INTEGRATING VEHICLE DATA

Mart Oude Weernink

University of Twente

Allego

Developing a system for the integration of vehicle data

by

M.J.P. Oude Weernink

to obtain the degree of Master of Science
at the University of Twente,
to be defended publicly on Friday August 31, 2018 at 02:00 PM.

Student number:	1371649	
Email:	m.j.p.oudeweernink@alumnus.utwente.nl	
Project duration:	January 1, 2018 – August 31, 2018	
Thesis committee:	Dr.ir. J.M. Moonen,	Universiteit Twente, supervisor
	Prof.dr. ir. A. Rensink,	Universiteit Twente
	Ir. F. Verhulst,	Allego

UNIVERSITY OF TWENTE. allego

Abstract

In the fast growing E-Mobility market, it is increasingly important to properly support the charging process of Electric Vehicles (EV). Allego as Charge Point Operator (CPO) offers services to manage EV-charging infrastructures. Integration of vehicle data can help to support the current services and offerings and also offers opportunities for new services. The integration of vehicle data can thus enable a CPO to extend and expand its service offering.

This thesis discusses the different data sources and protocols that can be used to gather vehicle data. In addition, the various applications of vehicle data within the scope of a CPO are identified.

Due to the great diversity of data sources and protocols that can be used to access vehicle data, a versatile platform needs to be set up to cope with this diversity in an efficient, flexible and scalable manner. This thesis gives an architecture that enables the integration and combination of multiple data sources. Based on existing middleware techniques and applications in practice, an architecture for a platform is created that connects (internal and external) systems in in an action-driven manner.

The architecture developed has been designed in such a way that data can be integrated in a flexible and scalable manner. The principles developed can also be applied to other assets, such as charging stations.

Glossary

Abbreviation	Definition
API	Application Programming Interface
CAN	Controller Area Network
CPO	Charge Point Operator
EV	Electric Vehicle
EVSE	Electric Vehicle Supply Equipment (Charge Point)
GDPR	General Data Protection Regulation
GTFS	General Transit Feed Specification
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
MVP	Minimum Viable Product
NDOV	Nationale Data Openbaar Vervoer
OCPP	Open Charge Point Protocol
OCPI	Open Charge Point Interface
OEM	Original Equipment Manufacturer
PTO	Public Transport Operator -
PTA	Public Transport Authority -
SoC	State of Charge
VPN	Virtual Private Network

Contents

1

Introduction

1.1. Introduction

Computer Science is a broad scientific field that studies the possibilities of automating algorithmic processes that scale. It is in this context that many applications of Computer Science have evolved from relatively simple structures to more complex setups that support the ever rising scale at which the applications are deployed.
In the beginning, computer systems were mainly used for educational purposes and for corporations that ran mainframe systems that were accessed through terminal computers. The main goal was communication and there often was no higher function. The introduction of the IBM PC started a new era in which the computer was available to a wider public [1]. In the nineties when the internet emerged [2] computers started to become more widespread and available for more people. During the nineties, the internet evolved from a document-centric communication platform to a more service-oriented 'Dynamic Web' [3].

This new demand for online services is seen as the start of the concept of cloud-computing. Amazon made the first move to cloud services as we now know them; the Amazon datacentres had an overcapacity of 90 percent to handle peak loads, and Amazon decided to develop a cloud architecture on their hardware to use this overcapacity also in off-peak time [4].

The EV-market, especially seen from a CPO perspective, is a very good example of usage of a service-oriented Web. The main objective of a CPO is to provide a service for managing of charging stations in the field. The amount of charging stations grows rapidly, thus challenging the scalability of the services that the CPO develops [5]. Only managing of charging stations is not enough; in the growing market for electric vehicles (EVs), new challenges arise to improve the efficiency and effectiveness of the EV-driving experience. Customers are demanding more functionality and features.

In the Public Transport market, for example, Allego offers services to manage EV charging infrastructures. These services stretch further than basic insights in the status of charge poles. In the future they want to offer value-added services such as active monitoring of entire fleets of buses. Fleet owners are, for example, increasingly interested in the exact position and state of their vehicles [6].

This is a reason for Allego to investigate the possibilities for the integration of vehicle data into their existing IT-platform. With these new demands, Allego faces new challenges both on functional and non-functional level. Allego has to be more flexible then ever to support the newly requested features, but at the same time, the infrastructure has to be resilient to the enormous scale-up that is currently occuring.

This final project reports on means to extract realtime vehicle data from vehicles in a both flexible and scalable way. The project covers both an analysis of existing offerings, but also proposes a new framework for handling this challenge. Lastly, the thesis focuses on the current cloud-infrastructure and how this infrastructure may be improved so that it is ready to support applications that will be enriched with vehicle data.

1.2. Research Purpose

The main purpose of this final project is to develop a scalable method for the integration vehicle data from electric vehicles into an existing backend-system for charge pole management.
An overview of the integration of vehicle data is given in figure 1.1. The figure shows the extraction of vehicle data and third-party data related to vehicles. This data sources are integrated in such a way that they can support the internal services within the Allego platform.

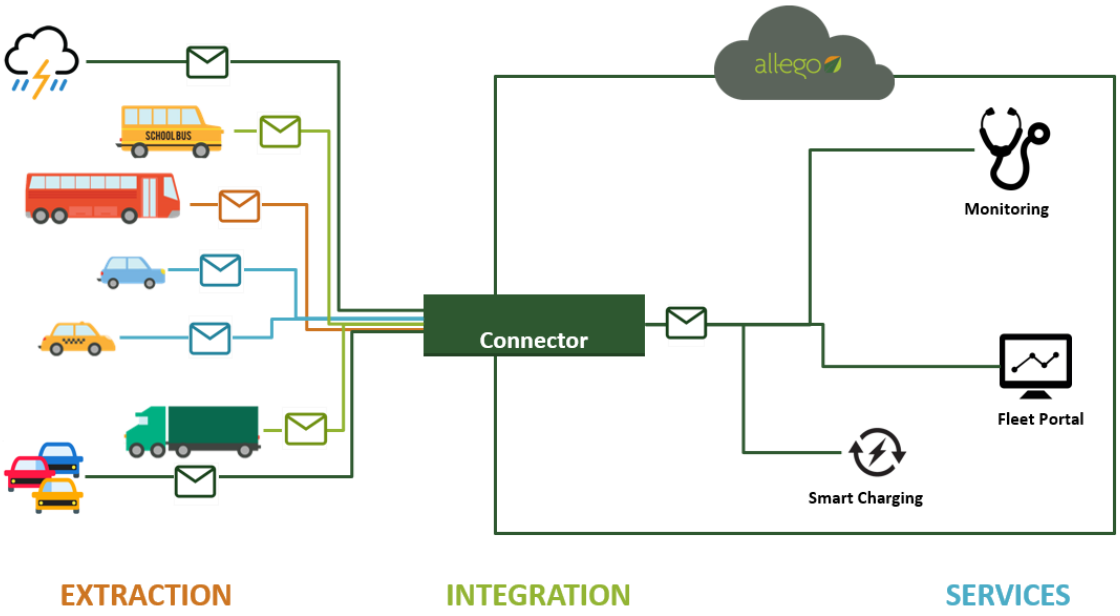


Figure 1.1: Overview integration vehicle data (Extraction, Integration, Services)

Extraction

In order to integrate data from electric vehicles, the data has to be gathered from the vehicles. With conventional vehicles there are several methods for tracking and receiving vehicle data, for example ODB-dongles, and CAN-tracking devices [7]. Beside the existing data extraction methods there are developments in the field of data exchange for connected cars. For example, the Car Connectivity Consortium is developing the Car Data model which makes it possible to exchange car data between OEM and third parties in a standardized manner [8]. Within Europe, car manufacturers are working together to support the exchange of vehicle data to service providers [9].
In this project no new extraction methods are developed, however the different extraction methods available influence the solution developed within this project.

Integration

In order to support the different types of extraction sources, the architecture should be capable of supporting a large amount of different data extraction methods. This requires the architecture to be flexible. Currently there are already a multitude of methods to extract vehicle data. For this final project, we will use the public transport sector to demonstrate different extraction methods. In the future there might be a demand for the integration of even more types of (vehicle) data as well, for instance data from trucks or passenger cars. Seen from an even broader perspective, it is very well possible that a wide variety of other data sources will have to be integrated in an equal way. Examples could be traffic density information or current weather conditions. The architecture should be developed in such a way that it is possible to support these future demands regarding types of

supported vehicles and supported services.

Services
After all the data has been gathered in the right way and the integration layer has processed the data in such a way that the data is available in a standardized format, there will be a multitude of services that would profit from being able to use this data. There are currently already some services that would benefit from the availability of vehicle data. For example, the Smart Charging solution currently in development within Allego can use the integration of vehicle data to improve the charging process. In the analysis phase of this thesis several (existing and new services) that can benefit from vehicle data are described. The project does not aim to develop new services, but it should at least fully support the use cases for all these services.

This thesis is focused on the second step in the integration process: the integration of vehicle data from different sources and providing it in a standardized format to support multiple services in the backend platform. The focus is not on developing new standards for the extraction of vehicle data. Based on the exploration made during the Research Topics phase we concluded that different standards (both on data and physical level) are already developed, and these standards can also be very well used in the EV-domain. In addition, this research will not focus on developing new services in the backend, because it is a crucial prerequisite to set up a good architecture and standardized central data format to provide data to these services before developing them.
The research thus focuses on the bridge between extraction and services: the integration phase. It strives to develop a platform that ensures that internal consumers of vehicle data do not have to take the data source into account. Instead they can rely on a abstraction that delivers them a standardized, prepared dataset. The platform must ensure that different protocols can communicate with the existing backend. In this way the services can communicate with vehicles in a abstract way.

1.2.1. Research Questions
The main research question for this thesis is:

"What architectural setup would provide both scalable and flexible solutions for integration of vehicle data?"

In this thesis the main research question is answered by first answering the following sub-questions:

1. What is the current state-of-art in EV charging solutions and data integration?
2. What elements are required to set up a system for the integration of vehicle data?
    - Which actors are involved?
    - Which protocols are used to communicate between actors?
    - Which services can benefit from vehicle data?
3. How can we combine data from different sources?
    - How will we determine the standardized central format?
    - How can we deal with conflicting data?
    - How can we deal with complementing data?
4. How can we develop an architecture that is able to deal with future growth?
    - How can we develop a software architecture that is flexibly extensible with new data sources?

- How can we deal with new services in the backend that use vehicle data?
- How can we make sure that this architecture remains scalable?

5. What lessons can be learned evaluating the developed architecture?

1.2.2. Scope

To reduce the amount of dependencies from external actors, this thesis is focused on the design of a software architecture that supports services in the existing backend by combining several internal and external data sources. This would be described as the 'Integration'-phase in the process of getting all kinds of data from different sources to many different services. In the future new data sources and services can be flexibly added to the integration layer.

1.2.3. Contribution

The result of this thesis is a clearly defined architectural design for the integration of different data sources for vehicle data into an existing back-end for charge pole management. The architecture is unique in the sense that should be able to remain flexible even without knowing what new sources or consuming services will be connected to it in the future.
In the new market of electric vehicles it is important to develop such an universal way to monitor these type of vehicle data sources. The research focuses on the development of a widely applicable methodology to combine data from different sources and integrate it into a generic representation of vehicles.

1.2.4. Approach

This final project is a design research where a software architecture is defined for the integration of vehicle data. The research is divided in four phases. Each phase is discussed in a separate chapter in this report.
These phases are:

- **The Analysis Phase** Chapter 2
  In the Analysis Phase, the EV domain is discussed in more detail. In addition, the various existing and new data sources are discussed and the existing and future services within the Allego backend are described. Based on this analysis, use cases and requirements have been drawn up that the software architecture for vehicle data must meet.

- **The Design Phase** Chapter 3
  In this phase the proposed solution for a model to integrate vehicle data into an existing back-end is described and modeled. Based on the different datasources and services from the analysis phase a design is made. The different elements of this design are discussed in this phase.

- **The Implementation Phase** Chapter 4
  The developed and described solution from the design phase is developed as a proof of concept. In this phase specific implementation choices are discussed. This leads to a prototype which can run in a scalable Microsoft Azure cloud-environment.

- **The Validation Phase** Chapter 5
  In the validation phase the developed architecture is validated. This validation was carried out by means of an expert validation. In addition, the feasibility of adding new services and data resources in the future was tested.

1.3. Allego and research projects

This thesis, conducted at the University of Twente, has been executed externally at Allego. Allego is a charge point operator (CPO) with over 4500 charge points in the Netherlands, Germany, Great Britain and Belgium. The company has approximately 200 employees and four offices. One in the Netherlands (Arnhem), two in Germany (Düsseldorf and Berlin) and one in Belgium (Mechelen).

The company was founded five years ago and has experienced an impressive growth, both in size of the company itself as in the amount of charging poles the company manages.

As a CPO, Allego offers services for the installation of chargers. Allego offers support in requesting the electricity grid connection as well as physically placing the charger. In addition, Allego's IT-backend is able to handle charging sessions and manage charging points. Allego is thus completely relieving its customers in in the management of charging stations.

As e-mobility grows, new challenges are emerging in the field of charging electric vehicles. Charging infrastructure must adapt to the capacity of the electricity grid and at the same time cause the user as little inconvenience as possible. Within Allego, a team is working on a smart charging platform in which it will be possible to determine the optimal charging behaviour on the basis of available capacity and desired demand.

In addition, Allego works closely with other market parties to ensure standardization in the rapidly evolving EV market. In recent years, in this way, together with ELaadNL and various manufacturers of charging stations and other market parties, a standardization of the communication between charging station and backend has been ensured by means of OCPP [10]. Currently, Allego is working on European research projects with manufacturers of EVSEs and vehicles to improve communication with vehicles. Among other things, ISO 15118 is used for this purpose, which ensures that the communication is better secured by means of certificates. The participation in such research projects is important to improve standardization in the market. Currently the research projects are not focused on direct communication between EV and the CPO. It is focused on the communication between EV and charger and the communication between charger and CPO.

2

The Analysis Phase

In the Analysis Phase, the EV domain is discussed in more detail. Various existing and new data sources are discussed and the existing and future services within the Allego backend are described. Based on this analysis, use cases and requirements have been drawn up that the software architecture for vehicle data must meet. Finally, this chapter discusses related solutions within and outside the EV domain.

2.1. Domain

2.1.1. Electric Vehicles

An Electric Vehicle (EV) is a vehicle that is powered by one or more electric motors with rechargeable battery packs. Due to tax advantages, the Plug-in Hybrid (PHEV) was very popular in the Netherlands. However, as from 2017, the tax rules have changed and the number of PHEVs has decreased [11]. In table 2.1 the number of registered electric cars and buses is given.

Table 2.1: Number of registered electric vehicles according to RDW (Dutch Road Admission Authority) [12]

	31-12-2015	31-12-2016	31-12-2017	31-06-2018	31-07-2018
Passenger car (EV)	9.368	13.105	21.115	29.210	30.237
Passenger car (PHEV)	78.163	98.903	98.217	97.946	97.950
Bus	94	168	296	327	352

Based on these numbers, we see a considerable increase in the number of full electric cars (EV). In the case of electric buses, it is clearly noticeable when the new concessions start, often in December.

In order to show the efficiency of electric vehicles, a comparison to conventional vehicles can be made. In figure 2.1 a comparison between battery powered electric vehicles, fuel cell vehicle and a conventional vehicle is given. In order to make the comparison as fair as possible all the vehicles use 100% renewable energy in this example. The efficiency of the whole chain from Well-to-Wheel is much higher with electric vehicles compared to conventional or hydrogen vehicles [13]. Beside the efficiency of electric vehicles, another important advantage is the zero emission during operation based on the usage of renewable electricity.

This comparison is based on the GREET[1] (**G**reenhouse Gases, **R**egulated **E**missions and **E**nergy use in **T**ransportation) model [14] which evaluates energy and mission impact for different fuel types. It allows to evaluate the efficiency of different types of vehicles. Figure 2.1 is based on a research focused on cargo transport but the GREET-software shows that other vehicles have the same characteristics.

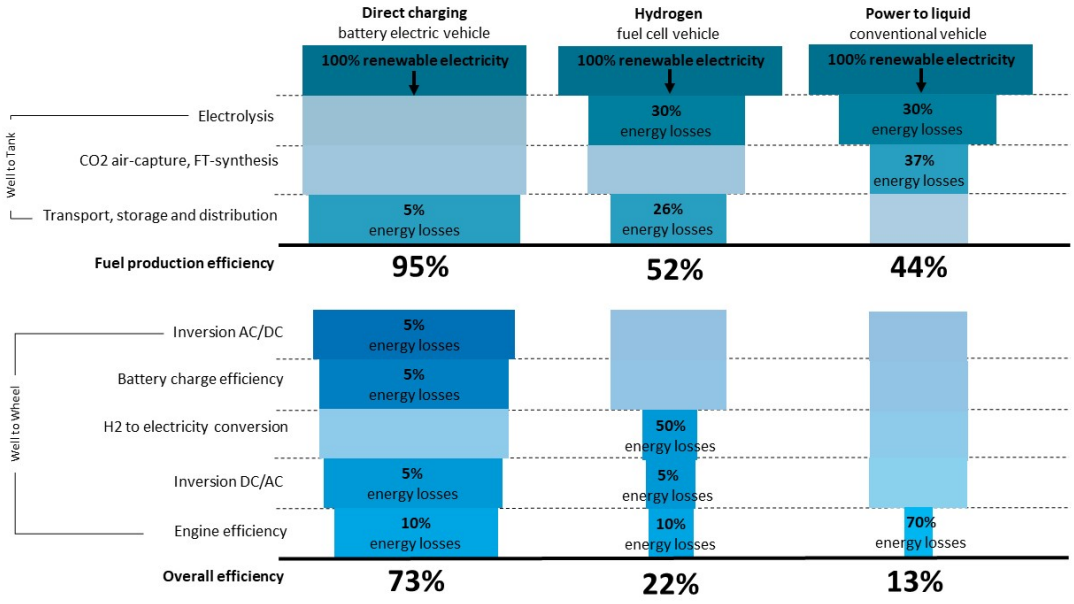


Figure 2.1: Comparison efficiency Well-To-Wheel, based on Ambel et all. [13]

[1] https://greet.es.anl.gov/

2.1.2. Electric Vehicles in Public Transport

In recent years, various pilot projects have been carried out experimenting with electric buses in public transport. One of these projects is the electric fleet in Eindhoven which is used by Transdev as a pilot project for the operation of a fleet electric buses. Transdev is a large bus operator in France which also is the owner of Connexxion, an important bus operator/public transport operator/you name it in the Netherlands. An increasing number of municipalities are setting ambitious political targets for an emission-free public transport system. A goal in The Netherlands is that all new buses in 2025 in The Netherlands fulfill the zero emission norm.[11] In addition to this, the region Brabant has the ambition for zero emission public transport in 2025. This means that not only new buses but also existing buses in 2025 need to be zero emission [15].
In The Netherlands electric buses are used in a regular transport schedule in Eindhoven, Limburg and on the Waddeneilanden (Schiermonnikoog, Vlieland, Terschelling and Ameland)[16]. The Amsterdam Meerlanden concession, started in 2018, has the largest fleet of electric buses in Europe. This fleet consist of a total of 118 fully electric buses [17]. With the electric vehicles available today, the entire operation depends on complex relationships between operation, electric vehicle and charger infrastructure.

Energy Consumption

A very relevant aspect in the operation of a full electric public transport service is the energy consumption of the buses. The energy consumption of an electric bus depends on the size of and weight of the bus. In table 2.2, an overview is given from a test to five different types of electric buses in Slovenia in 2015 [18].
The different vehicles are developed for their own purposes; as an example we take the Siemens Rampini, which has a low range per charge. This can be explained by the fact that this vehicle is developed for in motion charging. As a result, only a small battery pack is required for this vehicle.

Table 2.2: Energy consumption of electric buses in Slovenia

Type	Length	Passenger capacity	Range per charge	Average Consumption	Remarks
SOR EBN 10.5	10.5 m	85	152km	0.8 kWh/km	Electric energy only for driving, diesel heating
AMZ CitySmile 10E	10 m	85	170km	1.1 kWh/km	Operated without passengers
SKD Stratos LE 30E	6.9 m	30	150km	0.5 kWh/km	Electric energy only for driving, diesel heating
Siemens Rampini	7.7 m	46	60km	1.3 kWh/km	In motion charging (panthograph)
Skoda Perun	12 m	81	140km	1.3 kWh/km	

While a conventional bus is able to run a complete shift without having to refuel, this is not possible with electrically powered vehicles. It is therefore important that the charging infrastructure provides the best possible support for the operation of the timetable.

Charging Models in Public Transport

To charge electric buses there are several charging models that can be used.

- **Depot Charging**
  Static charging in the depot during breaks and at night. Charging is possible with up to 150kW. The vehicles will be charged during the night to have range for the next day. For depot charging several connection methods to the charging station can be used: a cable with connector or an pantograph.

- **Opportunity Charging**
  Static charging on-route during the day, usually up to 500kW. With this mode an electric bus can charge during short stops on the route. This on-route charging is usually conducted through a pantograph which connects the bus to the charging station.

- **In Motion Charging**
  Charging a bus powered by electricity obtained from an overhead cable. This is the way of charging used by the trolleybuses in Arnhem. The standard trolley buses have no battery pack, new models make it possible to drive small distance without overhead contact line. This makes it possible to drive to surrounding areas without overhead contact line without major modifications to the existing infrastructure [19]. Currently Allego is not involved in projects with in motion charging.

For the existing projects with electric buses in public transport a mix of these models is often applied. During the night the PTO makes use of depot chargers. In order to have a full battery pack at the start of the operation the buses have the whole night to charge.
In addition to the depot chargers opportunity chargers are installed for the operation during the day. The electric buses can charge on-route. In a 15 minute break the buses can charge to continue their route during the day. With this technology the size of the battery packs can be decreased in order to meet the requirements during the day.

In Eindhoven, for example, the fast chargers (Opportunity Charging) have a socket that is capable of charging at 300kW.
In the Allego backend we can find charging sessions that illustrate this. A session from 11-12-2017 at 10:36 shows us that in 24 minutes the charger used 114.724 kWh to charge the bus. At the start of this charging session the state of charge of the vehicle was 4%, at the end of the session 94%. Effectively the average power is 201.658 kW during this charging session. With an average consumption of 1 kWh / km it is possible to continue the route for another 100km. In Eindhoven there are buses which drive more than 340 km a day [17]. With opportunity charging sessions during the day, it is possible to run the timetable successfully.

An illustration of the charging model is given in figure 2.2 . This is an illustration from Heliox, the manufacturer of the EVSE in Eindhoven. The illustration shows opportunity charging on route and depot charging at night.

