APPSTACLE

(ITEA 3 – 15017)

open standard APplication Platform
for carS and TrAnsportation vehiCLEs

Deliverable: D 1.1

Specification of In-car Software Architecture for Car2X Applications

Work Package: 1

In-Vehicle Platform

Task: 1.1, 1.2

Cloud Platform Architecture Specification, Evaluation of Existing IoT- and Automotive Cloud Platforms

Document Type:	Deliverable	**Classification:**	Public
Document Version:	final	**Contract Start Date:**	01.01.2017
Document Preparation Date:	29.01.2018	**Duration:**	31.12.2019

ITEA 3
INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

EUREKA

History

Rev.	Content	Resp. Partner	Date
0.01	Initial creation of the document	Robert Hoettger	07.04.2017
1.0	First structure added	Marco Wagner	27.04.2017
1.01	SotA structure added	Marco Wagner	05.05.2017
1.02	Permission Chapter removed (will be taken care in WP2)	Sebastian Schildt	08.05.2017
1.03	Use Case Chapter added	Sebastian Schildt	04.07.2017
1.04	Added the first group of requirements for the Platform Study	Marco Wagner	17.07.2017
1.05	Added an additional use case	Philipp Tendyra	20.07.2017
1.06	Added the use cases proposed by SecurityMatters	Marco Wagner	20.07.2017
1.07	Added the use case proposed by Fraunhofer	Marco Wagner	21.07.2017
1.08	Removed the Communication Services Chapter	Marco Wagner	22.09.2017
2.0	Structured the document into two parts	Marco Wagner	22.09.2017
2.01	Added the input of Alexios Lekidis regarding in-vehicle connectivity	Marco Wagner	28.09.2017
2.02	Restructured IDS sections	David Schubert	05.10.2017
2.03	Updated references for the state-of-the-art for QoS	Robert Hoettger	05.10.2017
2.04	Added content on state-of-the-art for IDS	Alexios Ledikis	05.10.2017
2.05	Update on state-of-the-art for in-vehicle connectivity based on the input from Harald Weiler	Marco Wagner	19.10.2017

Rev.	Content	Resp. Partner	Date
3.0	Created the initial structure for the specification chapters	Marco Wagner	19.10.2017
3.01	Added some initial content for the APPSTACLE API specification	Marco Wagner	20.10.2017
3.02	Added content on the state-of-the-art regarding testing from Peter Kruse	Marco Wagner	24.10.2017
3.03	Added content on the state-of-the-art regarding App IDS	David Schubert	02.11.2017
3.04	Switched order of state-of-the-art chapters	Sebastian Schildt	02.11.2017
3.05	Added content on the state-of-the-art regarding Automotive APIs from M. Wagner and P. Adelt	Marco Wagner	02.11.2017
3.06	Added content on the state-of-the-art regarding Network-based IDS	Alexios Lekidis	14.11.2017
3.07	Finished QoS SotA and specification sections	Robert Hoettger	14.11.2017
3.08	Finished App IDS SotA	Hendrik Eikerling	16.11.2017
3.09	Removed Testing Chapters	Marco Wagner	21.11.2017
3.10	Removed all Chapters relevant to End-to-End Security	Marco Wagner	30.11.2017
3.11	Finished draft for Platform and App Runtime Spec	Marco Wagner	30.11.2017
3.12	Finished draft for OTA Spec	Peer Adelt	05.12.2017
3.13	First draft for ex-vehicle conectivity spec from Joost van Doorn	Marco Wagner	09.12.2017
3.14	Added Network IDS spec and additional ex-vehicle conectivity spec	Alexios Lekidis	10.01.2018
4.00	First complete draft ready	Marco Wagner	09.12.2017
4.01	Added Network IDS spec and additional ex-vehicle conectivity spec	Alexios Lekidis	10.01.2018
4.02	Added remarks by Pedro Cuadra	Marco Wagner	12.12.2017
4.03	Integrated the feedback by Rene Stallen	Marco Wagner	16.01.2018
4.04	Update SotA and Specification NetIDS	Alexios Lekidis	19.01.2018
4.05	Update SotA Ex-Vehicle Communication	Alexios Lekidis	24.01.2018
5.00	Final Version	Marco Wagner	29.01.2018

Contents

List of Figures

List of Tables

Summary

This deliverable is the first deliverable of the APPSTACLE Work Package 1 "In-Car Platform". It contains the results of two tasks: "State-of-the-Art analysis for In-Car Software with regard to Internet Connectivity" (T1.1) as well as "Specification of In-Car Software Architectures for Car2X Environments" (T1.2). Hence, this document is split into two parts. While part one summarizes the results of the state-of-the-art analysis, part two contains the specification of the different parts of the system. Figure 1 illustrates the relationship of Deliverable 1.1 with the other deliverables in APPSTACLE.

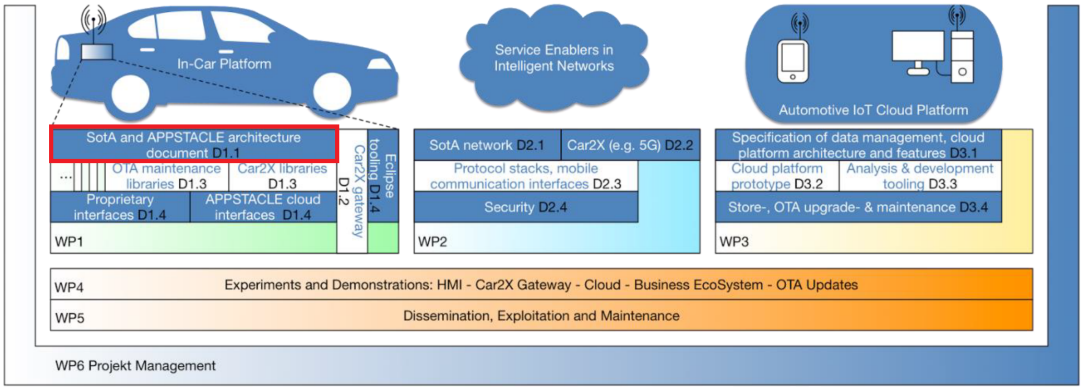

Figure 1.: APPSTACLE Work Packages Structure and Technical Deliverables.

Part one starts with a collection of use-cases brought in by the different APPSTACLE partners. These 22 different use-cases are used in two different ways: in the early stages of the APPSTACLE project, they acted as an exchange platform to visualize, discuss and align the different views, approaches and goals of the project partners. In the later course, they were used as a basis for deriving requirements or setting the focus of research in this broad technological field.

In the further course of part one, the state-of-the-art of the relevant fields of research and technology is presented. This includes a discussion of existing software platforms and app runtimes as well as automotive APIs and connectivity technologies (both in- and ex-vehicle connectivity). Additionally, the state-of-the-art for important parts of the APPSTACLE project are given namely Intrusion Detection Systems. QoS Monitoring and Over the Air updates. The second part of this deliverable specifies the in-vehicle platform to be developed in the course of the project in the form of a list of requirements. Regarding its structure, this part begins with some important definitions before iterating through the different building blocks of the platform. This includes namely the basic software platform and app runtime environment, the APPSTACLE API as well as the two planned intrusion detection systems (application-based IDS and network-based IDS). Furthermore, it defines the requirements of the ex-vehicle connectivity system, the QoS monitoring module as well as the over the air update functionality. The document is closed by a specification of the hardware for the in-vehicle platform.

Please note that this complemented by the APPSTACLE Security Architecture document which will describe the cross Work Package security issues.

1. Introduction

The purpose of this document is to describe the APPSTACLE in-vehicle platform. As such it also documents the work that has been done in Task 1.1 (State-of-the-Art Analysis for In-Car Software with regard to Internet Connectivity) and Task 1.2 (Specification of In-Car Software Architectures for Car2X Environments) of the APPSTACLE project.

The APPSTACLE in-vehicle platform is composed of both software and hardware, that is integrated into a vehicle. The goals of this platform are to (i) allow to install and operate Apps, (ii) access in-vehicle resources using the in-vehicle communication networks and (iii) exchange data with a cloud-based infrastructure. The fundamental system structure of the APPSTACLE in-vehicle platform has been jointly developed by all APPSTACLE partners during the initial architecture workshop. This structure, illustrated in Figure 1.1, comprises several software and conceptual elements that together realize the full APPSTACLE in-vehicle platform. These elements include security related blocks such as Authentication and Encryption[1], (Secure) Boot Loader[1], as well as an Application-level and Network-level Intrusion Detection System. Additionally, components to facilitate updates of both, the APPSTACLE in-vehicle platform itself as well supporting functionality for other electronic control units update procedures are integrated.

Furthermore, the in-vehicle platform has components that enable it to communicate both within the vehicle and with external devices. In order to allow the platform to host and execute Apps, an App Runtime block is integrated alongside with an APPSTACLE API block. The App Runtime sandboxes Apps and provides a standardized runtime environment, which abstracts the vehicle's resources as well as provides access to in- and ex-vehicle connectivity and APPSTACLE specific functionalities via the APPSTACLE API. Finally, the APPSTACLE in-vehicle platform and ecosystem will provide services such as Quality of Service Monitoring.

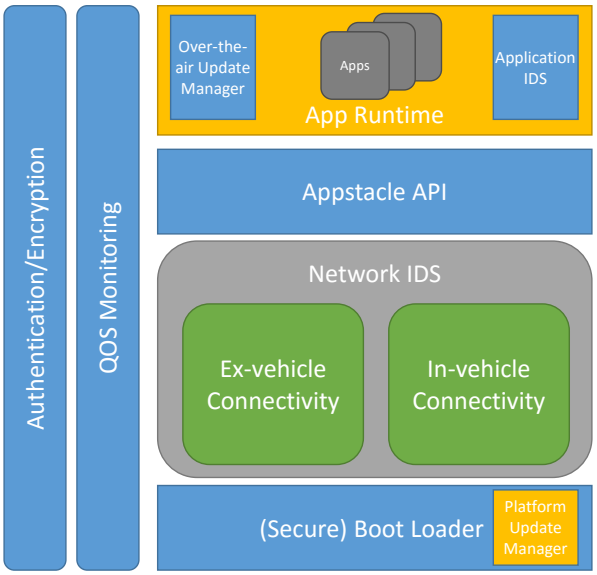

Figure 1.1.: The logical architecture of the APPSTACLE in-vehicle platform

[1]Will be described in a separate document called APPSTACLE Security Architecture

This document is divided into two parts: Part 1 presents the state-of-the-art with regard to the integral components and concepts of the APPSTACLE in-vehicle platform. Part 2 outlines the specification of the APPSTACLE in-vehicle platform. Within these two parts, the sub-chapters are largely aligned to the structure described here.

Part I.

State-of-the-Art

2. APPSTACLE Use Cases

This chapter gives a rough idea about the use cases envisioned for the overall APPSTACLE system. This is not meant as a detailed analysis, ready to be implemented (as the project will not implement all use cases, but rather some exemplary demos), but rather to guide the discussion of existing State of the Art solutions within the context of APPSTACLE and motivate the choices for the proposed APPSTACLE architecture.

2.1. Stakeholders

The envisioned APPSTACLE ecosystem consists of different stakeholders, with different requirements. These section lists the basic stakeholders considered.

Car Driver/Owner

This is the end-customer who eventually will have paid the cost of the system. He wants to select Apps and to run them on the in-vehicle platform. The vehicle owner also has a strong interest in controlling access to data from his vehicle.

Application Developer

The application developer develops Apps and solutions for the in-car platform and/or the cloud backend. They Apps may be offered to the end customer using an open platform (App Store), or can be part of a specific service functionality from the OEM, Suppliers, or third parties. Consequently the customer of an App might be the end-customer, or an OEM or service provider.

The App developer expects an open powerful development environment giving easy access to data the car and related cloud services. An App developer expects good integration with existing ecosystems.

Car Manufacturer / In-vehicle Platform Vendor

This persona developed or customized the in-vehicle platform and installed it into a car. This can either be an OEM offering the platform in his cars or an aftermarket product. It is expected that this persona also operates or manages operation of some additional infrastructure like an App store or a billing system. It is the goal of this stakeholder to generate revenue by operating and controlling the ecosystem.

Services Provider

Operates connected services that may include in-vehicle platform Apps or cloud-based Apps and offers them to the Car Driver / Owner. The Service Provider may be identical to the Car Manufacturer / In-vehicle Platform Vendor or just use the infrastructure provided by them.

2.2. User Story: Roadside Assistance

Idea

The technology developed in APPSTACLE helps a driver in the case his car breaks down. The basic idea is to have a Roadside Assistance app installed on the vehicle that allows a Roadside Assistance provider to diagnose and potentially repair the vehicle remotely.

Development phase

The App developer engineers a Roadside Assistance application most likely in cooperation with the Car Manufacturer / In-vehicle Platform Vendor and the Service Provider.

Setup phase

The App might already be installed by the OEM. Alternatively the Car driver / Owner downloads and installs a "Roadside Assistance" application to its in-vehicle platform using the corresponding app store. This may or may not be part of a contract between the Car Driver/Owner and a Service Provider (e.g. Roadside Assistance provider such as ADAC, RAC, AAA, or as part of an insurance policy).

Usage phase

The driver of a vehicle detects a problem with his car (e.g. indicated by warning lights) and pulls over or the car breaks down. The driver contacts his Roadside Assistance service provider (e.g. by a smartphone app or by phonecall, or through Headunit in the car) to request support. During this interaction the Service Provider is able to remote control the Roadside Assistance Apps installed in order to:

- Retrieve data from the car
- Execute diagnostic workflows and retrieve the results
- Control actuators within the vehicle

As a result, the Service Provider may be able to assist the driver by giving accurate information on the problem occurred, provide instructions for next steps (e.g. drive to the closest workshop, wait for help), solve the issue remotely (e.g. by re-setting fault memories) or dispatching a roadside assistance associate (e.g. by sending the right expert with the needed spare parts).

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API granting access the data and procedures needed
- APPSTACLE API able to be enriched by OEM specific extensions
- Roles and Rights management controlled by the Car driver /owner

2.3. User Story: Vehicle Tracking

Idea

The owner of a car or a third party need to track the position of a specific vehicle. Such scenario may occur for several reasons (e.g. for fleet management, stolen vehicle tracking, pay-per-drive insurance tariffs, car sharing, social networks: Let friends track your vehicle).

Development phase

The App developer implements a tracking application for the cloud. Additionally, an App in the in-vehicle platform needs to enable forwarding of time and position information.

Setup phase

The Car driver / Owner installs the App and allows the Service Provider to activate the forwarding of the relevant information to its cloud server.

Usage phase

The App forwards position information alongside with a timestamp periodically to the cloud server of the Service Provider. Within the cloud, this information is used for enhanced services.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API granting access to location
- Roles and Rights management controlled by the Car driver /owner to protect security and privacy of such sensitive information

2.4. User Story: Wrong Way Driver Warning

Idea

A vehicle takes part in a wrong way driver warning system in order to increase its own and other vehicle's safety. This is done by enabling numerous cars to forward position and direction data to a central server instance which matches them to a map and detects wrong way drivers. In this case, all vehicles that might be at risk are warned by the service.

Development phase

The App developer implements the wrong way driver warning service for the cloud. Additionally, an in-vehicle application is created, forwarding time, position and speed vector information.

Setup phase

The Car driver / Owner downloads and installs the specific app on his in-vehicle platform and allows access to position information. The app registers with the server instance.

Usage phase

The app forwards position and speed vector information alongside with a timestamp periodically to the cloud server of the Service Provider. Within the cloud, this information is used for map matching and risk analysis. In the case a wrong way driver is detected, the originator of the warning and the vehicles at risk are warned about the situation which is presented to the driver using the APPSTACLE HMI connector.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API granting access to location and speed vector

- Roles and Rights management controlled by the Car driver /owner to protect security and privacy of such sensitive information
- Interface to HMI (in-car or BYOD) to display warnings

2.5. User Story: Augment vehicle functionality

Idea

This user story describes a scenario where a vehicle will be enhanced by a specific functionality in order to adjust to special equipment. For example, adding a roof rack or a trailer to a vehicle might come with some software modifying brakebooster, suspension and ESP settings. For commercial vehicles such as semi-trailers, different trailer variants might want to extend and adapt functionalities of the tractor unit: When transporting livestock, the capability of having a live camera feed from the trailer might be added. When using a cooling trailer, temperature and state of refrigeration system can be communicated to the driver.

Development phase

The App developer implements some in-vehicle application that provides additional functionality or makes functionalities from added components available.

Setup phase

The Car driver / owner downloads and installs the specific app from an App Store on his in-vehicle platform.

Usage phase

The app is provided in the app ecosystem and the owner can download or use the functionality based on the business model (rent or buy). The SW will be downloaded and installed in the in-vehicle platform.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API granting access to data and functions from in-car ECUs and externally connected equipment

2.6. User Story: Data Collection Fleet Learning

Idea

A lot of data is available at a fleet of vehicles that can be used to enhance performance of autonomous driving systems, enhance understanding of component aging and enabling predictive maintenance algorithms. As a single car cannot provide all generated data, a backend needs to be able to orchestrate which individual cars deliver what information.

Development phase

The App developer implements a configurable data acquisition app for the car, that delivers data to the cloud according to the current connectivity situation and requirements from the backend. A backend system collecting the data is developed (enabling applications such as predictive maintenance or machine learning) to access the data.

Setup phase

Most likely such an application is pre-installed by an OEM. However, installing such an application from a third party is possible if the Car Driver/Owner installs it and grants the required access.

Usage phase

The app collects data and sends it to the cloud. It can be used to improve the cars software, keeping unplanned maintenance down, and improving future revisions of the same car. In the predictive maintenance use case the driver can be notified in case of expected problems.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API granting access to data from in-car ECUs
- Security and Privacy of the Data

2.7. User Story: IoT Data concentration

Idea

This user story describes a scenario where the technology developed in APPSTACLE could help to reduce bandwidth usage on the mobile internet connection. Many experts expect that the data that will be generated by future vehicles and is of interest for cloud-based services exceeds the bandwidth being available by far (or would cause significant costs respectively). The APPSTACLE in-vehicle platform could support in such cases by hosting domain specific apps that reduce the amount of data forwarded to the cloud (e.g. by preprocessing, detection of irrelevant data, data compression).

Development phase

The App developer engineers a specific data concentration application most likely in cooperation with the Service Provider.

Setup phase

The Car driver / owner downloads and installs a data concentration application for a specific domain / use case to its in-vehicle platform using the corresponding app store. This may or may not be part of a contract between the Car Driver/Owner and a Service Provider.

Usage phase

During the usage of the vehicle, the data concentration app constantly reads the relevant data using the in-vehicle connectivity of the APPSTACLE platform. This data is then concentrated in any of the forms explained earlier and the result of this procedure is forwarded to a connected cloud instance.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API including the data and procedures needed
- Roles and Rights management controlled by the Car driver /owner

2.8. User Story: Driver Seat Configuration

Idea

Cars used by several drivers can store the configuration of each driver. Car fleets (e.g. bus or truck companies) may use a cloud service to store the configuration for each driver, independent of the current car (truck, bus).

Development phase

The app developer implements the seat configuration by using the access to the driver seat ECU. He also implements the corresponding cloud service to exchange driver configuration data with the cloud.

Setup phase

The car driver / owner downloads and installs the specific app on his in-vehicle platform and allows access to the driver seat. The app registers with the server instance.

Usage phase

The car driver identifies himself by an NFC tag or fingerprint reader. Afterwards the app downloads the appropriate driver configuration from the cloud.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API granting access to driver seat and personal identification system

2.9. User Story: Parking Space Finder

Idea

The driver of a car needs to find a nearby parking space - as quick as possible, and as close as possible. Traffic or street surveillance systems may help to solve this frequent problem, thereby also minimizing the parking search traffic.

Development phase

The app developer implements the "finder app". The appropriate cloud service is probably provided by a second party which provides the "intelligent street" system data to find free parking spaces.

Setup phase

The car driver / owner downloads and installs the specific app on his in-vehicle platform and allows access to the HMI and navigation system. The app registers with the server instance.

Usage phase

The car driver enters a "find parking space" command to the app using the HMI. The finder app provides location information to the cloud instance and receives data of available parking spaces. The app will display this information on the HMI.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API providing access to car location (GPS data) and HMI

2.10. User Story: Improved Carpooling System

Idea

Current carpooling systems require medium term registration (at least one day ahead usually) and much administrative effort. Automated data service may minimize the time necessary to find a driver or passenger. As security risks are involved, it may be necessary to provide a secure authentication for both driver and passenger.

Development phase

The app developer implements the "Carpooling App". The carpooling provider implements the cloud service which finds out appropriate passenger/driver pairings.

Setup phase

The car driver / owner downloads and installs the specific app on his in-vehicle platform and allows access to the HMI. The app registers with the server instance. The driver proves his identity by password, fingerprint reader, or iris scan.

Usage phase

The car driver enters a "find passenger" command to the app using the HMI. The app provides location information to the cloud instance and receives data of possible passengers. The app will display this information on the HMI. This app may well be augmented by a chat service.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API providing access to car location (GPS) and HMI, as well as to the authentication system (fingerprint reader, iris scanner) of the car

2.11. User Story: Car Accident Registration by Video

Idea

"Dash Cam (Crash Cam)" videos may upload the most current seconds preceding an accident to the appropriate cloud service.

Development phase

The app developer implements the "Accident Registration App" as well as the cloud service receiving the video data.

Setup phase

The car driver / owner downloads and installs the specific app on his in-vehicle platform and allows access to the car's camera(s). The app registers with the server instance.

Usage phase

This app does not usually need a HMI, but runs in the background, invisible to the driver. However, ist may need configuration and test functions requiring access to the HMI.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API providing access to car camera

2.12. User Story: Car Theft Registration, Car Vandalism Registration

Idea

"Dash Cam" videos may upload the current seconds preceding and during a car theft to the cloud. Microphone and other sensor data may provide further valuable information to identify thieves and vandals.

Development phase

The app developer implements the "Theft Registration App" as well as the cloud service receiving the video and sensor data.

Setup phase

The car driver / owner downloads and installs the specific app on his in-vehicle platform and allows access to the car's camera(s), microphones, and shock/vibration sensors. The app registers with the server instance.

Usage phase

This app does not usually need a HMI, but runs in the background, invisible to the driver. However, it may need configuration and test functions requiring access to the HMI.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API providing access to car camera and microphone as well as shock/vibration sensors. Configuration and test may require access to the HMI.

2.13. User Story: Traffic Jam Warning and Traffic Jam Avoidance

Idea

Traffic surveillance systems provide information on current traffic jams and calculate routes to minimize the travel time by considering the current traffic situation.

Development phase

The app developer implements the "Traffic Jam Warning App". The traffic jam warning system relies on a global system and will be provided by a second party.

Setup phase

The car driver / owner downloads and installs the specific app on his in-vehicle platform and allows access to the car's GPS and navigation system. The app registers with the cloud service.

Usage phase

The app downloads current traffic information from the appropriate cloud service and displays warnings on the HMI if necessary. It may also connect to the navigation system to calculate alternative routes in order to avoid traffic jams.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API providing access to HMI and navigation system

2.14. User Story: Chat Service for Car Drivers

Idea

Providing a chat service with nearby car drivers, in order to get acquainted, share traffic information (e.g. traffic jams, construction sites, speed cameras), or admonish car drivers.

Development phase

The app developer implements the "Car Chat App". This must be able to spot nearby cars. A cloud service will be implemented for interconnection of car drivers.

Setup phase

The car driver / owner downloads and installs the specific app on his in-vehicle platform and allows access to the car's GPS system, HMI, microphone and speaker. The app registers with the cloud service.

Usage phase

The app provides information about nearby drivers who are prepared to communicate. A visual inquiry may be sent to the HMI of a nearby car, thus providing the chance to start communication.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API providing access to HMI, microphone, and speaker

2.15. User Story: Traffic Enforcement Camera Warning

Idea

Cloud service providing information on the position of "speed cameras" or "red light cameras" may increase traffic security by reminding car drivers to follow traffic rules.

Development phase

The app developer implements the "Speed Camera Warning App". An appropriate cloud service will be provided by second party.

Setup phase

The car driver / owner downloads and installs the specific app on his in-vehicle platform and allows access to the car's GPS system and HMI. The app registers with the cloud service.

Usage phase

The app uses the cloud service to get information about nearby "speed cameras" and the like. A warning message will be displayed on the HMI. A useful extension may provide a method to share informations about recently spotted cameras to the cloud service.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API providing access to HMI and speaker

2.16. User Story: Advertising Services for Drivers

Idea

Providing information to travelers about local shopping facilities, restaurants, public events, points of interest.

Development phase

The app developer implements the "Advertising Service App". An appropriate cloud service will be provided by second party.

Setup phase

The car driver / owner downloads and installs the specific app on his in-vehicle platform and allows access to the car's GPS system and HMI. The app registers with the cloud service.

Usage phase

The app uses the cloud service to get information about nearby advertising partners. The app may also filter this information according to the wishes of the driver.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API providing access to GPS, HMI and speaker

2.17. User Story: Social Media

Idea

Everybody wants to stay connected all the time. The idea is going for a wide spread of interpreting. The driver will be informed about Facebook friends on the same road or near by. Sharing beautiful

landscapes while driving with one click. Using all the possibilities of Twitter, Facebook, Blogs, Youtube (livestreaming) and every kind of social web. The core is to share your life with others and let others follow you on your way of life.

Development phase

The app developer implements the "Social Media App".

Setup phase

The car driver / owner downloads and installs the specific app on his in-vehicle platform and allows access to the car's GPS system, Cameras and HMI. The app registers with the cloud service.

Usage phase

The app uses the cloud service to get information about nearby social media partners or "friends". The app may also filter these information according to the wishes of the driver. Live stream of camera data will be very important.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API providing access to GPS, Camera, mic, HMI and speaker

2.18. User Story: Ambulance Assist

Idea

Normally, a path is made for the Ambulance when vehicle drivers hear the Ambulance's siren and move away to the next lane or so. This is easier when the path of the ambulance has fewer vehicles traveling in its direction or when the ambulance is not at a traffic signal. But if there is a traffic jam or there is a red signal at a busy junction, making way for the ambulance is difficult and time consuming. In order to avoid this, it would be better if the ambulance could communicate with the traffic signals well in advance so as to have a clear path. This user story describes the technical requirements for the APPSTACLE Platform to enable such a system.

Development phase

The app developer develops a cloud specific app that communicates with the cloud which in turn communicates with the traffic system or, alternately develops an in vehicle app that communicates directly with traffic signals in the vicinity.

Setup phase

This will be a pre-installed app. This could be a part of the contract between the platform provider (OEM/Supplier/3rd party), the organization(s) that provide the ambulance service and the Traffic authority of the region/state/country.

Usage phase

Once the app is switched on it communicates to the cloud the GPS information of the vehicle along with the source and destination locations. The cloud tracks the Ambulance and communicates with the next immediate 1-2 traffic signals so as to make a clear path for the ambulance. Additionally,

the cloud could also receive the traffic information in this path so as to communicate perhaps with traffic signals that are much ahead in the ambulance's path so that it could ensure a continuous free path for the ambulance.

Additional Use-case 1: The cloud also communicates with the vehicles in the region of the ambulance informing them to make way in advance.

Additional Use-case 2: The app also provides a means to communicate the patient's conditions to the hospital authorities so that the hospital has the necessary environment ready to treat the patient with no delays.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- The platform has necessary APIs, protocols to communicate with a system outside its ecosystem
- APPSTACLE API providing access to GPS etc.

2.19. User Story: System Surveillance and Maintenance

Idea

Diagnosis services (in form of either internal or external components) need to get access to sensor data, actuator status, application state or further parameters of diverse ECUs and components. New technologies allow collecting such properties remotely. The need of this may occur when the system has issues in performing its tasks. Diagnosis services could even have the possibility to perform sophisticated analysis of problems via collected information over time and live data. This information includes QoS relevant data as the correctness or timeliness of data and transmissions. The range of tracked information may be extended.

Development phase

In-vehicle systems run through this phase in development and implement monitoring interfaces for gathering information. An independent App for collecting and sending this information to the cloud is developed. At the Cloud, Apps for aggregated views of the information are developed.

Setup phase

The related monitoring App can be installed by an authorized user (may also be the OEM).

Usage phase

While in use the monitoring App collects and sends information to the cloud. The maintainer of the car reacts to performance issues. A view at the Cloud App with the collected monitoring data gives an outline on the circumstances and possible areas of conflict.

Technical requirements

- in-vehicle App-runtime
- Roles and Rights management
- Definition of interfaces for monitoring
- QoS mechanisms and QoS-monitoring

2.20. User Story: Pool car management

Idea

An organization has a pool of cars to be used by employees. These employees can make reservations for pool cars and then unlock and start the car via an app on his mobile phone.

Development phase

The App developer develops an app for the car, a mobile phone app for the user and a backend that allows those apps to work together. The vehicle app needs to be able to unlock the car and start the engine.

Setup phase

The vehicle app is installed on the vehicle and the car is added to a pool. The user installs the mobile phone app. The pool owner allows the user to use vehicles in the pool.

Usage phase

The user makes a reservation for a vehicle. He walks to the vehicle and requests the car to be unlocked. The authorization for the user is verified and the vehicle is unlocked.

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- APPSTACLE API allowing to unlock vehicle and start engine
- Roles and Rights management controlled by the Car driver/owner to protect security

2.21. User Story: In-vehicle behavior learning

Idea

In the complex architecture of the in-vehicle system car manufacturers may find it quite challenging to configure the communication between ECUs and calibrate each ECU individually in an optimal manner. Additionally, technicians as well as car manufacturers often require network analytics to perform in-vehicle diagnostics.

Development phase

A web interface application will be developed to provide a network / asset inventory map to the technician / car manufacturer, in order to provide real-time information about misconfigurations or potential threats to the vehicle.

Setup phase

A network monitoring device is connected on the gateway or the OBDII port of the vehicle and the car driver / owner can access the web application from any device smartphone or tablet device that is present inside the car.

Usage phase

The car driver identifies himself / herself in the web application and can observe in real-time the communication by ECUs in the form of an in-vehicle network map as well as the detailed messages that are sent. The web application notifies the user when it detects:

- Misconfigurations in the in-vehicle architecture
- Potential threats from abnormal behavior in the in-vehicle network

Additionally, the app can perform detailed data logging and reproduction to be transmitted to the car manufacturer for generating in-vehicle network performance reports or for deriving information about the state or misbehavior of the vehicle's ECUs (e.g. components requiring replacement, faults that need repairing).

Technical requirements

- Network monitoring device (IDS) connected to the vehicle gateway or diagnostic port

2.22. User Story: Secure Car2X data exchange

Idea

Data from the vehicle's physical environment are often received or transmitted from / to nearby stations (Car to Infrastructure) or vehicles (Car to Car) to enhance awareness about road conditions (e.g. traffic jams). As the exchanged data may contain sensitive information, security aspects should be considered to avoid that they are spoofed or malformed.

Development phase

A web interface application will be developed to provide access to the messages exchanged in the Car2X network as well as information about potential threats that will impose message leakage.

Setup phase

A network monitoring (IDS) device is connected on the gateway of a target vehicle and the car driver / owner can access the web application from any device smartphone or tablet device that is present inside the car, but also remotely.

Usage phase

The car driver identifies himself / herself in the app and can observe in real-time the communication between the target vehicle and the nearby stations or vehicles. A real-time network map can also be shown with the current connected devices in the wireless network. The app notifies the user when it detects:

- Addition or withdrawal of devices in the network
- Potential threats during C2X data exchange

Additionally, the app can perform detailed logging and reproduction of the secured data received by the target vehicle, derive only the required data for analysis and transmit them further to the car manufacturer or the nearest station for inferring information about road conditions.

Technical requirements

- Network monitoring device (IDS) connected remotely or directly to the vehicle gateway

2.23. User Story: Emergency Braking & Evading Assistance System (EBEAS)

Idea

The basic idea of the EBEAS app is to automate reactions in the case of detected obstacles in front of a vehicle. For this purpose EBEAS coordinates different in-car subsystems.

Development phase

The app developer engineers a EBEAS application most likely in cooperation with the Car Manufacturer / In-vehicle Platform Vendor, Service Provider, and suppliers that developed the participating automotive subsystems. As EBEAS encompasses highly safety-critical functionality, app developers have to be trusted, e.g. utilizing certification etc.

Setup phase

The app might already be installed by the OEM. Alternatively the car owner downloads and installs EBEAS from an app store.

Usage phase

The basic version of EBEAS has three functions. The radar sensors of the vehicle are used to detect obstacles in front of the vehicle. The EBEAS logic component decides whether to evade the obstacle or to brake and performs these actions using, e.g., the adaptive cruise controls functionality. Furthermore, the detected obstacle is reported to a service deployed in the automotive IoT cloud. This service informs other vehicles about the detected obstacle.
Extended versions of EBEAS could include:

- a pre-crash system is used to prevent serious damage from the driver and other passengers in cases in which an accident is not avoidable
- a car-to-car communication system used to negotiate on the reaction to a dangerous situation with other vehicles in range

Technical requirements

- Runtime environment for Apps on the in-vehicle platform
- Real-time scheduling
- In-vehicle connectivity and coordination of participating ECUs

3. State of the Art

3.1. Platforms and App Runtimes

3.1.1. Scope

This chapters lists, describes, examines and compares several available platform technologies that could be potentially used as a basis for the APPSTACLE in-vehicle platform. The idea is to investigate existing platforms to get an idea of the state-of-the-art and to compare these candidate platforms and select the most suitable one to be used as a basis for the APPSTACLE in-vehicle platform.

The modus operandi used was as follows: In a first step, user stories were collected from all APPSTACLE partners. This has been done to get an idea of the planned utilization of the APPSTACLE in-vehicle platform by the partners. The collected user stories are documented in Chapter 2 of this document. These user stories have on a next step been used to derive requirements regarding the in-vehicle platform. Overall, 57 different requirements have been defined and grouped into seven blocks in order enhance the readability and allow to set focus points during the study. A description of these requirements along with the full listing of the seven groups are documented in Section 3.1.1.

Candidates

In parallel with the derivation of the requirements, the APPSTACLE partners were asked to suggest potential candidates that should be analyzed during this study. This query lead to an extensive list of 30 candidates. This so called long list contains the following candidates:

- Android Auto
- Android Automotive
- Android Things
- Apertis
- ARM - mBed OS
- Automotive Grade Linux
- C3 IoT Platform
- CORBA
- DDS
- Eclipse Agail
- Eclipse ioFog
- Eclipse Kapua
- Eclipse Kura
- GE Predix platform
- Google Fuchsia
- GENIVI SW Architecture
- IBM Watson
- IoTivity
- Kaa
- Legato
- LineageOS / Android
- Macchina.io
- Contiki
- QNX
- Qt Over-The-Air Update
- SiteWhere
- Suse embedded

- ThingSpeak
- Thingworx
- Ubuntu Core

In order to be able to conduct a deep examination and apply all 57 requirements, a short list has been created that reduces to the number of candidates to eight potential platforms. This has been done by requesting votes from all APPSTACLE partners. The voting executed created the following short list of candidates:

- Legato
- QNX
- GENIVI SW Architecture
- Android Automotive
- Automotive Grade Linux
- Apertis
- Ubuntu Core
- SuSE embedded

During the course of the execution of the platform study, the GENIVI SW Architecture has been scratched from the list of candidates due to two major reasons: (I) due to the heavy workload of the participants of the study and (II) due to the fact that GENIVI does only offer a reference architecture and a demo image but not a fully managed, administrated and maintained distribution.

Requirements of the APPSTACLE in-vehicle platform

In order to analyze and compare available platforms the APPSTACLE project team defined a structured list of requirements. These requirements were derived from the future usage scenarios that are described in Chapter 2. The requirements were organized in several groups in order to enhance the readability of the study. In the remainder of this section these groups as well as the actual requirements are introduced and described.

The first group of this requirements list contains all items that are connected to the overall features of the platforms under examination. The first element of this group is the request for an update functionality of the overall platform in order to ensure to have the possibility to integrate new features, bug fixes or security patches to the platform while in use. This is accompanied by the request for some kind of platform diagnostics to be able to detect malfunctioning of the software. Furthermore, the platform to be chosen should be well documented in order to decrease the training period for new developers. The next three requirements ask for specific features that are beneficial in order to implement the user stories given in Chapter 2. This includes mechanisms for in-vehicle connectivity (to be able to gather data from the car), ex-vehicle connectivity (to be able to communicate with the outside world) and an HMI support (to allow those user stories that require direct customer interaction). Finally, the platform features group also asks for the minimal system requirements of each candidate and the in-built security features. Table 3.1 summarizes the requirements building the platform features group.

The second group of requirements contains items that directly correspond to the runtime behavior of the candidate platforms. This includes the need for a stable platform as well as the booting times. For the latter one, the reference board chosen is the Raspberry Pi 3 as most candidates are deployable on this platform. Another requirement to be examined is the responsiveness of the platform when executing as well as the openness of the candidates for adaptations or changes. Table 3.2 summarizes

Group 1: Platform features	
R1.1 Platform Updateability	
Is there an update functionality available? Is it secured (authenticity and integrity)?	e.g. OS updater
R1.2 Platform Diagnostics	
Is there any kind of diagnostics tooling available? Are interfaces available to collect system properties? Is there a secure logging or secure audit tool available?	e.g. error reporting, detection of manipulation
R1.3 Platform Documentation	
Is there adequate documentation available	e.g. platform overview, quick start guides
R1.4 In-vehicle Connectivity included	
Are there any modules available that allow to connect to typical vehicular networks? What level of abstraction is available? Are security features available?	e.g. CAN; raw sockets / drivers only, complete network stack; support for AUTOSAR Secure Onboard Comm.
R1.5 Ex-vehicle Connectivity included	
Are there any modules available that allow to connect to to typical car-2-cloud networking technologies? What level of abstraction is available? How is the connection secured?	e.g. LTE; raw sockets / drivers only, complete network stack; TLS
R1.6 HMI support available	
Is there any kind of HMI support available?	e.g. X Window System
R1.7 Minimal System Requirements	
What are the minimal system requirements to deploy the platform?	• Min. RAM needed • Min. Storage needed • Architecture flexibility (x86, ARM, ...) • Min. Computing Power needed
R1.8 Security features	
Is the a secure store for crypto keys? Is there support for crypto operations or security hardware? Is there support for secure boot?	

Table 3.1.: Platform Feature Requirements

the four requirements building the platform runtime group.

The third group of requirements examines the application runtime environments of the candidates. This includes the request for permission transparency (is the user able to see which permissions have

Group 2: Platform runtime	
R2.1 Stability of the Platform	
Did the platform run stable during our tests?	e.g. any blue screens
R2.2 Booting Times	
How long does it take to boot the platform on a Raspberry Pi 3 or comparable?	[s]
R2.3 Platform Responsiveness	
Are there any noticeable delays when actuating on the platform?	
R2.4 Adaptability	
Are system / OS services adaptable?	

Table 3.2.: Platform Runtime Requirements

been granted) and the level on which such authorization takes place (e.g. whether it is possible to grant permissions on an app level). Additionally, the isolation of applications within the runtime is examined as well as features that allow to manage resources or the installation of apps. Other requirements within this group focus on error handling and reporting functionality, as well as abstract access to vehicular data and functions or the features of the platform itself. Moreover, the support of a human machine interface and for multi-user concepts is also examined in this group. Table 3.3 lists the ten requirements of the app runtime group.

The fourth group of requirements tackles issues important to the developers of applications running on the platform. This includes the extent (e.g. including an IDE), the accessibility (e.g. costs for usage) and the support of different programming languages. Additionally, testing issues are examined by inspecting the availability of both testing hardware and virtual testing environments. Furthermore, the documentation given to potential developers as well as the existence of coding guidelines is tested. Other issues examined are the speed a new app can be developed as well as the support given by the infrastructure and the support of know-how protection. Table 3.4 summarizes the requirements of the app development and SDK group.

The fifth group of requirements contains all relevant issues regarding an app store connected to the platform candidate. This includes the actual availability of such a store as well as its accessibility. Furthermore, technical features are examined such as the existence of a management interface, the ability to create device-based profiles or a billing system. Additionally, the app authorization management is analyzed. Table 3.5 summarizes the requirements of the app store group.

Group 3: App Runtime
R3.1 Permission Transparency
Is it possible for the owner or driver to detect which permissions an App has?
R3.2 Permission Management per App
Is it possible to define the permissions on an App level?
R3.3 Isolation
Is there an isolation mechanism to protect the system and other Apps from harmful / misbehaving Apps? What access control and sandboxing features are available?
R3.4 Resource Management
Does the platform offer mechanisms to manage its resources (e.g. free memory, available runtime) and start or stop Apps? Is it possible to define resource limits for Apps?
R3.5 Installation Management
Does the platform offer a system to install / uninstall Apps? How are Apps authenticated? Who decides on the App installation?
R3.6 Error handling and reporting
Is there any kind of diagnostics tooling available?
R3.7 Abstract access to vehicle data and functions
Is there an API available that allows access to vehicular data or functions (independent of car model etc.)? How are the access rights managed?
R3.8 Abstract access to platform features
Is there an API available that allows access to platform features (e.g. memory)? How are the access rights managed?
R3.9 GUI support available
Is there a GUI framework available for the Apps?
R3.10 Multi-user concept
Is there multi-user support? How are the users identified?

Table 3.3.: App Runtime Requirements

The sixth group of requirements analysis the platform candidates regarding license issues. This includes issues such as the strength of the copyleft rules. Furthermore, the licenses used by the candidates are examined regarding the approval by the Open Source Initiative (OSI) or its compatibility to other licenses. Additionally, the obligations for using and contributing respectively restrictions for commercial use or licensing costs are analyzed. Tabel 3.6 summarizes this group.

Group 4: App Development and SDK
R4.1 App SDK available
Is there a SW development kit available? What is included (e.g. with / without an IDE)?
R4.2 App SDK accessibility
Is the SDK available for free? Is it OSS?
R4.3 SDK language support
Which programming languages are supported?
R4.4 Testing hardware accessibility
Which HW test environments are available (e.g. Raspberry Images)? Are they accessible and affordable?
R4.5 Virtual test environment availability
Is there a virtual test environment available and accessible (e.g. VM Image)?
R4.6 SDK and App development documentation availability
Is the SDK and the App development procedure well documented?
R4.7 Platform Architecture and Guidelines
Does the architecture affect the design/implementation of applications? How? Does the architecture consider performance, security and safety capabilities? Does the architecture include guidelines and standards?
R4.8 Customizable SDK
Can the SDK be extended? What kind of interfaces are provided by the SDK? Is the SDK fully open source? Is the SDK full featured? Does it have all that the actual development/testing needs? Is the SDK flexible enough to comply with goals in APPSTACLE and does it include usual development features like debugging?
R4.9 New App deployment speed
How fast is it to deploy a new App to the platform for testing? Is support required for deploying apps to the platform? Is it intuitive? What's the fastest time I can get results from a test? What kind of skills are needed for the development? Are those covered in APPSTACLE?
R4.10 Infrastructure Support
Is specific infrastructure required to support the development for this platform? Can there be infrastructure within APPSTACLE to speed-up development or testing?
R4.11 Support for Know-How protection
Is there support for protecting intellectual property in Apps?

Table 3.4.: App Development and SDK Requirements

The seventh and last group of requirements brings together issues regarding the developer community behind each candidate. This includes requirements regarding the size, liveliness or responsiveness

Group 5: App Store
R5.1 App Store available
Is there an App Store framework available?
R5.2 App Store accessibility
Is the App Store available for free? Is it OSS?
R5.3 App Store management interface availability
Is there functionality available to manage the App Store (e.g. to select which Apps are presented)?
R5.4 Profile availability
Is it possible to create profiles for different devices in order to only present Apps fitting to the device connected?
R5.5 Billing system availability
Is there a billing system available?
R5.6 App authorization
Does the App Store support an authorization scheme for Apps to define which are legitimate?

Table 3.5.: App Store Requirements

of the community, as well as management issues and the number and strength of the members behind the candidate. Furthermore, key figures such lines of code, the existence of a project roadmap, the re-usability of the code as well as the state of the project. Lastly, related projects are examined to determine the relevance of the candidate. Table 3.7 summarizes the requirements of this group.

Voting scale and weighting factors

In order to execute an objective examination, a voting scale as well as weighting factors have been defined. The voting scale reaches from -2, which equals to "strongly contradicts the requirements" or "highly negative behavior" respectively, to +2, which equals to "strongly fulfills the requirement" or "highly positive behavior" respectively. Please note that the actual meaning is defined separately for each requirement. This information is included in the tables 3.1 - 3.7. Table 3.8 summarizes the voting scale used to analyze the candidates. Additionally, in order to be able to underline the importance of some requirements compared to others, weighting factors have been defined. These reach from 0, which equals to "not important at all", to 3, which underlines that it is extremely important. While Table 3.9 summarizes these weighting factors, the determination of the values are documented in the tables 3.1 - 3.7.

