklung der Komplexität ICT Architektur und der Anzahl der Funktionen im Auto, nach der Studie der fortiss GmbH (2011).

3.3 Ansätze neuer Kommunikationsarchitekturen in Automobilen

Um eine neue Kommunikationsarchitektur zu entwickeln, sind Änderungen an verschiedenen Stellen in der ICT des Automobils erforderlich. Diese lassen sich in Hardware, Software und Kommunikationstechnologien unterteilen. Nur wenn Änderungen auf all diesen Gebieten gleichzeitig erfolgen, können neue Kommunikationsarchitekturen etabliert werden.

Im Folgenden sind die zentralen Aspekte der Hardwareveränderungen beschrieben:

- *Verwendung von Standardkomponenten:* Statt spezialisierter Komponenten sollten laut Broy (2006) und Pretschner u. a. (2007) Standardkomponenten eingesetzt werden, um Updates und Mehrfachnutzung zu gewährleisten.
- *Hochintegrierte Mechatronische Komponenten:* Laut fortiss GmbH (2011) müssen Sensoren und Aktoren intelligenter werden und universell einsetzbar sein.
- *Zentralisierung der Rechner:* Die Steuergeräte im Auto werden zu skalierbaren Multi-Core Prozessoren, die nach Burgio u. a. (2016) mehrere Software Komponenten ausführen. Gleichzeitig werden laut fortiss GmbH (2011) komplexe Aufgaben, wie das autonome Fahren, auf Großrechnern mit Ähnlichkeit zu Server Technologien ausgeführt.
- *Leistungsfähiges Kommunikationsmedium:* Die Bussysteme im Auto internen Netzwerk werden durch ein zentrales Medium mit hoher Bandbreite ersetzt, laut fortiss GmbH (2011) und Bello (2011) wahrscheinlich durch Automotive Ethernet Technologien. Um über Internet mit anderen Anwendungen zu interagieren, werden neue Standards für mobile Netzwerke geschaffen. Mit dem 5G Netz wird laut Gupta und Jha (2015) bereits der erste Schritt getan. Dieses bringt Reformen für bewegliche Knoten im Mobilen Netz (*Moving Networks*) und M2M Kommunikation im IoT (*Massive Machine Communications*).

Neben den Änderungen in der Hardware Technologie, werden auch bei der Entwicklung von automobilen Anwendungen verschiedene Konzepte eingesetzt werden:

- *Modularisierung:* Das entscheidende Konzept im Auto, die Abkapselung von intelligenten Modulen, bleibt laut der fortiss GmbH (2011) erhalten. Allerdings beschreibt Kugele u. a. (2017) wie sich der Entwurfsprozess von Funktionen ändert. Diese werden nicht mehr als einzelne ECUs im Netzwerk sondern als Dienste in einer SOA (siehe unten) umgesetzt. Die Studie der fortiss GmbH (2011) betont die dadurch entstehende Abstraktion der physischen Interaktionen zur logischen.

- *Virtualisierung:* Die Umsetzung der Software als Dienste erlaubt die Virtualisierung in getrennten Laufzeitumgebungen. So können laut fortiss GmbH (2011) Dienste auch wenn sie nicht mehr auf getrennten Steuergeräten ausgeführt werden, trotzdem in getrennten Umgebungen ausgeführt werden. Dies gibt, neben neuen Möglichkeiten für Sicherheitskonzepte, vor allem den Zulieferern die Möglichkeit ihre Geheimnisse zu bewahren.

- *Miniaturisierung:* Um Dienste auf Wiederverwendbarkeit zu optimieren, hilft es laut fortiss GmbH (2011) diese in möglichst kleine Aufgabenbereiche aufzuteilen und zu intelligenten Modulen zusammenzusetzen.

Im Folgenden sind die verschiedenen Ansätze zur Kommunikationsarchitektur beschrieben:

- *Daten zentriert:* Der Fokus liegt laut der Studie der fortiss GmbH (2011) auf der effizienten Verbreitung von Daten im Netzwerk. In einem Daten zentrierten System (siehe Abbildung 3.3), produzieren Sensoren Daten und bieten diese an. Anschließend werden sie dann von Logischen Diensten verarbeitet und schließlich von Aktoren konsumiert.

- *Heterogene Anforderungen:* Es muss im Autonetz sowohl Echtzeit basierte Kommunikation (siehe Bello (2014) und Echtzeiterweiterungen für Ethernet in Kapitel 2.1) als auch Kommunikation von großen Datenmengen möglich sein. Des Weiteren müssen laut fortiss GmbH (2011), um mit Internetdiensten kommunizieren zu können, die etablierten Webstandards unterstützt werden. Außerdem werden Dienste auf verschiedene Arten miteinander interagieren: Entfernte Funktionsaufrufe zum Ausführen von Aktionen, das Abonnieren von Sensordaten zur effizienten Verbreitung, aber auch das Senden von Nachrichten von einem Dienst zum anderen, müssen realisiert werden.

- *dienstorientiert:* Laut der Studie der fortiss GmbH (2011) ist die Grundlage zukünftiger Kommunikationsarchitekturen eine SOA (siehe Kapitel 2.3). So können standardisierte versionsübergreifende Schnittstellen, sowie das Plug & Play Konzept realisiert werden. Auch Kugele u. a. (2017) untersucht in seiner Arbeit die Anwendung einer SOA im Auto und hält dieses Konzept für gut geeignet, da die Black Box Implementierung von Diensten mit festen Schnittstellen eine Langzeitnutzung und Wiederverwendung ermöglicht. Softwarekomponenten können ausgetauscht oder verändert werden, solange die Funktion und die Schnittstelle gleichbleiben (keine Funktionen dürfen wegfallen, nur neue hinzukommen).

- *Metadaten Unterstützung:* Sowohl Kugele u. a. (2017) als auch die Studie der fortiss GmbH (2011) kommen zu dem Ergebnis, das eine SOA im Auto nur mithilfe von zusätzlichen

Metadaten realisiert werden kann. Diese erweitern die Schnittstellenbeschreibung eines Dienstes um zusätzliche Informationen. Basierend auf diesen Daten, können Klienten des Dienstes beim Aufbau der Verbindung einen Vertrag über verschiedene Verbindungseigenschaften vereinbaren. So können die Anforderungen des Anbieters sowie des Klienten optimal bedient werden.

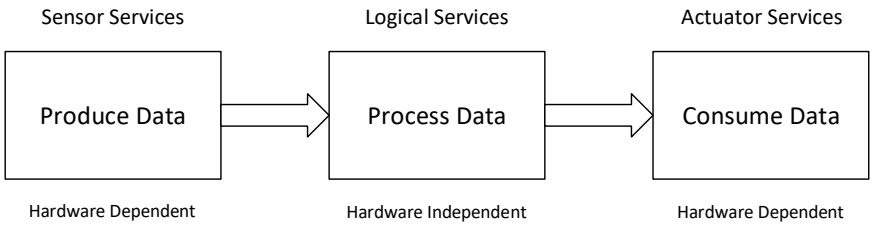


Abbildung 3.3: Ablauf eines Daten zentrierten Systems.

Laut der Studie der fortiss GmbH (2011) kann eine neue zentralisierte ICT Architektur nur mithilfe einer gemeinsamen dienstorientierten Middleware-Basis (siehe Kapitel 2.2) die Komplexität reduzieren. Diese Middleware muss, neben den oben beschriebenen Kommunikationsanforderungen, auch die folgenden Konzepte umsetzen:

- *Einheitlichkeit:* Datenzugriffe, Entfernte Funktionsaufrufe und Schnittstellen sollten, so Kugele u. a. (2017), einheitlich gestaltet sein. Dies ermöglicht ein intuitives Benutzen und verhindert unnötige Heterogenität.

- *Portierbarkeit:* Um eine gute Performance zu gewehrleisten, sollte die Middleware zwar auf die Hardware optimiert werden, laut Pretschner u. a. (2007) darf dies allerdings nicht die Portierbarkeit verhindern, da im Auto eine Vielzahl verschiedener Rechner eingesetzt werden wird.

- *Erweiterbarkeit:* Beim Entwurf muss auf die Zukunftssicherheit geachtet werden. Die Middleware muss also modular und erweiterbar sein, da sonst laut Pretschner u. a. (2007) und fortiss GmbH (2011) ein ähnliches Szenario wie jetzt bevorsteht.

- *Virtualisierung:* Der Zugriff auf andere Funktionen erfolgt laut fortiss GmbH (2011) nur über die Middleware. Die darunterliegende Hardware und Laufzeitumgebung sowie die verwendeten Protokolle und Kommunikationsmedien werden vor der Anwendung versteckt.

- *Nicht funktionale Eigenschaften:* Um diese Abstraktion zu ermöglichen, muss die Middleware auch nicht-funktionale-Eigenschaften, wie Fehlertoleranz und verschiedene Sicherheitsmechanismen umsetzen. So können laut fortiss GmbH (2011) Funktionen auch außerhalb des Autos realisiert werden.

Mit Hilfe dieser Ansätze kann die ICT Architektur des Autos, wie in Abbildung 3.4 aus der Studie der fortiss GmbH (2011) dargestellt, in 3 Schritten angepasst werden. Der erste Schritt sind hochintegrierte Sensor und Aktor Systeme, die auf einem hohen Level gekapselt werden. Diese Aufgabe ist in heutigen Automobilen bereits nahezu erfüllt. Im zweiten und aktuellen Schritt, wird die ICT Architektur umstrukturiert. Hierfür wird ein zentrales Netzwerk mit einer gemeinsamen Middleware-Basis, wobei die Subnetze im Auto erhalten bleiben und durch die Middleware verknüpft werden. In der Zukunft werden dann in einem dritten Schritt, die komplette das Fahrzeug betreffende Software, mit einer Middleware miteinander verbunden. Hierbei ist es egal, ob die Funktionen relevant für das Fahren sind, ob sie Sicherheitskritische Funktionen bedienen, oder sogar außerhalb des Fahrzeugnetzes in der Cloud angesiedelt sind. Dies erlaubt Anpassungen des Autos je nach dem Belieben des Fahrers auch durch Dritthersteller.

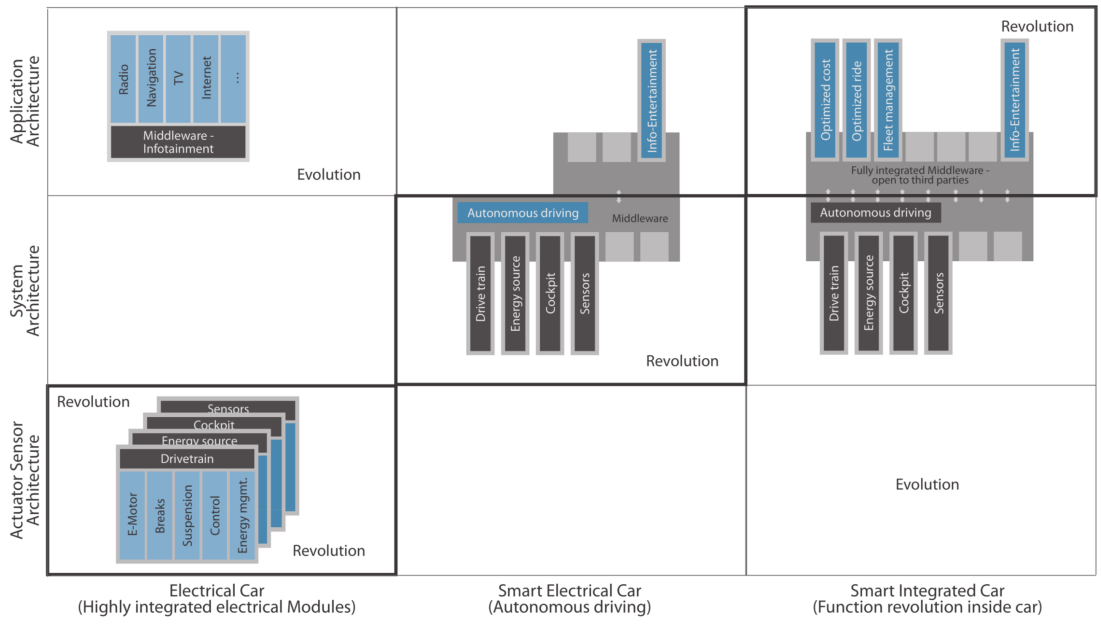


Abbildung 3.4: Evolution der ICT Architektur in Automobilen in drei Stufen, aus der Studie der fortiss GmbH (2011).

3.4 Problemstellung und Zielsetzung

In dieser Arbeit soll ein Konzept entwickelt werden, wie der oben beschriebenen zweite und dritte Schritt der Anpassung der Kommunikationsarchitektur zu lösen ist. Hierfür wird in einer Machbarkeitsstudie eine Middleware entwickelt, die die beschriebenen Ansätze vereint. Insbesondere soll die Frage geklärt werden, wie eine datenzentrierte, dienstorientierte Architektur für Dienste mit heterogenen Anforderungen im Auto aussehen muss, um den Anforderungen zu entsprechen. Hierfür sollen die verschiedenen beschriebenen Middleware Konzepte umgesetzt werden. Der Fokus liegt dabei auf der Kommunikation zwischen Diensten aus verschiedenen Anforderungskategorien, wofür ein Konzept zur Dienstgütevereinbarung entwickelt wird. Um den Umfang einer Masterarbeit nicht zu sprengen, werden an verschiedenen Stellen Vereinfachungen vorgenommen, entweder da diese Aspekte wohlbekannte und viel genutzte Konzepte sind oder da sie ein komplett neues Thema öffnen. Aus diesem Grund werden die Themenbereiche Virtualisierung von Diensten und die verschiedenen Sicherheitsmechanismen nicht behandelt.

4 Konzepte

Um die dargestellten Probleme der Kommunikationsarchitekturen zukünftiger Automobile zu lösen, werden in den folgenden Kapiteln verschiedene Konzepte und verwandte Arbeiten verglichen und vorgestellt. Dafür werden zuerst Automobile Dienste ihren Anforderungen entsprechend in Klassen eingeordnet. Anschließend werden verschiedene existierende Service-Orientierte Middleware-Lösungen verglichen und ermittelt welche Bestandteile für diese Anforderungsklassen geeignet sind. Ein weiterer wichtige Aufgabe bei der Einführung einer neuen Kommunikationsarchitektur ist, wie man von der alten auf eine neue Architektur wechseln kann. Dafür werden verschiedene Migrationsstrategien beschrieben.

4.1 Klassifizierung von Automobilen Diensten

Da Dienste verschiedene Anforderungen an die Kommunikation haben, muss eine Zukünftige Zentralisierte Middleware Lösung alle diese Anforderungen unterstützen (vgl. Buckl u. a. (2012)). Um die Komplexität nicht explodieren zu lassen, ist es daher sinnvoll die Dienste in verschiedene Klassen aufzuteilen, die dann von der Middleware einheitlich behandelt werden können. So können Dienstanbieter vermerken, welche Klassen von Diensten sie unterstützen oder als Konsumenten akzeptieren. Im Folgenden werden zunächst verwandte Arbeiten beschrieben, die sich mit Kriterien beschäftigen, mit deren Hilfe, Dienste in Kategorien unterteilt werden können. Anschließend werden auf dieser Basis die in dieser Arbeit ausgewählten Kriterien hergeleitet und abschließend drei Dienstklassen aufgestellt.

4.1.1 Verwandte Arbeiten

Es gibt bereits verschiedene Ansätze um Automobil-Software in Kategorien einzuteilen, Broy (2008) und Pretschner u. a. (2007) stellen dies heraus und ordnen die Software Komponenten innerhalb des Autos verschiedenen Domänen zu.

- *Multimedia:* Fasst alle Komponenten des Infotainment[1]-Systems, sowie Multimedia, Telematiks und Human-Machine Interface (HMI) Software zusammen.
- *Passenger/Comfort:* Bezieht sich auf Komponenten, die den Komfort der Passagiere betreffen und das Auto personalisieren. Wie z.B. Klimaanlagen oder automatische Sitzeinstellung.
- *Safety Electronics:* Umfasst Sicherheitsrelevante ECUs, wie das ABS oder Airbags.
- *Engine/ Drive Train:* Vereint Komponenten, die Funktionen des Fahrwerks realisieren und ohne die das Fahren nicht möglich wäre. Wie z.B. Motorsteuerung, Steer-by-wire Systeme oder das Akkumanagement in elektrischen Fahrzeugen.
- *Diagnostics:* Bezieht sich auf Diagnose und Infrastruktur Komponenten.

Eine ähnliche Unterteilung nehmen Schäuffele und Zurawka (2012) in ihrem Buch *Automotive Software Engineering* vor und beziehen außerdem die konkrete Architektur von Fahrzeugnetzen und Bussystemen mit ein. So unterteilen sie die Komponenten des Autos in Subsysteme, die in Abbildung 4.1 dargestellt sind. Sie betonen aber, dass die Einteilung in Subsysteme bereits heutzutage verschwimmt, da es immer mehr Subsystem übergreifende Funktionen gibt, wie z.B. die Antriebsschlupfregelung, welche sowohl im Subsystem *Fahrwerk*, als auch im Subsystem *Antriebsstrang* benötigt wird. Moderne Fahrassistenzsysteme und das autonome Fahren verstärken diesen Effekt (Schäuffele und Zurawka, 2012, Seite 6-7).

Wie bereits in Kapitel 3.3 beschrieben, setzen viele Vertreter der Automobilindustrie und der Forschung auf ein zentralisiertes Dienst basiertes Kommunikationsnetz, um die Aufgaben der Zukunft meistern zu können. Um einem solchen Netz Struktur zu geben, schlagen Jobst und Prehofer (2016) eine hierarchische Architektur vor, welche in Abbildung 4.2 dargestellt ist. In dieser Architektur werden alle Dienste in einem Baum als Hierarchie aufgebaut. So kann eingeschränkt werden, wer mit wem kommuniziert, da den Diensten beispielsweise nur erlaubt wird parallel und senkrecht, jedoch nicht diagonal zu kommunizieren. Des Weiteren kann man steuern, ob über mehrere Ebenen hinweg kommuniziert werden darf oder nicht. So können schwächere Komponenten vor stärkeren geschützt und sicherheitsrelevante Teilbäume von nicht sicherheitsrelevanten getrennt werden.

Neben dem Konzept der eingeschränkten Kommunikation und Anordnung in einer hierarchischen Architektur, beschreiben Jobst und Prehofer (2016) außerdem, verschiedene Gesichtspunkte, nach denen die Dienste angeordnet werden können (siehe Abbildung 4.3) und betonen

[1] Infotainment (zusammengesetzt aus dem englischen information und entertainment) fasst alle Anwendungen und Geräte zusammen, die zur Information oder Unterhaltung der Fahrgäste verbaut sind.

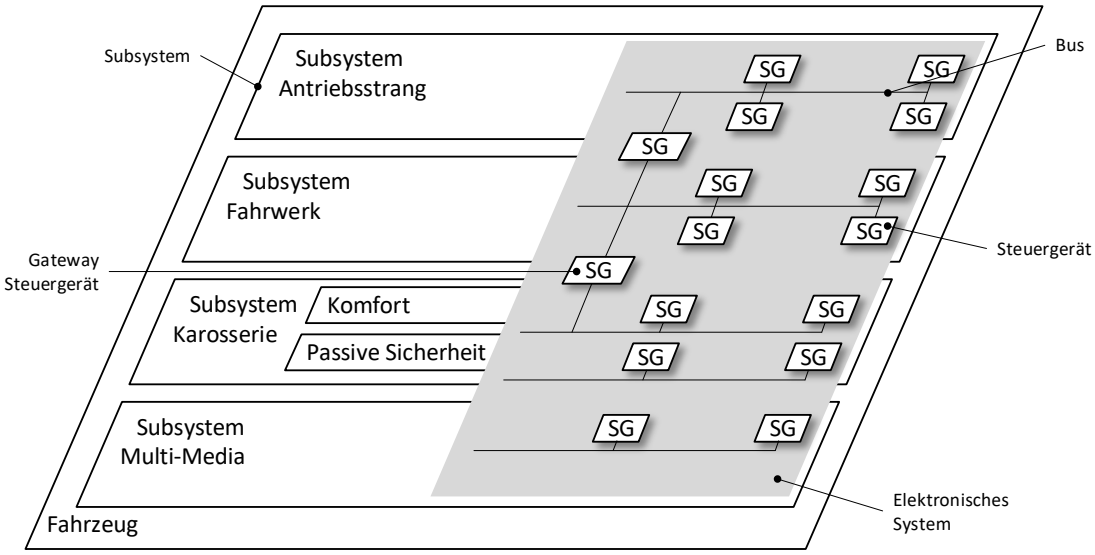


Abbildung 4.1: Zuordnung von Funktionen zu den Subsystemen des Fahrzeugs (Schäuffele und Zurawka, 2012, Seite 6).