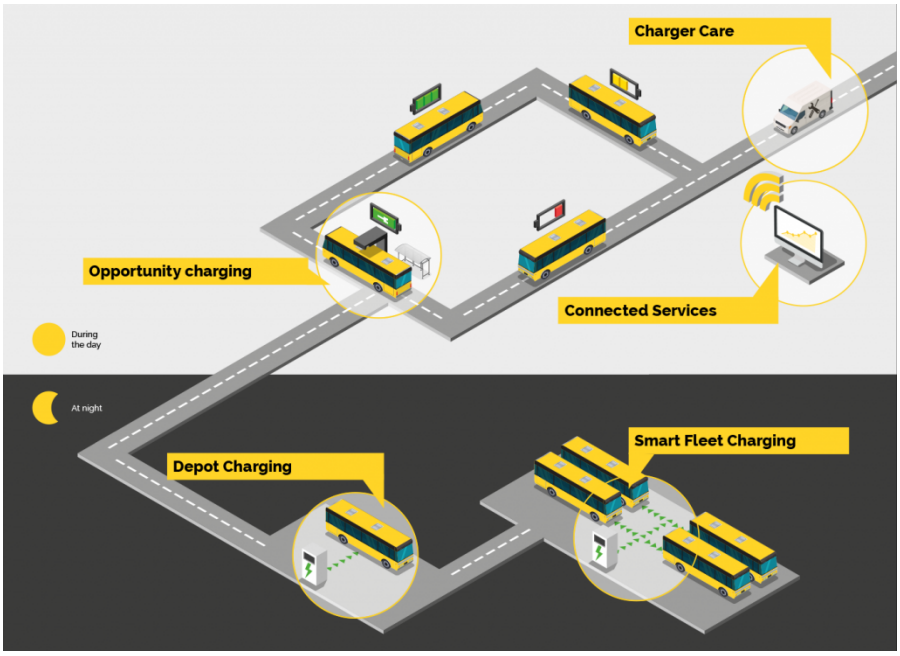


Figure 2.2: Illustration of the Heliox charging infrastructure [20]

In the night the buses are charged with an 50 kW charger in the depot (Depot Charging). In order to be able to charge all vehicles, there is a demand for smart charging (Smart Fleet Charging). The goal of Smart Charging is to optimize the charging sessions with the adjustment of the provided output of a specific charger based on available capacity and charging requirements. During the day opportunity chargers (Opportunity Charging) are used in order to recharge the buses on route. With a backend (Connected Services) all the chargers (depot and opportunity) can be monitored. Session information and exceptions are stored here. Technical assistance can be provided if irregularities are discovered in the backend. (Charger Care)

2.1.3. Standarization in the EV-domain

In the EV-domain, there is a wide variety of different standards. Moreover, there are many different parties involved in charging an electric vehicle. ELaadNL, a Dutch knowledge and innovation center for electric mobility, has developed several models for the communication between the different parties in the EV-landscape in terms of standardization. The different models are focused on the charging infrastructure and how to communicate between the different parties. At this moment there is no universal standard for the communication between OEM and CPO [21]. This can be explained by the fact that until now there was no need for direct communication between CPO and the vehicle. With the advent of new services for which live vehicle data is necessary, data must be exchanged between the vehicle and third parties such as a CPO. Figure 2.3 and 2.4 show the different standards between the parties involved in charging an electric vehicle. For the integration of vehicle data, the vehicle, charging station and charge point operator are especially important. Communication between Electric Vehicle Supply Equipmen (EVSE) and CPO is based on the OCPP-protocol.

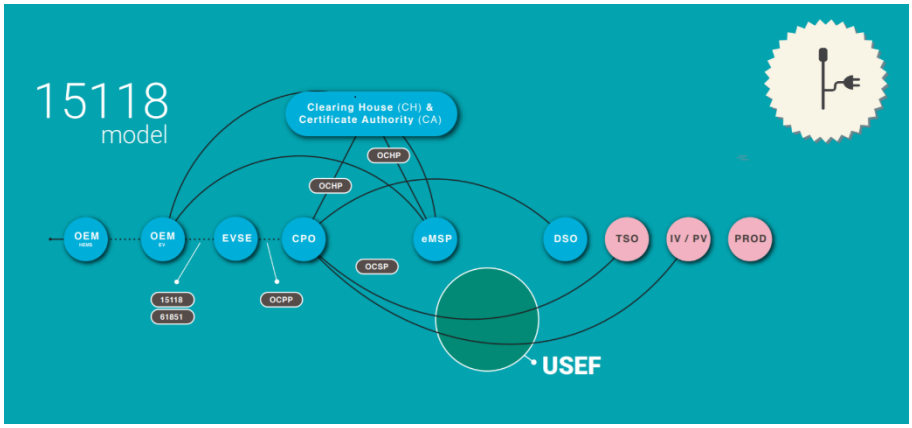


Figure 2.3: Electric Vehicle Landscape - 15118 model [21]

Although there is no direct communication between the vehicle and the CPO, the CPO obtains information via the EVSE. ISO15118 is a communication protocol for communication between the EV and the EVSE. With this protocol process as authentication, authorization, information exchange and billing are managed in a user-friendly way. Figure 5 gives an overview of a charging model in which ISO15118 plays an important role. Communication from the EV/OEM to the CPO is in this case only possible through the EVSE.

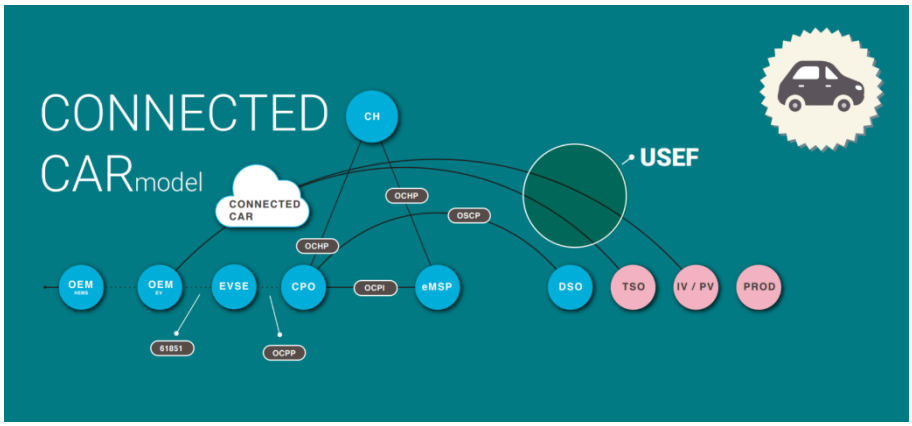


Figure 2.4: Electric Vehicle Landscape - ConnectedCarModel [21]

Another model defined is the Connected Car Model in which the EV communicates with a backend of the vehicle OEM. Through this backend it is possible to provide connections to other parties. This model is illustrated in figure 2.4. In this model, a vehicle is connected when in contact with the manufacturer's platform. This is to make software updates and data exchange possible.

Currently Allego works together with vehicle and EVSE OEMs to support the ISO 15118 model. Currently more and more vehicles become connected with the backend of the OEM of the vehicle, so in the future it will be possible to connect to a connected car platform.

2.2. Market Requirements

In the tenders for the electric buses and infrastructure, the combination of vehicle data with charging station data is often mentioned. Below are a number of examples of input provided from tenders and market consultations.
These are tenders and consultations for services in the EV-domain. These market requirements are the starting point for new services.

- "We will require to create Smart Charging profiles ourselves in our Energy management System" - tender VHH Hamburg Q3 2018)
- "We have to offer a customer portal for white label customers with integrated bus and charger data" - VDL 2017/2018
- "We need an integrated e-bus solution; we appreciate what ViriCity offers and would look for comparable systems, chargers and services" - BSAG possible tender Bremen 2019
- "We require a performance system which can predict Energy consumption and remaining range. This requires intelligent route and topographical energy and power monitoring and GPS input data" - tender Freiburg April 2018
- "The system availability of assets and related to vehicles will be measured with the backend provided by the supplier" - tender VHH Hamburg
- "Central overview for service on vehicle and assets is in that way more important‘ We would not opt for separation of IT systems" - BVG Berlin
- "We expect an integration with our operational busline management to determine delays and critical vehicles" - market consultation 2019 RET
- "We are going to develop a depot management system for electic buses but want to support an open interface to charging backends." - consultant PSI, Berlin (German Tenders)
- Prefers the Viricity solution simple monitoring both asset and vehicle, based on experience with the way fleet data is handled - Osnabrück tender won by Schaltbau/Viricity

Based on this, Allego intends to develop services with integrated vehicle data. These services must be in line with the current service provision and the future wishes of customers.
Beside these market requirements Allego business sees opportunities in the future for the expansion of extra services related to e-mobility and vehicle data. These requirements in combination with several workshops with the public transport team, discussions with customers, and partners have resulted in the services described in the next section.

2.3. Services

Based on the findings from the market, various services have been put together based on input from the Transport as a Business team (TaaB) from Allego. These services can improve their current offering to customers. This relates to the improvement of the core business of Allego and on the other hand to an expansion of the product portfolio. Although this research does not aim to develop the various (new) services, it is good to have insight into the services that will make use of vehicle data. This helps to develop the platform for the integration of vehicle data, as the services will use the platform that has been developed to obtain and process the vehicle data.

The services related to vehicle data are divided into the following categories.

Insight Give the customer insight in their assets and operations. For charging stations Allego already offers the tools to give customers insights. The Allego backend manages the charge stations and corresponding sessions. For a Transport-as-a-Business-customer it is important to have insight into the vehicles. For these customers it is important that the operation of their vehicles can continue, even if a charger malfunctions. The first step is to give customers insights into vehicles, chargers and the impact on the operation. Based on this, a number of services have been defined:

- Location Management Information about state of depots and charger locations
- Vehicle Management State of a vehicle
- Operation Management Insight into the operation (timetable)

Adapt With the combination of vehicle and charging data, it is possible to respond to unforeseen situations. For example, by giving a vehicle priority while charging if it is delayed according to the timetable. These services make it possible for an operator to adapt early, so that as little negative impact on the operation as possible can be realized. There are several models available in literature which can be used to improve and predict the behavior of electric vehicles. [22]. For the batteries, there are models to predict the state of charge available. [23] [24]

- Smart Charging Smart Charging helps to support more chargers in the existing energy infrastructure by adapting the power output to the available power.[25].
- Active Monitoring Active Monitoring is currently used by Allego to monitor their chargers. With vehicle data Allego would be capable to offer monitoring for vehicles as well, but more importantantly, Allego could use vehicle data to improve the current offering by combining the data.
- Charger Reservation Reservation of chargers can help to improve the number of supported vehicles with equal charging infrastructure [26].

Efficiency When reflecting on the performance using the data that is gathered via various channels, it is possible to determine whether it would have been possible to do better; to gain a higher efficiency if other choices had been made. For this, analytical services could be developed in the efficiency service category.

- Predict Failures Predict when a vehicle or charger will stop working as expected.
- Analysis Vehicle Get overview of the historic performance of a vehicle with the help of BI dashboards.
- Analysis Chargers Improve the existing analysis of chargers by adding vehicle data (for example to determine the root causes of failures).

These services will be elaborated on in the following paragraphs. The goal of each of the services is given and a minimum viable product is described. Also explained is what vehicle data is required for the service.

2.3.1. Location Management

The purpose of this service is to provide insight into daily operations at the locations of chargingstations and buses. In a depot, it is important to gain insight into the vehicles that are currently being charged, the vehicles that are stationary and the vehicles that are on their way to the depot.
Currently, in the evening, an employee manually checks the depot to ensure that all vehicles are correctly connected. By providing insight into the current status of the deposit, this becomes superfluous.

MVP A map of the depot/location that shows the chargers and the vehicles in an integrated overview.

Data For this service a minimum requirement is the availability of the position of a vehicle.

2.3.2. Vehicle Management

Vehicle Management is a service which helps operators to get insight into the vehicles and their history. The current state of the vehicle is represented here. Based on this the operator can inspect vehicles remotely in a control-room without physical access to the vehicle.
This service can also help the to have a service and maintenance track record for a vehicle.

MVP Display the current state of a vehicle. Showing the measurements available from different datasources

Data SOC, State, Range, Charging moments.

2.3.3. Operation Management

This service provides insight into the impact on the operation. If, for example, a charger is defective, this can have an impact on the operation. However, when all services work well together, it may also be possible to compensate for this by redirecting vehicles to another charger or using the released grid capacity on other chargers. When the actual operation is in danger, it must be made clear with the help of this service. It is not a problem if the bus has a critical battery value if it is near an (available) charger.

MVP Combination overview of chargers (faulted, occupied) and the vehicle state.

Data Charger data, Vehicle State based on interpretations of vehicle data.

2.3.4. Active Monitoring

Active Monitoring is a service which is currently offered by Allego for their charging stations. Based on real life data the system detects the state of the charging station. When a charging station does not have a connection with the Allego backend an automated reset is performed. A charger can also be faulted. For example during the charging session or a defect component in the charger. Sometimes the root cause of a faulted charger is not the charger itself but the connected vehicle. With the combination of vehicle-data and charger-data the Active Monitoring process can be improved. Active Monitoring for vehicles can be added to the system in a later stage. With an active monitoring system for vehicles, the up-time of vehicles can be monitored and when necessary field engineers can be called upon to solve the problem.

MVP Monitor if a vehicle provides data as expected

Data The MVP will check if the datasources from vehicles provide data. In the first phase the required data are dataupdates (any kind).

2.3.5. Smart Charging

Smart charging is a valuable service in the future, since net capacity is limited and the demand from electric vehicles continues to increase. In order not to overload the grid, Allego could build a service

to support 'Smart charging'. This service would take various parameters as input and generate a personal charging profile for every specific connected vehicle, that takes into account local availability of power, external inputs and constraints given by the EV-driver.

MVP Exchange vehicle data with the smart charging solution currently in development within Allego.

Data Arrival time, Estimated Departure Time, Vehicle Information, SoC at arrival, Grid constraints.

2.3.6. Charger Reservation

For a Public Transport operator, it is important that a charger is available when a vehicle needs to charge. In the current implementations the installed chargers are dedicated for use by the public transport customer. However, there are new requests for multimodal chargers. This means that a charger can be used by cars, taxis, vans, buses and trucks. In the current backend it is possible to configure when a charger is available for which user group (public/private). With the integration of vehicle data it become possible to get a better indication of the charging demands of connected vehicles. A bus operator can make use of multimodal-chargers when the system makes sure that the charger is available when a bus needs to charge.

MVP In the first phase manual reservation of the charger should be possible (by the operator), this could be automated based on predictions in the future.

Data Arrival time, Estimated Departure time

2.3.7. Prediction

Based on the available vehicle data it should be possible to predict vehicle state. Based on environmental conditions and the data from the vehicle it should be possible to predict arrival and departure times.
The predictions made in this service can be used as a input for other services. Not for all vehicles all information is available. Based on the information that is available a prediction could be made for the missing information. For example the SoC can be predicted based on the driven route and historical information.

MVP Predict arrival and departure times, Prediction of SoC

Data Historical data for arrivals and departures in combination with SoC.

2.4. Actors

2.4.1. Business Actors

From a business perspective, there are several actors involved in the process of retrieving vehicle data. These parties should work together to enable successful integration of vehicle data.

- **Charge Point Operator (CPO)**
  The Charge Point Operator (CPO) is responsible for a network of charge points in a certain geographical area. It maintains the charge poles both physically and digitally. It provides the infrastructure for the charging process. The CPO wants to extend the current platform with vehicle data in order to improve current services. Beside this vehicle data can help to extend the business and improve the market position of the CPO.

- **Vehicle Manufacturer (OEM)**
  The Vehicle Manufacturer produces a electric vehicle. In addition to the development and production of vehicles, a vehicle OEM is often engaged in service and maintenance. This manufacturer is therefore not always willing to work together to share vehicle data. It is important to clarify the practical application of vehicle data exchange. For a manufacturer, the main focus here is on improving the availability of the entire infrastructure. After all, if a bus is unable to charge it (for whatever reason), the OEM will be indirectly blamed for this.

- **Fleet Owner**
  The fleet owner is the owner of the vehicles. In the traditional setup with fuel buses, a vehicle can be refuelled only once all day long. When electric vehicles are used, they are charged several times during the day. This ensures that the fleet owner wants and needs to have much more insight into the current status of his fleet. After all, a small disruption can have major consequences.

- **E-driver**
  The e-driver is the driver of the electric vehicle. In the scope the e-driver are the bus drivers of public transport busses. In the future it could be possible that a e-driver is a driver of car who is also the owner of that car. He than can opt-in to exchange vehicle data with the CPO in order to improve his/her charging experience.

- **Equipment Manufacturer**
  The Equipment Manufacturer is the party that develops a tracking device. Depending on the situation, a third party device is required to extract vehicle data from the vehicle. In the ideal world, this actor is not required and the feed data is obtained directly from the fleetowner or vehicle OEM. However, it is not always possible to enter into direct cooperation with a vehicle OEM. In addition, the fleetowner does not always have the necessary vehicle data at his disposal. A tracking device can be placed in order to obtain vehicle data. This is where the Equipment Manufacturer is positioned, this is the company that produces the tracking device.

2.4.2. Software architecture

In the scope of the software integration there are two types of actors involved, providers and consumers.

- **Provider**
  A provider is an actor that supplies vehicle (related) information. All the different datasources are a data provider for the software architecture. A data update from a provider leads to an update in the platform. Such an update could lead to several sequential actions.

- **Consumer**
  A consumer is an actor that is able to receive data from the platform. It is able to respond on the actions. An example of a typical consumer is the Smart Charging Service which retrieves information from the platform in order to be able to adapt the charger behavior on the vehicle status.

A data source is typically a provider of data. However it is possible that a data source also responds on updates from the platform and acts as a consumer. A example of a data source that can also act as a consumer is

2.5. Data Sources

In order to get an as complete as possible picture of the vehicle, data from multiple sources must be combined. This section contains a number of data sources that would be suitable for providing relevant vehicle information.
In the ideal world, the information can be obtained directly from the source. There would then have to be a direct exchange of information with the vehicle. Currently, there are initiatives to facilitate such data exchange[8]. Also, some initiatives are developing plug-and-play retrofit devices and backends for vehicle information. [27] However, there is not yet an unequivocal way of steering this. Within the Public Transport sector, the ITxPT consortium is working hard to standardize IT systems so that they can fit together properly. [27]

This section provides more information about the current protocols and possibilities for exchanging data.In the first phase of this study, we looked at various solutions for obtaining vehicle data. These solutions are described in broad outline in Appendix A.

2.5.1. Government provided information: OpenData

In the context of OpenData, there are various sources within Europe that offer some information with regard to public transport. Within the Netherlands, the NDOV-loket provides information with regard to the timetable and the live positions of vehicles in operation. By using these publicly available data sources, it is possible to determine at least the position of vehicles and use this as input in the data integration platform.

2.5.2. Vehicle OEM provided information

(r)FMS protocol

The Fleet Management System Standard (FMS) is a combined standard (data protocol and hardware standard) defined by manufacturers of trucks and busses to give third parties access to vehicle data. With this interface it is possible to retrieve information from the ECUs without direct connection to the CAN bus. The bus OEMs involved in this project are: Daimler Buses, MAN Truck & Bus, Scania CV, Volvo Bus Corporation, CNH IrisBus and VDL Bus International. Currently, Allego uses both VDL and Volvo buses on their chargers. However, in the near future, more mixed fleets will be created in which different OEMs will jointly use the charging infrastructure.
The FMS standards for trucks and buses are developed in 2004. A common connector is added in 2009. This makes it possible to for third parties to gather information from vehicles with a manufacture independent solution. Version 4 of the FMS documentation, released on 13-10-2017 contains measurements which are important in the EV domain. Most important is the “Hybrid Battery Pack Remaining Charge” which gives the SoC of a vehicle. With the FMS protocol it is necessary to have a device physical connected to the bus in order to retrieve the data from the bus and send this to a server. Since most of the buses already are connected vehicles the industry has developed a new extension to the existing standard in order to make it possible to retrieve information through the server of an OEM. This new standard is called rFMS (remote Fleet Management System). In figure 2.5 the two different versions of the FMS-standard are illustrated. This figure shows schematically how the data is linked to the Allego cloud.

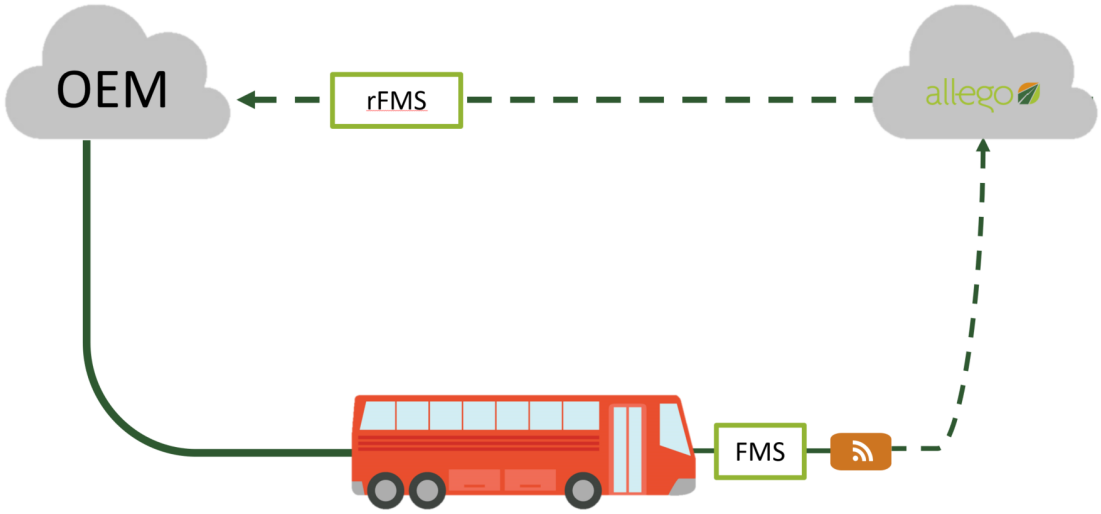


Figure 2.5: rFMS and FMS connection in relation with the Allego Cloud

In addition to the connection between the vehicle and the vehicle's OEM, PTOs often also use vehicle data in their planning systems. They make use of on-board computers such as the devices from Groeneveld ICT for monitoring their fleet on the one hand, and on the other hand to be able to communicate with the drivers. The FMS and rFMS interface are examples of a standardized protocol defined by the OEMs. However not all manufacturers have a FMS interface and these standard is developed by West European manufacturers. The current EV market for buses and large goods vehicles also other OEMs are involved. For example the Chinese BYD has several buses in operation in the Netherlands.

Car Connectivity Consortium

The Car Connectivity Consortium tries to build a Car Data Ecosystem [28]. This consortium earlier developed MirrorLink which is a standard for the integration of Android smartphones into the infotainment system of the vehicle. With the Car Data Model they try to create a universal standard to exchange vehicle data [8]. This model is mainly interesting for the future where multiple parties would like to retrieve information from the vehicle. As illustrated in figure 2.6 a standardized interface could help to extract data in a universal way.