3.1.2. Overview

Automotive Grade Linux

Automotive Grade Linux is a Linux Foundation workgroup that is dedicated to developing open source software solutions for the Automotive Industry. AGL started off as an IVI system, but is now also

Group 6: Licensing
R6.1 Copyleft rules
Is it strong / weak / non copyleft?
R6.2 License approval
Is the license used approved by the OSI?
R6.3 License interoperability
To which licenses is the license used compatible?
R6.4 Usage obligations
What are the obligations that have to be fulfilled when using or further developing the software?
R6.5 Contribution obligations
Is there a Committer Agreement that one has to sign? What are the obligations that come with that?
R6.6 Restrictions for commercial use
Are there any restrictions regarding commercial use of the software?
R6.7 License costs
Are there license costs that have to be paid when using the software commercially?

Table 3.6.: Licensing Requirements

being developed to support telematics systems. AGL has partners from the OEM, Suppliers the semiconductors industry and other automotive software providers.

The goals of AGL are to provide [9]:

1. An automotive specific Linux operating system with a broad developer community comprising individuals, academic institutions and private organizations.
2. A collaborative, transparent and open environment to all the tiers that are involved in the automotive ecosystem.
3. An embedded Linux distribution that enables rapid prototyping for developers new to Linux but with a background of open source.
4. A collective voice for working with other open source projects and developing open source solutions.

AGL Software Architecture AGL has a well-defined layered architecture as shown in Figure 3.1. The App/HMI layer consists the pre-installed app together with its business logic and the HMI.

1. The Application Framework layer provides APIs for managing the application lifecycle on the AGL system.
2. The Services Layer contains the user space services that all applications can access. These services provide either an inter-process communication type interface or a subroutine/function type interface.
3. The Operating System layer provides the Linux kernel, the necessary drivers and additional utilities.

Group 7: Developer Community
R7.1 Community size
How big is the community contributing to and using the software?
R7.2 Community liveliness
How lively is this community? (age of the last commit, release frequency, forums...)
R7.3 Community responsiveness
How good is the support of the community when facing problems?
R7.4 Community management
How is the community organized? Is it organized at all?
R7.5 Community members
What kind of developers are contributing to this project? (hobbyists / maker only, industrial members, ...)
R7.6 Project lines of code
How many lines of code are part of this project?
R7.7 Project Road-Map
Is there any? Is it clear? Are APPSTACLE goals applicable / compliant to the platform expected developments?
R7.8 Code re-usability
How much of the code will potentially be used in APPSTACLE? How much is missing and how much must be adapted?
R7.9 State of the Project
What's the state of the project? Is it already finished? Just starting? Stable? Does the project have a QA plan? Does the project have a good testing framework and CI infrastructure?
R7.10 Relevant related projects
Are other automotive or relevant projects also related/linked/co-developed with the given platform?

Table 3.7.: Developer Community

Legato

The Legato project is an initiative started by Sierra Wireless to provide an open, secure and easy to use application framework to the Internet of Things. The goal is to provide an open source, Linux based embedded platform to simplify the IoT application development.

Legato Software architecture The Legato platform is an open source embedded platform built on Linux. As shown in Figure 3.2, it includes an Application framework built on a fully tested, long term support version of a Linux kernel and tightly integrated to eclipse based development environment.

- Application framework comes with a C-based runtime library to maximize processing power and

Value	Meaning	
-2	Strongly contradicts the requirement	Highly negative behavior
-1	Contradicts requirement	Negative behavior
0	Doesn't fulfill the requirement	Neither positive nor negative behavior
1	Fulfills the requirement	Positive behavior
2	Strongly fulfills the requirement	Highly positive behavior

Table 3.8.: The voting scale used to benchmark the candidates

Value	Meaning
0	not important at all
1	"default" importance
2	quite important
3	very, very important

Table 3.9.: The scaling factors of the requirements

multi-language support thus enabling application written using different languages to run on Legato.

- The Linux distribution uses free open source packages validated by the Linux Foundation's Yocto Project and a long term version of the Linux kernel.
- The development environment is an Eclipse IDE with feature rich tools based on Sierra Wireless' developer studio.

Apertis

Apertis is a FOSS platform for infotainment for automotive vehicles. It was started by Robert Bosch GmbH and is now made an open source platform maintained and developed by Collabora. It is derived from Ubuntu/Debian distributions and geared towards ARM and Intel x86 architectures. Apertis comes with a range of built-in features that can be used and/or extended upon to build custom applications. It also provides an app store for the custom applications [30].
However, it is important to note that Apertis is not tailored for mission critical operations such as engine control, etc., but is centered around an extendable infotainment system.
"The goal of Apertis is to maximize the sharing across products, with the aim to improve time to market and reduce the efforts on long-term maintenance, in particular to enable quick and consistent response times for security issues in internet-enabled products." [39]

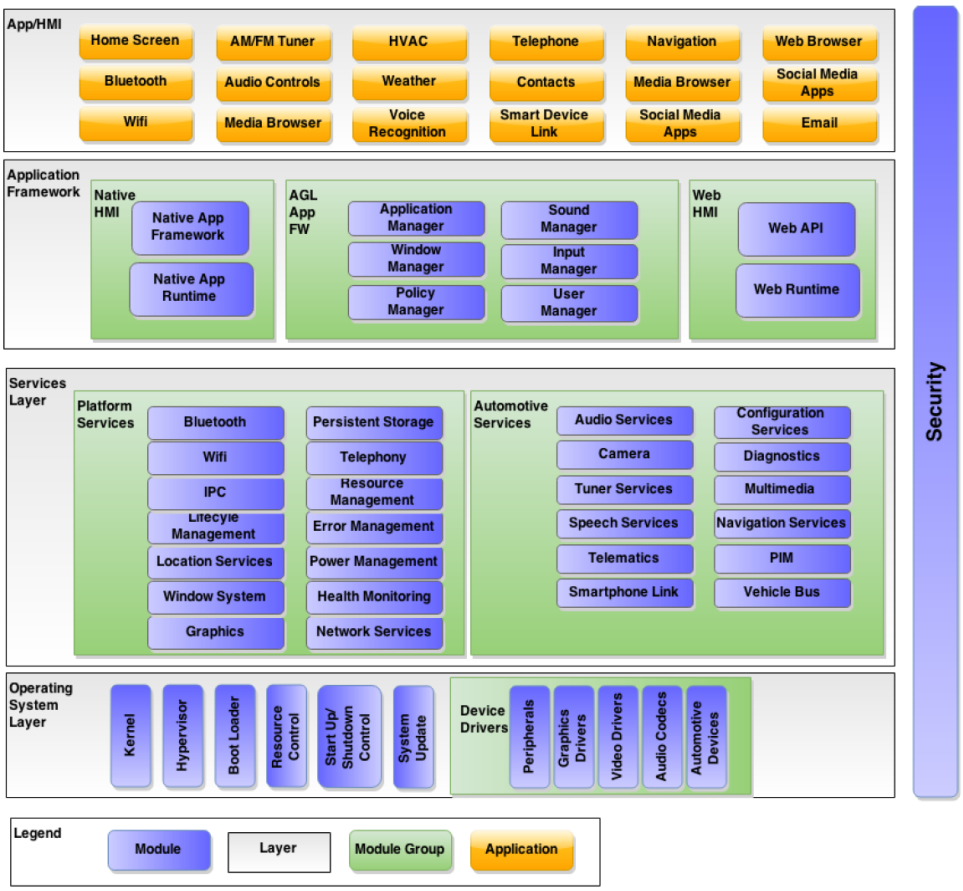

Figure 3.1.: AGL Software Architecture [9]

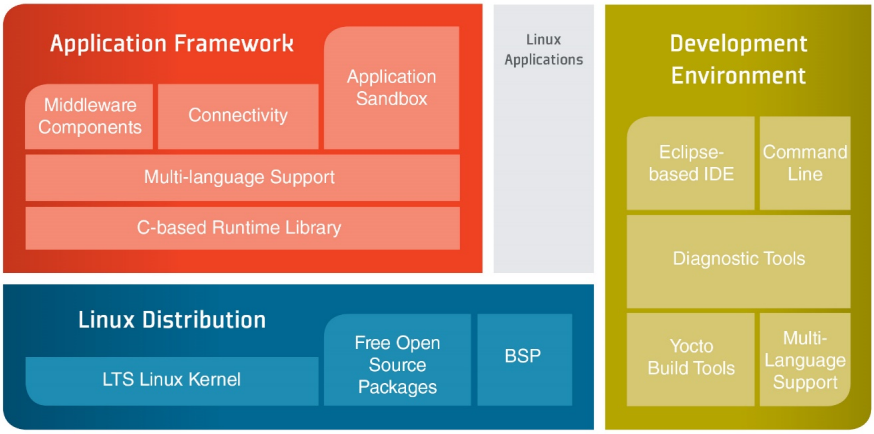

Figure 3.2.: Legato Software Architecture [135]

Ubuntu Core

Ubuntu Core is a light weight, minimalistic version of the Ubuntu OS. It is designed for the secure deployment of IoT applications on embedded devices, cloud, autonomous machines and other internet connected devices [160].

The Ubuntu Core offers the following important features [160]:

1. Secure, reliable and faster upgrades.
2. These upgrades are atomic and can be rolled back. The OS is also considered as an application and can thus be rolled back.
3. Provides a app stores that can be customized.
4. Provides signature based authentication for greater user end security.
5. Clear distinction between OS and the applications.

Ubuntu Core architecture

The Ubuntu core architecture is split into distinct layers. Every module in the diagram including the OS are applications called snaps. Each snap is a contained application. The security of the platform is ensured by the fact that the OS snap is provided by Canonical and the Kernel snap and gadget snap (containing all the necessary BSPs) are provided by the device manufacturer. As a developer, one needs to be concerned with the application security. Snappy core provides tools for this as well. The architecture is designed in such a way that the any security issues pertaining to a particular snap is curtailed only to that snap and the remaining snaps can function independently without any hindrance.

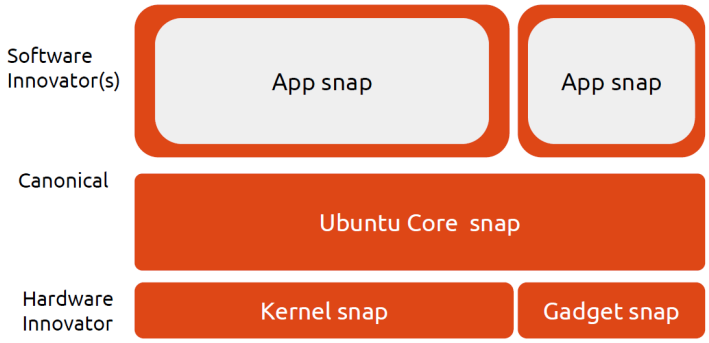

Figure 3.3.: Ubuntu Core Software Architecture [160]

SUSE Embedded

SUSE Embedded Linux version from SUSE that requires a paid subscription. SUSE Embedded works as a "Just Enough Operating System" and is then tailored to meet the needs of individual customers (e.g., there is no single release or a specific point to evaluate SUSE Embedded).

SUSE Embedded is based on the SUSE Enterprise Linux desktop version of the SUSE Enterprise Linux Server these of which are based on the openSUSE and numerous other products. Therefore, the study considers all SUSE platforms to get a complete picture of all the possibilities of SUSE Embedded.

Android Automotive

Android is an open source mobile operating system based on the Linux kernel developed by Google. Android gives us the freedom to implement our own device specifications and drivers by means of Hardware Abstraction Layers (HAL). These HALs provide a standard method for creating software hooks between the Android platform and the hardware [53]. Android Automotive is a subset of the above mentioned HALs. Currently, this subset provides a set of vehicle HALs and properties which can be defined and implemented by the OEMs to render the Android platform into an automotive IVI system. Although the platform supports no in-vehicle protocols such as CAN, LIN, etc., the Android platform claims that the HALs provides a consistent interface to the Android framework regardless

of physical transport layer [52]. At the moment OEMs such as Honda and Hyundai have IVI systems based on Android but on older versions. However, more organizations are partnering with Android to develop IVI systems based on Android Automotive standard [145].

QNX

The QNX CAR platform for infotainment is a set of pre-integrated technologies from QNX Software Systems and its partners. Currently more than 40 OEMs use QNX on about 50 million cars [82]. The usage of the QNX requires separate and paid subscriptions for the IVI platform and the SDK. QNX boasts of a complete system – the QNX Neutrino Operating System, in-vehicle telematics, HMI with a support for a variety of UI frameworks, multi-media engine, mobile connectivity framework, an app store, eclipse based SDK, Secure OTA platform, certified security modules, etc., all of which are claimed to be customizable as per the OEMs' needs. The QNX platform can also be deployed on a number of different hardware architectures [121], [122].

3.1.3. Discussion

This chapter discusses the seven candidates in detail. This is done by outlining and justifying the results of the analysis regarding the requirements introduced in section 3.1.1.

Group 1: Platform Features

Requirement	W	AGL	Apertis	Ubuntu Core	SuSE embedded	Legato	QNX	Android Automotive	Comment
R1.1 Platform Updateability	1	1	1	2	2	1	-1	2	• -2 - Technically impossible to support it • -1 - No implemented means to support it or 3rd party license (Ex: QNX) • 0 - No support, but can be implemented • 1 - OTA mechanism already supported • 2 - Secured OTA mechanism already supported
R1.2 Platform Diagnostics	1	2	2	1*	1*	2	1	2	• -2 - Technically impossible to support diagnosis capabilities • -1 - No means to support diagnosis capabilities • 0 - No diagnosis means are supported • 1 - Partial software stack diagnosis capabilities. Some layers have a Built-in API for diagnosis (e.g. Logging or tracing). An issue can't be fully tracked through the entire software stack. • 2 - Full software stack diagnosis capabilities. Each abstraction layer has a Built-in API for diagnosis (e.g. Logging or tracing). An issue can be tracked through the entire software stack.
R1.3 Platform Documentation	2 (Important for the start)	1	1	1	0	2	2	0	• -2 - No documentation available and no means to get information • -1 - No documentation but support is provided • 0 - Poor, misleading or unclear documentation • 1 - Documentation available • 2 - Clearly organized and structured documentation
R1.4 In-vehicle Connectivity included	3 (Highly Important for the Start)	2	0	1	1	1	2	0	• -2 - Impossible to support any protocol • -1 - No implemented means to support any protocol • 0 - No protocol support • 1 - Supports (Socket)CAN at least • 2 - Supports multiple protocols
R1.5 Ex-vehicle Connectivity included	1	1	1	2	1	1	1	1	• -2 - Doesn't provide means to support more than one of the points, see below. • -1 - Doesn't provide means to support one of the points, see below. • 0 - Provides none of the points, see below • 1 - Provides one or two points, see below. • 2 - Provides all three points, see below. • Points are – Cellular connectivity – WiFi, WiFi-APi or Bluetooth – APIs/Protocol support for cloud connectivity
R1.6 HMI support available	1	2	2	0	1	0	2	2	• -2 - Impossible to support desktop environment • -1 - No implemented means to support desktop environment • 0 - Nothing but command line but could be supported • 1 - At least desktop environment • 2 - HomeScreen like env or means to create home screen like environment
R1.7 Minimal System Requirements	0	1	1	1	0	2	1	1	Best score for the least system's requirements
R1.8 Security Features	2 (Important for the start)	-1*	-1	1*	2	1	-2	1*	Security features were graded inside the interval [SL,SH], where SL denotes lowest security features and SH highest security features between the platforms studied. Reference platform for SL was QNX and for SH was SuSe embedded. • -2 - Approximately equal to SL • -1 - Closer to SL • 0 - In the middle of SL and SH • 1 - Closer to SH • 2 - Approximately equal to SH

Table 3.10.: Results for the Platform Features group (* see note)

Notes

- Why a requirements doesn't apply for a platform
- There is missing information about the requirement for a platform
- The requirement analysis was splitted and multiple scores were obtained. How the final score was calculated?
- **R1.2 - Ubuntu Core:**No specific or central entity to track end-user application logs
- **R1.2 - SuSE embedded:**No specific or central entity to track end-user application logs
- **R1.8 - AGL:**security considerations are in place but there is no implementation for most of them and for the ones in place are performance demanding.
- **R1.8 - Ubuntu Core:** security is outstanding but seems that it's performance demanding and it's not optimized for Automotive usage, whereas for other platforms AGL, Apertis, QNX and Android Automotive it is.
- **R1.8:Android Automotive**: Implements a lot of security measures, but the mechanisms are considered less advanced than SuSE embedded

Justification for the scoring

- **R1.1 - AGL:** Score is 1 because we do have OTA using OSTree, but the secure part of it is still a work in progress.
- **R1.1 - Apertis:** Does support OTA update using BTRFS. But every update happens as an entire file-system backup -> For either Platform or an app. Autonomous roll-back possible for Platform but app, it's users choice if he wants to roll-back.
- **R1.1 - Ubuntu Core:**support OTA update. Autonomous roll back possible. Additionally, update secured by the use of a controlled app-store.
- **R1.1 - SuSE Embedded:** Update based on RPM packages. Security ensured by means of package signing, repository meta-data signing and repository identification.
- **R1.1 - Legato:** (Can be between 1 and 2) Update-ability ensured. Autonomous roll-back possible. hooks available for digitally signing the packages. To use this feature, we must build your own signing/encryption tool and replace it with the default one.
- **R1.1 - QNX:** Uses a third party - Red Bend's OTA update tooling. Secure update possible.
- **R1.1 - Android Automotive:** OTA update possible. Secure update possible. No information on roll-back possibility.
- **R1.3 - AGL:** (Between 1-2) Has vast documentation on the Application framework, AGL image and SDK building for different HW platforms, SDK servers, CAN signalling, installation management, security, has examples for binding references. However it lacks comprehensive documentation on the binding library (this is very important considering that AGL doesn't have a built-in library to access platform features), SDK integration with IDEs, testing and debugging. Additionally, there is no single repository for the documentation that covers everything completely -> Split into parts at multiple locations.
- **R1.3 - Apertis:** Has a vast requirement specification document, but no documentation that details the extent to which these have been implemented. Has a good developer (application developer) documentation -> Has coding guidelines, documentation on memory management, process handling, Apertis API definitions (but no documentation on usage). But some of the documentation is old and is sometimes plagued with broken links.
- **R1.3 - Ubuntu Core:** Has documentation on building custom images, building simple snaps, lists the various interfaces (but no documentation on the usage), stores and its usage. Documentation split as snapcraft (application packaging tool), ubuntu core. Documentation also available on help forum too.
- **R1.3 - SuSE Embedded:** (Between 0-1) Tons of documentation available, but very little specific to SuSE Embedded.
- **R1.3 - Legato:** Legato has extensive documentation on the platform, runtime library (together with definitions of the APIs and its usages), building the platforms, usage of the IDE, developing apps. The IDE has additional documentation on testing and debugging the application on the

Hardware as well.

- **R1.3 - Android Automotive:** (Between 0-1) Tons of documentation available on Android, but very little specific to Automotive.
- **R1.3 - QNX:** (Between 1 and 2). Has extensive documentation but hard to find what we need as the documentation is split into multiple sources.
- **R1.4 - AGL:** Supports CAN and LIN. Microchip a partner of AGL is currently developing Linux drivers for MOST (but licensed as GPLv2).
- **R1.4 - Apertis, Core, SuSE:** Do not have any support for In-Vehicle connectivity.
- **R1.4 - Legato:** (Between 0-1). Possible to integrate SockectCAN. CAN drivers for mangOH board available.
- **R1.4 - QNX:** CAN,MOST, Ethernet / IP and implementation of AUTOSAR architecture (without MCAL drivers)
- **R1.4 - Android Automotive:** (Between 0-1) Provides Hardware Abstraction Layers to implement the necessary network interfaces. Provides CAN support (through vehicle HAL library)
- **R1.5 - AGL:** Supports WiFi, Bluetooth, oFono for telephony based on Bluetooth and has a network manager. Possible to support LTE but not tested. AGL provides no Cloud infrastructure -> therefore 1
- **R1.5 - Apertis:** Connman provides cellular and WiFi connectivity. Bluetooth based on BlueZ. No documentation on Cloud Infrastructure. -> therefore 1
- **R1.5 - Ubuntu Core:** Has modem-manager and network-manager that provide 2G, 3G, 4G, WiFi, WWAN and Ethernet connectivity, Bluetooth based on BlueZ. Provides cloud infrastructure.
- **R1.5 - SuSE Embedded:** Has complete network stack but no documentation on Cloud infrastructure -> therefore 1
- **R1.5 - Legato:** (Between 1-2): Supports 2G, 3G, 4G, WiFi, WiFi access point, telephony services and cloud infrastructure but no Bluetooth and also since a single HW for WiFi is used, both WiFi client service and WiFi access point service cannot be used at the same time. Also supports AirVantage
- **R1.5 - QNX:** Has complete IP stack, IEEE 802.11a/ b/ g/ p, Bluetooth, USB
- **R1.5 - Android Automotive:** IEEE 802.11, Bluetooth / Bluetooth LE, Cellular radio, USB
- **R1.6:AGL, Apertis, QNX, Android Automotive** have a Home-screen based environment. **SuSE** has a desktop based environment and **Ubuntu Core** and **Legato** supports neither a home-screen/desktop based environment
- **R1.7:** The scores decided as the system with least requirement gets 2 and highest requirements gets 0. AGL and Apertis work on minnow board (1.33GHz, 2GB, 1GB), Snappy core is between 1-2 (900MHz, 1GB, 256MB), SuSE and QNX (1.5GHz, 2.2GB, 512MB), Legato (550MHz, 256MB, 128MB), No info on Android Automotive
- **R1.8**: **AGL**: Recommendations for secure boot, Certificate and Key Management but no actual implementation
- **R1.8**: **Apertis**: No secure boot, SSL, Netfilter firewall mechanism
- **R1.8:Ubuntu Core**: UEFI Secure Boot, SHA (256-512) / DES / MD5 crypt, Netfilter firewall mechanism
- **R1.8:SuSe embedded**: UEFI Secure Boot, SCAP, TLS, SHA (256-512) / DES / MD5 crypt, Netfilter firewall mechanism (Firewall configuration with custom ruledefinition and logging with VNC /web browser - protected by firewall)
- **R1.8:Legato**: Secure boot only for Sierra Wireless platforms (e.g. AR / WP Series), cryptographic pseudo-random number generator (CPRNG), TLS, SSL
- **R1.8:QNX**: Elliptic Curve and Public key Cryptography (Certicom's Asset Management System), SSL
- **R1.8:Android Automotive**: Secure boot (hash table), Watchdog against DoSattacks, AES - RSA - DSA and SHA cryptography, AFWall+ (Android Firewall)

Group 2: Platform Runtime

Table 3.11.: Results for the Platform Runtime group (* see note)

Requirement	W	AGL	Apertis	Ubuntu Core	SuSE embedded	Legato	QNX	Android Automotive	Comment
R2.1 Stability of the Platform	1	1*	1	1	1	1	1	1	• -2 - Doesn't even boot • -1 - OS Boots but the interface isn't responsive • 0 - OS Boots and interface is responsive • 1 - OS Boots and demo applications can run • 2 - OS Boots and demo applications run without hanging
R2.2 Booting Times	0	1*	0*	-1	-2	1	1	1*	• -2 >120s • -1 - 60s to 120s • 0 - 30s to 60s • 1 - 15s to 30s • 2 - <15s
R2.3 Platform Responsiveness	1	1*	1	2	0	1	1	1	• -2 - Doesn't even boot • -1 - OS Boots but the interface isn't responsive • 0 - Responsive, but huge delay is experienced • 1 - Responsive, but some delay is experienced • 2 - Fully responsive and no delays is experienced
R2.4 Adaptability	0	2	2	1	2	1	2*	2	• -2 - Technically impossible to add adaptability • -1 - No implemented means to add adaptability • 0 - No support for adaptablity to the user • 1 - Possible to add interfaces to the User • 2 - Already available • e.g. Setting Screen resoultion, WiFi settings, Bluetooth pairing

Notes

- **R2.1 - AGL:**There's a known issue about AGL not booting in the first trial. After first trial it boots normally.
- **R2.1 - QNX:** Sample applications were tested on QNX 6.5/6.6 and QNX 7.0. All demo applications are low-level (C/C++ with pthreads) but still can be compiled and executed
- **R2.1 - Legato:**Legato is stable on the older version of the Raspbian OS. On this it is also possible to run the demo apps.
- **R2.2 - AGL:**~20s booting on Raspberry Pi 3 and ~30s booting on MinnowBoard Turbo (2 - cores). The latest uses UEFI and the first legacy.
- **R2.2 - Legato:** ~21s with Raspberry Pi 3. However, Sierra Wireless suggests MangOH board is most suitable and perhaps takes less time..

- **R2.3 - AGL:**Resposiveness was tested on Raspberry Pi 3 and ~30s and MinnowBoard Turbo (2 - cores). The second was way more responsive than the first.
- **R2.2 - Apertis:**~45s booting on MinnowBoard Turbo (2 - cores).

- **R2.1 to R2.3-Legato:**not possible to test because only supported in manOH and the Hardware wasn't avaialbel for testing.
- **R2.4 -Legato**: based only on documentation
- **R2.4 - QNX**: QNX CAR platform allows Bluetooth pairing / adjusting Wifi settings / screen resolution as described by documentation. Was tested only for QNX 6.5 / 6.6 / 7.0 and not for QNX CAR platform extension due to licensing problems
- **R2.1 to R2.4 - QNX:** Not possible to test with the actual QNX platform for Automotive nor

the graphical interface because of licensing issues

- **R2.2 - QNX:**~6s booting on VMWare virtual machine. This information can't be actually compared with others because others were boot on actual Hardware.
- **R2.2 - Android Automotive:**~25s booting on Raspberry Pi 3.

Group 3: App Runtime

Requirement	W	AGL	Apertis	Ubuntu Core	SuSE embedded	Legato	QNX	Android Automotive	Comment
R3.1 Permission Transparency	1	0	0	0	0	0	0	1	• -2 - No permissions transparency can be supported • -1 - No permissions transparency exist • 0 - No permisions transparency exposed to the user • 1 - Permissions are exposed to the user • 2 - User can manage permissions for applications
R3.2 Permission Management per App	1	2	2	2	2	2	2	2	• -2 - Technically impossible to support it • -1 - No implemented means to support it • 0 - No permissions management per app available • 1 - Coarse grained permissions management per app available • 2 - Fined grainde permissions management per app available
R3.3 Isolation	1	2	2	2	2	2	1	2	• -2 - Technically impossible to support isolation • -1 - No implemented means to support isolation • 0 - No Isolation mechanisms supported • 1 - Basic isolation mechanisms supported (e.g. chroot, user/group, cgroups) • 2 - On top of 1, also security specific mechanisms supported.
R3.4 Resource Management	1	1	1	1	1	1	1	1	• Every platform defined resources differently and provides mechanisms to manage them. • That's why all al considered as fulfilled but no meaningful comparison could be made between them.
R3.5 Installation Management	1	1	1	2	2	1	0	2	• -2 - Technically impossible to support an installation management system • -1 - No implemented means to support an installation management system • 0 - No installation management supported • 1 - Application life-cycle operations supported (e.g. install, remove, start, stop) • 2 - On top of 1 applications authentication is done during installation.
R3.6 Error handling and reporting	1	1	1	1	1	1	1	1	• -2 - Technically impossible to support reporting or handling • -1 - No means to support reporting nor error handling • 0 - No error handling nor reporting mechanisms available • 1 - Reporting mechanisms available • 2 - Error handling and reporting mechanisms available
R3.7 Abstract access to vehicle data and functions	2	1	-1	-1	-1	-1	-1	-1	• -2 - Technically impossible to provide APIs to access the vehicle data and functions • -1 - Doesn't provide means to provide APIs to access the vehicle data and functions • 0 - Doesn't provide APIs to access the vehicle data and functions • 1 - Provide some APIs to access the vehicle data and functions • 2 - Provide APIs to acess most of the vehicle data and functions
R3.8 Abstract access to platform features	2	1	1	1	1	2	0	1	• -2 - Technically impossible to imposible to create APIs • -1 - No implemeted means to create APIs • 0 - No special APIs • 1 - At least means to create APIs • 2 - Already available APIs
R3.9 GUI support available	1	2*	1	1	2*	-1	2*	1	• -2 - Impossible to support a windows server nor GUI framework • -1 - Don't have any windows server • 0 - Windows server available • 1 - Single GUI Framework • 2 - Multiple GUI framework is available
R3.10 Multi-user support	1	0	0	0	1*	0	0	2	• -2 - Technically impossible to add multi-user support • -1 - No implemented means to add multi-user support • 0 - No multi-user support available • 1 - Partial multi-user support available • 2 - Full multi-user support available

Table 3.12.: Results for the App Runtime group (* see note)

Notes

- Why a requirements doesn't apply for a platform
- There is missing information about the requirement for a platform

- The requirement analysis was split and multiple scores were obtained. How the final score was calculated?
- **R3.7 - QNX:** CAR platform has applications to control features of the dashboard (e.g. velocity, engine RPM)
- **R3.9 - AGL:**Qt and HTML5/js based GUI frameworks supported.
- **R3.9 - SuSE Embedded:** At least Qt but other frameworks could be added on demand.
- **R3.9 - QNX:** Qt, HTML5/js, Photon supported.
- **R3.10 - SuSE embedded:** Could be provided through linux users although is not the best practice.
- **R3.3 -AGL** (user/group + cgroups + SMACK(Linux security Module) + Cynara (privilege enforcement system) + AppFW (Application Firewall)), **Apertis** (user/group + cgroups + AppArmor(Linux security Module) + polkit (privilege controlling system)),**Ubuntu Core**(user/group + cgroups +AppArmor(Linux security Module) and Seccomp), **SuSE Embedde**d (user/group + cgroups + AppArmor(Linux security Module), SELinux), **Legato** (user/group + cgroups + chroot + SMACK + ulimit), **QNX** (sandboxing feature available but no documentation found on the technology used),**Automotive Android** (Android generates a UUID for each app, only processes with this ID can access resources meant for this application)
- **R3.8 - QNX:** No concluding information was available

Justification for the scoring

- **R3.1 - Automotive Android:** Permission transparency is possible. Although not sure if this is also relevant for Automotive Android.
- **R3.3 -AGL** (SMACK), **Apertis** (AppArmor), **Ubuntu Core** (AppArmor and Seccomp), **SuSE Embedde**d (AppArmor, SELinux), **Legato** (chroot), **QNX** (sandboxing feature available but no documentation found on the technology used), **Automotive Android** (Android generates a UUID fior each app, only processes with this ID can access resources meant for this application)
- **R3.4 - AGL:** The score is 1 because AGL does have a resource manager but AGL doesn't give the app developer freedom to control resources for his app.
- **R3.4 - Apertis:** The score is 1 (should be between 0 and 1). Does not have a built-in manager but does have functions specifically meant for managing memory and handling processes and also provides guidelines for the same.
- **R3.4 - QNX:**The score is 1. Similar to Apertis. Provides a huge library for handing resources and guidelines with examples for using them.
- **R3.4 - Legato:** The score is 2 because the user can explicitly control the resource usage and these resources limits are interpreted by the platform automatically.
- **R3.4 - Android Automotive:** The score is 1 (tentative) because the result is based on Android O. Not sure if the same for Android Automotive.
- **R3.4 - Ubuntu Core and SuSE Embedded:** 0 because both do not have anything specifically for resource management.
- **R3.7.**-1 because no candidate focusses on this. All can get GPS (Apertis wraps it in some navigation lib). The studies do not agree on **AGL** one says a "through well defined interfaces" (but mentions no specifics)the other only mentions GPS. **QNX**is similar to the Linux although CAN/OBD is mentioned more (but CAN is easy on vanilla Linux to), but also no abstraction
- **R3.8. -SuSe embedded**: Intrusion detection using AIDE, Security-Enhanced Linux (kernel security module), NIS,SSSD, Samba
- **R3.8. -Android Automotive**: Security-Enhanced Linux (SELinux), Data access policies (i.e. System only, accessible from the applications with permission / without permission)
- **R3.10 - AGL:** But it's under development.
- **R3.9 - Ubuntu Core:** Qt, Deepin-UI.
- **R3.9 - AGL:** 2 because it supports Qt and Web technologies like HTML5 and JS also using a handful of Frameworks like Angular.js

Group 4: App Development and SDK

Notes

- **R4.1 - Apertis:** IDE is eclipse Juno based
- **R4.1 - Ubuntu Core**: No SDK available. Applications to be built with the standard SDKs meant for the HW and can be packaged using snapcraft.
- **R4.1 - SuSE Embedded:** SDK withour IDE available. Although provides and IDE for GNOME development.
- **R4.2 - SuSE Embedded:** The SDK is available for free, but updates require a paid subscription (See also comment on R4.1)
- **R4.2 - QNX:** SDK needs paid license (academic licenses and evaluation versions can be provided after registration with no cost)
- **R4.3 - Apertis:** Only C Programming is supported
- **R4.3 - QNX:** C and C++; Cordova based HTML 5 Apps
- **R4.4 - Apertis:** A lot of supported hardware but set-up instructions only available for Minnowboard
- **R4.4 - SuSE Embedded:** An image is available for Raspberry Pi 3, but requires a subscription that must be renewed (for free) yearly. Additional images can be built with SUSE Studio and/or Open Build Service. Image configurations need to be created and a hardware adaptation, which probably requires support from SUSE
- **R4.4 - Legato:** Currently works only with Sierrawireless HW. Thus 1.
- **R4.4 - QNX**: List of BSPs
- **R4.5 - AGL:** (Between 1-2) Server based development environment available. They can also be used locally on the development host using docker. But no emulator to test the graphical parts)
- **R4.5 - Apertis:** SDK available as a VM, possible to test graphical parts using an emulator. But the emulator seems to be buggy as the app seems to persist on screen even after the app is closed
- **R4.5 - SuSE Embedded:**Virtual test environments can be built with SUSE Studio and/or Open Build Service, or locally using the KIWI Image System. And multiple image formats available.
- **R4.5 - Legato:** Has a VM for only mangOH board.
- **R4.5 - QNX:** Has a VM based development environment, but no emulator to test the graphical parts.4.7
- **R4.5 - Android Automotive:** Scored as 1 because we are not sure if Android's emulator can be used for Android Automotive as well.
- **R4.6 - AGL:** Contains documentation on generating the SDK, Installing it. Additionally has documentation on downloading/building and installing the Server based development environment but no documentation on integrating it with an IDE (but there is a video on its usage after integration), The Application framework binding library has a documentation on usage, but it can be better. App structure documentation available. Additionally, documentation not available in a single location (Actual score between 1-2).
- **R4.6 - Apertis:** Has good documentation. Although some parts of the documentation is outdated (setting up a workspace). Has good documentation on the Apertis specific APIs, and also contains functional descriptions of the APIs. All of this available in the SDK VM.
- **R4.6 - Ubuntu core:** There is no SDK available. But does have documentation on the Interfaces and it's usage. Documentation also on introducing new APIs, but no functional descriptions available for any of the APIs. (Actual score 0-1)
- **R4.6 - Legato:** Legato has vast and updated documentation (Complete documentation available for every release) all in one place with a search engine available for it. Documentation available on setting up the development environment, the runtime library (usage with examples), testing and debugging. Documentation also available for Legato specific APIs and app

structure. Everything available on-line, on Eclipse IDE and as PDF.

- **R4.6 - QNX:** Tons of documentation available. but split as multiple documents some of which are available only to a registered user. (Actual score 1-2)
- **R4.6 - Android Automotive:** Again 1 because, although there is lots of documentation, there is nothing specific to Android Automotive.
- **R4.7 - AGL:** Has detailed documentation on the security concepts, application framework. Provides templates that can be used for writing an application and guidelines for using them. No mention of the system's performance and no specific coding guidelines.
- **R4.7 - Apertis:**Has guidelines of coding, resource management, security. Platform considers impact of security on performance. Documentation on Application framework is limited though.
- **R4.7 - Ubuntu Core:** Has documentation on security, interfaces and all the stacks of Ubuntu core. But no mention of performance considerations and no specific programming guidelines.
- **R4.7 - SuSE Embedded:**The platform architecture has less effect on the design and implementation of apps. No guidelines or standard specific to SuSE, No special performance or security guidelines.
- **R4.7 - Legato:**Uses a C-based runtime library to enhance performance, but no mention of it's impact on app development. Has guidelines for usage of Interfaces. But no specific programming guidelines.
- **R4.7 - QNX:** HTML5, QT 5.3, JavaScript, C/C++
- **R4.7 - Android Automotive:** Has vast documentation. But not sure if this is applicable to Android Automotive.
- **R4.8 - AGL:** AGL makes use of a building system based on Yocto. This enables us to extend the SDK and add additional tools. Thus 2
- **R4.8 - Apertis:**The SDK is available as a VM. Not sure if it can be extended. Although it is important to note that the SDK is integrated with Eclipse inside the VM. Thus -1.
- **R4.8 - Ubuntu core:** There is no SDK available for Ubuntu core. I.e we use an SDK meant for a specific hardware (say raspberry pi) and then package it for ubuntu core using snapcraft. Thus 0.
- **R4.8 - SuSE Embedded:** Limited customization. Gives the possibility to add Independent Software/Hardware Vendor packages to port applications from one platform to another, supported by SuSE.
- **R4.8 - Legato & QNX:** SDK is integrated into eclipse. Which perhaps gives us an option to extend.
- **R4.8 - Android Automotive:** Android studio does provide these options, but not sure if it is the same for Android Automotive.
- **R4.9 - AGL:** Requires Knowledge of QML/HTML5 and C++. If writing an HTML5 app or a native app that needs to access system resources, one needs to know how to write AGL specific bindings. Added to this one needs to set up the development environment.
- **R4.9 - Apertis:** For the development of applications one needs to know the following technologies: C, GLib, GIO, gSTREAM; Clutter, Automake. From the development environment, one needs to download the sdk vm and load it.
- **R4.9 - Ubuntu Core:** Development speed depends on the platform it is being built for. From the development environment, one needs to use the dev-kit meant for the platform he/she intends to develop the application and additionally needs to know how to make use of the snapcraft tool for packaging.
- **R4.9 - Legato:** Since legato is developed to be language independent, there are many legato specific APIs. One needs knowledge on this and also the runtime library. From the development environment, one needs to download the dev-kit and install it.
- **R4.9 - QNX:** Requires knowledge of QML/HTML5 and C++. Knowledge of QNX APIs is a must. Although knowledge level required is similar to AGL and additionally considering that the SDK is provided as an eclipse environment, it should be easier and faster to develop but

because of the unavailability of the SDK -> scored to 0!

- **R4.9 - Android Automotive:** Easy to work with Android Studio. Knowledge of HAL abstraction necessary.
- **R4.10 - AGL:** Currently, there is no integrated development environment that host all the necessary features. Also no single emulator for the graphical parts. Thus needs a better infrastructure.
- **R4.10 - Apertis, SuSE Embedded & Legato:Apertis** SDK comes in a VM with an eclipse based development environment and an emulator and thus not possible to extend it further. **SuSE** requires no special support but using some additional tools provided by SuSE would be beneficial. **Legato** also has an eclipse based development environment but has no emulator. but this is not in the scope of Legato.
- **R4.11 - AGL:** Possible to license applications. Applications based on C/C++ deployed as a shared library.
- **R4.11 - Apertis:** Possible to license applications. Applications deployed as compiled code bundles.
- **R4.11 - Ubuntu Core:** Possible to digitally sign the application and Possible to license applications.
- **R4.11 - Legato:** Applications can be licensed and deployed as binaries. Additionally possible to digitally sign the packages but needs an unpack tool to be implement on legato hooks and deployed on the platform.
- **R4.11 - QNX**: Pseudonumber generation when signing and uploading the packages
- **R4.11 - Android Automotive:** Possible to license and also possible to digitally sign the applications. But requires the developer to share 30% of the transaction to be paid to Google play store

Requirement	W	AGL	Apertis	Ubuntu Core	SuSE embedded	Legato	QNX	Android Automotive	Comment
R4.1 App SDK available	2 (Important for the start)	2	2	0	1	2	2	2	• 2 - SDK available with IDE or possible to integrate with IDE • 1 - SDK available without IDE • 0 - No SDK available
R4.2 App SDK accessibility	3 (Quite important for commercial usage)	2	2	0	-1	2	-1	2	• 2 - SDK is available and is OSS. • 0 - No SDK available • -1 - SDK available but needs a paid license.
R4.3 SDK language support	1	2	1	2	2	2	2	2	• 2 - Wide number of languages supported 3-4 • 1 - Supports limited number of languages. • 0, -1, -2 - Irrelevant for this requirement.
R4.4 Testing hardware accessibility	2 (Important for the start)	2	1	2	1	1	2	2	• 2 - Supports multiple architectures and also a number of boards and are cost effective • 1 - Limited number of HW support. • 0, -1, -2 - Irrelevant for this requirement.
R4.5 Virtual test environment availability	2 (Important for the start)	1	1	0	2	1	1	2	• 2 - VM/development host independent with complete infrastructure and ability to test apps on an emulated environment. • 1 - VM with limited capabilities or dedicated to only specific HW, or a development environment that can be hosted on a server. • 0 - No such above mentioned capabilities.
R4.6 SDK and App development documentation availability	3 (Highly important for the start: ensures parallel development of apps and platform)	1	1	0	0	2	2	1	• 2 - Clearly organized and structured documentation. • 1 - Documentation available, but needs more with better structuring. • 0 - Documentation available, but not specific to platform and/or is misleading with outdated data or is insufficient for the start. and/or broken links. • -1 - No documentation, but support available. • -2 - Useless.
R4.7 Platform Architecture and Guidelines	2 (Similar to requirement R4.6)	2	2	1	0	2	2	0	• 2 - Clearly organized and structured documentation. • 1 - Documentation available, but needs more with better structuring. • 0 - Documentation available, but not specific to platform and/or is misleading with outdated data and/or broken links. • -1 - No documentation, but support available. • -2 - Useless.
R4.8 Customizable SDK	1	2	-1	0	1	2	2	2	• 2 - Highly customizable (Yocto SDK or Eclipse based SDK) • 1 - Not customizable but possible to add additional libraries or add-on • 0 - No SDK available. • -1 - Not possible to customize the SDK • -2 - Irrelevant as -1 serves the purpose already.
R4.9 New App deployment speed	1	1	1	0	1	1	0 (see note)	1	• 2 - Intuitive, easy to develop and test, additional knowledge required is not so much. • 1 - Knowledge on platform must. • 0 - Knowledge on platform must, might needs additional libraries to be written. • -1 - time consuming due to lack of documentation or support. • -2 - Redundant. -1 pretty much serves the purpose.
R4.10 Infrastructure Support	1	1	0	0	1	1	1	1	• 2 - Good infrastructure available and accessible. Does not necessarily need additional support. • 1 - Infrastructure available. Needs additional set-up to speed up development and testing process. • 0 - Infrastructure available. Needs additional set-up to speed up development and testing process. But means to set-up not available. • -1 - No infrastructure available.
R4.11 Support for Know-How protection	3 (Highly important for commercial usage)	2	2	2	0	2	1	-2	• 2 - Licensed and digitally signed. • 1 - Licensed • 0 - No support in this direction. • -1 - Apps cannot be licensed. • -2 - Apps can be licensed but at the cost of a share in profit.

Table 3.13.: Results for the App Development and SDK group

Group 5: App Store

Requirement	W	AGL	Apertis	Ubuntu Core	SuSE embedded	Legato	QNX	Android Automotive	Comment
R5.1 App store availability	1	0	1	2	0 (see note)	0	1	1	• 2 - App store available and so is the framework • 1 - only App store available • 0 - No App store
R5.2 App store accessibility	1	0	0	2	0	0	0	-1	• 2 - OSS • 0 - No App store • -1 - Needs license
R5.3 App store management interface availability	1	0	1	2	0	0	0	0	• 2 - Interface available and can be customized (source code is available) • 1 - Interface available but cannot be extended • 0 - No App store
R5.4 Profile availability	1	0	0	2	0 (see note)	0	0	1	• 2 - Profiling based on Hardware • 1 - Limited profiling available • 0 - No App store
R5.5 Billing system availability	1	0	1	0 (see note)	0	0	0	2	• 2 - Complete billing system available • 1 - Supports a billing system (But no documentation about how it is supported) • 0 - No App store
R5.6 App authorization	1	0	1	2	0	0	0	1	• 2 - Infrastructure to authorize who upload apps and provide control over authorizers • 1 - Infrastructure to authorize who upload apps but no control over curators • 0 - No App store

Table 3.14.: Results for the App Store group

Notes

- **R5.1 - Apertis:** App store exists, but is currently down and no documentation available on it.
- **R5.1 - SuSE Embedded:** If platform has container support, then SuSE Gallery can be used.
- **R5.1 - Ubuntu Core:** App store available. Possible to create a Brand store for hosting a private store too. It is also possible for an OEM to host multiple stores.
- **R5.1 -QNX**: App World exists in QNX CAR 2
- **R5.1 - Android Automotive:** Android's play store is licensed.
- **R5.2 - Ubuntu Core:** The git repository of the brand store available. This makes it possible for us to extend the store with additional features.
- **R5.2, R5.4 and R5.5 - Apertis:** No documentation available in this direction.
- **R5.2 - QNX:** App store is integrated with the platform and is not free (requires platform license). The App store is OS-specific
- **R5.2 - Android Automotive:** Involves licenses to be dealt with. -> Thus -1.
- **R5.3 - Apertis:** There exists an App store curator who decides whether or not an application can be published. But there is no documentation on how this is achieved or regarding the guidelines.
- **R5.3 - Android Automotive:** Android's playstore does provide tools to fetch statistics, but in comparison to ubuntu core and apertis, there is no concept of a curator who controls the publication of an app and thus 0.
- **R5.2 & R5.4 - Ubuntu Core:** Each device is connected to only one store (But also possible to share apps between stores if necessary), this way it is possible to present apps meant for a particular car. Additionally the roles management enables us to control who publishes an app on a store.
- **R5.4 - SuSE Embedded:** Packages intended for different devices can be separated by, for

example, platform target configuration or repository.