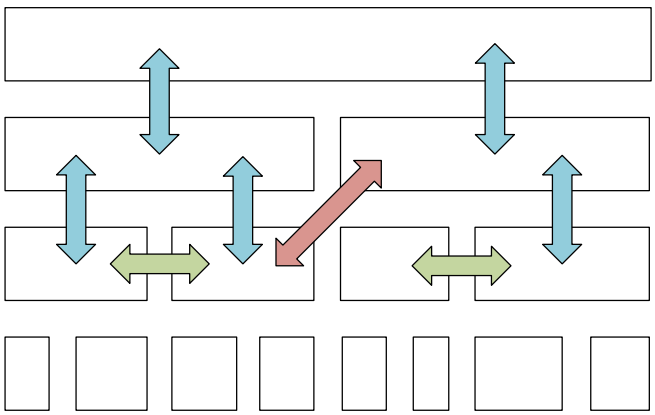
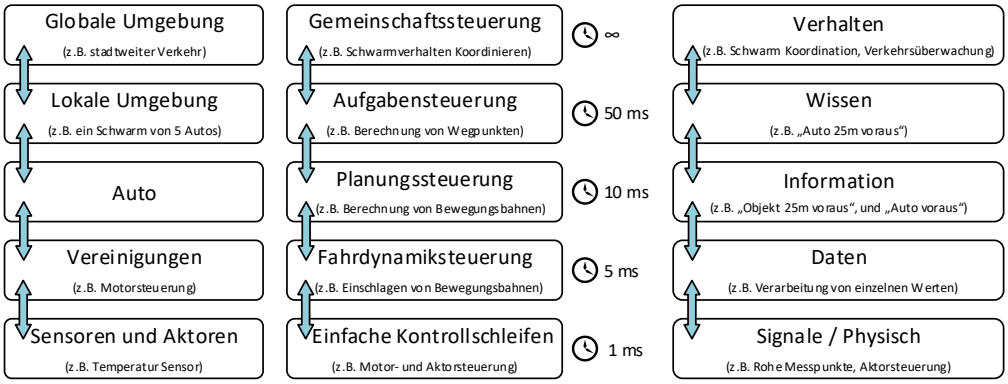


Abbildung 4.2: Hierarchische Architektur zur Strukturierung von Diensten nach Jobst und Prehofer (2016). Die Pfeile markieren beispielhaft die möglichen Kommunikationswege: Blau senkrecht Ebenenübergreifend, Grün horizontale Teilbaumwechselnd und Rot diagonale sowohl Ebenenübergreifend als auch Teilbaumwechselnd.

die Wichtigkeit der Wahl der Anordnungsmethodik. Das erste Konzept (4.3a) ordnet Dienste nach räumlichen Gesichtspunkten unter Vergrößerung der Umgebung von unten nach oben; Das Zweite (4.3b) bezieht sich auf die Vergrößerung der Ausführungszyklen; Und das Dritte (4.3c) ordnet die Dienste nach Abstraktionslevel mit größer werdenden Verantwortungsbereichen von unten nach oben.



(a) Räumlich (b) Zeitlich (c) Abstraktionsebene

Abbildung 4.3: Vergleich verschiedener Gesichtspunkte zum Anordnen von Automobilsystemen in Ebenen nach Jobst und Prehofer (2016).

4.1.2 Kriterien

Basierend auf den oben genannten Arbeiten, können 5 wichtige Kriterien zur Klassifizierung von Diensten genannt werden, die starke Auswirkungen auf die Kommunikationsfähigkeiten von Diensten haben. Diese werden im Folgenden beschrieben.

Die *Software Domäne* (vgl. Broy (2008); Schäuffele und Zurawka (2012)) beschreibt Gruppen von Software Komponenten mit ähnlichen Anforderungen und ist daher ein zentrales Kriterium zur Klassifizierung von Diensten. Auf dieser Basis können Informationen über Zeitanforderungen oder die Art der Daten ermittelt werden. In dieser Arbeit werden die bereits beschriebenen Software Domänen nach Broy (2008) verwendet, die sich laut Pretschner u. a. (2007) in ihren Anforderungen nach Echtzeit, Datenkomplexität und Kommunikationsmustern unterscheiden:

- *Multimedia:* Weiche Echtzeitanforderungen, vor allem QoS-Anforderungen für Nutzerzufriedenheit; Komplexe und große Datenpakete; Diskrete eventbasierte Kommunikation und Datenverarbeitung von Streams.

- *Passenger/Comfort:* Weiche Echtzeitanforderungen im Bereich von 100 bis 250 Millisekunden für Nutzerinteraktionen; Diskrete eventbasierte Kommunikation und Kontrollschleifen.
- *Safety Electronics:* Harte Echtzeitanforderungen im unteren Millisekunden Bereich; Niedrig komplexe Daten mit einzelnen Werten; Diskrete eventbasierte Kommunikation mit strikten Sicherheitsanforderungen.
- *Engine/ Drive Train:* Harte Echtzeitanforderungen im Micro- bis Millisekunden Bereich; Niedrig komplexe Daten mit einzelnen Werten; Kontrollschleifen dominieren über diskreter eventbasierter Kommunikation, hat strikte Anforderungen and die Verfügbarkeit.
- *Diagnostics:* Weiche und Harte Zeitanforderungen; Diskrete eventbasierte Kommunikation.

Die *Abstraktionsebene* (vgl. Jobst und Prehofer (2016)) beschreibt, welchem Verantwortungsbereich ein Dienst zugeordnet wird. Der Fokus liegt dabei auf der Funktionalität und den Informationen, die zur Verfügung gestellt werden. Dies ist ein wichtiges Kriterium für die Kommunikation, da es Aufschluss darüber gibt welche Art von Daten übertragen werden müssen. Es werden folgenden Abstraktionsebenen nach Jobst und Prehofer (2016) unterschieden:

- *Verhalten:* Höchste Abstraktionsebene, verarbeitet Wissen um daraus Entscheidungen zu treffen, die das Verhalten des Fahrzeuges steuern.
- *Wissen:* Kombinieren von Informationen, um einen Wissenszustand als Grundlage für Verhaltensentscheidungen bereit zu stellen.
- *Information:* Bereitstellung von einzelnen Informationen die aus der Verarbeitung von Daten gewonnen werden.
- *Daten:* Verarbeitet und Aggregation von einzelnen Messwerten.
- *Signale:* Niedrigste Abstraktionsebene, arbeitet mit rohen Messdaten und Werten zur Aktor Steuerung.

Der *Standort* (vgl. Jobst und Prehofer (2016) siehe Abbildung 4.3a) beschreibt, wo (geographisch) innerhalb des Kommunikationsnetzes ein Dienst angesiedelt ist. Dies kann Aufschluss über die verfügbaren Kommunikationsprotokolle geben. Durch die Öffnung des Autonetzes zum Internet und die Einführung von Cloud-Computing Ansätzen (siehe Kapitel 2.4) im Auto, wird der Standort eines Dienstes zu einer wichtigen Anforderungsgruppe für Dienste. In diesem Projekt werden die folgenden Standorte nach Jobst und Prehofer (2016) unterschieden,

allerdings ohne eine genauere Aufspaltung der fahrzeuginternen Dienste, da diese bereits über andere Kriterien abgedeckt sind.

- *Globale Umgebung:* Cloud Dienste, die an einem beliebigen (unbekannten) Ort im Internet ausgeführt werden (siehe Kapitel 2.4); Einsatz von Standard Internet Protokollen möglich, keine Speziellen Protokolle unterstützt; Unbekannte Latenzen, die meist im zwei- bis dreistelligen Millisekunden Bereich liegen.

- *Lokale Umgebung:* Dienste in Local-Adhoc Netzwerken, z.B. *Car-to-Car* Kommunikation (siehe Kapitel 2.5); Einsatz von Standard Internet Protokollen und Speziellen Protokolle möglich; Keine garantierten Latenzen, allerdings ungefähr im zweistelligen Millisekunden Bereich.

- *Auto intern:* Dienste im Auto; Einsatz von Standard Internet Protokollen und Speziellen Protokolle möglich; Latenzen können vor der Ausführung bestimmt werden und liegen im unteren Millisekunden Bereich.

Der *Ausführungszyklus* (vgl. Jobst und Prehofer (2016) siehe Abbildung 4.3b) beschreibt, die Zeit, die bis zur nächsten Ausführung vergeht. Diese steht in direktem Zusammenhang, mit der gestatteten Übertragungszeit von Nachrichten, da z.B. Sensorinformationen immer zum richtigen Zeitpunkt vorliegen müssen. Außerdem kann Anhand der Ausführungszeiten abgelesen werden, ob diese Dienste überhaupt dynamisch realisiert werden können. Bei starker Anfälligkeit für Jitter und Latenzen (siehe Kapitel 2.1), ist eine statische Kommunikation mit exakt definierten Abläufen der einzige Weg um Echtzeit zu garantieren. Diese Ausführungszyklen sind in die folgenden Gruppen unterteilt:

- *Gemeinschaftsverhalten:* Keine Zeitbeschränkung, dynamisch ausführbar, bei einem Ausfall bleibt das Auto funktionsfähig.

- *Aufgabenermittlung:* Weiche Deadline unter $50ms$, keine Anfälligkeit für Jitter, dynamisch ausführbar, bei nicht Einhalten der Deadline bleibt das Auto funktionsfähig (auch Autonom gibt es einen Safestate[2]).

- *Fahrtplanung:* Harte Deadline unter $10ms$, nicht Jitter anfällig, dynamisch ausführbar, bei nicht Einhalten nur noch bedingt funktionsfähig (Safestate nur manuell möglich).

[2] Safestate (deutsch: Sicherer Zustand), beschreibt beim Auto einen Zustand, in dem trotz einer Fehlfunktion keine Gefährdung der Insassen und Umgebung besteht, dieser besteht üblicherweise darin, an den Fahrbahnrand zu fahren und anzuhalten.

- *Fahrsteuerung:* Harte Deadline unter $5ms$, im Extremfall (z.B. Nachrichtenstau) Jitter anfällig, dynamisch ausführbar, jedoch bevorzugt statisch, bei nicht Einhalten ist das Auto nur noch bedingt funktionsfähig (Safestate erschwert manuell noch möglich).
- *Kontrollschleifen:* Harte Deadline unter $1ms$, stark Jitter anfällig, nicht dynamisch ausführbar, bei nicht Einhalten nicht mehr funktionsfähig (Safestate teilweise unmöglich).

Die *Leistungsfähigkeit der Umgebung* (vgl. Schäuffele und Zurawka (2012)) beschreibt, wie Leistungsfähig die Hardwarekomponenten sind, die den Dienst ausführen. Basierend auf diesem Kriterium können Kleinstgerät vor Überlastung geschützt werden. Die Leistungsfähigkeit teilt sich in folgende Gruppen:

- *Cloud-Infrastruktur:* Skalierende Umgebung von Großrechnern mit nahezu unbegrenzter Leistungsfähigkeit, sie unterstützen Internet basierte Protokolle und stellen ausreichend Rechenleistung für komplexe Operationen bereit.
- *PCs:* Betriebssystem basierte Rechner mit potenter Hardware, sie unterstützen Internet basierte, sowie lokale Protokolle und stellen ausreichend Rechenleistung für komplexe Operationen bereit.
- *Mikroprozessor basierte ECUs:* Kleinstrechner mit stark limitierter CPU Leistung und geringem Speicher, sie unterstützen die üblichen lokalen Kommunikationsschnittstellen, sind allerdings nicht in der Lage komplexe Operationen, wie Verschlüsselungen und Deserialisierungen mit annehmbarem Aufwand umzusetzen.

4.1.3 Klassifikation

Aus den beschriebenen Kriterien werden nun verschiedene Dienstklassen gebildet, die jeweils eine Anforderungsgruppe von Diensten an die Kommunikationsarchitektur repräsentieren. Die Merkmale und Kriterien können sich bei den Klassen in einigen Bereichen durchaus überschneiden, jedoch legt jede Klasse den Fokus auf einige einzigartige Kriterien. In dieser Arbeit haben wir uns für drei Dienstklassen für dynamisch integrierbare und nutzbare Softwarekomponenten entschieden. Außerdem haben wir eine statische Klasse erstellt, welche die aus verschiedenen Gründen nicht dynamisch nutzbaren Softwarekomponenten enthält. Tabelle 4.1 zeigt welche Gruppen der jeweiligen Kriterien in den Dienstklassen abgebildet werden. Im Folgenden werden die vier Klassen beschrieben, jedoch spielt die Statische Klasse in dieser Arbeit keine Rolle, da diese Softwarekomponenten nicht als Dienste abgebildet werden können und somit nicht über die Middleware bezogen werden können.

Tabelle 4.1: Dienstklassifikation in drei dynamische und eine statische Klasse unter Einordnung der Kriterien.

	Web Services	Standard IP Services	Real-Time Services	Static Communication
Domäne	Multimedia, Passenger/-Comfort, Diagnostics	alle	Safety Electronics, Engine/Drive Train, Diagnostics	Safety Electronics, Engine/Drive Train
Abstraktionsebene	Verhalten, Wissen, Information	alle	alle	Signale
Standort	alle	Lokale Umgebung, Auto intern	Auto intern	Auto intern
Ausführungszyklus	Gemeinschaftsverhalten, Aufgabenermittlung	Gemeinschaftsverhalten, Aufgabenvermittlung	Aufgabenermittlung, Fahrtplanung, Fahrsteuerung	Kontrollschleifen
Leistungsfähigkeit der Umgebung	Cloud-Infrastruktur, PCs	PCs, Mikroprozessor basierte ECU's	PCs, Mikroprozessor basierte ECU's	Mikroprozessor basierte ECU's

Die *Web Service (WS)* Klasse gruppiert alle Dienste, deren Fokus auf einer hohen Abstraktionsebene und globalem Angebot liegt. Hierbei ist es nicht wichtig, ob Deadlines eingehalten werden. Aufgrund des globalen Angebotes und der hohen Abstraktionsebene werden für diese Dienste vor allem PCs mit ausreichenden Kapazitäten und Cloud-Infrastruktur verwendet. Als WS werden vor allen Dingen Dienste der Domains Multimedia, Passenger/Comfort und Diagnostics realisiert. Allerdings vor allem alle Dienste, die Informationen ins Web teilen sollen. Für die Kommunikationsarchitektur ist es wichtig, diesen Diensten die üblichen Kommunikationsstandards des Internets zu bieten, wie z.B. Protokolle zur Übertragung und Serialisierung. Obwohl der Fokus auf Global genutzten Diensten liegt, können diese Dienste auch Auto intern oder in der Lokalen Umgebung verwendet werden.

Die Klasse der *Realtime Service (RTS)* legt den Fokus auf zeitkritische, Auto interne Kommunikation mit Harten Deadlines. Diese wird vor allen Dingen auf PCs und Mikroprozessor basierten ECUs ausgeführt. Die zentralen Domänen in dieser Klasse sind Safety Electronics und Engine/Drive Train, allerdings haben auch Multimediastreams harte Echtzeitanforderungen, die jedoch nicht dieselbe Priorität haben. Für die Realisierung in der Middleware müssen hierfür Ethernet basierte Echtzeitübertragungsprotokolle verwendet werden, die dynamische Kommunikation erlauben.

Die *Standard IP-based Service (SIPS)* Klasse enthält alle Dienste, welche nicht in den Klassen WS und RTS umgesetzt werden, bei denen der Fokus also weder auf Echtzeitfähigkeit, noch auf globaler Erreichbarkeit und Serialisierungsfunktionen liegt. Dementsprechend können Dienste aus allen Domänen in dieser Klasse umgesetzt werden. Um in der Lokalen Umgebung kommunizieren zu können, werden Standard Internet Protokolle zur Datenübertragung verwendet. Außerdem sollten diese Dienste aufgrund der niedrig komplexen Protokolle auf nahezu allen Geräten ausgeführt werden können.

Da verschiedene Signal basierte Kontrollschleifen aus den Domänen Safety Electronics und Engine/Drive Train sehr stark anfällig für Jitter sind und harte Übertragungsdeadlines im Millisekunden Bereich haben, können Nachrichten dieser Dienste nicht über dynamische Protokolle übertragen werden und müssen zu Statisch definierten Zeitpunkten gesendet und empfangen werden. Hierfür können die bisher genutzten Bussysteme wie der CAN-Bus verwendet werden, jedoch auch Time Division Multiple Access (TDMA)-basierter Verkehr über Ethernet. Auch wenn diese Dienste nicht von der Middleware dynamisch angeboten werden, können sie über ein Gateway nach außen als dynamische Dienste angeboten und angesprochen werden.

4.2 Vergleich existierender Service-Orientierter Middleware-Lösungen

Um die verschiedenen Dienste im Auto kommunizieren lassen zu können, muss eine SOA basierte Middleware (siehe Kapitel 2.2) gefunden werden, die den Anforderungen der verschiedenen Dienstklassen (siehe Kapitel 4.1) entspricht. Die in den Grundlagen beschriebenen bekannten Vertreter der SOAs und Middlewares genügen leider nicht den Echtzeitanforderungen der autointernen Dienste und bringen meist einen großen Overhead an Serialisierung und Nachrichtengröße mit sich, wie Völker (2013) herausstellt. Daher muss eine andere Lösung gefunden werden, die eventuell verschiedene bekannte Technologien kombiniert. Im Folgenden werden zunächst verschiedene verwandte Arbeiten vorgestellt und bewertet. Anschließend wird der Vergleich ausgewertet und beschrieben, welche Konzepte für die Entwicklung der Middleware in dieser Arbeit verwendet werden.

4.2.1 Verwandte Arbeiten

In vielen verschiedenen Bereichen der Informatik wird an SOAs geforscht. Im Folgenden werden Konzepte und Entwicklungen aus Industrie und Forschung vorgestellt.

Die von der OSGi Alliance (2018) entwickelten Middleware Plattform Open Services Gateway initiative (OSGi) unterstützt Dienst basierte Kommunikation und bietet ein ausgefeiltes Lifecycle Management von Diensten. In Abbildung 4.4 ist der Aufbau des Java basierten Frameworks OSGi dargestellt. OSGi liefert seine eigene Ausführungsumgebung für Dienste, die auf der JVM und dem Betriebssystem aufbaut und die OSGi-Spezifischen Klassen und Methoden Definiert. Darüber liegen drei Schichten, die die Verwaltung der Dienste (in OSGi als Bundles definiert) übernehmen. Die *Modules* Ebene definiert, wie neue Bundles dynamisch importiert oder exportiert werden können. Der *Life Cycle* bietet eine API zum Installieren, Starten, Stoppen, Updaten und Deinstallieren von Bundles. Ganz oben im OSGi Framework liegt die *Services* Schicht, welche als Schnittstelle für Bundles fungiert. Bei ihr können mithilfe eines publish-find-bind Modells Bundles registriert oder gefunden werden. Die auf der linken Seite dargestellten Bundles, sind die gegen das OSGi Framework entwickelten Dienste eines Entwicklers. Durch das Security Modul können Sicherheitsregularien festgelegt werden. An der Technische Universität München (2015, 2016) wurde mit OSGi als Umgebung für Dienste im Auto experimentiert. Hierbei wurde es vor allem für Dienste aus hohen Abstraktionsebenen ohne Echtzeitanforderungen, die größtenteils aus der Multimedia Domäne kamen, eingesetzt. Sie stellten fest, dass OSGi generell geeignet für die Umgebung des Automobils ist. Es gibt Zahlreiche weitere Projekte die OSGi verwenden oder erweitern, eines davon von Gu u. a.

(2004), der eine Vorschlag zu einer kontextbewussten Middleware basiert auf OSGi darstellt. Diese versuche stellen alle den hohen Komfort der Arbeit mit OSGi heraus, verwenden es allerdings nur für Dienste ohne Zeitanforderungen in Lokalen Netzwerken. Außerdem ist das Framework Java basiert und somit nicht direkt auf den bereits im Auto existierenden Komponenten einsetzbar, auch wenn es bereits verschiedene Versuche gibt (siehe The C++ Micro Services Blog (2012)), OSGi auf anderen Plattformen und Programmiersprachen umzusetzen, was allerdings jedes Mal eine komplette neue Implementierung bedeutet.

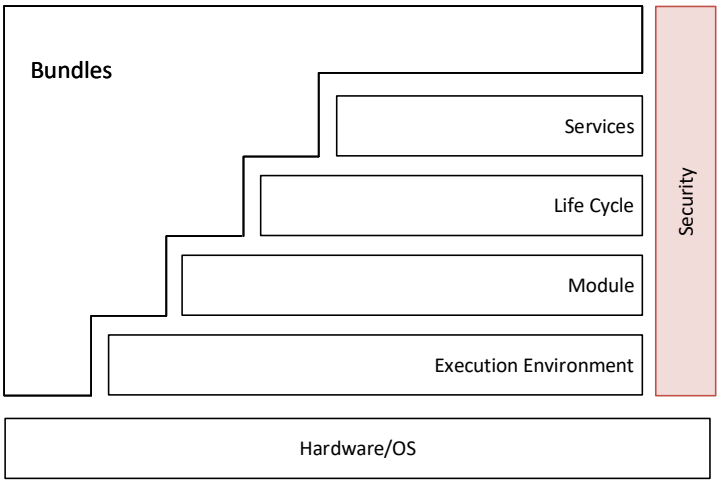


Abbildung 4.4: Aufbau des OSGi Frameworks (OSGi Alliance (2018)).