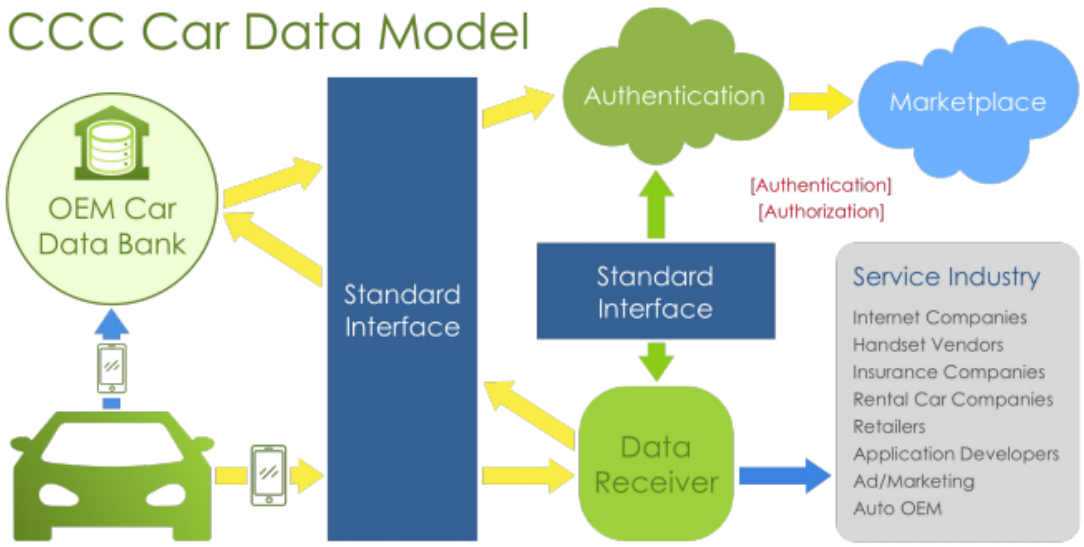


Figure 2.6: CCC Car Data model

Allego as Charge Point Operator is a player in the Service Industry field and is in this case a data receiver. The universal model is not developed yet but could be in the future a way to extract data from the vehicle with for example mobile apps and integration with the databank of the OEM.

2.5.3. Tracking Device provided infomation

There are different manufacturers of tracking devices. These devices are are connected to the vehicle and send positional data but also vehicle data to a external server. During this research we have made use of a Teltonika tracking device which can be connected to the vehicles can-bus and or fms-bus in order to retrieve information from the vehicle. It communicates with a server with a socket connection where the device connects to a configurable endpoint where it sends its updates to.

2.5.4. CPO Platform provided information

The existing backend platform from Allego gathers information about vehicles in order to support the charging process. This information can be used to build up the internal vehicle representation. For example, during the loading session, the course of the State of Charge can be read out at certain charging points. In addition, it the charger location is for used chargers is fixed. This existing data can be used to determine the position of the vehicle during a charging session.

2.5.5. Environmental information

In addition to information directly related to the vehicle, data sources that provide additional information are also important. This may include conditions that may affect the operation of the bus. Examples of this are:

- **Weather Data**
  Various open sources such as the KNMI in the Netherlands provide detailed information on current weather conditions [29]. These conditions can be important for e.g. range-estimation.
- **Traffic Information**
  The Dutch government published a database for the current situation on the road, both actual and historical. This is done by the NDW, the National Data Warehouse for Traffic Information.[30]

This information can be used by the road authorities to redirect traffic with route information panels and other traffic management instruments. The historical record of real-time traffic data can be used to generate traffic analysis. Based on this historic models combined with for example the energy consumption of electric vehicles a model can be developed in order to determine the expected energy consumption. This data sources can be very useful for predictive analysis. The NDW database contains the following data: traffic flow, realized travel time, estimated travel time, traffic speed and vehicle classes.

2.6. Scenarios

This section describes different scenarios for vehicle data. The aim is to provide support for this type of scenario within the final platform. These scenarios were created by analyzing the different data sources and services.

2.6.1. Combination of data

A typical scenario for the integration of vehicle data is combining different datasources which are delivering the same data. A example of a use case for this scenario is the determination of the vehicle location in a depot. Based the GPS location from a tracking device it is not possible to determine the exact location.
However, when a vehicle is loading, the exact location of the vehicle is known. This makes it possible to determine the location more accurately than using the GPS position.

In the future, sensors could be added to the depots in order to accurately measure where vehicles are located. This would form a new data source that is more accurate than the data from both the charging station and a GPS tracking device.

2.6.2. Data delivery

Not all datasources are delivering information continuously. For example the NDOV-loket only gives data about a vehicle if the vehicle is in operation. In this case 'in operation' means that it is connected to a line. The chargers only deliver information about vehicles if they are connected to a charger. Many of the data sources used are therefore not capable of constantly providing information. Within the platform it must be possible to process these data sources anyway.

2.6.3. Fetch other data

Based on incoming data, other data can be fetched from other data sources. This mainly ensures that relevant extra measurements can be retrieved. Based on the current position, additional data can be retrieved. A number of examples of concrete applications are given below:

- **Weather data**
  Based on a location update of a vehicle, weather information can be retrieved. This weather information can be used to predict, for example, the battery capacity of a vehicle. After all, its performance depends on temperature [31].
- **Traffic**
  Road conditions make it possible to make more accurate predictions about the expected delay and thus the estimated time of arrival at a loader.
- **Geocode**
  Based on a position, additional information about the location can be retrieved. With the aid of geocoding it is possible to obtain a textual representation of the location.

The retrieval of additional data from other sources can help to achieve a more complete vehicle status that allows various services in the Allego platform to respond.

2.7. Requirements

In order to give a clear overview of the expectation to the software architecture that is developed in this thesis, requirements have been defined. The requirements cover both functional and non-functional aspects of the software product that is to be the result of the design and implementation phases in this research.
The requirements here are developed based on the current scope of the services and datasources which are supported by the platform. Based on discussions with customers, partners and internal guidelines for new developments within Allego these requirements are drawn up.

2.7.1. Functional Requirements

1. The system should provide a mechanism to install a new data source.
2. The system should provide a mechanism to combine data from different data sources related to one vehicle.
3. The system should provide a mechanism to deal with conflicting data from different data sources.
4. The system should provide a mechanism to dynamically respond on incoming data depending on the vehicle.
5. The system should provide a mechanism to keep track of historic vehicle data.
6. The system should provide a mechanism for other services to retrieve vehicle updates.

2.7.2. Non Functional Requirements

1. The system should be scalable, to support thousands of vehicles at the same time.
2. The system must make use of the Microsoft Azure Platform.
3. The system should be designed in such a way that it is high-available. Downtime seriously effects business.

The requirements are verified in the Validation Phase of this research on page 47.

2.8. Related Work

2.8.1. Architectural Patterns

Middleware
For the integration of different in/output streams a middleware architecture can be used. Middleware is applied as an abstract layer between, for example, an operating system and various software applications. [32].
One application that has common ground is the use of middleware in sensor networks where different types of hardware are linked and integrated. [33] Usually middleware focuses on the integration of several different systems into a single system. In the case of vehicle data for Allego, it is not necessary to connect to a single system, but multiple data sources must be connected to multiple services. Within the platform to be developed, certain principles of middleware will come back. One can think, for example, of the way in which abstraction takes place within a middleware solution.

Rule Engine
In recent years, software solutions have been made possible by means of Model-Driven Engineering (MDE) [34]. By means of a graphical interface it is possible to build applications in this way. An example of a software system that makes this possible is Mendix, where users can build their application visually. [35]
Following on from this, there are several standards that make it possible to define rules when interacting between different software systems. Think of BPEL and biztalk, for example. BPEL is focussed on Web Services and is designed to orchestrate the flow of a business process. [36]
With BPEL it is possible for business analists to define processes in BPMN which can be converted to executable BPEL. Both Mendix and BPEL ensure that business users can be more involved in the development process. However, they do not facilitate the actual integration of different data sources and services. . These standards are very powerful and comprehensive, but are mainly intended to allow well-defined applications to talk to each other. A business-user needs to be trained in order to be able to create new rules.

Binding Pattern
A binding pattern is used in recent developments. Different services are linked together so that new resources can be added during runtime. In practice, this pattern is applied within home automation (see section 2.8.4) but also within the Azure Cloud platform for calling up/triggering serverless code. In addition, there are platform specific solutions such as OSGi [37] which, based on the dynamic component model, allow components to be added and bind to the platform during runtime.

2.8.2. Smart Connected Product
The integration of external data into a CPO platform can be seen as the development of a smart connected products. For the development Porter and Heppelmann describe “The new technology stack”[38].
When adding vehicle data to the existing backend, there are multiple datasources. This deviates slightly from the scenario described by Heppelman et All. However, many principles are still easy to apply. For example the capabilities for smart connected products match the Goals (Insight, Adapt, Efficiency) of Allego:

- **Monitoring**: with sensors it is possible to monitor the condition of the product, it’s environment and it’s operation and usage.
- **Control**: software in the product cloud enables the control of product functions and the penalization of the user experience.
- **Optimization**: based on the monitoring and control capabilities the product performance can be enhanced and predictive diagnostics for service and repair becomes possible.
- **Autonomy**: with the combination of monitoring, control and optimization it become possible to create a full autonomous product.

In the existing backend a product (the charging pole) is made smarter by adding connectivity to it. When vehicle data is added, a new smart connected product is created. However, information can also be transferred here, the charging point functions as a source of information for the vehicle and vice versa.

2.8.3. Allego
Within the existing infrastructure within Allego, a number of applications can be found that perform tasks for chargepoints that can also be found within the vehicle integration platform.

- **Active Monitoring**
  In the implementation of active monitoring for Allego the platform monitors incoming messages and checks if the message flow is as expected. Responding to data within vehicle data shows similarities with the monitoring of charging points within the current infrastructure. In addition, Active Monitoring is a relatively new service for Allego, partly because of which a lot of serverless components have been used to guarantee scalability. For this project, we therefore looked at the structure of Active Monitoring.

- **Communicator**
  The communicator takes care of all the communication with charge points. The communicator is the entry point for charge points in the IT landscape. When a service in the platform needs to send data to a charge point this data is sent via the communicator. With the vehicle data multiple new communicators for the different protocols are designed.

2.8.4. Home Automation

Within the home automation market, there are a number of solutions that make it possible to combine different systems. These principles can also be applied to vehicle data. Status updates from a vehicle are comparable to status updates from home automation devices. This is the reason that it is interesting to see the solutions in the home automation market. A number of the solutions for home automation will be discussed in this section.

OpenHab is a platform which helps to connect different protocols for home automation in order to be able to control a smart home from one central point. It ensures that the different stand-alone systems can communicate with each other [39]. For this purpose, an abstraction layer is used that ensures that a light with protocol x can be controlled through protocol y. With this platform it is also possible to have certain actions take place when arriving at a certain location.
Although the domain of vehicle data is very different from that of home automation, there are also similarities. Within the new platform to be developed, we want to be able to respond to events from different data sources. In addition, the type of data being treated is very similar; the battery status of a vehicle is not much different than a temperature status in a smart home. OpenHab can be used for home automation, but also for security networks. For example, OpenHab is used to read out smart meters. [40].

In general, OpenHab's architecture (see figure 2.7) is built around a service bus. Each specific communication protocol has its own binding that ensures the translation between protocol and platform. Each binding can send a status update to the event bus from which the OpenHab repository then responds. When a light needs to be switched on, a command is placed on the event bus. The binding can control the lamp in question and takes this command and ensures that the correct signal is sent to the light bulb.

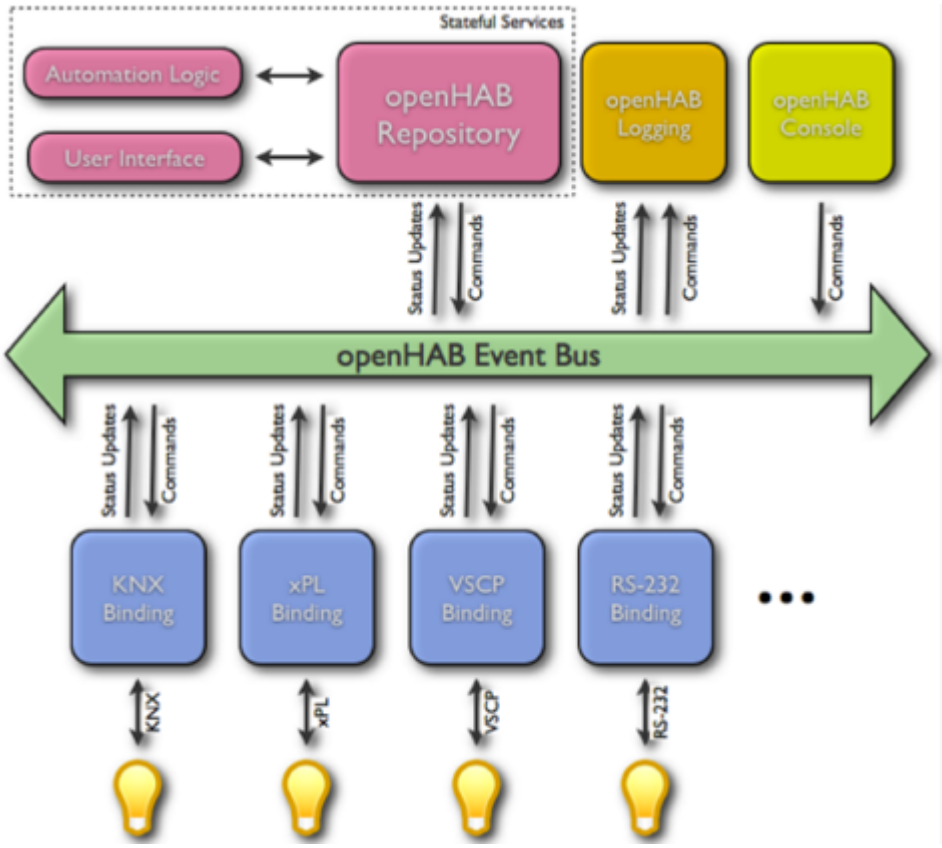


Figure 2.7: Overview of the OpenHab architecture [41]

Another service that is often used for automating tasks is IFTTT. IFTTT is a web-based service which let users create, customize and enable chains of conditional statements, which are triggered by existing webservices.[42]
This service shows similarities with the objectives around the integration of vehicle data. After all, we want to respond to incoming events.
The IFTTT model has already been applied to the integration of various home automation applications by Vorapojpisut [43]. This same principle can be used to integrate different sources of vehicle data. The basic principle behind IFTTT is simple, but it will not always be enough. This is because it is not possible to combine the input from multiple data sources [44]. With Zappier as a alternative for IFTTT it becomes possible to create a chain of actions performed on a data update. This allows to handle more complex situations.
Another example of the combination of different sources in a single platform is Homey, a development by the Dutch start-up Athom. This device makes it possible to connect to different home automation sources. With a simple flow editor end-users are able to define flows which are performed when a certain action occurs. For example put the lights on when someone comes home. Figure 2.8 gives a example of the floweditor.

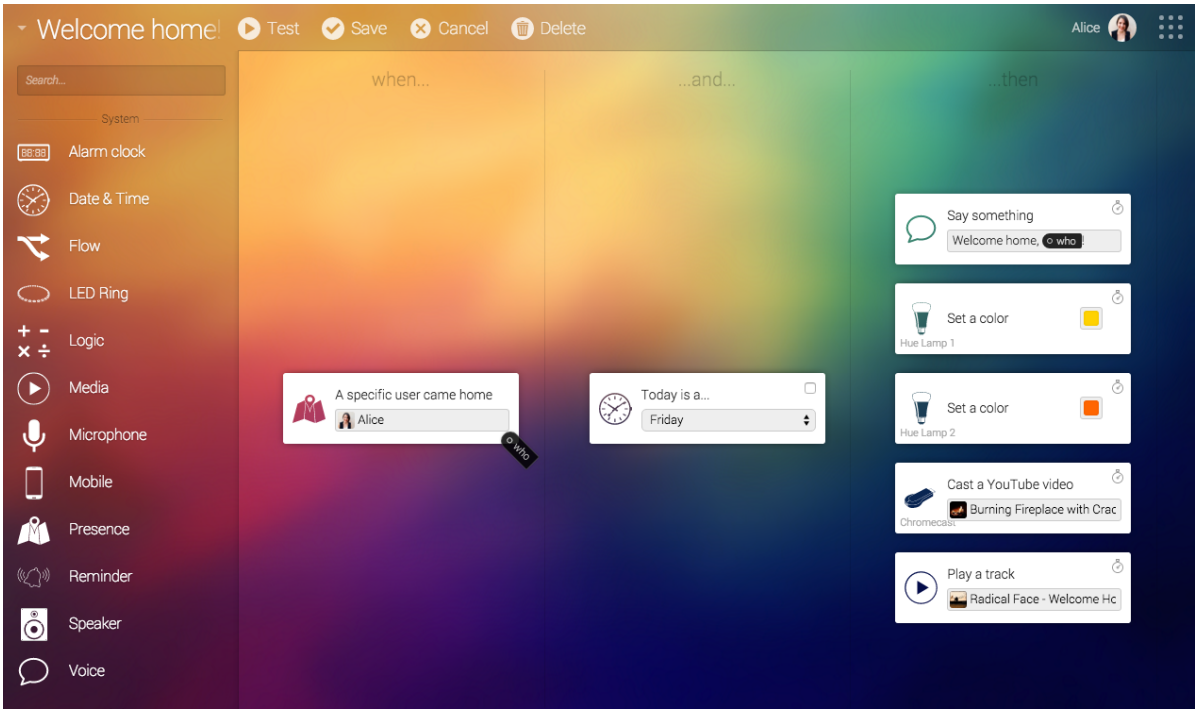


Figure 2.8: Homey flow editor

Apple, Google and Amazon are also entering this market, each with their own product making it possible to combine different smart applications. A voice assistant is used to assist the user in operating his/her devices [45].

2.8.5. Microservice Architecture

Krylovskiy made a design for a Smart City Internet of Things Platform with a microservice architecture [46]. There are similarities between such an application and that of vehicle data integration. In both cases, data of different kinds is combined in one platform. In the case of test data, this concerns different types of vehicles, but also different organizations that provide the vehicle data. As a result, a multitude of different protocols are used. A wide range of sensors must be connected within the application of a smart city. The platform developed by Krylovskiy deals with a changing IoT environment which is able to adopt new technologies an requirements over time. In order to adapt to a microservice architecture new technologies are used. In order to create a scalable solution serverless components are used. [47]

3

The Design Phase

In this phase the proposed solution for a model to integrate vehicle data into an existing backend is described and modeled. Based on the different data sources and services from the analysis phase a design is made. The different elements of this design are discussed in this phase.

3.1. Architecture

As the analysis phase has shown, there are many different extraction methods to obtain vehicle data. Unfortunately, it is currently not yet possible to enforce a single standard within the market. In the future, there will always be a mix of different standards and protocols. . In order to deal with this, an architecture has been developed that makes it possible to integrate different data sources into one backend system. This abstraction layer makes it easier to replace or add a data source.
Within this section the architecture will be discussed. This architecture is the result of a combination of multiple real-life applications in other application areas (Homey, IFTTT, Sensor networks) and the application of existing patterns such as microservices and middleware solutions. None of the existing solutions can be directly applied to the integration of vehicle data, but their principles can be used.
The Allego backend is developed based on a microservice architecture. An overview of this architecture is given in Appendix B. This principle is used by designing the architecture for the integration of vehicle data. Several (micro)services in the architecture need vehicle data which is provided by other microservices. Each data source has its own microservice connector and each service in the platform has its own connection to the vehicle data service.

Figure 3.1 gives an overview of the integration of various internal and external services. Based on the platform it is possible to combine a variety of data sources. For this, we introduce the concept of a *Binding*. This is the same concept as already used by Azure Functions [48] and OpenHab [39]. Each data source has its own *Binding*. By means of the *PathEngine* it is possible to process the data dynamically. Dynamic means that the data can be treated differently for different vehicles. It is possible for users to configure the steps to be taken. As a result, business rules can be added by business users. More about the structure and responsibilities of a binding and the PathEngine are discussed in the next sections.

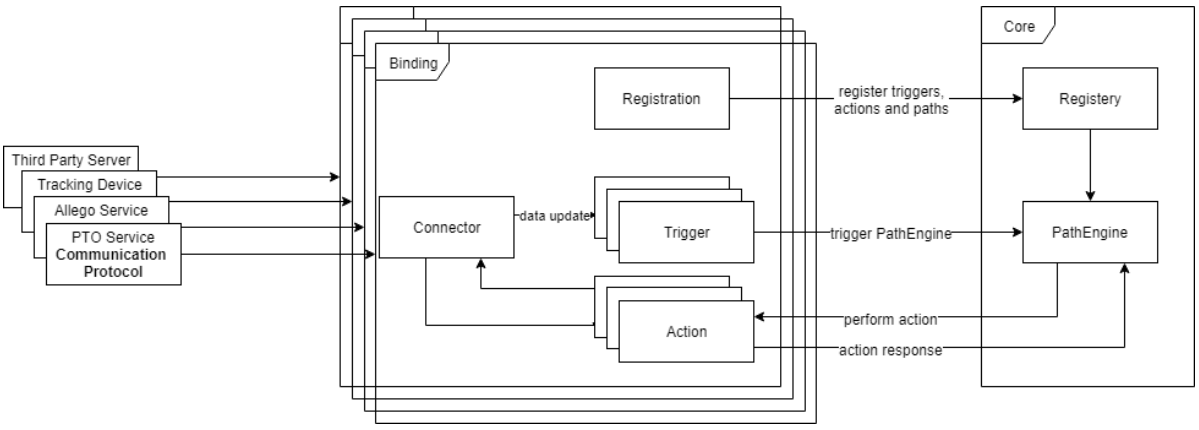


Figure 3.1: Overview of the system architecture

3.1.1. Binding

A binding is an independent software service that can communicate with third parties, devices and other services (for instance in the Allego-backend). Examples of these third parties and devices are NDOV-loket, the vehicle OEM or a tracking device. The concept of bindings will also be applied to other data sources in the implementation phase. A binding always contains four types of parts: the communicator, trigger, action and registration. These parts are explained in the following paragraphs.

Connector
The connector module takes care of all the communication from and to the data source. The connector module has been developed specifically for each protocol. Based on the protocols investigated a number of generic principles can be distinguished.
For the communication use can be made of push and pull based models:

- **Push** In a push-based protocol, the external party will notify its update when it is available. Based on this, the message will be further processed. The vehicle data from the NDOV-loket is a example of a push based protocol.
- **Pull** In a pull-based protocol, the connector checks the availability of new data at fixed intervals and retrieves it when available. An example of a pull-based protocol is the GTFS-standard. Files are updated on a server, they can periodically be pulled from this server.

A push-based protocol has an advantage over pull-based protocols. With a pull-based protocol a binding uses resources to periodically check for new updates from the data source. This means the system will also check (and use resources) when no new data is available and thus consume resources unnecessarily. However, the Allego platform is often only a consumer of data and can therefore exert relatively little influence on the way data is sent by data providers.