- **R5.4 - Android Automotive:** Possible to filter applications based on device features, platform version and screen configuration.
- **R5.5 - Ubuntu core:** There is no billing systems, but it will be available soon (information from ubuntu forum).
- **R5.5 - Apertis:** Has a billing portal, but no documentation on how it works and no documentation on the the payment methods.
- **R5.6 - Ubuntu Core:** The roles management of the app store enable only authorized developers to publish apps which before publishing are reviewed by developers specified as reviewers using built-in tools or manually.
- **R5.6 - Apertis:** The developer registers the app and uploads it. The app curator reviews it checks for any violation and then publishes the app if everything is okay. But there is no documentation on the guidelines and thus 1.
- **R5.6 - Android Automotive:** Developer program policies must be taken into account before publishing an app. But no documentation about a review mechanism for the app and thus 1.

Group 6: Licensing

Requirement	W	AGL	Apertis	Ubuntu Core	SuSE embedded	Legato	QNX	Android Automotive	Comment
R6.1 Copyleft rules	1	1	1	-2	1	1	2	1	• -2 - Strong copyleft • -1 - Mostly strong copyleft • 1- Mostly weak copyleft • 2 - Weak or no copyleft
R6.2 License approval	1	2	2	2	2	-1	0	2	• -2 - All used licenses are not approved by OSI • -1 - Some used licenses are not approved by OSI • 0 - n/a • 1 - Some minor parts are under a non-approved license • 2 - All used licenses are approved by OSI
R6.4 Usage obligations	3	1	1	1	1	1	2	1	• -2 - Unreasonable obligations • -1 - Some obligations • 1 - No obligations but no support granted • 2 - No obligations and support granted
R6.5 Contribution obligations	1	2	-1	1	1	-1	0	0	• -2 - Unreasonable high hurdles • -1 - High hurdles • 0 - n/a • 1 - Low hurdles • 2 - Highly supportive structure
R6.6 Restrictions for commercial use	3	2	2	2	1	2	2	1	• -2 - Not allowed to be used for commercial products • -1 - High hurdles for commercial usage • 0 - n/a • 1 - Some minor restrictions • 2 - No restrictions
R6.7 License costs	3	2	2	2	-1	2	-2	-1	Three potential sources of costs: • Costs for customization/services/trademarks • Costs per device sold • Costs for the Development Environment • -2 - All three sources apply, • -1 - Two sources apply • 0 - One source applies • 2 - No costs

Table 3.15.: Results for the Licensing group

Notes

- R6.3 License Interoperability has been left out of the comparison due to a team decision

Justification for the scoring

- **R6.2 - AGL** different licenses for different modules, mainly Apache 2.0 and MPL 2.0
- **R6.2 - Apertis** Mozilla Public License Version 2.0
- **R6.2 - Ubuntu Core** Strong copyleft, Mostly GPL
- **R6.2 - SuSE Embedded** SUSE provides tools for managing the licenses in the platform making it possible to tailor, for example, a non-copyleft only platform.
- **R6.2 - Legato** The framework as such is licensed under MPLv2. But there are exceptions where some packages are licensed under LGPL v2+, Apache-2.0, BSD License, MIT License, RSA License, Paul Hsieh exposition license and a lot more.
- **R6.2 - QNX** Commercial License, no copyleft
- **R6.2 - AGL, Apertis, Ubuntu Core, SuSE Embedded, Android Automotive** All used licenses are approved by OSI
- **R6.2 - Legato** Paul Hsieh exposition license is not approved by OSI
- **R6.2 - QNX** Not applicable since commercial license
- **R6.4 - AGL, Apertis, Ubuntu Core, SuSE Embedded, Legato, Android Automotive** No usage obligations but also no support granted
- **R6.4 - QNX** No usage obligations, support is part of the contract

- **R6.5 - AGL** Well organized and structured contribution procedure
- **R6.5 - Apertis** Several accounts needed, account support not very responsive
- **R6.5 - Ubuntu Core** Well organized and structured but quite demanding contribution agreement needs to be signed
- **R6.5 - SuSE embedded** Quite structured and supportive contribution procedure
- **R6.5 - Legato** Accounts needed, quite demanding contribution agreement needs to be signed
- **R6.5 - QNX** Not applicable since not open source
- **R6.5 - Android Automotive** Not clear how to contribute
- **R6.6 - AGL, Apertis, Ubuntu Core, Legato, QNX** No restrictions for commercial usage
- **R6.6 - SuSE Embedded** The restrictions for commercial use depend on the components chosen for the platform
- **R6.6 - Android Automotive** Some restrictions regarding the google-owned parts and the Android trademark
- **R6.7 - AGL, Apertis, Ubuntu Core, Legato** No costs
- **R6.7 - SuSE Embedded** Pay per device and Pay for a support contract
- **R6.7 - QNX** Pay per device and pay for additional contracts and pay for the IDE
- **R6.7 - Android Automotive** Pay per device if using the google-owned modules (such as Play store), potentially pas for the trademark

Group 6: Developer Community

Requirement	W	AGL	Apertis	Ubuntu Core	SuSE embedded	Legato	QNX	Android Automotive	Comment
R7.1 Community size	3	2	-2	0	1	0	1	1	• -2 - Small Community with only very few supporters • 0 - Decent Community size with some industry support • 1 - Big Community, but potentially no focus on automotive • 2 - Huge, major support by the industry
R7.2 Community liveliness	2	2	-2	2	1	0	1	0	• -2 - Almost dead • 0 - Medium activity • 1 - Quite lively • 2 - Really active
R7.3 Community responsiveness	2	1	-2	1	2	-1	2	0	• -2 - Slow (days, sometimes weeks) • -1 Rather slow (hours, sometimes days) • 0 - Not measurable • 1 - Fast (within minutes) • 2 - Dedicated support available
R7.4 Community management	1	2	0	0	1	0	0	1	• 0 - Well structures • 1 - Well structured and somehow managed • 2 - Well structured and transparent management
R7.5 Community members	3	2	-2	1	1	-1	-2	1	• -2 - Run by a single institution • -1 - Mainly run by a single institution, only minor contributions from others • 1 - Decent mixture of institutions with only a few big players • 2 - Good mixture of institutions with a lot of big players
R7.7 Project Road-Map	1	2	-1	-2	1	-2	1	1	• -2 - No roadmap available • -1 - Rough or short-term roadmap available • 1 - Roadmap available • 2 - Roadmap available and mapped to the development tools
R7.8 Code reusability	2	2	0	1	1	1	1	1	• 0 - Provides a basis only • 1 - Some features could be used • 2 - Most features could be used
R7.9 State of the Project	1	2	-1	2	2	0	2	0	• -1 - Projects future seems to be questionable • 0 - Stable but future unclear • 2 - Stable and promising future
R7.10 Relevant related projects	1	2	-2	1	-2	0	2	0	• -2 - none • 0 - Plans only • 1 - Some use in commercial projects • 2 - Heavy usage in commercial projects

Table 3.16.: Results for the Development Community group

Notes

- R7.6 Project lines of code has been left out of the comparison due to difficulties in measuring
- For SuSE embedded the Open SuSE Community was inspected
- For QNX, the open source community foundry27 and OpenQNX were examined

Justification for the scoring

- **R7.1 - AGL** Huge Communtiy, 100+ company members (incl. 10 OEMs), big number of contributors
- **R7.1 - Apertis** Only a few number of members, all by one company
- **R7.1 - Ubuntu Core** While the Ubuntu Community is huge, the Ubuntu Core Community is rather small (about 1000 users)
- **R7.1 - SuSE embedded** Well sized Community (about 40000 users)
- **R7.1 - Legato** Decent sized Community (thousands of users on the forum, but only 21 contributors), strong dependency to SierraWireless
- **R7.1 - QNX** Well sized Community (more than 25000 users)

- **R7.1 - Android Automotive** Almost 400000 users on the forums and a huge number of contributors on Android itself, however only a few seem to actually look on Automotive
- **R7.2 - AGL, Ubuntu Core** Really active (6.7 posts / day for AGL plus active mailing lists and very active IRC, 48 posts / day for Ubuntu)
- **R7.2 - Apertis** Almost dead (only 2-4 mails / month)
- **R7.2 - SuSE Embedded, QNX** Quite lively (25 mails / day for SuSE embedded, approx. 20 posts / day for QNX)
- **R7.2 - Legato, Android Automotive** Medium activity for Legato (9 topics / week) High activity for overall Android (post every 2 minutes) but only very few are for Automotive
- **R7.3 - SuSE Embedded, QNX** Dedicated support engineer is part of the contract
- **R7.3 - AGL, Ubuntu Core** Responses often within minutes
- **R7.3 - Apertis** Responses take days if not weeks
- **R7.3 - Legato** Most responses came after a couple of hours
- **R7.3 - Android Automotive** Not measureable due to the lack of a specific Automotive community
- **R7.4 - Apertis, Ubuntu Core, Legato, QNX** All are quite structured but lack of an active management (Apertis) or the management is intransparent (Ubuntu Core, Legato, QNX)
- **R7.4 - AGL** Well structured with dedicated management boards
- **R7.4 - SuSE Embedded, Android Automotive** Well structured but with some doubts regarding management transparency
- **R7.5 - AGL** More than 100 partners, mainly from industry, big players such as Toyota, Denso or ARM
- **R7.5 - Apertis, QNX** Run by a single institution
- **R7.5 - Ubuntu Core, SuSE Embedded, Android Automotive** Good mixture of companies
- **R7.5 - Legato** Mainly run by SierraWireless, some contributions from different institutions
- **R7.7 - AGL** Long-term roadmap available and mirrored to JIRA
- **R7.7 - Apertis** Rough and outdated roadmap only
- **R7.7 - Ubuntu Core, Legato** No roadmap available
- **R7.7 - SuSE Embedded, QNX, Android Automotive** Roadmap available
- **R7.8 - AGL** Promises to provide 70-80% of the platform
- **R7.8 - Apertis** Provides a basis only
- **R7.8 - Ubuntu Core, SuSE Embedded, Legato, QNX, Android Automotive** Provide many useful features
- **R7.9 - AGL, Ubuntu Core, SuSE embedded, QNX** All stable and in commercial use
- **R7.9 - Apertis** Implemented features seem to be quite mature, however there are major doubts on the further life of the project
- **R7.9 - Legato, Android Automotive** Stable but rely on a single institution
- **R7.10 - AGL, QNX** Heavy usage in the automotive industry
- **R7.10 - Apertis, SuSE Embedded** none
- **R7.10 - Ubuntu Core** Used in industrial automation projects (e.g. by DELL)
- **R7.10 - Legato, Android Automotive** Some plans to introduce it into commercial projects exist (e.g. VW)

Overall Results

The following paragraphs try to summarize the detailed examination results given above.
Automotive Grade Linux is exceptionally strong when looking at the developer community group. It offers a huge and vibrant community alongside with a stable governance process and feature roadmap. It also offers a good amount of features and ran stable throughout all our tests. While offering an app runtime environment with UI support and isolation features as well as an free and open source deveopment kit for creating apps, it lacks of a binding to an app store implementation. In summary, the verdict regarding AGL can be formulated as follows:

Automotive Grade Linux is the reference in terms of community and organization with an extensive automotive feature set but lacking of an app store.

Apertis on the other hand lacks an active development community. Furthermore, the number of features implemented is for example in comparison to AGL rather low. While it offers an application runtime and a quite well implemented and documented app development environment, the app store stated as a feature was unavailable at the time of the study execution. Finally, the examined image for the Minnowboard ran quite stable. As a summary, this leads to the following verdict:

Apertis has virtually no community. While it is aimed at the automotive infotainment use-cases it does not include much automotive-specific functionalities.

Ubuntu Core is developed to be used as a headless operating system within the IoT. This leads to the fact that the set of features focuses more on things like updatability and security rather than on automotive specific issues. It is a stable and responsive platforms with a well implemented app runtime environment in form of a snap system that is already connected to an app store environment. However, regarding the app development the support and documentation is rather limited. The license situation is quite well fitting to the APPSTACLE requirements. The developer community highly depends on Canonical but is quite active and responsive. These facts lead to the following verdict:

Ubuntu Core is a reasonable Linux System geared for embedded usage that put a lot of thought into updating and deployment of the platform and apps. Small, but active community, yet no relation to automotive.

Just as Ubuntu Core, SuSE embedded tries to position itself as a base OS for the IoT. In this regard, it focuses on security and updatability but lacks of specific automotive features. During our examination, the Raspberry PI 3 image ran stable but was not very responsive. SuSE embedded offers an app runtime environment with isolation and UI support. Additionally, the system is one of the few that inherently comes with multi-user support. However, it does not offer an app store and the development environment is under a commercial license by SuSE. This issue in conjunction with the fact that SuSE charges their customers per device sets a high commercial barrier for being used within APPSTACLE. In summary, the verdict regarding SuSE embedded is formulated as follows:

SuSE Embedded is a trimmed down desktop/server version. The business model is not a good fit for APPSTACLE. It is based on customization for a customer by SuSE and licensing fees.

The outstanding feature of Legato is definitely its well maintained and documented app development environment. Additionally, it offers a well-featured app runtime environment as well as a good

set of features for ex-vehicle communication. On the other hand it lacks additional automotive features especially regarding in-vehicle connectivity and an app store. While the licenses used are a good fit for APPSTACLE, the development community is rather small. Summarizing all these findings the verdict on Legato is as follows:

Legato has an excellent App Development Kit and platform documentation. Hardware support is limited and the community is still small.

QNX has turned out to be a average in most of the requirements. This applies for example to the platform features group or the runtime experience. While it offers a well suited app runtime environment with features like isolation or permission management, the development environment (which is also well maintained and feature-rich) is subject to a charge. The same applies to the app store and the OS itself. Surprisingly, the QNX community is rather large and active taking into account that it is a commercial product. Bringing all these findings together, the verdict regarding QNX is formulated as follows:

QNX is a closed, proven platform that is already used in vehicles from various OEMs. There are safety certified and real-time capable QNX derivatives. It is a commercial platform that requires licenses.

Android Automotive profits from many features of its parent project Android OS. This applies especially to the app runtime, app development and app store groups. Admittedly, these services do not come for free as selling apps over Google's play store is subject to a 30% revenue share. In other areas the automotive feature set is rather small and often lacks a good documentation. While all licenses are OSI approved, there is a charge per device if the more or less unavoidable play services (e.g. play store) are integrated. Additionally, trademark issues have to be clarified when using it in a commercial product. Finally, while the Android developer community is huge and very active, it is hard to filter out the the actual effort taken to develop the automotive features as there is no clear line of demarcation between Android itself and its automotive derivate. In summary, the verdict regarding Android Automotive can be formulated as follows:

Android Automotive is an automotive focused extension of the Android ecosystem. Currently it is hard to see what is only in the architecture and what the implemented scope is. The relation to Android and Android Auto is not quite clear.

Figure 3.4 illustrates the strengths and weaknesses of the different candidates with regard to the seven requirement groups. Examining the Platform Feature group, although the candidates are quite different in their feature sets, most of them are quite close together when it comes to the overall scores (between 9 (QNX and Andorid Automotive) and 13 (Legato) points). Only Apertis is a bit behind with only 6 points. The Platform Runtime analysis on the other hand shows no big differences between the candidates (between 1 point for SuSE Embdedded and 3 points for Ubuntu Core). Examining the App Runtimes, three candidates stand out: AGL (13 points), Android Automotive (12 points) and SuSE Embedded (11 points) offer the richest feature set. On the other end, QNX has a long list of shortcomings (5 points). Differences between the candidates are very apparent regarding the App Development and SDK group. Here, two candidates do really stand out (Legato with 36 and AGL with 35 points), followed by Apertis (28 points), QNX (25 points) and Android Automotive (21 points). At the bottom of the list, Ubuntu Core (14 points) and SuSE Embedded (10 points) do lack a lot of the bells and whistles the other candidates offer. The App Store evaluation showed that three of the candidates do not offer an App Store at all and hence did not receive any points. The most convenient and advanced solution is offered by Ubuntu Core (10 points), followed by Apertis and Android Automotive (4 points each) and QnX (1 point). Investigating the results

of the Licensing examination, one can separate between two groups: the ones that are proprietary or under strong commercial influence (Android Automotive (6 points), SuSE Embedded (7 points), QNX (8 points)) on the one hand, and those that have a more open character (AGL (20 points), Apertis (17 points), Ubuntu Core (16 points) and Legato (14 points)). The last requirements group Development Community again showed quite some differences between the candidates. On the lower end, Apertis (-24) and Legato (-5) even received a negative scoring while AGL on the other end of the scale stood out with 30 points. In the midrange SuSE embedded (16 points), Ubuntu Core and Android Automotive (12 points each) as well as QNX (10 points) are quite close together.

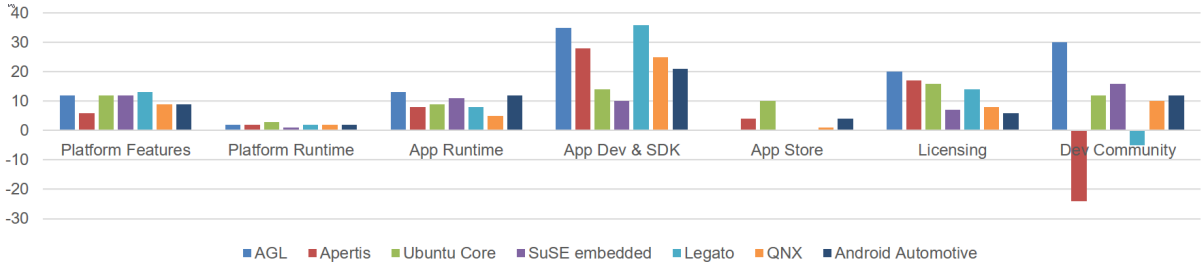

Figure 3.4.: The results of the different requirements groups

The results of the different requirement groups are summarized and illustrated in Figure 3.5. While there is one clear leader regarding the overall points (AGL with 112 points) and Apertis clearly ending up in the last place (41 points), the remaining candidates are quite close together: Ubuntu Core (76 points), Legato (68 points), Android Automotive (66 points), QNX (60 points) and SuSE Embedded (57 points).

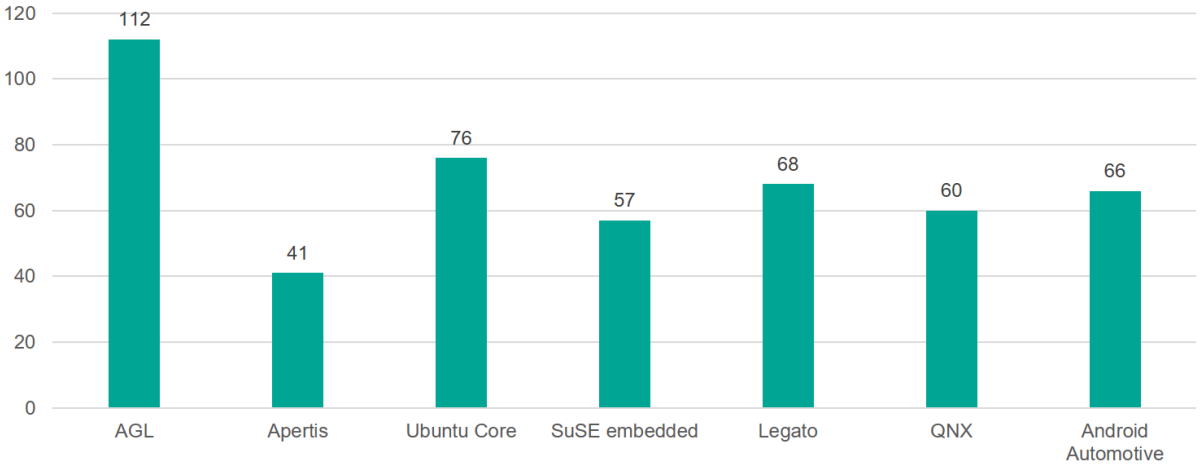

Figure 3.5.: The overall results of the platform study

3.2. Automotive APIs

3.2.1. Scope

This chapter introduces and discusses current approaches to create an Application Programming Interface (API) for vehicles. All of them try to achieve this by (a) merging the potentially very complex device and network structure of a car into a single virtual device and (b) hiding the differences between manufacturers, models and makes behind a common interface. On the other hand these interfaces strongly differ in their scope (data-subset or use-case), technological approach and creators.

3.2.2. Overview

AUTOSAR

AUTomotive Open System ARchitecture (AUTOSAR) is a cooperation between car manufacturers, OEMs and tool manufacturers and defines a software development paradigm for Electronic Control Units (ECUs) in the automotive domain. In order to separate the development process of application software from the chosen ECU hardware platform, AUTOSAR is introducing a layer model with the three layers Application Software, Runtime Environment and Basic Software (illustrated in Figure 3.6).

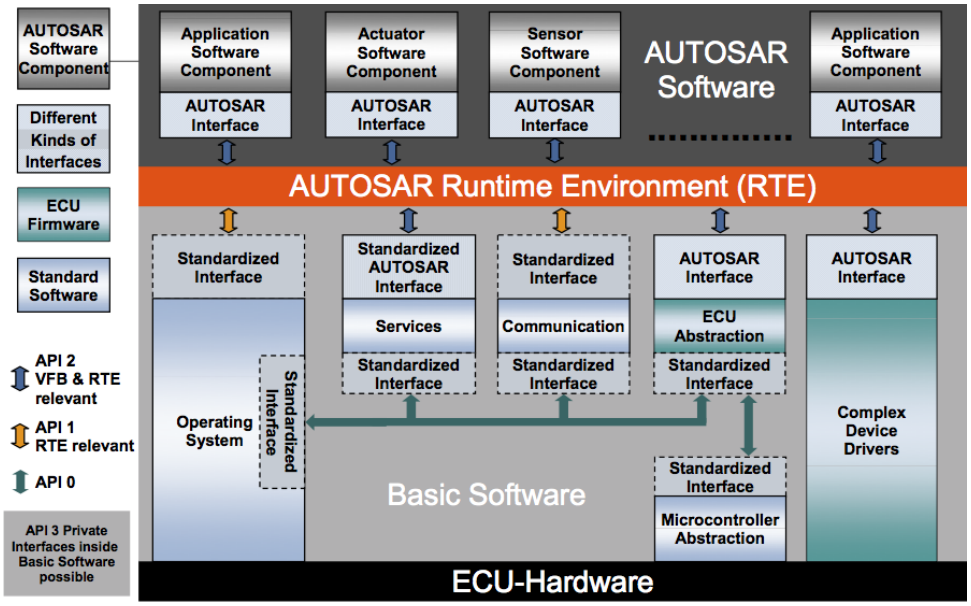

Figure 3.6.: The AUTOSAR Layer Model [11]

The top layer is formed by the application software. It is divided into software components, each of which realizes a part of the application and can consume and provide data via so-called ports. Any communication that does not take place via port connections is forbidden. A port is classified via a port interface (here referred to as interface). Two ports can only be connected to each other if both ports use compatible interfaces.

Two important communication paradigms, that are selected by interfaces, are client-server and sender-receiver communication [11]. For client-server communication, a server component provides functions (C, C++) which can be called by clients. A 1:n communication is also possible (i. e. a server can provide its functionality to several clients). In sender-receiver communication, a sender provides data that can be consumed by receiver components. Both 1:n and m:1 communication is possible here (i.e. a date can be consumed by several components or several senders provide a date for one receiver concurrently). Many-to-many communication is not provided.

The lower layer consists of the basic software and contains the hardware drivers, the operating system and the communication stack. The communication stack handles communication from and to other ECUs that are connected via network interfaces like CAN, LIN, Flexray, automotive Ethernet, etc.

All communication, whether between software components on the upper level or between software components and basic software on the lower level, is realized via the runtime environment (RTE), which forms the middle layer. The RTE specification document [12] defines a schema for API functions (C, C++), which are usually generated by code generators of the AUTOSAR modeling tools according to the modeled communication between software components and basic software. All communication must take place via the (generated) API functions. Other communication is not permitted. Likewise, all communication interfaces must be defined at the time of development, which makes it impossible to dynamically extend the software architecture at runtime.

GENIVI Common API

The CommonAPI is an abstract API that intends to enable service-oriented communication in distributed systems [49] . It is an open source project within the GENIVI alliance and provides implementations in C and C++. GENIVI makes strong use of an interface description language called Franca IDL. The basic idea behind the GENIVI CommonAPI is to describe interfaces at development time using Franca IDL and use the tools provided in order to generate the code of the actual interface. In order to create the Franca IDL descriptions, tooling such as an Eclipse-based editor [40] is available. The two main communication technologies on which CommonAPI is focussing are the service-oriented, ethernet-based, automotive protocol SOME/IP and the inter process communication mechanism D-Bus.

GENIVI Vehicle Signal Specification

The GENIVI Vehicle Signal Specifcation (VSS) is a specifcation of an in-vehicle data interface. It was created at the GENIVI consortium mainly to overcome the lack of a standardized interface for vehicular data [45]. In order to create a technically simple, lightweight and easy to change interface, the VSS follows a tree approach organizing the different signals in branches (illustrated in Figure 3.7). This allows to

- work with subsets of the overall specification (e.g. if some sensors are not present in a specific car)
- extend the specification without versioning overhead (e.g. by adding additional branches or signals)
- enrich the interface by non-standardized additions (e.g. OEM specific interfaces)
- discover the actually available signals and branches online

The VSS hereby adopts elements from other GENIVI standards like for example the set of signal types from FrancaIDL [77]. The VSS is described using a single YAML file including all branches and signals as self-contained list elements.

W3C Vehicle Information Services Specification

The W3C Vehicle Information Services Specification (VISS) is a WebSocket-based interface specified by the World Wide Web Consortium [158]. This specification is the successor of two former specifications by the W3C Automotive Working group, namely the W3C Vehicle Signal Server API and the Vehicle Information Access API. It is meant to be an abstraction layer between the automotive domain and all of its specifics and the web domain. Therefore, it provides the content of the GENIVI Vehicle Signal Specification (VSS) from the automotive domain behind a WebSocket-based, service-oriented API. Additionally, the VISS adds features like a discovery mechanism to allow the detection of available data, a token-based access control mechanism and he introduction of commonly used access mechanisms such as get and set methods or a publish/subscribe functionality. As an overall concept, this specification makes use of a so called signal tree approach [142]. This approach

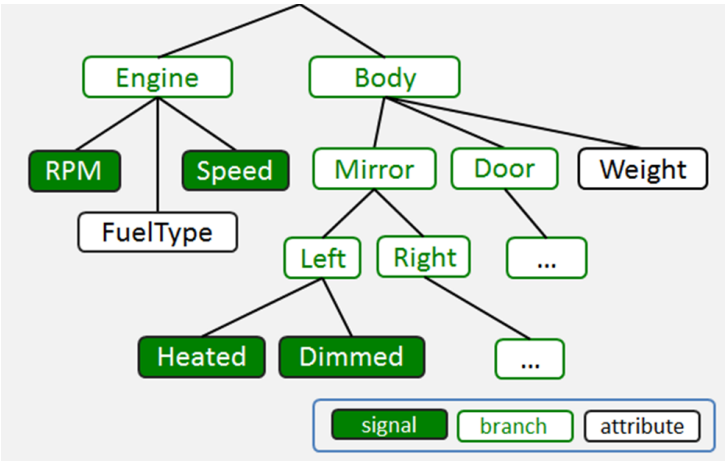

Figure 3.7.: An excerpt of the Vehicle Signal Specification [77]

does not define a "fat library" containing all interactions and coded functions, but provides a simple interface with get, set, subscribe and unsubscribe as the only functionality. The data itself is arranged in a tree style which allows discoverability, different subsets of content and easy extendibility at the same time. As illustrated in Figure 3.8, the VISS is intended to be used by different types of clients. On the one hand, it is meant to be used by native applications deployed on the same ECU. On the other hand, the W3C specification does also envision the deployment of language bindings as clients. In Figure 3.8, a JavaScript library acts as a client to the VISS and provides the services of the server via a JavaScript API.

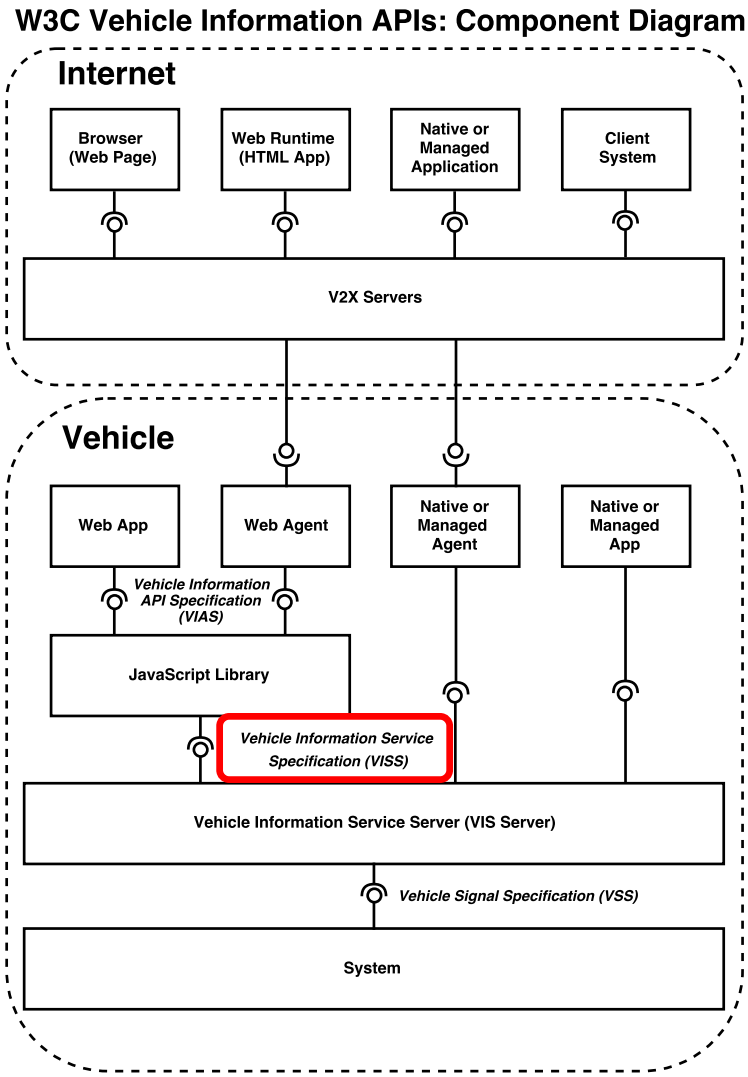

Figure 3.8.: Overview of the W3C information API structure [158]

On-board diagnosis

While not strictly being an "API", the on-board diagnosis (OBD) interface is one of the most important interfaces today for accessing vehicular data. The story of the OBD interface started in 1988 when the California Air Resources Board (CARB) launched the first stage of its legislation [85]. The very first version of the OBD had to monitor emission-related components of the car and to signal any malfunction using a malfunction indicator lamp. In the second stage (OBD II) introduced in 1994, the set of functions to be monitored was extended to all of those potentially increasing toxic exhaust as well as to the monitoring functionality itself. This legislation was adopted by other administrations as well, e.g. by Europe in 2000. The legislation does not only define the set of data to be available but also the protocols used to access them (e.g. ISO 14230 (Keyword Protocol 2000) [139] or ISO 15765 (CAN)) as well as the connector used to access the internal network. While the legislation itself focuses on data regarding the exhaust system, the OBD II connector slowly became the most used diagnosis port in the automotive domain. This is due to the fact that the connector, once integrated into the vehicle, was used by the car manufacturers for additional vehicle diagnostics as well. Due to this development, the data accessible at this connector strongly depends on the manufacturer and model of the car and only the legislative part is accessible in a standardized way [41].

3.2.3. Discussion

As the APPSTACLE project wants to support a manifold list of potential usage scenarios instead of being restricted to a single use case the shape and extent of the API cannot be tailored but needs to be open and flexible. Hence, the analysis of the API approaches focuses on the following requirements:

- **Flexibility** The API should be able to be extended and or tailored.
- **Provide Content** The API should actually define some content rather than only providing a technological framework.
- **Discovery** The API should allow to discover the content actually available at a given instance of the API.
- **High level of abstraction** The API should hide the manufacturer, model and make specific characteristics.
- **Access Control** The API should provide ways to grant or restrict access to specific elements.

AUTOSAR is a widely used framework that benefits from a high number of software implementations and tooling solutions. However, AUTOSAR does not define any content available at the API but restricts itself to creating an API schema instead. Moreover, access control is not directly integrated and the level of abstraction depends on the actual implementation created on the basis of the API schema. Finally, there is no discovery mechanism provided by the AUTOSAR RTE.

While the GENIVI CommonAPI provides a widely applicable framework to design and use APIs alongside with optimal tooling support, it suffers of several drawbacks, too. First of all it is a quite static approach that does not allow any changes of the API after development time. Second, CommonAPI framework tends to create a significant overhead to the development cycle by requiring extensive tooling and expecting a high level of understanding from the developer. Finally, CommonAPI does not specify any standardized content provided at the API or any additional features such as access control of data discovery.

The GENIVI Vehicle Signal Specifcation is a very flexible and easy to use interface specification. Due to being created at GENIVI, it is quite well-known in the automotive industry. Due to its tree structure, it is very flexible and allows to add discovery functionality. Additionally, it describes the concrete content behind the interface. However, as VSS is not meant to be used stand-alone but only as an input for other standards [45], it does not contain actual functions for example to manage access control.

W3C Vehicle Information Services Specification: The W3C VISS is an interesting approach to combine the automotive world with the web domain. It establishes a bridge between the GENIVI Vehicle Signal Specification and WebSockets, a commonly used technology in the web. Besides providing a standardized API, The VISS adds valuable features such as content discovery and access control mechanisms. Additionally, by making use of the GENIVI Vehicle Signal Specification, it offers a set of available content without creating barriers for OEM specific extensions.

While OBD is being heavily used in the automotive industry it does not fulfill all the requirements regarding an APPSTACLE API. For example, although it defines a rich set of data to be accessed, it is too much restricted on diagnostic information for combustion engines. Additionally, it is not flexible in its set of resources nor does it define an access control scheme. Finally, as it is rather an electrical interface than an API, there is no standardization on how it can be accessed via software.

Besides the in-vehicle API approaches discussed here, some elements of cloud-based APIs could be considered as well. This includes e.g. scientific approaches such as the Common Vehicle Information Model (CVIM) [119] which defines an open and harmonized data model, allowing the aggregation

of brand independent and generic data sets. Additionally, ISO standards like the Extended Vehicle (ExVe) [140] which define a complete web interface to exchange vehicular including a rights and roles concept. Finally, open specifications such as the SENSORIS car-to-cloud interface [60] could give valuable input for the data set actually needed in state-of-the-art applications.

3.3. In-vehicle Connectivity

This section provides an overview of the communication protocols that are currently used in the existing automotive architectures as well as their interconnections in the Electrical / Electronic (E/E) in-vehicle architecture.

3.3.1. Scope

For the sake of brevity, we restrain the scope only to the protocols that are in use in the network architecture of modern cars. These protocols also define the in-vehicle communication interfaces for the APPSTACLE platform (Section 3.1).

3.3.2. Overview

Architecture

Modern automotive embedded systems consist of several subsystems, which are comprised of one or several Electronic Control Units (ECUs). In turn, the ECUs are made up of a micro-controller and a set of sensors and actuators. They are able to communicate through the transmission of electronic or optical signals through a dedicated communication unit. The subsystems that rely on network communication in automotive systems are divided into five main categories: power train, chassis, body, HMI, and telematics (illustrated in Figure 3.9). Each subsystem uses a different protocol to communicate, which is selected based on the architectural requirements and the subsystem functionality. Specifically, the powertrain domain is related to the systems that participate in the longitudinal propulsion of the vehicle, including engine, transmission and all subsidiary components. This domain is supported by a dedicated subsystem called *Drive CAN* using the Controller Area Network (CAN) [18] for data exchange. The chassis domain refers to the four wheels and their relative position and movement; in this domain the systems are mainly steering and braking. In this subsystem category we find two protocols that are used for high-critical communication, namely CAN and FlexRay [46], as well as the Local Interconnect Network (LIN) [93] for the lower critical functionalities (e.g. door locking, window raising / lowering). According to the EAST-EEA [1] project definition the body domain includes the entities that do not belong to the vehicle dynamics (i.e., being those that support the car's user) such as airbags, wipers, lighting, etc. Today's cars sometimes use two CAN buses (*peripheral CAN* and *body CAN*) which interconnect the ECUs of the comfort domain. The telematics domain includes the equipment allowing information exchange between electronic systems and the driver (displays and switches). Such interactions are possible through the infotainment subsystem that is supported by the MOST protocol [106]. Finally additional peripheral systems (e.g., cameras) allow the in-vehicle system to monitor and extract information from its physical environment through the use of Automotive Ethernet technologies [57]. All the aforementioned systems are able to exchange data through a central gateway (Figure 3.9) that is able to map (through packet encapsulation) or forward messages from one subsystem to another.

Protocols

Automotive protocols are classified by the Society of Automotive Engineers (SAE) into four categories according to the transmission rate and their role in the automotive architecture. Specifically, Class A defines the protocols that are used for convenience systems (e.g. lighting, windows, seat controls) and require inexpensive, low-speed communication. Class B defines the protocols supporting instrument cluster or vehicle speed communication and require medium-speed communication.

[1] itea3.org/project/east-eea.html

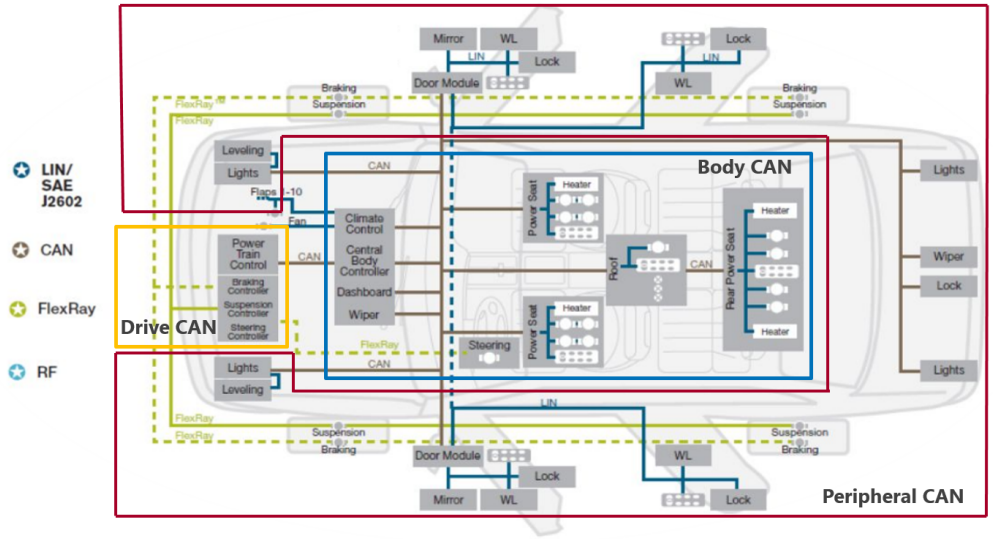

Figure 3.9.: Automotive network

Furthermore, Class C is defined for real-time control ECUs such as the engine, braking and steer-by-wire and require high-speed communication. Finally, telematics systems usually require higher communication speed for multimedia (audio / video) and navigation, and therefore SAE defined the additional Class D communications. All four protocol Class categories are illustrated in Table 3.17 along with the protocols that belong to each category and are used for in-vehicle communication in terms of their characteristics.

Bus	LIN	CAN	CAN FD	FlexRay	MOST	Automotive Ethernet
Used in	Subnets	Soft real-time	Soft real-time	Hard real-time	Multimedia	Multimedia
Application domains	Body	Powertrain, Chassis	a	Chassis, Powertrain	Multimedia and Telematics	Telematics and active safety
Message transmission	Synchronous	Asynchronous	a	Synchronous and Asynchronous	Synchronous and Asynchronous	Synchronous and Asynchronous
Access control	Polling	CSMA/CA	CSMA/CA	TDMA	CSMA/CA	CSMA/CD
Maximum Data Rate	20 kbps	1 Mbps	10 Mbps	10 Mbps	24Mbps	100Mbps
Protocol Class	A	BC	D	D	D	D

Table 3.17.: Characteristics of the communication protocols

LIN

The most common SAE Class A protocol (transmission speed less than 10Kbit/s) is Local Interconnect Network (LIN) [93]. LIN is a low-cost serial communication system enabling fast and cost-efficient implementation of multiplex systems vehicles networks. It is mainly found in parts of the architecture where the implementation of higher-bandwidth multiplexing networks is not required. We usually use it for controlling doors, windows or power seats. LIN has a data transmission rate up to 20Kbit/s and uses single wire for the connection between the nodes. In addition to using a time-triggered approach, LIN is a broadcast serial network comprising a maximum of 16 nodes (one master and typically up to 15 slaves). The scheduling in the LIN bus is done by the system designer during the development based on the type of message sent by the nodes and is organized in slots based on message transmission time. The slots are divided into mini-slots in which the master node processes the schedule for the communication. Figure 3.10 illustrates an example of communication in a LIN network. In this example the locking system acts as a Master and broadcasts a message to all the ECUs on the network to request information (as shown in Slot 1), but also to trigger the communication of

other ECUs between them (e.g. Roof Unit sending a close command to the Window ECU in Slot 2).

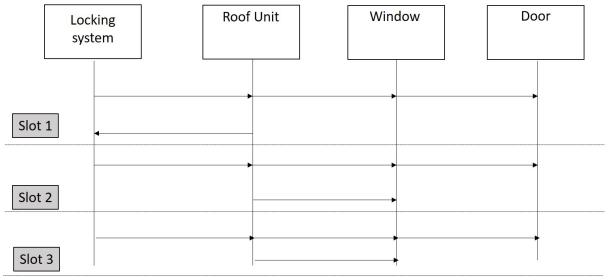

Figure 3.10.: LIN system example

CAN

Safety-critical applications on the car such as powertrain require higher bandwidth data transmission protocols. For this reason SAE defines the Class B and Class C protocols where we find the commonly used Controller Area Network (CAN) [18]. CAN was invented by Bosch to resolve the problem of wiring from point-to-point connections, as the number of ECUs inside the car was growing. To this end, CAN's purpose was to reduce the number of wires through a serial bus system connecting all the ECU's. Gradually, its use expanded in the domain of automotive embedded systems, due to the efficient, yet simple, Medium Access Control (MAC) mechanism and the ease of deployment it offers. The CAN protocol is defined by the communication standards ISO 11898-1 [72] and ISO 11898-2 [73] as CAN 2.0.

Message exchange is handled by the CAN station, which includes all functional units of a Basic CAN Controller [118], such as the CAN Protocol Controller as well as the hardware acceptance filtering mechanism. The CAN Protocol Controller (often referred as CAN Protocol Handler) is responsible for all messages transferred via the bus.

An example of an automotive system using the CAN protocol is illustrated in Figure 3.11. It presents a set of automotive control units, such as the engine and traction control systems, that exchange messages through a serial bus system.

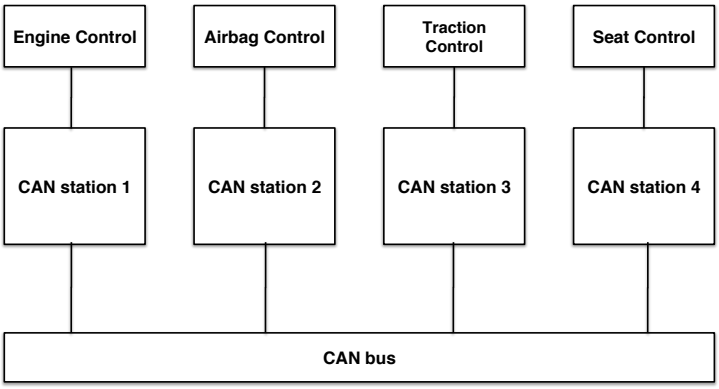

Figure 3.11.: CAN system example

CAN uses the Carrier Sense Multiple Access Collision Avoidance (CSMA/CA) approach in order to solve bus contentions deterministically. Its protocol stack implements only the physical (PHY) and the data link (DLL) layers of the OSI reference model, thus reducing the message processing delays and simplifying the communication software. The physical layer is responsible for data transmission, the data link layer for managing the access on the bus. CAN is a message-oriented transmission protocol based on a multi-master access scheme to a shared medium. CAN messages are denoted as frames and are assigned with a unique identifier which defines both the content and the priority of the frame. The absence of source or destination addresses facilitates the addition of new devices

to the network, without stopping its operation. Another advantage is the network multi-casting capabilities. Additionally CAN also includes an acceptance filtering mechanism to determine if the received message on each local reception buffer are relevant to the specific node or not.