Als Vertreter der Industrie ist AUTOSAR eine der wichtigsten Gruppen für die Standardisierung in der Automobil Branche. Gopu u. a. (2016) untersuchten AUTOSARs Software Plattformen und stellt heraus, dass die Unterstützung von etablierten Legacy-Komponenten in der Automobilindustrie eine große Rolle spielt und durch Standardisierung erreicht werden kann. Außerdem stellt sich AUTOSAR der Herausforderung und integriert dienstorientierte Kommunikation in ihre Plattform. Hierzu entwickeln sie die Middleware Scalable service-Oriented MiddlewarE over IP (SOME/IP) und integrierten sie bereits im Standard. Dieses Protokoll löst die folgende Probleme mit den dazugehörigen Teilkomponenten: Daten Übertragung basieren auf TCP/User Datagram Protocol (UDP)-Internet Protocol (IP) (AUTOSAR Standard 809 (2016)), Daten Serialisierung zur Kompatibilität zu Kleinstgeräten (AUTOSAR Standard 637 (2014)), Service Discovery im Netzwerk (AUTOSAR Standard 616 (2016)) und die Übertragung von ECU Modulen in Dienste und Methoden (AUTOSAR Standard 660 (2016)). Mit dieser Architektur realisiert AUTOSAR Dienst basierte Kommunikation im Auto für Kleinst-

geräte auf einer sehr rudimentären Basis. Um dieses Konzept zu erweitern, realisiert ? eine zusätzliche Softwareplattform, die Adaptive Platform. Diese wird von Fürst und Bechter (2016) vorgestellt und rückt in ihrer Ausführungsumgebung die dynamischen Aspekte von Diensten in den Vordergrund. Die Adaptive Platform ist in Abbildung 4.5 dargestellt. Auch diese basiert auf der Middleware SOME/IP mit den oben genannten Standards.

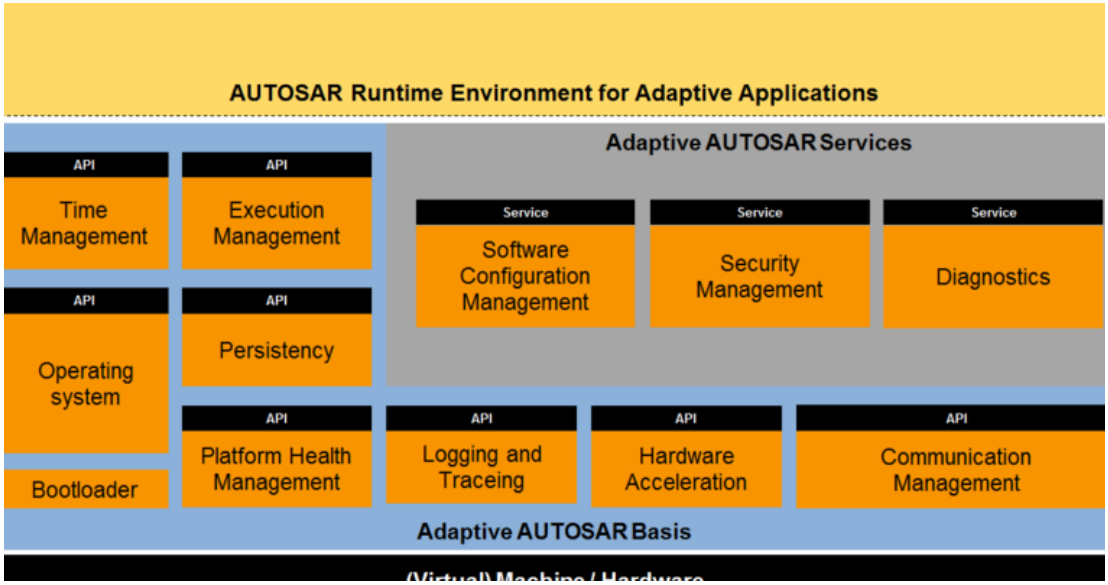


Abbildung 4.5: Aufbau der AUTOSAR Adaptive Platform (nach Fürst und Bechter (2016)).

Das Konzept von SOME/IP wird von Völker (2013) beschrieben und im Folgenden wiedergegeben. Zur Nachrichtenübertragung verwendet SOME/IP TCP oder UDP über IP mit einer einfachen datentypbasierten Serialisierung, welche auch die Kommunikation mit Kleinstgeräten ermöglicht. Außerdem implementiert die Middleware eine Service Discovery zum Auffinden und Registrieren von Diensten im Netzwerk. Diese unterstützt dank statischer Vorkonfiguration einen schnellen Aufstart des Gesamtsystems. Des Weiteren realisiert SOME/IP verschiedene Kommunikationsmuster: Entfernte Methodenaufruf mit (und ohne) Rückantwort, Eigenschaftsfelder mit Methoden zum Setzen und Abfragen der Werte, sowie Publish/Subscribe Mechanismen für Events und Event-Gruppen (eine Kombination aus mehreren Events oder Feldern).

Der Ansatz von AUTOSAR und SOME/IP ist ein gutes Vorbild für die in dieser Arbeit zu entwickelnde Middleware, gerade in Bezug auf die SIPS Dienstklasse. Jedoch ist zurzeit noch keine Echtzeit basierte Kommunikation über Ethernet geplant und auch keine Kommunikation in globale Netzwerken möglich.

Neben SOME/IP gibt es weitere Experimente aus dem Automobil und Cyberphysicalsystems Bereich, die hier kurz erwähnt werden sollen. Lim u. a. (2011) untersuchen Herausforderungen in einem zukünftigen IP/Ethernet-basierten Fahrzeug Netzwerk für Echtzeitanwendungen. Scholz u. a. (2009b) entwickeln eine angepasste SOA für Netzwerke von Eingebetteten Systemen. Außerdem gibt eines einige Middleware Plattformen wie OASIS von Hill u. a. (2010), RACE von Sommer u. a. (2013), iLand von Valls u. a. (2013) und SODA von Wagner u. a. (2014). Alle diese Systeme konzentrieren sich jedoch lediglich auf die Umsetzung IP-basierte Kommunikation in Automobilen und bringen wenig Neues im Bereich der Middleware Systeme oder heterogener Anforderungen.

Aufgrund der starken Ähnlichkeit der Sensor Aktor Systeme eines Automobils mit denen der Automatisierungstechnik, können auch Konzepte aus diesem Forschungsbereich ins Auto übertragen werden. Cucinotta u. a. (2009) und Jammes und Smit (2005) beschäftigen sich mit der Einführung der Service-Orientierung in die Automatisierungstechnik. Hierbei stellen sie ähnliche Anforderungen wie die in Kapitel 3.3 beschriebenen Aufgaben der Kommunikationsarchitektur im Auto dar. Der stärkste Unterschied ist, dass die Zeitanforderungen im Bereich von $20 - 200ms$ liegt und die im Auto noch etwas härter sind. Die größte Neuerung in der von ihnen entwickelten Softwareplattform *RI-MACS*, ist die Vorstellung eines geteilten Stacks von Kommunikationsprotokollen für verschiedenen Dienstklassen, welche von einer QoS basierten Middleware bereitgestellt und kontrolliert werden. Dieser Stack ist in Abbildung 4.6 dargestellt. Die dazugehörige Middleware stellt den Protokollstack laufenden Diensten über eine API zur Verfügung und überwacht die QoS-Eigenschaften der Kommunikation. Auch wenn das im Protokollstack verwendete DPWS (siehe Kapitel 2.3) nicht stark verbreitet und die von Cucinotta u. a. (2009) mit UDP und TCP realisierten Echtzeitdienste keine Zeitgarantien bereitstellen, ist die Aufteilung des Protokollstacks nach Dienstklassen ein sehr guter Ansatz.

Menascé u. a. (2007) betonen, dass QoS basierte Middleware-Lösungen besonders in stark verteilten dienstorientierten Systemen bei Komponenten mit heterogenen Anforderungen die beste Lösung sind. Sie präsentieren in ihrer Arbeit eine Middleware zur Verhandlung von Eigenschaften der Performanz. Die Verhandlung lösen sie über die in Abbildung 4.7 dargestellte QoS-Broker[3] Architektur. Der sogenannte QoS-Broker dient als Mittelsmann zwischen den Diensten. Mit ihm verhandelt ein Client, über ein Protokoll zur Verhandlung von QoS, die Performance Eigenschaften der Verbindung. Ein ähnliches Vorgehen wird sich auch in dieser Arbeit wiederfinden.

[3] QoS-Broker (deutsch: Dienstgüte Vermittler), ist eine häufig verwendete Middleware-Komponente, die zwischen Anbieter und Konsumenten vermittelt, um einen Dienstgütevertrag zu schließen.

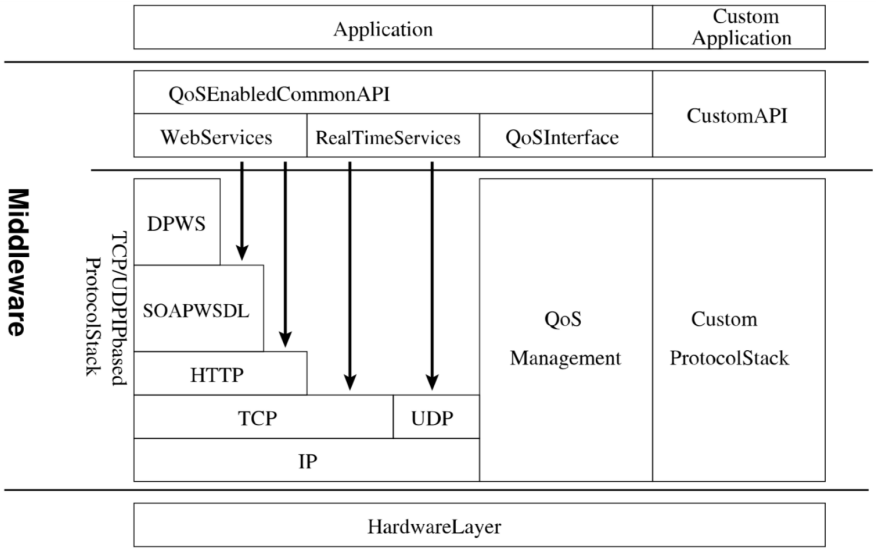


Abbildung 4.6: Protokollstack der RI-MACS Automatisierungsplattform von Cucinotta u. a. (2009).

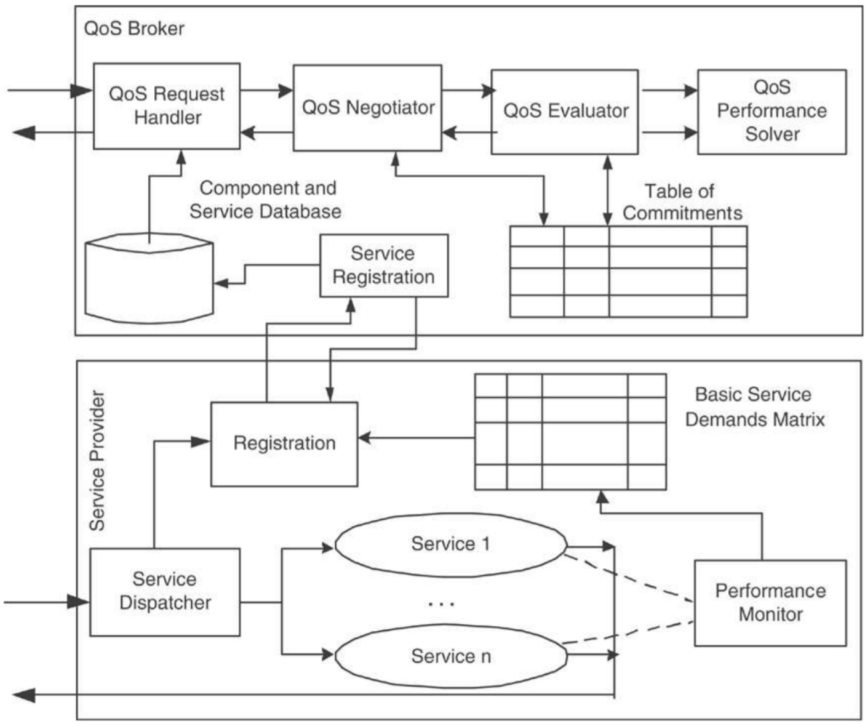


Abbildung 4.7: QoS-Broker Architektur nach Menascé u. a. (2007).

Es gibt verschiedenste weitere QoS basierte Architekturen aus dem Multimedia Bereich, die von Aurrecoechea u. a. (1998) verglichen werden und einen guten Überblick geben. Des Weiteren wenden Abdelzaher u. a. (2000) QoS basierte dienstorientierte Kommunikation im Echtzeitsystem Flugzeug an und legen den Fokus darauf Funktionsfähigkeit und Echtzeit für die "wichtigen" Komponenten zu garantieren. Der entscheidende Faktor für diese Garantien, ist die Vorhersagbarkeit des Verhaltens und der Anforderungen von Diensten an die Kommunikation. In ihrer Architektur müssen Dienste ihre Anforderungen über QoS Parameter klar bekanntgeben und auch ihre Priorität beschreiben. Je nach definiertem Ressourcen verbrauch, werden ihnen dann Anteile eines statisch definierten Ressourcenpools zugewiesen. Falls die angefragten Ressourcen nicht reserviert werden können, beginnt ein Vorgang den sie *Graceful Degradation* nennen. Statt Diensten die Kommunikation zu verwehren, werden Dienste anhand ihrer Priorität ausgewählt und in ihren QoS Anforderungen herabgestuft, zum Beispiel statt einen Wert alle 20 ms zu erhalten, erhalten sie ihn nur noch alle 50 ms. Auch diese verschiedenen Anforderungsstufen werden bei der Erstellung eines Dienstes als QoS Parameter angegeben. So kann die Kommunikation für die entscheidenden Komponenten auch in Extremsituationen garantiert werden. Dieser Ansatz ist besonders interessant, wenn man eine Plattform für Automobile Dienste entwickeln möchte, da dann ein Ressourcenpool realisiert werden kann. Für eine Middleware ist allerdings gerade der Ansatz der vorher definierten benötigten Ressourcen in Echtzeitsystemen interessant. Ein ähnliches Vorgehen ist bereits aus Echtzeit basierten Ethernet Standards wie AVB bekannt (siehe Kapitel 2.1.2).

Ein weiterer Ansatz für QoS basierte Middleware-Lösungen ist Data Distribution Service (DDS) aus dem Standard der Object Management Group (2015). Auch hier werden QoS Parameter benutzt um die Kommunikationseigenschaften von Endpunkten zu steuern. Der Unterstandard *Data Centric Publish Subscribe (DCPS)* definiert dabei eine mögliche Umsetzung des Konzeptes über Publish Subscribe. Abbildung 4.8 stellt das Grundlegende Klassenkonzept da. Im Netzwerk werden Dienste von einem Publisher für die Anbieter oder von einem Subscriber für die Konsumenten repräsentiert. Die Daten werden über einen Writer an den Publisher weitergegeben und über einen Reader vom Subscriber empfangen. Außerdem definieren sie verschiedene Klassen für QoS Eigenschaften. Wie schon im Abschnitt zu SOME/IP erklärt, ist das Publish Subscribe Muster sehr passend für die Kommunikation im Auto. Die Entkopplung des Publishers vom Dienst selbst ermöglicht, dass ein Writer mehrere Publisher bedient.

4.2.2 Auswertung

Wie gezeigt, ist die Forschungslandschaft im Bereich Middleware-Lösungen sehr groß und es gibt für die verschiedensten Anforderungen bereits verschiedene Lösungen. Einige versuchen

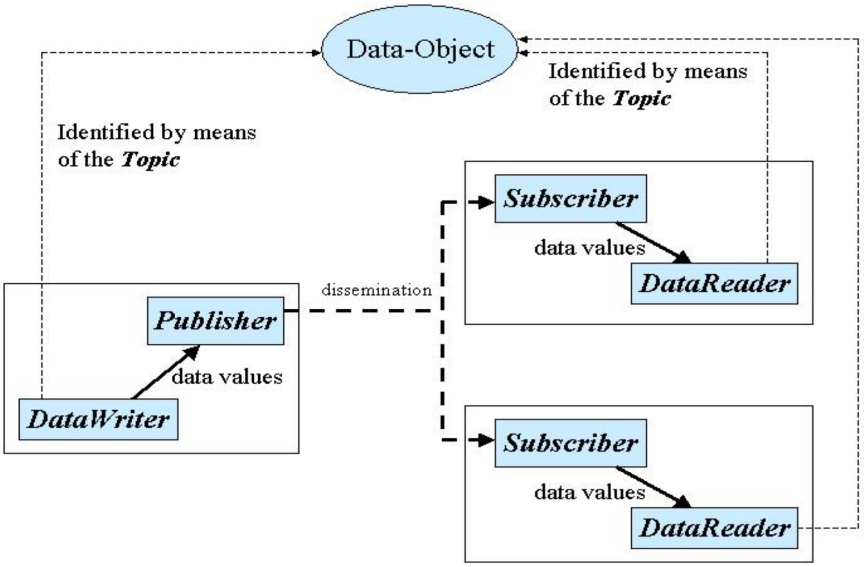


Abbildung 4.8: Umsetzung von Publish/Subscribe nach DDS Standard der Object Management Group (2015).

bereits SOA im Automotive Bereich umzusetzen und erfüllen schon einige der in Kapitel 3 beschriebenen Anforderungen. Außerdem kann aus anderen Bereichen viel über QoS orientierte Middlewares gelernt werden. Einige Ansätze zeigen, dass diese Konzepte auch in Echtzeitsysteme, wie das Auto, übertragen werden können. Allerdings unterstützt derzeit kein Ansatz alle definierten Anforderungen der Dienste. Daher werden im Folgenden die ausgewählten Konzepte dargestellt. Diese fließen später in den Entwurf mit ein.

Besonders interessant sind für diese Arbeit: Die Publish/Subscribe Architektur der DDS der Object Management Group (2015); Die IP basierten Dienste und definierten Kommunikationsmuster aus SOME/IP von Völker (2013) und Fürst und Bechter (2016); Die Auswahl verschiedener Wege durch einen Multi-Protokollstack wie bei Cucinotta u. a. (2009), um verschiedene Anforderungsklassen zu realisieren; Sowie die Verwendung von vorher definierten Anforderungen bei Echtzeitdiensten, die für Echtzeitprotokolle benötigt werden, wie bei Abdelzaher u. a. (2000).

Für eine eventuelle Erweiterung sind besonders die Plattform basierten Ansätze interessant, da diese den Diensten nicht nur eine Middleware für die Kommunikation bereitstellen, sondern eine Komplette Laufzeitumgebung. Nur so kann volle Kontrolle für Echtzeitanwendungen geboten werden. Gerade die von Fürst und Bechter (2016) beschriebene AUTOSAR Adaptive Platform bietet hier viele interessante Ansätze, aber auch das Konzept der Degradierung von Diensten in Extremsituationen von Abdelzaher u. a. (2000) ist für eine solche Plattform nützlich.

4.3 Verbreitung und Migrationsstrategien

Bei der Entwicklung eine Middleware-Lösung ist es wichtig festzustellen, wer diese später nutzen soll. Durch die drastischen Änderungen, die eine zentralisierte dienstorientierte Architektur über das Gesamtsystem Auto bringt, ist es wichtig den Autoherstellern mögliche Migrationsstrategien darzulegen. Diese müssen es ermöglichen die bereits existierenden Komponenten weiterzuverwenden und Schritt für Schritt in die neue Architektur zu integrieren. Solche Strategien werden im Folgenden vorgestellt.

4.3.1 Verwandte Arbeiten

Um den Konzernen verschiedenen Migrationsstrategien offen zu lassen, überarbeitete AUTOSAR ihr Portfolio an Softwarearchitekturen. Die bisher existierende Plattform wurde zur *Classic Platform* und die bereits beschriebene neue *Adaptive Platform* wurde hinzugefügt. Beide teilen sich gemeinsame Komponenten aus der *Foundation*. In die Classic Platform wurde Ethernet als neues Kommunikationsmedium integriert und Service basierte Kommunikation eingeführt. Für neue dynamische Dienste wurde die Adaptive Platform als Service Execution Environment (siehe Kapitel 2.4) etabliert. So können alte Komponenten auf der Classic Platform laufen und mithilfe der neuen Kommunikationsmedien mit Diensten der Adaptive Platform kommunizieren. Dies ermöglicht eine Migration in kleinen Schritten.