The communication of a connector can take place according to the following principles:

- **One way** Data updates only take place from the external party towards the backend platform.
- **Bi-directional** With bi-directional communication it is possible to send data back to the sending party. This can include acknowledgments as well as commands that need to be executed. This could involve feedback to the driver or even intervention in the operation.

It depends on the goal of a binding if a bi-directional data connection is necessary. When we want to influence behavior in the vehicle, a bi-directional connection is necessary.

Within the current set of different connectors, a number of different communication protocols can be distinguished.

- **Socket** When using a socket, a direct connection is established with the external party. This method is often used in tracking devices. When configuring such a device, it is possible to specify a server address to which the connection will be made. There is thus a direct connection which can be communicated in two directions.
- **HTTP** There are several protocols based on HTTP. Examples of this are implementations in which use is made of a SOAP. For vehicle data, this applies, for example, to the rFMS protocol in which an API is available on the basis of HTTP for exchanging data.
- **PubSub** The publisher/subscriber model allows an end user to subscribe to new data updates. When an update is available it will be pushed to the subscriber.

Most of the time a binding is dependent of the protocol specification for a certain communication protocol. However a socket or pubsub connection are statefull. There are more challenges to make this kind of connections scalable.

For every new binding it depends on the protocol specification how the communication is handled. Therefore new bindings will have their own connector with the datasource.

Trigger

A trigger is a part of the binding which gives a vehicle/data update to the path engine from where it can be sent to the rest of the platform. When a data update is received, it is transformed into the right format and then sent as an event on basis of which the platform is able to react.

A trigger works according to the principle of Fire-and-Forgot, which means that a trigger sends an update to the rest of the platform and does not check if and how the message is handled. In the developed design it is possible that no reaction on a trigger is necessary. With the triggers a universal EventStream is created which is handled by the core application.

The trigger types have been defined during the development of a new binding. A type defines which data elements are available. In addition, in this configuration it is indicated what the object type is that this trigger is about. A trigger can, for example, relate directly to the vehicle but also to, for example, a certain bus line.

Action

An action can be used in a path (described later) to convert or respond to data based on a certain trigger. Two different types of actions can be distinguished: the commands and functions:

- **Command** A command is a one-way process. The action accepts input but does not provide any feedback. Examples of commands: sending an email, sending a text message or updating a vehicle characteristic (e.g. low battery = true).
- **Function** In addition to input, a function also has output values. An example of a function is checking the current weather on the basis of vehicle position. The current weather can be retrieved from the weather-binding. The weather-binding is responsible for whether or not to cache weather data.

Registration

A binding itself has the responsibility to register with the core application. The purpose of this is that the core application knows which bindings (and thus which triggers and actions) are available. The following characteristics are included in the registration of a binding:

- **Dependencies** Defines the other bindings on which this binding depends. For example, if the binding defines a standard path that performs actions in other bindings.
- **Triggers** Defines the triggers generated by this application. The expected outputs are defined for each trigger.
- **Actions** Indicates the actions that can be performed by the binding. An action can have inputs and/or outputs that are defined in this configuration.
- **Paths** The paths are installed during registration of the binding defining the standard flows. A binding for a datasource will provide a path to define a standard flow for data updates. While a service subscribes with a flow on updates in vehicle state.

In order to keep registered in the core system a binding needs to re-register itself. The timeout for the registration can be defined on initial registration. A critical binding must re-register itself for example every minute.

3.1.2. Core

Paths

In order to be able to process and respond to vehicle data, a path can be defined. A path is a chain of actions that follows a trigger. These actions can lead to subsequent actions being influenced or even no longer being carried out. In figure 3.2 a simple example of a Path is given.

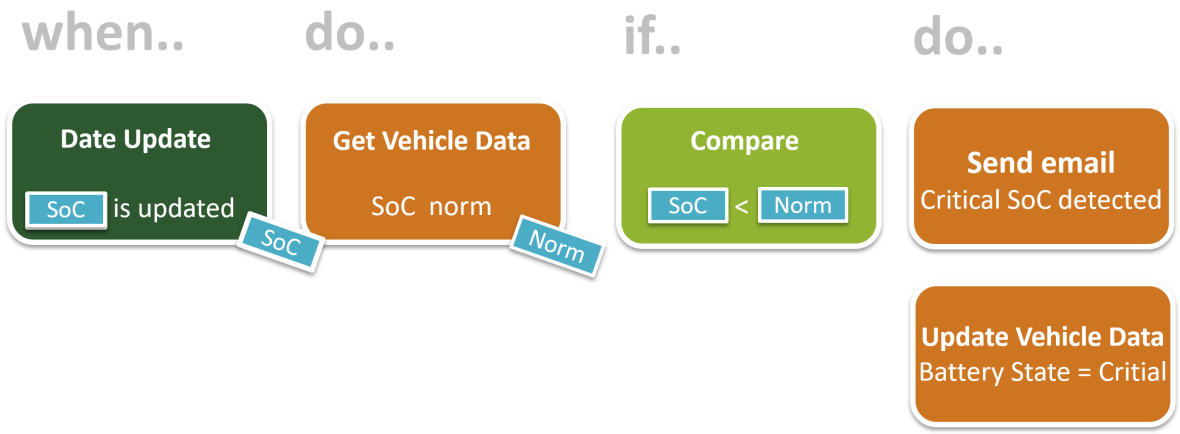


Figure 3.2: Example of a trigger flow

- **When (Trigger)** A path is activated when a binding update causes a trigger to be sent to the system. Each path starts with a certain trigger on basis of which the next steps are executed. The example path in the figure below reacts to an update of the State of Charge of the vehicle.
- **Do (Action)** An action that retrieves the current SoC norm from the vehicle service. This makes it possible to have a dynamic standard that differs per vehicle and even changes during the day.
- **If (Action)** An action that returns a single Boolean value can be used as an if statement. The rest of the Path will only be continued if the condition is met. In the example below, the State of Charge must be smaller than the norm. This standard has been dynamically raised in the previous action.
- **Do (Action)** A follow-up action can be informing the end user via email. In addition, the vehicle's battery status can be adjusted to 'Critical'. Based on this last change it is possible that another Path is triggered.

As the example described demonstrated, it is possible to carry out several actions in a row and use them to create a chain (path).

3.1.3. Contracts

The communication between a binding and the core platform is defined in a contract. This contract specifies the format for the messages exchanged. This contract can be seen as a internal protocol. This protocol defines also the available types of measurements. Comparable to the primitive types in a programming language.

3.1.4. Repository

The repository keeps track of all triggers, actions and paths available in the platform. Based on this the available triggers and actions new paths can be added to the system.

- Paths
- Triggers
- Actions

The basis for this repository is a DocumentDatabase in which the registered bindings are registered. A binding has the responsibility to register itself in the core repository. The uninstallation of bindings is handled by the repository to disable the binding when the binding has stopped sending a registration request.

3.1.5. PathEngine

The PathEngine ensures that the correct Paths are executed on the basis of an incoming Trigger. When a Trigger arrives from the Repository, the correct paths are requested, which are then executed (independently of each other). The PathEngine ensures that incoming messages are processed correctly. This is done in the order defined in a Path. This ensures that processes are carried out in the correct order. An action gives feedback to the PathEngine after which follow-up actions can be carried out. Figure 3.3 gives a schematic overview of the PathEngine.

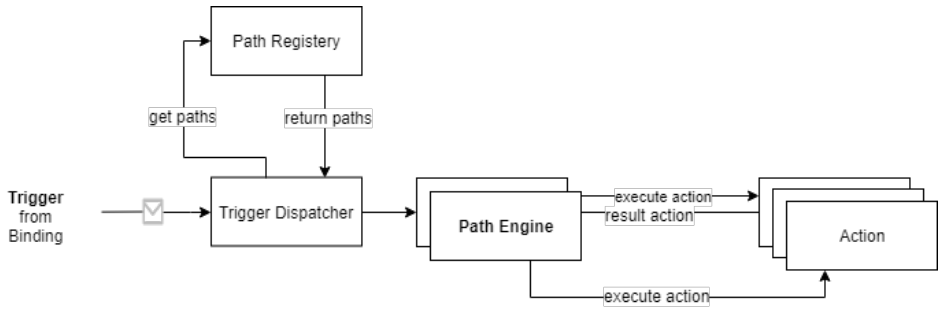


Figure 3.3: Overview of path engine

3.2. Receive data from datasource

Figure 3.4 gives the standard dataflow for incoming data. The process starts when a binding receives or fetches a data update from the data source. This data update is formatted according to the protocol from the data source. It is converted within the binding so that the core platform can process it correctly. Within the core platform, the repository retrieves which actions must be carried out to correctly process the incoming message. These messages are then outputted. An action always gives a response back to the core platform. However, it is also possible that an action may change the status. This ensures that a new data update takes place, which is processed independently by the platform.

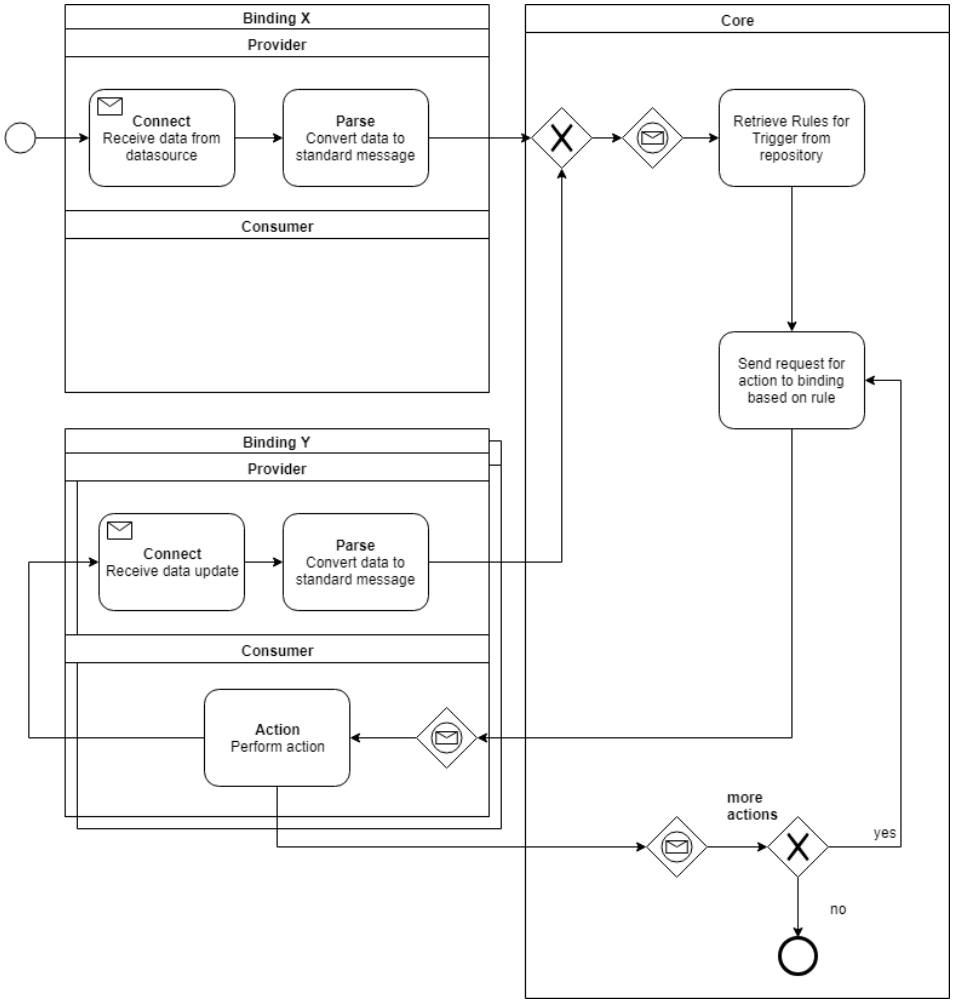


Figure 3.4: Process incoming data

In figure 3.4 a binding is splitted into a consumer section and a provider section. The provider connects to the datasource to receive data updates. These updates are send to the core platform as a Trigger. The consumer section receives a request from the core platform and handles that request. In the consumer section actions registered in the platform are handled.

3.3. Approaches to vehicle data integration

For the integration of vehicle data different approaches are considered in order to force actions are performed in a certain sequence. These different approaches are visualized in figure 3.5.

- **Messageflow**
  Messages are sent directly from the trigger event to the appropriate action. From there the message is sent directly to the next action. The advantage of this approach is that there is direct communication between the different actors in the system. However, all the actors in the platform must be aware of the other actors and their location. Furthermore every trigger must know the which actions are performed. This introduces new synchronization challenges.

- **Orchestration**
  With a orchestration component all the messages are send to one or (when scaled up) multiple instances which are handling the incoming messages.

- **PathEngine** With the PathEngine approach for every message (trigger) a new instance is started. This instance is responsible to process that message (trigger). By making use of serverless components it is possible to develop a scalable solution. The result of a action can be routed back to the appropriate PathEngine. When there are multiple Paths based on one trigger different instances of the PathEngine are started up in order to process the Path in parallel.

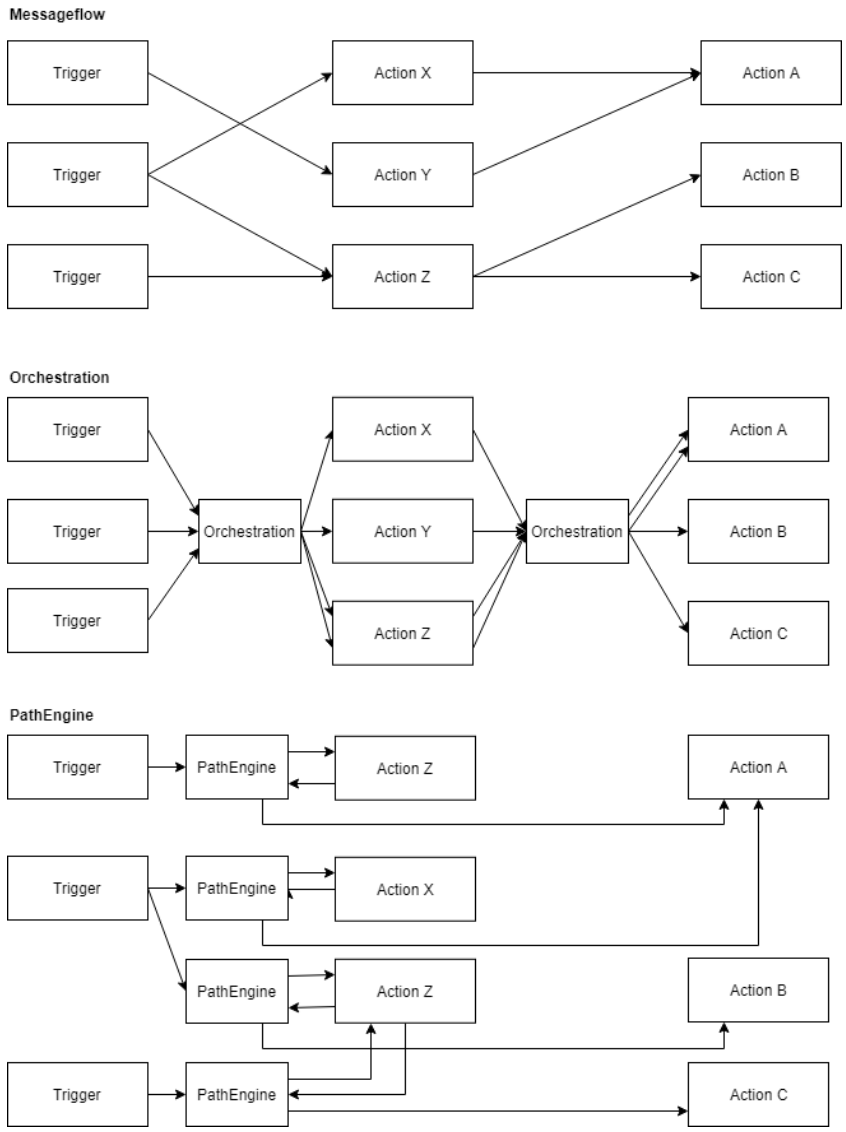


Figure 3.5: Approaches for message routing

In this thesis we have chosen the PathEngine approach as the most appropriate approach in order to create a scalable system. Because multiple instances of the PathEngine are initialized in a dynamic way, the entire platform is not dependent on one single orchestration instance. This would create an undesirable dependency. In addition, not all of the different services that need to communicate with each other need to know the exact location and sequence or the actions to be followed. This makes it possible to manage the order in which certain actions based on new data are carried out.

4

The Implementation Phase

The developed and described solution from the design phase is developed as a proof of concept. In this phase specific implementation choices are discussed. This leads to a prototype which can run in a scalable Microsoft Azure cloud-environment.

4.1. Platform & Techniques

4.1.1. Place in existing architecture

The platform is divided into two parts. The bindings that communicate with (external) services and the core platform which is able to respond on messages from different bindings.

Bindings that are coupled to existing or new services in the backend platform should be placed in line with these new services. This means that a binding for SmartCharging should be placed inside the SmartCharging project. Bindings which communicate with external parties should be placed in the communication module. The communication with external parties takes place there.

4.1.2. Platform

The prototype is developed in C#/.NET and runs on the Azure Cloud platform. We have not specifically chosen for Azure; Amazon's and Google's cloud would have provided the same benefits. Azure was selected because Allego has chosen it as the main platform for future developments.
Microsoft employees give support to Allego's IT-developers to develop new solutions in Azure.
Using a cloud solution a scalable solution can be created. The Azure cloud platform provides many services as PaaS (Platform as a Service). Within PaaS the infrastructure is managed by the system's provider. They determine how the application infrastructure works. [49]

4.1.3. Serverless

In the implementation phase when available for the purpose serverless components are used. By using serverless components, it is possible to create a scalable architecture.
The Azure platform ensures that the various applications scale well. This section explains a number of components used in the implementation.

Function as a Service

Azure Functions is a technology that makes it possible to execute code in a serverless architecture. Azure Functions is not entirely unique here; Google and Amazon also have techniques to run code without having to take a server into account. [48] Since the Azure Platform is used within Allego, Azure Functions is a logical choice.
An Azure Function can listen to events on a eventbus and then execute code. Since the Azure Platform ensures that the right computing power and memory is made available, a scalable system is created.
An Azure Function is invoiced on the basis of actual consumption. Most of the architecture described can be developed using Azure Functions. An exception to this are the bindings that use a statefull connection method. This may include client/server sockets and pubsub based communication.

In order to make such connections as scalable as possible, the communication module is kept as small as possible in these cases. In these cases, the communication module places messages on an internal service bus, where they can then be processed by a separate function. This ensures that a single communication module is able to handle more traffic. After all, a message only needs to be forwarded. If this is insufficient to be able to process the data, a load balancer can be used to route traffic to various instances of the communication module, after which the data will come together again in the rest of the platform.

Messaging
For the messaging between different components in the architecture EventHubs and ServicesBusses are used. This PaaS components help to set up a scalable architecture.
An EventHub helps to handle event based messages on a huge scale. Multiple apps or devices are sending event to an Event Hub. Different consumers can read from the EventHub concurrently at their own rate. Figure 4.1 gives overview of multiple senders and consumers on one EventHub.

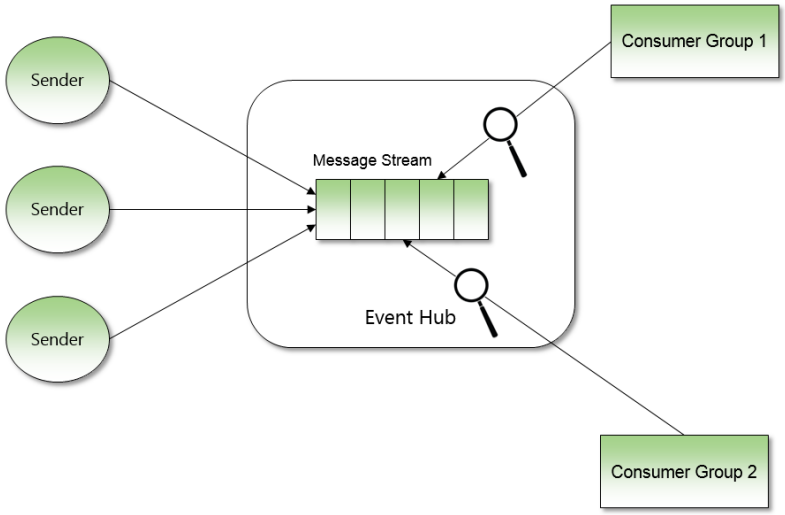


Figure 4.1: EventHub [50]

A ServiceBus is a message queue on which the sender places a message which the receiver can consume at its own pace. A service bus is a choice when Request/Reply Messages or Command Messages are implemented. In figure 4.2 a simple service bus queue visualized.

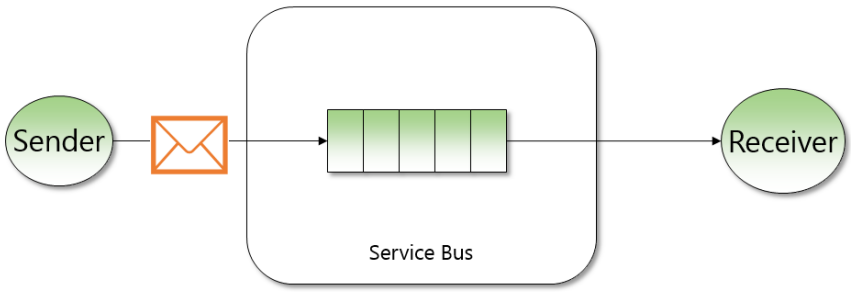


Figure 4.2: ServiceBus [50]

When a message is sent to another service, the expected response address can be defined in the request message. This mechanism helps to create a scalable platform where at run time the destination of a message is determined.

4.2. Core Functionality

The core functionality is implemented with several independent components that are connected to each other in order to be able to process incoming data. For every incoming message a new instance of the PathEngine is initiated.