The ability to resolve collisions deterministically when more than one CAN station initiate data transmission simultaneously is one of the protocol's main characteristics. This is accomplished through the arbitration mechanism, ensuring that only the station with the highest priority frame will transmit it's data to the bus (Figure 3.12). This process is serial, meaning that the frame's identifier is transmitted bit-per-bit. The level of the bus will be dominant if at least one CAN station is transmitting a dominant bit (binary 0). If a station is transmitting a recessive bit (binary 1) and senses the bus at dominant level, it will immediately halt, since it will understand that it lost the contention. It will only retry whenever the current frame transmission ends and accordingly senses the bus idle again. An example of the arbitration mechanism is illustrated in Figure 3.12 where two CAN stations attempt a transmission on the bus simultaneously, nevertheless Station 1 senses the bus at recessive level and switches to receiving mode. As Station 2 succeeds, it will continue to transmit its frame.

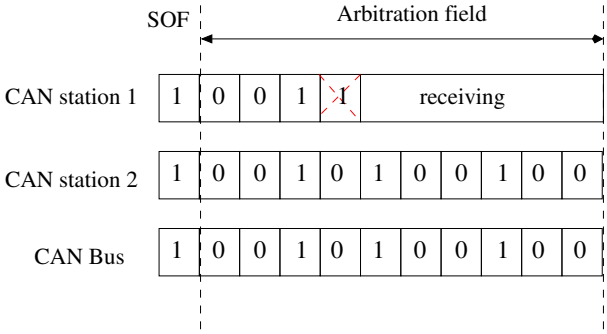

Figure 3.12.: CAN arbitration mechanism

The identifier used in the CAN arbitration mechanism is part of the arbitration field of CAN standard frame (Figure 3.13). Additionally, a standard frame CAN allows data transmission of up to 8 bytes.

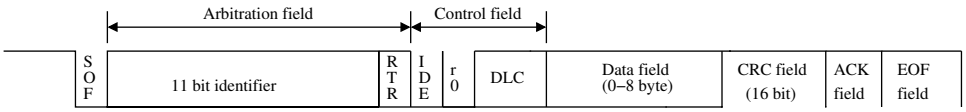

Figure 3.13.: CAN standard frame

CAN FD

The growing use of in-vehicle networks and the rising data load are increasing the complexity of CAN systems nowadays. One of the main reasons behind this is the low bandwidth (1 Mbit/s) and the limitation in the network length as well as the bus length (e.g., up to 1km with 10kbs bandwidth and up to 40 meters at 1Mbps bandwidth). CAN FD [19] was introduced in order to ameliorate the former limitation. Nonetheless, the bandwidth is only increased during the data transmission period. Depending on the CAN Controller capabilities, transmission speed of CAN FD is much higher compared to 1Mbit/s in normal CAN high-speed bus. Additionally, the CAN FD data field length is up to 64 bytes long compared to 8 bytes in the normal CAN (Figure 3.14).

FlexRay

To provide communication for safety-critical x-by-wire systems, the FlexRay consortium developed the FlexRay protocol [46], offering higher rates (up to 10 Mbps) as well as bigger frames (up to 254 bytes). It is mostly found in advanced power train systems, chassis electronics or X-by-wire functionalities (e.g., steering-by-wire) where communication requirements differ, especially concerning the latency of messages or their periodic/sporadic nature.

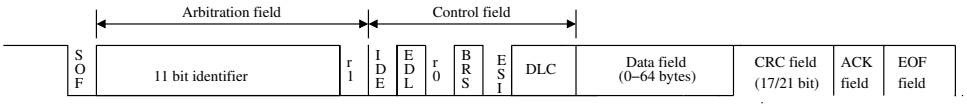

Figure 3.14.: CAN FD standard frame

FlexRay has a very flexible architecture with regard to topology and transmission support redundancy since it can be configured as a bus, a star or a multistar topology. It allows both asynchronous transfer mode and real-time data transfer, and operates as a dual-channel system (i.e. to achieve redundancy), where each channel delivers the maximum data bit rate (10 Mbps). As with CAN, it operates on the data-link layer and physical layer. FlexRay is based on a time-division multiple access (TDMA) scheme and organized in communication cycles, which are periodically executed from the startup of the network until its shutdown. One communication cycle is subdivided into time slots, in which the information transfer from one sender to one or more receivers takes place. The slots are allocated at design time. The schedule of a FlexRay cluster determines in which time slots the FlexRay nodes are allowed to send their frames. FlexRay uses two types of segments to transmit data, called respectively *static* and *dynamic segment*.

The static segment has slots of equal length, where each one (per channel) is exclusively owned by a specific ECU's communication controller for transmission of a frame. In a communication cycle, each node has exclusive access to transmit the message to the receiver in an assigned time (time-triggered communication). Figure 3.15 shows an example of FlexRay's static segment with four nodes, where the ones offering critical functionalities are connected to both channels (e.g. Anti-lock braking system i.e. ABS, Advanced driver-assistance systems i.e. ADAS control) to ensure redundancy and others with non-critical functionalities are only connected to one of them. Furthermore, according to the channel they belong to, they are respectively allocated slots to transmit data.

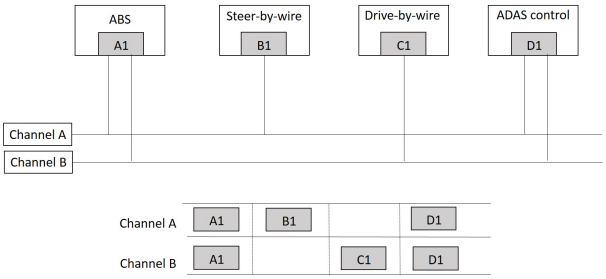

Figure 3.15.: FlexRay Static Segment

The dynamic segment of a communication cycle, a more flexible media access control method, uses the so-called flexible TDMA (FTDMA) scheme. This scheme is based on byteflight protocol [BFL] developed by BMW [15], which is a priority and demand driven protocol. Additionally, the dynamic segment is used to achieve event-driven (e.g. asynchronous) communication as well as allows the nodes to transmit data with a variable payload length, sporadic frames, or frames with a period higher than the communication cycle length. Furthermore, the dynamic segment is subdivided into minislots, which have only a short duration and allow to distinguish between frames in the dynamic segment. Similar to the static slots, these minislots can be assigned to frames, but the transmission of a frame will only be started if at least one node controller has data to send. Otherwise, the dynamic segment will remain unused. Figure 3.16 provides an example frame sequence that is transmitted over both channels. As depicted by the figure the minislot counter in each channel advances only when there is no frame to be transmitted or a frame transmission has been finished. This also means that the minislot number in the two FlexRay channels during the dynamic segment is not the same in contrast to the static segment.

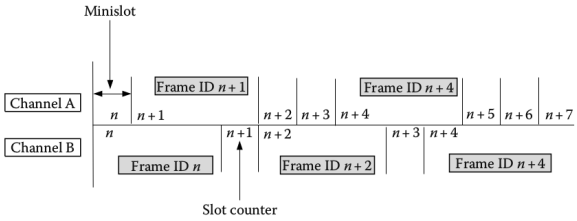

Figure 3.16.: FlexRay Dynamic Segment

MOST

To support end-user applications like radio, global positioning system (GPS) navigation, video displays and entertainment systems, the MOST consortium (involving carmakers and component suppliers) has defined the Media Oriented System Transport (MOST) protocol [106] in 1998. MOST uses three different bandwidths for data transmission: 25, 50 and 150 Mbit/s. As a multi-point data flow system (the streaming data has a source and any desired number of sinks), all devices share a common system clock pulse derived from the data stream (Figure 3.17). In a vehicle the system clock is usually located in the head unit of the infotainment system, which is therefore referred to as TimingMaster. All other nodes are synchronized onto this system clock pulse by means of a Phase-Locked-Loop (PLL) connection (see Figure 3.17) and are thus referred to as TimingSlaves.

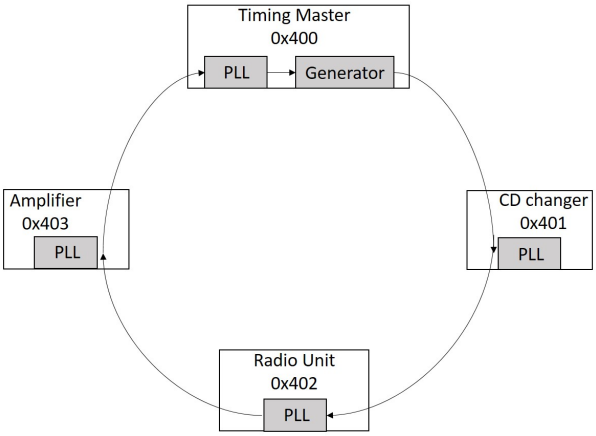

Figure 3.17.: MOST ring

The third revision of MOST [2] has introduced the support of a channel that can transport standard Ethernet frames and is thus well suited to transmit IP traffic. Ongoing and future in-vehicle architectures proceed gradually in the adoption of the automotive Ethernet [57]. The reason behind this adoption is the need for low-cost and mature technology that offers much more bandwidth than what is available today, which is of interest for infotainment and active safety. Additionally the use of Ethernet in the communication architecture allows the deployed switches to handle ports having different speeds.

Automotive Ethernet

Initially in 2008 the first version using 100BASE-TX in the physical layer was provided for diagnostics (using ISO 13400 standard [81]) and code upload. The next and currently used version (from 2015 onwards) supports infotainment and camera-based ADAS using Ethernet Audio Video Bridging (AVB) [68] and BroadR-Reach physical layer [58]. The third generation of Ethernet (under-development) from 2020 onwards, gigabit Ethernet, should become the backbone interconnecting most other networks, thus replacing today's gateways. Figure 3.18 provides an insight on the interactions between

[2] http://www.mostcooperation.com/publications/specifications-organizational-procedures/

the head unit (i.e. infotainment ECU) and the display or the speaker units through the BroadR-Reach BCM89200 switch [3].

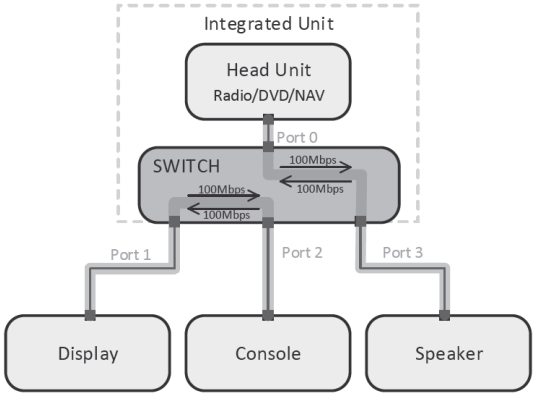

Figure 3.18.: Automotive Ethernet

In-Vehicle protocols and APIs

Introduction

This chapter shows protocols and APIs, which are in use in the automotive business. For a short overview only most common APIs and protocols are mentioned here.

CCP

The CAN Calibration Protocol (CCP) is a measurement protocol defined by the ASAM (Association for Standardization of Automation and Measuring Systems). In general the CCP is used for networking with the CAN bus. The CCP was developed in the 1990s as a manufacturer-independent standard for the parameterization of ECUs. At that time, CAN was the only dominant automotive networking system. The speciality is that the CCP supports flexible access to memory contents.
With the further development of automotive electronics, additional bus systems such as LIN, MOST and FlexRay have become established in ECU networking. The limitation to the CAN bus pushed CCP to its limits, which led to the development of a newer protocol. In contrast, the Universal Measurement and Calibration Protocol (XCP) is traded as a follow-up protocol, which can be employed as well on field buses as the CAN bus [24].

XCP

XCP stands for "Universal Measurement and Calibration Protocol" and is a networking protocol developed by ASAM The primary goal of XCP is connecting calibration systems to electronic control units (ECUs).
It supports different variety of layers, for example CAN, Ethernet and USB. Therefore it is independent from which kind of network is used. It is based on the single-master and the multi-slave concept. The measurement and calibration system becomes the XCP master, while the ECUs become the slaves. For the configuration there is an ECU description file (A2L format) for each slave. Furthermore the XCP master (for example CANape) is able to communicate with different XCP slaves simultaneously [150].

ASAM MCD MC

In the computer science ASAM MCD MC can be denoted as a middleware. Thus, this describes its main task. The application areas of this standard are test stand automation, automated calibration

[3] https://www.broadcom.com/products/ethernet-connectivity/switch-fabric/bcm89200

and data logging. Typically calibration tools use a MC-server to have a unified access to the data of an ECU. The ASAM MCD MC set up on the MC-servers. It specifies the functions of an MC-server. Furthermore, the MCD provides an API for the client, which is object oriented. The communication between the MCD and the ECUs is based on CPP or XCP via CAN. The picture below shows how the MCD sets up on an ECU.
From ASAM MCD's point of view there is the MCD-Device (extern computer), which acts as a master. On the other side the ECU acts as a slave. During the development of an ECU there are more buses that can be used than the CAN. XCP can be used also with the Ethernet, USB or Firewire. Even though the MCD uses these buses as a "tunnel", it orients its payload on CAN (8 bytes). The idea behind the using of different buses during the development relies on reducing the bus load [5].

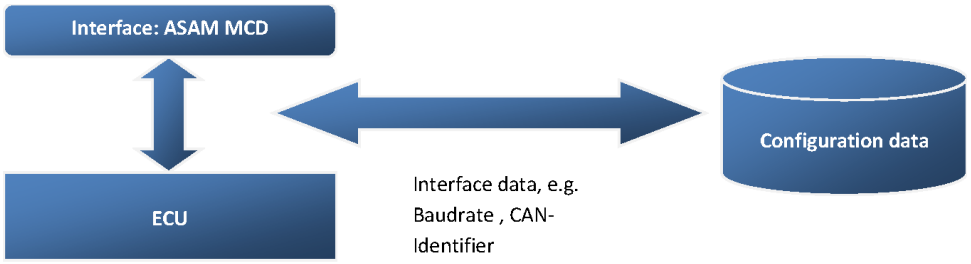

Figure 3.19.: Interface to the ECUs (for MC-Application) [5]

KWP2000

The KWP stands for "Key Word Protocol 2000". This protocol is used for on-board vehicle diagnostics. This is used by the OBD. Basically the KWP2000 is used with the K-Line, but nowadays also with the CAN. It is not depending on the used bus, because it acts on the application layer of the OSI model. The use cases for KWP are (amongst other things) flashing, parameterization of ECUs or read fault memory [156].

UDS

The UDS (Diagnostic Communication Protocol) is an international standard, which is derived from KWP2000. Basically UDS contacts every ECU in a vehicle, which provides an UDS service. It acts on the fifth and seventh layers of the OSI model. Therefore UDS is independent from used bus. Furthermore this protocol enables off-board diagnostic. For this purpose an extern computer (tester) connects with the bus system of a vehicle. For example the use cases are to read fault memory or to update the firmware via UDS,[157].

3.4. Ex-vehicle Connectivity

The technology evolution in automotive vehicles as well as public transportation let to a demand for smarter mobility solutions. These solutions are focused on several types of V2X communication:

- Vehicle-2-Vehicle (V2V)
- Vehicle-2-Infrastructure/Infrastructure-2-Vehicle (V2I,I2V)
- Vehicle-2-Pedestrian (V2P) / Pedestrian-2-Vehicle (P2V)
- Vehicle-2-Network (V2N) / Network-2-Vehicle (N2V), 5) Infrastructure-2-Network (I2N) / Network-2-Infrastructure (N2I).

These types along with their interactions are demonstrated in Figure 3.20.

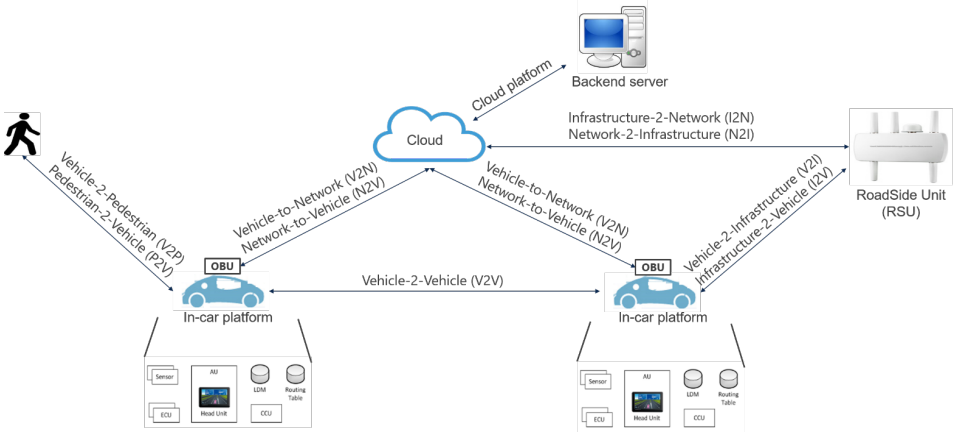

Figure 3.20.: V2X communication types

The units supporting V2X communication are:

- *RoadSide unit (RSU)*: It is connected to road sensors (e.g. induction loops, cameras) and a local control center, such that it performs actions or exchanges critical information other vehicles or servers about road or traffic management.
- *OnBoard unit (OBU)*: The on-board unit (OBU) is a radio built-in vehicle device mounted on each vehicle that transmits vehicle data (i.e. identification and location) to a transponder. The OBU itself is a transponder, that is, a data exchange takes place automatically and only on request of one of the participating devices. It allows Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I, I2V) communications with other OBUs or RSUs.
- *Backend server*: It is composed by a PKI, traffic management and roadside unit management servers, all accessible via the RSU's or cellular base stations.

In order to facilitate this evolution a couple of solutions were defined that are split into 3 main categories:

- *5G radio access technologies*: This technology provides wide area, broadband access. The 5G technology is currently in the process of conceptual development and standardization by the World Radiocommunication Conference (WRC). The 5G technology is expected to have a specific V2X aspect of the 5G technology in a practical scale after 2020. However, in this document we are leveraging the limited standardization to illustrate conceptually its main scope and architectural view.
- *Pre-5G radio access technologies*: Multiple cellular technologies were identified by the ETSI 3rd Generation Partnership Project (3GPP), *LoRa* Alliance and other organizations, such as Narrowband IoT [125] , Long Term Evolution for Machines (LTE-M) [124], LoRA are [94] considered. Even though these technologies are already used in V2P/P2V, the main challenge when adopting them in other V2X communication types are reliability and safety, which are currently not addressed in the scope of Low-Power Wide Area Networks (LPWAN). A thorough view on these technologies as well as their standardization process towards an LTE-V2X concept

is provided in deliverable D2.1

- *Non-cellular technologies providing wireless access*: IEEE has defined different standards for wireless communication, such as 802.11ac and 802.11p, however only 802.11p is flexible in terms of throughput and offers higher reliability, even though its maximal throughout is more limited than 802.11ac (from 3 to 27 Mbps raw data rate). The reason behind this is that 802.11p was designed particularly for for safety-related Vehicular Ad-hoc NETworks (VANET), including the V2V and V2I/I2V concepts. IEEE 802.11p technology is currently fully specified and already deployed in different locations.

The following paragraphs start with a description of the scenarios supported by 802.11p communication and cellular communication. This is followed by a description on both the 802.11p and 5G technologies. In the scope of this section we focus on these two technologies, because, to the best of our knowledge, they are considered as the leading candidates for V2X communication. However, further information about pre-5G radio access technologies is provided in deliverable D2.1.

3.4.1. 802.11p / cellular communication scenarios

This paragraph describes the various communication scenarios that are executed through either an 802.11p or a cellular communication channel. These scenarios depend on the type of message that is communicated. The following sub-paragraphs provide a description per message type.

ETSI safety message communication though 802.11p

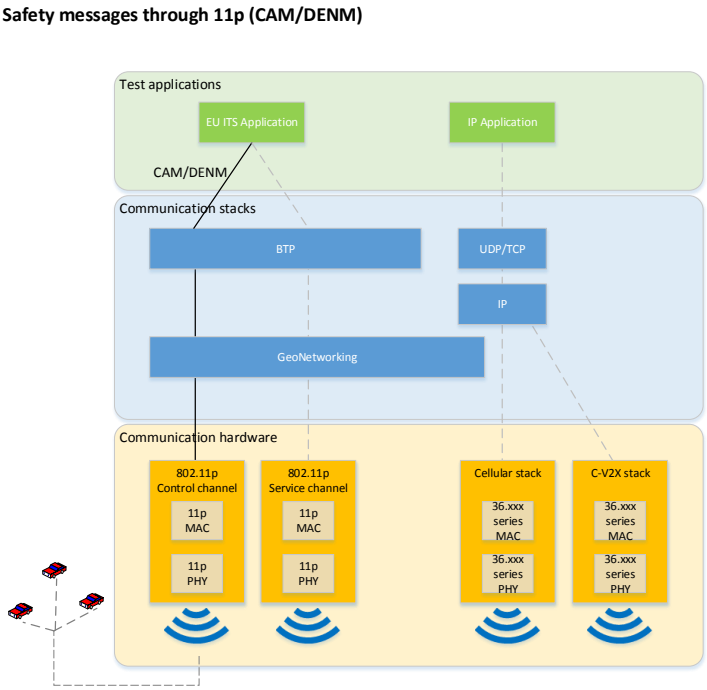

Figure 3.21.: ETSI safety message communication through 802.11p

The set of ETSI safety messages consists of CAM and DENM messages. These messages are generated by an ITS application targeting the use of the European standard. This application sends the CAM and DENM messages to the layer with communication stacks. The sent messages already include the Basic Transport Protocol (BTP) header. The communication stack completes the message by adding the GeoNetworking header. This header makes sure the message is delivered to the correct geographical region.
After the message is created completely it is sent to the 802.11p Control Channel. This channel broadcasts the message to all the vehicles in the vicinity.

ETSI service message communication through 802.11p

The set of ETSI service messages consists of SPAT, MAP and IVI messages. These messages are generated by an ITS application targeting the use of the European standard. This application sends the SPAT, MAP and IVI messages to the layer with communication stacks. The sent messages already include the Basic Transport Protocol (BTP) header. The communication stack completes the message by adding the GeoNetworking header. This header makes sure the message is delivered to the correct geographical region.
After the message is created completely it is sent to the 802.11p Service Channel. This channel broadcasts the message to all the vehicles in the vicinity.

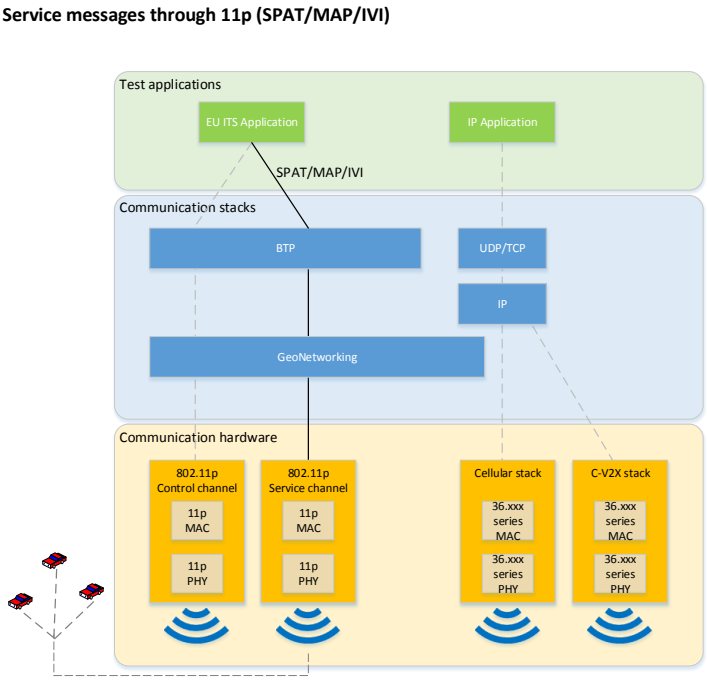

Figure 3.22.: ETSI service message communication through 802.11p

IP message communication through 802.11p

IP messages are either TCP or UDP messages. These messages are generated by an IP application. This application sends the TCP and UDP messages to the layer with communication stacks, where the GeoNetworking header is added. This header makes sure the message is delivered to the correct geographical region.
After the message is created completely it is sent to the 802.11p Service Channel. This channel broadcasts the message to all the vehicles in the vicinity.

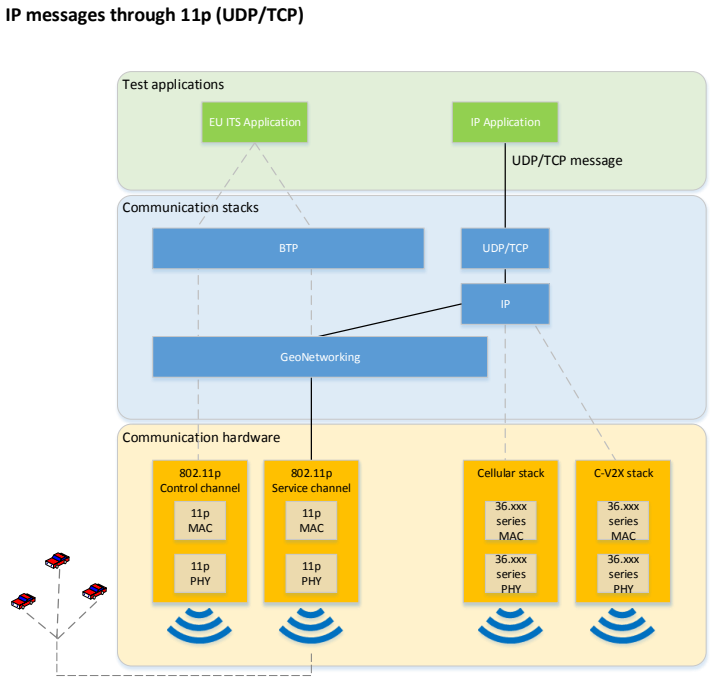

Figure 3.23.: IP message communication through 802.11p

Message communication through cellular

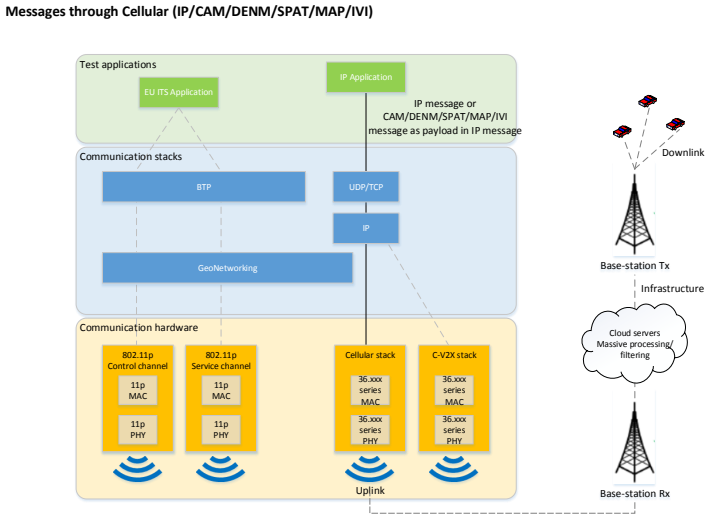

Figure 3.24.: Message communication through cellular

Messages to be sent through the cellular network are IP messages. The message payload can be an ETSI message (CAM /DENM/SPAT/MAP/IVI) or a custom payload. These messages are generated by an IP application. This application sends the IP messages to the layer with communication stacks, where the IP header is added. This header makes sure the message is delivered to the correct cloud service.
After the message is created completely it is sent to the Cellular stack. This stack sends the message through the Uplink to a nearby base-station. This base-station is connected to the internet. The message is sent through the internet to reach the destination could service.
In case the message payload is an ETSI message the ETSI information must be sent to the other vehicles near the sending vehicle. The cloud service determines the set of vehicles that must receive

the ETSI information and sends an IP message containing the ETSI information to each vehicle.

Message communication through Cellular-V2X Mode 3

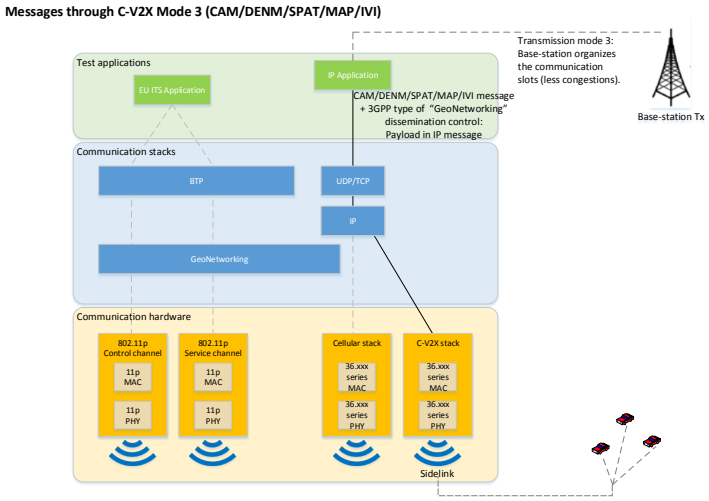

Figure 3.25.: Message communication through Cellular-V2X Mode 3

Messages to be sent through cellular-V2X are IP messages containing an ETSI message (CAM/-DENM/SPAT/MAP/IVI) payload. These messages are generated by an IP application. This application sends the IP messages to the layer with communication stacks, where the IP header is added. A 3GPP type of GeonNetworking dissemination control makes sure the message is delivered to the vehicles in the vicinity.
After the message is created completely it is sent to the Cellular-V2X stack. This stack sends the message through the Sidelink directly to the cars in the vicinity.
The Cellular-V2X stacks runs in Mode 3. In this mode, all communication is performed in a synchronous manner. Timing information is provided by the network to determine the timeslots in which transmission takes place. So, although there is direct communication between vehicles, the connection to the network is still needed to enable communication.

Message communication through Cellular-V2X Mode 4

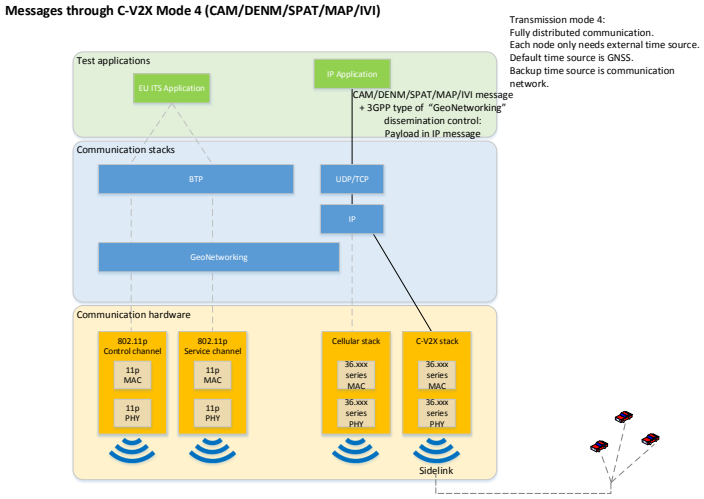

Figure 3.26.: Message communication through Cellular-V2X Mode 4

Messages to be sent through cellular-V2X are IP messages containing an ETSI message (CAM/-DENM/SPAT/MAP/IVI) payload. These messages are generated by an IP application. This application sends the IP messages to the layer with communication stacks, where the IP header is added. A 3GPP type of GeoNetworking dissemination control makes sure the message is delivered to the vehicles in the vicinity.
After the message is created completely it is sent to the Cellular-V2X stack. This stack sends the message through the Sidelink directly to the cars in the vicinity.
The Cellular-V2X stacks runs in Mode 4. In this mode, all communication is performed in an ad-hoc manner. As a result, no network connection is required and all communication is directly between vehicles.

3.4.2. 802.11p

Hardware architecture

The hardware of the 802.11p module can be divided into several functional blocks. These blocks are listed in Figure 3.27. Interfaces between these blocks are indicated using dashed vertical lines. The horizontal solid line indicates there are two different antenna/modem configurations:

- Combination of one antenna (roof antenna) and a modem.
- Combination of two antennas (roof and mirror antenna) and a modem.

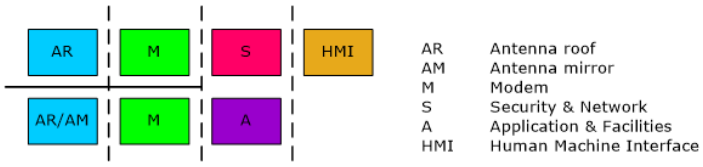

Figure 3.27.: 802.11p functional blocks

Interfacing to the Appstacle in-vehicle platform can be done using USB or Ethernet, where Ethernet is preferred.

Software architecture

A wide variety of ETSI standards are related to the 802.11p communication. Figure 3.28 presents an overview of these standards.

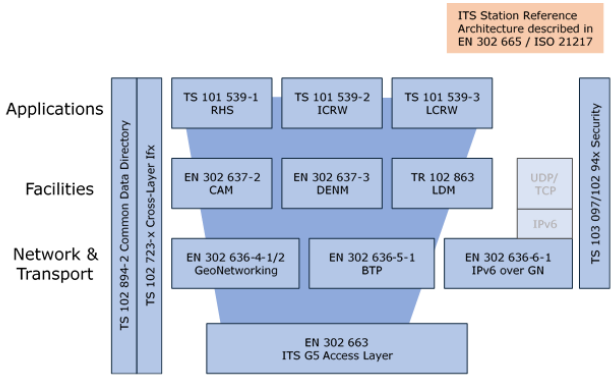

Figure 3.28.: Overview ETSI standards

The list below contains short descriptions on the most important standards:

- EN 302 663 ITS G5 Access Layer: This standard describes how the network is accessed. It is basically a mapping of 802.11p incorporating European specific changes.
- EN 302 636-4-1/2 GeoNetworking: This standard describes geographically bound communication, for example used to exchange information on road conditions.
- EN 302 636-5-1 BTP (Basic Transport Protocol): This standard describes the format of the low-level packets which are used as containers for the higher-level messages.
- EN 302 637-2 CAM (Cooperative Awareness Messages): This standard describes the format of the messages that are used to publish vehicle information. These messages contain current position, speed, direction etc. They are sent at a frequency from 2Hz to 10Hz, using a single-hop broadcast.
- EN 302 637-3 DENM (Distributed Environmental Notification Message): This standard describes the format of the messages that are used in case of special occasions, for example accidents and road condition warnings. These messages are sent based on events, using multi-hop or GeoNetworking broadcast.
- TR 102 863 LDM (Local Dynamic Map): This technical report describes the conceptual data store that is located within an ITS station. It contains information that is relevant to the safe and successful operation of ITS applications. Data can be received from a range of sources such as vehicles, infrastructure units, traffic centers and on-board sensors. The data range from very static to highly dynamic data (road topography, static speed limit, signs and signals, road works, temporary speed limits, current information on vehicles or infrastructure nearby).
- TS 101 539-1 RHS (Road Hazard Signaling): Road Hazard Signaling is one of the applications within the defined Basic Set of Applications (BSA). This document describes the application requirements of the application. Goal of the application is to present a warning between 6 and 30 seconds before actual collision.
- TS 101 539-2 ICRW (Intersection Collision Risk Warning): Intersection Collision Risk Warning is one of the applications within the defined Basic Set of Applications (BSA). This document describes the application requirements of the application. Goal of the application is to present a warning between 2 and 6 seconds before actual collision.
- TS 101 539-3 LCRW (Longitudinal Collision Risk Warning): Longitudinal Collision Risk Warning is one of the applications within the defined Basic Set of Applications (BSA). This document describes the application requirements of the application. Goal of the application is to present a warning between 2 and 6 seconds before actual collision.

Through the NXP/Cohda Wireless MK5 module an implementation of most standards is provided. No implementation is provided for the applications as well as the LDM (Local Dynamic Map) since these are customer specific.

Software deployment

The deployment of the various parts of the software onto the hardware block is described in Figure 3.29.

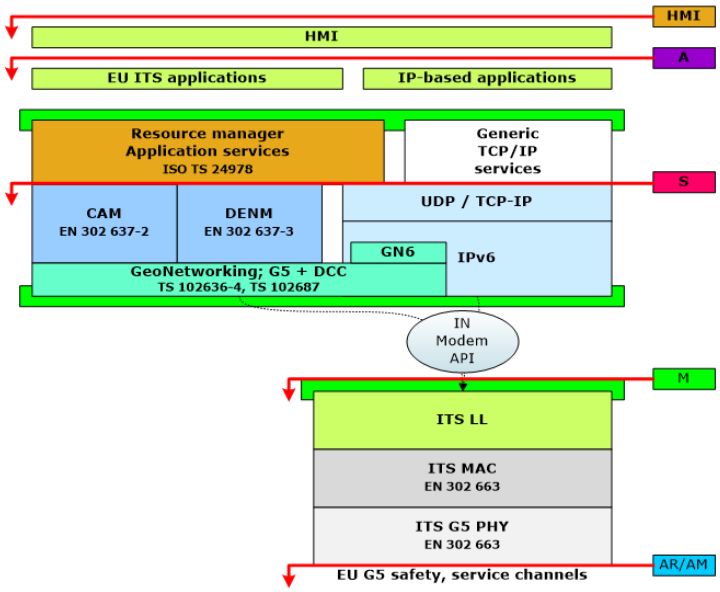

Figure 3.29.: Deployment of software on the 11p hardware

802.11p communication channel

All 802.11p communication evolves around the 5.9 GHz frequency. Several channels are defined, where each channel is allocated to a specific type of ITS applications. The types are:

1. ITS-G5A: ITS road traffic safety applications.
2. ITS-G5B: ITS non-safety road traffic applications.
3. ITS-G5D: Future road traffic applications.

An overview of the type of channels allocate in the 5 GHz frequency range is presented in Figure 3.30. Details on the allocated channels types are presented in Figure 3.31. The maximum limit of the mean spectral power density per channel is presented in Figure 3.32. In this figure, the name of each channel reflects the intended use: A CCH channel is a control channel used to communicate traffic control messages. A SCH channel is a service channel used to communicate traffic service messages. The service channels are likely to be used to communicate the IP messages. All the channels listed in Figure 3.32 will use a default data of 6 Mbit/sec, except for SCH2 which has a default data rate of 12 Mbit/sec. Figure 3.30 to Figure 3.32 are present in the ETSI standard: "Draft ETSI EN 302 663 V1.2.0 (2012-11)".

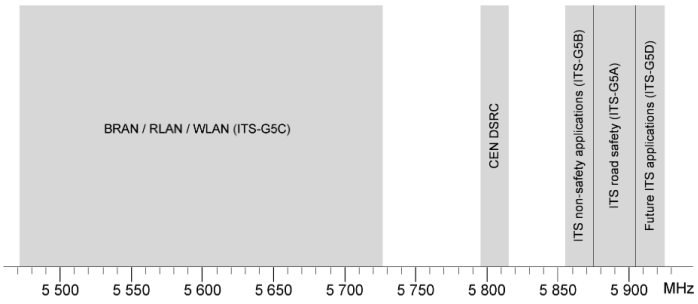

Figure 3.30.: Overview of channel types allocated for the 5 GHz frequency range

	Frequency range [MHz]	Usage	Regulation	Harmonized standard
ITS-G5D	5 905 to 5 925	Future ITS applications	ECC Decision [i.4]	EN 302 571 [1]
ITS-G5A	5 875 to 5 905	ITS road safety related applications	Commission Decision [i.7]	EN 302 571 [1]
ITS-G5B	5 855 to 5 875	ITS non-safety applications	ECC Recommendation [i.2]	EN 302 571 [1]
ITS-G5C	5 470 to 5 725	RLAN (BRAN, WLAN)	ERC Decision [i.3] Commission Decisions [i.5] and [i.6]	EN 301 893 [i.14]

Figure 3.31.: Details of channel types allocated for the 5 GHz frequency range

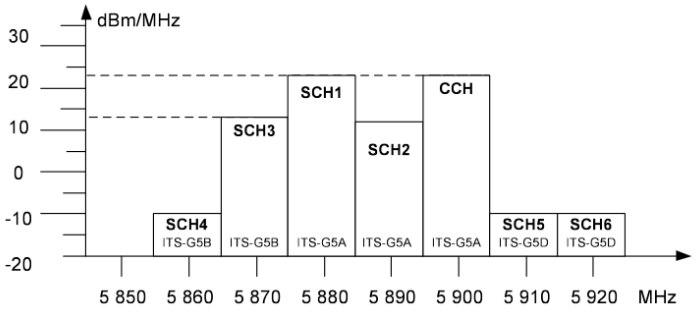

Figure 3.32.: Maximum mean spectral power per channel

3.4.3. 5G

5G is the next generation of mobile communication technology. It is expected to be defined by the end of this decade and to be widely deployed in the early years of the next decade. As opposed to earlier 3G and 4G technologies, 3GPP conceptualized 5G to be more than another mobile broadband connectivity, covering a variety of use-cases and industries.

5G was initially based on the conceptual composition as well as evolution of cellular technologies. This is because the different technologies have communication requirements, that are focused in a local (LAN) or wide area network (WAN) communication.

In particular, IEEE's 802.11p has been developed to support different types of wireless communications (e.g. PAN, LAN), but since it is based on CSMA/CA its performance degrades quickly as network load increases. This happens because a high number of transmitting stations will increase the number of collisions on the communication medium. Additionally, since it is was designed for short-range transmissions (transmission range up to 1km), many vendors introduce a multi-hop functionality to increase the transmission range. An example of multi-hop functionality is introduced in the European Cooperative Intelligent Transport System (C-ITS) protocol stack [44] named as GeoNetworking. Furthermore, another technology that could not support as standalone broadband (e.g. WAN) communications is LTE [133]. The main drawback of this technology is that every transmitted packet must traverse the infrastructure, meaning that each infrastructure failure will have a strong impact on connectivity of the entire network. Furthermore, even though broadband connections can be supported in LTE, scenarios where the infrastructure is not available due to out-of-coverage are also quite probable. Finally, since LTE was designed to use radio resources in order to allow broadband communication, its extension in V2X connectivity where smaller data packets and higher bandwidth are required is suboptimal in terms of consumed resources still remains a great challenge.

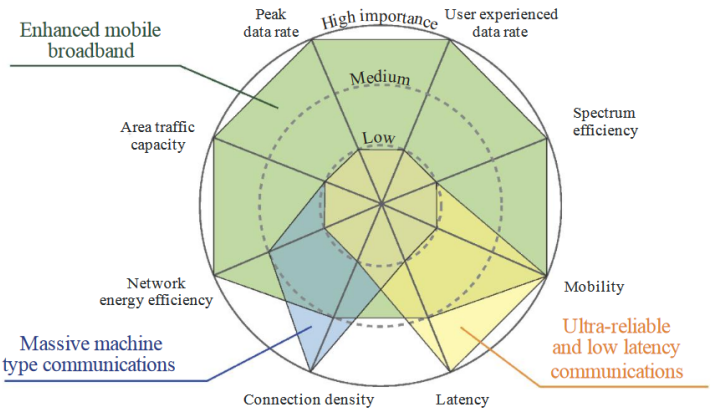

Figure 3.33.: 5G use case categories [74]

For these reasons, the radio communications research community investigates currently on several scenarios to enhance the characteristics cellular technologies by leveraging the 5G standardization, in order to achieve five main objectives: 1) lower power consumption in the individual devices, 2) smaller end-to-end latency in communication, 3) better performance than wireless technologies [123] 4) in and out-of coverage communication support and 5) security-by-design in the system. Specifically, a key feature of 5G is that it can use more frequency bands than 3G and 4G, and has a focus on low latency so that messages can be sent to any other station on the network quickly. Figure 3.33 introduces the 5G technology aspects according to the World Radiocommunication Conference (WCC). 5G is foreseen to integrate a mix of Radio Access Technologies (RATs) enabling a combination and cooperation of various RATs, some already existing (e.g., Narrowband IoT, LoRa, LTE-M), others to be designed (e.g., future releases of LTE).

Even though cellular V2X communication is quite promising in terms of spectrum and energy

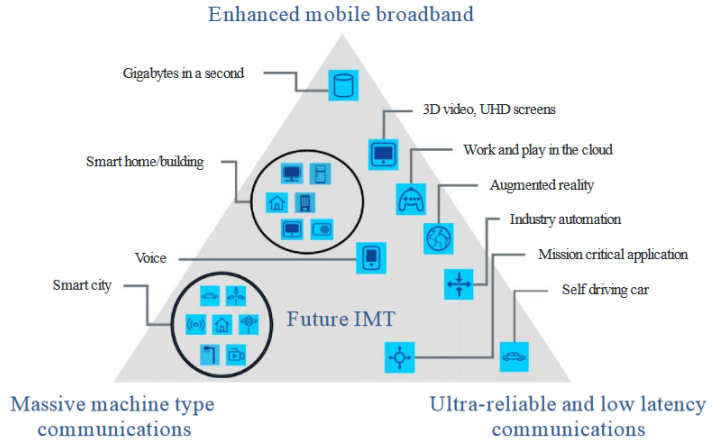

Figure 3.34.: Key capabilities per use case category [74]

efficiency as well as latency and mobility, the current state of cellular transmissions is still on the process of V2X deployment in release 14. This is because, only release 13 is being deployed and tested currently Figure 3.35). Therefore, as the integration in V2X context is not tested yet, a first challenge lies in respecting the technological requirements that we set. On top of that, another challenge is derived from the 3GPP specification release 15 on 5G (scheduled for September 2018) itself. More specifically, this release is divided into two parts: the 5G part and the second LTE-V2X part, which incorporates improvements on release 14 LTE-V2X that are however not backwards compatible with it. This is due to the 5G New Radio technology (NR), which will be introduced by release 15. To this end, 3GPP has set a goal for the rapid completion of a first release of 5G NR, which will be followed as well by the LTE-V2X release or so called V2X phase 3 (or even eV2X). This hampers the release 14 LTE-V2X adoption by automotive manufacturers. Moreover, a third and less critical challenge lies in the impact of mapping the new technology into each ITS stack layer of the ETSI ITS reference architecture.