Stähle u. a. (2013) stellen in ihrer Arbeit die folgenden produktunabhängigen Strategien vor: In der ersten wird die neue Architektur in einem abgeschotteten Subsystem umgesetzt und an ein zentrale Gateway angeschlossen. So können Komponenten Schritt für Schritt in das neue Subsystem überführt werden. Dies hat den Vorteil, dass einzelne Dienste integriert werden können. Allerdings ist dies aufgrund der engen Zusammenarbeit der Dienste innerhalb einer Domain nicht immer Sinnvoll, da dies die Kommunikationsanforderungen an das zentrale Gateway drastisch erhöhen würde. Daher müssten eigentlich ganze Domains überführt werden. Diese Vorgehensweise wird in dem *5 Module Concept* umgesetzt (siehe Abbildung 4.9). Hiernach werden die Funktionen nach ihren Anforderungen, ähnlich der Domänen aus Kapitel 4.1, in Module eingeordnet. Die Module bleiben voneinander getrennte Systeme, die intern ihre Systemarchitektur beibehalten können. Im ersten Schritt werden so lediglich die Module untereinander mit Hilfe der neuen Architektur verbunden. Die Kommunikation innerhalb der Module kann dann Schritt für Schritt umgestellt werden. Dieses Vorgehen erlaubt den Autoherstellern, die derzeitigen Domains beizubehalten und voneinander zu trennen, bis sie vollständig und sicher integriert werden können. Die dritte Migrationsstrategie, die Stähle u. a. (2013) präsentieren, bezieht sich auf Elektromobilität. Da hier eine komplette Neuimple-

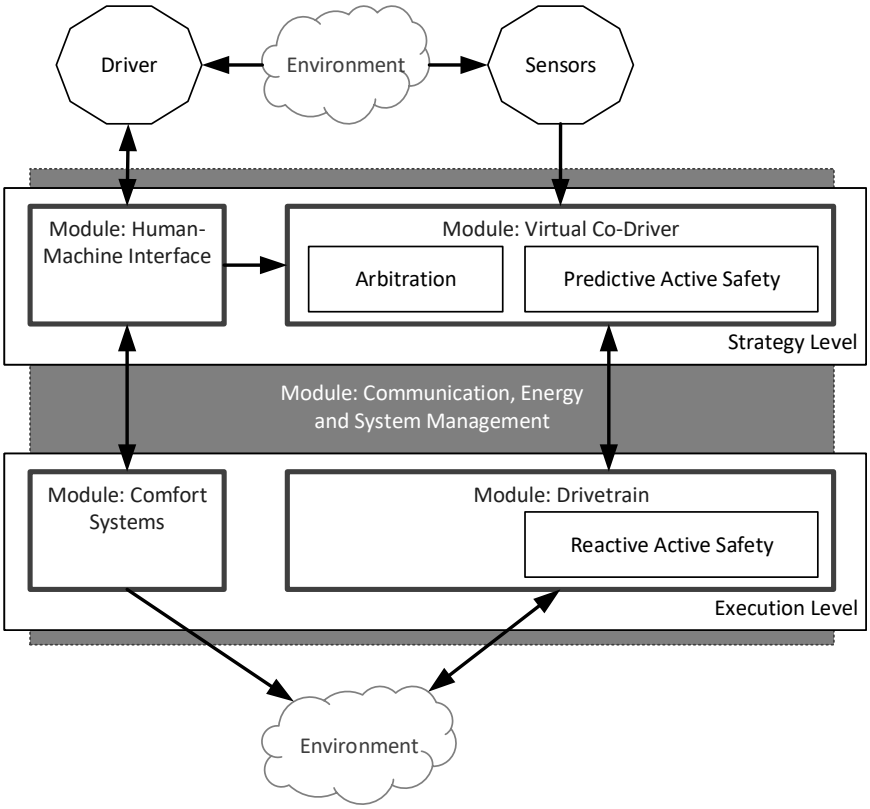


Abbildung 4.9: *5-Module Concept* nach Stähle u. a. (2013).

mentierung des Drive Trains nötig ist, kann dabei auch die Systemarchitektur von Grund auf erneuert werden. So würde ein Komplettumstieg in einem Schritt erfolgen. Neue Hersteller, die in der Elektromobilität auf den Markt drängen, integrieren die neuen Ansätze direkt in ihre Konzepte. Basierend auf dieser Vorarbeit könnten die etablierten Hersteller entscheiden, ob sie ebenfalls einen Umstieg wagen.

4.3.2 Vergleich und Auswertung

Das 5 Module Concept hat den Vorteil, dass Subsysteme generalisiert behandelt werden können. Dieser Ansatz ähnelt den vorgestellten Domänen und wird daher in dieser Arbeit verfolgt. Mit Hilfe von Gateways sollen diese Subsysteme an die neue Architektur gekoppelt werden. Dieser Ansatz wird in Kapitel 6.2 präsentiert.

5 Middleware-Entwurf

Basierend auf den beschriebenen Middleware-Konzepten wird im Folgenden der Entwurf der Entwickelten Middleware dargestellt. Hierfür werden die Middleware-Architektur, die Protokolle und Verhandlungsstrategien zur Ermittlung der Dienstgüte, sowie die umgesetzten Verbindungsendpunkte beschrieben. Abschließend werden Strategien zur Qualitätssicherung dargestellt.

5.1 System Architektur

Um das Middleware-Konzept aus Kapitel 4 umzusetzen wird ein Entwurf erstellt, der die Service Klassen, wie in Kapitel 4.1 beschrieben, enthält. Die daraus entstandene grundlegende Middleware-Architektur ist in Abbildung 5.1 dargestellt. Sie besteht aus vier Hauptkomponenten: Local Service Manager (LSM), Local Service Registry (LSR), Service Endpoint Factory (SEF), Quality-of-Service Manager (QoSM).

Der LSM dient als Eintrittspunkt der Middleware und realisiert das API. Der LSM überwacht alle auf dem lokalen Gerät laufenden Dienste und hält Referenzen zu ihnen. Über das API bietet der LSM die Möglichkeit, Dienste mit QoS Eigenschaften anzubieten und zu konsumieren. Um dem *seperation of concern* Paradigma (Practices, 2009, Kapitel 2) zu folgen, werden die verschiedenen Aufgabenbereiche in die folgenden drei Submodule unterteilt, die jeder einen Teil des API implementieren.

Um Dienst orientierte Kommunikation zu ermöglichen, müssen Klienten Dienste im Netzwerk finden können. Die LSR realisiert ein solches Verzeichnis, durch die Implementierung des *Registry Patterns* (Fowler, 2002, Seite 480) in einer lokalen Datenbank. Diesen enthält alle bekanntgemachten Dienste und bietet die Funktionalität Dienste im Netzwerk zu suchen. Da dieses Muster eine weit verbreitete Technologie ist, wird die Implementierung in dieser Arbeit nicht im Detail behandelt. Die LSR wird lediglich in einer statischen Version, als Liste aller existierenden Dienste realisiert, die jeder Netzwerkknoten enthält.

Die SEF stellt Fabrik Methoden nach dem *Fabrik Muster* (Gamma, 1995, Seite 107) zur Verfügung, mit deren Hilfe Dienstendpunkte für die verschiedenen Kommunikationsmuster erstellt werden können. Die SEF entscheidet anhand von QoS Parametern, welcher Endpunkt

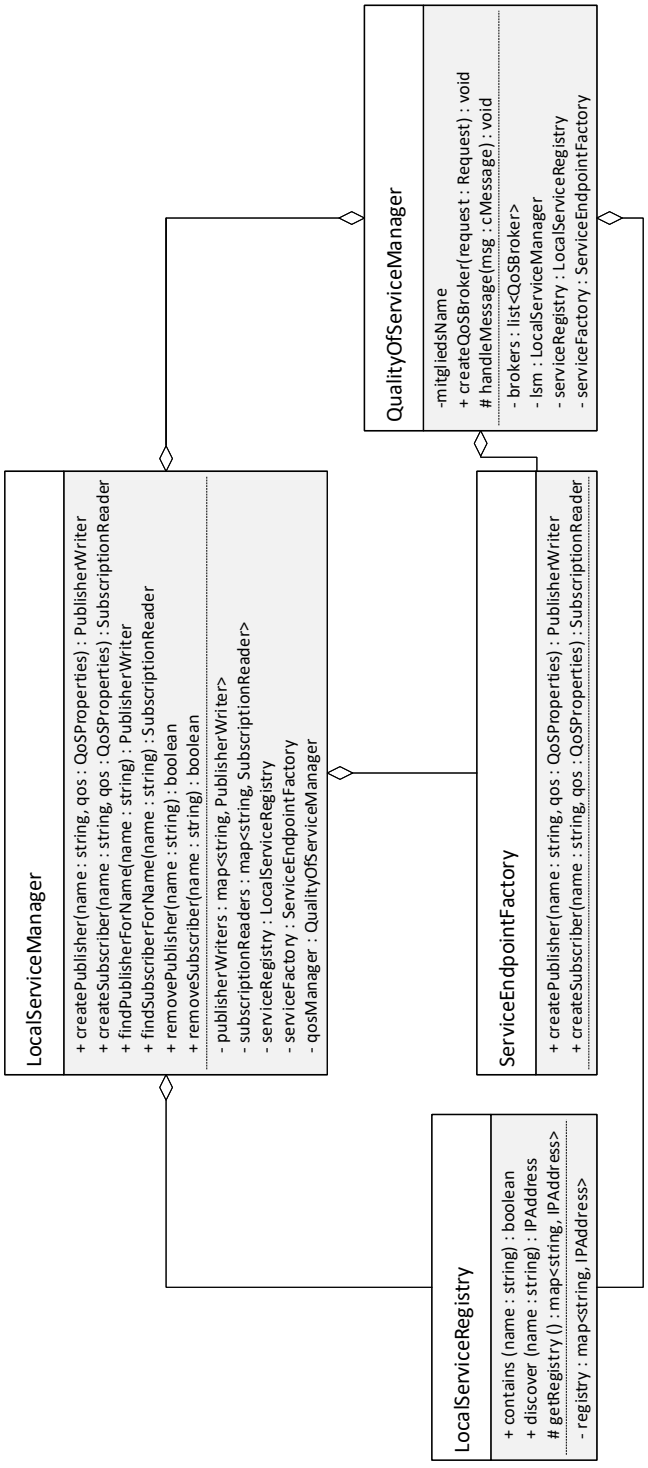


Abbildung 5.1: Grundlegende Komponenten der Middleware-Architektur.

mit welchen Eigenschaften erstellt wird, z.B. ein echtzeitfähiger Subscriber. Eine genauere Beschreibung dieses Prozesses folgt in Kapitel 5.3.

Um einen Verbindung mit QoS Eigenschaften zu erstellen, realisiert das QoSM ein Protokoll zur Dienstgüteverhandlung (engl. QoS Negotiation Protocol). Es nutzt außerdem die SEF dafür Endpunkte zu erstellen und die LSR um Dienste im Netzwerk zu finden. Des Weiteren hat das QoSM in einer zukünftigen Version die Aufgabe, die Einhaltung der Dienstgütevereinbarung zwischen zwei Diensten zu Kontrollieren und bei Problemen zu reagieren. Die Kommunikation aller mithilfe des QoSM erstellten Dienste sollte überwacht werden, um zu überprüfen, ob maximale Latenzen eingehalten werden und die beantragte Bandbreite nicht überschritten wird. Dies geschieht auf Basis von Statistiken, die beim Eingang neuer Nachrichten aktualisiert und zur Laufzeit überprüft werden können. Bei Problemen können die QoS Anforderungen der Dienste auf vorher definierte Werte reduziert werden, anstatt die Verbindung ganz zu beenden. Der Vorgang der Dienstgüteverhandlung wird in Kapitel 5.4 beschrieben.

5.2 Protokollstack

Cucinotta u. a. (2009) entwickeln in ihrer Arbeit einen Protokollstack für verschiedenen Anforderungsklassen, wie in Kapitel 4.2 beschrieben. In diesem Stack hat jede Anforderungsklasse ihren eigenen *Pfad* durch den Protokollstack. In dieser Arbeit wird dieses Konzept mit einer dienstorientierten Middleware kombiniert. Die Aufgabe der Middleware besteht darin, die Pfade durch den Protokollstack anhand der QoS Eigenschaften auszuwählen und den Protokollstack vor der Anwendung zu verbergen. Im Folgenden wird der verwendete Protokollstack (siehe Abbildung 5.2), aufgeteilt nach den Dienstklassen (siehe Kapitel 4.1), beschrieben. Die verwendeten Ethernet Echtzeiterweiterungen sind bereits im Grundlagen Kapitel beschrieben: Time-Sensitive Networking siehe Kapitel 2.1.3, Time-Triggered Ethernet siehe Kapitel 2.1.1 und Audio Video Bridging siehe Kapitel 2.1.2. Im Entwurf wird zwar mit einem Vollständigen TSN Stack geplant, allerdings werden in der Simulation die entsprechenden Komponenten von AVB verwendet.

Die statische Kommunikation mit Definierten Routen, Sendezeitpunkten und Datenmengen wird mit Hilfe von TSN mittels TTE ähnlicher Kommunikation direkt auf Layer 2 des OSI Modells umgesetzt. Dies wurde bereits von Steinbach u. a. (2014) und Meyer u. a. (2013) getestet. Da dieser Verkehr nicht über die Middleware läuft, wird dieser Weg hier nicht weiter beschrieben.

Die Kommunikation der RTS Dienstklasse wird ebenfalls mittels TSN bzw. AVB basierter Echtzeitkommunikation der Klasse A und B umgesetzt. So werden Streams im Netzwerk

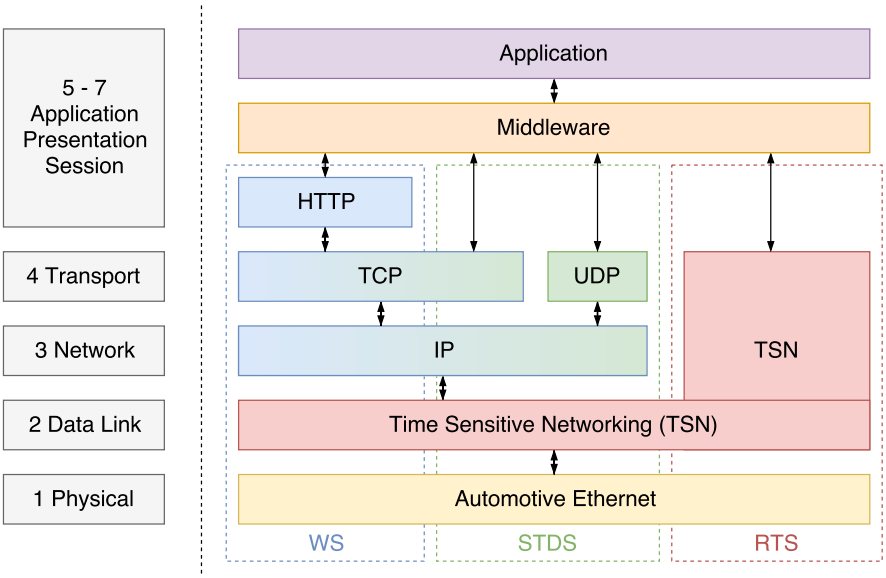


Abbildung 5.2: Entwurf des Protokollstacks.

reserviert und es können harte Echtzeitgarantien bis zu 2 ms für vorher angemeldete und reservierte Datenmengen gegeben werden. TSN setzt hierbei nicht nur die Datenübertragung auf Layer 2 um, sondern auch die Auffindung von Diensten im Netzwerk und den Transport von Nachrichten.

Für Dienste der SIPS Klasse, wird ein IP-basierter Stack verwendet. Dieser wird auf Layer 2 über die BE Klasse von TSN übertragen. Auf der Transportebene wird je nach QoS Anforderungen TCP oder UDP verwendet. So können Dienste angeben ob sie nach dem Konzept "Fire and Forget" über UDP übertragen oder ein zuverlässiges verbindungsorientiertes Protokoll verlangen. In einer zukünftigen Plattform Architektur könnte zwischen der Transportebene und der Middleware noch SOME/IP eingefügt werden, um Kommunikation mit SOME/IP Diensten zu realisieren.

Die WS Klasse setzt auf den TCP-IP Stack noch das HTTP. So können alle modernen Web Service Implementierungen realisiert werden, wie z.B. SOAP oder ReST.

5.3 Verbindungsendpunkte

Um mit einer Middleware dienstorientierte Kommunikation durchzuführen, muss sie der Anwendung, wie in Kapitel 2.3 beschrieben, Endpunkte bereitstellen. Diese Endpunkte realisieren die tatsächliche Kommunikation und entkoppeln den Dienst von Protokollen. Im Konzept der

Middleware ist zwar die Umsetzung verschiedener Kommunikationsmuster geplant, in dieser Arbeit wird jedoch lediglich ein Publish/Subscribe Mechanismus implementiert, um an ihm die QoS-Verhandlung darzustellen. Weitere Kommunikationsmuster wie Events, Nachrichten oder RPCs werden nicht realisiert, können allerdings analog dazu umgesetzt werden. Da die Middleware die Endpunkte entsprechend der in Kapitel 5.4 QoS-Verhandlung auswählt, müssen für alle Dienstklassen (siehe Kapitel 4.1) und die Protokolle des Stacks (siehe Kapitel 5.2), spezialisierte Endpunkte vorhanden sein.

Für die Umsetzung wurden die in Kapitel 4.2 beschriebenen Konzepte verwendet. Insbesondere wurde das Konzept der Data Distribution Service Middleware der Object Management Group (2015) für den Aufbau der Publish/Subscribe Architektur verwendet. Im Folgenden werden die entwickelten Endpunkte am Beispiel des Publish/Subscribe Mechanismus beschrieben.

Die verschiedenen Kommunikationsmuster realisieren alle dieselbe, stark entkoppelte Klassenstruktur, um Erweiterbarkeit zu gewährleisten. Es gibt eine Schnittstelle *IEndpoint*, die von allen Endpunkten implementiert werden muss. So kann die Middleware leicht um zusätzliche Kommunikationsmuster erweitert werden. Für jedes Kommunikationsmuster wird ein Anbieter und ein Konsument realisiert. Diese realisieren die generelle Funktionalität des Kommunikationsmusters. Sowohl für die Basisklasse des Anbieters, als auch für die des Konsumenten, müssen nun drei Subklassen für die Dienstkategorien (siehe Kapitel 4.1.3) realisiert werden. Anschließend wird die konkrete Umsetzung eines Protokolls für das Kommunikationsmuster einer dieser drei Kategorien zugeordnet.

In Abbildung 5.3 sind die Endpunktklassen des Publish/Subscribe Mechanismus beschrieben. Für diese Arbeit wurden sowohl für TCP als auch für AVB spezialisierte Publisher und Subscriber implementiert. Diese erben die Eigenschaften der übergeordneten Dienstklassen. Die Abstrakten Publisher und Subscriber Klassen implementieren die Grundfunktionalität, wie z.B. das veröffentlichen neuer Pakete. Außerdem gibt es die Möglichkeit, die Verbindungsinformationen auszulesen. So kann das Quality-of-Service Negotiation Protokol (QoSNP) diese an die andere Seite weiterleiten (siehe Kapitel 5.4).

Ein Dienstanbieter kann mehrere Endpunkte besitzen, die den gleichen Dienst an verschiedene Konsumenten mit verschiedenen QoS-Anforderungen ausliefern. Hierfür muss die Anwendung des Dienstanbieters von den Endpunkten entkoppelt werden. Dies geschieht mittels sogenannter Verbindungsklassen. Für das Publish/Subscribe Muster gibt es zwei verschiedene, welche in Abbildung 5.4 dargestellt sind.

Auf Publisher-Seite gibt es einen *PublisherWriter*, dessen Aufgabe darin besteht, die von der Anwendung veröffentlichten Nachrichten, an alle Publisher-Endpunkte weiterzugeben.

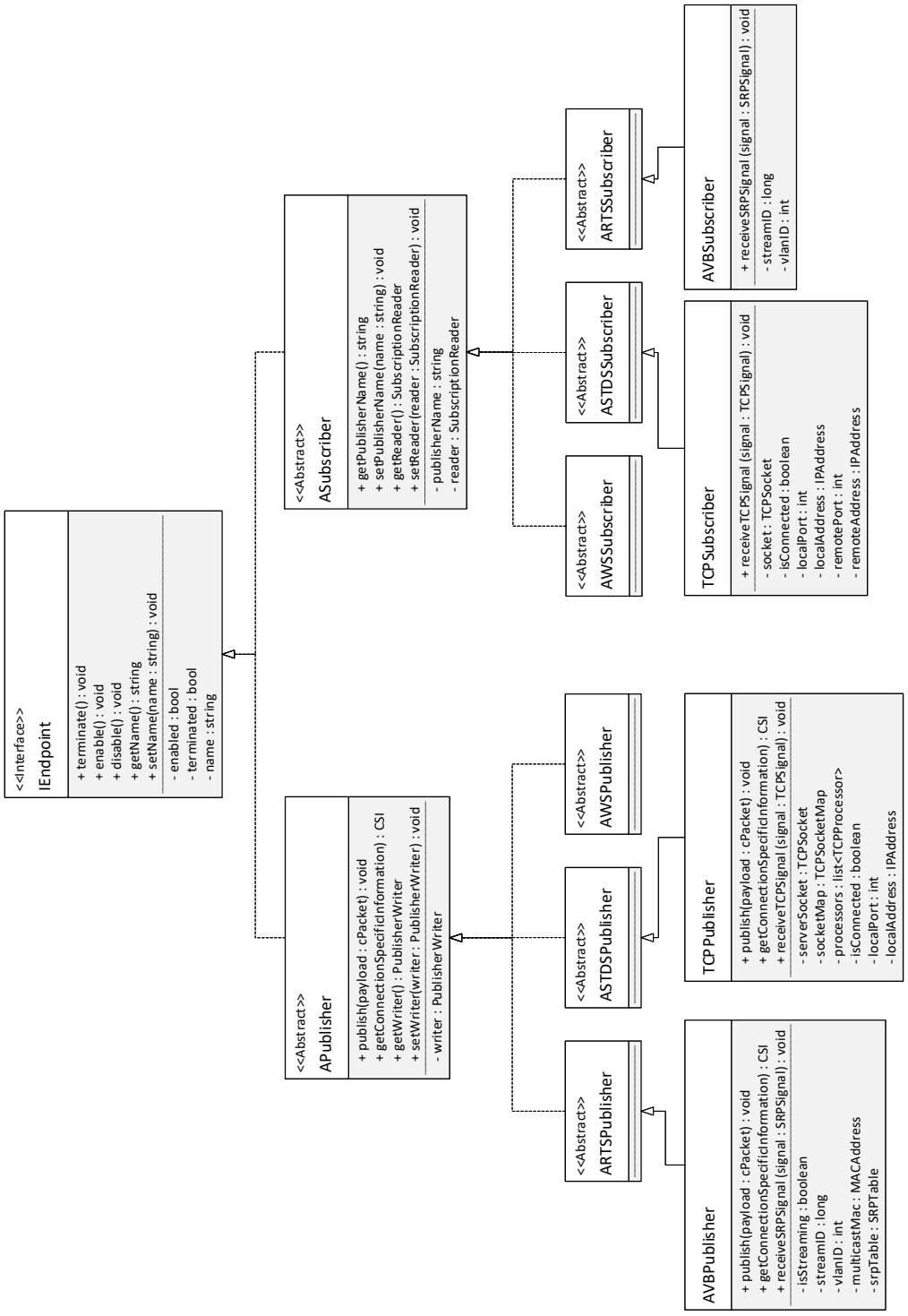


Abbildung 5.3: Klassenhierarchie der Endpunkte am Beispiel Publish/Subscribe.