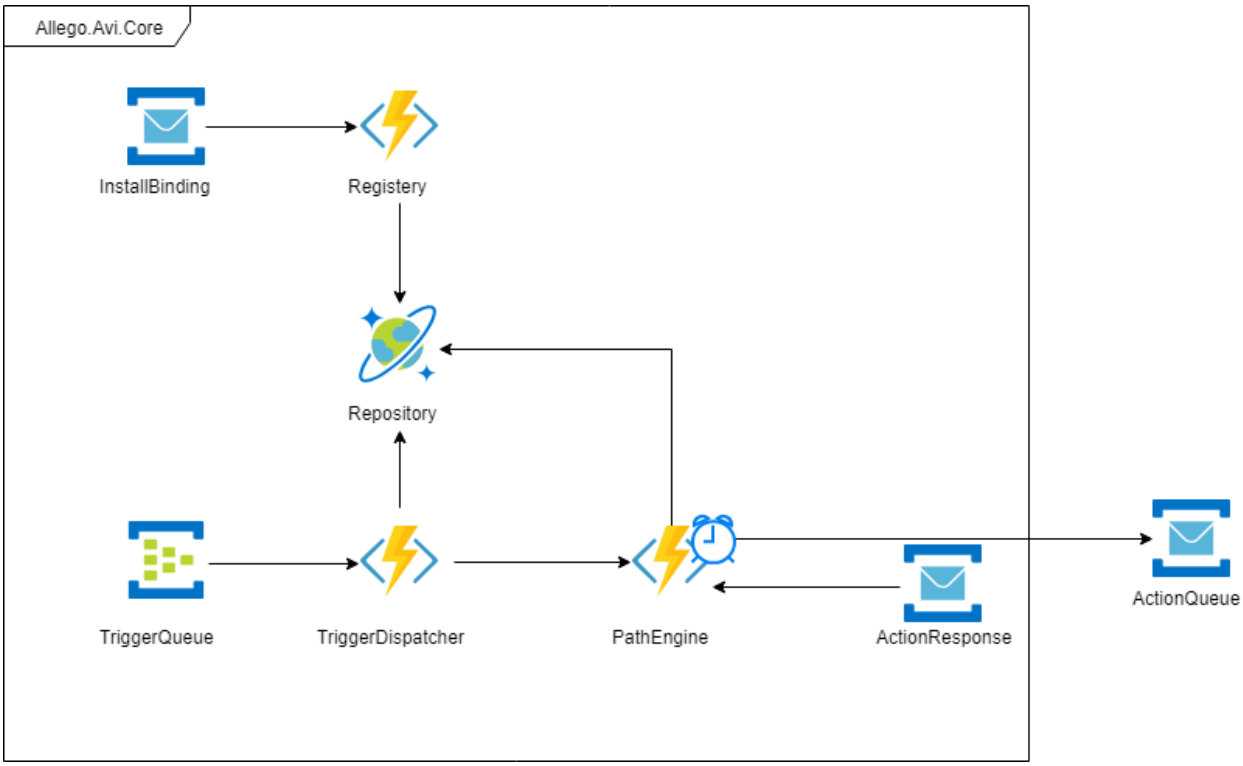


Figure 4.3: Components for the core functionality

4.2.1. TriggerQueue

For the input of Trigger events to the core platform a EventHub is used. From here several Function Apps are handling the incoming data and process them.

4.2.2. TriggerDispatcher

The TriggerDispatcher captures incoming events. The dispatcher then checks in the repository whether and which actions should be taken on this incoming message. If several separate Paths have been defined, they will be performed in parallel with each other, each with its own instance of the PathEngine.

4.2.3. PathEngine

The PathEngine is the central component in the system. When the PathEngine is activated by the TriggerDispatcher, all actions within the path are performed in sequential order.

Normally, an Azure Function is stateless. In order to ensure that the actions be carried out in the correct order, the principle of Function Chaining is applied.
This is an extension to the standard implementation that makes it possible to put together long-term functions.
This principle has been applied for the PathEngine. For each incoming message, the follow-up actions are checked. Next, these actions are carried out by means of Durable Functions.

4.2.4. Repository

A repository keeps track of all the applicable paths and actions. When the core platform receives a Trigger from a binding based on this repository it is determined what would happen.
There could be a Path that responds on a update in the State of Charge to check if it is below a certain threshold. This Path is saved to the repository and on a update of the State of Charge it is executed by the path Engine.

4.2.5. ServiceBus

A servicebus is used for the direct communication between a binding and the core platform. This servicebus has two topics which are used for different purposes:

- **Registration**
  The registration of a new binding into the platform. A binding needs to register itself to core platform. This registration should be done with some regularity. The expiration time must be specified for the initial registration. If a binding does not register again, it will be deactivated.

- **ActionResponse**
  An action gives a response on this topic. Together with this response the instanceid of the PathEngine is given. With this instance id the PathEngine can ensure that it continues with the subsequent actions.

4.2.6. Interface

End-users want to have control over what will happen after an update (Trigger). In this phase a simple interface is developed which helps end-users to define Paths themselves. They can react on a data update perform certain actions. With this interface it is possible to apply an action only to certain vehicles (filtered with a condition). A screenshot of this mockup interface is shown in figure 4.4.

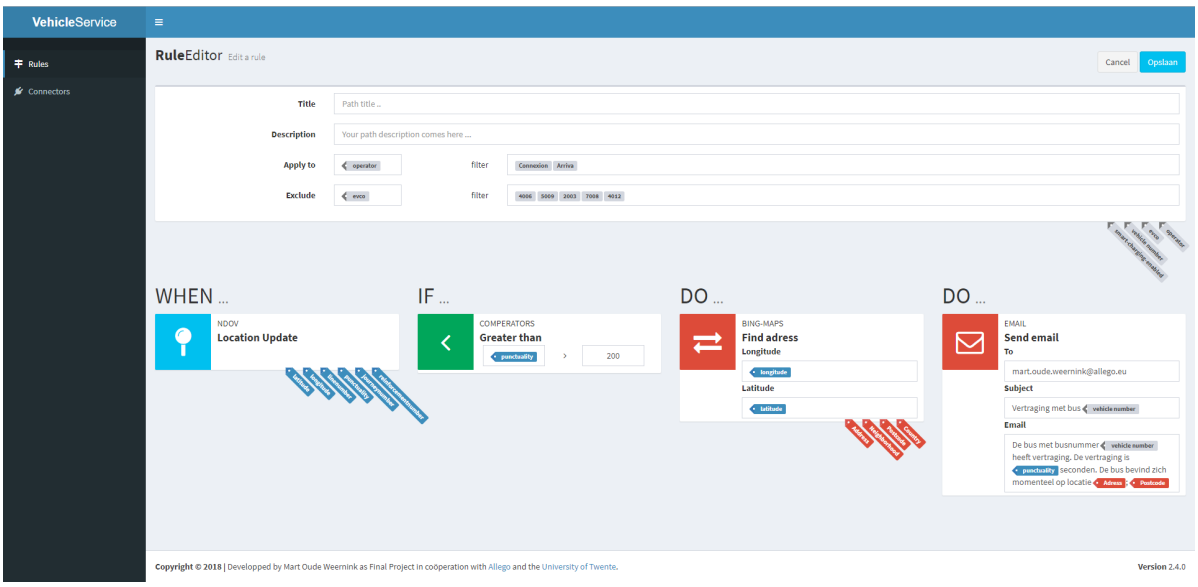


Figure 4.4: PathEditor Interface

In order to connect vehicles to datasources this interface is not necessary since all the default paths are configured during installation of a new binding.

With this interface it is possible to work out new scenarios in which the same technology can be used. For example, where a path can be defined that sends an email to the operator at a critical State or Charge.

4.3. Contracts

Contracts are used for the exchange of data between a binding and the core platform.
In this way, a universal way of data exchange is created. A contract is modelled by means of a C# class that is used by the software to transform the data into json.
The process of serializing and deserializing within C# is easy to perform, allowing messages to be sent in a universal format. By using separate contracts, both the core engine and a binding can communicate with each other in an unequivocal manner.

During the implementation phase, various contracts were developed that simplify communication.

4.3.1. Types

In order to exchange measurements between bindings standard measurement types are defined.

Simple Types

Simple Types are the classes for measurements which are comparable to the primitive types in programming languages. However in order to be able to exchange the information and serialize and de-serialize the measurements to json the simple types are explicitly defined.

- **String** A textual representation of a measurement. A example is the driver name.
- **Number** A number which can be used for calculations. A number is comparable to a double in programming languages.
- **Switch** A switch represents a true/false value. It can be used for the representation of a state of a physical switch.

Complex Types

Complex Types are a combination of multiple simple types. Multiple measurements that are logically grouped together.

- **Position** A position is a type which contains a latitude and a longitude. Both of these subtypes are numbers.
- **Group** A group is a type which contains a list of other types. With this type it is possible to create a tree of measurements which are kept together.

In the future more ComplexTypes can be added to the system to make abstractions of real-world objects. For example a battery can be combined in a ComplexType in such a way that a battery always contains information about its capacity, health, degree of charge, etc.

4.3.2. Distribution

The contracts are shared between the core application and the bindings. Therefore the communication protocol is defined in a separate package. This package can be used as a dependency in the implementation of a binding.

For the distribution of this package NuGet is used. By using NuGet it is possible to easily update the protocol in all components that need to communicate with the platform. Figure 4.5 gives a overview of the NuGet principle where the project is packaged as assembly-files which are uploaded

to a NuGet host. Allego has it's own nuGet-private host which is used for this package. Then the NuGet package is used in the implementations.

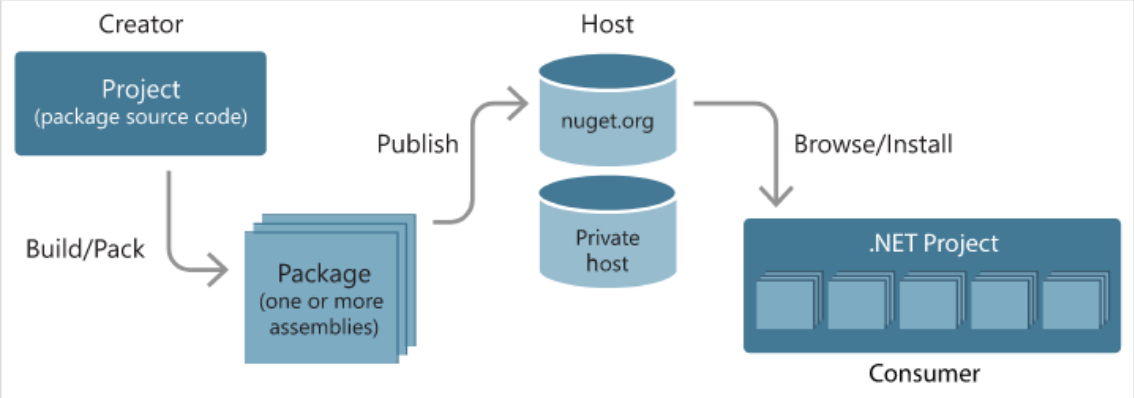


Figure 4.5: NuGet flow from project to dependency

4.4. Defined Bindings

In the implementation phase, a number of different applications were worked out. The following bindings have been designed for this purpose. The purpose of these bindings is to show how different data sources are combined in order to obtain a good representation of the vehicle.

The following bindings were formulated during the implementation phase:

- NDOV
- GTFS-RT
- Teltonika
- VehicleState
- BingMaps

In the following sections a short explanation will be given for each binding.

4.4.1. NDOV

A binding for the communication with the dutch NDOV-loket. This binding receives updates from a central server which are send to the platform as trigger events. This data source delivers data from the dutch buses. The data is available through a ZeroMQ interface. This allows to subsribe to data updates that are pushed by the NDOV-server when available. To subscribe to a ZeroMQ queue a small application is developed which receives the data updates and sends them to a EventBus for further processing. Figure 4.6 gives a overview of the components in the implementation of the NDOV binding.

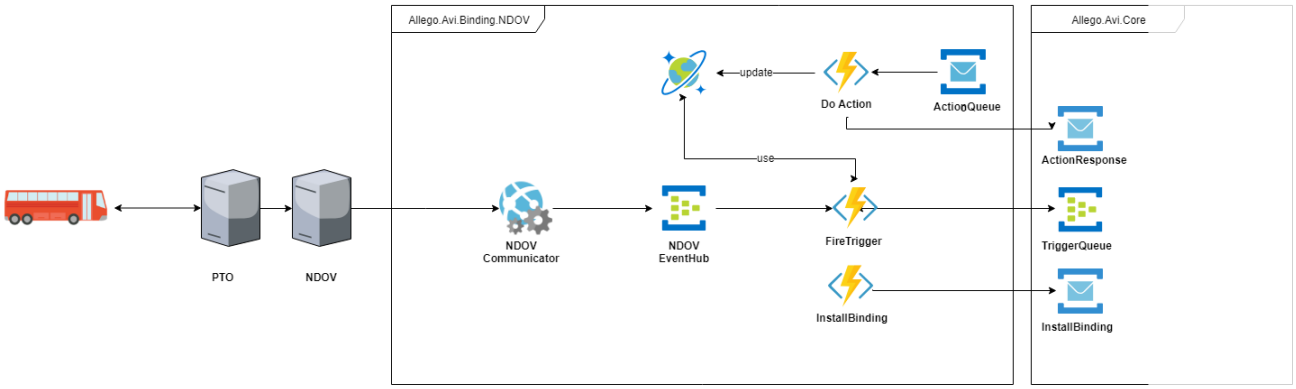


Figure 4.6: Azure Components in NDOV-binding

The Allego backend receives the data from the NDOV-server. The information on the NDOV server comes from the PTO (Public Transport Organizer) which receives the data from the vehicle or uses a third party solution for the data extraction. The binding is responsible to subscribe to a topic on the NDOV server to retrieve data updates. When a data update is received this update is placed on an EventBus. The data is then converted into a Trigger that can be collected by the Core platform.
In order to be able to provide the platform-wide vehicle number for Triggers, the NDOV connection builds up an internal mapping table itself. This does, however, require that changes in vehicle numbers and carrier are passed on to the binding. For this purpose, a Path is defined during installation which ensures that changes in carrier and vehicle number are passed on.

During the installation of this binding several default paths are configured. It is possible to subsequently define separate paths for specific groups of vehicles in order to determine the behaviour of the vehicle.

The NDOV-binding has the following characteristics:

package	Allego.Avi.Binding.NDOV
depends on	Allego.Avi.Binding.VehicleState
actions	Carrier update, VehicleNumber update
triggers	Arrival, Departure, On-Route, Off-route,
paths	Listen to data update (evco) from VehicleState
	Listen to data update (carrier) from VehicleState
	Arrival-trigger to VehicleState
	Departure-trigger to VehicleState
	On-Route-trigger to VehicleState
	Off-route-trigger to VehicleState

4.4.2. GTFS-RT

A binding to retrieve realtime positions of vehicles. The GTFS standard is an international standard. In the developed binding the data is fetched from a third party server. The protocol is pull-based which means that the binding has the initiative to request data at specified services.
The binding is developed in such a way that multiple different GTFS-feeds can exist next to each other. In a local configuration the different sources are defined.
The GTFSRT binding is developed in such a way that no Virtual Machine is necessary for the operation. With the usage of different Azure Functions that are linked together with a service bus, a

scalable binding is created. This is in line with the requirement drawn up for the scalability of the entire platform.

The GTFSRT-binding has the following characteristic:

package	Allego.Avi.Binding.GTFSRT
depends on	Allego.Avi.Binding.VehicleState
actions	-
triggers	PositionUpdate
paths	listen to data update (evco) from VehicleState listen to data update (cpo) from VehicleState send dataupdate to VehicleState

4.4.3. Teltonika

A binding which is used for the connection of Teltonika tracking devices to the platform. These tracking devices are built into the vehicle and connected to the CAN bus in order to send up-to-date vehicle information. This information can then be processed in the core platform. Figure 4.7 shows the components of this binding.

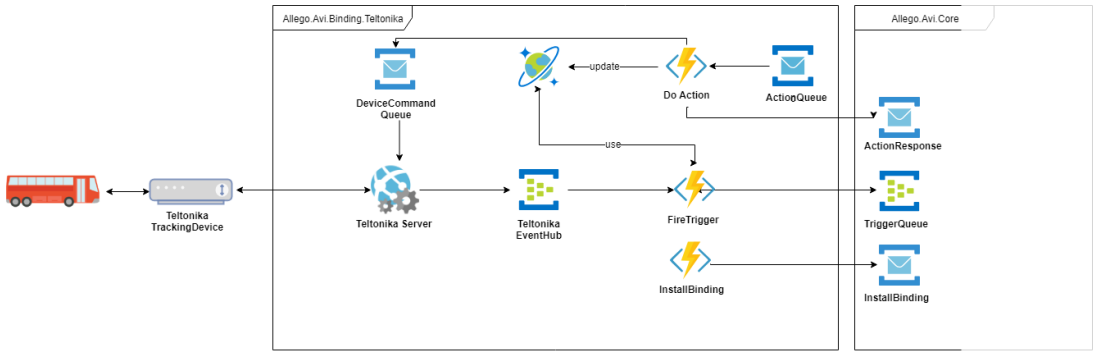


Figure 4.7: Azure Components in Teltonika-binding

The Teltonika-trackingdevice is built into the vehicle. The device establishes a connection with the platform using a server socket in order to send data updates. A TeltonikaServer has been created for this purpose, which the devices can connect to. This server has been deliberately kept as small as possible. This is done to ensure that the binding is as scalable as possible. One or more virtual machines are required to communicate with devices. The final processing of incoming messages is scalable by using Azure Functions.

It is also possible to send commands to the tracking device. On the basis of these commands, a relay can be sent to the bus. For example, when working with the bus OEM, it is possible to restart a charging session. In the current setup, only a reset command is included. This is because this tracking device is currently not built into a moving vehicle. However, this binding does show the possibilities of two-way communication.

The Teltonika-binding has the following characteristics:

package	Allego.Avi.Binding.Teltonika
depends on	Allego.Avi.Binding.VehicleState
actions	Reboot
triggers	PositionUpdate
paths	Listen to data update (mac-adress) from VehicleState

Position-trigger to VehicleState
Soc-trigger to VehicleState

4.4.4. VehicleState

The VehicleState binding is an important binding for the operation of the system as intended. This binding gets input from other bindings which are processed and saved as an internal vehicle state. Since this binding gives the internal representation of a vehicle, a DataUpdate-Trigger from this binding means the state of the vehicle is changed. For example a DataUpdate from measurement 'position' is independent from the datasource used to retrieve the position from the vehicle.

The VehicleState-binding has the following characteristics:
package Allego.Avi.Binding.VehicleState

depends on -
actions DataUpdate
triggers DataUpdate
paths -

In order to manage priorities of different data sources a priority is given to a certain data update. This meets the requirement to offer a solution for conflicting data (Functional Requirement 3). The VehicleState binding will determine based on the priority which input from the different sources is the current value for a vehicle.
For example the position update from the NDOV-loket has a higher priority than the GTFS-datasource. This can be configured vehicle/group based by configuring a new Path.

4.4.5. Bing Maps

A simple binding that connects with the Bing Maps api. This binding is an example of a binding that does not have triggers. This binding can be invoked when a path is triggered from another binding. In the example of Bing Maps an address can be found by coordinates. In this way, data can be retrieved in a universal way. Figure 4.8 gives a overview of the components in this binding.

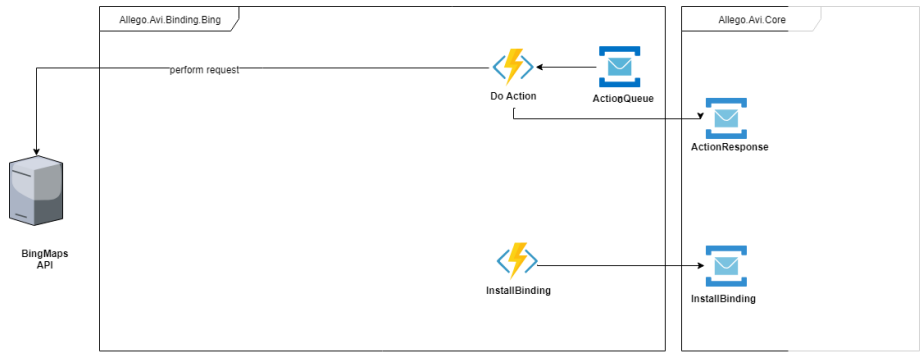


Figure 4.8: Azure Components in NDOV binding

This binding listens to requests from the core platform. A call is then made to the Bing Maps API. The advantage of this setup is that when the Bing API changes only the binding needs to be adjusted. The different services that depend on this will then operate as usual.

The BingMaps-binding has the following characteristics:
package Allego.Avi.Binding.BingMaps

depends on	-
actions	Geocode, Distance
triggers	-
paths	-

4.5. Overview

In this implementation phase, the various defined concepts have been worked out. The appropriate techniques of the Azure Platform have been applied in this respect. This has resulted in a complete design.
Figure 4.9 shows the packages from this implementation phase. Looking back at figure 1.1, we see a number of parallels here. Both the Allego Services and sources of third parties have been integrated with each other with a binding. The core platform (green) provides integration between vehicle data extraction (orange) and services (blue).

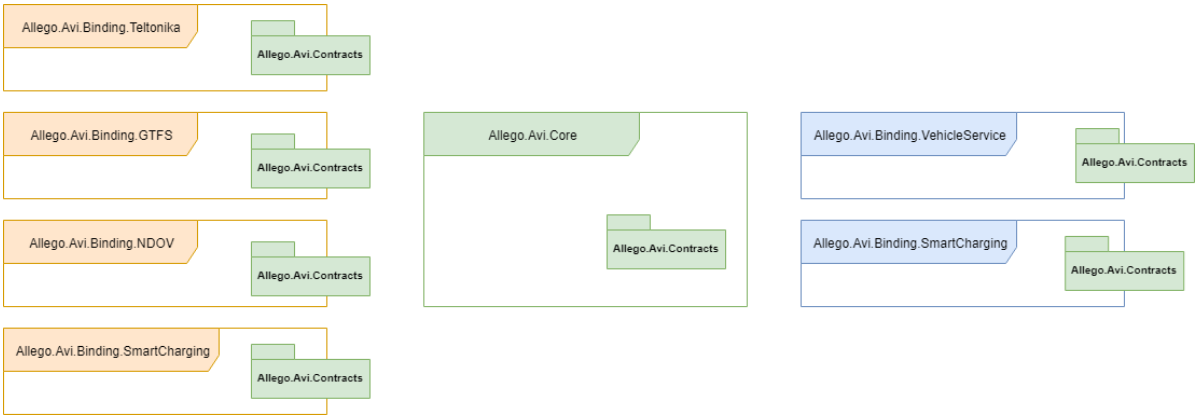


Figure 4.9: Overview of packages in the prototype

5

The Validation Phase

In the validation phase the developed architecture is validated. This validation was carried out by means of an expert validation. In addition, the feasibility of adding new services and data resources in the future was tested.

5.1. Requirements

In this phase the validation of the framework is performed based on the requirements defined in the analysis phase. An explanation for every requirement is given.

5.1.1. Functional Requirements

1. **The system should provide a mechanism to install a new data source.**
   The system provides a way to install new datasources without having to reboot/restart other components. A new datasource can be added to the core platform to register itself and tell which data it provides and which action it has.

2. **The system should provide a mechanism to combine data from different data sources related to one vehicle.** The developed VehicleState binding provides a mechanism to save the vehicle state based on priorities. By defining default or customized paths for a certain binding it is possible to set the priority of new data.

3. **The system should provide a mechanism to deal with conflicting data from different data sources.**
   Based on the priorities it is possible to determine which data should be used from the different datasources. Currently a basic priority system is defined however when this needs improvement than a new version of the VehicleState binding could be developed. For example a TTL can be added to improve the mechanism which deals with conflicting data.

4. **The system should provide a mechanism to dynamically respond on incoming data depending on the vehicle.**
   The system is designed in such a way that all vehicles can have different behavior based on incoming data. This is implemented by the definition of a Path. With the installation of a new binding a default Path is defined, for specific vehicles an other path can be added. Furthermore the system is able to respond on a certain state and perform actions based on this.