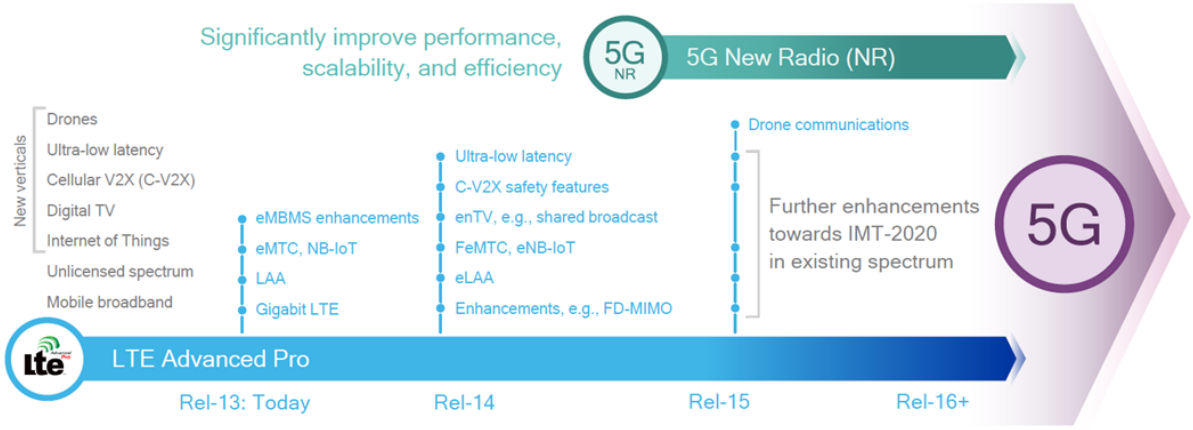

Figure 3.35.: 3GPP roadmap towards 5G

Given all the aforementioned challenges, in the scope of this document we restrain the focus to the standard definition for release 14 LTE-V2X, which has been already released as of March 2017.

Specifically, in its release 14 3GPP denotes that that the new short-range radio interface (PC5) transport support for V2V services is of highest priority. PC5 is based on 3GPP's Release 12 [92] proximity services communications (Proximity Service or simply "ProSe") feature for Device-to-Device (D2D) in/partial and out of coverage communication. The reason behind the high priority is

that it allows the support of V2V communication, without the need of going through a base-station. However, it also allows user-to-user (Uu) network communication modes, where the base-station may assist the communication. Figure 3.36 illustrates all the above modes according to the communication types that were introduced in Figure 3.20. This Figure illustrates the two modes as described in release 14 and distinguished as a) Mode 3 and b) Mode 4, where:

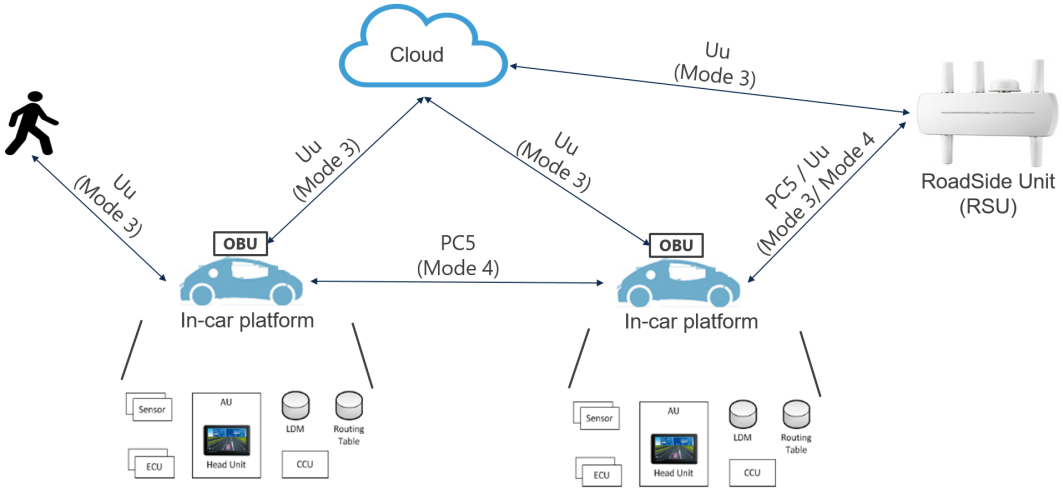

Figure 3.36.: V2X Cellular communication for the communication types of Figure 3.20

1. **Mode 3** is the base-station assisted Uu mode. In this mode each node needs to be connected through the base-station to the network. Although the link to the network is not used to do the actual communication the link is used to determine the communication timing and allocation of communication resources (e.g. which frequencies to use). Mode 3 supports V2P/P2V communication types, covering LTE-based communication between a vehicle and individual devices (e.g. smartphones or tablets), as well as V2I/I2V and V2N/N2V communication types covering LTE-based communication between a vehicle and RSUs / networks. This mode uses the existing LTE Wide Area Network (WAN) and is suitable for more latency-tolerant use cases (e.g., situational awareness, mobility services). Furthermore, Mode 3 uses the LTE operator spectrum.
2. **Mode 4** is the standalone V2V mode through direct PC5 interface between the individual vehicles. In this mode no connection through the base-station to the network is required. This mode is suitable for proximal direct communications (hundreds of meters) and for V2V safety applications that require low latency (e.g. ADAS, situational awareness) and can work both in and out of network coverage. In contrast to Mode 3 which uses the operator spectrum, the PC5 mode may be deployed in a separate spectrum meant for direct communication for V2X applications.

It is also important to note that modes 1 and 2 were initially defined by 3GPP's release 13 and included the introduction of the PC5 communication interface for in and out of coverage respectively. Although the definition of both modes is stable deployment details are not specified yet and are specific to the deployment and testing setup of the individual network providers that will start once the LTE-V2X modules are ready (approximately in the end of 2018). Moreover, deployment and testing setup may have a potential impact on mode 3.

An important area that should be considered both in mode 3 and mode 4 operations is that devices that are subscribed with one operator should still be able to communicate and share information with devices subscribed with another operator. In the case of common spectrum for V2X, this operation is straightforward. However, it is expected that some agreement between operators with regards to spectrum usage will be necessary.

Another aspect of the cellular-based V2X system defined in 3GPP for LTE and 5G is that is meant

to reuse the service and application layers already specified by the automotive community, such as the Society of Automotive Engineers (SAE) International. To this end, LTE V2X is also provisioned to support the SAE J2735 [59] V2X-specific message types. These types define information or safety-related messages (e.g. alerts), such as the basic safety message (BSM), which broadcasts the vehicle state to provide by-lane target classification for enhancing advanced driver assistance systems (ADAS).

Overall, LTE-V2X technology as of 3GPP's release 14 is a pre-mature technology for the adoption in the automotive industry, as LTE was not designed for safety-critical applications. Instead its focus was on for cellular and mobile devices that are not meant to last long (maximum time 3 years), as opposed to the life-cycle of automotive vehicles that is approximately 15 to 30 years and should be robust, reliable and mature. However, a tremendous amount of effort is being made currently by 3GPP to adapt and improve the proposed LTE-V2X concept by December 2018 (release 15 date).

In deliverable D2.1: "SotA Research with regard to Car2X Communication, Cloud and Network Middleware and corresponding Security Concepts" we focus on providing in-depth information on the reference architecture as well as network layers of 5G and pre-5G communication technologies up to 3GPP's release 14.

3.5. Intrusion Detection Systems

This section provides an overview of the existing techniques and methods to monitor the in-vehicle systems and detect misconfigurations and attack/threat scenarios through Intrusion Detection Systems (IDS). To this end, it first presents the scope as well as the role of intrusion detection in the APPSTACLE project. Then, it continues with an analysis of the previous work in in-vehicle intrusion detection systems and finally concludes by discussing the existing gaps and research directions that will be considered in the project.

Opening automotive systems via Car-to-X connectivity and extending them by application runtime environments as planned in APPSTACLE raises new threats with respect to targeted attacks on these systems. Here, security incidents might compromise safety and, thereby, result in enormous financial cost or even danger of death. Thus, automotive security solutions are fundamental to the success of the APPSTACLE platform. Incidents like the prominent Jeep Hack [4] have shown that this aspect is only regarded to a limited extend in the automotive domain today.

In particular, one of the vital security-related challenges to solve in in-vehicle systems is the lack of integrity and authentication, rendering them vulnerable to injection and tampering attacks. Data authentication usually involves heavy cryptographic computations complicated to perform inside the car due to the real-time constraints and the limited ECUs' resources. Nilsson et al. proposed an efficient delayed data authentication using compound Message Authentication Codes (MAC) [112]. Another idea to guarantee authentication is to look for means to assess the legitimacy of ECUs. One approach [54] is to identify these units with certificates and to create trusted communication groups, guaranteeing that only authentic controllers are able to be part of these closed communication groups. In the same fashion, Oguma et al. proposed an attestation based security architecture [115] for in-vehicle systems. A "master ECU" will act as a verification server, distributing cryptographic keys to ECUs and ensuring their trustworthiness. Finally some other efforts have been made towards hardware-based attestation, which can be leveraged by using trusted computer platform containing a special hardware component on its chip, called the "Trusted Platform Module" (TPM). Its role is to "establish a chain of trust through the basic input/output system (BIOS) to the operating system (OS)" [117].

However, the applicability of these approaches to productive automotive systems is still unclear as not all of them integrate easily within the vehicle. Furthermore, regarding the safety-critical nature of automotive systems an additional line of defense is required. Approaches representing such a second line of defense may cope with attacks that bypass security measures like the ones mentioned above.

A prominent security measure to cope with these challenges are automotive *Intrusion Detection Systems (IDS)* [62]. According to the National Institute of Standards and Technology[5] (NIST) *Intrusion Detection* can be defined as "the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents" [130]. In our context incidents relate to any events compromising the confidentiality, integrity, or availability (CIA) of an automotive system or any attempts to bypass its security mechanisms [13]. This monitoring process can be automated by using an intrusion detection system (IDS) which identifies eventual incidents.

Upon detection the system will generate a response, which can be categorized as passive or active response [13]. An active response will try to mitigate the incident by, for example, collecting additional information about the attacks, deterring the intruder by terminating her connection and reconfiguring the other network and security devices (e.g., router and firewall) to block further packets originating from the malicious source IP address. An IDS responding in a passive manner will solely report the offense, by simply notifying the system administrator or the incident response team with, for instance, a pop up on their monitoring screen or an SMS, and leaves the next actions to be taken up to them. The "Intrusion Detection System" generally implies passive response while the term "Intrusion Prevention System" has been coined to differentiate systems supporting active response.

[4] https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
[5] www.nist.gov

Even though there seems to be no clear distinction between these two terms in the literature and on the market, we can use the term IDS to address these systems in a generic fashion. Further in the document we will explicitly mention when an IDS possesses prevention capabilities.

In accordance to [13] and many other publications we distinguish between host-based (HIDS) and network-based IDS (NIDS) approaches. They differ mainly in the data source they monitor and analyze. A HIDS collects information and monitors events occurring within a single system. By analyzing the diverse data gathered such as system logs or file accesses and modifications, it can identify precisely the ongoing activities on that host and determine the user(s) and process(es) involved in an attack on the system [13]. A NIDS monitors network packets for a specific part of the network [130]. Moreover, it protects end-points by analyzing the traffic going to and coming from them. Generally speaking an IDS architecture contains several sensors deployed at strategic points on the network where they monitor and analyze traffic and report attacks to a central management console [13].

Both IDS approaches complement each other in a collaborative manner. For example, many HIDS can cope with encrypted traffic, whereas, this is problematic for NIDS. The underlying reason behind this is that data is decrypted upon reception at the host, which enables the HIDS to work on this decrypted data. At the same time, a HIDS is harder to manage (one IDS per host is required). It lacks the overall context and is blind to network attacks. This is where the NIDS comes in place. A NIDS can detect network-specific attacks and has no impact on the performance of the host [37].

In the following, we elaborate on application-based IDS, which is a subset of HIDS (cf. Section 3.5.1). Application-based IDS focus on the execution of applications and the data they generate. This makes same a promising solution in the context of APPSTACLE, where applications are of particular interest. Thereafter, we refer to network-based IDS and their application in the automotive context (cf. Section 3.5.2).

3.5.1. Application Intrusion Detection Systems

This section introduces the state-of-the-art of application-based Intrusion Detection Systems. General purpose host-based IDS do not consider information of applications that are deployed on the host machine. However, knowledge about these applications can potentially improve the precision and performance of an IDS. Such an improvement might result from choosing the optimal data to monitor, choosing a tailored or adjusted analysis technique, or even adapting the architecture of the IDS to the concrete use case.

Bace and Mell define application-based IDS as a subset of host-based IDS "that analyze the events transpiring within a software application" [14]. We follow this terminology but broaden their definition slightly to cover the full range of host-based IDS that focus on or adapt to application specifics.

Definition 3.5.1.1 *Application-Based Intrusion Detection Systems are host-based Intrusion Detection Systems that monitor and analyze events within applications or whose monitored and analyzed data is relatable to applications or processes, respectively.*

In the following, we explain the scope and style of our state-of-the-art analysis. Thereafter, we give an overview of the reviewed approaches and elaborate on their classification. Finally, we discuss every approach in detail.

Scope

The scope of this state-of-the-art analysis is the discussion and categorization of ten scientific publications. The main reason for excluding commercial or open source products from our analysis is that there are few examples that can be categorized as application-based IDS.

One of these examples is the QNX anomaly detector (qad)[6]. It detects anomalies in the runtime behavior of pre-specified processes. The qad analysis comprises a training phase where the normal behavior of the processes is learned and a detection phase where the productive system is monitored for deviations between the actual runtime behavior and the learned normal behavior. Such deviations can indicate unauthorized accesses to the system and are logged for further investigations by an administrator.

Unfortunately, qad is also a good example for another problem we encountered when trying to evaluate corresponding commercial or open source projects. There is typically none or very few and scattered information about the employed (analysis) techniques. Therefore, we restrict our elaborations to scientific publications and the classification of these as explained in the next section.

Overview

In order to cover a broad range of publications on the topic of application-based intrusion detection and at the same time limit the number of publication to skim, we restrict the search to the following publishers: ACM, IEEE, Elsevier, and Springer. To ensure current approaches in our state of the art analysis, we restrict ourselves to publications published from 2012 onwards. We aim to examine ten publications in this analysis and choose the publications from our search results to show the variation among the approaches.

The search results are acquired by using the publishers' respective digital libraries. We base the search phrases on the terms *intrusion detection systems*, *host-based*, and *application behavior*. After expanding the search terms by using synonyms and similar meanings, we arrive at the following basic search phrase:

("Intrusion Detection" OR "Anomaly Detection" OR "Application Intrusion Detection") AND ("Host-Based" OR "Application Based" OR "Application Specific") AND ("Program Behavior" OR "Application Behavior" OR "Software Behavior").

[6] http://www.qnx.com/developers/docs/7.0.0/index.html#com.qnx.doc.neutrino.utilities/topic/q/qad.html

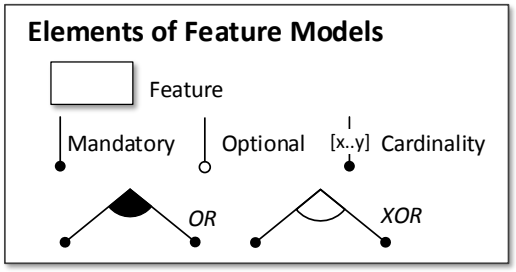

Figure 3.37.: Elements of feature models that we use for our taxonomy

Because not all digital libraries support boolean operators out of the box, the search phrase is adapted for each search platform. Ten publications are selected from the search terms for review. Note that we do not limit the reviews to publications focusing on the automotive domain because not many automotive intrusion detection approaches could be found during our initial searches. The constraints that the automotive context imposes on an IDS is to be addressed at later stages during the project.

For the classification of the reviewed approaches we developed the taxonomy depicted in Figures 3.38 to 3.44 in the form of a feature model. Kang et al. introduced feature models in 1990 to facilitate the systematic discovery and documentation of domains of related software systems [84]. The formalism allows to visually model features and their relations in tree-like structures. Generally, features are characteristics of related approaches. Due to this generality, feature models can not only be used to analyze software systems but for all kinds of domains. In the context of our state-of-the-art analysis they bring all semantic elements needed for the formalization of our taxonomy (cf. Figure 3.37). These are (i) *features*, (ii) *mandatory features*, (iii) *optional features*, (iv) *cardinalities*, (v) *OR feature groups*, and (vi) *XOR feature groups*.

(i) We use *features* to express characteristics of approaches in the domain of application-based IDS. Features are hierarchically structured. Thus, a feature can comprise several features and feature groups.

(ii) *Mandatory features* are features that are present in all approaches.

(iii) If we found features that are not common across all reviewed approaches we mark them as being *optional*.

(iv) We use cardinalities [31] to express whether an approache realizes a feature more than once. Here, `x` is the lower bound of instances of the corresponding feature and `y` its upper bound.

(v) An *OR feature group* expresses that at least one feature of the group is present in each approach.

(vi) An *XOR feature group* expresses that exactly one feature of the group is present in each approach.

We inferred a basic version of our taxonomy from the taxonomies used in [131], [90] and [96]. Both publications do not focus on application-based IDS, e.g., they also refer to network-based approaches. Therefore, we tailored our basic version of the taxonomy to the domain of application-based IDS. Furthermore, we extended it continuously during the reviewing process. Meaning, if we found a new characteristic while reviewing a publication (e.g. a new analysis technique), we added it to the taxonomy.

The feature diagram is split across Figures 3.38 to 3.44 to increase its readability. Figure 3.38 depicts the main distinctive features of an application-based IDS that guide through the following figures. These are:

- the kind of general `Approach` its follows (cf. Figure 3.39),
- the `Context` for which it is developed and evaluated on (cf. Figure 3.40),
- its `Architecture` (cf. Figure 3.41),
- the data it monitors and analyzes (`Monitored Data`, cf. Figure 3.42),

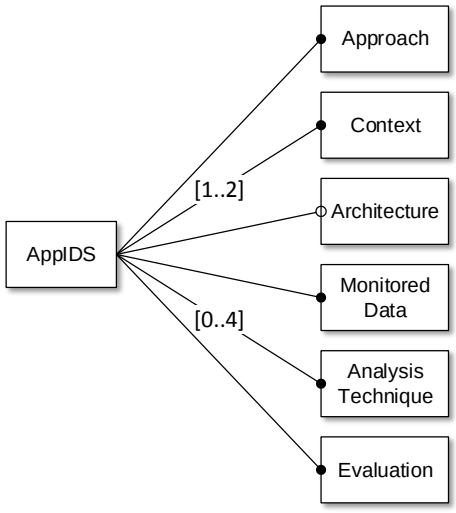

Figure 3.38.: Feature model showing the first hierarchy of our taxonomy

- the `Analysis Techniques` it employs (cf. Figure 3.43),
- and the kind of `Evaluation` the authors used (cf. Figure 3.44).

Please note that not all publications give an architectural description of their approach. Thus, the `Architecture` feature is marked as optional. Furthermore, some publications combine different analysis techniques. Thus, the corresponding feature has the cardinality `[0..4]`. Similarly, the `Context` feature has a cardinality of `[1..2]`, which enables to distinguish different contexts. In the following, we describe the feature model in more detail.

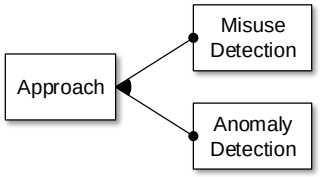

Figure 3.39.: `Approach` sub-tree of feature diagram

Main Feature: Approach (cf. Figure 3.39) Following the NIST classification of IDS [14], there are two basic approaches: `Misuse Detection` and `Anomaly Detection`. `Misuse Detection` (also known as signature based approach) refers to the detection of intrusions by comparing system information with fixed, defined patterns of known security breaches, e.g., through malware or attacks. Contrary to defining signature for intrusion recognition, `Anomaly Detection` based approaches compare a representation of normal system behavior with the behavior of the running system. According to the NIST, the IDS learns the normal system behavior before being deployed on a system. This learning phase can be realized through a machine learning approach or another method of data gathering. We interpret any approach as anomaly detection in which normal behavior is compared with current system behavior (as opposed to comparing signatures of malicious behavior with current behavior), regardless of the existence of a learning process. In some cases, classifiers are trained using a representation of benign and malicious behavior or even random datasets. This is usually referred to by the authors of said approaches as anomaly detection although it could also fall into the misuses

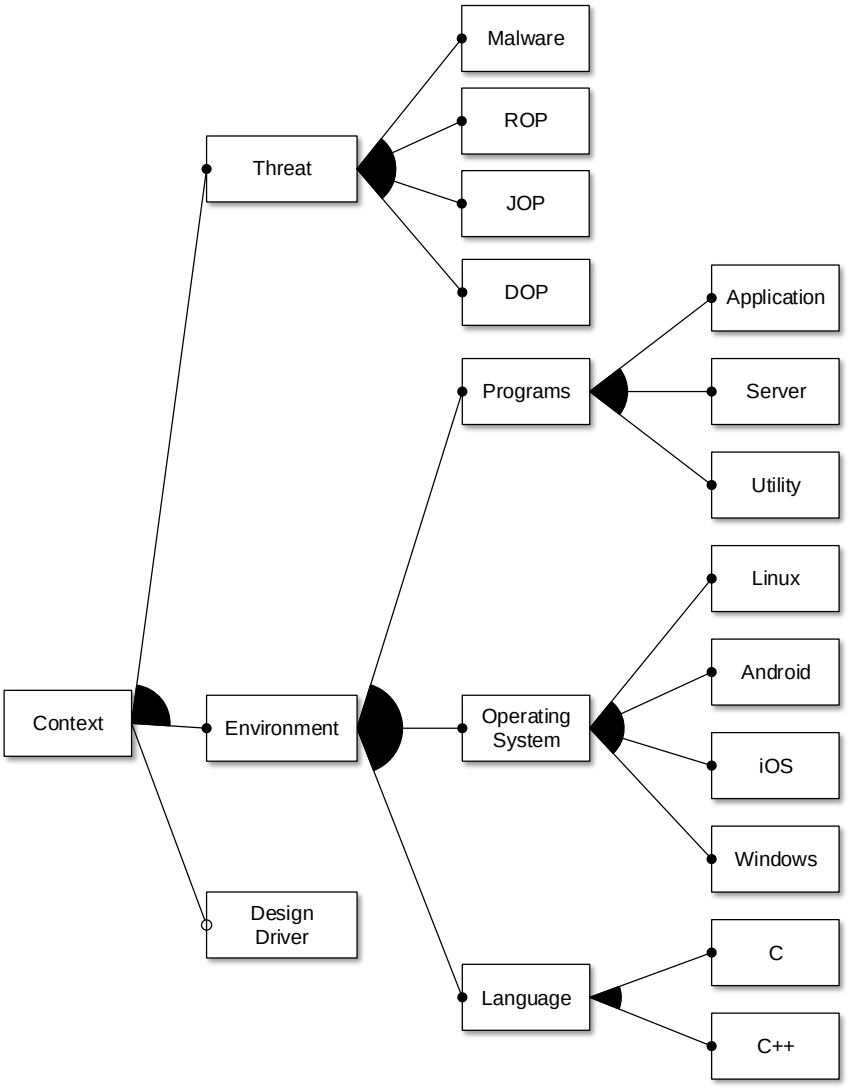

Figure 3.40.: `Context` sub-tree of feature diagram

detection category, since attack information is used to train the model. We refer to such approaches as anomaly detection since the feature model clearly identifies approaches where malicious data is used in the `Data` feature.

In a later NIST publication [131], a third approach was added called "Stateful Protocol Inspection", which is quite rare. We found no publication using such an approach in our state of the art analysis and, therefore, omitted it from our feature model.

Main Feature: Context (cf. Figure 3.40) The `Context` feature shows the specific `Environment` and the `Threat` an intrusion detection approach is targeted at. The environment consists of different `Program` types, `Operating System`, or a programming `Language`. The `Program` feature indicates if the IDS is targeted at a certain type of program, i.e., an `Application`, `Server` program, or a `Utility`. `Application` differs from `Utility` in that an application is targeted at ordinary users and a specific use-case (e.g. the Firefox Webbrowser) and a utility is used for general tasks mainly in the context of analyzing or maintaining a system (e.g. the SPEC CPU 2006 benchmark). Some IDS approaches are specifically targeted at certain `Operating Systems`. In our state of the art analysis, we incorporated examples of IDS on `Linux` systems, `Android`, `Windows`, and `iOS`. Furthermore, some IDS also target programs written in a specific `Language`, examples were found for `C` and `C++`.

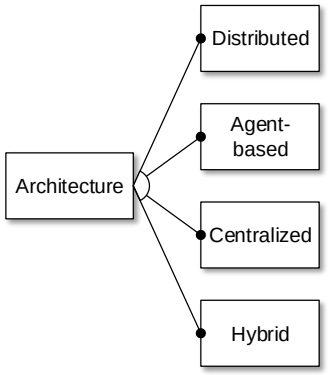

Figure 3.41.: Architecture sub-tree of feature diagram

Typically, these approaches employ a static analysis on the application's source code that has to respect language particularities.

The threats an approach is targeted at are attacks using return-, jump-, or data oriented programming (ROP, JOP,or DOP respectively) or generally Malware. More threats or attack techniques may be added in the future.

The context stated in the classification of the approaches is in most cases specific to the prototype implementation used in the publication. This does not necessarily imply that the approach is exclusively applicable to this context. The cardinality of the context feature allows to differentiate between several contexts. This is used to mark certain contexts as being directly respected in the conceptual design of an approach. To this end, we utilize the Design Driver feature. We select this feature if the authors identify a certain characteristic in the context of their approach and concretely argue how they respect this characteristic. In particular, a characteristic has to be more concretely elaborated on than general argumentations. For example, it is not sufficient to state that certain attacks alter the control flow of an application and use this as a motivation to utilize anomaly detection to identify these deviations.

Main Feature: Architecture (cf. Figure 3.41) The Architecture feature shows how the IDS is structured. We refine the taxonomy introduced by Lazarevic et al. [90] who defined two different types of architectures: "centralized" and "distributed & heterogeneous". A Centralized architecture means that the whole IDS is located on a single system, which it monitors. In this case, all data collection and decision making is performed on the central system.

An Agent-based system is split into a central part and agents deployed on several hosts. The agents perform data collection and send data to the central IDS instance, which performs the analysis and decides on potential reactions to a threat. Please note that we utilize this terminology because the term "agent" is used predominantly in the host-based IDS domain to refer to (remote) monitoring components of an IDS. However, it does arguably not comply to the definition of an agent in the domain of agent-based software engineering. There, an agent is defined as "an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives." [80]

In a Distributed application-based IDS, there is no central IDS instance. Autonomous instances are deployed across several hosts and perform data collection, analysis, and decision making. However, the IDS instances can communicate amongst each other and cooperate thusly, e.g., by sharing data or warning other instances of imminent threats.

Hybrid architectures combine the previously described architectures. This means that an approach featuring a hybrid architecture supports the flexibility to realize each of the architecture styles mentioned above.

Since most approaches covered in the state of the art analysis evaluate prototypical implementa-

tions, we could encounter specific architectures seldomly. Therefore, the feature is optional in our feature model.

Main Feature: Monitored Data (cf. Figure 3.42) The data source of an IDS greatly influences its capabilities. An IDS that monitors function calls can give very detailed feedback as to what exactly caused an intrusion within an application, but is potentially quite intrusive. An IDS monitoring system calls can give less information on an application specific cause for a raised alarm, but is usually able to use existing kernel instrumentation for monitoring.

Under the `Monitored Data` feature we list all data items monitored by the IDS found in our state of the art analysis, as shown in Figure 3.42. The monitored data ranges from `System Calls` over `Library Calls` to `Function Calls`. Some IDS monitor inter-process communication (`IPC`), while others are at a lower level, monitoring `Hardware Sensors` or `Network Events`, all the while providing feedback on the application level. Metadata is also sometimes monitored, as shown by the `Process` and `Thread Attributes` features.

Main feature: Analysis Technique (cf. Figure 3.43) Luh et al. [96] performed a systematic literature review on semantics-aware attack detection with a focus on intrusion and malware detection. To this end, they developed a categorization for detection system approaches. They establish four main categories, "Analysis & Detection", "Knowledge Generation", "General Properties", and "Data Collection". The "Analysis & Detection" and "Knowledge Generation" categories specify how detection works. They comprise the data processing, basic detection method, temporal domain, and also machine learning, clustering, and visualization techniques. We feel that these two categories characterize the analysis technique as a whole and define our `Analysis Technique` with inspiration from these two categories.

The technique employed by an IDS to process, analyze, and classify data is the most important and also usually the most complex part of an IDS. Therefore, the part of the feature model detailing our taxonomy for the `Analysis Technique` is rather complex as well (see Figure 3.43). The techniques are split into six sub-features: `Model Building`, `Conformance`, `Frequency Based`, `Machine Learning`, `Pre-Processing`, and `Classification`.

The cardinality of `Analysis Technique` is `[0..3]` because three is the highest number of techniques employed by a single approach in our state-of-the-art analysis. Therefore, the sub-features allow for the characterization of many different phases during the intrusion detection process. An `Analysis Technique` can be in the `Classification` or the `Pre-Processing` phase. Then, an `Analysis Technique` also has sub-features that determine how data is processed. The data analysis can either be used for `Model Building`, `Conformance` testing, a `Frequency Based` analysis, or a `Machine Learning` technique. Note that only one of these is possible, they are connected using a Boolean "XOR". In the following, we describe the sub-features of `Analysis Technique` in more detail.

Model Building `Model Building` is usually done during pre-processing phases, but could also occur when a model for result presentation is built during a classification. The model building process can be either `dynamic` (when it uses records of an actual system/program execution) or `static`, when source code or binaries are processed.

In the publications selected for the state of the art analysis, we encountered four different models used for intrusion detection (excluding the models used in machine learning, these are found in the `Machine Learning` sub-feature). A Control Flow Graph (`CFG`) is a graph representation of the control flow of a program, made up of basic blocks representing single program code lines without jumps and edges, representing jumps from one block to another.

A `Process Tree` is a tree representation of all processes running in a system, in which child nodes are the processes created by the parent node.

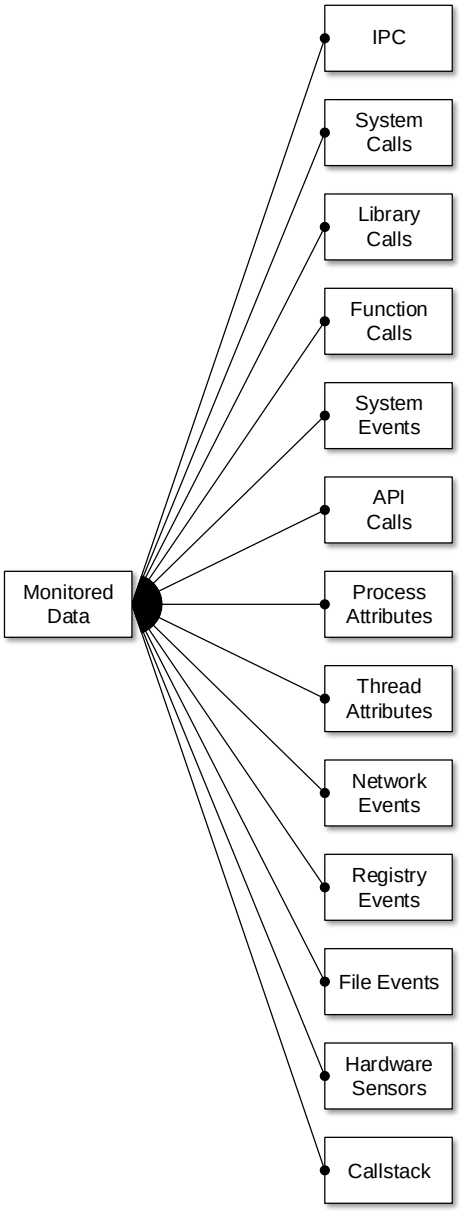

Figure 3.42.: `Monitored Data` sub-tree of feature diagram

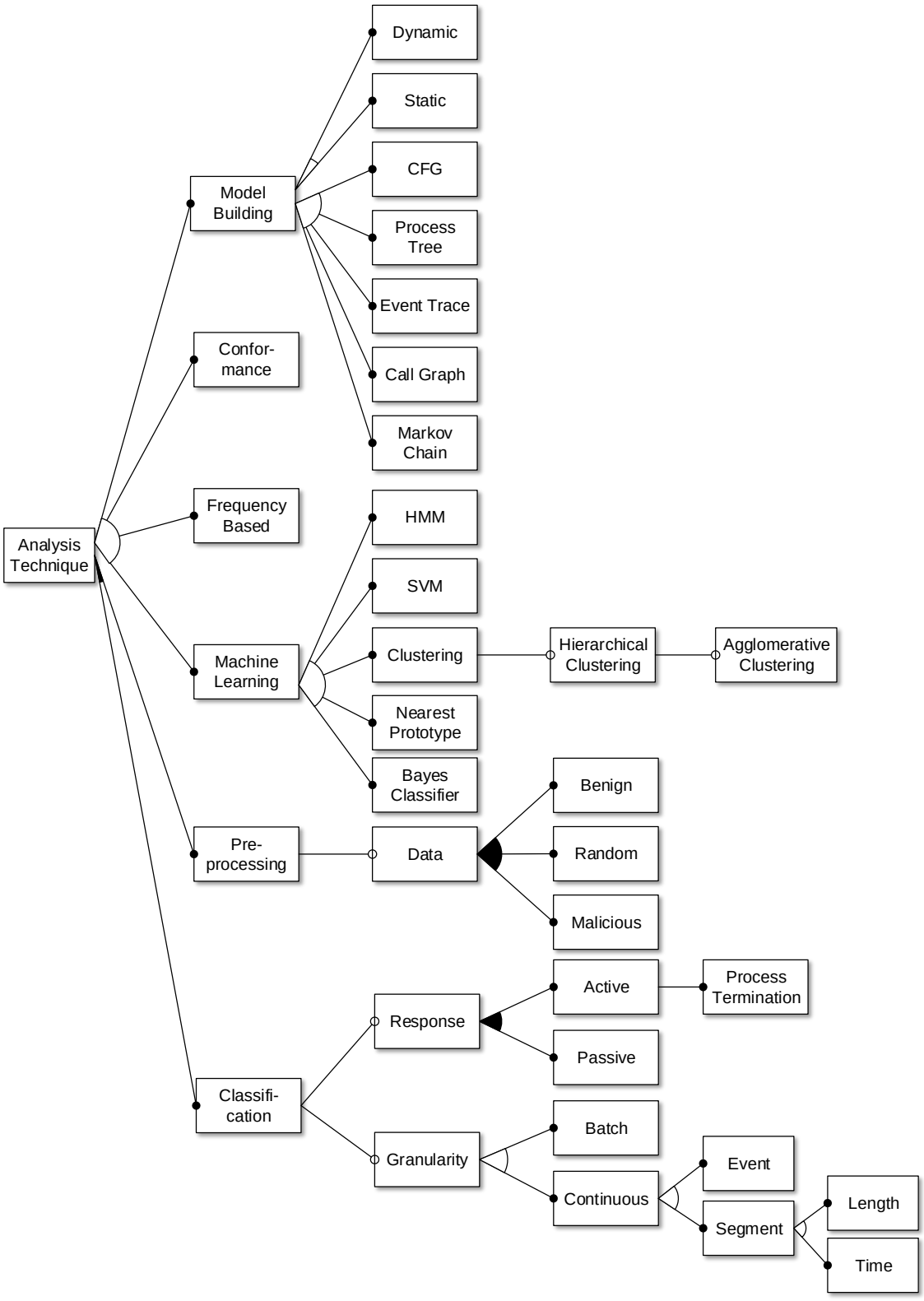

Figure 3.43.: Analysis Technique sub-tree of feature diagram

An Event Trace is a representation of events occurring in a system and the causes or originators of these events. Similar to a Process Tree, an event trace may contain processes and the creation of more processes, but is more general and can contain other events such as system calls, network information, or errors thrown by running applications.

The Call Graph Model usually represents application specific information. It records which routines in a program call other routines. The calls can also be represented in a matrix instead of a graph form.

Markov Chains are stochastic models. One may arguably expect Markov Chains as being a sub-feature of the Machine Learning feature rather than the Model Building feature. However, in the only approach employing Markov Chains they are only loosely coupled with the actual classification task. Therefore, we opted to define Markov Chain as a sub-feature of Model Building.

Conformance The Conformance feature is used to indicate that during the classification step an artifact is checked for conformance to a fixed specification. For example, a series of system calls from a monitored system could be checked against a system call sequence that is typically used by malware or the call graph of a monitored system could be checked against a call graph created by a static analysis of the applications source code. In particular, Conformance is not based on a statistical analysis. For example, a trace conforms to a CFG if it constitutes a path in this CFG.

Frequency-Based The Frequency Based analysis technique feature is always invoked when an analysis uses the frequencies of events or other monitored data fragments to draw conclusions about them. For example, a call graph may record how often a certain routine typically calls another routine.

Machine Learning Many IDS approaches use Machine Learning techniques, especially in the anomaly detection sector. Such approaches usually require a training phase, which can be indicated using the Pre-Processing feature. However, the Classification feature can also be used in conjunction with Machine Learning, to indicate that a previously trained classifier is used. The machine learning methods that are used in an analysis technique depends on the authors' approach. More detailed descriptions of the machine learning techniques can be found in the detailed discussions below when applicable.

Pre-Processing The Pre-Processing feature indicates that an analysis technique is used for processing before the IDS is actively monitoring and classifying system behavior. In the case of machine learning, the training phase is considered to be a pre-processing step. Model building is usually also done in pre-processing phases of the IDS operation.

The Pre-Processing phase may use different classes of data. When using machine learning techniques in anomaly detection systems, the data is usually Benign, meaning that it represents normal system operation without malicious intent. When Malicious training data is used, the model learns how attacks or malware manifest in the data. A malicious dataset needs to be clearly labeled as such. Training with Random data, i.e., data that is to a certain extent randomized, is also possible. In addition to machine learning, the different types of data can also be used for modeling or other pre-processing stages. If, for example, a control flow graph is built from an applications normal, non-malicious source code is built during a pre-processing step, the source code used is classified as benign data.

Classification The Classification feature shows how the IDS monitors a system and what its Response to a threat is. The response to an intrusion can be Active or Passive. In an active response, the IDS interferes with the program execution on the monitored system, e.g., through Process Termination of the process accountable for the intrusion. A passive response is more typical in the approaches evaluated in the state-of-the-art analysis, which usually give an alarm when an intrusion is detected.

Another part of the classification feature is the Granularity with which data is processed by the IDS. In the case of Batch processing, data is collected over a certain, relatively long, period of time and then analyzed by the IDS as a whole. This is usually the case if the goal is post mortem analysis because live system monitoring is not possible this way. The alert can only be thrown after a whole batch of data has been collected.

In the case of Continuous classification, the IDS monitors and analyzes a system continuously during its execution. The IDS either monitors every Event of the type it is designed to monitor and classifies the event as malicious or benign, or collects several events in Segments to be classified. The size of the segments can be determined in different ways: the feature Length indicates that a segment is a number of events long (though it does not have to be a fixed number) and the Time feature indicates that a segment contains all the events occurring in a time frame, no matter how many events occurred within that time frame.

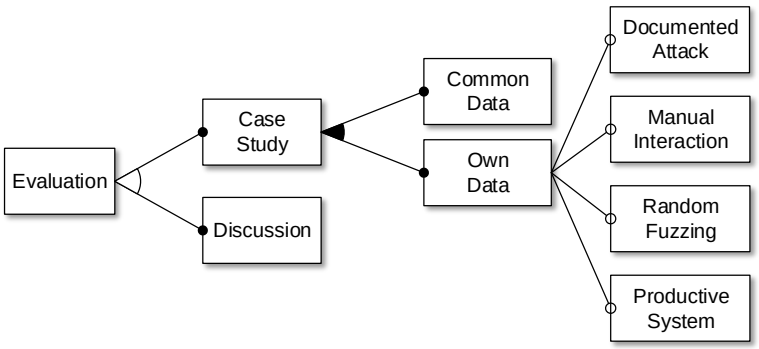

Figure 3.44.: Evaluation sub-tree of feature diagram

Main feature: Evaluation (cf. Figure 3.44) The Evaluation feature indicates how the authors of the examined papers evaluate their IDS approach. The evaluation can be a Discussion, in which the approach is reviewed informally, or a Case Study, in which an implementation (usually prototypical) of the approach is tested using test data. The data used for the case study can be Common Data, which are datasets available to the public specifically for the testing of IDS, like the datasets of system calls supplied by the University of New Mexico. The Own Data feature indicates that the data was collected or generated by the authors themselves. The way in which the data was acquired is also indicated in the feature model. The data can be generated manually through executing Documented Attacks or Manual Interaction with the system. Fuzzing indicates that the data was acquired through automated testing of the system, with unusual or randomized inputs. Typically, existing testing tools are employed for this kind of data generation. If data is collected from a system in active use, the Productive System feature is used.

Discussion

This section starts with giving the main insights we gained while reviewing the publications. Thereafter, we go into the detailed discussion and classification of each approach.

First, most approaches are application-specific due to a pre-processing phase where a static or dynamic analysis is executed on application specific data such as application logs. The advantage of these approaches is that they do not rely on meta information of the corresponding application, which makes them applicable in many situations. An exception to this are approaches that conduct a static analysis of the applications source code, which may not always be available.

Second, we could not find a technique that heavily utilizes information given by specifications of the corresponding applications. Especially in the automotive domain, where specifications are much more common due to restrictive regulations, this seems to be a valuable research direction. In the

context of anomaly detection based approaches, this information may be used to relieve complexity from the training phase of classifiers while maintaining or even increasing precision.

Third, none of the reviewed publications chooses a classifier based on application specifics. Very often, the argument for choosing a certain classifier is the availability of a corresponding implementation. A thorough evaluation examining the performance and precision of different classifiers in different application contexts seems to be missing.

Fourth, IDS having active response components seem to be rare. Systems relying on IDS with active response can be regarded as being self-adaptive in the sense of self-protection, one of the well-known self-X attributes [2]. A consolidation of these research domains seems to be promising. Thus, ideas like the MAPE-K reference architecture for self-adaptive systems [2], and insights from the research on quiescence [88] and state-transfer [56] might increase the self-protection capabilities of a system by means of systematic and targeted adaptations.

Fifth, monitoring the runtime behavior of applications comes with the risk of compromising intellectual property protection. This is even more severe in settings that may have to cope with commercial interests as it is the case for the APPSTACLE platform. Thus, research on privacy preserving monitoring in the sense of intellectual property protection is required.

Lastly, transferring computational expensive analyses to the cloud is a promising approach in the context of APPSTACLE. This is mainly the case as the APPSTACLE in-car platform certainly has to cope with resource constraints making costly analyses difficult to execute. Furthermore, APPSTACLE will provide a lot of the infrastructure needed for deploying analyses remotely. However, the possibility to deploy time-criticial analyses on the in-car platform still has to be given. Furthermore, the in-car platform can conduct certain pre-processing steps to lessen network traffic.

In the following, we present a detailed discussion and classification of the approaches that led to the feature model introduced in the proceeding section. For this classification, we label the discussed approach with the leaf features of the feature model. Here, all leaf features that are recursively associated with one of our main features (cf. Figure 3.38) are in square brackets. As these leaf features have unique names, the corresponding, complete configuration can directly be inferred from the labels.

Probabilistic Program Modeling for High-Precision Anomaly Classification [161] {Anomaly Detection}, {Server, Utility, Linux, C, C++, ROP}, {System Calls, Library Calls}, {Pre-Processing, Static, CFG}, {Benign, Random, Malicious, HMM}, {Benign, Random, Malicious, Length, HMM}, {Manual Interaction, Fuzzing}

Xu et al. introduce an approach to increase the precision of HMM-based anomaly detection in the context of IDS [161]. The main idea of the approach is to use statistical information about the execution of a monitored application to initialize the HMM.

Generally, the approach is in the domain of anomaly-based detection. Their prototype implementation STILO is implemented as a Linux application and used to secure server (proftpd and nginx) and utility (flex, grep, gzip, sed, bash, and vim) programs written in C and C++. The authors target threats that alter the control flow of applications and explicitly mention ROP exploits. Furthermore, they monitor and analyze system and library calls of these applications. Montitoring is realized by using strace and ltrace.

The first step to do so is to execute a static analysis of the applications' source code. This yields an control flow graph that is further processed to compute the probabilities of consecutive system calls.

Then, they utilize this information for the initialization of an application-specific HMM. The HMM is used for the classification of system call sequences as being benign or malicious, respectively. To further refine the HMM, Xu et al. train it with system call segments (n-grams) of a certain length. In their evaluation they used segments of length 15.