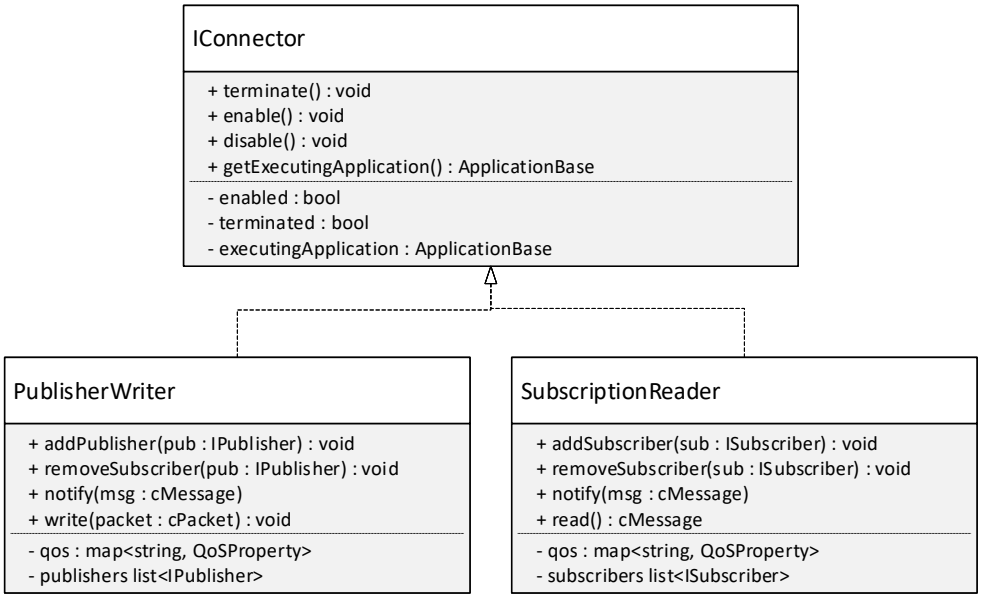


Abbildung 5.4: Verbindungsklassen des Publish/Subscribe Mechanismus, mit ihnen wird die Anwendung mit dem zuständigen Endpunkt verbunden.

Diese Endpunkte übertragen dann, je nach ihren QoS-Eigenschaften, die Nachrichten an den verbundenen Subscriber-Endpunkt.

Auf Seite des Subscribers gibt es einen *SubscriptionReader*, welcher die Anwendung mit dem Subscriber-Endpunkt verbindet. Wenn also Nachrichten im Subscriber-Endpunkt eintreffen, gibt er diese an den verbundenen *SubscriptionReader* weiter, der sie wiederum an die abonnierende Anwendung weiter gibt.

Will eine Anwendung also einen Dienst anbieten, so erhält sie nur den dazugehörigen *PublisherWriter*. Wenn mithilfe des QoSNP neue Verbindungen aufgebaut werden, verbindet der QoSM den *PublisherWriter* mit dem neu erstellten spezialisierten Publisher-Endpunkt. Alle neuen Nachrichten des Dienstanbieters werden dann über diesen Endpunkt verbreitet.

Auf Seite des Konsumenten funktioniert es auf die gleiche Weise. Eine Anwendung, die einen Dienst konsumieren will, erhält einen dazugehörigen *SubscriptionReader*. Bei Abschluss der Verhandlungen, erstellt der QoSM einen spezialisierten Subscriber-Endpunkt und verbindet diesen mit dem *SubscriptionReader*. Anschließend werden beim Subscriber-Endpunkt eintreffende Daten über den *SubscriptionReader* an die Anwendung des Konsumenten übertragen.

5.4 Protokoll zur Dienstgüteverhandlung

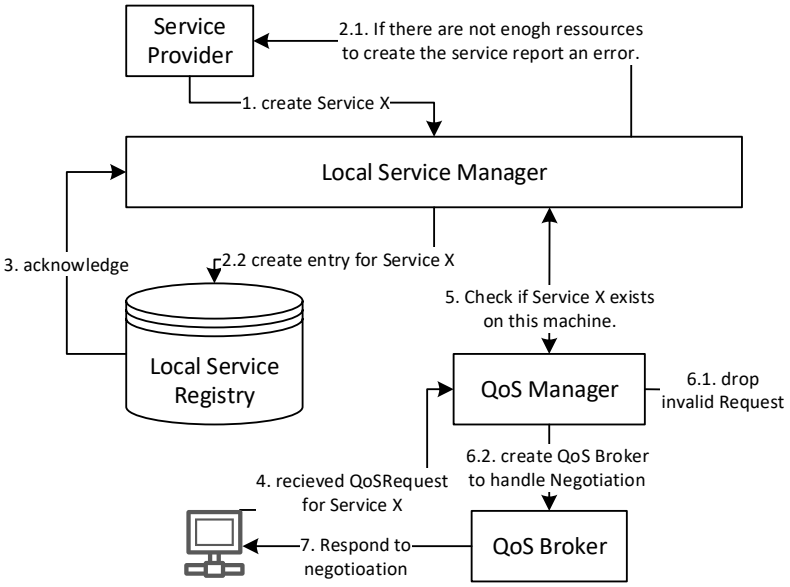
Um Verbindungen mit Dienstgüterichtlinien aufbauen zu können, muss zunächst eine Dienstgütevereinbarung zwischen Anbieter und Konsumenten ausgehandelt werden. In diesem Kapitel wird der Entwurf des in der Middleware verwendeten Protokolls zur Verhandlung der Dienstgüte beschrieben. Dieses bestimmt am Ende, welche Protokolle zur Übertragung genutzt werden und welche anderen Verbindungseigenschaften relevant sind. Der Entwurf baut auf den Erkenntnissen (siehe Kapitel 4.2) von Cucinotta u. a. (2009) im Bereich der QoS-basierten Multi-Protokollstacks und von Menascé u. a. (2007) im Bereich der QoS-Verhandlung mithilfe eines QoS-Broker auf.

Bevor mit der Aushandlung der Dienstgütevereinbarung begonnen werden kann, muss sowohl der Konsument, als auch der Anbieter einige Vorbereitungen treffen. Diese sind in Abbildung 5.5 getrennt nach Konsument 5.5b und Anbieter 5.5a dargestellt. Zu Beginn erstellt der Anbieter einen neuen Dienst und erhält einen einen entsprechende Implementierung der Verbindungsklasse. Dieser Dienst wird in der LSR eingetragen und ist somit im Netzwerk bekannt. Anschließend können Konsumenten Anfragen an den Dienst richten. Der Konsument stellt nun eine solche Anfrage, mit seinen QoS-Anforderungen an die Verbindung und gibt sie an den LSM weiter. Dieser versucht den Dienst im Netzwerk ausfindig zu machen und startet, falls er gefunden wurde, über den QoSM einen QoS-Broker, der die Aushandlung für den Konsumenten übernimmt.

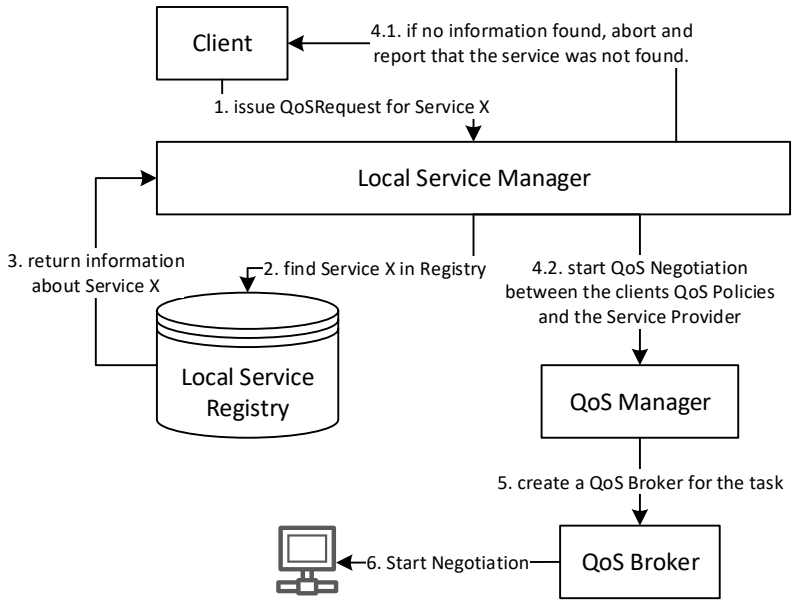
Wenn auf der Anbieter Seite eine Anfrage eines QoS-Brokers ankommt, überprüft der LSM zunächst, ob der angefragte Dienst auf dieser Maschine existiert. Falls er existiert, wird auch auf dieser Seite ein QoS-Broker erstellt, der den Anbieter bei der Verhandlung vertritt. Falls er nicht existiert, wird der Konsument benachrichtigt. Nun kann die Verhandlung über das QoSNP beginnen.

Das QoSNP wird sowohl auf Seite des Konsumenten, als auch auf Seite des Anbieters durch den QoS-Broker umgesetzt. Dieser Broker implementiert dazu die in Abbildung 5.6 dargestellten Zustandsautomaten, für die Seite des Konsumenten 5.6b und des Anbieters 5.6a.

In Abbildung 5.7 ist ein Ablaufdiagramm eines erfolgreichen Verbindungsaufbaus dargestellt über das Protokoll dargestellt. Das Protokoll besteht aus zwei Phasen. In der ersten Phase wird ein Handshake ausgeführt. Der Broker des Konsumenten sendet eine Anfrage mit den geforderten QoS-Eigenschaften. Der Broker des Anbieters bestätigt, dass der Dienst unter der angegebenen Adresse erreichbar ist und die ausführende Maschine des Anbieters generell in der Lage wäre, die Anfrage des Konsumenten zu bedienen. Dies bedeutet, dass sie sowohl die angeforderten Protokolle und QoS-Eigenschaften unterstützt, als auch genügend Ressourcen

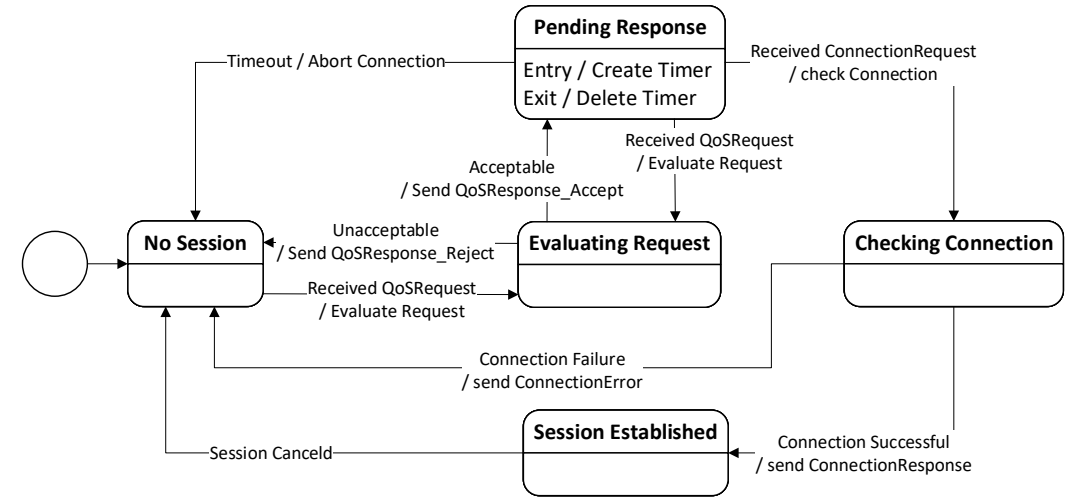


(a) Seite des Anbieters.

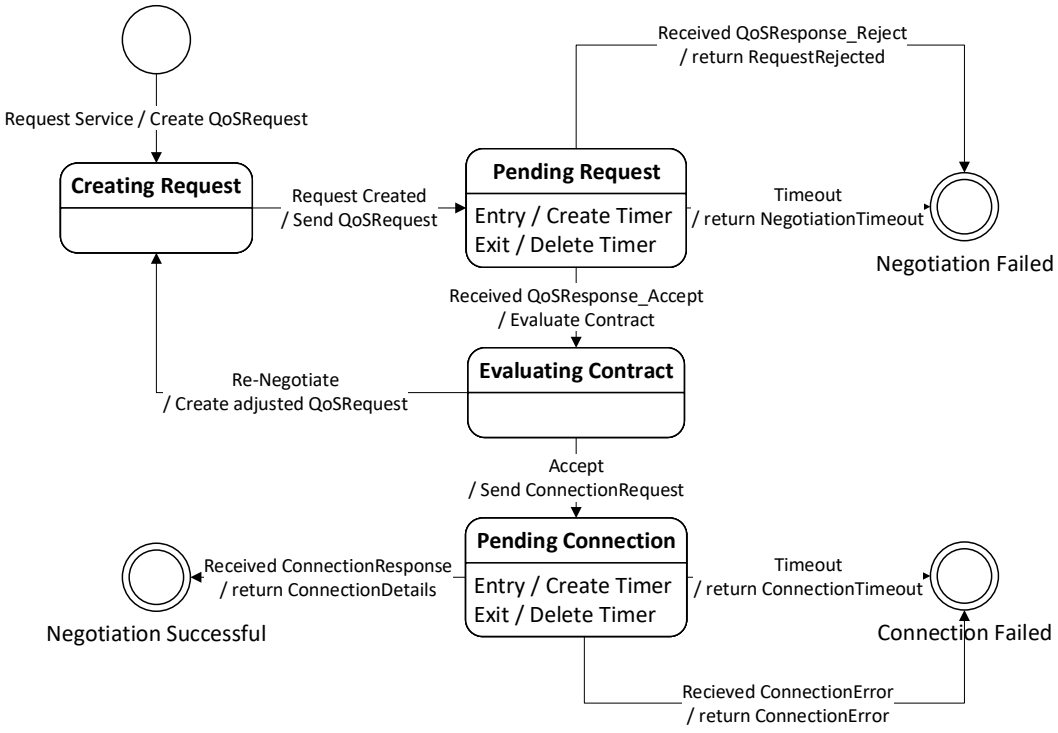


(b) Seite des Konsumenten.

Abbildung 5.5: Vorbereitung der QoS-Verhandlung.



(a) Seite des Anbieters.



(b) Seite des Konsumenten.

Abbildung 5.6: Umgesetzte Zustandsautomaten für das QoSNP.

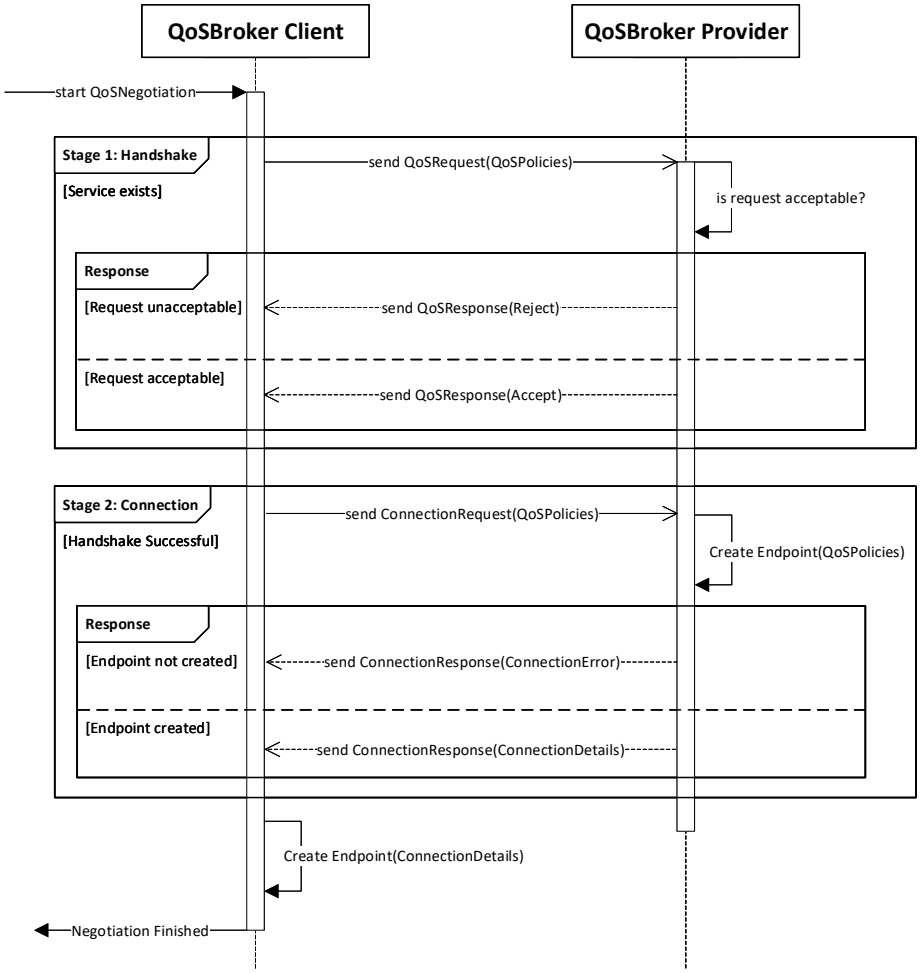


Abbildung 5.7: Ablaufdiagramm der Dienstgüteverhandlung.

zur Verfügung hat, um einen neuen Endpunkt auf dem System zu erstellen. Falls dies also der Fall ist, wird eine Antwort gesendet, dass die Anfrage akzeptabel ist. Falls die Anfrage nicht akzeptiert werden kann, wird dies ebenfalls mitgeteilt.

An dieser Stelle könnte als zukünftige Erweiterung eine Nachverhandlung mit möglicher Degradierung, wie in Kapitel 4.2 beschrieben, durchgeführt werden. So könnten auch bei (teilweiser) nicht Erfüllung der Anforderungen, Daten ausgetauscht werden.

In der zweiten Phase werden anschließend die Endpunkte für die Verbindung, entsprechend der getroffenen Vereinbarung, erstellt. Hierzu wird vom Broker des Konsumenten eine Nachricht gesendet, die den Verbindungsaufbau mit den Verhandelten Eigenschaften fordert. Daraufhin wird auf Seite des Anbieters ein entsprechender Endpunkt erstellt (falls er nicht schon existiert), der dann mit dem Anbieter über die Verbindungsklasse verbunden wird. Anschließend werden vom Broker des Anbieters die Verbindungsdetails (z.B. Port, AVB-Klasse, o.ä.) an den Broker des Konsumenten gesendet und auch auf dieser Seite ein entsprechender Endpunkt erstellt. Dieser baut dann eine Verbindung mit dem Endpunkt des Anbieters auf und wird mit der Verbindungsklasse an den Konsumenten angebunden wird.

5.5 Qualitätssicherung

Die Qualitätssicherung[1] in Verteilten Systemen ist häufig eine große Herausforderung, da es viele unvorhersagbare Abläufe gibt. Die Qualität eines Software Produkt wird laut Hoffmann (2013) in die in Abbildung 5.8 dargestellten Bausteine unterteilt. Um diese Kriterien zu überprüfen, gibt es verschiedene Methoden. Im Folgenden werden die angewandten und geplanten Vorgehensweisen beschrieben.

Die Hersteller orientierten Qualitätsmerkmale Übertragbarkeit, Wartbarkeit und Transparenz werden insbesondere durch ein gutes Softwaredesign gewährleistet. Daher wurde die Middleware in erweiterbare und klar strukturierte Komponenten aufgeteilt, nach den Prinzipien des *Information Hiding* und *Seperation of Concern*. Des Weiteren wurde die weit verbreitete Programmiersprache C/C++ gewählt, um zu möglichst vielen Gerätetypen kompatibel zu sein. Außerdem wurde viel Wert auf eine gute Dokumentation der Designentscheidungen gelegt. Durch diese Prinzipien wird auch die Testbarkeit erhöht.

Zur Überprüfung der Funktionalität und der Zuverlässigkeit wurden Unit Tests für Methoden und Komponenten erstellt. Außerdem wurden an verschiedenen Stellen Stub-Implementierungen eingesetzt, um das korrekte Zusammenspiel von Komponenten nachzuweisen. Sowohl Unit

[1] Qualitätssicherung in der Software bedeutet die Überprüfung der Erfüllung der festgelegten Anforderungen anhand verschiedener Merkmale und Metriken.