5. **The system should provide a mechanism to keep track of historic vehicle data.**
   In the current platform all the historical data related to the vehicles is saved in a database. theoretically this data is available for other services, but at this moment there is no interface / mechanism to provide this data to other services.

6. **The system should provide a mechanism for other services to retrieve vehicle updates.**
   Services which want to retrieve vehicle data from the system must create a binding to the system. This binding acts as a consumer and will receive dataupdates when the VehicleState is changed. When a third party service wants to listen to the raw data from other bindings it can subscribe to this data as well.

5.1.2. Non Functional Requirements

1. **The system should be scalable, to support thousands of vehicles at the same time.**
   The architecture of the system is designed in such a way that all the individual components are scalable. In the current setup there are some limitations in number of messages that can be processed by the servicebus [50]. The current setup is able to process all data available through the NDOV loket.

2. **The system must make use of the Microsoft Azure Platform.**
   During the design of the architecture and the implementation of this architecture the functionalities of the Microsoft Azure Platform are used. By using serverless technologies from this platform it is possible to create a scalable solution.

3. **The system should be designed in such a way that it is high-available. Downtime seriously effects business.**
   The architecture is designed in such a way that all the components are loosly coupled. This means that the malfunction of a component does not influence the whole system. However when a individual component does not work as expected endusers can notice this because they do not receive data updates. By making use of a existing platform with standard components it is possible to detect failures in a early state and automatically apply a failback mechanism. In the Azure Stack it is possible to monitor services and processing times to guarantee and monitor uptime.

5.2. Expert Validation

During the final project several meetings were held with experts to validate the design. This is an iterative process in which in the beginning the focus was primarily on various workshops with the business team. Then, at an early stage, there was consultation with architects to discuss various ideas and alternatives. In the end, an expert meeting took place on 21 June at which the developed architecture and principles were discussed in detail.

During the final project, the following experts were involved for the validation of the developed architecture:

- Jeroen Quakernaat Architect at Allego
- Michiel van Schaik Architect at Microsoft
- Maarten van Sambeek Prinicipal Consultant at NAVARA
- Remco Tjeerdsma Product Owner / Software Engineer at Allego
- Tim Smal Scrummaster at Allego
- Mattheo van der Molen Project Manager Public Transport at Allego
- Frank Verhulst Team Lead - Transport as a Business at Allego

During the expert meetings several topics are discussed. Each topic is elaborated below in the following steps:

- **What** The subject of this topic/what is being discussed?
- **Why** Why is this topic interesting in the context of vehicle data integration?
- **How** How was this topic taken into account during the final project?
- **Experts** What is the outcome of the expert meetings?
- **Conclusion** How does this affect the model?

5.2.1. Scalability

What Check that the platform is designed to be scalable.

Why From the start, it was important to develop a scalable platform. At the moment, the e-mobility market is still relatively small. However, once strong growth is expected, it is necessary for the developed platforms to be able to grow along with it.

How A traditional platform uses software that runs on certain assigned hardware. This platform may be virtualized. A load balancing method can be used to ensure that the platform remains scalable. The chosen setup for the integration of vehicle data makes use of serverless components. This means that it is possible to run the platform on a cloud environment that ensures that the application is automatically scaled. When little data enters the platform, few resources will be used. When the data intensity increases, the platform will automatically scale up.

Experts It is clear to the experts that the architecture was designed with a view to find a scalable solution. By using the components that the Azure platform offers. In addition, the way in which different data sources are connected (as a separate software service) ensures that less scalable components do not directly affect the rest of the platform.
For the experts it is clear that the scalability of an individual software binding depends on the developer of this binding. It is therefore important to set up new data sources in a scalable way in the future.
The core functionality of the platform currently has domain knowledge. As a result, it is conceivable that in the future changes will have to be made to the core when new data sources or services have to be opened up. According to the experts, it should be possible to validate input and output, for example by means of JSON schemes. This would ensure that the core functionality is more universally applicable. Not only on vehicles, but also, for example, on charging stations.
The current architecture is based on heartbeats to detect whether certain data sources are still available. The experts note that this may cause scalability problems because this needs to be kept up to date.

Conclusion With the design and implementation of the architecture scalability is taken into account. This is sufficient for the first application of the architecture. The solutions proposed by the experts can be implemented within the model without the need to adjust the basic principles.

5.2.2. Extensibility

What Checking the extensibility of the developed architecture. Is it possible to support new data sources and services?

Why As the analysis phase has shown, it is not possible to use only one protocol for the integration of vehicle data. To deal with this, the developed platform must be able to handle multiple different data sources in order to be able to provide different services with data. As a result, it is necessary to be able to easily open up new services.

How The system is designed in such a way that it is extensible. New datasources can be added to the system without a restart of other components. The architecture is designed in such a way that the different components are able to communicate with each other with the help of standards datatypes. The input and output to the rest of the platform is defined. The in and output to the data source is binding specific. This makes it possible to communicate with different types of protocols and end-points.

Experts The principle of using a binding as an abstraction layer between the various data sources is experienced by the experts as a good choice. This helps to be able to support different types of protocols and communication methods. The experts suggested that generic data names could be registered in the register in order to avoid a jumble of different variations on the same measurement value. For example, the state of charge can be recorded through different bindings, but the same type of measured value is involved. By using a generic registry it is possible to increase the extensibility of the platform. It is a great added value that new bindings can be added to the system during run-time. This allows new services to be linked without interfering with other services.

Conclusion The developed architecture makes it possible to open up different data sources to a variety of services.
Based on the feedback from experts, it is possible to develop the platform in order to increase the extensibility of the platform.

5.2.3. Security

What How is security addressed in the architecture?

Why The security aspect is becoming increasingly important. However, the focus of this project has been from the beginning to come up with a solution for the integration of vehicle data.

How Within the project, the assumption has been made that the different services in the platform can trust each other. The entire platform is a closed box within which new software cannot simply be installed. Nevertheless, the platform developed could not support authentication and authorization.

Experts In order to be able to apply the principle properly, security mechanisms still need to be built in. Currently, the source of incoming events is not checked. Theoretically, this can be used to create a malicious binding that appears as another binding. Within the current design, the assumption is made to trust the other software services within the platform. However, there are conceivable methods in which the identity of a binding can be better guaranteed by exchanging certificates and/or tokens. According to the experts, it would be useful to develop a method that enables authentication and authorization between the different services.
This security aspects are also important to protect the vehicle data which is gathered. Since this data can tell something about the vehicle and its user it is important to secure and encrypt the data. Also to comply with the new GDPR regulation.

Conclusion Although the security aspect has not been an important area of attention from the start, it is necessary to look at security considerations before such an architecture can be applied practically.

5.2.4. Process

What This topic looks at the definition of the requirements and use cases and the way in which the architecture has been developed.

Why To see how the process went and whether there are any requirements missing that had impacted the result.

How During the process, the main focus was on developing a solution for data integration. An iterative process was used, in which new insights ensured that the model was adapted from time to time.
In addition, the approach has also changed during the project. Initially based on a single tracking device for the input of vehicle data, the study has shown that a combination of different data sources will always be required to provide a complete overview of the situation.

Experts The requirements and use cases developed are in line with the internal guideliness within Allego. However, it is clear that a number of requirements are not defined but were indirectly taken into account during the process. Although the requirements are not always S.M.A.R.T. defined, the architecture developed does fit in. Allego has gained the insight that there is not only one data source that can be accessed in order to obtain all vehicle data.

Conclusion The tracability of the requirements and use cases could have been better, the final result meets the wishes of Allego. With the developed architecture it is possible to access and display various data sources in an interface.

5.2.5. Alternative Approaches

What The architectural design in relation to alternative approaches in literature and real life applications.

Why During this study, a new methodology was developed for the processing of vehicle data. Within this step, alternative methods to integrate vehicle data are discussed with the experts.

How Within the process, we mainly looked at other application areas where the same problem applies to money. That is, that different data sources want to communicate with each other. For data integration, a middleware solution is often used. These middleware solutions enable multiple data sources to communicate with a single system. The final solution is strongly based on the IFTTT-model.

Experts The experts note that the pattern used is often used. Not only for end applications, but also within Microsoft's cloud platforms, for example. The experts indicate that it is very similar to the way Azure Functions deals with different inbound and outbound services. In addition, the parallel with Microsoft Flow is drawn almost immediately, giving end-users the opportunity to connect different data sources with the aid of a graphical interface.
Since data often originates from IOT devices, it could be possible to make more use of existing techniques within the Azure Cloud Platform. For example, the Azure IOT-Hub can be used for communication with external devices.

Conclusion The architecture purposed is a combination of different approaches. The application of existing models to the domain of vehicle data is very valuable. The suggestions made by the experts can be used as a supplement to the model set up in this thesis.

5.2.6. Usability

What Is the developed architecture practically applicable for the integration of vehicle data?

Why For Allego it is important to have an application that can be used for the integration of vehicle data. Within this topic, the usability of the developed architecture within Allego will be examined.

How The architecture has been developed on the basis that there are different combinations of data sources for vehicles. The data sources used depend on the type of vehicle and the type of concession. The platform has also been developed with this in mind.
However, initially only a few customers will use vehicle data. this can be explained by to the fact that the exchange of vehicle data currently requires investments in an additional tracking device or links to the vehicle manufacturer's backend.
As a result, it is particularly interesting for a number of large organizations such as PTOs and lease companies to set up an integration with vehicle data. This means that in the first instance the platform can be slightly overkill. With the expected growth and diversity of vehicles and customers, combining different data sources becomes increasingly important.

Experts The graphical interface is seen by the experts as an added value. This interface shows the possibilities of responding to incoming data at bus/group level.
At the moment, there is mainly a direct demand for vehicle data for the application of Smart Charging. If only this application is considered, the developed platform may be slightly overkill. It is then easier to send the data directly to the Smart Charging service. Although the platform was initially developed for vehicle data, the experts also see possibilities for, for example, charging data. The architecture has been designed in such a way that it is possible to react dynamically in which the different bindings complement each other and thus ensure that a single system-wide value of a measurement is created. Within charging stations, the problem now arises that different services all do their own interpretation on the incoming data, which results in different definitions for, for example, an offline charging station. The architecture as it is set up for vehicle data can help in this.

Conclusion During this project, it became clear that integrating vehicle data is more complex than connecting a single data source. In this way, the project mainly contributed to gaining insights into the possibilities for the integration of vehicle data.
The applied architecture is useful for reacting dynamically to incoming data. Now applied to vehicle data, in the future also for other objects, such as loaders and parking sensors.

5.3. Business Scenario

In order to validate if the solution developed during the final project fits the needs of the business, together with Maurits Doetjes and Mattheo van der Molen a design for the MVP of Vehicle Services is created, see figure 5.1. This is a new service which needs to be developed, the current mockup is based on the input of this report (see Location Service in Analysis Phase). This service aims to provide insight into vehicles and their location. This makes it possible for end customers to gain insight into their vehicles. The first graphic design is used to see if the developed architecture can successfully support such a service. Based on the designed screens an indication of the data required for this service can be made. Based on this, the data providers (which result in bindings) are determined. For all the developed screens see Appendix C.

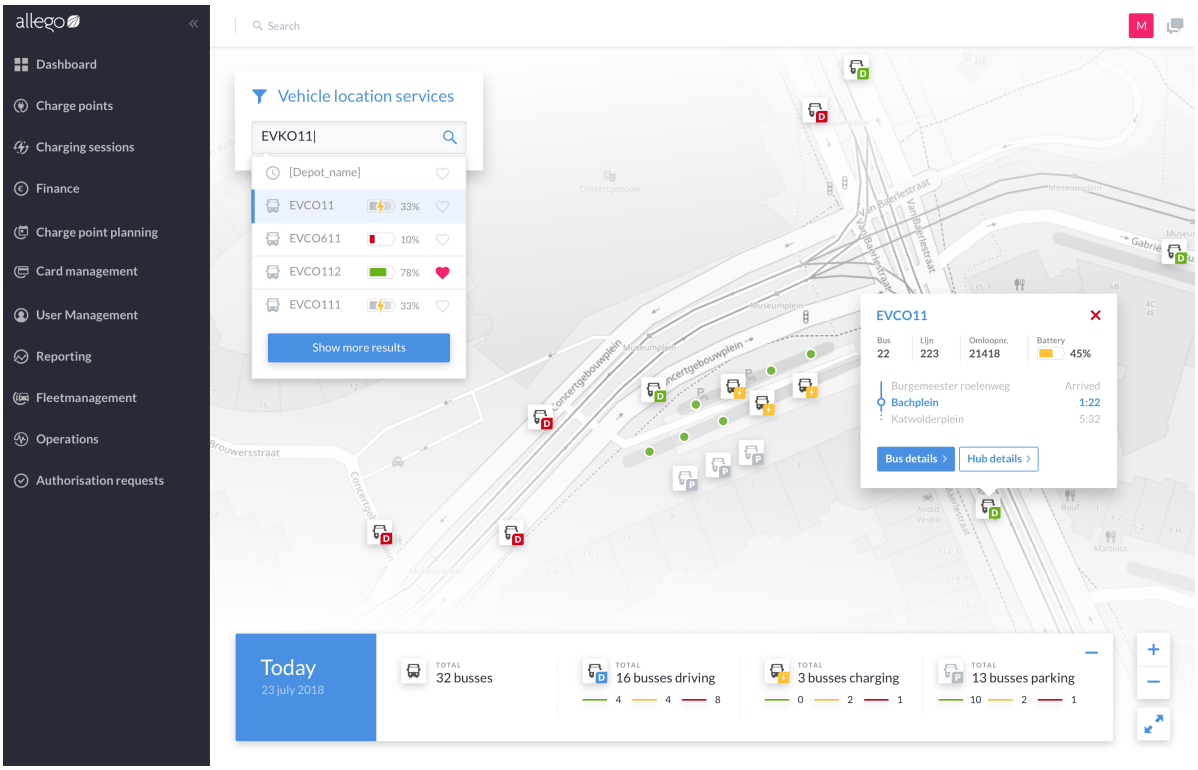


Figure 5.1: Mockup for vehicle services

For this service there is input required for the measurements described below. For every measurement, the data providers that can be used for these measurements are specified. The measurements and the data sources are summarized in table 5.1.

- **Busnumber**
  It is possible to retrieve the bus number from various datasources such as Charging Services and the NDOV-loket.

- **Linenumber & Omloopnumber**
  The extraction of the current line number is only possible with the integration of data sources such as NDOV and GTFS. Based on standardization of in vehicle systems such as on-board computers it can be possible to extract this information from the vehicle itself. At the moment it is possible to connect to on-board computers, however, each manufacturer of on-board computers has its own protocol for it. However, the ITxPT consortium is working on a better exchange of data for different systems within pubic transport [27].

- **Position**
  Several datasources have information about the current position of a vehicle. Where the NDOV-loket only gives the position of buses which are currently active on a certain busline a tracking device can deliver this information always. Also the charging services can help to determine the location of a vehicle. Based on the charger location the location of a bus can be determined.

- **SoC**
  The extraction of the SoC for a vehicle is possible with a physical device on the bus. A tracking device can extract the state of charge from the CAN-bus of the vehicle. It is more desirable if data can be exchanged directly with the vehicle OEM. Allego has now started working with vehicle OEMs to enable the exchange of data. It is actually not desirable to install a new (connected) device in the vehicle. The aim is therefore, together with market parties, to look for the most efficient possible exchange of data.

- **Bus State**
  Based on the different datasources the system should determine the bus state. For example the charging services notificies the platform when a vehicle is charging. Charging / Driving / Parking

- **Time to charge**
  The current set of data sources do not deliver a time to charge. But based on the current SoC, the position and route information such as linenumber a prediction can be made to determine the expected time to charge. When in the future charging schedules are exchanged this prediction can be replaced with the planned time.

- **Time since charge**
  Based on the data available from charging services the time since last charge can be calculated. Technically the last charging time will be saved in a database from where on request the time since charge can be calculated.

- **Charging moments today**
  To get insight in the charging moments today integration of charging service data is needed. The Vehicle Location Service is responsible to save this data in the appropriate format.

Table 5.1: Measurements and the data-providers for MVP - Vehicle Location Services

Measurement	**NDOV**	**Tracking Device**	**Prediction**	**Charging Service**
Busnumber	yes	yes	no	yes
Linenumber	yes	no	no	no
Omloopnumber	yes	no	no	no
Timetable	yes	no	no	no
Postition	yes	yes	no	yes
SoC	yes	yes	no	no
Bus State	yes	no	no	no
Time to charge	no	no	yes	no
Bus State	yes	no	no	yes
Charging moments today	no	no	no	yes

All in all, it is possible to support this new service with the help of the architecture that has been developed. Current data sources can support such a service. The architecture developed is the

bridge between the various data sources and the final service. In order for this data integration to run smoothly, a new binding needs to be created that subscribes to the various data updates.

6

Conclusions & Recommendations

6.1. Conclusions

Within this study, a solution is developed in four phases for the integration of vehicle data into the existing backend for the management of charging stations.
The aim of this research was to develop a solution for the integration of vehicle data. The different chapters deal with the research questions from chapter 1 by means of the four phases (Analysis, Design, Implementation and Validation). Finally, this section provides answers to the research questions.

6.1.1. RQ 1 - What is the current state-of-art in EV charging solutions and data integration?

In recent years, there has been strong growth in the application of e-mobility. Consumers as well as companies use EV-charging solutions. It is crucial for commercial parties to have insight into their operations and vehicles. In the EV domain, various protocols have been developed for communication between the parties involved in the charging process. Until now, there was no need for the exchange of data between the vehicle and a backend. However, with the expected growth, it is necessary to optimize the charging behavior in order to be able to support large quantities of cars with a relatively small network. This optimization can consist of adjusting the available amount of power as well as assigning a suitable charger. Vehicle data can contribute to this optimization. This vehicle data can be used for current demands as well as for building predictive models. There is no universal standard for the real-time exchange of data between vehicle and third parties. Based on different manufacturer and project specific solutions it is possible to obtain vehicle data.
During the analysis phase of this research we looked at middleware systems and solutions in other domains. Many of the principles for abstraction and the application of rules have been used in the architecture developed during this research.

6.1.2. RQ 2 - What elements are required to set up a system for the integration of vehicle data?

There is a wide variety of data sources and services that want to make use of them.
Ideally, communication between vehicle and service parties should be standardized. Such a standardization will reduce the number of protocols required to be able to access vehicle data. However, there will always be different protocols and data sources.
During this project the different data sources are connected to services within the platform with a kind of connector, called a Binding. This binding allows a data source to communicate with (existing and new) services. In order to integrate vehicle data, both a producer and a consumer are required. Producers are generally the data sources and consumers are the software services that will use vehicle data.

6.1.3. RQ 3 - How can we combine data from different sources?

During this research, an architecture has been developed that helps to combine different data sources. Different data sources can complement each other in order to obtain a complete picture of the vehicle. By separating the communication with specific data sources into a separate software service, it is possible to set up a scalable platform. The internal exchange of data takes place in a standardized format.
Also when different data sources that contradict each other, the data can be processed. Incoming data can be cleaned up and and based on a priority-system data from different sources with the same measurement for the same vehicle can be processed.
With this framework it is possible to handle different datasoures. Different datasources for one vehicle. But also different datasources for the same measurement for different vehicles.

6.1.4. RQ 4 - How can we develop an architecture that is able to deal with future growth?

The platform has been developed in such a way that future growth is possible. This could include adding new data sources or applications of vehicle data. During this study, only a limited number of protocols fell within the scope. However, new protocols can be added to the platform without directly affecting the existing data sources.
In addition, during the design and translation to Azure components, the creation of a scalable solution was taken into account. By using serverless components, it is possible for the platform to grow in line with the growth in connected vehicles.

6.1.5. RQ 5 - What lessons can be learned evaluating the developed architecture?

The developed architecture makes it possible to react dynamically to vehicle data from different sources. The way in which data updates can be used in an IFTTT-like way in order to initiate actions within the platform is considered very valuable. Developing new data sources still requires some development work, so in the future we will also have to look at which data sources need to be opened up. In addition, this architecture is especially useful when there are multiple data sources that need to communicate with multiple different services. When only one service requires certain data, it is better to apply a middleware solution. The main goal of the architecture is to integrate vehicle data, but an important result is to be able to react dynamically to incoming messages (both internal and external).
In order to be able to apply the developed architecture successfully in practice, additions will have to be made. This is mainly related to practical aspects such as manageability and security. These additions and extensions are mainly the result of the expert validation and are further elaborated in the future work section.

This has resulted in the development of an architectural setup that enables Allego to deal with the integration of vehicle data in a both scalable and flexible way. The four phases in this thesis together form the answer to the main research question **"What architectural setup would provide both scalable and flexible solutions for integration of vehicle data?"** by giving a definition of a architectural platform which supports the existing backend to integrate vehicle data to improve current business and extend the market position of Allego.

6.2. Discussion

This section lists a number of cases that may have had an impact on the outcome of this research.

6.2.1. Middleware

An often mentioned alternative to integrating vehicle data is the application of a middleware layer. A middleware layer has the ability to combine multiple different sources within a system. In this way an abstraction layer is created. However, the challenge in this case is that the data has to be shared with a variety of applications. In addition, there must be different responses to incoming data on the basis of the vehicle concerned. However, concepts such as data mapping and transformation do return. The developed solution can actually be seen as a combination of a middleware architecture in which events can be used to react dynamically to incoming (and outgoing) activities.
It is in particular the combination of IFTTT-like solutions such as Homey, Zappier, OpenHab and the abstraction and mapping methods of middleware solutions that make it possible to integrate vehicle data with this solution.

6.2.2. Cloud Platform

During the development of the architecture, Microsoft Azure was the target platform from the start as a target platform. However, a split has been made between the design and implementation. In principle, the design could also be implemented on other cloud platforms. The concepts used such as an event bus and serverless computing are also possible in the cloud environments of Amazon and Google [51]. The way in which the different data sources are combined by means of Bindings and a central PathEngine can therefore be applied perfectly.

6.2.3. Standardization

During this research, we looked at a way of bringing together different protocols and integrating them into software architecture. However, each of these protocols does require development and maintenance. From the perspective of Allego as a data consumer it is desirable that the data exchange is standardized. Just as Allego has argued for a standardization in the communication between EVSE and CPO (OCPP) and CPO and MSP (OCPI), it would be good if a standardization in the communication between EV and third party is initiated. In this case Allego is the third party.

It is good to set up a new standard that makes it possible to communicate with all (connected)vehicles, but here we need to look at a standard that is supported by at least a major part of the market. Otherwise, the problem will be exacerbated by the creation of a new standard which will be applied by a small part of the market.
When developing a new standard, however, it should be noted that a new protocol must replace other protocols. The cartoon in 6.1 illustrates the potential risk.