They define tree different kinds of segements:

- *Normal* (Bening) segments obtained from manual interaction with the application

- *Abnormal-A* segments (Malicious) obtained by reproducing real-world exploits that are not referred to in more detail
- *Abnormal-S* segments (Random) obtained by replacing the last third of a normal segment with randomly ordered calls from the legitimate call set

All tree kinds of segments are used to train the HMM. However, in their evaluation they only refer to the classification of *Normal* and *Abnormal-S* segments.

Classifying malicious system behavior using event propagation trees [99] {Anomaly Detection}, {Windows, Malware}, {Agent Based}, {Process Attributes, Thread Attributes, Network Events, Registry Events, File Events}, {Benign, Dynamic, Process Tree}, {Benign, Agglomerative Clustering}, {Batch, Nearest Prototype}, {Productive System}

Marschalek et al. present an approach that mainly relies on process trees. These are "human-readable representations of an executables and all the activity they displayed during their lifetime" [99]. Basically, they classify these process trees as being bening or anomalous. Thus, their approach is in the domain of anomaly detection. Their main motivation is to identify malware on Windows systems. To this end, they deploy monitoring agents on these systems and gather the following data in a central database:

- Process Attributes, i.e., the image name, PID, parent PID, usercontext (username, group, owner), time of start and miscellaneous other information that they do not refer to in more detail
- Thread Attributes, i.e., thread IDs
- Network Events, i.e., access of network resources and establishment of new inbound or outbound connections via IPv4 or IPv6
- Registry Events, i.e., accesses and changes of the windows registry (the granularity of these events is not explained in more detail)
- File Events, i.e., loading of protable executables or DLLs; creation, access, reading, modification, or deletion of other files

In the next step Marschalek et al. consolidate this dynamic information in form of a process tree. This process tree is then converted in a format that can be processed by the Malheur tool suite [7].

Malheur uses a prototype based hierarchical clustering to cluster, in this case, benign process trees. Marschalek et al. then utilize Malheur's classification phase to check whether newly gathered process trees belong to an benign cluster or are anomalous (rejected). Since the approach classifies process trees, we used the Batch feature instead of the sequence feature. The approach may also be used on partial process trees of certain length or gathered in certain time intervals. However, the authors do not elaborate on this in more detail.

In the evaluation of the approach, data from a company network is used. Furthermore, an isolated machine was infected with three different malware samples.

LEAPS: Detecting Camouflaged Attacks with Statistical Learning Guided by Program Analysis [55] {Anomaly Detection}, {Windows}, {System Calls, Library Calls, Function Calls}, {Benign, Malicious, Pre-Processing}, {Dynamic, Event Trace, Pre-Processing}, {Hierarchical Clustering, Pre-Processing}, {SVM, Pre-Processing}, {SVM, Batch}, {Productive System}

Gu et al. aim to detect attacks that use benign programs to cover up their malicious behavior. To achieve their goal of classifying behavior as being benign or malicious, they employ anomaly detection as their basic approach. Though, they also use malicious executions to train their classifier.

The approach is demonstrated using a prototypical implementation on a Windows system running a specific set of utility applications (vim, notepad++, putty, winscp). However, the approach could work on other platforms as well. They perform several case studies demonstrating the effectiveness of their prototype using different exploits for the applications.

[7] http://www.mlsec.org/malheur/

Their method of detecting anomalies relies on the collection of system events, library calls, and function calls, depending on the application and availability of data on the platform. On their Windows evaluation platform, they are able to monitor "system events", caused by function calls, library calls, or system calls.

To generate the system event logs, Gu et al. execute benign and malicious applications on a test system. The collected raw event logs are then pre-processed by correlating function and library calls gathered through a stack walk to the system events, leading to a list of system events with attached function and library information for the malicious and benign executions. This list is then split into application and system stack traces to differentiate application and operating system behavior.

Hierarchical clustering is used to partition the system stack traces into behavioral clusters, which leads to the benign and malicious (or mixed) dataset later to be used in a statistical machine learning process. Gut et al. use the application stack traces to build CFG's for the benign and malicious application executions. The execution paths in the CFGs can be mapped to system events in the datasets generated from the system stack trace. Differences in the execution paths in the benign and malicious CFG are calculated and used to assign weights to the events in the mixed dataset, forming a weighted dataset. They train a weighted SVM to classify behavior as malicious or benign, using the benign dataset as the positive and the weighted dataset as the negative samples in the training phase of the SVM. The learned SVM classifier is then applied to system logs gathered and processed exactly like the training data in the aforementioned pre-processing phases, making the approach a batch analysis.

Gu et al. evaluate their approach in a case study using data gathered in a productive system.

Host Intrusion Detection for Long Stealthy System Call Sequences [105] {Anomaly Detection}, {Server, Linux}, {System Calls}, {Benign, Malicious, Dynamic, Markov Chain}, {Length, Bayes Classifier}, {Common Data}

Elgraini et al. introduce a host-based intrusion detection approach targeting hidden malicious intent in long process system call sequences. Since they train their classifier using benign and intrusive data sets, we classify the basic approach as anomaly detection. The authors evaluate their approach on a Linux platform running server applications such as sendmail. Their IDS requires only system calls to be monitored.

The classifier employs a naïve Bayes approach and the prior probabilities are estimated based on the training data, which contains both benign and malicious sequences. The estimation is based on the fraction of sequences labeled as either benign or malicious To increase the accuracy of their model, the authors use a Markov model to calculate the class conditional probability of a sequence. The Markov Model estimates the probability of a sequence occurring when given a supposed classification (benign or intrusive). The transition probabilities in a sequence are again estimated using malicious and benign training data.

Using both the prior probabilities and the class conditional probabilities derived using the Markov model, the sequences are classified using the Bayes classifier.

The authors evaluate their prototype using a common dataset provided by the University of New Mexico containing labeled system call sequences from a system running *sendmail*. The sequences vary in length and are classified one after the other by the prototype.

Long-Span Program Behavior Modeling and Attack Detection [134] {Anomaly Detection}, {Server, Utility, Linux}, {Design Driver, DOP}, {System Calls, Function Calls}, {Benign, Dynamic, Call Graph}, {Benign, Agglomerative Clustering}, {Passive, Batch, Agglomerative Clustering}, {Benign, SVM}, {Conformance, Batch}, {Manual Interaction, Productive System}

Shu et al. introduce an approach to detect stealthy attacks in very long system call traces by employing machine learning techniques to correlate events in the traces that can be very far apart. Their main goal is to be able to detect attacks that do not directly alter the application control flow (data oriented programming [65]) or exploit legal control flows (e.g. denial of service attacks). They

propose using a mildly context sensitive grammar as the basis for such an approach. In order to achieve this, they design a program anomaly detection approach, which they evaluate on a Linux system using both server and utility applications. They propose to monitor generic system events such as jumps, function calls, or even generic instructions in program traces. However, in their implementation and evaluation the authors opt to monitor system and function calls, depending on the monitored application. The Analysis is split into two main phases: training and detection. The basis for both phases is the "behavior profiling" step, which splits raw program traces into trace segments. These segments are used to create event co-occurrence and event transition frequency matrices containing routines that occur together and the number of occurrences of calls from one routine to another respectively. The routine names are retrieved using static analysis. The matrices essentially represent a dynamic call graph. Then, the first training step uses an agglomerative clustering algorithm to separate all normal behavior instances into clusters.

The first detection step of the analysis tests whether a behavior instance fits into a cluster. It is checked if the co-occurred events in the behavior instance are consistent with the co-occurred events in a cluster, if so, the behavior instance fits into the cluster. If no fitting cluster is found an alarm is raised, otherwise the behavior instance is passed on to the occurrence frequency analysis.

The occurrence frequency analysis is based on a training step called intra-cluster modeling. Therein, a deterministic method and a probabilistic method are used to further refine the boundary for normal behavior within the clusters according to the co-occurrence frequency of events. The probabilistic method is an SVM and the deterministic method employs variable range analysis.

In the second detection step (occurrence frequency analysis), the behavior instances are examined for quantitative frequency relational anomalies. If the frequencies do not fit the models derived in the previously described training step, the instance is reported as anomalous.

The authors evaluate their approach using a prototypical implementation on a Linux platform using the applications sendmail, sshd, and libpcre. The training data is gathered through monitoring a productive system or manual interaction, depending on the application.

A Host-based Anomaly Detection Approach by Representing System Calls as States of Kernel Modules [108] {Anomaly Detection}, {Linux, Application, Utility}, {System Calls}, {Frequency Based, Benign} {Frequency Based, Batch}, {Common Data, Known Attack, Productive System}

Murtaza et al.'s approach is centered around representing system call sequences as kernel module states to allow an analyst to easier deduce the behavior of an application in the case of an anomaly, by examining the states. In contrast to many anomaly detection approaches, this approach does not rely on machine learning techniques but on a frequency-based analysis.

The approach is targeted at Linux based systems and the prototype is evaluated using various Linux programs, both utilities and end user applications.

Since Linux system calls (usually) belong to one of eight Linux Kernel modules, the system calls can be mapped to states that represent the kernel module they belong to. The authors propose analyzing these state sequences in terms of the proportions to which the module states occur, based on the observation that the distribution of states is significantly different in anomalous traces compared to normal traces. To this end, the authors develop a "Kernel State Modules" algorithm which analyses traces that are separated into training, validation, and testing sets. The proportions of kernel module states are calculated using the training set. Then, a trace from the validation set is used to adjust the threshold with which anomalous traces are recognized in the case that a normal validation trace is falsely categorized as anomalous. This is repeated with all traces in the validation set as long as false positives occur. The testing set is then used to evaluate the technique with the calculated threshold and the testing set. To analyze potentially anomalous traces, the traces are passed to an evaluation function (also used in the threshold calculation with the validation set). The approach is evaluated using a variety of applications and utilities on a Linux platform (Firefox, Login, PS, Stide, Xlock) with different datasets, depending on the application. They use both common data and data generated on

their own through testing frameworks and productive system monitoring. The anomalous traces are generated by using known attacks on the evaluated programs. Murtaza et al. compare their approach to popular anomaly detection algorithms, namely a Hidden Markov Model approach and Stide.

The Best of Both Worlds. A Framework for Synergistic Operation of Host and Cloud Anomaly-based IDS for Smartphones [35] {Anomaly Detection}, {iOS}, {Hybrid}, {IPC, System Calls, Library Calls, API Calls, Hardware Sensors}, {Discussion}

Damopoulos et al. focus on the architectural aspects of developing an IDS for smartphones [35]. They argue that in this domain certain analyses should be performed on the mobile device where other analyses should be performed in the cloud. For example, an analysis that includes private data is typically more acceptable to be performed on the mobile device. Whereas, computationally expensive analyses should be executed in the cloud.

In general Damopulos et al. focus anomaly-based detection. Their prototype is implemented for iOS. Additionally, they claim that an implementation for Android is straight forward.

Their architecture consists of the following components:

- *Event Sensors* collect information from different layers of the operating system. The authors state that their prototype is able to collect system calls, inter-process communication, data from hardware sensors, API calls, and library calls.
- The *System Manager* decides which analysis technique is used and forwards events to the Detection Manager
- The *Detection Manager* supports two types of engines. A host-engine runs on the mobile device, whereas, a cloud-engine executes the analysis in the cloud
- The *Response Manager* supports modules that specify different kinds of actions that are executed in the case that an event was classified as being suspicious, e.g., requesting user input, blocking of events, or demanding re-authentification.

Damopulos et al. classify their approach as being hybrid because of the flexibility in the deployment of analyses. We follow this classification to separate their approach from agent-based approaches that do not share this kind of flexibility. However, their approach is similar to agent-based approaches in the sense that event sensors can generally be regarded as being agents.

The discussed publication focuses on the architecture of their IDS and only states that they use a Random Forest classifier. They implemented four smartphone detection mechanisms using the proposed architecture and refer to the corresponding publications [33, 34, 36, 83] for further details on the analysis technique. However, we do not incorporate this information into this classification.

The evaluation of the four detection mechanisms was performed on user-behavior profiles created based "on real data collected from a critical mass of participants" [35].

Efficient, Scalable and Privacy Preserving Application Attestation in a Multi Stakeholder Scenario [3] {Misuse Detection, Anomaly Detection}, {Application, Linux}, {Distributed}, {System Calls}, {Discussion}

Ali et al. [3] introduce an approach to measure and report dynamic behavior of applications in accordance to the Trusted Platform Modul specification of the Trusted Computing Group (TCG)[8]. Monitoring data is stored in so called Platform Configuration Registers (PCR). They address the problem that the TCG restricts the number of PCRs to 24. Thus, it is not possible to use one PCR per application. Ali et al. use one PCR per stakeholder, define an algorithm to aggregate monitoring data of multiple applications of the same stakeholder, and specify an protocol for the remote attestation of application behavior. This protocol also ensures the integrity of the monitored data. Furthermore, this concept tackles the problem of privacy violations that may result from the solution of using only one PCR for monitored application data.

The authors use their approach in the context of anomaly detection. However, the publication focuses on measurement and reporting. Thus, we assume that the concept is not strictly restricted

[8] https://trustedcomputinggroup.org/tpm-library-specification/

to this kind of approaches. The prototype is essentially a Linux kernel module. Furthermore, it is tested monitoring the applications AMIDE, FreeMedicalProjectsForms, Firefox, Thunderbird, and GNUCash.

Each system stores the monitored data and transmits it in certain intervals to a challenger for verification. The verification itself is not split across several challengers. Nevertheless, we classify the architecture of the approach as being distributed because the role of the challenger is not restricted to one central entity. The monitoring itself is performed by a specific Linux kernel module that intercepts system calls.

This publication does not focus on analysis techniques but the authors refer the reader to [71] for a detailed description of the employed technique. Therefore, we do not incorporate this aspect in our classification. Furthermore, the evaluation of the approach is conducted in form of a discussion. Here, the authors refer to insights they gained while testing their prototype for the analysis of the applications mentioned above.

Detecting Code Reuse Attacks with a Model of Conformant Program [76] {Anomaly Detection}, {Server, Utility, Linux}, {Design Driver, ROP, JOP}, {System Calls, Callstack}, {Pre-Processing, Static, CFG}, {Conformance, Process Termination, Event}, {Documented Attack}

Jacobson et al. target their approach [76] specifically at return oriented programming (ROP) and jump oriented programming (JOP) attacks. ROP and JOP are used typically utilized to realize code reuse attacks. Both aim at constructing exploits from code that is already present within a process. To this end, ROP attacks overwrite return addresses and JOP attacks overwrite jump addresses. Thus, both alter the control flow of processes.

Essentially, the approach of Jacobson et al. identifies control flow deviations between the runtime behavior of a process and a CFG that is inferred from the corresponding binary. Hence, we classify their approach as belonging to the domain of anomaly detection.

ROPStop, the implementation of their approach, is build on top of the Dyninst toolset [9]. It is evaluated running on Red Hat Enterprise Linux [10]. ROPStop intercepts system calls and essentially checks whether the current callstack is valid w.r.t. the CFG constructed by statically analyzing the corresponding binary. Jacobson et al. state that control flow deviations manifest such that at least one of the following assumptions is invalid:

- the currently executed instruction is specified in the binary
- the callstack is valid, i.e.,
  - the instruction at the return address of each stack frame is immediately preceded by a call instruction
  - the CFG inferred from the binary defines control flow transfer between each caller and callee frame

If these assumptions do not hold, the process is terminated before executing the system call.

The authors evaluate their appraoch against five documented ROP and JOP exploits and where able to detect all of them. Furthermore, the performance of ROPStop was evaluated by utilizing Apache [11] and SPEC CPU2006 [12]. The result was an 6.3% (Apache) and 5.3% (SPEC) overhead on average.

System Call Anomaly Detection using multi-HMMs [162] {Anomaly Detection}, {Server, Linux}, {System Calls}, {Benign, Clustering}, {Length, HMM}, {Common Data}

Yolacan et al. use HMMs to detect anomalies in system call sequences. However, in their approach, multiple HMMs are used, one for each cluster of similar process executions.

[9] http://www.dyninst.org/
[10] https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux
[11] https://httpd.apache.org/
[12] https://www.spec.org/cpu2006/

For the evaluation of this anomaly based approach, the authors use the University of New Mexico (UNM) system call set of the "sendmail" program, a Linux server program. Though the approach could work in different contexts, we categorize it according to the dataset used for evaluation.

No other data than the system call traces from the aforementioned training set are used in this approach.

The system call sequences are first pre-processed to improve the accuracy of the HMM classification in the further operation. First, the program traces contained in the UNM dataset are partitioned by process IDs. Then, identical processes are removed, in order to avoid same sequences in training and testing. After this data reduction step, clusters based on the structural similarities of processes are formed. In this case, three clusters were identified. Lastly, the system call traces in the training set are split into fixed-length subsequences of length six.

For the classification of system call trace segments, the authors employ one HMM per cluster identified in the pre-processing stage. The length of the system call segments is the same as in the training step.

The authors report high detection accuracy and comparatively low false positive rates for their multi-HMM approach.

3.5.2. Network IDS

Scope

Network monitoring and intrusion detection for CAN network have already been addressed for over a decade [22]. A Network-based IDS (NIDS) monitors network packets for a specific part of the network [130]. Furthermore, it protects end-points by analyzing the traffic going to and coming from them. Generally speaking an NIDS architecture contains several sensors deployed at strategic points on the network where they monitor and analyze traffic and report attacks to a central management console [13]. Some of the main advantages of NIDS systems are that they can detect network-specific attacks and have no impact on the performance of the host [37]. However despite their many advantages, car manufacturers seem not very eager to embed such capabilities inside their vehicles leaving cars unprotected against malicious attacks, such as [1]. These attacks have led to current considerations of adding a security layer in automotive environments, which consist of legacy network protocols that were designed with no security in mind and are rather trivial to abuse. In this section we will mention different attempts proposed in the literature with regard to securing and protecting automotive systems and justify our interest for NIDS.

Threats surface Adversaries target in- and ex-vehicle vulnerabilities in order to gain access to in-vehicle network and the different car functionalities. To this end, an attack may target non-sensitive system components that can be used as a medium to grant access to the rest of the network. These components can be later used to interrupt the normal operation of critical components in different car states (e.g. driving or parking mode). In general, the available attack surfaces are that can be exploited in order to gain direct access to the physical in-vehicle interfaces are shown in Figure 3.45.

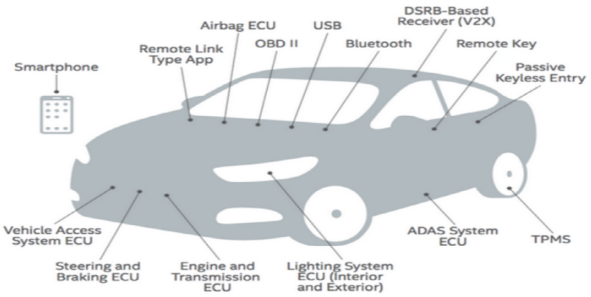

Figure 3.45.: Vehicle attack surfaces (source [23])

According to this figure they can be divided into three categories, namely interfaces providing:

1. direct physical access such as the On-Board Diagnostics (OBD-II) port (federally mandated in the U.S. after 1996). Other examples include the anti-theft key, that is used to lock the car as well as the USB/CD drives of the infotainment unit.
2. short-range wireless access, such as the Tire-pressure monitoring system (TPMS) interface, consisting of pressure sensors to measure the air within each wheel and indicate it to the car dashboard. Other examples are the keyless entry, used to lock/unlock the car on distance and the radio, providing in-car entertainment and information for the vehicle passengers
3. long-range wireless access, such as the Data Set Ready (DSRB) receiver for Vehicle-to-Vehicle communication, Telematics interface that is used in the infotainment unit of the vehicle. In this category we also find the OBDII wireless and Bluetooth modules that were mentioned in previous paragraph and provide physical access to the OBD-II port. Finally, Over-the-Air updates are another attack surface. They are usually issued by the car manufacturer in the form of firmware (FOTA) or software (SOTA) updates.

Securing those surfaces become an initial concern in 2010 when Ishtiaq et al. [70] attempted an short-range wireless access attack by penetrating into the TPMS system.

Few years later Miller and Valasek in [103] performed different attacks targeting a number of the attack surfaces of Chrysler's Jeep Grand Cherokee. They successfully issued various commands remotely to kill the car's engine, and even stop the braking or steering functionality. As a result, Chrysler was forced to take costly measures to fix the security issues, and recalled 1.4 million vehicles.

Having introduced the in-vehicle surfaces that may be vulnerable and exploited for adversary attack scenarios, in the following section we provide deeper insight on the motivations and techniques that are used by adversaries to achieve their goal.

Motivations and means The lack of security mechanisms makes it relatively easy for adversaries to gain access to in-vehicle functionalities. And even though Miller and Valasek have spent almost 2 years and approximately 1 million dollars to understand the Jeep's architecture, find its vulnerabilities and perform attacks, they have demonstrated that vital threats to the vehicle's passengers are possible. This triggers a serious question on the possibility that such attacks did not happen in the context of a research study, but an actual attack scenario. Diverse types of persons could have interest in car hacking, ranging from car reseller or a manufacturer competitor, to cyber-criminal. Even though each category includes different motivations, the techniques and procedures that are followed to exploit the vulnerabilities of in-vehicle architectures are generally similar to the research study of Miller and Valasek. Motivations to this end could be:

1. *vehicle theft* or copy the vehicle's architectural designs and specifications
2. *espionage* for tracking and recording sensitive information (e.g. current location, contacts and addresses), in order to track people, eavesdrop on their calls and in-cabin conversations.
3. *suppression of vehicle notifications* and avoidance of incurring replacement expenses targeting the safety of the vehicle. This technique is observed in second-hand (used) car dealers, in order to hide faulty components and increase the price of the vehicle.
4. *physical harm and wide-spread damage*. This is the main type of motivation used by nation-states, underworld and terror organizations who are well-founded to exploit in-vehicle vulnerabilities and be able to control them remotely.

Up to this point we have described the most common threat surfaces in the vehicle architecture and the motivations of eventual adversaries. To be successful, an adversary should know thoroughly the target in-vehicle architecture, in order to 1) understand the information that are exchanged and 2) format the information to be injected in a way that they are handled by the other ECUs. Otherwise, the packets may be treated as malformed and therefore discarded by the gateway firewall. Additionally, the adversary needs to reverse-engineer the manufacturer-specific way that is used to encapsulate the data in a network frame, which may require deep in-vehicle protocol understanding. In turn, a security engineer should also have the same understanding in order to be able to block potential threats. Furthermore, threats are also classified based on their severity should and likelihood. For example, safety-related threats that put in danger passenger lives are the most vital and they blocked first. In this context we understand the needs to protect cars from such threats. To this end, in the following section we provide thorough details on intrusion detection systems and how such techniques have been developed to protect in-vehicle networks.

Overview

Recent demonstrations [1] have shown that automobiles present a threat surface rather large which triggers the interest of hackers and intelligence agencies. The criticality is such that developing an effective security module for in-vehicle networks as the NIDS is now a pressing matter. But first let us have a look of what "security module for in-vehicle networks" means.

To be able to secure and protect a network in the "traditional" IT desktop world, one first needs to know what is happening on this network. Network monitoring is the capability to provide in real-time a clear picture of the current health status of the network and therefore facilitate anomaly detection. Network monitoring is a broad term which could convey different meaning depending on the context.

More generally speaking, it focuses on monitoring the performance of the network. However in a security context, network monitoring can help spotting malicious activities and is commonly referred as *intrusion detection*.

Besides the technical differences, automotive networks could be regarded as similar to computer networks to a certain extent and therefore require the same monitoring capabilities to identify anomalies. Researchers have been already investigating how to apply monitoring techniques to the automotive world for some years. In the following state of the art we will review the different work efforts proposed until now and will identify some leads to guide us in our future development. To make this possible we will first introduce a set of notations and definitions to provide the reader with the necessary background information. Then different metrics will be presented in order to be able to distinguish and rate the different NIDSs. Finally we will detail a taxonomy of detection systems for automotive in-vehicle networks in which we will investigate the different technologies proposed in the literature.

Different approaches exist on how to design a NIDS. Before discussing them, we will first look at a set of metrics we can use to evaluate and compare them.

Metrics Since we will investigate in this state of the art the different NIDS available for in-vehicle networks, it is important to agree on metrics in order to assess the efficiency of the detection systems. Porras and Valdes proposed the following three metrics [120]:

a. Accuracy: Describes how (un)successful the system is in detecting incidents. For a given event, a *false positive* occurs when the NIDS flags wrongfully the benign event as malicious. By opposition when the NIDS does not recognize a malicious event and considers it legitimate, a *false negative* occurs. Finally a *true positive* happens any time the NIDS correctly detects a malicious event. We will propose below some additional values we can use to better assess the accuracy of a system. The ability for a system to dtect all the attacks is called as *detection rate*.

b. Performance: Decribes the event processing rate. Depending on the system being monitored, the rate needs to be high enough to guarantee real-time detection. The processing rate has a direct impact on the *detection latency*, defined as "the time that elapses between the onset of an attack and recognition of the attack by an intrusion detection system" [143]. The smaller the latency, the greater the chance to thwart the attack. Due to the real-time constraints of automotive systems, the performance is a crucial factor for the NIDS assessment.

According to Mitchell et al. [104] the accuracy can be determined based on its detection rate on a specific attack set. In particular, Table 3.18 introduces a terminology that will be used in this document for measuring the detection rate of a NIDS based on actual attacks.

Actual attack / Detection rate	Yes	No
Yes	True Positive (TP)	False Positive (FP)
No	False Negative (FN)	True Negative (TN)

Table 3.18.: Attack detection terminology

Specifically, the most basic and commonly used metrics introduced by this table are the True Positive Rate (TPR) and False Positive Rate (FPR), which indicate the probability that the IDS outputs an alarm when there is an intrusion and the probability that the IDS outputs an alarm when there isn't respectively. The remaining rates of this Table are the False Negative rate (FNR) and the True Negative Rate (TNR), which can be calculated using the formulas: FNR = 1-TPR and TNR = 1-FPR. Ideally a perfect NIDS would have an FPR equal to 0 and therefore a TPR of 1, meaning it is capable to detect every attack without making any mistake.

NIDS Taxonomy

The field of intrusion detection has evolved over time and new approaches have been proposed to implement an IDS. For our study we will rely on the revised taxonomy of IDS [37] in which Debar et al. introduced five concepts to classify IDS. First of all the *detection method* describes the two complementary approaches used for detection. The first one, called knowledge based, uses information about the attacks to detect them. The second approach, known as behavior based, builds a reference model of the normal system behavior during a learning phase and detect attacks by looking for deviations from this model. The reference model can also be specified manually. This approach is referred as behavior-specification based and can be seen as a subset of behavior based [104] or as an independant approach [86]. We will cover in greater details these approaches below. The second concept, *behavior on detection*, qualifies how the IDS reacts to detected attacks. It can actively take corrective or pro-active measures to counter the attack, or it can simply raise an alarm in a passive manner. The *audit source location* relates to the kind of data used by the IDS for analysis. It also helps categorizing IDS. A network based IDS will use packets on the network. An application based IDS will leverage application logs, while a host based IDS will focus on system logs and audit trails. The fourth concept, entitled *detection paradigm*, defines how the IDS will detect attack: it can either evaluate states or transition [67]. Finally depending on its *usage frequency*, an IDS can either monitor continuously in real-time or periodically by taking snapshots regularly.

There are several ways to categorize IDS. According to the literature, we commonly rely on either the location of the audit source or the detection method, where we find application-based and network-based IDS solutions. However, since application-based IDS was already addressed in Section 3.5.1 of this document, in this section we are solely focusing on network based solutions, which we' ll categorize them according to their detection methods. For each method we will first discuss how they work and then present the detection systems proposed in the literature for in-vehicle networks. An overview of the available state-of-the-art detection methods is provided in Figure 3.46.

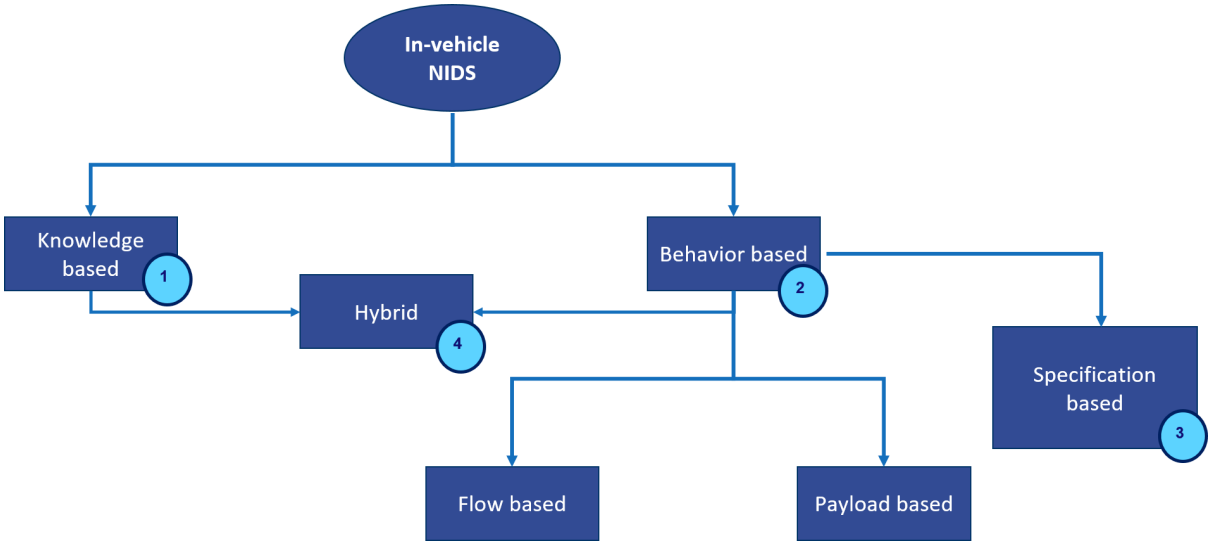

Figure 3.46.: In-vehicle NIDS taxonomy

1. **Knowledge based**: Also known as *signature based* or *misuse detection*, a knowledge based NIDS uses information about attacks, so-called signatures, as a pattern characterizing a known threat [104, 37]. The NIDS will compare the signatures against observed events to identify possible attacks [130]. Upon detection (i.e., an event matches a signature) certain action(s) will be executed.
2. **Behavior based**: Referred sometimes as anomaly based NIDS, it first creates a reference model

(or "profile") of a target system by collecting and recording "normal" (legitimate) operations. Once done, the NIDS starts monitoring the current activity of the system. It then generates an alert anytime it identifies a significant deviation from the model [38].

3. **Behavior-specification based:** In the same fashion than behavior based approach, this technique detects attacks by identifying deviations from a norm [132]. But instead of creating the normal behavior model during an initial learning phase, specifications are manually developed to characterize legitimate program behaviors [147].
4. **Hybrid NIDS**: By combining two (or more) detection techniques together, one can obtain the best of them while overcoming their limitations. As we previously mentioned, a common combination is knowledge based and behavior based detection.

Below we will detail more on existing works in the scope of the presented taxonomy categories.

Knowledge based NIDS are effective to detect known threats with great *accuracy*: they generally present a very low rate of false positives since the use of signatures guarantees that every match means that a malicious event has been successfully detected. Although the main drawback comes with *completeness*: the set of signatures cannot be exhaustive and therefore such NIDSs are unable to catch unknown threats. With a finite and limited number of signatures, attackers have no difficulties to find and exploit new vulnerabilities. Moreover some techniques can be employed in order to bypass detection systems [29], such as payload encoding. Additionally the more advanced and complex the signature database becomes, the bigger the CPU system load during analysis.

In the automotive context the application of knowledge based detection seems to be limited. The research encountered in the literature tend to discard this approach [111]. First of all as observed by Miller and Valasek, all known CAN injection attacks take one of two forms: either CAN diagnostic messages or standard messages sent with a highly inflated rate [102]. According to the standards no diagnostic message should be seen on the network when the car is moving. It is rather straightforward to develop a knowledge based NIDS to detect attacks relying on the use of diagnostic messages. Miller and Valasek proposed such approach in [102]. We will describe it further in the section entitled "hybrid NIDS". The rest of the attacks can be detected with another intrusion detection approach called "behavior based" (see next section). It seems there has been no other network knowledge based NIDS proposed in the literature.

The main drawback of knowledge based detection in in-vehicle networks is the frequent update requirement of the signature database [110]. As new attacks will undoubtedly emerge, how can we make sure to have the latest NIDS update installed at any given time in cars? Asking the car owners to manually update the systems themselves could be a daunting task for the non-technical persons. Another idea could be to have the mechanics with the adequate skills performing such tasks during a checkup at the garage. However the timeframe between two visits at the garage would be too big, leaving the car exposed to eventual threats for an unacceptably long period. Hopefully the future adoption of (secure) Over-The-Air (OTA) updates and the deployment of 5G tehnology could remediate this challenge.

Even though knowledge based NIDS should not be completely discarded: once combined with a behavior based approach (as detailed below), an hybrid NIDS can demonstrate great performance. We will come back to it later.

Behavior based NIDS will first develop a model, or profile, of a system's normal behavior, and will then spot behavior deviations from the norm. Building such profiles can be done more or less autonomously with machine learning algorithms using a sample of historical data [91], a training data set [107] or on the fly [149]. Diverse methods such as statistics based [97], rule based [148] or immunology based [48] methods can facilitate the model creation. Two of the most distinctive categories of Behavior-based NIDS are the flow- and payload-based (Figure 3.46. Their main difference is that the former relies on the analysis of communication patterns within the network, whereas the

later in the actual contents of the individual packets.

An example of model creation for behavior based NIDS applied to automotive systems is given by Rieke et al. in [126]. They combined Process Mining (PM) [98] with Complex Event Processing (CEP) [95] technologies to perform "analysis of anomalous behavior characterized by anomalies in sequences of events and not detectable in single events on a CAN bus of a modern vehicle". As they explained, anomalous behavior detection can be described as a two-step process: 1) *discovery*, where the reference model is created, and 2) *confidence checking*, where the model is used to identify deviations (i.e., incidents).

Put simply, anything that does not happen according to the normal behavior previously learned is regarded as an incident. The main advantage of the behavior based approach is its capability to detect unforeseen attacks: any attack would (in theory) change the normal behavior of a system by, for instance, accessing unusual resources or establishing connections with new machines outside of the trusted network. Compared to knowledge based NIDS, behavior based detection systems do not have the limitation of maintaining a signature database constantly up to date. Moreover they do not need any specific knowledge about the underlying system they protect. Finally they can also detect "abuse of privilege" types of attacks, in which a legitimate user does not actually exploit any security vulnerability but accesses restricted file [37]. By opposition to knowledge based, behavior based NIDS might be *complete*, but unfortunately not *accurate*: if it can detect new attacks, it comes with the price of (high) false positives. It is very challenging to build accurate profiles because systems' activities are usually quite complexe. Covering all legitimate operations during the learning phase is not trivial, especially in dynamic environments. Therefore behavior based NIDS are prone to produce many false alarms due to legitimate unanticipated activities deviating from the model [130].

Despite these limitations, the behavior based approach could overcome the drawbacks of knowledge based systems aformentioned. In the automotive context, CAN traffic is predictable, making behavior based techniques well-suited for intrusion detection. More specifically in-vehicle communications are much more deterministic than traditional IT-desktop, since it only implies machine-to-machine communications [102]. There has been a certain number of publications focusing on the behavior of cars' network to detect intrusions. We will now detail the ones relevant for our study.

As observed by Muter and Groll in [110], messages can be sent cyclically in the automotive network with fixed intervals or on-demand (e.g., the driver activates a function by pressing a button). They suggested that since most packets arrive at a strict frequency, performing an injection attack will likely modify the expected arrival times of certain packets. More precisely, a spoofed message should be sent 20 to 100 times faster than the legitimate one to make the target ECU accept the message [102]. In order to detect these attacks, several papers have been published leveraging this idea. Taylor et al. proposed a frequency based anomaly detector which measures inter-packet timing over a sliding window and then compares current and historical packet timing [79]. Any deviations from historical averages will be flagged as anomaly. In the same fashion Gmiden et al. proposed a simple intrusion detection method for CAN network based on the analysis of time intervals of messages [50]. Similarly Song et al. extended the same concept by also including a Denial of Service (DOS) detection feature[136]. Finally Moore et al. modelled inter-signal arrival times by first observing few seconds of CAN data (i.e., learning phase) and then detect deviations from the expected arrival time.

One limitation to these approaches is that once an incident has been identified by a NIDS, there is no way to know where the offence comes from. Due to the broadcasting nature of CAN it is hard to know which ECU is responsible (i.e., has been compromised) by simply looking at the packets on the network. To overcome this issue, Cho and Shin developed a *Clock based NIDS* (CIDS) [27] which not only uses a frequency based approach to detect attacks like described above, but also fingerprints ECUs to facilitate root-cause analysis. The idea behind CIDS is to "monitor the intervals of (commonly seen) periodic in-vehicle messages, and then exploits them to estimate the clock skews of their transmitters which are then used to fingerprint the transmitters".

Regarding machine learning algorithms, Taylor et al. proposed an anomaly detector based on a Long Short-Term Memory (LSTM) recurrent neural network (RNN) to detect CAN bus attacks [144].

The idea is to train the neural network to predict the next packet data values. Any errors in prediction are regarded as anomalies spotted in the sequence. One key benefit of this approach is that it requires no domain knowledge of the system it is modelling, making it suitable for many different vehicles, regardless the vendor or model, without substantial modifications.

Mutter et al. introduced the concept of information-theoretic detection approach to the area of in-vehicle networks [109]. They noticed that automotive network traffic is not only deterministic but much more restricted than traditional IT traffic: every packet in a CAN network and its data content need to respect the CAN protocol specifications, defining for example the allowed packet payloads in term of permitted value range and packet function. With this characteristics in mind, they developed an entropy based anomaly detection system. Entropy can be described as a "measure of how much coincidence a given data-set contains. The more coincidence it comprises, the higher the entropy it contains."[109]. The restriction of the CAN protocol makes it suitable for such approach. The authors measured the entropy in an automotive network under normal circumstances (i.e., without undergoing attack) and used this value as norm. Then by continuously measuring the entropy, injection attacks can be detected since it would reduce the entropy value.

The main challenge with behavior based anomaly detection is the initial creation of the reference model. As stated previously the quality of the model has a direct impact on the detection rate. Instead of trying to develop it empirically as discussed above, we can make use of the restricted specifications of the CAN protocol. Let us discuss further this behavior-specification based approach.

Behavior-specification based: Another approach to intrusion detection is referred as behavior-specification based (or "anomaly-specification" based), as they rely on a model of a system's normal behavior manually defined according to its specifications. The main benefit of behavior-specification based approach is its capability to distinguish deviations that are legitimate from the malicious ones. A legitimate deviation could be caused by an action which never occured before during normal operations. During the learning phase of the behavior based approach mentioned in the previous section, it is difficult to include such rare/unfrequent events in the normal model, leading to false positives. However with the behavior-specification based approach, unforeseen legitimate events would not be detected as incident. The strength of this approach lies in the specifications: if correctly developed, "the false positive rate can be comparable to that of knowledge based detection" [147].

This category of NIDS considers the semantics of the message payload and checks whether their data content is realistic according to the protocol specifications. For example, for any given CAN-ID present on the network, we know already according to the communication specifications what are the type of data expected as payload. Therefore we can define in advance how does a well-formed packet look like, what is the fields sizes, the expected data range, how frequent a packet should be sent, in what subnet it should only be seen, and so on. When an infeasible correlation with previous values or other messages occurs, we can tell with high confidence that an incident is occuring. Let us consider an example in which an attacker would like to confuse a driver by modifying the speed displayed on the speedometer by switching instantely from 30 to 200 km/h. If we consider that the attacker is capable of injecting a packet containing the spoofed speed value within the right time frame, behavior based NIDS relying on the frequency of packets as described previously would not be able to detect the attack. However with the correct behavior-specifications established, the detection system would know that, according to the system specifications, it is physically infeasible for that vehicle to reach abruptly this speed.

Behavior-specification based NIDS sound extremely promising. However it is not that discussed in the literature. Larson et al. in [89] have illustrated a method for creating security specifications for communication and ECU behavior. The approach uses a high-level protocol for CAN, CANopen [25], which provides a communication- and application-specific configuration in order to detect the traffic which does not fit the protocol-level security specifications and ECU-behavior security specifications. On a higher level, Muter et al. also proposed an approach to behavior-specification by introducing a set of anomaly detection sensors which allow the recognition of attacks during the operation of the

vehicle without causing false positives [110]. They left the sensors implementation for future work.

The benefits promised by behavior-specification are undoubtedly very attractive. In addition it looks like the automotive networks protocols (at least CAN) are well suited for this approach. However the main hurdle lies in the development of the model based on the specifications. Since no NIDS approach seems to smoothly fit into the automotive context, an idea would be to combine them together into an hybrid NIDS.

Hybrid NIDS: this type of IDS corresponds to the combination of different approaches discussed above. For instance Miller and Valasek implemented such an hybrid NIDS in [102] by combining a knowledge and behavior based IDS. As they noted, all known CAN injection attacks rely on either CAN diagnostic messages or standard messages sent with a highly inflated rate. In a knowledge based approach they implemented a detection module looking for illegitimate diagnostic messages. The second module they developed corresponds to a behavior based detection scheme applying a frequency based approach, similar to the ones discussed previously. They designed a small device to be plugged into the OBD-II port of a car. It starts first by learning traffic patterns, and then detects anomalies. Upon detection, the device would take preventive measures by short circuiting the CAN bus, thus disabling all CAN messages.

Ujiie et al. [146] proposed an hybrid NIDS to protect CAN networks. The idea is to add a new ECU dubbed "centralized monitoring and interceptor ECU" (CMI-ECU) which detects malicious CAN messages and prevents them from being received by other ECUs. By implementing the CMI-ECU on a gateway ECU, the NIDS will be able to monitor all in-vehicle CAN buses (depending on the car architecture). The CMI-ECU implements two types of algorithms, namely "pattern matching" (i.e., knowledge based) and behavior based algorithms. As they accurately observed, "using only a single algorithm is not enough when considering the need to be able to detect various types of malicious CAN messages". For pattern matching, two algorithms are used:

- **CAN-ID** looking for messages with a CAN-ID not listed on a whitelist of legitimate IDs, and
- **Fixed payload** looking for messages with invalid (not on whitelist) payload and Data Length Code (DLC).

Regarding the behavior based algorithms, three are implemented. First of all, **Cycle** will look for cyclic CAN messages sent outside of normal cycles. Then **Frequency** will focus on non-cyclic messages sent at an abnormal frequency. Finally **Variable payload**, which will be looking for messages containing variables in the payload that do not match statistical values.

The hybrid detection approach sounds very powerfull. By combining several techniques and algorithms together, one can leverage the strengths and advantages of knowledge, behavior and behavior-specification based methodologies, while remediating their weaknesses. Although one has to keep in mind that the processing power available on ECUs is limited. Developing an NIDS combining too many detection features could have an unacceptable impact on the real-time data processing of the devices on the automotive network.

Now that we explored and discussed the in-vehicle network NIDSs proposed in the literature, in the following section we will focus on performing a gap analysis in order to identify directions for future work.

Discussion

Network monitoring and intrusion detection have been investigated by researchers for some years. The field is still relatively new and some improvements need to be done. After reviewing the current literature related to in-vehicle NIDS, we identified some questions we would have to answer to give us directions for further work and research. First of all the main question could be "what is currently missing?". In the following part we try to categorize the research directions that should be followed in the scope of NIDS.

Evaluation criteria

According to the classification of the previous section we accordingly try to classify in Table 3.19 the current attack classes that are described in the literature along with the IDS used to detect them. The IDS detection method is organized according to Section 3.5.2.

Attack class	Description	Associated IDS categories
Denial Of Service [87] [116]	Network flooding with certain frame ID	Behavior-based:[136], [109] Hybrid: [146], [102]
Frame injection [101]	Inserting known/unknown frame ID	Behavior-specification: [136] Behavior-based: [109], [155], [27]
Diagnostic messages [87]	Extract diagnostic info from ECUs allowing to reprogram them	Knowledge-based: [102]
Masquerade / impersonation attack [4]	ECU spoofing or frame replay	Behavior-based: [27]
Frames dropping (e.g. blackhole attack) [16]	Preventing legitimate frames from being received by all the nodes	Behavior-based: [27]

Table 3.19.: Existing attacks and their detection by existing IDS

Open challenges

Detection based on limited in-vehicle protocols: It seems that most efforts had focused on the CAN protocol. However as detailed previously, the other protocols such as LIN, MOST, FlexRay or even automotive Ethernet should be equally investigated, both from the attack and the defense perspectives. Quite some attacks on CAN have been published and at the same time researchers proposed different approaches to defend the bus as we just saw. Nilsson et al. have looked at abusing FlexRay [113], although nobody proposed an NIDS for that protocol. Hence more work needs to be done with regards to automotive protocols and how to protect them.