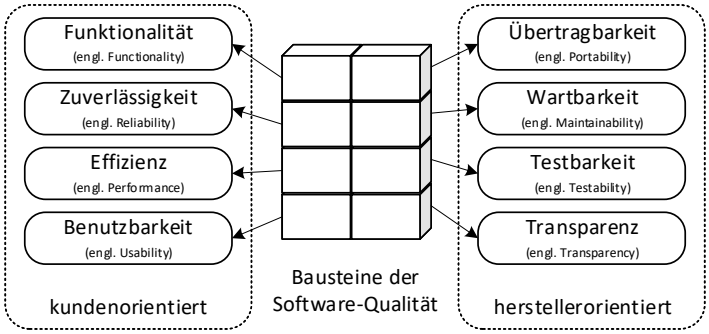


Abbildung 5.8: Qualitätsmerkmale eines Softwareprojekts (Hoffmann, 2013, Seiten 7-9).

Tests mit guter Testabdeckung, als auch Stubs sind mit einem hohen Entwicklungsaufwand verbunden und konnten daher nur für die wichtigsten Komponenten angewandt werden. Hilfreich ist es auch, die Teilkomponenten erst einzeln zu Testen und anschließend mittels Integrationstests Stück für Stück zu einem Gesamtsystem zusammenzufügen.

Da die Implementierung in einer Simulationsumgebung stattfindet, wo die Laufzeit von Methoden und Komponenten keine entscheidende Rolle spielt, wurde der Effizienz der Implementierung wenig Aufmerksamkeit geschenkt. Die Benutzbarkeit wurde mithilfe eines Beispielszenarios geprüft, welches in Kapitel 6 vorgestellt wird.

Während der Qualitätssicherung kamen die üblichen Probleme beim Testen von verteilten Anwendungen zum Vorschein, wie die Reihenfolge von asynchronen Datenübertragungen. In der Realität haben sich Testmethoden, wie das Schritt für Schritt Debuggen mithilfe des in OMNET++ integrierten Werkzeugs oder Textausgaben als Hilfreich erwiesen um schnell die Fehlerquellen auszumachen.

Wünschenswert wäre außerdem eine Analyse des Zeitverhaltens gewesen, die allerdings aufgrund der Grenzen der Simulationsumgebung nicht durchgeführt werden konnte. Während der Recherchen wurden auch verschiedenen Werkzeuge und Verfahren aus dem Forschungsbereich automatischer Middleware Tests, wie DNetSpec von Ishihara u. a. (2017) oder BPEL Engines von Harrer u. a. (2014) aufgefunden. Diese Werkzeuge wären bei der Herstellung einer Marktreifen Middleware sehr nützlich.

6 Evaluation

Da der Kern dieser Arbeit die Konzeption und der Entwurf einer zentralisierten dienstorientierten Middleware-Lösung zur Unterstützung von Dienstgütevereinbarungen ist, wird zur Auswertung das entworfene Konzept (siehe Kapitel 4 und 5) mit den zu Beginn aufgestellten Anforderungen (siehe Kapitel 3) verglichen. Anschließend werden verschiedene Anwendungsszenarien für die Evaluation in der Simulation ausgearbeitet, in der Simulationsumgebung umgesetzt und ausgewertet.

6.1 Auswertung Middleware-Konzept

Um das Konzept auszuwerten, wird die in Kapitel 5 entworfene Middleware-Lösung mit den in Kapitel 3 aufgestellten Anforderungen verglichen. Insbesondere wird überprüft, ob die Middleware den Aufgaben zukünftiger Kommunikationsarchitekturen in Automobilen (siehe Kapitel 3.1) gewachsen ist. Im Folgenden werden die Anforderungen einzeln evaluiert und abschließend zusammenfassend das Gesamtkonzept bewertet.

- *Anzahl der Funktionen:* Der rasant steigenden Anzahl der in Software realisierten Funktionen tritt die entwickelte Middleware-Lösung mit Hilfe einer SOA entgegen. So können Funktionen, mithilfe einer *Service Discovery*, zur Laufzeit im Netzwerk aufgefunden werden. Außerdem können als Dienst implementierte Softwarekomponenten, dank der Abstraktion von den Kommunikationsprotokollen und der Laufzeitumgebung durch die Middleware, auf verschiedenen ECUs ausgeführt werden. Zusätzlich ist es möglich mehrere Instanzen eines Dienstes gleichzeitig anzubieten (siehe Skalierbarkeit in Kapitel 2.4), um einer großen Anzahl von Anfragen zu begegnen.

- *Heterogene Anforderungen:* Um die heterogenen Anforderungen von Softwarekomponenten an die Kommunikation zu entsprechen, wurde das Konzept der Dienstgütevereinbarung (siehe Kapitel 5.4) umgesetzt. So können Klienten, bei der Nutzung eines Dienstes, die Verbindungseigenschaften ihren Anforderungen entsprechend angeben. Die Middleware erlaubt so einen Dienst anzubieten, der dann im Netzwerk mit verschiedenen Anforderungen genutzt werden kann, ohne einen Mehraufwand beim Dienstanbieter

auszulösen. Die im Entwurf beschriebenen Kommunikationsmuster (siehe Kapitel 5.1) erlauben außerdem verschieden Kommunikationsarten, wie Publish/Subscribe oder RPC, wurden allerdings im jetzigen Stand der Middleware noch nicht implementiert.

- *Steigender Kommunikationsbedarf:* Dem steigenden Bandbreitenbedarf und dem hohen Vernetzungsgrad der Softwarekomponenten im Auto wird mithilfe eines neuen Kommunikationsmediums begegnet. Ethernet bietet sowohl eine sehr hohe (und noch immer ansteigende) Datenrate, als auch eine gute Vernetzung durch verschiedene Protokolle und Multicast Konzepte.

- *Zentralisierung des Kommunikationsmediums:* Die vier in dieser Arbeit entwickelten Dienstklassen (siehe Kapitel 4.1) bilden die Anforderungsprofile aller Softwarekomponenten im Auto ab. So können die verschiedenen Bussysteme durch einen unterschiedlichen Protokollstack für die Dienstklassen (siehe Kapitel 5.2) in einem zentralen Ethernet Netzwerk umgesetzt werden. Mit Hilfe von Gateways (siehe Kapitel **??**) ist es möglich Subsysteme abzuschotten, oder aber Schritt für Schritt an das gemeinsame Netzwerk anzubinden.

- *Plug & Play und Wiederverwendbarkeit:* Dank der Dienstorientierung können neue Dienste einfach integriert und effizient aufgefunden werden. Trotz dieser Dynamik ist es mit Hilfe des Protokollstacks (siehe Kapitel 5.2) möglich, Echtzeitanforderungen zu garantieren. Wenn die Schnittstellen der Dienste geschickt entworfen werden, ist auch eine Wiederverwendbarkeit aufgrund der SOA ohne weiteres möglich.

- *Varianten und Konfigurationen:* Die verschiedenen Konfigurationen der Software im Auto können, dank der Flexibilität der SOA, leicht umgesetzt werden. In der jetzigen Version der entwickelten Middleware ist allerdings kein Versionsabgleich implementiert. Dies könnte allerdings in der QoS-Verhandlung (siehe Kapitel 5.4) umgesetzt werden.

- *Internet of Things:* Der Entwurf der Middleware-Lösung unterstützt verschiedene Web Standards bis hin zu Restful Web Services. Auf diese Weise wird den Herausforderungen des IoT begegnet. Allerdings sind in der jetzigen Version der Middleware noch keine Web Services umgesetzt.

- *Sicherheit:* Da die Sicherheitsanforderungen, wie in Kapitel 3.4 beschrieben, außen vorgelassen wurde, kann sie hier auch nicht bewertet werden. Allerdings ist es nötig verschiedene Sicherheitskonzepte in die Middleware zu integrieren, wie z.B. die Möglichkeit von Verschlüsselung oder Virtualisierung von Subsystemen durch Software Defined Networking.

Der Entwurf realisiert alle aufgestellten Anforderungen, bis auf die der Sicherheitskonzepte. Allerdings sind noch nicht alle im Entwurf beschriebenen Konzepte in der jetzigen Version der Middleware umgesetzt. Bis auf die Sicherheitsanforderungen, ist das entwickelte dienstorientierte Middleware Konzept mit Dienstgütevereinbarungen den Aufgaben der Zukunft gewachsen.

6.2 Szenarien

Für die Evaluation in der Simulation wurden die folgenden vier Anwendungsszenarien ausgearbeitet:

1. *Kleines Netzwerk:* Der Zweck dieses Szenarios ist die Visualisierung der Kommunikation und des Verbindungsaufbaus. Es wird ein kleines Netzwerk aus drei Knoten aufgebaut. Auf einem Knoten wird ein Publisher-Dienst angeboten, welcher von den Subscriber-Anwendungen auf den anderen Knoten abonniert wird. Beide Subscriber fordern mit Hilfe der QoS-Verhandlung ein unterschiedliches Dienstprofil an.

2. *CAN-Gateway Netzwerk:* Mit diesem Szenario soll die Integration von Subnetzen anderer Bussysteme, mit Hilfe eines Gateways, überprüft werden. Es wird ein Ethernet Backbone[1] aufgebaut, an dem drei Knoten angeschlossen werden. Zwei dieser Knoten setzen ein Gateway zu einem CAN-Bus um. Der eine bietet die vom CAN-Bus empfangenen Daten, als Publisher-Dienst im Netzwerk an. Das zweite Gateway realisiert eine Subscriber-Anwendung mit Echtzeit Anforderungen, welche die Daten des abonnierten Dienstes des Ethernet Backbone, an den CAN-Bus weiterleitet. Der dritte Knoten realisiert eine Subscriber-Anwendung eines Bordcomputers, der ebenfalls die Daten des Publisher-Dienstes abonniert, jedoch auf einem anderen Anforderungsprofil. Diese Simulation zeigt, dass es mit Hilfe der Middleware möglich ist, zwei CAN-Bus Subsysteme so zu verbinden, das Dienste aus beiden Systemen miteinander kommunizieren können. Ein Beispiel für dieses Szenario ist die Vernetzung des Fahrwerksubsystems mit dem der aktiven Sicherheitsfunktionen und einer Anbindung des Bordcomputers für die Informationsaufbereitung für den Fahrer.

3. *Reales Autonetzwerk:* Ziel dieses Szenarios ist es, ein vollständiges reales Autonetzwerk, auf Basis der entwickelten Middleware, in der Simulation zu realisieren. So wird gezeigt, ob alle Dienste eines Autos mit Hilfe der vier Anforderungsklassen und der Middleware umzusetzen sind.

[1] Backbone (deutsch: Rückgrat, Basisnetz) bezeichnet den zentralen Kernbereich eines Kommunikationsnetzes.

4. *Probleme und Engpass:* Ziel dieses Szenarios ist es zu testen, welche Probleme bei der Middleware basierten Kommunikation auftreten, falls Engpässe und Probleme die Kommunikation beeinflussen. Dies soll Simulieren, wie Beispielsweise mit Nachrichten Überschwemmung aus einem Subsystem umgegangen wird.

6.3 Umsetzung in der Simulation

Aus Zeitgründen konnten leider nur die ersten beiden Szenarien in der Simulationsumgebung OMNeT++ mit den beschriebenen Erweiterungen (siehe Kapitel 2.6) umgesetzt werden. Diese Umsetzung wird im Folgenden beschrieben.

Im ersten Szenario wurde ein kleines Netzwerk entsprechend der Beschreibung aus Kapitel 6.2 aufgebaut. Dieses Netzwerk ist in Abbildung 6.1 dargestellt. Der *Publisher* bietet seinen Dienst über das Netzwerk an, welcher dann vom den beiden Subscribern abonniert wird. Der *RTSubscriber* beantragt die Realtime Service Kommunikationsklasse, der *STDSubscriber* beantragt die Standard IP-based Service Klasse.

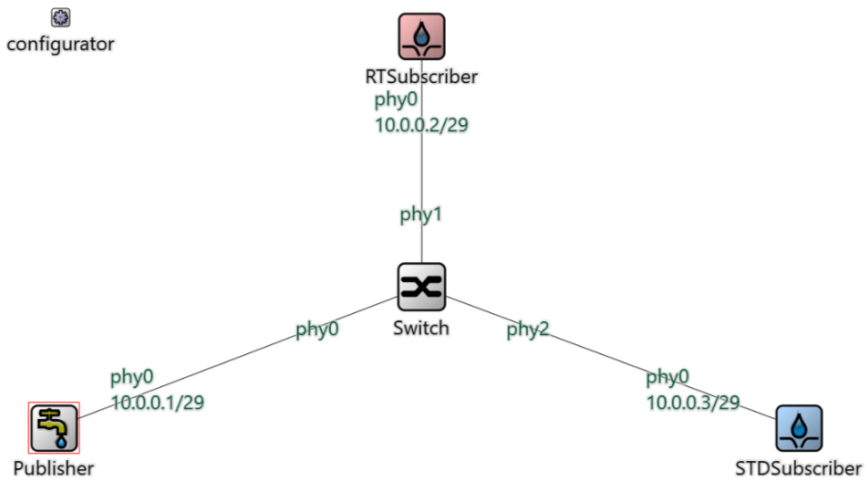
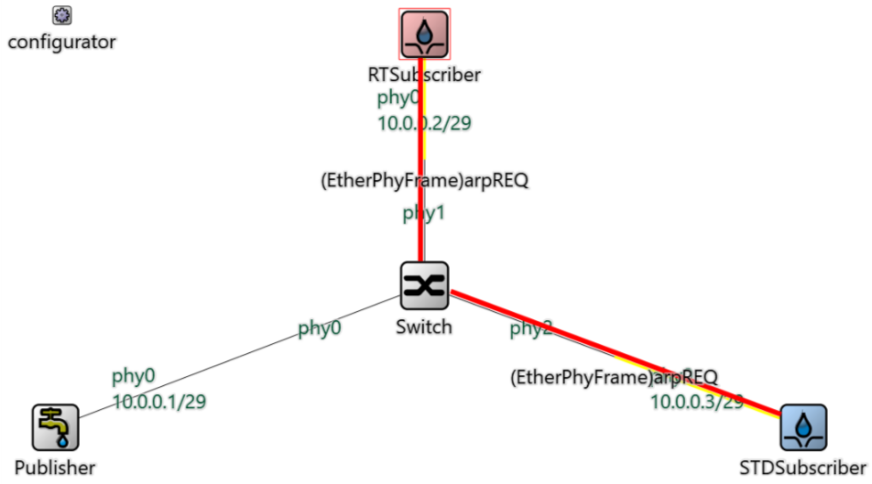
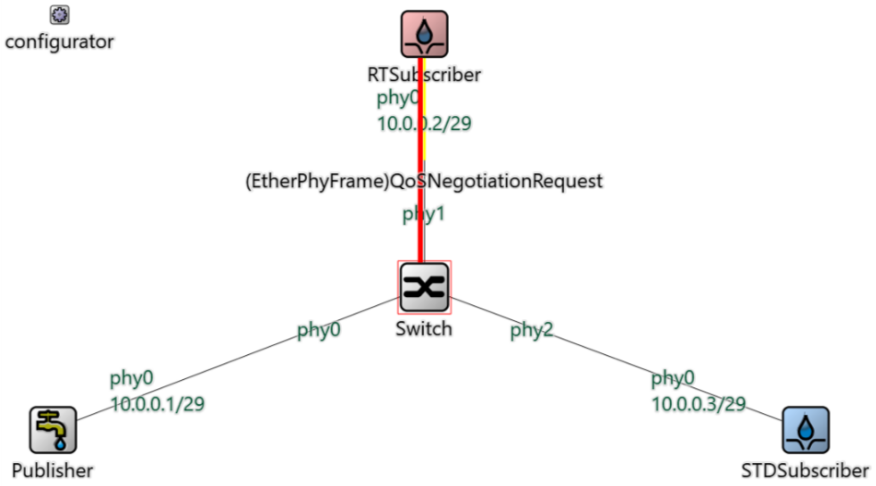


Abbildung 6.1: Aufbau des Netzwerks des ersten Szenarios.

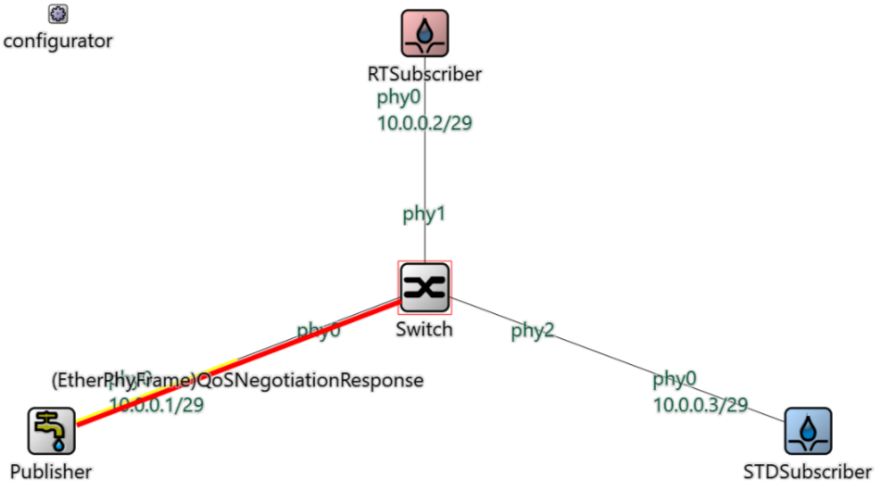
Anhand der Abbildungen in 6.2, kann der Ablauf des Verbindungsaufbaus bis hin zu ersten Übertragung nachvollzogen werden. Da die Nachrichten des Quality-of-Service Negotiation Protokol sowohl für den RTS Subscriber als auch für den SIPS Subscriber identisch ablaufen, werden nur die des RTS Subscribers dargestellt. Im Folgenden sind die Zwischenschritte beschrieben:



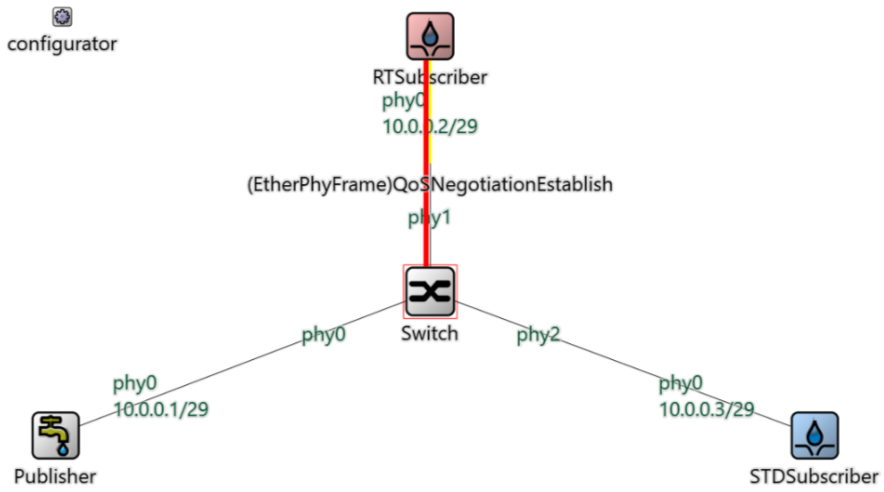
(a) Address Resolution Protocol (ARP)-Request, gefolgt von einer ARP-Response des Publishers