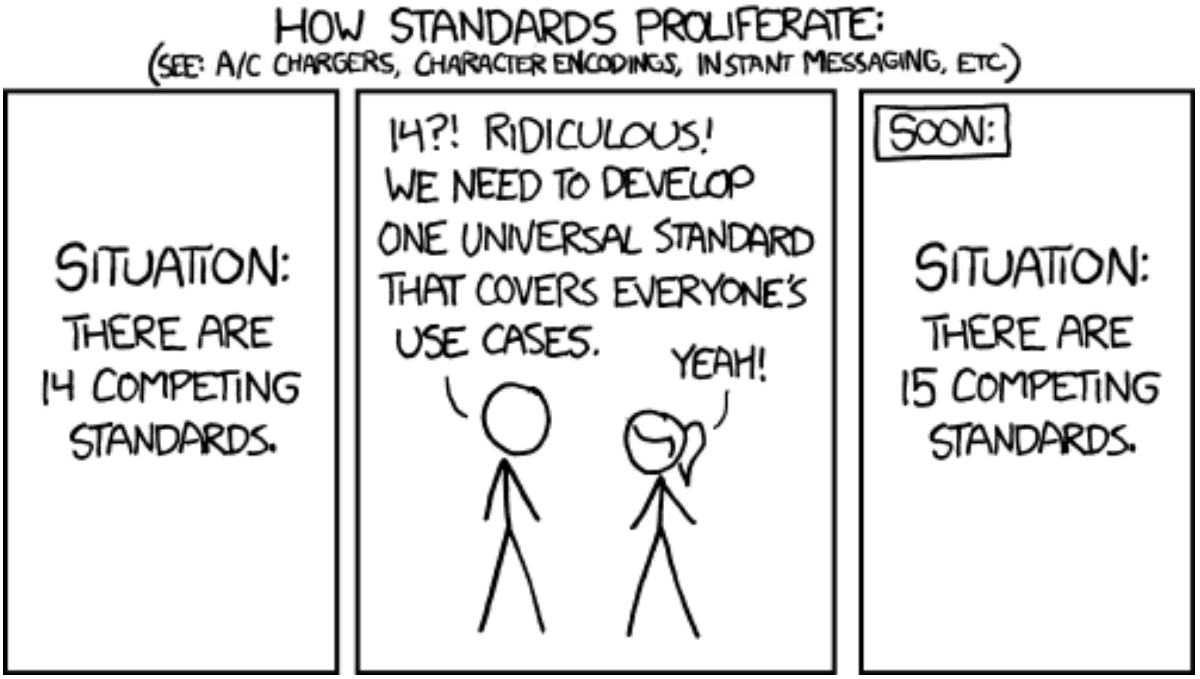


Figure 6.1: XKCD: Standards [52]

When standardization is applied in a proper manner, there will be less fragmentation in different protocols. However, there will always be different protocols. The architecture developed offers a solution for this.

6.2.4. Customers

It is important to work together with customers in order to unlock vehicle data. In addition, there must be a clear goal for customers to share data. In today's market, parties do not just want to exchange data. Companies realize that data is a valuable asset [53].
Partly because of this, no customers where willing to share their vehicle data with us for this research. However, when Allego can demonstrate the added value of the integration of vehicle data, this helps to find pilot customers for this.
It is important to find a customer who is willing to share the available data as soon as possible in order to be able to setup a pilot. The Transport as a Business team is aware of this and is looking at the possibilities for cooperation.

6.2.5. Existing Platforms

During the research topics phase, existing solutions for the integration of vehicle data were examined. The solution finally developed cannot be compared with the existing solutions, see also appendix A. The solutions in this appendix are based on a one type of device that is connected to a specific cloud platform. The solution developed during this research makes it possible to combine multiple data sources.
Allego wants to be able to connect many different vehicles. In order to make this possible, it is necessary not to depend on one way to unlock vehicle data. The developed platform therefore offers the possibility to access data in a universal way by means of various Bindings.

6.3. Future work

This section discusses the future work resulting from this project. This concerns both practical work that has to be carried out before the platform can be applied successfully and further research that can be carried out in the field of data integration and analysis.

6.3.1. Security & Privacy

When setting up the architecture, the security aspects within the platform were not taken into account. On the one hand, this concerns the security between the various components in the platform. For this purpose, certificates and encryption that are already offered by the Azure cloud platform as standard can be used [54]. In addition, it is also important to look at the security in the communication between IoT devices and the cloud platform. Although this is the responsibility of the individual bindings, this is a recurring theme. Both existing literature (e.g. Chandu et all. [55]) and developments within Allego in the area of loaders, for example, can be taken into account.
In the case of charging stations, the possibility of securing them with certificates instead of a secured VPN tunnel is currently being examined. In connection with the EU General Data Protection Regulations, among other things, it has become increasingly important to secure IoT-data. [56]. After all, the data collected from individual vehicles or devices can also be seen as personal data. This is therefore an aspect that needs to be further investigated.

6.3.2. Version Management

Besides the development of new bindings, in the future also bindings will be updated. For the time being, the responsibility for compatibility lies with the developer. However, a mechanism for this can be developed in order to make it more automated. For example, continuous deployment solutions. But another point of research could be to dynamically determine the impact on an updated binding. After all, when the defined Triggers and Actions change, this may have an impact on the services that depend on a Binding.

6.3.3. Universal application - Binding Pattern

The structure set up in this thesis has proved to be very useful for the integration of vehicle data. This pattern is often applied in practice and may have a number of additional areas of application. During this thesis it was specifically applied to one domain. However, it would be useful to develop the concept in such a way that it can be applied to several domains. More generic solutions for a secure IoT solution can be considered. [57]
The binding pattern as set up for vehicles can easily be applied to other assets. For Allego the most important other asset is the charging station. Further research can look to the implementation of the platform developed in this research for IoT devices in general.

6.3.4. Other Markets

The research was initiated from within the Public Transport market. Within this market, there is a demand from operators to gain insight into their fleet and operations. However, the same principle can also be applied to other market segments. When an ecosystem is developed which allows to exchange data between vehicles and third parties, there will be great opportunities for Allego. A owner of a vehicle can choose to share his personal vehicle data, such as position and state of charge. With this data Allego can improve the service they are offering.
This can be done in a similar way to the permissions that users can give to apps on their smartphones. Ideally, the user installs an Allego app in his vehicles multimedia system. This app then ensures that its charging experience is improved. This can be done, for example, by referring the user to an available charger in good time and reserving this charger in advance. Google has applied for a patent on such a permission system to exchange data between different apps [58]. Such

a structure could also be set up for this purpose.

Currently, the integration of vehicle data is seen as a product only for the public transport market. However, it would be much more valuable to see the broader scope of application. As a CPO, Allego will not be providing a complete planning tool for Public Transport or other markets. Allego can, however, stay close to their core and provide information about the entire charging process. For example, it can be very interesting for lease companies to combine information about their vehicles with charging information. But it is not only within the EV domain that the platform developed is interesting. Insurers would also like information about the insured vehicles and their behaviour on the road [59]. In the Netherlands, for example, ANWB offers discounts on car insurance when you purchase an OBD-connector.

6.3.5. Practical Usability

In order to be able to apply the developed architecture practically for testimonials within Allego, there are still a number of practical issues that need to be solved.

- **Historic data**
  The current design is based on the exchange of real-time data. Events that come in are processed immediately. When a binding within the Platform registers itself, it may want to retrieve historical data.
  Although the data of the different data sources is now being stored, it is not possible to obtain this data in a universal way. It would be an idea to be able to 'play' the historical event stream in the future. This means that a new service can connect to the platform and then indicate the time from which this data must be forwarded.

- **Implementation framework**
  In order to speed up the development process for making a connection for new data sources and services, it is advisable to develop a framework on the basis of which the implementation of a new binding can take place. This framework can then be used as dependency, in the same way as contracts are shared now, so that a developer does not have to worry about sending and receiving messages from the framework.

- **Bindings**
  In addition to the integration of vehicle data with the services in the cloud platform, the developed architecture can also be used as an application for end-users. An example is that users can define their own Paths themselves via a graphical interface. A example of this scenario is already given in the implementation phase of this thesis.
  By defining new paths, a user can easily be informed of changes to the vehicle. In order to make the system practically applicable for this purpose, however, a number of Bindings still need to be developed. This could include sending an email or sending text messages. This allows end-users to configure in an IFTTT-like way what happens when there is a change in vehicle status. The core architecture supports this kind of use cases.

- **Specification of contracts**
  This final project has mainly ensured that an architecture in which services and data sources are loosely coupled. One of the most important results is the principle that can be applied for this purpose. However, there is still some work to be done in further developing the communication between the Bindings and the core platform (PathEngine).
  In order to be applied in a practical way, the way in which information is exchanged must be improved. this applies in particular to for the defined Types, by defining a wider range of different types it is easier for a developer to develop a new binding. Furthermore, the installation process needs to be improved.

- **Services**
  New services can be developed on the basis of vehicle data. Prediction models can be developed on the basis of historical data, making it possible, for example, to reserve charging stations dynamically baed on a combination of historical behaviour and current circumstances. In this way, vehicle data can ultimately ensure that the experience of EV-drivers is improved. New big data and machine learning techniques can help to improve existing services.

Bibliography

[1] How ibm's 5150 pc shaped the computer industry - cnet. https://www.cnet.com/news/how-ibms-5150-pc-shaped-the-computer-industry/, august 2011. (Accessed on 08/08/2018).

[2] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. Theworld-wide web. In *Readings in Human–Computer Interaction*, pages 907–912. Elsevier, 1995.

[3] Daniel Miller and Don Slater. The internet: an ethnographic approach. 2001.

[4] Thanksgiving one of the reasons aws got into cloud computing | linkedin. https://www.linkedin.com/pulse/20141021145904-5415076-how-did-amazon-com-get-into-cloud-computing/, oktober 2014. (Accessed on 07/11/2018).

[5] Amsterdam Roundtables Foundation and McKinsey & Company. Electric vehicles in europe: gearing up for a new phase? 2013.

[6] Paul Benjamin Fleischer, Atso Yao Nelson, Robert Adjetey Sowah, and Appah Bremang. Design and development of gps/gsm based vehicle tracking and alert system for commercial inter-city buses. In *Adaptive Science & Technology (ICAST), 2012 IEEE 4th International Conference on*, pages 1–6. IEEE, 2012.

[7] Gustav Wiklander. Real time tracking in truck fleet management, 2017.

[8] Car Connectivity Consortium. Building car data ecosystem. *URL https://carconnectivity.org/*, 2017.

[9] Cardatafacts.eu - safe and secure access to vehicle data.

[10] Jens Schmutzler, Claus Amtrup Andersen, and Christian Wietfeld. Evaluation of ocpp and iec 61850 for smart charging electric vehicles. *World Electric Vehicle Journal*, 6(4):863–874, 2013.

[11] Rijksdienst voor Ondernemend Nederland. Elektrisch vervoer in Nederland. page 22, 2015.

[12] Statistics electric vehicles in the netherlands (up to and including june 2018). july 2018.

[13] Carlos Calvo Ambel, Thomas Earl, Samuel Kenny, Stef Cornelis, and Jori Sihvonen. Roadmap to climate-friendly land freight and buses in Europe. *Transport & Enviroment*, Jun 2017.

[14] Michael Wang. The greenhouse gases, regulated emissions, and energy use in transportation (greet) model: Version 1.5. *Center for Transportation Research, Argonne National Laboratory*, 2008.

[15] Provincie Braband. Inspanningen provincie t.a.v. zero emissie. https://www.brabant.nl/politiek-en-bestuur/provinciale-staten/vergaderingen-ps/lis/20150317/download?qvi=53140. (Accessed on 08/20/2018).

[16] Sjoerd Bakker and Rob Konings. The transition to zero-emission buses in public transport–the need for institutional innovation. *Transportation Research Part D: Transport and Environment*, 2017.

[17] Vdl bus & coach - 1,000,000 electric kilometres in eindhoven. `http://www.vdlbuscoach.com/News/News-Library/2017/1-000-000-elektrische-kilometers-in-Eindhoven.aspx`. (Accessed on 06/11/2018).

[18] Jiří Černý. Testing of Five Different Types of Electric Buses. (October), 2015.

[19] R Beekman and R Van Den Hoed. Operational demands as determining factor for electric bus charging infrastructure. 2016.

[20] Heliox | public transport. `https://www.heliox.nl/markets/public-transport`. (Accessed on 02/20/2018).

[21] ElaadNL. EV Related Protocol Study v1.1. page 95, 2017.

[22] B Cho. Control of a hybrid electric vehicle with predictive journey estimation. 2008.

[23] A Vasebi, SMT Bathaee, and M Partovibakhsh. Predicting state of charge of lead-acid batteries for hybrid electric vehicles by extended kalman filter. *Energy Conversion and Management*, 49 (1):75–82, 2008.

[24] Fengchun Sun, Xiaosong Hu, Yuan Zou, and Siguang Li. Adaptive unscented kalman filtering for state of charge estimation of a lithium-ion battery for electric vehicles. *Energy*, 36(5):3531–3540, 2011.

[25] Rahul Mehta, Dipti Srinivasan, Ashwin M Khambadkone, Jing Yang, and Anupam Trivedi. Smart charging strategies for optimal integration of plug-in electric vehicles within existing distribution system infrastructure. *IEEE Transactions on Smart Grid*, 9(1):299–312, 2018.

[26] Thomas Conway. On the effects of a routing and reservation system on the electric vehicle public charging network. *IEEE Transactions on Intelligent Transportation Systems*, 18(9):2311–2318, 2017.

[27] Joshua Puerta, Alfonso Brazález, Angel Suescun, Olatz Iparraguirre, and Unai Atutxa. Standardizing it systems on public transport: An eco-driving assistance system case study. In *Communication Technologies for Vehicles: 13th International Workshop, Nets4Cars/Nets4Trains/Nets4Aircraft 2018, Madrid, Spain, May 17-18, 2018, Proceedings 13*, pages 149–158. Springer, 2018.

[28] Car Connectivity Consortium et al. Mirror link. *URL http://www. mirrorlink. com/. Last visited*, pages 10–15, 2013.

[29] Knmi data centre - search. `https://data.knmi.nl/datasets`. (Accessed on 06/19/2018).

[30] Francesco Viti, Serge Hoogendoorn, Lambertus Immers, Chris Tampère, and Sascha Lanser. National data warehouse: how the netherlands is creating a reliable, widespread, accessible data bank for traffic information, monitoring, and road network control. *Transportation Research Record: Journal of the Transportation Research Board*, (2049):176–185, 2008.

[31] YK Tan, JC Mao, and KJ Tseng. Modelling of battery temperature effect on electrical characteristics of li-ion battery in hybrid electric vehicle. In *Power Electronics and Drive Systems (PEDS), 2011 IEEE Ninth International Conference on*, pages 637–642. IEEE, 2011.

[32] Anne H Ngu, Mario Gutierrez, Vangelis Metsis, Surya Nepal, and Quan Z Sheng. Iot middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*, 4(1):1–20, 2017.

[33] Christian Seeger, Kristof Van Laerhoven, and Alejandro Buchmann. Myhealthassistant: An event-driven middleware for multiple medical applications on a smartphone-mediated body sensor network. *IEEE journal of biomedical and health informatics*, 19(2):752–760, 2015.

[34] Jon Whittle, John Hutchinson, and Mark Rouncefield. The state of practice in model-driven engineering. *IEEE software*, 31(3):79–85, 2014.

[35] Martin Henkel and Janis Stirna. Pondering on the key functionality of model driven development tools: The case of mendix. In *International Conference on Business Informatics Research*, pages 146–160. Springer, 2010.

[36] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F Ferguson. *Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more*. Prentice Hall PTR, 2005.

[37] OSGi Alliance. Osgi-the dynamic module system for java, 2009.

[38] Michael E. Porter and James E. Heppelmann. How Smart, Connected Product Are Transforming Competition. *Harvard Business Review*, (November):64 – 89, 2014. ISSN 00178012. doi: 10.1017/CBO9781107415324.004.

[39] K Kreuzer et al. Openhab-empowering the smart home. *Openhab. org, Tech. Rep.*, 2013.

[40] Jiri Hosek, Pavel Masek, Dominik Kovac, Michal Ries, and Franz Kropfl. Universal smart energy communication platform. In *Intelligent Green Building and Smart Grid (IGBSG), 2014 International Conference on*, pages 1–4. IEEE, 2014.

[41] Smart home using arduino, raspberry pi and more. `https://www.slideshare.net/oriolrius/smart-home-using-arduino-raspberry-pi-and-more`. (Accessed on 06/11/2018).

[42] Matthew B Hoy. If this then that: An introduction to automated task services. *Medical reference services quarterly*, 34(1):98–103, 2015.

[43] Supachai Vorapojpisut. A lightweight framework of home automation systems based on the ifttt model. *JSW*, 10(12):1343–1350, 2015.

[44] When "if" is not enough: Superpowers for ifttt – philipp ebneter – medium. `https://medium.com/@pebneter/when-if-is-not-enough-55b6e57d8742`. (Accessed on 06/20/2018).

[45] Matthew B Hoy. Alexa, siri, cortana, and more: An introduction to voice assistants. *Medical reference services quarterly*, 37(1):81–88, 2018.

[46] A. Krylovskiy, M. Jahn, and E. Patti. Designing a smart city internet of things platform with microservice architecture. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 25–30, Aug 2015. doi: 10.1109/FiCloud.2015.55.

[47] Wes Lloyd, Shruti Ramesh, Swetha Chinthalapati, Lan Ly, and Shrideep Pallickara. Serverless computing: An investigation of factors influencing microservice performance. In *Cloud Engineering (IC2E), 2018 IEEE International Conference on*, pages 159–169. IEEE, 2018.

[48] G. McGrath and P. R. Brenner. Serverless computing: Design, implementation, and performance. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 405–410, June 2017. doi: 10.1109/ICDCSW.2017.36.

[49] G. Lawton. Developing software online with platform-as-a-service technology. *Computer*, 41 (6):13–15, June 2008. ISSN 0018-9162. doi: 10.1109/MC.2008.185.

[50] Azure event hubs vs azure messaging. `http://microsoftintegration.guru/2015/03/03/azure-event-hubs-vs-azure-messaging/`. (Accessed on 06/10/2018).

[51] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing*, pages 1–20. Springer, 2017.

[52] xkcd: Standards. `https://xkcd.com/927/`. (Accessed on 08/22/2018).

[53] Alexandros Labrinidis and Hosagrahar V Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12):2032–2033, 2012.

[54] Shijimol Ambi Karthikeyan. Introduction to azure iaas. In *Practical Microsoft Azure IaaS*, pages 1–38. Springer, 2018.

[55] Y Chandu, KS Rakesh Kumar, Ninad Vivek Prabhukhanolkar, AN Anish, and Sushma Rawal. Design and implementation of hybrid encryption for security of iot data. In *Smart Technologies For Smart Nation (SmartTechCon), 2017 International Conference On*, pages 1228–1231. IEEE, 2017.

[56] David Thomas Ford and Sreman Qamar. Seeking opportunities in the internet of things (iot):: A study of it values co-creation in the iot ecosystem while considering the potential impacts of the eu general data protection regulations (gdpr)., 2017.

[57] Duo Lu, Dijiang Huang, Andrew Walenstein, and Deep Medhi. A secure microservice framework for iot. In *Service-Oriented System Engineering (SOSE), 2017 IEEE Symposium on*, pages 9–18. IEEE, 2017.

[58] Gary A Morris, Scott M Belliveau, Esteban Cabrera, Rian Draeger, Laura J Dunn, Timothy Joseph Goldsmith, Hari Hampapuram, Christopher Robert Hannemann, Apurv Ullas Kamath, Katherine Yerre Koehler, et al. Systems and methods for inter-app communications, October 5 2017. US Patent App. 15/475,010.

[59] Philippe Baecke and Lorenzo Bocca. The value of vehicle telematics data in insurance risk selection processes. *Decision Support Systems*, 98:69–79, 2017.

List of Figures

List of Tables

A

Existing Platforms

There are several real world solutions which integrate vehicle data into an online platform for further analysis/monitoring. In this chapter examples of these solutions are given and are discussed shortly. The purpose of this chapter is to give an overview of the different type of solutions there are out there.

A.1. Viriciti

Viriciti is company which has a main focus on monitoring electric buses and optimize the operation. It gives PTO's insights in their EV's and their operations. This means that viriciti with their solution can collect and analyze all data from the CAN bus in the vehicle. Based on planning information, this can be linked to driver and route to gain insight into individual performance. The use of cruise control, brake pedal, air conditioning, etc. are elements which are considered.

Figure A.1: Viriciti - the datahub

Datahub

A hardware device developed by Viriciti which helps them to gather vehicle data from electric vehicles. This hardware device is illustrated in A.1. The device streams data from multiple CAN buses real-time to the cloud. This can connect with internet through onboard ethernet, WiFi or with a 4G connection. Beside this it has a hardware interface with several analog and digital in and outputs and a GPS antenna.

Docker

The datahub of Viriciti is a solution which uses Docker technology for their implementation. They call it AppLayer. With this AppLayer it is possible to build apps in almost every programming language. In a Meetup they have shown that this runs on docker and how they manage the platform. The box itself runs on Docker technology with several containers on top. This containers can be updated through a cloud solution. It is possible for third parties to develop several apps on top of the Viriciti platform for example to develop a passenger counting system which uses sensors connected to the hardware box. Since the datahub connects to the internal CAN-bus of a vehicle all data available here (measurements, commands) is available in the Viriciti cloud platform. Examples include State of Charge (SoC), brake position, battery temperature, etc. The information that is available as standard depends on the bus manufacturer. After all, the data must be placed on the can bus.

A.2. Microsoft Connected Vehicle

The Microsoft Connected Vehicle Platform provides a set of services built on top of the Microsoft Azure infrastructure. It is designed for OEM's of vehicles to create connected vehicle experiences. The Microsoft Connected Vehicle Platform is a combination of different components from Microsoft Azure. For Microsoft the current focus lies on vehicle OEMs. However the cloud components of the Connected Vehicle Platform could be interesting for a third party as well.

For the connected platform several key areas are defined:

- **Telematics and Predictive Services -** Telematics are the core function of a connected vehicle platform. Based on Azure IoT hub a scalable solution can be build. With the predictive services of predictive analysis of the gathered data can be performed.
- **Productivity and Digital Life -** Technologies like Skype and Cortana can help end user to improve their experience in the vehicle. For example voice-control to reduce distraction while being on the road.
- **Connected Advanced Driver Assistance Systems (ADAS) -** With the combination of real-time road and environment information the safety and performance can be improved. Can be integrated into an autonomous driving system.
- **Advanced Navigation -** Make it possible to customize the navigation based on end-users habits. With personalized routing and dynamic location based services the traditional navigation experience can be improved.
- **Customer Insights and Engagement -** With better insights into customers behavior the relationship can be improved with the integration of CRM and brand loyalty.

The Microsoft Connected Vehicle Platform is a tailor made solution. It an agile platform that starts with Microsoft technologies. The platform is targeted on businesses who want to accelerate in a connected car platform. Renault-Nissan is a OEM that develops in cooperation with Microsoft a connected car platform based on Microsoft Connected Vehicle for their new connected vehicles.