Common benchmark: Regarding the NIDSs presented above, it would be interesting to have a way to assess them using a common benchmark. The results the authors respectively give in terms of detection rate need to be reproduced and confirmed. Having a standard data set with number of attacks could help us identify the NIDSs performing the best. In addition, feasability studies could be conducted to investigate the possibility of combining them together, as suggested in the "hybrid NIDS" section. As we mentioned earlier, it would be mandatory to also respect the real-time constraint of automotive systems. Lightweight techniques were proposed to perform intrusion detection [136]. The in-vehicle subsystems consist of resource-constrained devices. Therefore the NIDS should not require high computing power while still performing fast and effective incident detection.

Lack of implementation: Another drawback of the detection systems above is the lack of actual implementation within a vehicle. In most of the papers found, the solution the authors proposed has been at best tested in a simulated environment. However it is hard to tell whether an implementation in a car would work, and ultimately whether it would also give the same detection results. Some researchers did test their NIDS in a vehicle, but unfortunately they do not mention anything about

the brand and model. Since every car has a different architecture [102], we could eventually see significant changes in the NIDS performance when deployed in two different vehicles.

Creation of a normal behavior baseline: Regarding the behavior based and behavior-specification based intrusion detection approaches, a point of attention should be put on the creation of the model. It is mandatory to find an efficient way to build a robust model, which would avoid generating false positives. The risk with false positives is that the driver could get tired of false alarms and ignore future alerts, rendering the NIDS useless.

Incident response: The previous open problem also rises another crucial question: how to react to an incident? Hoppe et al. proposed an approach called "adaptive dynamic reaction" which categorizes incidents into three different levels depending on their severity [63]. For non-critical incidents, a visual warning will be displayed (e.g., light on the dashboard). In the case of a critical incident, an acoustic message will be played to the driver (e.g., sound and speech similar to aircraft warnings to pilots). Finally for severe incidents, an haptic action will be taken (e.g., automatic braking). This idea, even though interesting, brings another discussion: how to rate the severity of incidents? As discussed in the previous part, it is difficult to tell whether an attack is life-threatening since it will depend on the state the car is in. An attack could have a different outcome if the vehicle is stopped, driving slow or fast. Regarding autonomous response from the NIDS, some ethical questions arise. For example, how far do we want the car to react for us? We often hear this type of discussions with autonomous vehicle. In certain countries such as Germany, it is forbidden by law for a system to take critical decisions by itself without the supervision of a human.

Network detection layer: Finally, while all the solutions found in the literature focus on detection at Layer 2 of the OSI model (Data link), it could be also valuable to investigate if detection at the physical layer would work, in order to detect for instance attacks that attempt to electronically manipulate network-transmitted data. As stated by Hoppe, "these low-level communication characteristics could be analysed by a special detection unit to identify the authentic device which has sent the current message" [61]. This idea has been already evoked few times in [146, 26]. Furthermore, by relying on CAN monitoring we can only detect threats that are already too close to the main in-vehicle components which we want to protect. So an interesting direction would be to also have solutions in higher layers than CAN, for early-stage threat detection (i.e. before it arrives to the CAN layer).

3.6. QoS Monitoring

The Quality of Service (QoS) describes the performance of a service according to diverse quality requirements that are mostly outlined from a user's point of view [75]. In contrast, customer requirements are usually presented on a more abstract level. E.g., Cisco [28] defines QoS in a direct relation to networking and managing network resources. It is mostly used in context of networks and their ability to provide VoIP. Furthermore, monitoring supervises and guides a system such that monitoring is often combined with QoS measurements to collect information on the different system activities and their performance and react consequently. Therefore, the gathered information has to be filtered and aggregated. Indicators are often derived from QoS analysis which show if all system components work properly and meet their quality requirements.

Many real time systems, e.g., within the automotive domain, demand certain qualities e.g. regarding timing behavior, correctness, safety, security, clarity, comprehensiveness, among others. Such qualities are constrained by available resources or technologies in many cases such as communication, processor frequency, memory size and every other shared resource of a system. To achieve predefined quality levels of different services, various system levels must be addressed. Since single QoS mechanisms may not be aware of their surrounding QoS approaches, system-wide monitoring can combine several independent QoS in a system-wide context. Other QoS services are already considered within early system design phases via sophisticated tooling, e.g., regarding timing behavior via Worst Case Execution Time (WCET) analysis.

3.6.1. Scope

In general, QoS monitoring is a cross cutting concern applicable to a variety of abstraction levels, components, and services. This section provides a brief summary of QoS monitoring systems.

A monitor is usually a standalone component that gathers data and eventually ensures a specific state, processing, or service. Therefore, the monitor may consist of structures that analyze or interpret recorded data while introducing as low overhead or interference to the existing system as possible. Hence, the state to which the monitor introduces bias or simply influences the behavior of a system should be avoided. As an example, a monitor can be placed into a vehicle and connected to the available networks, e.g., CAN, LIN, MOST, etc. or even a gateway component. In order to keep its influence as low as possible, the monitor often filters and aggregates live information. Within the APPSTALCE project, such aggregated information can be transmitted to the cloud in order to achieve sophisticated analysis of vehicle related software services such as scheduling, task response times or timing in general, communication, I/O executions, memory performance and more. In terms of AUTOSAR, a service is defined as a basic sofware (BSW) available to any software component (SWC) via the standardized interfaces related to (a) the operating system, (b) memory, and (c) communication. Since APPSTACLE will not consider detailed AUTOSAR development activities, the developed QoS measurements will be prototypical and AUTOSAR-independent.

An APPSTACLE QoS-Monitor component could consider four layers: (1) The ECUs at the bottom layer that are interconnected to a (3) gateway via different in-vehicle (2) communication buses such as CAN, LIN, MOST, FlexRay or Ethernet. Beyond this, the (4) communication stack including the mobile network towards the cloud builds up the last layer. Collected measurements can further comply to different safety levels related to the processes they are involved with. Measured values can also directly relate to the bus system performances, e.g., to assess throughput, missed packages, number of applied error corrections, number of delayed messages, average delay time of messages, and more. Furthermore, application status can be assessed such as the amount of warnings, errors, time spent in different states, etc. Finally, the network layer parameters like bandwidth, collisions, connections (correct/not/wrong/doubled/slow/ established connections, early connection loss), non delivered packets, packet errors, jitter and delay could be monitored.

3.6.2. Overview

While searching for relevant technologies and projects, no open-source approaches were found, which were totally compliant to the description of a monitor in the scope. Nevertheless, projects and technologies are proposed in this section that may contribute in different ways to the APPSTACLE QoS-monitoring.

For monitoring development and maintenance activities in the automotive domain, mostly commercial products, e.g. from Vector Informatik GmbH [13], are used. The tool *CANalyzer*[14] can monitor CAN, FlexRay, Ethernet, LIN and MOST buses. *CANoe*[15] extends this further with the support for simulation and test tools [152]. The various features of these products include

- Analyzing single ECUs or even whole networks
- Operating directly in the vehicle networks
- Configuring several channels in each network
- Working with previous recorded data
- Using bus simulation for testing
- Use of several database types: DBC, LDF, FIBEX, Function Catalogs, Autosar System Description, Car2x databases
- Visualization with a GUI
- Including OEM-Packages for integration of OEM specific features into the simulation
- Stimulating ECUs
- Use of KWP2000 and UDS for diagnostics
- Integration of an OBD-II tester

In order to connect networks to the workstation running CANoe or CANalyzer, proprietary hardware from Vector is recommended [151]. The software is a commercial product, hence closed source, and is more focused on maintenance or in-house diagnosis activities apart from regular vehicle operation.

An open-source approach, similar to the previous described commercial one, is BUSMASTER [16]. It is a joint project developed by RBEI[17] and ETAS Group [18]. The software can be used for monitoring, analyzing and simulating CAN and LIN messages. It features[128]:

- Support of CAN 2.0A, 2.0B and LIN
- Analyze data bytes in raw or logical data format
- Separate monitoring of physical data and signals
- Logging for off-line analysis
- Use of a message database (DBC, LDF)
- Support of USB CAN hardware with different controllers
  - ETAS BOA/ES581.3/ES581.4/ISOLAR-EVE
  - i-VIEW
  - InterprediCS neoVI
  - IXXAT VCI
  - Kvaser CAN
  - MHS Tiny-CAN
  - NSI CAN-API
  - PEAK USB
  - Vector XL
  - VScom CAN-API
- Network statistics
  - number of Standard, Extended, RTR and Error messages transmitted and received

[13] https://vector.com
[14] https://vector.com/vi_canalyzer_de.html
[15] https://vector.com/vi_canoe_de.html
[16] https://www.etas.com/en/products/applications_busmaster.php
[17] http://www.bosch-india-software.com/en/homepage/rbei_homepage.html
[18] https://www.etas.com/en/etas_group.php

  - Network load as Bus traffic
- Use of an API exposed by BUSMASTER over a COM interface
- Use of a C library for Node simulation
- Use of diagnostics standards like UDS or KWP2000
- FlexRay and CAN FD as commercial add-ons

Usually, BUSMASTER is operated through its Windows based GUI which also includes editors for simulation scripts.

In addition to the previously mentioned products, some publications and concepts are available that APPSTACLE can make use of. For example, the paper "Monitoring CAN performance in distributed embedded systems" [20] describes a hardware monitoring approach. This hardware - a dsPIC33 - is directly connected to the CAN bus and also offers a serial connection to a PC. The board runs a FreeRTOS with the monitoring application witch is capable of receiving and sending messages on the CAN. It accesses the CAN bus, extracts information and sends those to the PC host application. The developed application detects communication errors, traffic amount and types, overdue responses, and bandwidth. Since the presented approach can only be accessed conceptually and its development is not open source and requires dedicated hardware, its partial applicability towards APPSTACLE needs must be carefully checked.

3.6.3. Discussion

Since open source development activities along with monitoring vehicle service qualities are quite rare, BUSMASTER [129] is probably a prior choice to monitor bus systems established in the vehicle domain, e.g. CAN. However, due to the wide application field of QoS-Monitoring in general, its utilization in APPSTACLE is restricted to the CAN bus system at first, in order to provide an easy starting point for interested parties getting involved to QoS-Monitoring for vehicles. APPSTACLE's QoS-Monitoring adaptation towards BUSMASTER can involve its automatic data transmission to a an Eclipse Hono instance and a cloud application consumer that accesses, analyzes, and visualizes the data.

3.7. Over the Air updates

The ability to update software in vehicles via a wireless connection over-the-air (OTA) is becoming increasingly important. The reasons are multifaceted: First, it makes the distribution of software updates efficient. Such software updates have become necessary because software complexity in vehicles is vastly increasing and it is necessary to react to security issues that are likely be caused by the proliferation of connectivity via different channels. Second, software updates are the basis for feature updates that allow the vehicle manufacturer to stay in touch with customers throughout the vehicle's lifetime and to meet consumer electronic device expectations based on customer experience.

Today, software updates in vehicles are mostly processed in the workshop under defined conditions and supervised by experienced personnel. Soon, various stakeholders (e.g., fleet managers, vehicle owners, or drivers) will be able to perform software updates over the air nearly anywhere without additional assistance.

Especially when installing updates, it is important to keep the update time as short as possible. In most cases during the update, the vehicle – or at least parts of its functionality – are not available to the driver. While the download phase may be performed while the vehicle is in motion, the installation phase itself normally takes place while the vehicle is parked and the engine is not running.

3.7.1. Scope

The following section describes three different types of OTA updates. In the simplest case, inactive software functionality is activated by reconfiguring the ECU without the need to modify its firmware. Afterwards it will be explained how individual, self-contained parts of the ECU firmware can be replaced in order to add or update individual functions. Finally, the update process of a complete firmware image is described.

3.7.2. Overview

Activation of software features

This section describes how individual software features can be activated without altering the firmware of the ECU. In this case, the ECU software must be supplied by the OEM with a full feature set. The available functionality within this firmware is then configured by software switches, depending on the options selected by the buyer of the vehicle. For the vehicle manufacturer or OEM, this offers the advantage that fewer different firmware versions have to be developed and kept in stock. In addition, there are no costs for re-testing and certifying different versions of the ECU software. Furthermore, this approach also makes it possible to sell or rent functions without great effort (i. e. without prior firmware update process) even after delivery of the vehicle, provided that the customer is willing to pay for the new functionality. Finally, the customer of the vehicle also benefits from this as he can postpone the purchase decision for or against individual features into the future and no longer necessarily has to make this decision when ordering the vehicle. Likewise, he can decide to rent functions for a limited period of time, if the manufacturer offers this.

Although most vehicle manufacturers still require a visit to the workshop to change the software configuration, there is a trend towards offering, selling or renting new features for a limited period of time directly in the car (e. g. via the central display) to the vehicle owner instead. In order to avoid a workshop visit and to enable the user to activate the new functionality at home or on the road, the vehicle must be connected to the Internet (e. g. via WiFi or mobile phone). The license for a new functionality can then be purchased, for example, in a proprietary "app store" of the car manufacturer Over-the-Air.

A popular example for this is the Tesla brand. Autonomous driving functions are available for both the model S and the SUV Model X, which enable automatic lane changes and offer an Adaptive Cruise Control (ACC) mode. These and other features are sold as part of the "Autopilot" option. The necessary hardware is installed in every vehicle produced from September 2014, regardless of whether

the autopilot has been ordered or not [127]. The autopilot can be ordered as described above when purchasing the vehicle. On the other hand, this functionality can also be purchased over-the-air in the vehicle's App Store. In order to make it easier for the customer to make a purchase decision, the autopilot can be tested for one month free of charge. After this month, the purchase price is due for payment if the function is still wanted. A visit to the workshop is neither necessary for the beginning of the test month nor for the permanent installation of the equipment option. It can be assumed that no change in the firmware of the ECUs involved is necessary for autopilot activation, since this alone would require up to two (possibly time-consuming) firmware flash operations to test the autopilot. On the other hand, a customer could fraudulently prevent the original, reduced functional software version from being re-installed after the end of the test month by blocking the vehicle's Internet connection and thus keep the Autopilot without charge. Therefore, it is more likely that a (possibly temporary) software certificate is used here.

In [100] Mercedes-Benz suggests that it wants to follow a similar path to Tesla. As an example, an S Class is given here which is equipped with complete hardware for all features available for this model, but the functions themselves are disabled by default. The customer can then decide on-demand which functions he needs and in which period of time he wants to use or rent them. Although Mercedes-Benz does not explicitly mention here how the functions are technically activated and deactivated, it can be assumed that this will be done over-the-air, thus saving the customer time-consuming visits to the workshop.

Audi is currently working on new *Level 3* assistance systems that no longer require the driver's full attention. For example, they mention a "Traffic Jam Pilot", that will automatically accelerate and brake on the motorway in case of a traffic jam and, if necessary, form a rescue lane. However, as the legal basis for Level 3 assistance systems has not yet been created in Germany, such functions may not yet be used. Audi, however, suggests that such functions may be sold in advance and should remain deactivated until the law is amended. It is still unclear, however, whether this function would then have to be activated over-the-air or in a workshop [137].

Update or add self-contained software features

A step further from activating features and software parts which are already present is the update of existing software features or the installation of new features. For this use case, most car manufactureres currently restrict OTA updates to telematics units and In-Vehicle-Infotainment (IVI) systems and keep back from updating safety-critical systems or, generally speaking, the real-time systems of their E/E architecture. A notable exception is Tesla, who claim to do exactly this [10]. For example, they have already used the center console to update a part of the transmision system [42]. But since these kind of OTA updates are out of the scope of this project and not yet very common, the following part deals mainly with updates of the IVI system (like the update of map material for navigation systems).

One problem when looking at the updating process of the infotainment systems for different car manufacturers is how to get the neccessary information. Most car manufacturers do not provide extensive information on how their OTA update process really works. So the subsequent short survey relies mostly on what is published on the companies marketing and support websites. If OTA software update features are missing, it does not neccessarily mean that they are not existent, but that they might just not have been found. For example, for Daimler's COMAND online system [32], no specifics about the updating process have been found.

Audi with their "Audi connect" system [6, 8] offers an online updating possibility for navigation map data [7]. Additionally to map updates, VW makes the import of rout data possible in their VW Car-Net system [153]. BMW with their ConnectedDrive system states that navigation maps are updated automatically by the system. With the BMW ConnectedDrive Store the configuration of services and apps via internet (not in car) is possible, and after that, the feature is available in the car [17]. If the new feature is installed or enabled is not elaborated.

Ford's Sync 3 system offers automatic updates via wifi. This includes updates of maps, of appli-

cations and even extends to new system software versions [47, 64]. The system can be configured, so that these updates are done automatically, e.g. when connected to the home wifi.

Volvo provides OTA capabilities for map updates, new apps and app updates, and system updates [154]. The updates are managed by a download center app (illustrated in Figure 3.47). Here the driver is informed of new software versions and can than start the installation manually. The update is not restricted to wifi (like with Sync 3) but may use any internet connection the system is capable of.

Figure 3.47.: Volvo's download app [154]

All in all, the car manufacturers seem to be on seperate stages of the three stages of OTA infotainment updates: map updates, app updates, and system updates. Almost all manufacturers offer OTA updates for maps and navigation material. This is the obvious first use case, since map material has to be updated frequently and the update is apparently without any risk (a counter example is [51]). Some manufacturers, however, already let applications and the system software be updated over-the-air.

Since software parts are actually transfered to the vehicle (instead of just enabling existing ones), security and robustness of the update process are of imminent importance. It has to be made sure that the update does not render the functionality or the whole car useless and does not allow attackers access to the car infrastructure. So for securitay matters authentication and integrity of the update file have to be assured and for robustness a failsafe update process and some kind of rollback strategy have to be established. What happens, if this is not the case, shows the example of an OTA update of the IVI and navigation system of the Lexus [51]. After the update the system crashed repeatedly due to faulty data from the traffic and weather service.

Firmware updates

This section describes how ECU firmware can be updated as a whole. Firmware-Over-The-Air (FOTA) is a method of performing software updates to vehicle ECUs remotely over the air. For the description of FOTA, the following network architecture and mechanism (see 3.48) is used as reference, as there are varieties of architectures for different vehicle manufacturers. With FOTA, the main cloud based server is capable of sending a software update to all the vehicles in the fleet that are eligible for the update. Therefore, the client software and hardware in the vehicle must support the update mechanism.

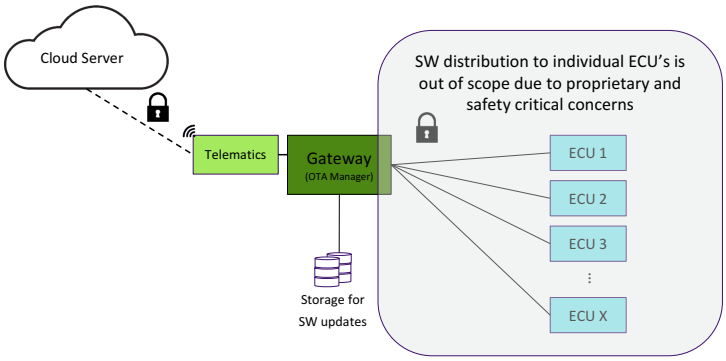

Figure 3.48.: FOTA Environment Overview

The E/E architecture of a high-end vehicle is mostly structured into different domains as shown in Figure 3.49. In this architecture, a powerful ECU – called a domain controller (DC) – is available for each domain. All DCs can be connected via a high-speed backbone network, such as automotive Ethernet. Other domain ECUs are connected to the respective DC with a domain-specific network (such as CAN or FlexRay). Although the E/E architectures vary considerably, this model is a good basis for conceptual explanations. (Remark: As APPSTACLE won't be able to handle individual E/E vehicle architectures the scope within the project will stop at DC level. Distribution within the network has to be handled by network internal update management systems.)

For the FOTA mechanisms, the software update will be defined, maintained, and initiated on a backend infrastructure. On the backend, among others, the definition of the software update packages is managed for a complete fleet of vehicles, for given vehicle platforms, and for vehicles at an individual level considering functionality and vehicle configurations. These are needed to run the management for rolling out the update. The update packages are protected with security signatures and CRCs and made available for download on the server (DLS).

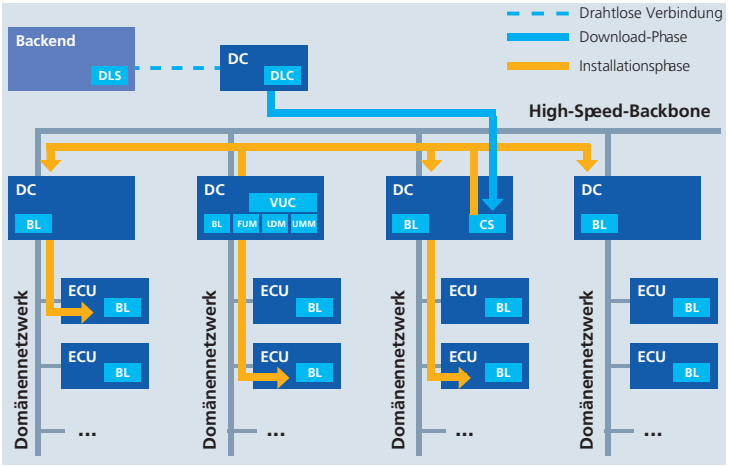

Figure 3.49.: Overview of ECU network in high-end vehicles [43]

Update packages on the backend are available based on "software baselines" for the complete vehicle. This is understood as a released set of firmware software versions for all ECUs in a vehicle that allows for safe and compatible operation. On a technical level, the release includes the validity for specified vehicle configurations, such as hardware, software, sensors, and actuators, and validity between the firmware versions for the different ECUs themselves. This is typically specified and released by the OEM. The update package contains the information and firmware necessary for updating all affected ECUs in order to reach the state from one valid baseline to another valid baseline.

On the vehicle side, update packages are received by the download client (DLC) running on a connectivity unit or Telematics Communication Unit (TCU) that can interact with the backend via a wireless connection. After downloading, update packages are stored in a content storage unit (CS), which might be located on one of the domain controllers (DC), e.g. the head unit.

After an update package is completely downloaded and includes the correct dependencies and has been checked for security signatures and CRC, the DLC hands over the update package to the vehicle update manager (VUM). With that, the "download phase" is complete. The download phase from the backend to the CS is depicted in a dark color in Figure 3.49.

In the subsequent "installation phase", the VUM is responsible for organizing the preparation at vehicle level, and also distributing and installing the download to the affected ECUs in multiple domains. Preparation is defined as checking, achieving, and maintaining the necessary preconditions for distribution and installation. Distribution refers to either normal or fast distribution within the vehicle and across different domains, such as the power train or chassis, and over different busses. The distribution of the update packages is handled by an update distribution manager (UDM) within the VUM.

Downloading and installing update packages are only possible in dedicated modes known as "update modes" and are dependent on certain conditions, such as power supply and the vehicle's operational state. Such kind of mechanism is described with the following example methods of performing software updates to vehicle ECUs remotely over the air. The Figure 1 illustrates in general how a distribution can be performed. With FOTA, the main cloud based server is capable of sending a software update to all the vehicles in the fleet that are eligible for the update. Therefore, the client software and hardware in the vehicle must support the update mechanism.

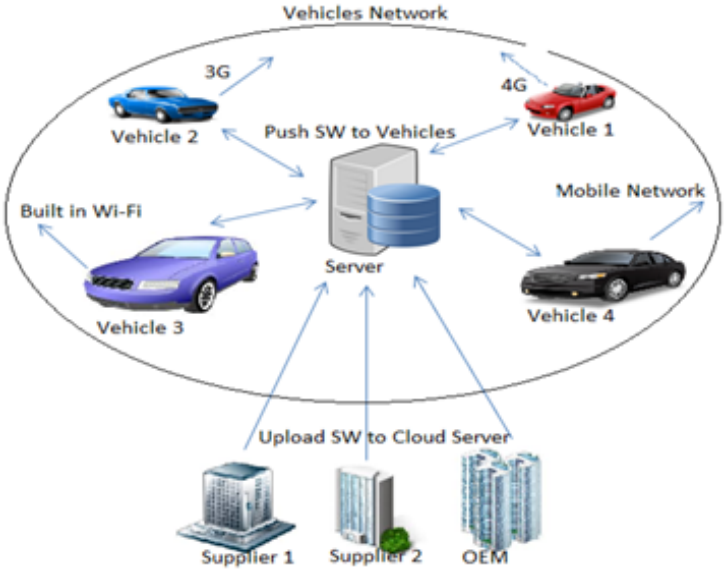

Figure 3.50.: Priciple sketch of distribution [114]

As the functionality for FOTA is still in its early phase for automotive there are just some basic illustrations of how a update workflow can be initiated. In figure 2 the workflow is illustrated and partly used within current vehicles depending on the needs, security and safety aspects within the vehicle.

The current vehicle infrastructure and safety regulations prevent updating firmware as most of the ECUs within the system are blocked to the outside to prevent outside attacks and legal conflicts.

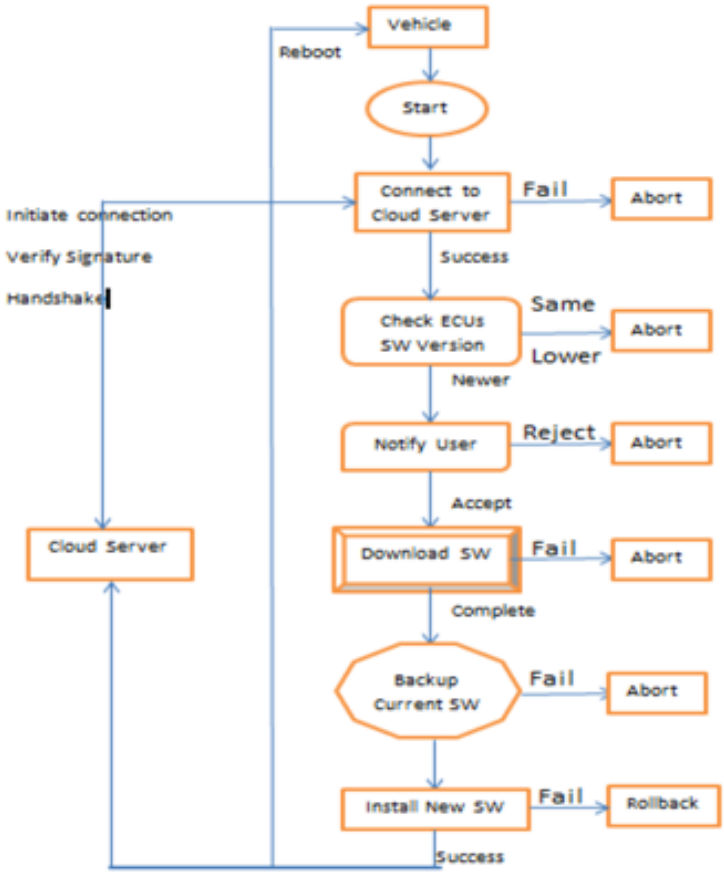

Figure 3.51.: Overview of software update [114]

3.7.3. Discussion

Currently the OTA updates on vehicles have been performed for non-safety specific ECUs like Multimedia, Navigation etc. For the safety specific ECUs like Engine Control, ABS, Airbag etc. FOTA is still not being performed by the OEMs because of the safety aspects involved. The updates on safety specific ECUs are performed by an authorized person under safe environment.

There are two challenges facing OTA updates on safety critical ECUs. First, car owners will not tolerate vehicle downtime for updates. Therefore, they must take place seamlessly and invisibly in the background. Additionally, the ability to remotely update vehicle firmware introduces a new attack vector for hackers. The two main motivations for hackers will be to use the OTA mechanism to reprogramme critical ECUs to ultimately take control of the vehicle or to steal OEM firmware.

To prevent reprogramming, the authenticity and integrity of the firmware needs to be protected. This ensures that the firmware originates from a trusted source and that it has not been modified. There are several methods of implementing such protection (e.g. HMAC, CMAC, or a digital signature).

Checking the authenticity of firmware guarantees that only trusted firmware is used in an update process but a hacker may still be able to read the plaintext (i.e. source binary) firmware to reverse engineer the source code, or steal data, which may lead to IP theft and/or privacy issues. Encryption of the firmware (for example, using AES) can prevent this. With integrity checking, the communication channel over which the firmware images are received will be protected to prevent the common "man-in-the-middle" attack.

Finally, the in-vehicle OTA update manager should be protected against manipulation [69].

In summary, the following countermeasures should be applied:

- Protect the authenticity and integrity of firmware to prevent a hacker from running modified code
- Encrypt the firmware to prevent a hacker from accessing code (IP) and data
- Establish secure end-to-end communication between the vehicle and OEM servers to prevent man-in-the-middle attacks
- Ensure a secure, trusted location for the OTA Manager application

Part II.

Specification

4. Introduction

4.1. Scope

This chapter specifies the APPSTACLE in-vehicle platform in the form of a requirements specification. Hence, it defines the requirements of the different elements of the APPSTACLE in-vehicle platform as well as its overall behavior and structure.
The target platform for this specification is exclusively the APPSTACLE in-vehicle platform. Use of this specification for other platforms is not not prohibited but is not covered in the design or content.

4.2. Terminology

Throughout this specification, the word usage for indicating different conformance levels is in line with the corresponding definitions given section 10.1 of the 2014 IEEE Standards Style Manual [66]. In particular, the following key words are used in order to signify the requirements of this specification:

- **Shall**
  The word shall indicates that a certain requirement or behavior is mandatory and has to be strictly followed in order to conform to the specification. Thus, no deviation from such a requirement or behavior is permitted.
- **Should**
  The word should is used to indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain course of action is deprecated but not prohibited. The key word should thus may be used to point out that a certain requirement or course of action may lead to a better performance or system behavior, for example, without excluding other alternatives.
- **May**
  The word may is used to indicate that a certain course of action is permissible within the limits of the specification.
- **Can**
  The word can is used for statements of possibility and capability, whether material, physical, or causal.

4.3. Definitions and Glossary

For the purpose of this specification, the following definitions apply. For terms not explicitly defined in this clause, it should be referred to [21].

APPSTACLE in-vehicle platform. The APPSTACLE in-vehicle platform is a combination of software and hardware components that can be integrated into a vehicle in order to (I) host applications, (II) allow access to vehicular data and functions and (III) set up a connection to a server instance in the cloud.

App. An App is an encapsulated software component that can be downloaded to, installed on, executed by and uninstalled from a dedicated application runtime environment.

5. APPSTACLE in-vehicle platform specification

5.1. Platform and App Runtime

5.1.1. Scope

This section defines the requirements regarding the Software Platform the APPSTACLE in-vehicle platform is based on as well as the App Runtime integrated to host additional functionality in the form of an App. As the APPSTACLE project team has decided to use Automotive Grade Linux (AGL) as a base operating system this chapter only points out the additions APPSTACLE will provide.

5.1.2. Requirements

Platform Updates

R_PA1.001 The Software Platform should secure such remote software updates through authentication and validation measures.

R_PA1.002 The Software Platform may implement platform diagnostics or error logging functionality.

R_PA1.003 The Software Platform shall be updated in an atomic manner, meaning as a single transaction and without intermediate state.

App Runtime

R_PA2.001 The App Runtime shall allow to install, uninstall, update, execute and kill Apps.

R_PA2.002 The App Runtime shall implement authentication and validation measures when installing an App.

R_PA2.003 The App Runtime should implement a permission management that is transparent and controllable by the driver or the owner of the vehicle.

R_PA2.004 The App Runtime shall isolate Apps in order to prevent interference amongst Apps or between an App and Software Platform.

R_PA2.005 The App Runtime may implement runtime diagnostics or error logging functionality.

R_PA2.006 The App Runtime may implement measures to protect the know-how and intellectual property of the App developers.

R_PA2.007 The APPSTACLE project shall provide a development environment that supports App developers in their task.

R_PA2.008 The APPSTACLE platform should provide a test environment for applications.

R_PA2.009 The App Runtime should provide an interface to connect to a cloud-based App Store.

R_PA2.010 The APPSTACLE Runtime should provide a garbage collection for applications.

R_PA2.011 The APPSTACLE Runtime should support crash and recovery management functionality in order to recover an application in the case of a crash and handle corruptions regarding runtime variables.

R_PA2.012 The APPSTACLE Runtime environment should allow applications to run in the background.

5.2. APPSTACLE API

5.2.1. Scope

This section defines the overall characteristics of the APPSTACLE API.

5.2.2. Requirements

Accessibility

R_AP1.001 The APPSTACLE API shall be accessible by all Apps installed on the APPSTACLE in-vehicle platform.

R_AP1.002 The APPSTACLE API may by accessible by other software components of the APPSTACLE in-vehicle platform.

R_AP1.003 The APPSTACLE API may by accessible by Apps running on a cloud server connected to the APPSTACLE in-vehicle platform.

R_AP1.004 The APPSTACLE API may provide its content using different access technologies and may support different programming languages.

Abstraction

R_AP2.001 The APPSTACLE API shall provide harmonized and abstracted access to dedicated resources of the vehicle. This includes an abstraction from the actual network topology and architecture of the vehicle.

R_AP2.002 The APPSTACLE API should allow to develop Apps independently from the vehicle manufacturer, makes or models.

Customization and Content

R_AP3.001 The APPSTACLE API shall define or adopt a common set of data.

R_AP3.002 The APPSTACLE API should allow to customize the extent of the resources available by adding or removing specific resources or resource groups.

R_AP3.003 The APPSTACLE API should integrate the content of APIs that are either standardized or publicly available. [1]

R_AP3.004 The resources available within the actual instance of the APPSTACLE API should be discoverable by the Apps accessing it.

R_AP3.005 The resources available within the actual instance of the APPSTACLE API may be browsable by the App accessing it.

R_AP3.006 The APPSTACLE API may provide historical data.

Security

R_AP4.001 The APPSTACLE API shall establish a roles and rights concept to gain or prevent access to resources.

[1] e.g. the interfaces that are defined by GENIVI [78] or W3C [159] or standards like OBD [138] or Extended Vehicle Remote Diagnostic Support [141]

R_AP4.002 The APPSTACLE API shall provide adequate mechanisms to prevent denial of service attacks on the interface itself.

R_AP4.003 The APPSTACLE API should provide mechanisms to hide API interaction from other Apps.

Behaviour

R_AP5.001 The APPSTACLE API shall respond in an appropriate way if a request by an App cannot be fulfilled.

Documentation and Versioning

R_AP6.001 The APPSTACLE API shall be documented in a human readable form.

R_AP6.002 The APPSTACLE API shall implement a versioning schema based on major versions, minor versions and patches were a major version increment indicates an incompatibility at the API level, a minor version increment indicates an extension of the API without compatibility issues and a patch indicates a bug fix without any compatibility issues.

R_AP6.003 New versions of the APPSTACLE API should be downwards compatible.

In-vehicle communication

R_AP7.001 The APPSTACLE API may provide the possibility to determine the available in-vehicle communication channels.

R_AP7.002 The APPSTACLE API shall provide the possibility to directly send and receive CAN messages without the use of higher layer protocols.

R_AP7.003 The APPSTACLE API shall provide the possibility to directly send and receive IP packages without the use of higher layer protocols.

5.3. Application IDS

5.3.1. Scope

This section defines the overall characteristics of the application Intrusion Detection System.

5.3.2. Requirements

General

R_AI1.001 The application IDS shall be able to monitor all deployed and running Apps.

R_AI1.002 The application IDS should not expose information to Apps that do not have extended privileges.

R_AI1.003 The application IDS may be able to block all monitored events such that processing continues after the events are identified as being benign.

Architecture

R_AI2.001 The application IDS shall consist of loosely coupled components.

R_AI2.002 The application IDS should be extensible with respect to monitoring agents.

R_AI2.003 The application IDS should be extensible with respect to analysis components.

R_AI2.004 The application IDS should be extensible with respect to active response components.

R_AI2.005 The application IDS may be extensible with respect to pre-processing components.

R_AI2.006 Analysis components of the application IDS may be deployed on the APPSTACLE IoT cloud platform.

Monitoring

R_AI3.001 The application IDS shall provide data that is referable to the App it originates from.

R_AI3.002 The application IDS should be able to monitor system calls.

R_AI3.003 The application IDS should be able to monitor library calls.

R_AI3.004 The application IDS should be able to monitor inter-process communication.

R_AI3.005 The application IDS should provide an interface such that instrumented Apps can report status information.

Analysis

R_AI3.001 The application IDS should provide the possibility to execute multiple analyses for each App.

R_AI3.002 The application IDS should provide the possibility to execute different analyses for different Apps.

R_AI3.003 The application IDS should provide the possibility to execute the same analysis for different Apps.

Response

R_AI3.001 The application IDS shall store alerts.

R_AI3.002 The application IDS should store events that produced alerts.

R_AI3.003 The application IDS should be able to send alerts to an authority (OEM, App company, marketplace, etc.).

R_AI3.004 The application IDS should be able to inform the driver of an alert.

R_AI3.005 The application IDS may provide active response mechanisms.

5.4. Network IDS

5.4.1. Scope

The available NIDS solutions of Section 3.5.2 can be used for monitoring and early-stage detection in in-vehicle environments when deployed in different components of the vehicle to allow traffic visibility. Such components include the On-Board Unit (OBU), the OBD-II port as well as the central gateway or network switch that certain vehicles feature [2]. Even though each deployment will introduce a different setup and configuration, the APPSTACLE framework and its interfaces to the NIDS are modular and are illustrated in Figure 5.1.

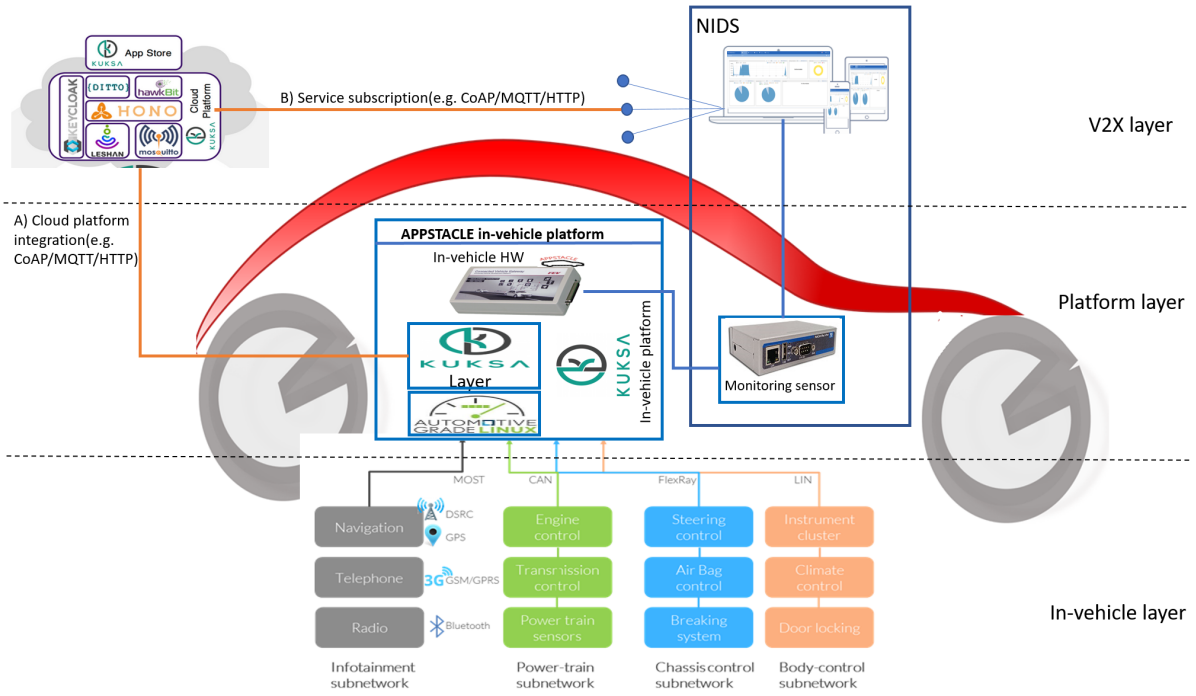

Figure 5.1.: Interactions of NIDS with the APPSTACLE in-vehicle platform

This Figure presents the three main communication layers that are usually encountered in in-vehicle environments and are considered as a part of the APPSTACLE project. In particular, the in-vehicle layer includes all the subnetworks that are required for the correct functionality of the vehicle and have a direct interface with the vehicle electronics, such as the body-control, chassis, powertrain and infotainment subnetwork. Accordingly, the platform layer features the APPSTACLE in-vehicle platform as well as its connections to the NIDS through different interfaces, such as the connection between the hardware and the monitoring sensor of the NIDS. Through this connection the NIDS is operating as a passive solution that is able to achieve real-time monitoring of the communications occurring inside the vehicle and detect potential security threats or misconfigurations. In the same time it is capable of minimizing overall impact in terms of performance on the overall system. The NIDS is also connected to a Web server or an application running on an embedded device (e.g. smartphone/tablet) that provides a user-friendly environment for the representation of the monitored environment and the detected threats or misconfigurations on it. This environment is provided as a service to the APPSTACLE Cloud platform that resides on the V2X layer, namely the layer that represents all the ex-vehicle interactions. The APPSTACLE Cloud platform can also receive monitored in-vehicle communication events directly from the APPSTACLE in-vehicle platform through the Eclipse Kuksa layer [3].

[2] https://standards.ieee.org/events/automotive/2017/d1-05_ziehensack_smart_ethernet_switch_architecture_v1.1.pdf
[3] https://projects.eclipse.org/proposals/eclipse-kuksa

5.4.2. Requirements

Information collection

R_NI1.001 The APPSTACLE in-vehicle platform SHALL support network monitoring.
The NIDS needs to have access to the in-vehicle network data, for example through a span port on the APPSTACLE in-vehicle platform, or the NIDS could be integrated in the APPSTACLE in-vehicle platform itself.

R_NI1.002 The NIDS MAY have access to unencrypted data for analysis
To achieve that it should become a trusted component, hence have the key for encrypted traffic in the APPSTACLE in-vehicle platform. In case this is not desirable, then the NIDS will not be able to provide in-depth monitoring in case of encrypted traffic. In such case it can only perform analysis context-agnostic analysis (e.g. message frequency, timing or header analysis)

R_NI1.003 All the network traffic observed/available to the APPSTACLE in-vehicle platform SHALL be forwarded/mirrored to the NIDS.
The NIDS analysis is restricted to the provided messages. To perform analysis the NIDS requires full content capture of the messages that are exchanged in the APPSTACLE in-vehicle platform.

Information Analysis

R_NI2.001 The NIDS SHOULD analyze network data against known malicious patterns

R_NI2.002 The NIDS SHOULD be able to receive regular security updates containing known malicious patterns in a timely fashion

R_NI2.003 The NIDS' updates MAY be Over The Air (OTA) updates in order to guarantee that the NIDS is up to date and possesses the latest malicious patterns/signatures

R_NI2.004 The NIDS SHOULD analyze network data to detect anomalous behavior.
The detection will be based on different classes of NIDS that were presented in Section 3.5.2. In order to select which detection classes will be selected and implemented, an evaluation still has to be conducted.

Result Dissemination

R_NI3.001 The NIDS SHALL log detected events/alerts for later evaluation.

R_NI3.002 The events/alerts SHOULD be obtained from the APPSTACLE platform

R_NI3.003 The NIDS MAY provide live alert feeds to the required stakeholder.
The stakeholders generally belong to different categories, such as transport network authorities, car owners, drivers or even System Operations Center (SOC). Each of these authorities should have different access rights on the alert feeds, which will be forwarded by the NIDS.

Security

R_NI4.001 The NIDS SHALL not interrupt normal network operation. I.e. network performance should not be impacted.
Network monitoring is meant to be a passive, non-invasive security solution. As such it shall not take any actions when abnormal behavior is identified nor have an impact on the network performance, but only generate event logs/alerts

R_NI4.002 The APPSTACLE in-vehicle platform SHOULD prevent the network monitoring solution from impacting the normal operation.
As NIDS is a part of the APPSTACLE platform, it can become an attack surface itself. Therefore, it is essential that Compromising the NIDS itself should not provide an attacker a foothold to attack the rest of the car architecture. The suggested location to enforce this is in the APPSTACLE in-vehicle platform (firewall). For an external solution this relates to restricting the monitoring to reading data. If the NIDS is

integrated in the APPSTACLE in-vehicle platform this also involves process separation and resource handling.

5.5. Ex-vehicle Connectivity

5.5.1. Scope

Intelligent Traffic Systems (ITS) will typically combine the 802.11p communication channel with the cellular communication channel. Which channel or channels to use will depend on the application and its communication requirements. The conceptual architecture of the communication channels in an ITS is presented in Figure 5.2.