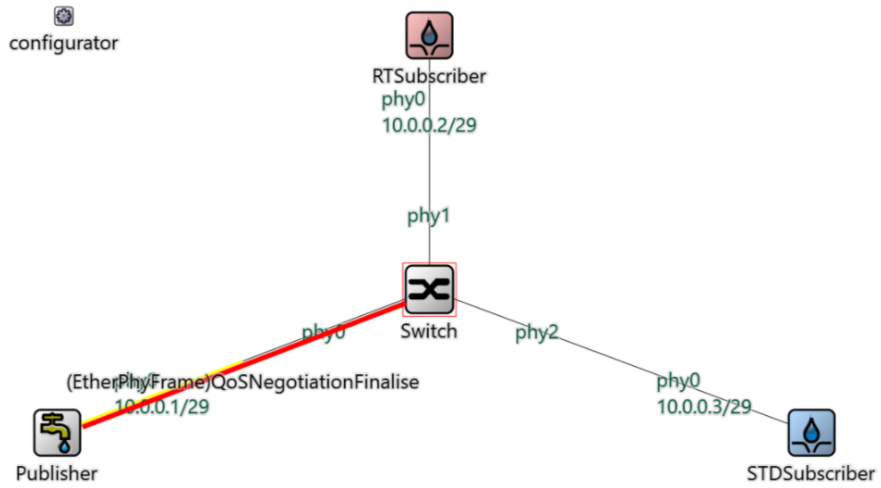
(b) QoSNP-Request



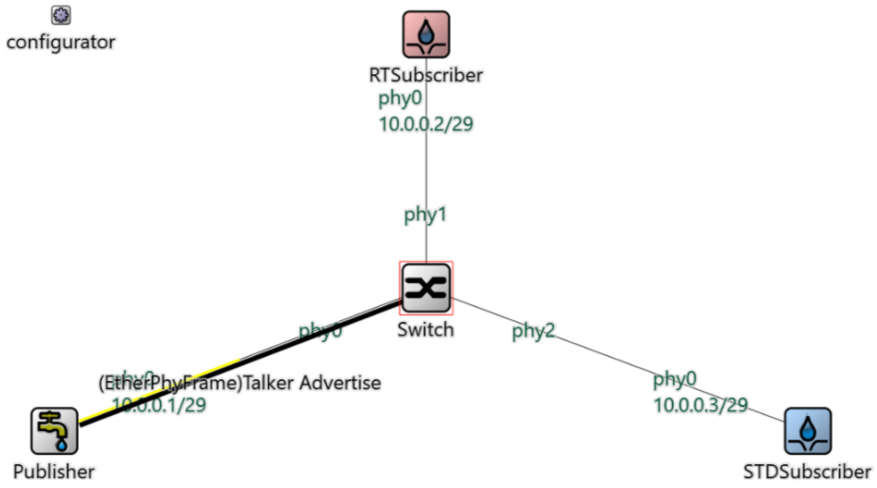
(c) QoSNP-Response



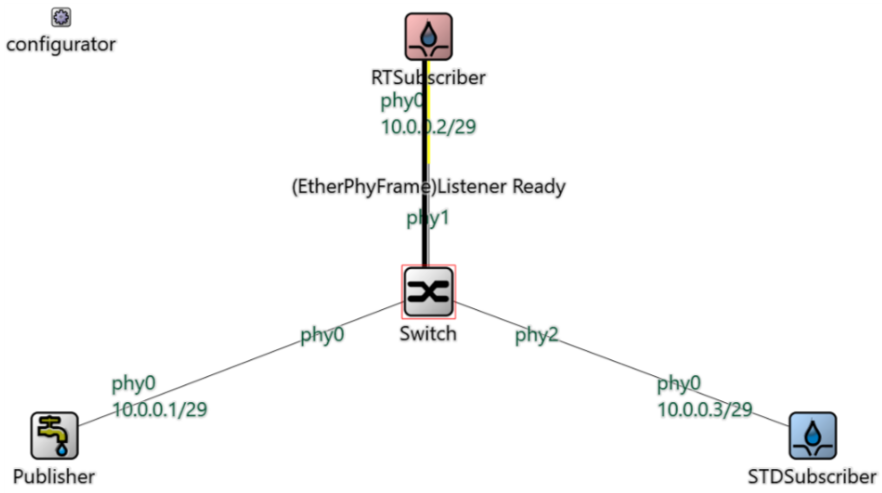
(d) QoSNP-Establish



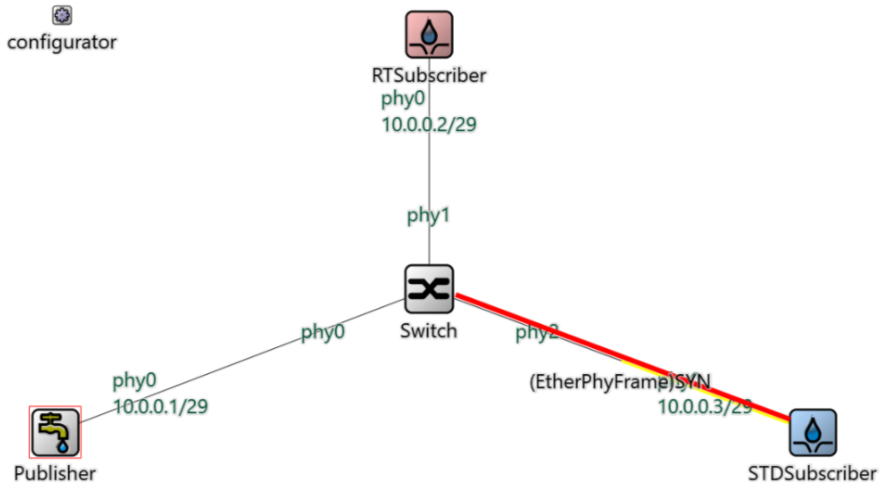
(e) QoSNP-Finalise



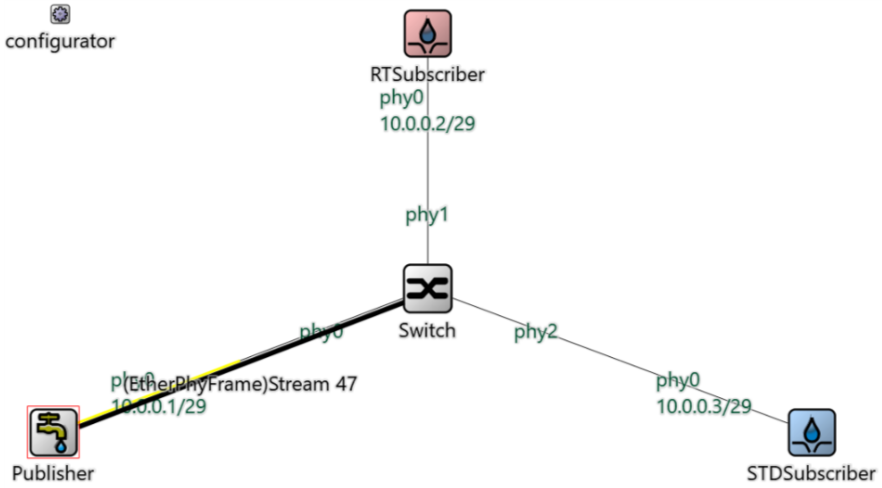
(f) SRP Talker Advertise



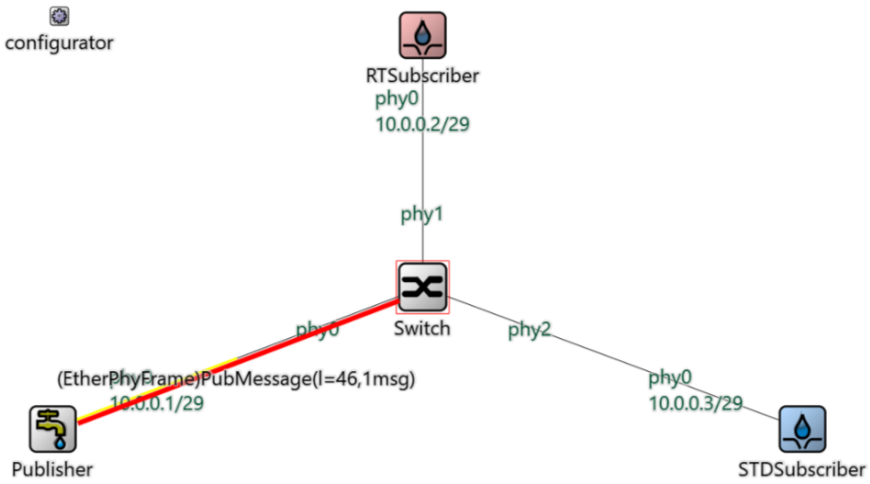
(g) SRP Listener Ready



(h) TCP Handshake



(i) Übertragung des AVB Streams



(j) Übertragung der TCP Nachrichten

Abbildung 6.2: Kommunikationsverlauf des ersten Szenarios.

1. *ARP:* Da das QoSNP auf UDP basiert, müssen mit Hilfe des ARP die IP-Adressen der Kommunikationspartner herausgefunden werden, bevor eine Nachricht übertragen werden kann. Dies geschieht mittels eines ARP-Requests gefolgt von einer ARP-Resonse des Zuständigen Knotens (siehe Abbildung 6.2a).

2. *QoSNP-Request:* Die Subscriber senden einen QoSNP-Request, der ihre QoS an die Kommunikation enthält (siehe Abbildung 6.2b).

3. *QoSNP-Response:* Der Publisher sendet eine QoSNP-Response, der die QoS des Subscribers akzeptiert oder ablehnt (siehe Abbildung 6.2c).

4. *QoSNP-Establish:* Wenn die QoS des Subscribers akzeptiert wurden, sendet dieser eine QoSNP-Establish Nachricht und fordert damit die Vorbereitung der Verbindung des Publishers mit den Besprochenen QoS an (siehe Abbildung 6.2d).

5. *QoSNP-Finalise:* Der Publisher bereitet die Verbindung im Falle eines RTS Subscribers vor, indem er einen Endpunkt erstellt, der einen AVB Stream eröffnet und mit Hilfe eines *Talker Advertise* (siehe Abbildung 6.2f) des SRP bekannt macht. Im Falle eines SIPS Subscribers erstellt der Publisher einen Endpunkt, der ein *TCP Socket* eröffnet und auf Anfragen wartet. In beiden Fällen sendet der Publisher anschließend die Verbindungsdetails des Endpunktes als Connection Specific Information (Connection Specific Information) in einer QoSNP-Finalise Nachricht an den Subscriber (siehe Abbildung 6.2e).

6. *SRP Listener Ready:* Der AVB basierte RTS Subscriber nutzt achließend die Connection Specific Information (CSI), um einen AVB Listener Endpunkt zu erstellen. Daraufhin sendet er ein Listener Ready des SRP, um den Stream zu abonnieren (siehe Abbildung 6.2g).

7. *TCP Handshake:* Der TCP basierte SIPS Subscriber nutzt die CSI, um einen TCP Endpunkt zu erstellen und sich mit dem Socket des Publishers zu verbinden. Dies geschieht mittels des *TCP Handshakes*, bei dem eine *SYN* Anfrage des Subscribers (siehe Abbildung 6.2h) gefolgt von einem *SYN+ACK* des Publishers und eines abschließenden *ACK* des Subscribers die Verbindung der beiden Endpunkte aufbaut.

8. *Übertragung des AVB Streams:* Wenn ein AVB Listener beim Publisher registriert ist, beginnt dieser mit dem Senden des AVB Streams an den RTS Subscriber (siehe Abbildung 6.2i).

9. *Übertragung der TCP Nachrichten:* Wenn eine TCP Verbindung beim Publisher aufgebaut wurde, beginnt dieser mit dem Senden der Nachrichten zum SIPS Subscriber (siehe Abbildung 6.2j). Diese werden im TCP jeweils mit einem *ACK* des Subscribers bestätigt.

Für das zweite Szenario wird ein Gateway Netzwerk aufgebaut, welches in Abbildung 6.3 dargestellt ist. Über den *ethswitch* sind die drei Knoten *InfotainmentSystem*, *GatewaySource* und *GatewaySink* des Ethernet Backbone miteinander verbunden. Der Knoten *GatewaySource* ist außerdem mit dem CAN-Bus des *Fahrwerk* verbunden, an dem der CAN Knoten *Reifen* angeschlossen ist Der Knoten *GatewaySink* mit dem CAN-Bus der *Fahrdynamik*, an dem der CAN Knoten *Controller* angeschlossen ist. *GatewaySource* bietet den Dienst *Reifendruck* im Netzwerk an und die beiden anderen Ethernet Knoten abonnieren diesen, *InfotainmentSystem* als SIPS Subscriber und *GatewaySink* als RTS Subscriber. So werden die Daten des CAN Knoten *Reifen* bis zum CAN Knoten *Controller* durch die beiden Gateways übertragen.

Leider konnte dieses Szenario, aufgrund eines Fehlers im *FiCo4OMNeT* Framework, nicht vollständig simuliert werden. Das CAN-Bus Modul erzeugt einen Laufzeitfehler, der im Quellcode 6.1 markiert ist. Aufgrund eines Fehlers beim Freigeben von Speicher, werden keine Daten über den CAN-Bus übertragen. Dieser Fehler entsteht auch in den Beispielen aus dem *FiCo4OMNeT* Framework und wird vermutlich etwas mit einem Versionskonflikt zu tun haben. Allerdings konnte der Fehler aus Zeitmangel nicht behoben werden, weshalb das Szenario nicht vollständig umgesetzt wurde.

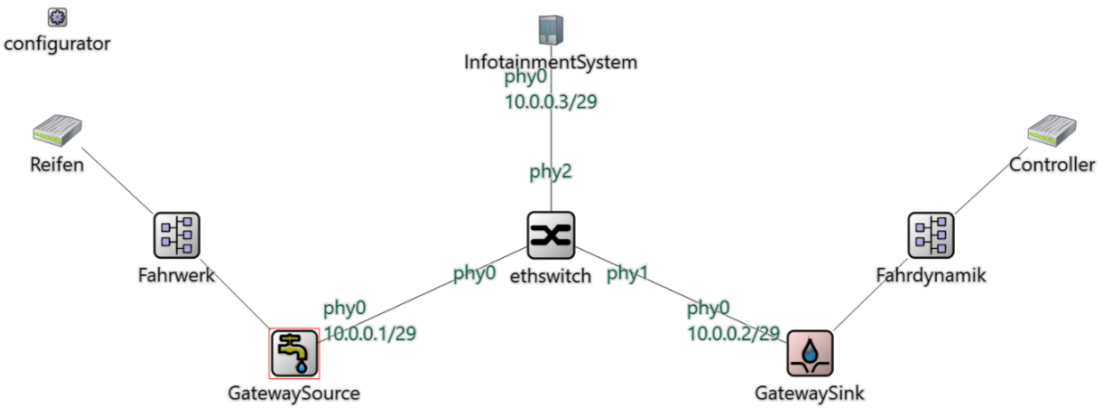


Abbildung 6.3: Aufbau des Netzwerks des zweiten Szenarios.

```
void CanBusLogic::sendingCompleted() {
    colorIdle();
    emit(stateSignal, static_cast<long>(State::IDLE));
    CanOutputBuffer* controller = check_and_cast<
       CanOutputBuffer*>(sendingNode);
    controller->sendingCompleted();
    for (unsigned int it = 0; it != eraseids.size(); it++) {
        ids.erase(eraseids.at(it));
        delete *(eraseids.at(it)); //--> error freeing memory.
    }
```

Listing 6.1: Fehler in FiCo4OMNeT beim Freigeben eines Pointers.

6.4 Auswertung

Den Simulationen der beiden Szenarien kann man entnehmen, dass die Middleware sich wie erwartet Verhält. Sie setzt also das Konzept einer zentralisierten dienstorientierten Middleware-Lösung zur Unterstützung von Dienstgütevereinbarungen um. Die Szenarien haben einige Einsatzgebiete gezeigt, wobei die komplexeren Szenarien aus Zeitmangel nicht umgesetzt werden konnten. Außerdem gab es einen Fehler in einem Framework der Simulationsumgebung. In weiteren Arbeitsschritten sollten also die Probleme gelöst und die weiteren Szenarien implementiert werden. Außerdem wäre es wichtig das Zeitverhalten der Middleware zu analysieren. Dies geht in OMNeT++ allerdings nicht. Daher müsste ein Demonstrator aufgebaut werden.

7 Fazit und Ausblick

In dieser Arbeit wurde auf Basis der Forschung im Bereich zentralisierte Netzwerke mit Ethernet Echtzeiterweiterungen eine dienstorientierte Kommunikationsarchitektur entwickelt, die auf die Anforderungen des Autos zugeschnitten ist. Das beschriebene Middleware-Konzept, ist in der Lage die heterogenen Anforderungen mit einem variablen Protokollstack zu bedienen und erlaubt Diensten diese Anforderungen in Dienstgütevereinbarungen auszuhandeln. Dieses Konzept wurde in einer Machbarkeitsstudie in einer Simulationsumgebung an verschiedenen Szenarien ausgewertet. Im Folgenden werden die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick gegeben.

7.1 Ergebnisse

In dieser Arbeit wurde hergeleitet, warum aktuelle Kommunikationsarchitekturen im Fahrzeug, den Herausforderungen der Zukunft nicht gewachsen sind. Es wurden die Aufgaben einer zukünftigen Kommunikationsarchitektur ermittelt und Ansätze zur Lösung beschrieben. Auf der Basis verschiedener verwandter Arbeiten wurden Konzepte zur Klassifizierung von Automobilen Diensten, dienstorientierte Middleware Lösungen und Strategien zur Verbreitung und Migration entwickelt. Aufbauend auf diesen Konzepten wurde eine Middleware Architektur aus vier Komponenten entworfen. Dieser Entwurf setzt die Dienstklassen in einem Variablen Protokollstack um und ermöglicht die Aushandlung der Dienstgüte mithilfe eines Protokolls zur Dienstgüteverhandlung. Der Entwurf setzt besonders das Konzept der Modularisierung, die heterogenen Anforderungen Automobiler Anwendungen und die Dienstorientierung in einem zentralen Ethernet Netzwerk um. Sicherheitskonzepte wie Verschlüsselung und Virtualisierung wurden nicht umgesetzt.

Die Simulation wurde leider größtenteils zu Visualisierungszwecken eingesetzt, konnte allerdings zeigen, dass die Middleware wie beschrieben Funktioniert. Leider sind einige der implementierten Komponenten stark auf die Simulation spezialisiert und können nicht direkt in die reale Umgebung eines Demonstrators übertragen werden.

7.2 Ausblick

Als Fortsetzung dieser Arbeit ist besonders die nicht abgeschlossene Evaluation durchzuführen. Die Simulation von verschiedenen Szenarien großer Netzwerke in Fehlersituationen, kann Schwachstellen des Konzeptes offenlegen. Außerdem können auf Basis dieser Informationen Design-Richtlinien festgehalten werden, die z.B. einen maximal nutzbaren Bandbreitenanteil vorschreiben. Des Weiteren wäre die Umsetzung der Middleware in der realen Umgebung eines Demonstrators hilfreich, um Laufzeitung und Verzögerungen durch die Middleware zu ermitteln und die Wechselwirkungen mit dem Gesamtsystem zu analysieren.

Aufbauend auf einer zentralisierten dienstorientierten Kommunikationsarchitektur, können dann Konzepte zur Anordnung von Diensten (vgl. Jobst und Prehofer (2016)) und der effizienten Verteilung von Diensten im Netzwerk (vgl. Scholz u. a. (2009a)) angewandt werden. Diese Prozesse könnten dann in einem weiteren Schritt automatisiert werden, so das neue Komponenten effizient und voll automatisch integriert werden können (vgl. Buckl u. a. (2009)).

Ein weiteres Projekt ist die Absicherung dieser Kommunikationsarchitektur. In einer Plattform Lösung müssen verschiedene Sicherheitskonzepte, wie zum Beispiel Virtualisierung von ECUs oder Netzwerken (Software Defined Networking) umgesetzt werden.