A.3. ANWB Connected

A recently launched product from the ANWB which makes it possible to read Vehicle data through the OBD connector in passenger cars. This device collects data from cars to make them connected. A.2 shows a marketing image from the ANWB website.

Figure A.2: The ANWB Connected dongle with the mobile interface

With ANWB Connected it is possible to monitor driver behavior and give an advice how to save fuel. This behavior is monitored with a built-in accelerometer. It is possible to read out error codes and a breakdown can prevented by monitoring these error codes. With a car that is not connected the garage can read out these error codes with a physical connection. For an end user it is possible to register trips in the ANWB connected app to register the driven kilometers for the dutch tax authorities. The ANWB integrated the ANWB Connected service to ANWB services such as their insurance and the ANWB Wegenwacht. You can get a discount on your car insurance with the usage of the ANWB connected.

The ANWB connected dongle is a plug-and-play device which makes it easy for end-users to install in their vehicle. This example shows an easy-to-use solution. Although our research initially focused on electric buses, it is interesting to look at solutions for passenger cars. In addition, a plug-and-play solution for commercial vehicles would exist in the ideal world.

A.4. OpenMatics

OpenMatics is a platform for vehicle telematics. They offer both hardware and software. With their on-board unit, they work together with OEM's to integrate these into their vehicles.
The software is based on the Microsoft Azure stack on which they develop different apps to get insights into the vehicle data gathered with their hardware devices. They allow third-parties to develop Apps for their platform in order to extend the functionality.
The architecture of this solution is more or less similar to the architecture of Viriciti.

A.5. TrustTrack

TrustTrack is a platform developed by Ruptela, a manufacturer of tracking devices. This platform makes it possible to track vehicles and see the status of these vehicles. Depending on the tracker configuration different types of information can be send to their own software solution or a cloud server of a third party. A.3 shows the TrustTrack interface. One car is shown on the map with available information of the vehicle concerned.

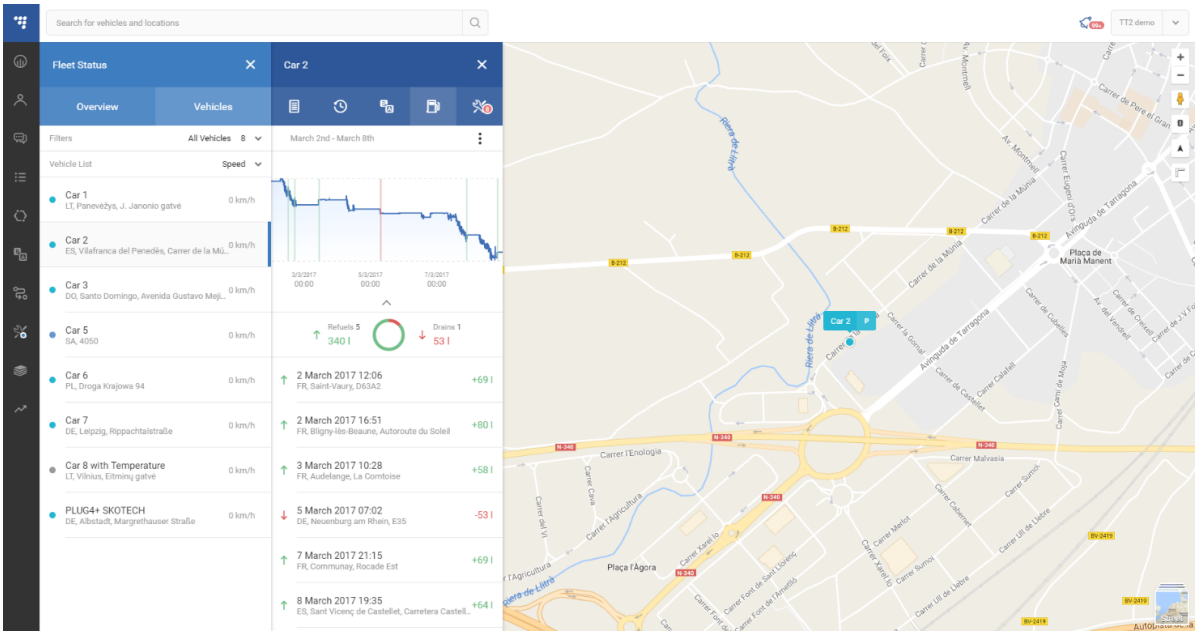
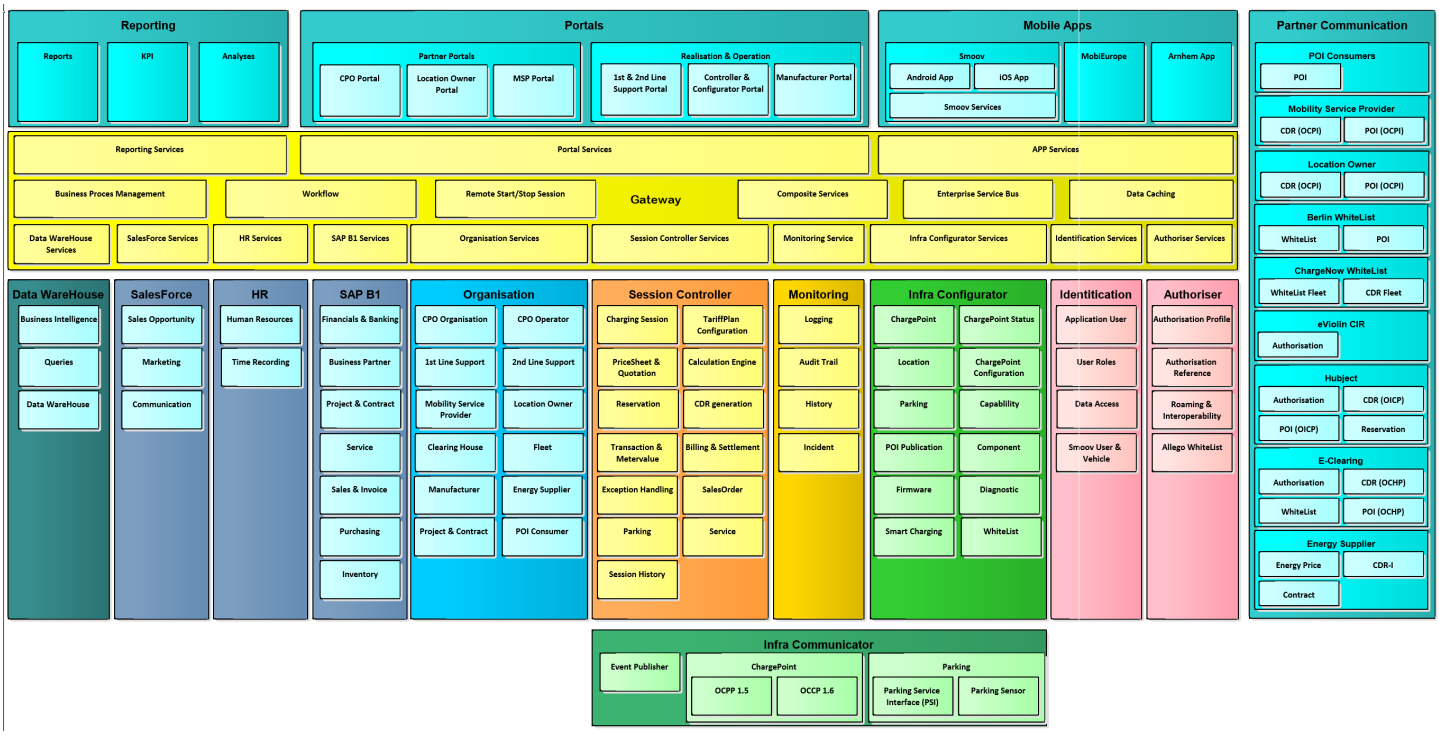


Figure A.3: Screenshot of the TrustTrack-interface

B

Software Services Allego



C

Vehicle Service - Interface Design

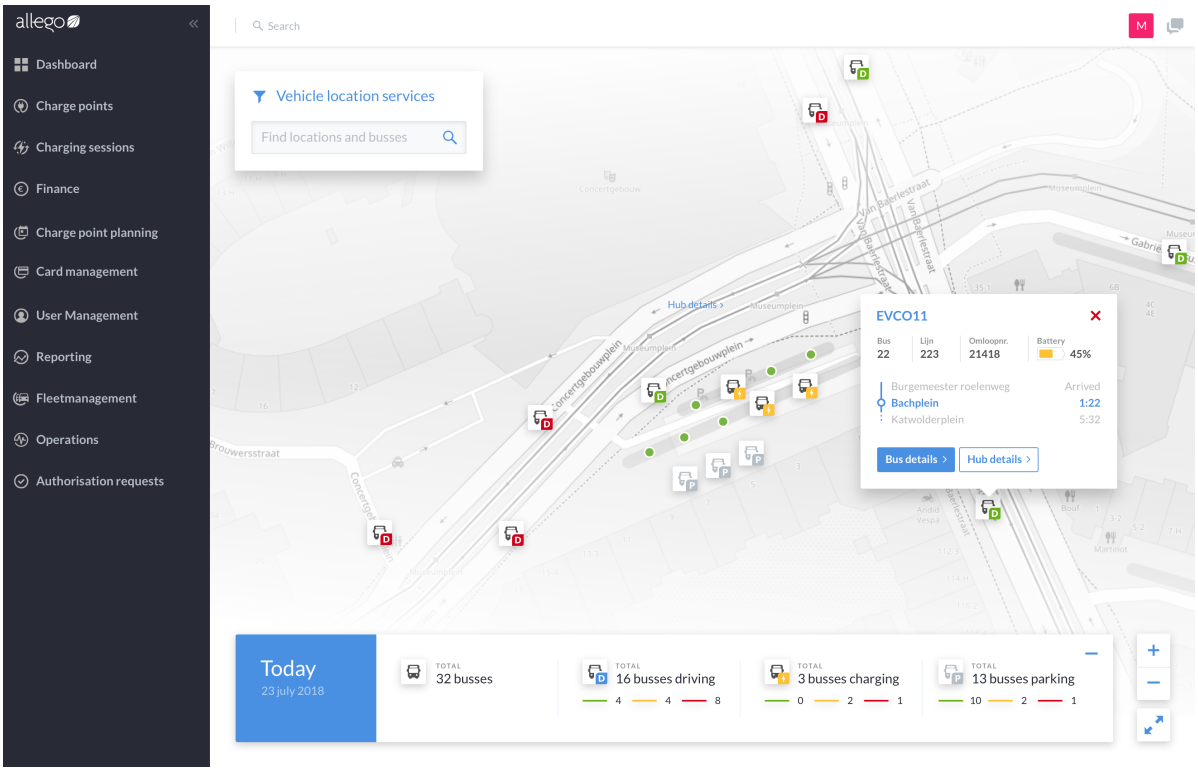


Figure C.1: Dashboard - Get a overview of the location of the buses and their current state (Charging, Parking, Driving). Additional vehiccle information is displayed in a popup box

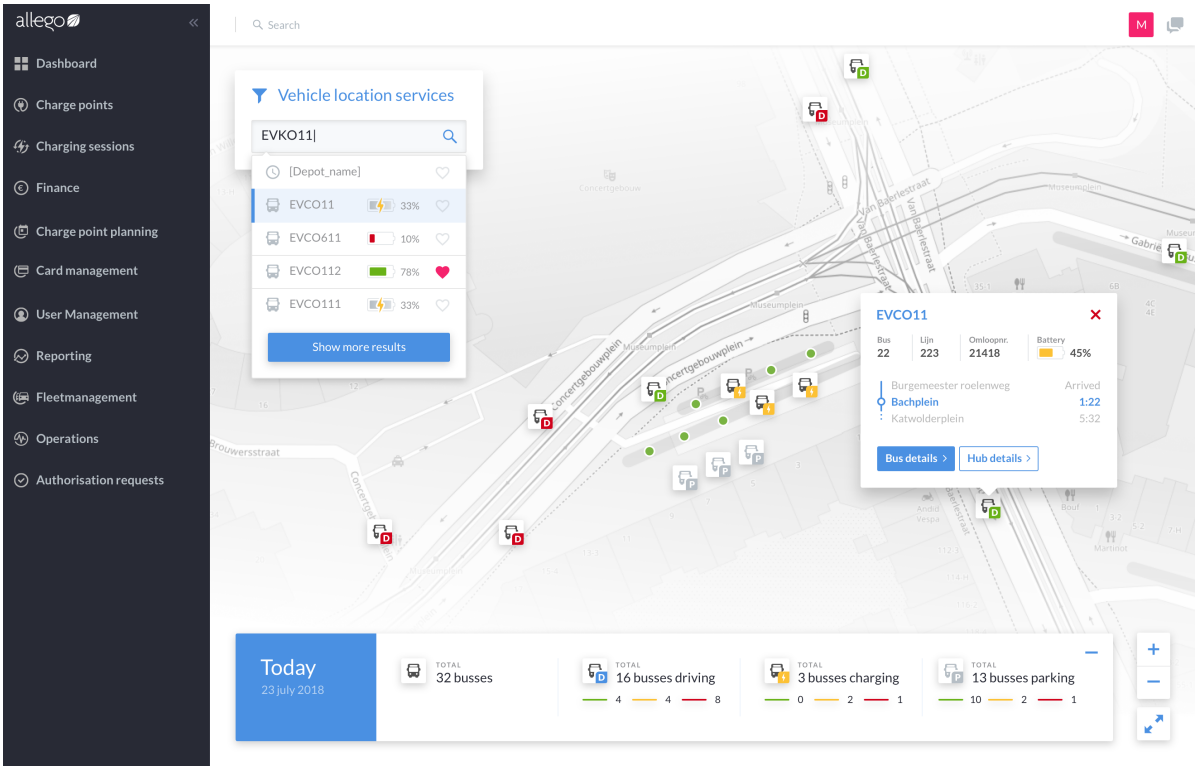


Figure C.2: Search - Possibility to search for specific vehicles or bus lines. Get direct a indication of the SoC of the vehicle

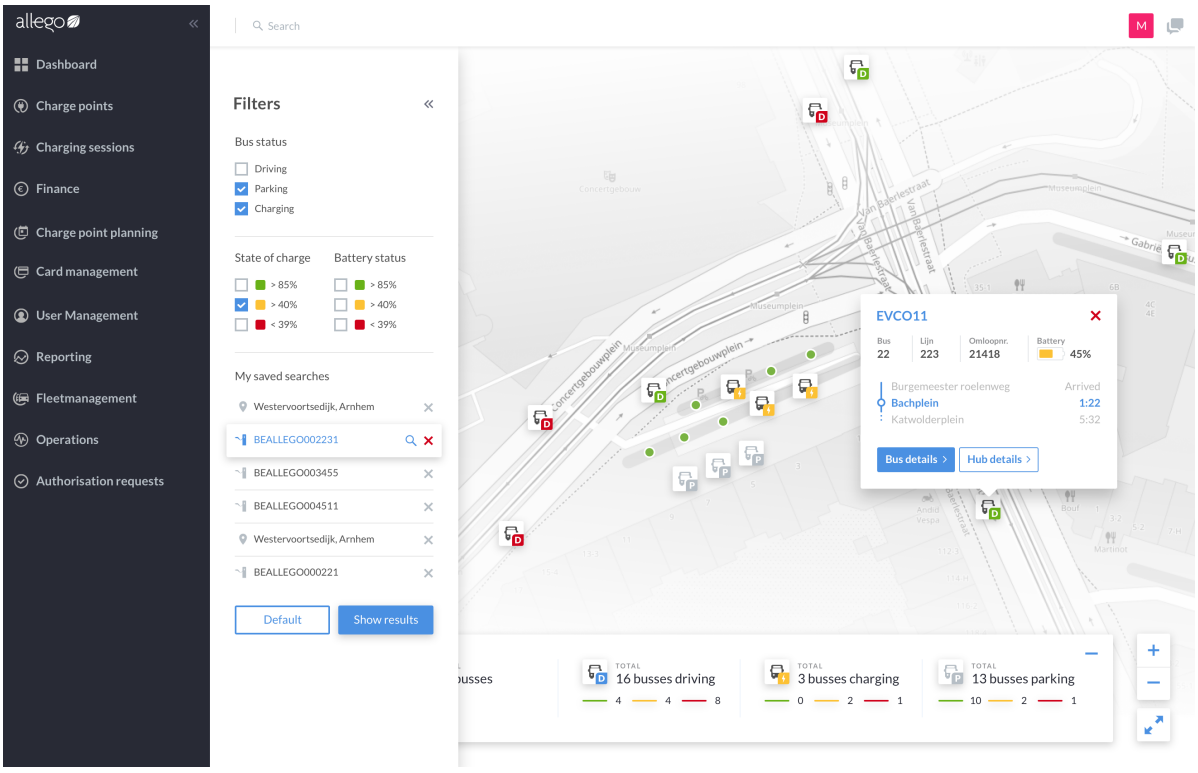


Figure C.3: Filter - A filter pane to quickly narrow down the displayed vehicles

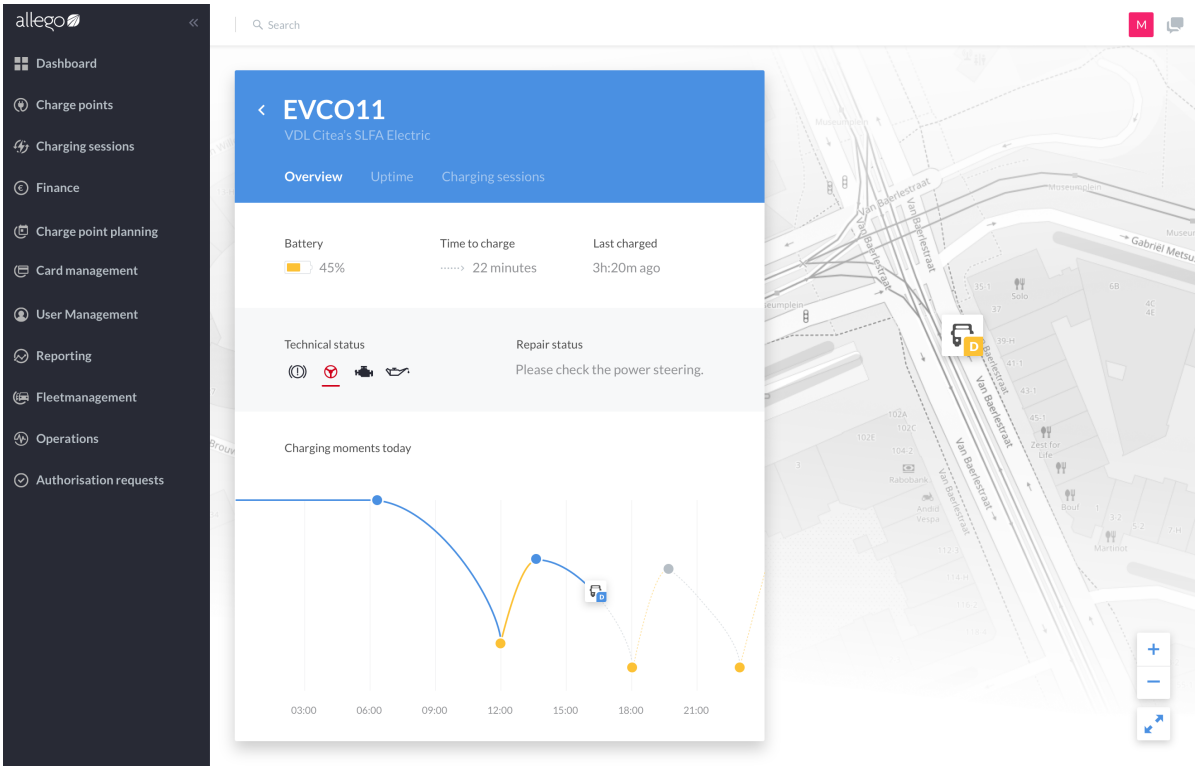


Figure C.4: Detail - Detail overview to get insights into the behavior of the bus. In this detail view a end user can easily check the charging sessions and aditional vehicle information

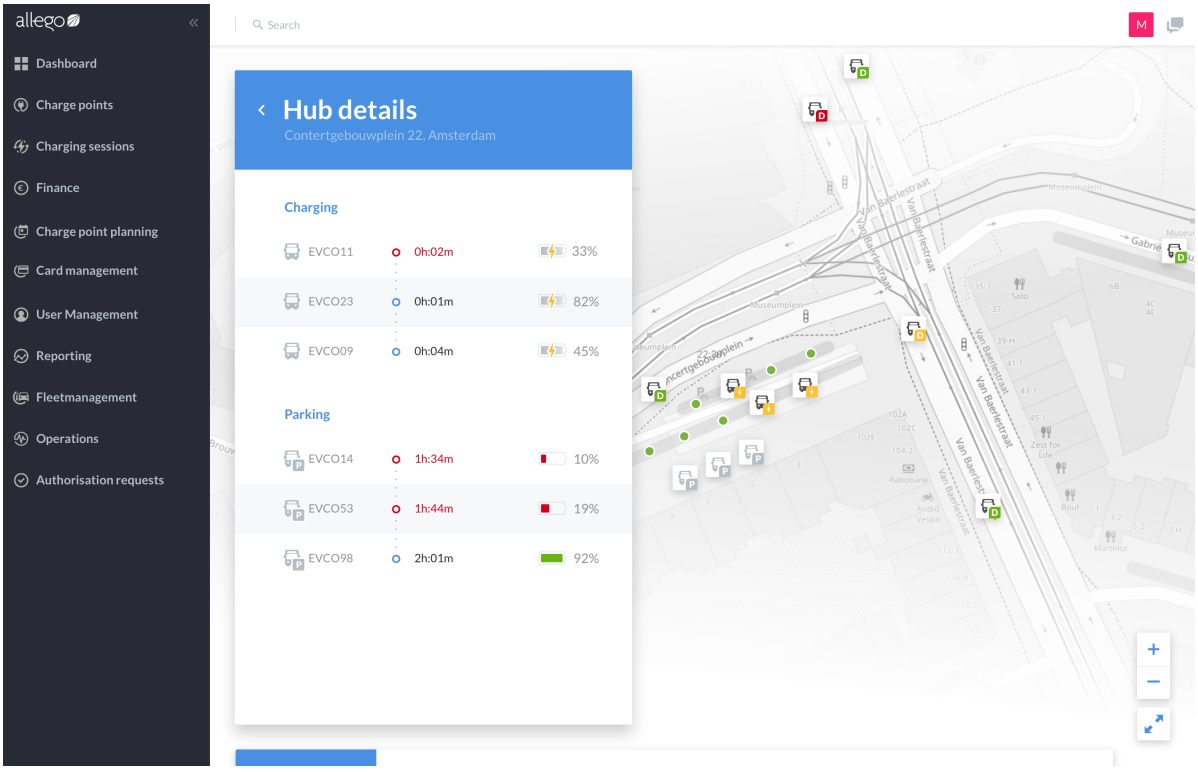


Figure C.5: Hub - Get a overview of the state of the vehicles in a depot/hub.