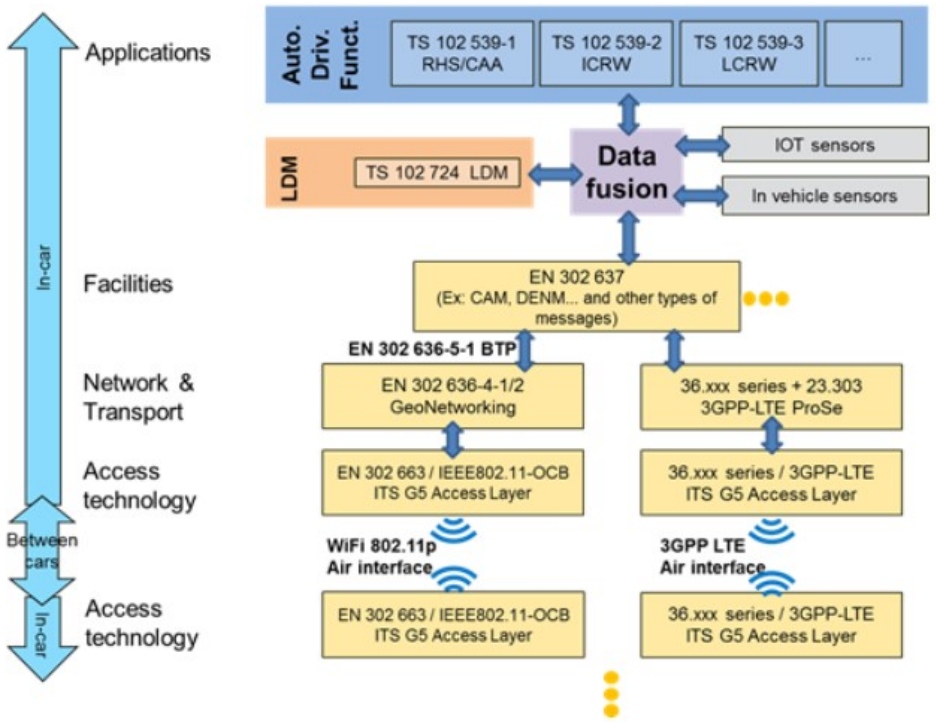

Figure 5.2.: Conceptual architecture of V2X communication channels in ITS

Both communication channels are hidden from the applications by using the facilities layer. This layer provides a common top-level interface that can be used to send messages both through the 802.11p communication as well as the cellular communication channel.

5.5.2. Requirements

Communication of ETSI messages

R_EC1.001 The Ex-vehicle communication subsystem shall be able to communicate ETSI CAM messages through all the channels that support ETSI CAM message communication.

R_EC1.002 The Ex-vehicle communication subsystem shall be able to communicate ETSI DENM messages through all the channels that support ETSI DENM message communication.

R_EC1.003 The Ex-vehicle communication subsystem shall be able to communicate ETSI MAP messages through all the channels that support ETSI MAP message communication.

R_EC1.004 The Ex-vehicle communication subsystem shall be able to communicate ETSI SPAT messages through all the channels that support ETSI SPAT message communication.

R_EC1.005 The Ex-vehicle communication subsystem shall be able to communicate ETSI IVI messages through all the channels that support ETSI IVI message communication.

Communication of IP messages

R_EC2.001 The Ex-vehicle communication subsystem shall be able to communicate TCP messages through all the channels that support TCP message communication.

R_EC2.002 The Ex-vehicle communication subsystem shall be able to communicate UDP messages through all the channels that support UDP message communication.

Configuration publication

R_EC3.001 The Ex-vehicle communication subsystem shall be able to list all the channels that are running within the subsystem (e.g. 802.11p, cellular).

R_EC3.002 The Ex-vehicle communication subsystem shall be able to list, per channel, the ETSI message types that can be communicated through this channel.

R_EC3.003 The Ex-vehicle communication subsystem shall be able to list, per channel, the IP message types that can be communicated through this channel.

5.6. QoS Monitoring

5.6.1. Scope

This section defines the overall characteristics of APPSTACLE's in vehicle QoS-Monitoring.

5.6.2. Requirements

QoS monitoring for CAN messages

R_QM1.001 QoS-Monitoring **may** measure CAN messaging performance via, e.g., measurements for traffic throughput, number of erroneous messages, delayed messages, etc.

R_QM1.002 In order to perform CAN message measurements, APPSTACLE **may** use a RPI CAN shield comprising a MCP2515 CAN-Bus Controller and a MCP2551 CAN-Bus Transceiver to connect to an existing CAN network.

R_QM1.003 For CAN message quality analyses, BUSMASTER [129] **can** be used.

Other QoS attributes

R_QM2.001 QoS-Monitoring **may** assess software processes. Therefore, APPSTACLE **should** access QoS attributes written into dedicated QoS files by the respective software processes.

R_QM2.002 QoS-Monitoring **may** access further networks such as LIN, MOST, or Flex-Ray.

R_QM2.003 QoS-Monitoring **may** provide collecting available vehicle QoS measurements as an extension to the APPSTACLE API (see Section 5.2).

5.7. Over the Air updates

5.7.1. Scope

This section defines the overall characteristics of Over the Air updates. Update mechanisms for APPSTACLE Apps and the gateway will be implemented directly in the APPSTACLE platform, while for OEM-specific ECU updates only extension points will be specified.

5.7.2. Requirements

General

R_OU1.001 The APPSTACLE In-Car-Platform shall provide a mechanism for *software updates*.
Rationale: The update mechanism is needed to install bug fixes, security patches and new software features.

R_OU1.002 The software update mechanism shall enable Over the Air updates.
Rationale: With OTA updates, the car owner does not have to go to the garage to install an update.

R_OU1.003 The software update mechanism shall provide the APPSTACLE App *OTA Update Manager* that is responsible for downloading and caching software updates for later installation.
Rationale: The OTA Update Manager standardizes the download of software updates and enables background prefetching.

R_OU1.004 The OTA Update Manager App shall have access to the Ex-Vehicle-Connectivity component.
Rationale: Wireless internet/cloud access is required in order to download update packages Over-the-Air.

R_OU1.005 The OTA Update Manager App shall have sufficient storage capacity for prefetching of software updates for later installation.

R_OU1.006 Non volatile user settings should be preserved during updates.

R_OU1.007 In case of an error during an update (e. g. due to a corrupt installation), a rollback mechanism should restore the system to its previous state.
Rationale: This prevents the vehicle from becoming unusable due to a fault.

R_OU1.008 The OTA Update Manager shall only download and apply authenticated and validated update packages.
Rationale: The OTA Update Manager requires transparent methods for securely downloading authenticated software updates that are ready for installation. The OTA Update Manager expects this functionality from the APPSTACLE In-Car-Platform.

R_OU1.009 The software update mechanism may offer extension points to enable the implementation of OEM-specific updates.
Rationale: This can be used, for example, to flash a new firmware image to an ECU installed in the vehicle or to activate/deactivate software features.

R_OU1.010 The OTA Update Manager should provide settings for automatic installation of updates.

Rationale: This is especially intended for cases where safety- or security-critical patches are to be installed.

Feature activation and deactivation

R_OU2.001 The software update mechanism should be able to activate and deactivate features of APPSTACLE Apps by adjusting settings/parameters.

R_OU2.002 The software update mechanism may be able to activate and deactivate functions in network-connected ECUs by modifying settings/parameters depending on the OEM-specific extension points described in **R_OU1.009**.

Update or add self-contained software features

R_OU3.001 The software update mechanism shall be able to exchange or update software features of the APPSTACLE Gateway.

R_OU3.002 The software update mechanism may be able to exchange or update software features of network-connected ECUs depending on the OEM-specific extension points described in **R_OU1.009**.

Firmware updates

R_OU4.001 The software update mechanism should support updating the APPSTACLE Gateway firmware.

R_OU4.002 The software update mechanism may support updating network-connected ECU firmware depending on the OEM-specific extension points described in **R_OU1.009**.

5.8. Hardware

5.8.1. Scope

Within the APPSTACLE project, two different hardware platforms are planned to be used for the APPSTACLE in-vehicle platform. The first one is a Raspberry Pi 3 as a so called "Community Board". This Community Board is meant to be used for initial developments and serves as an easy to obtain platform for interested people outside the APPSTACLE project. However, this Community Board does not contain automotive specific elements. The second hardware platform used for the APPSTACLE in-vehicle platform is the APPSTACLE project board. This APPSTACLE project board is a piece of hardware created during the course of the project and tailored to its needs as well as to the automotive domain. This section defines the hardware requirements regarding the Appstacle Project board.

5.8.2. Requirements

General

R_HW1.001 The APPSTACLE Project board shall be suitable to develop, test and demonstrate all of the APPSTACLE software.

R_HW1.002 The APPSTACLE Project Board shall consist of a gateway CPU unit providing connectivity between the Cloud, the central CPU unit, and all of the in-car ECUs.

R_HW1.003 The APPSTACLE project board shall be able to run Automotive Grade Linux.

R_HW1.004 The APPSTACLE project board should support security measures as far as necessary by means of hardware functions. This involves means to protect the system against manipulation and sabotage. The hardware functions may include AES encryption and tamper protection.

Interfaces

R_HW2.001 The main task of the distinct APPSTACLE project board consists in carrying out the connection between the central CPU and the Cloud via LTE and, later on, 5G. This involves all necessary software protocols and security checks. Apart from that, the Gateway serves as the central router for various in-vehicle fieldbus interfaces which may include Automotive Ethernet, CAN, CAN-FD, LIN, (MOST, Flexray). Furthermore, ex-vehicle interfaces, e.g. ITS-G5, may be connected via USB or Ethernet.

R_HW2.002 The APPSTACLE Project board may include a Human-Machine-Interface (HMI). This may consist of a TFT (e.g. 7") and a capacitive touchscreen.

R_HW2.003 The APPSTACLE project board should provide one or more Automotive Ethernet (e.g. IEEE 802.3 100Base-T1) interfaces.

R_HW2.004 The APPSTACLE project board should provide one or more Controller Area Network (CAN) interfaces.

Environmental Requirements

R_HW3.001 The power supply shall be compliant with vehicle electric systems, i. e. 12V to 24V. It also involves protection against load dumps and inductive spikes.

R_HW3.002 The operating temperature range should be defined as -40 to +85°C.

Bibliography

[1] *WikiLeaks Vault 7 Conspiracy: Michael Hastings Assassinated by CIA Remote Car Hack?* http://heavy.com/news/2017/03/wikileaks-vault-7-remote-car-hack-assassination-michael-hastings-conspiracy/. – Accessed: 2017-08-22

[2] An architectural blueprint for autonomic computing. In: *IBM White Paper* 31 (2006)

[3] Ali, Toqeer ; Ali, Jawad ; Ali, Tamleek ; Nauman, Mohammad ; Musa, Shahrulniza: Efficient, Scalable and Privacy Preserving Application Attestation in a Multi Stakeholder Scenario. In: Gervasi, Osvaldo (Publisher) ; Murgante, Beniamino (Publisher) ; Misra, Sanjay (Publisher) ; Rocha, Ana Maria A. (Publisher) ; Torre, Carmelo M. (Publisher) ; Taniar, David (Publisher) ; Apduhan, Bernady O. (Publisher) ; Stankova, Elena (Publisher) ; Wang, Shangguang (Publisher): *Computational Science and Its Applications – ICCSA 2016: 16th International Conference, Beijing, China, July 4-7, 2016, Proceedings, Part IV*. Springer International Publishing, 2016, P. 407–421. – ISBN 978-3-319-42089-9

[4] Ansari, Mohammad R. ; Miller, W T. ; She, Chenghua ; Yu, Qiaoyan: A low-cost masquerade and replay attack detection method for CAN in automobiles. In: *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on* IEEE (Organ.), 2017, P. 1–4

[5] ASAM: *ASAM MCD-3 MC*. 2017. – URL https://www.asam.net/standards/detail/mcd-3-mc/wiki/

[6] Audi AG: *Audi connect*. – URL http://www.audi.com/en/innovation/connect.html

[7] Audi AG: *Audi connect*. – URL http://www.audi.com/corporate/en/innovations/mobility-and-technology/audi-connect.html

[8] Audi AG: *Audi connect and infotainment*. 2016. – URL https://audi-illustrated.com/en/CES-2016/Audi-connect-und-Infotainment

[9] Automotive Grade Linux: *Automotive Grade Linux Requirments Specification v1.0*. URL https://www.automotivelinux.org/wp-content/uploads/sites/4/2017/08/agl_spec_v1_280515.pdf

[10] Automotive News: *Over-the-air updates on varied paths*. 2016. – URL http://www.autonews.com/article/20160125/OEM06/301259980/over-the-air-updates-on-varied-paths

[11] AUTOSAR: *AUTOSAR Software Component Template*. – URL https://www.autosar.org/fileadmin/files/standards/classic/4-3/methodology-and-templates/templates/standard/AUTOSAR_TPS_SoftwareComponentTemplate.pdf

[12] AUTOSAR: *AUTOSAR Specification of RTE*. – URL https://www.autosar.org/fileadmin/files/standards/classic/4-3/software-architecture/rte/standard/AUTOSAR_SWS_RTE.pdf

[13] Bace, Rebecca ; Mell, Peter: Intrusion Detection Systems. In: *Special Publication (NIST SP) - 800-31* (2001)

[14] Bace, Rebecca ; Mell, Peter: NIST special publication on intrusion detection systems / BOOZ-ALLEN AND HAMILTON INC MCLEAN VA. 2001. – Research report

[15] Berwanger, Josef ; Peller, Martin ; Griessbach, Robert: Byteflight a new protocol for safety critical applications. In: *Proceedings of the 28th FISITA World Automotive Congress. Seoul, Korea: FISITA*, 2000

[16] Bibhu, Vimal ; Kumar, Roshan ; Kumar, Balwant S. ; Singh, Dhirendra K.: Performance analysis of black hole attack in VANET. In: *International Journal Of Computer Network and Information Security* 4 (2012), Nb. 11, P. 47

[17] BMW Group: *The future of smart connectivity*. – URL `https://www.bmwgroup.com/en/innovation/technologies-and-mobility/connectivity.html`

[18] Bosch, Robert: CAN specification version 2.0. In: *Robert Bosch GmbH, Stuttgart* (1991)

[19] Bosch, Robert: CAN with Flexible Data-Rate specification. In: *Robert Bosch GmbH, Stuttgart* (2012). – http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can_fd_spec.pdf

[20] Braescu, F. C. ; Ferariu, L. ; Franciuc, A.: Monitoring CAN performances in distributed embedded systems. In: *15th International Conference on System Theory, Control and Computing*, Oct 2011, P. 1–6

[21] Breitfelder, Kim ; Messina, Don: IEEE 100: the authoritative dictionary of IEEE standards terms. In: *Standards Information Network IEEE Press. v879* (2000)

[22] Broster, Ian ; Burns, Alan: An analysable bus-guardian for event-triggered communication. In: *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003* (2003). ISBN 0-7695-2044-8

[23] Brown, D ; Cooper, Geoffrey ; Gilvarry, Ian ; Rajan, Anand ; Tatourian, Alan ; Venugopalan, Ramnath ; Wheeler, David ; Zhao, Meiyuan: Automotive security best practices. In: *White Paper* (2015), P. 1–17

[24] Buchverlag, DATACOM: *CAN calibration protocol*. 2017. – URL `http://www.itwissen.info/CCP-CAN-calibration-protocol.html`

[25] CAN in Automation: *CANopen application layer and communication profile, Draft Standard 301*. February 2011

[26] Cheifetz, Nicolas ; Same, Allou ; Aknin, Patrice ; De Verdalle, Emmanuel: A pattern recognition approach for anomaly detection on buses brake system. In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* (2011), P. 266–271. ISBN 9781457721984

[27] Cho, Kyong-tak ; Shin, Kang G.: Fingerprinting Electronic Control Units for Vehicle Intrusion Detection. In: *Usec* (2016), P. 911–927. – URL `https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cho`. ISBN 978-1-931971-32-4

[28] Cisco: *Quality of Service (QoS)*. 2017. – URL `https://www.cisco.com/c/en/us/products/ios-nx-os-software/quality-of-service-qos/index.html`

[29] Colajanni, Michele ; Dal Zotto, Luca ; Marchetti, Mirco ; Messori, Michele: The problem of NIDS evasion in mobile networks. In: *2011 4th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2011 - Proceedings* (2011). – ISBN 9781424487042

[30] Collabora Ltd.: *Apertis Developer Portal*. – URL https://developer.apertis.org/. – Access date: 2017-07-19

[31] Czarnecki, Krzysztof ; Helsen, Simon ; Eisenecker, Ulrich: Formalizing cardinality-based feature models and their specialization. In: *Software Process: Improvement and Practice* 10 (2005), Nb. 1, P. 7–29. – ISSN 1099-1670

[32] Daimler AG: *Network on board. Multimedia systems in the vehicle*. 2017. – URL https://www.daimler.com/innovation/case/connectivity/connectivity-in-the-vehicle.html

[33] Damopoulos, Dimitrios ; Kambourakis, Georgios ; Gritzalis, Stefanos: From keyloggers to touchloggers: Take the rough with the smooth. In: *Computers & security* 32 (2013), P. 102–114

[34] Damopoulos, Dimitrios ; Kambourakis, Georgios ; Gritzalis, Stefanos ; Park, Sang O.: Exposing mobile malware from the inside (or what is your mobile app really doing?). In: *Peer-to-Peer Networking and Applications* 7 (2014), Nb. 4, P. 687–697

[35] Damopoulos, Dimitrios ; Kambourakis, Georgios ; Portokalidis, Georgios: The Best of Both Worlds: A Framework for the Synergistic Operation of Host and Cloud Anomaly-based IDS for Smartphones. In: *Proceedings of the 7th European Workshop on System Security*. New York, NY, USA : ACM, 2014 (EuroSec '14), P. 6:1–6:6. – URL http://doi.acm.org/10.1145/2592791.2592797. – ISBN 978-1-4503-2715-2

[36] Damopoulos, Dimitrios ; Menesidou, Sofia A. ; Kambourakis, Georgios ; Papadaki, Maria ; Clarke, Nathan ; Gritzalis, Stefanos: Evaluation of anomaly-based IDS for mobile devices using machine learning classifiers. In: *Security and Communication Networks* 5 (2012), Nb. 1, P. 3–14

[37] Debar, Hervé ; Dacier, Marc ; Wespi, Andreas: A revised taxonomy for intrusion-detection systems. (2000), P. 361–378

[38] Denning, E ; Ave, Ravenswood ; Park, Menlo: An intrusion detection model. In: *IEEE Transactions on Software Engineering* (1986), P. 118–131

[39] E. Aina, S. S.: *Apertis Platform Guide*. – URL https://developer.apertis.org/index.html. – Access date: 2017-10-16

[40] Eclipse: *Franca*. – URL https://eclipse.org/proposals/modeling.franca/

[41] Electronics, BuB: *OBD II Background*. 2011. – URL http://www.obdii.com/background.html

[42] Embedded Computing Design: *OTA update possibilities put automotive on the road to V2X*. 2014. – URL http://www.embedded-computing.com/embedded-computing-design/ota-update-possibilities-put-automotive-on-the-road-to-v2x

[43] ETAS GmbH: *Fast Firmware Updates Over-the-Air*. 2017. – URL https://www.etas.com/download-center-files/engineering/Whitepaper_Bosch_FOTA_2017_web.pdf

[44] Festag, Andreas: Cooperative intelligent transport systems standards in Europe. In: *in IEEE communications magazine* 52 (2014), P. 12

[45] Feuer, Magnus: *Talk: Vehicle Signal Specification*. Apr 2016

[46] FlexRay Consortium: FlexRay Communications System Protocol Specification Version 3.1, 2010. In: *Available at http{ www. flexray. com}*

[47] Ford Motor Company: *The future of smart connectivity*. 2017. – URL https://www.ford.com/technology/sync/sync-3/

[48] Forrest, S. ; Hofmeyr, S.A. ; Somayaji, A. ; Longstaff, T.A.: A sense of self for Unix processes. In: *Proceedings 1996 IEEE Symposium on Security and Privacy* (1996), P. 120–128. – URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=502675. – ISBN 0-8186-7417-2

[49] GENIVI: *Common API*. – URL https://at.projects.genivi.org/wiki/display/COMMONAPICPP/CommonAPI-cpp

[50] Gmiden, Mabrouka ; Gmiden, Mohamed H. ; Trabelsi, Hafedh: An Intrusion Detection Method for Securing In-Vehicle CAN bus. (2016), P. 176–180. ISBN 9781509034079

[51] Golem.de: *Lexus-Navigationssystem nach Over-The-Air-Update unbrauchbar*. 2016. – URL https://www.golem.de/news/auto-lexus-navigationssystem-nach-over-the-air-update-unbrauchbar-1606-121393.html

[52] Google Inc.: *Android Automotive*. – URL https://source.android.com/devices/automotive/. – Access date: 2017-10-30

[53] Google Inc.: *Android Interfaces and Architecture*. – URL https://source.android.com/devices/. – Access date: 2017-10-30

[54] Groll, André ; Ruland, Christoph: Secure and authentic communication on existing in-vehicle networks. In: *IEEE Intelligent Vehicles Symposium, Proceedings* (2009), P. 1093–1097. – ISBN 9781424435043

[55] Gu, Zhongshu ; Pei, Kexin ; Wang, Qifan ; Si, Luo ; Zhang, Xiangyu ; Xu, Dongyan: LEAPS: Detecting Camouflaged Attacks with Statistical Learning Guided by Program Analysis. In: *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, IEEE, 2015, P. 57–68. – ISBN 978-1-4799-8629-3

[56] Gupta, Deepak ; Jalote, Pankaj: On-line software version change using state transfer between processes. In: *Software: Practice and Experience* 23 (1993), Nb. 9, P. 949–964

[57] Hank, Peter ; Müller, Steffen ; Vermesan, Ovidiu ; Van Den Keybus, Jeroen: Automotive ethernet: in-vehicle networking and smart mobility. In: *Proceedings of the Conference on Design, Automation and Test in Europe* EDA Consortium (Organ.), 2013, P. 1735–1739

[58] Hank, Peter ; Suermann, Thomas ; Müller, Steffen: Automotive Ethernet, a holistic approach for a next generation in-vehicle networking standard. In: *Advanced Microsystems for Automotive Applications 2012* (2012), P. 79–89

[59] Hedges, Chris ; Perry, Frank: *Overview and use of SAE J2735 message sets for commercial vehicles*. 2008. – SAE Technical Paper, No. 2008-01-2650

[60] here: *Vehicle Sensor Data Cloud Ingestion Interface Specification v2.0.2*. 2015. – URL https://lts.cms.here.com/static-cloud-content/Company_Site/2015_06/Vehicle_Sensor_Data_Cloud_Ingestion_Interface_Specification.pdf

[61] Hoppe, Tobias ; Kiltz, Stefan ; Dittmann, Jana: Security threats to automotive CAN networks–practical examples and selected short-term countermeasures. In: *International Conference on Computer Safety, Reliability, and Security* Springer (Organ.), 2008, P. 235–248

[62] Hoppe, Tobias ; Kiltz, Stefan ; Dittmann, Jana: Applying intrusion detection to automotive it-early insights and remaining challenges. In: *Journal of Information Assurance and Security (JIAS)* 4 (2009), Nb. 6, P. 226–235

[63] Hoppe, Tobias C. ; Kiltz, Stefan ; Dittmann, Jana: Applying Intrusion Detection to Automotive IT – Early Insights and Remaining Challenges. 4 (2009), Nb. May, P. 226–235

[64] Houston Ford of Pine River: *Can I update Ford Sync using my home Wi-Fi?* 2017. – URL http://www.houstonford.com/blog/update-ford-sync-software-using-wi-fi-information/

[65] Hu, Hong ; Shinde, Shweta ; Adrian, Sendroiu ; Chua, Zheng L. ; Saxena, Prateek ; Liang, Zhenkai: Data-oriented programming: On the expressiveness of non-control data attacks. In: *Security and Privacy (SP), 2016 IEEE Symposium on* IEEE (Organ.), 2016, P. 969–986

[66] IEEE: *IEEE Standards Style Manual*. – URL https://development.standards.ieee.org/myproject/Public/mytools/draft/styleman.pdf

[67] Ilgun, Koral ; Kemmerer, R.A. ; Porras, P.A.: State transition analysis: a rule-based intrusion detection approach. In: *IEEE Transactions on Software Engineering* 21 (1995), Nb. 3, P. 181–199. – URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=372146. – ISBN 0098-5589

[68] Imtiaz, Jahanzaib ; Jasperneite, Jürgen ; Han, Lixue: A performance study of Ethernet Audio Video Bridging (AVB) for Industrial real-time communication. In: *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on* IEEE (Organ.), 2009, P. 1–8

[69] IoT Now: *Securing automotive over-the-air updates*. 2017. – URL https://www.iot-now.com/2017/02/27/59018-securing-automotive-air-updates/

[70] Ishtiaq Roufa, Rob M. ; Mustafaa, Hossen ; Travis Taylora, Sangho O. ; Xua, Wenyuan ; Gruteserb, Marco ; Trappeb, Wade ; Seskarb, Ivan: Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In: *19th USENIX Security Symposium, Washington DC*, 2010, P. 11–13

[71] Ismail, Roslan ; Syed, Toqeer A. ; Musa, Shahrulniza: Design and Implementation of an Efficient Framework for Behaviour Attestation Using N-call Slides. In: *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*. New York, NY, USA : ACM, 2014 (ICUIMC '14), P. 36:1–36:8. – URL http://doi.acm.org/10.1145/2557977.2558002. – ISBN 978-1-4503-2644-5

[72] ISO: 11898-1–Road vehicles–Controller area network (CAN)–Part 1: Data link layer and physical signalling. In: *International Organization for Standardization* (2003)

[73] ISO: 11898-2, Road vehicles Controller area network (CAN) Part 2: High-speed medium access unit. In: *International Organization for Standardization* (2003)

[74] ITU-R: IMT Vision–Framework and overall objectives of the future development of IMT for 2020 and beyond. (2015)

[75] ITWissen: *QoS (quality of service)*. 2017. – URL http://www.itwissen.info/QoS-quality-of-service-Dienstguete.html

[76] Jacobson, Emily R. ; Bernat, Andrew R. ; Williams, William R. ; Miller, Barton P.: Detecting Code Reuse Attacks with a Model of Conformant Program Execution. In: Jürjens,

Jan (Publisher) ; Piessens, Frank (Publisher) ; Bielova, Nataliia (Publisher): *Engineering Secure Software and Systems: 6th International Symposium, ESSoS 2014, Munich, Germany, February 26-28, 2014, Proceedings*. Cham : Springer International Publishing, 2014, P. 1–18. – URL https://doi.org/10.1007/978-3-319-04897-0_1. – ISBN 978-3-319-04897-0

[77] Jaguar Land Rover: *Vehicle Signal Specification*. – URL https://github.com/GENIVI/vehicle_signal_specification

[78] Jaguar Land Rover: *Vehicle Signal Specification*. – URL https://github.com/GENIVI/vehicle_signal_specification

[79] Japkowicz, Nathalie ; Taylor, Adrian: Frequency-Based Anomaly Detection for the Automotive CAN bus. (2015), P. 45–49

[80] Jennings, Nicholas R.: On agent-based software engineering. In: *Artificial Intelligence* 117 (2000), Nb. 2, P. 277 – 296. – URL http://www.sciencedirect.com/science/article/pii/S0004370299001071. – ISSN 0004-3702

[81] Johanson, Mathias ; Dahle, Pål ; Soderberg, A: Remote vehicle diagnostics over the internet using the DoIP protocol. In: *The Sixth International Conference on Systems and Networks Communications*, 2011

[82] Juliussen, Egil: Connected Cars: Perspectives to 2025. Paris : Presented on the 14th GENIVI all members meeting, 2016. – URL https://at.projects.genivi.org/wiki/display/WIK4/14th+GENIVI+AMM. – Access date: 2017-10-30

[83] Kambourakis, Georgios ; Damopoulos, Dimitrios ; Papamartzivanos, Dimitrios ; Pavlidakis, Emmanouil: Introducing touchstroke: keystroke-based authentication system for smartphones. In: *Security and Communication Networks* 9 (2016), Nb. 6, P. 542–554

[84] Kang, K. C. ; Cohen, S. G. ; Hess, J. A. ; Novak, W. E. ; Peterson, A. S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study / Carnegie-Mellon University Software Engineering Institute. November 1990. – Technical Report No. CMU/SEI-90-TR-21

[85] Knirsch, Matthias ; Kesch, Bernd ; Tappe, Matthias ; Driedger, Günter ; Lehle, Walter: *Diagnosis*. P. 304–325. In: *Gasoline Engine Management*, Springer Fachmedien, 2014

[86] Ko, C. ; Ruschitzka, M. ; Levitt, K.: Execution monitoring of security-critical programs in distributed systems: a specification-based approach. In: *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)* (1997), P. 175–187. – URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=601332. – ISBN 0-8186-7828-3

[87] Koscher, Karl ; Czeskis, Alexei ; Roesner, Franziska ; Patel, Shwetak ; Kohno, Tadayoshi ; Checkoway, Stephen ; McCoy, Damon ; Kantor, Brian ; Anderson, Danny ; Shacham, Hovav u. a.: Experimental security analysis of a modern automobile. In: *Security and Privacy (SP), 2010 IEEE Symposium on* IEEE (Organ.), 2010, P. 447–462

[88] Kramer, Jeff ; Magee, Jeff: The evolving philosophers problem: Dynamic change management. In: *IEEE Transactions on software engineering* 16 (1990), Nb. 11, P. 1293–1306

[89] Larson, Ulf E. ; Nilsson, Dennis K. ; Jonsson, Erland: An approach to specification-based attack detection for in-vehicle networks. In: *Intelligent Vehicles Symposium, 2008 IEEE* IEEE (Organ.), 2008, P. 220–225

[90] Lazarevic, Aleksandar ; Kumar, Vipin ; Srivastava, Jaideep: Intrusion detection: A survey. In: *Managing Cyber Threats*. Springer, 2005, P. 19–78

[91] Li, Wei: Using genetic algorithm for network intrusion detection. In: *Proceedings of the United States Department of Energy Cyber Security Group* 1 (2004), P. 1–8

[92] Lin, Xingqin ; Andrews, Jeffrey ; Ghosh, Amitabha ; Ratasuk, Rapeepat: An overview of 3GPP device-to-device proximity services. In: *IEEE Communications Magazine, no* 52 (2014), Nb. 4, P. 40–48

[93] LIN Consortium: LIN specification package, revision 2.0. In: *Munich, Germany* (2003)

[94] LoRa Alliance: *A technical overview of LoRa and LoRaWAN*. November 2015

[95] Luckham, David C.: *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ©2001. – ISBN 0201727897

[96] Luh, Robert ; Marschalek, Stefan ; Kaiser, Manfred ; Janicke, Helge ; Schrittwieser, Sebastian: Semantics-aware detection of targeted attacks: a survey. In: *Journal of Computer Virology and Hacking Techniques* 13 (2017), Nb. 1, P. 47–85

[97] Lunt, Teresa F. ; Tamaru, Ann ; Gilham, Fred ; Jagan, Nathan R. ; Jalali, Caveh ; Neumann, Peter G.: A real-time intrusion-detection expert system (ides). (1992)

[98] Maggi, Fabrizio M. ; Montali, Marco ; Westergaard, Michael ; Van Der Aalst, Wil M P.: Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6896 LNCS (2011), P. 132–147. – ISBN 9783642230585

[99] Marschalek, Stefan ; Luh, Robert ; Kaiser, Manfred ; Schrittwieser, Sebastian: Classifying malicious system behavior using event propagation trees. In: Indrawan-Santiago, Maria (Publisher) ; Anderst-Kotsis, Gabriele (Publisher): *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services* ACM (Organ.), 2015, P. 1–10

[100] Mercedes-Benz: *In-Car Purchase: Special Features on Demand*. – URL `https://www.mercedes-benz.com/en/taubenheim-13/taubenheim13blog/in-car-purchase-special-features-on-demand/`

[101] Miller, Charlie ; Valasek, Chris: Adventures in automotive networks and control units. In: *DEF CON* 21 (2013), P. 260–264

[102] Miller, Charlie ; Valasek, Chris: A survey of remote automotive attack surfaces. In: *black hat USA* (2014)

[103] Miller, Charlie ; Valasek, Chris: Remote exploitation of an unaltered passenger vehicle. In: *Black Hat USA* 2015 (2015)

[104] Mitchell, Robert ; Chen, Ing-ray ; Tech, Virginia: A Survey of Intrusion Detection Techniques for Cyber-Physical Systems. 46 (2014), Nb. 4

[105] Mohammed Taha Elgraini, Nasser Assem, Taiieeddine Rachtdt: HOST INTRUSION DETECTION FOR LONG STEALTHY SYSTEM CALL SEQUENCES. In: *Colloquium in Information Science and Technology (CIST), 2012* Volume 22 - 24 Oct. 2012, Fez, Morocco ; proceedings. URL `http://ieeexplore.ieee.org/servlet/opac?punumber=6377152`

[106] MOST Cooperation: *MOST specification revision 2.3*. 2004

[107] Mukkamala, Srinivas ; Janoski, Guadalupe ; Sung, Andrew: Intrusion detection using neural networks and support vector machines. In: *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on* Volume 2 IEEE (Organ.), 2002, P. 1702–1707

[108] Murtaza, S. S. ; Khreich, W. ; Hamou-Lhadj, A. ; Couture, M.: A host-based anomaly detection approach by representing system calls as states of kernel modules. In: *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, Nov 2013, P. 431–440. – ISSN 1071-9458

[109] Müter, Michael ; Asaj, Naim: Entropy-based anomaly detection for in-vehicle networks. In: *Intelligent Vehicles Symposium (IV), 2011 IEEE* IEEE (Organ.), 2011, P. 1110–1115

[110] Müter, Michael ; Groll, André ; Freiling, Felix C.: A structured approach to anomaly detection for in-vehicle networks. In: *Information Assurance and Security (IAS), 2010 Sixth International Conference on* IEEE (Organ.), 2010, P. 92–98

[111] Müter, Michael ; Groll, André ; Freiling, Felix C.: Anomaly Detection for In-Vehicle Networks using a Sensor-based Approach. 6 (2011), P. 132–140

[112] Nilsson, Dennis K. ; Larson, Ulf E. ; Jonsson, Erland: Efficient in-vehicle delayed data authentication based on compound message authentication codes. In: *IEEE Vehicular Technology Conference* (2008), P. 1–5. – ISBN 9781424417223

[113] Nilsson, Dennis K. ; Larson, Ulf E. ; Picasso, Francesco ; Jonsson, Erland: A First Simulation of Attacks in the Automotive Network Communications Protocol FlexRay. In: *Springer* (2009), P. 84–91

[114] Odat, H. A. ; Ganesan, S.: Firmware over the air for automotive, Fotamotive. In: *IEEE International Conference on Electro/Information Technology*, June 2014, P. 130–139

[115] Oguma, Hisashi ; Yoshioka, Akira ; Nishikawa, Makoto ; Shigetomi, Rie ; Otsuka, Akira ; Imai, Hideki: New Attestation-Based Security Architecture for In-vehicle Communication. (2008), P. 1–6. ISBN 9781424423248

[116] Palanca, Andrea ; Evenchick, Eric ; Maggi, Federico ; Zanero, Stefano: A stealth, selective, link-layer denial-of-service attack against automotive networks. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* Springer (Organ.), 2017, P. 185–206

[117] Panos, Christoforos ; Xenakis, Christos ; Kotzias, Platon ; Stavrakakis, Ioannis: A specification-based intrusion detection engine for infrastructure-less networks. In: *Computer Communications* 54 (2014), P. 67–83. – URL `http://dx.doi.org/10.1016/j.comcom.2014.08.002`. – ISSN 01403664

[118] Pfeiffer, Olaf ; Ayre, Andrew ; Keydel, Christian: *Embedded networking with CAN and CANopen*. Copperhill Media, 2008

[119] Pillmann, Johannes ; Wietfeld, Christian ; Zarcula, Adrian ; Raugust, Thomas ; Alonso, Daniel C.: Novel common vehicle information model (CVIM) for future automotive vehicle big data marketplaces. In: *Intelligent Vehicles Symposium (IV), 2017 IEEE* IEEE (Organ.), 2017, P. 1910–1915

[120] Porras, Phillip A. ; Valdes, Alfonso: Live Traffic Analysis of TCP/IP Gateways. In: *ISOC Symposium on Network and Distributed System Security (NDSS'98), San Diego, CA* (1998)

[121] QNX Software Systems: *QNX CAR Platform for Infotainment*. – URL https://www.qnx.com/content/qnx/en/products/qnxcar/index.html. – Access date: 2017-10-30

[122] QNX Software Systems: *QNX for Automotive*. – URL http://blackberry.qnx.com/en/solutions/industries/automotive/index. – Access date: 2017-10-30

[123] Qualcomm: *The path to 5G: Cellular Vehicle-to-Everything (C-V2X)*. – URL https://www.qualcomm.com/documents/path-5g-cellular-vehicle-everything-c-v2x

[124] Ratasuk, Rapeepat ; Mangalvedhe, Nitin ; Ghosh, Amitava ; Vejlgaard, Benny: *Narrowband LTE-M system for M2M communication*. IEEE : in Vehicular Technology Conference (VTC Fall)

[125] Ratasuk, Rapeepat ; Mangalvedhe, Nitin ; Zhang, Yanji ; Robert, Michel ; Koskinen, Jussi-Pekka: Overview of narrowband IoT in LTE Rel-13. In: *IEEE conference on Standards for Communications and Networking (CSCN)*, 2016

[126] Rieke, Roland ; Seidemann, Marc ; Talla, Elise K. ; Zelle, Daniel ; Seeger, Bernhard: Behavior Analysis for Safety and Security in Automotive Systems. In: *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)* (2017), P. 381–385. – URL http://ieeexplore.ieee.org/document/7912675/. ISBN 978-1-5090-6058-0

[127] Road & Track Magazine: *Tesla Offers Free One-Month Autopilot Trial to Nearly Every Tesla Owner*. 2016. – URL http://www.roadandtrack.com/new-cars/car-technology/news/a28872/tesla-autopilot-trial-month-long/

[128] Robert Bosch Engineering and Business Solutions Limited: *Help*. 2017. – URL https://raw.githubusercontent.com/rbei-etas/busmaster-documents/master/help.pdf

[129] Robert Bosch GmbH, ETAS GmbH, MHS Elektronik, IXXAT, FKFS, Gigatronik, ICT Softwareengineering, VS com, NSI Altran, ASTR Soft: *BUSMASTER is an Open Source Software tool to Simulate, Analyze and Test data bus systems such as CAN, CAN FD, LIN, FlexRay*. November 2017. – http://rbei-etas.github.io/busmaster/

[130] Scarfone, Karen ; Mell, Peter: Guide to Intrusion Detection and Prevention Systems (IDPS) (Draft) Recommendations of the National Institute of Standards and Technology. In: *Nist Special Publication* 800-94 (2007), P. 127. – URL http://www.reference.com/go/http://csrc.ncsl.nist.gov/publications/nistpubs/800-94/SP800-94.pdf

[131] Scarfone, Karen ; Mell, Peter: Guide to intrusion detection and prevention systems (idps). In: *NIST special publication* 800 (2007), Nb. 2007, P. 94

[132] Sekar, R ; Gupta, A ; Frullo, J ; Shanbhag, T ; Tiwari, A ; Yang, H ; Zhou, S: Specification-based anomaly detection. In: *Proceedings of the 9th ACM conference on Computer and communications security - CCS '02* 26 (2002), Nb. 2, P. 265. – URL http://portal.acm.org/citation.cfm?id=586146{%}5Cnhttp://portal.acm.org/citation.cfm?doid=586110.586146. – ISBN 1581136129

[133] Seo, Hanbyul ; Lee, Ki-Dong ; Yasukawa, Shinpei ; Peng, Ying ; Sartori, Philippe: *LTE evolution for vehicle-to-everything services*. IEEE Communications Magazine, 2016

[134] Shu, Xiaokui ; Yao, Danfeng ; Ramakrishnan, Naren ; Jaeger, Trent: Long-Span Program Behavior Modeling and Attack Detection. In: *ACM Transactions on Privacy and Security* 20 (2017), Nb. 4, P. 1–28. – ISSN 24712566

[135] Sierra Wireless: *Legato Open Source Linux Platform*. – URL https://www.sierrawireless.com/products-and-solutions/embedded-solutions/open-source-initiatives/. – Access date: 2017-10-16

[136] Song, Hyun M. ; Kim, Ha R. ; Kim, Huy K.: Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network. In: *Information Networking (ICOIN), 2016 International Conference on* IEEE (Organ.), 2016, P. 63–68

[137] Spiegel Online: *Audi ist ein Level weiter*. 2017. – URL http://www.spiegel.de/auto/aktuell/audi-a8-audi-ist-beim-autonomen-fahren-ein-level-weiter-a-1169062.html

[138] Standardization (ISO), International O. for: *ISO 27145-3 Road vehicles – Implementation of World-Wide Harmonized On-Board Diagnostics (WWH-OBD) communication requirements – Part 3: Common message dictionary*. Aug 2012

[139] Standardization (ISO), International O. for: *ISO 14230 Road vehicles – Diagnostic communication over K-Line*. Aug 2015

[140] Standardization (ISO), International O. for: *ISO 20077-1 Road Vehicles — Extended vehicle (ExVe) methodology — Part 1: General information*. 2017. – URL https://www.iso.org/obp/ui/#iso:std:66975:en

[141] Standardization (ISO), International O. for: *ISO 20080 Road vehicles - Information for remote diagnostic support - General requirements, definitions and use cases*. 2017. – URL https://www.iso.org/standard/66979.html

[142] Streif, Rudolf J.: *Talk: Vehicle Data Interfaces*. Oct 2016

[143] Striki, Maria ; Manousakis, Kyriakos ; Kindred, Darrell ; Sterne, Dan ; Lawler, Geoff ; Ivanic, Natalie ; Tran, George: Quantifying resiliency and detection latency of intrusion detection structures Maria Striki Kyriakos Manousakis. In: *Proceedings - IEEE Military Communications Conference MILCOM* (2009). ISBN 9781424452385

[144] Taylor, Adrian ; Leblanc, Sylvain ; Japkowicz, Nathalie: Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks. In: *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)* (2016), P. 130–139. – URL http://ieeexplore.ieee.org/document/7796898/. ISBN 978-1-5090-5206-6

[145] The Open Automotive Alliance: *Open Automotive Alliance - Members*. – URL https://www.openautoalliance.net/#members. – Access date: 2017-10-30

[146] Ujiie, Yoshihiro ; Kishikawa, Takeshi ; Haga, Tomoyuki ; Matsushima, Hideki ; Wakabayashi, Tohru ; Tanabe, Masato ; Kitamura, Yoshihiko ; Anzai, Jun: A Method for Disabling Malicious CAN Messages by Using a CMI-ECU. (2016). – URL http://papers.sae.org/2016-01-0068/

[147] Uppuluri, P ; Sekar, R: Experiences with specification-based intrusion detection. In: *Recent advances in intrusion detection (RAID) '00* (2000), P. 1–18. – URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.2352. – ISBN 3540427023

[148] Vaccaro, H.S. ; Liepins, G.E.: *Detection of anomalous computer session activity*. 1989

[149] Valdes, Alfonso: Detecting novel scans through pattern anomaly detection. In: *DARPA Information Survivability Conference and Exposition, 2003. Proceedings* Volume 1 IEEE (Organ.), 2003, P. 140–151

[150] Vector: *Measurement and Calibration Protocol XCP - Fundamentals*. 2017. – URL https://vector.com/vi_xcp_basics_en.html

[151] Vector Informatik GmbH: *CANoe/CANalyzer Versions and Supported Hardware*. 2017. – URL https://kb.vector.com/upload_551/file/CANwin_Versions_and_Supported_Hardware_for%20KB_ext_555(9).pdf

[152] Vector Informatik GmbH: *Feature Matrix CANoe 10.0 and CANalyzer 10.0*. 2017. – URL https://vector.com/portal/medien/cmc/datasheets/CANoe_CANalyzer_FeatureMatrix_DataSheet_EN.pdf

[153] Volkswagen AG: *Guide & Inform*. 2017. – URL http://page.volkswagen-carnet.com/de_de/dienste-und-pakete/guide-und-inform.html

[154] Volvo Cars Support: *Car system updates*. 2017. – URL http://support.volvocars.com/uk/Pages/article.aspx?article=27e39ea875601dcfc0a801513b825d11

[155] Waszecki, Peter ; Mundhenk, Philipp ; Steinhorst, Sebastian ; Lukasiewycz, Martin ; Karri, Ramesh ; Chakraborty, Samarjit: Automotive Electrical/Electronic Architecture Security via Distributed In-Vehicle Traffic Monitoring. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2017)

[156] Wikimedia: *Keyword Protocol 2000*. 2017. – URL https://en.wikipedia.org/wiki/Keyword_Protocol_2000

[157] Wikimedia: *Unified Diagnostic Services*. 2017. – URL https://en.wikipedia.org/wiki/Unified_Diagnostic_Services

[158] World Wide Web Consortium: *Vehicle Signal Server Specification*. – URL https://www.w3.org/TR/vehicle-information-service/

[159] World Wide Web Consortium: *Vehicle Signal Server Specification*. – URL https://www.w3.org/TR/vehicle-information-service/

[160] XiaGuo, Liu: *Snappy Ubuntu Core - Enabling secure devices with app stores*. – URL http://7xi8kv.com5.z0.glb.qiniucdn.com/SnappyUbuntuCoreintroduction-changsha.pdf. – Access date: 2017-10-16

[161] Xu, Kui ; Yao, Danfeng D. ; Ryder, Barbara G. ; Tian, Ke: Probabilistic program modeling for high-precision anomaly classification. In: *Computer Security Foundations Symposium (CSF), 2015 IEEE 28th* IEEE (Organ.), 2015, P. 497–511

[162] Yolacan, Esra N. ; Dy, Jennifer G. ; Kaeli, David R.: System Call Anomaly Detection Using Multi-HMMs. In: *2014 IEEE Eighth International Conference on Software Security and Reliability-Companion*, IEEE, 2014, P. 25–30. – ISBN 978-1-4799-5843-6