Literaturverzeichnis

[PlattformAuto2018] : *Automobil - Wie verändern digitale Plattformen die Automobilwirtschaft?*. – URL http://plattform-maerkte.de/auto/. – Zugriffsdatum: 2018-04-10

[AUTOSAR] : *AUTOSAR – The Standardized Software Architecture - GI - Gesellschaft für Informatik e.V.*. – URL https://www.gi.de/service/informatiklexikon/detailansicht/article/autosar-the-standardized-software-architecture.html. – Zugriffsdatum: 2016-05-14

[Abdelzaher u. a. 2000] Abdelzaher, T. F. ; Atkins, E. M. ; Shin, K. G.: QoS negotiation in real-time systems and its application to automated flight control. In: *IEEE Transactions on Computers* 49 (2000), Nov, Nr. 11, S. 1170–1183. – ISSN 0018-9340

[all-electronics.de] all-electronics.de: *Security: Keine Chance zur Manipulation.* – URL http://www.all-electronics.de/security-keine-chance-zur-manipulation/. – Zugriffsdatum: 2018-04-10

[Aurrecoechea u. a. 1998] Aurrecoechea, Cristina ; Campbell, Andrew T. ; Hauw, Linda: A survey of QoS architectures. In: *Multimedia systems* 6 (1998), Nr. 3, S. 138–151

[AUTOSAR Standard 616 2016] AUTOSAR Standard 616: *Specification of Service Discovery*. 2016

[AUTOSAR Standard 637 2014] AUTOSAR Standard 637: *Example for a Serialization Protocol (SOME/IP)*. 2014

[AUTOSAR Standard 660 2016] AUTOSAR Standard 660: *Specification of SOME/IP Transformer*. 2016

[AUTOSAR Standard 809 2016] AUTOSAR Standard 809: *Specification of Module SOME/IP Transport Protocol*. 2016

[Barry und Dick 2013] Barry, Douglas K. ; Dick, David: *Web services, service-oriented architectures, and cloud computing: the savvy manager's guide*. Second edition. San Francisco, Calif. : Oxford : Morgan Kaufmann ; Elsevier Science [distributor], 2013 (The Savvy manager's guides). – ISBN 978-0-12-398357-2

[Bean 2010] Bean, James: *SOA and Web services interface design: principles, techniques, and standards*. Amsterdam ; Boston : Morgan Kaufmann/Elsevier, 2010 (Morgan Kaufmann OMG Press). – ISBN 978-0-12-374891-1

[Bello 2014] Bello, L. L.: Novel trends in automotive networks: A perspective on Ethernet and the IEEE Audio Video Bridging. In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, Sept 2014, S. 1–8. – ISSN 1946-0740

[Bello 2011] Bello, Lucia L.: The Case for Ethernet in Automotive Communications. In: *SIGBED Rev.* 8 (2011), Dezember, Nr. 4, S. 7–15. – ISSN 1551-3688

[Broy 2006] Broy, Manfred: Challenges in Automotive Software Engineering. In: *Proceedings of the 28th International Conference on Software Engineering*. New York, NY, USA : ACM, 2006 (ICSE '06), S. 33–42. – ISBN 1-59593-375-1

[Broy 2008] Broy, Manfred: *Model-Driven Development of Reliable Automotive Services - Second Automotive Software Workshop, ASWSD 2006, San Diego, CA, USA, March 15-17, 2006, Revised Selected Papers*. 2008. Aufl. Berlin Heidelberg : Springer Science & Business Media, 2008. – ISBN 978-3-540-70929-9

[Buckl u. a. 2012] Buckl, C. ; Camek, A. ; Kainz, G. ; Simon, C. ; Mercep, L. ; Stähle, H. ; Knoll, A.: The software car: Building ICT architectures for future electric vehicles. In: *2012 IEEE International Electric Vehicle Conference*, March 2012, S. 1–8

[Buckl u. a. 2009] Buckl, C. ; Sommer, S. ; Scholz, A. ; Knoll, A. ; Kemper, A. ; Heuer, J. ; Schmitt, A.: Services to the Field: An Approach for Resource Constrained Sensor/Actor Networks. In: *2009 International Conference on Advanced Information Networking and Applications Workshops*, May 2009, S. 476–481

[Burgio u. a. 2016] Burgio, P. ; Bertogna, M. ; Olmedo, I. S. ; Gai, P. ; Marongiu, A. ; Sojka, M.: A Software Stack for Next-Generation Automotive Systems on Many-Core Heterogeneous Platforms. In: *2016 Euromicro Conference on Digital System Design (DSD)*, Aug 2016, S. 55–59

[Chakraborty u. a. 2012] Chakraborty, Samarjit ; Lukasiewycz, Martin ; Buckl, Christian ; Fahmy, Suhaib ; Chang, Naehyuck ; Park, Sangyoung ; Kim, Younghyun ; Leteinturier, Patrick ; Adlkofer, Hans: Embedded Systems and Software Challenges in Electric Vehicles. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. San Jose, CA, USA : EDA Consortium, 2012 (DATE '12), S. 424–429. – URL http://dl.acm.org/citation.cfm?id=2492708.2492815. – ISBN 978-3-9810801-8-6

[CoRE Working Group a] CoRE Working Group: *CoRE Researchgroup*. – URL https://core-researchgroup.de/

[CoRE Working Group b] CoRE Working Group: *CoRE Simulation Models for Real-time Networks*. – URL http://sim.core-rg.de

[Cucinotta u. a. 2009] Cucinotta, T. ; Mancina, A. ; Anastasi, G. F. ; Lipari, G. ; Mangeruca, L. ; Checcozzo, R. ; Rusina, F.: A Real-Time Service-Oriented Architecture for Industrial Automation. In: *IEEE Transactions on Industrial Informatics* 5 (2009), Aug, Nr. 3, S. 267–277. – ISSN 1551-3203

[Datta u. a. 2016] Datta, S. K. ; Costa, R. P. F. D. ; Härri, J. ; Bonnet, C.: 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, S. 1–6

[Fehling u. a. 2014] Fehling, Christoph ; Leymann, Frank ; Retter, Ralph ; Schupeck, Walter ; Arbitter, Peter: *Cloud Computing Patterns - Fundamentals to Design, Build, and Manage Cloud Applications*. Berlin Heidelberg : Springer Science & Business Media, 2014. – ISBN 978-3-709-11568-8

[fortiss GmbH 2011] fortiss GmbH: The software car: Information and communication technology (ict) as an engine for the electromobility of the future / fortiss GmbH. mar 2011. – Forschungsbericht. summary of results of the ëCar ICT System Architecture for Electromobility"research project sponsored by the Federal Ministry of Economics and Technology

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

[Fürst und Bechter 2016] Fürst, S. ; Bechter, M.: 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, S. 215–217

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

[Gopu u. a. 2016] Gopu, G. L. ; Kavitha, K. V. ; Joy, J.: Service Oriented Architecture based connectivity of automotive ECUs. In: *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, March 2016, S. 1–4

[Gu u. a. 2004] Gu, T. ; Pung, H. K. ; Zhang, D. Q.: Toward an OSGi-based infrastructure for context-aware applications. In: *IEEE Pervasive Computing* 3 (2004), Oktober, Nr. 4, S. 66–74. – ISSN 1536-1268

[Gupta und Jha 2015] Gupta, A. ; Jha, R. K.: A Survey of 5G Network: Architecture and Emerging Technologies. In: *IEEE Access* 3 (2015), S. 1206–1232

[Harrer u. a. 2014] Harrer, S. ; Röck, C. ; Wirtz, G.: Automated and Isolated Tests for Complex Middleware Products: The Case of BPEL Engines. In: *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, March 2014, S. 390–398

[Hill u. a. 2010] Hill, J. ; Sutherland, H. ; Stodinger, P. ; Silveria, T. ; Schmidt, D. C. ; Slaby, J. ; Visnevski, N.: OASIS: A Service-Oriented Architecture for Dynamic Instrumentation of Enterprise Distributed Real-Time and Embedded Systems. In: *2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, May 2010, S. 10–17. – ISSN 1555-0885

[Hoffmann 2013] Hoffmann, D.W.: *Software-Qualität*. Springer Berlin Heidelberg, 2013 (eXamen.press). – URL https://books.google.de/books?id=XZEuBAAAQBAJ. – ISBN 9783642357008

[IEEE 802.1 TSN Task Group] IEEE 802.1 TSN Task Group: *IEEE 802.1 Time-Sensitive Networking Task Group*. – URL https://1.ieee802.org/tsn/

[Institute of Electrical and Electronics Engineers 2008] Institute of Electrical and Electronics Engineers: IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. In: *IEEE Std 802.3-2008 (Revision of IEEE Std 802.3-2005)* (2008), Dec, S. 1–2977

[Institute of Electrical and Electronics Engineers 2010] Institute of Electrical and Electronics Engineers: IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP). In: *IEEE Std 802.1Qat-2010 (Revision of IEEE Std 802.1Q-2005)* (2010), Sept, S. 1–119

[Institute of Electrical and Electronics Engineers 2011] Institute of Electrical and Electronics Engineers: IEEE Standard for Local and metropolitan area networks–Audio Video Bridging (AVB) Systems. In: *IEEE Std 802.1BA-2011* (2011), Sept, S. 1–45

[Institute of Electrical and Electronics Engineers 2014] Institute of Electrical and Electronics Engineers: IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks. In: *IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011)* (2014), Dec, S. 1–1832

[Institute of Electrical and Electronics Engineers 2016] Institute of Electrical and Electronics Engineers: IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic. In: *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q— as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q—/Cor 1-2015)* (2016), March, S. 1–57

[Institute of Electrical and Electronics Engineers 2017] Institute of Electrical and Electronics Engineers: IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing. In: *IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016)* (2017), Sept, S. 1–65

[Ishihara u. a. 2017] Ishihara, S. ; Masahiro, F. ; Akiyama, T.: DNetSpec: A Distributed Network Testing Toolset for Middleware Developers. In: *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)* Bd. 2, July 2017, S. 1–6. – ISSN 0730-3157

[Iwai und Aoyama 2011] Iwai, A. ; Aoyama, M.: Automotive Cloud Service Systems Based on Service-Oriented Architecture and Its Evaluation. In: *2011 IEEE 4th International Conference on Cloud Computing*, July 2011, S. 638–645. – ISSN 2159-6182

[Jammes und Smit 2005] Jammes, F. ; Smit, H.: Service-oriented paradigms in industrial automation. In: *IEEE Transactions on Industrial Informatics* 1 (2005), Feb, Nr. 1, S. 62–70. – ISSN 1551-3203

[Jobst und Prehofer 2016] Jobst, M. E. ; Prehofer, C.: Towards hierarchical information architectures in automotive systems. In: *2016 3rd International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC)*, April 2016, S. 41–46

[Kugele u. a. 2017] Kugele, S. ; Obergfell, P. ; Broy, M. ; Creighton, O. ; Traub, M. ; Hopfensitz, W.: On Service-Orientation for Automotive Software. In: *2017 IEEE International Conference on Software Architecture (ICSA)*, April 2017, S. 193–202

[Lim u. a. 2011] Lim, H. T. ; Völker, L. ; Herrscher, D.: Challenges in a future IP/Ethernet-based in-car network for real-time applications. In: *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2011, S. 7–12. – ISSN 0738-100x

[Menascé u. a. 2007] Menascé, Daniel ; Ruan, Honglei ; Gomaa, Hassan: QoS management in service-oriented architectures. 64 (2007), 08, S. 646–663

[Meyer u. a. 2013] Meyer, P. ; Steinbach, T. ; Korf, F. ; Schmidt, T. C.: Extending IEEE 802.1 AVB with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic. In: *2013 IEEE Vehicular Networking Conference*, Dec 2013, S. 47–54. – ISSN 2157-9857

[OASIS 2009] OASIS: *Devices Profile for Web Services (DPWS)*. 2009. – URL http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01. – Zugriffsdatum: 2018-04-10

[Object Management Group 2012] Object Management Group: *Common Object Request Broker Architecture (CORBA)*. 2012. – URL https://www.omg.org/spec/CORBA/. – Zugriffsdatum: 2018-05-13

[Object Management Group 2015] Object Management Group: Data Distribution Service / OMG. URL http://www.omg.org/spec/DDS/1.4, mar 2015 (DDS™ 1.4). – Standard. Access: 2018-02-08

[OpenSim Ltd.] OpenSim Ltd.: *INET Framework*. – URL https://inet.omnetpp.org/

[OSGi Alliance 2018] OSGi Alliance: *Open Services Gateway initiative (OSGi)*. 2018. – URL https://osgi.org/download/r7/osgi.core-7.0.0.pdf. – Zugriffsdatum: 2018-05-13

[Practices 2009] Practices, Team Microsoft Patterns ⁝ *.NET Application Architecture Guide - Version 2.0*. Microsoft Press, 2009. – ISBN 978-0-735-62710-9

[Pretschner u. a. 2007] Pretschner, A. ; Broy, M. ; Kruger, I. H. ; Stauner, T.: Software Engineering for Automotive Systems: A Roadmap. In: *Future of Software Engineering, 2007. FOSE '07*, May 2007, S. 55–71

[Richardson und Ruby 2007] Richardson, Leonard ; Ruby, Sam: *Web-Services mit REST* -. Köln : O'Reilly Germany, 2007. – ISBN 978-3-897-21727-0

[Scholz u. a. 2009a] Scholz, A. ; Gaponova, I. ; Sommer, S. ; Kemper, A. ; Knoll, A. ; Buckl, C. ; Heuer, J. ; Schmitt, A.: Efficient communication in control-oriented embedded networks. In: *2009 IEEE Conference on Emerging Technologies Factory Automation*, Sept 2009, S. 1–8. – ISSN 1946-0740

[Scholz u. a. 2009b] Scholz, A. ; Gaponova, I. ; Sommer, S. ; Kemper, A. ; Knoll, A. ; Buckl, C. ; Heuer, J. ; Schmitt, A.: ϵSOA - Service Oriented Architectures adapted for embedded networks. In: *2009 7th IEEE International Conference on Industrial Informatics*, June 2009, S. 599–605. – ISSN 1935-4576

[Schäuffele und Zurawka 2012] Schäuffele, Jörg ; Zurawka, Thomas: *Automotive Software Engineering: Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen.* Springer-Verlag, November 2012. – ISBN 978-3-8348-2470-7

[Society of Automotive Engineers - AS-2D Time Triggered Systems and Architecture Committee 2011] Society of Automotive Engineers - AS-2D Time Triggered Systems and Architecture Committee: *Time-Triggered Ethernet AS6802.* SAE Aerospace. November 2011. – URL http://standards.sae.org/as6802/

[Sommer u. a. 2013] Sommer, S. ; Camek, A. ; Becker, K. ; Buckl, C. ; Zirkler, A. ; Fiege, L. ; Armbruster, M. ; Spiegelberg, G. ; Knoll, A.: RACE: A Centralized Platform Computer Based Architecture for Automotive Applications. In: *2013 IEEE International Electric Vehicle Conference (IEVC)*, Oct 2013, S. 1–6

[Stähle u. a. 2013] Stähle, H. ; Mercep, L. ; Knoll, A. ; Spiegelberg, G.: Towards the deployment of a centralized ICT architecture in the automotive domain. In: *2013 2nd Mediterranean Conference on Embedded Computing (MECO)*, June 2013, S. 66–69. – ISSN 2377-5475

[Steinbach u. a. 2014] Steinbach, Till ; Müller, Kai ; Korf, Franz ; Röllig, René: Real-time Ethernet In-Car Backbones: First Insights into an Automotive Prototype. In: *2014 IEEE Vehicular Networking Conference (VNC).* Piscataway, NJ, USA : IEEE Press, Dezember 2014, S. 137–138. – ISBN 978-1-4799-7659-1

[Technische Universität München 2015] Technische Universität München: *Das Auto als Internet Hardware*. 2015. – URL https://www.tum.de/die-tum/aktuelles/pressemitteilungen/kurz/article/32277/. – Zugriffsdatum: 2018-04-10

[Technische Universität München 2016] Technische Universität München: *Entwicklung innovativer IT-Dienste im Auto mit OSGi*. 2016. – URL https://jaxenter.de/entwicklung-innovativer-it-dienste-im-auto-mit-osgi-32640. – Zugriffsdatum: 2018-04-10

[The C++ Micro Services Blog 2012] The C++ Micro Services Blog: *OSGi and C++*. 2012. – URL http://blog.cppmicroservices.org/2012/03/29/osgi-and-c++/. – Zugriffsdatum: 2018-05-10

[Valls u. a. 2013] Valls, M. G. ; Lopez, I. R. ; Villar, L. F.: iLAND: An Enhanced Middleware for Real-Time Reconfiguration of Service Oriented Distributed Real-Time Systems. In: *IEEE Transactions on Industrial Informatics* 9 (2013), Feb, Nr. 1, S. 228–236. – ISSN 1551-3203

[Völker 2013] Völker, Dr. L.: SOME/IP – Die Middleware für Ethernetbasierte Kommunikation. In: *HANSER automotive Networks/2013* (2013), S. 17–19. – German

[W3C 2006] W3C: *Web Services Description Language (WSDL)*. 2006. – URL https://www.w3.org/TR/2006/CR-wsdl20-20060327/wsdl20-z.pdf. – Zugriffsdatum: 2018-04-10

[W3C 2007] W3C: *Simple Object Access Protocol (SOAP) - Messaging Framework*. 2007. – URL https://www.w3.org/TR/soap12-part1/. – Zugriffsdatum: 2018-04-10

[Wagner u. a. 2014] Wagner, M. ; Zöbel, D. ; Meroth, A.: SODA: Service-Oriented Architecture for Runtime Adaptive Driver Assistance Systems. In: *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, June 2014, S. 150–157. – ISSN 1555-0885

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Glossar

AUTOSAR

AUTomotive Open System ARchitecture (AUTOSAR) ist eine Organisation mit dem Ziel, offene Standards für Steuergeräte und Softwarearchitekturen im Automobil zu schaffen und zu etablieren. Sie besteht aus den größten Partnern der Automobilindustrie . 14, 34, 35, 39, 40, 75

Backbone

Backbone (deutsch: Rückgrat, Basisnetz) bezeichnet den zentralen Kernbereich eines Kommunikationsnetzes. 57, 63

Best-Effort

Best-Effort (deutsch: größte Anstrengung) bezeichnet die geringste Übertragungspriorität in Kommunikationsnetzwerken. Es gibt keine Garantien für Erfolg und Dauer der Übertragung. 4–6, 45, 82

Cloud-Computing

Cloud-Computing (deutsch Rechnerwolke) beschreibt die Bereitstellung von IT-Infrastruktur wie beispielsweise Speicherplatz, Rechenleistung oder Anwendungssoftware als Dienstleistung über das Internet. 10, 28

Gateway

Gateway (deutsch Einfahrt und Ausfahrt) bezeichnet in der Informatik eine Komponente, welche zwischen zwei Systemen eine Verbindung herstellt. 11, 32, 40, 41, 56, 57, 63

Infotainment

Infotainment (zusammengesetzt aus dem englischen information und entertainment) fasst alle Anwendungen und Geräte zusammen, die zur Information oder Unterhaltung der Fahrgäste verbaut sind. 25

Listener

Listener (deutsch: Empfänger, Zuhörer) ist die Senke eines AVB Nachrichtenstroms. 5, 6, 61, 63

M2M

Machine-to-Machine Kommunikation (häufig mit M2M abgekürzt), bezeichnet den automatisierten Informationsaustausch zwischen Maschinen . 7, 10, 19

QoS-Broker

QoS-Broker (deutsch: Dienstgüte Vermittler), ist eine häufig verwendete Middleware-Komponente, die zwischen Anbieter und Konsumenten vermittelt, um einen Dienstgütevertrag zu schließen. 36, 37, 49, 76

Quality-of-Service

Quality-of-Service (QoS) (deutsch: Dienstgüte) bezeichnet die Güte eines Kommunikationsdienstes anhand gewisser Eigenschaften (wie z.B. maximale Latenz) aus Sicht der Anwender. 4, 27, 36, 38, 39, 42, 44–46, 48–50, 56, 57, 62, 76, 80, 82

Qualitätssicherung

Qualitätssicherung in der Software bedeutet die Überprüfung der Erfüllung der festgelegten Anforderungen anhand verschiedener Merkmale und Metriken. 53

RPC

Remote Procedure Call (RPC) (deutsch Entfernter Funktionsaufruf), erlaubt Funktionsaufrufe von im Netzwerk verteilten Objekten. 7–9, 46, 56

Safestate

Safestate (deutsch: Sicherer Zustand), beschreibt beim Auto einen Zustand, in dem trotz einer Fehlfunktion keine Gefährdung der Insassen und Umgebung besteht, dieser besteht üblicherweise darin, an den Fahrbahnrand zu fahren und anzuhalten. 29, 30

Talker

Talker (deutsch: Sender, Sprecher) ist die Quelle eines AVB Nachrichtenstroms. 5, 6, 60, 62

TDMA

TDMA ist ein Zeitmultiplexverfahren bei dem Daten von unterschiedlichen Sendern zu jeweils bestimmten Zeitabschnitten versendet werden. 5–7

XML

Extensible Markup Language (XML) (deutsch Erweiterbare Auszeichnungssprache) ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten im Format einer Textdatei. 8, 9

Abkürzungsverzeichnis

BE	Best-Effort	4–6, 45
QoS	Quality-of-Service	4, 27, 36, 38, 39, 42, 44–46, 48–50, 56, 57, 62, 76, 80
API	Application Programming Interface	9, 36, 42
ARP	Address Resolution Protocol	59, 62
AVB	Audio Video Bridging	5, 6, 11, 38, 44, 46, 61–63, 80
CAN	Controller Area Network	2, 11, 57, 63
CMI	Class Measurement Interval	6
CORBA	Common Object Request Broker Architecture	8
CoRE	Communication over Realtime Ethernet	11
CSI	Connection Specific Information	62, 63
DDS	Data Distribution Service	38, 39, 76
DPWS	Devices Profile for Web Services	9, 36
ECU	Electronic Control Unit	1, 2, 13, 14, 19, 25, 30, 32, 34, 55, 66, 75

HMI	Human-Machine Interface	25
HTTP	Hypertext Transfer Protocol	9, 10, 45
ICT	Information and Communication Technology	1–3, 13, 14, 17–19, 21, 22, 75
IEEE	Institute of Electrical and Electronics Engineers	6
IoT	Internet of Things	2, 11, 17, 19, 56
IP	Internet Protocol	32, 34–36, 39, 45, 58, 62, 84
Java RMI	Jave Remote Method Invocation	8
JMS	Java Message Service	8
LIN	Local Interconnect Network	2
LSM	Local Service Manager	42, 49
LSR	Local Service Registry	42, 44, 49
MOM	Message Oriented Middleware	8
MOST	Media Oriented System Transport	2
OEM	Original Equipment Manufacturer	14
OSGi	Open Services Gateway initiative	33, 34, 75
OSI	Open Systems Interconnection	4, 44
QoSM	Quality-of-Service Manager	42, 44, 48, 49
QoSNP	Quality-of-Service Negotiation Protokol	46, 48, 49, 51, 58–60, 62, 76
RC	Rate-Constrained	5

ReST	Representational State Transfer	10, 45
RTS	Realtime Service	32, 44, 58, 62, 63
SEF	Service Endpoint Factory	42, 44
SIPS	Standard IP-based Service	32, 35, 45, 58, 62, 63
SOA	Service-Oriented Architecture	4, 8, 9, 19, 20, 33, 36, 39, 55, 56
SOAP	Simple Object Access Protocol	9, 45
SOME/IP	Scalable service-Oriented MiddlewarE over IP	34–36, 38, 39, 45
SRP	Stream Reservation Protocol	6, 60–63
TCP	Transmission Control Protocol	9, 34–36, 45, 46, 61–63
TDMA	Time Division Multiple Access	5, 32, 81
TSN	Time-Sensitive Networking	5–7, 11, 44, 45
TT	Time-Triggered	5
TTE	Time-Triggered Ethernet	5, 6, 11, 44
UDP	User Datagram Protocol	34–36, 45, 62
WS	Web Service	32, 45
WSDL	Web Service Description Language	9

Index

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 07. Juni 2018 Timo Häckel