Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Digital Twin for Fleet Management of Connected Vehicles

Jan Patrick Gerhards

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Holger Schwarz
Supervisor:	Dr. Pascal Hirmer
Commenced:	April 15, 2021
Completed:	October 15, 2021

Abstract

Connected cars are capable of communicating with other devices (e.g., other vehicles, smartphones, road side units) in their environment. The domain in which they operate is a highly dynamic one. Reasons for this include changing demands of users and heterogeneity of technologies and interfaces. Due to this, developing a concept for a digital twin of a connected car that is capable of operating under these conditions is a challenging task. Especially handling of data and behavior of connected cars is an area of interest.

In this thesis, we propose a digital twin architecture which can be used to implement connected car scenarios. We analyze challenges of the domain and assess in which ways and under what circumstances connected cars may interact. Based on these expectations, we introduce our concept for a digital twin. Our main goal is to provide interoperability between different car manufacturers, while still allowing developers to implement functionality in a way they deem appropriate.

In order to verify our concept, a prototype of a digital twin is implemented. We explain this implementation and share experiences gained during this development. Finally, we draw conclusions on the concept as a whole and how it may be used in practice.

Contents

List of Figures

Acronyms

AAS Asset Administration Shell. 13

AdC Adapter Commander. 34

AM Adapter Manager. 30, 36

AMM Adapter Manager Message. 39

AW Adapter Watcher. 30, 33, 37

CF Cloud Fragment. 30, 32

CL Communication Layer. 30

CLC Communication Layer Coordinator. 38

EH Event Handler. 35

IA Information Aggregator. 34

ICH Internal Communication Handler. 38

IDT Internal Digital Twin. 29, 31

IoT Internet of Things. 17

LIDT Local Internal Digital Twin. 32

MBP Multi-purpose Binding and Provisioning Platform. 7

MD Message Distributor. 34

MH Message Handler. 38

OPC Open Platform Communications. 14

OPC UA OPC Unified Architecture. 14

QoS Quality of Service. 43

RqC Request Constructor. 38

SOA Service-Oriented Architecture. 15

VANET Vehicular ad hoc Network. 17

1 Introduction

Today, the Internet of Things is an important trend leading to the interconnectedness and integration of different devices. This takes place on both a small scale, for example light controls in a smart home, and on a much larger one. The idea of smart cities is one instance of this approach on a larger scale, where devices cooperate in order to reach common goals [AVF20; MM17; WF15].

In order to work, most Internet of Things applications use data from different devices to automate processes. It is also possible to use this information as input for another device. Depending on how it is used, it is important to manage and present data in an appropriate way. For example, if devices which should communicate with each other do not support the same data format, no value would be added to the system, since no information could be exchanged.

One domain in which the idea of things communicating is being adopted are connected cars. A *connected car* is a car able to communicate with other devices [CM16]. The goal of this is to exchange information for use in different applications. Due to the large amount of data relevant to cars and their environment, it stands to reason that handling information in an efficient manner is of great importance. This can be especially crucial in traffic situations where many cars communicate with each other at the same time, for example in an inner city. Another important factor to consider in this domain is that different car manufacturers may use varying technologies and standards to communicate. A Reason for this might be different priorites regarding communication aspects.

As more cars are becoming connected, it seems likely that car manufacturers will begin to implement new features and also try to introduce unique selling points. This can be facilitated by a well-designed architecture for the software of a connected car. One concept which seems promising for these requirements is a *digital twin*, a software representation of a real-world object, which is synchronized to include the current state of the physical object [SKZ+20]. Currently, digital twins are mostly used in the context of smart manufacturing in the manufacturing industry. They provide information on a physical object and can be used in different phases of the lifecycle of a product. The approach to use a digital twin as a representation of a connected car may make it easier to access relevant data. However, some aspects are different compared to industry use. For example, the domain in which cars operate is much more dynamic regarding potential communication partners. Furthermore, while communication standards in the manufacturing industry are already well-established (e.g., OPC-UA), this is not yet the case for connected cars. These factors all have to be considered when designing a concept for a digital twin in the domain of connected cars.

In this tesis, a concept and architecture for a digital twin of a connected car are proposed. They are based on an assessment of challenges relevant to the domain and possible use cases in which such a digital twin may be used. This includes considerations on how functionality provided by software in a connected car can benefit stakeholders. As a way to verify the concept and architecture, a prototype was developed and tested. The results of this test and experience gained during the implementation are included in the thesis. Additionally, an evaluation of how the proposed digital twin could be used in practice was carried out.

2 Related Work

In this chapter, related work is described which either has similar goals to this thesis or serves as foundation for understanding the domain and its use cases.

2.1 Asset Administration Shell

The Asset Administration Shell (AAS) is a concept that was introduced in 2015 as part of the Reference Architecture Model for Industrie 4.0. In the context of Industrie 4.0, it is also sometimes called administration shell. Since its introduction, AAS was often subject of further discussion and its specification was improved multiple times [21; BML+20].

According to Dorst et al. [DGH+16], Industrie 4.0 components can be based on objects with "*at least passive communication ability*" [DGH+16]. This is required to make the properties of such a component available. In addition, virtual representation and technical functionality are mentioned as factors of an Industrie 4.0 component. The virtual representation of an object contains its data and meta information about it. Furthermore, obligatory data of an Industrie 4.0 component is also stored in this representation. The technical functionality can be used to include functions relevant to the object, for example, planning or other functionalities relevant to the business logic. The intention of the AAS is to "*turn[...] an object into an Industrie 4.0 component*" [DGH+16]. In order to achieve this, an administration shell can be used to surround an object, which would not be an Industrie 4.0 component on its own [DGH+16]. Communication between the AAS and the object uses an internal interface that can be manufacturer-dependent. A standardized external interface is used by the AAS to manage the communication with other systems. Both interfaces in combination make the concept of AAS able to be standardized, while still being able to support different functionalities of an entity [TA17].

The structure of the AAS contains a body and a header. A manifest is a part of the header and contains important information. Administrative information regarding communication, identification, and functionality are all included in this manifest. Additionally, it contains an index of submodels, which together with a component manager make up the body of the AAS and characterize the associated asset. The body of the AAS contains the component manager. This is an interface used to access the payload of the submodels that can also include other capabilities [TA17].

Submodels are meant to be standardized according to certain characteristics. Every single one should represent an aspect of the asset. The general idea is to include multiple submodels in an AAS, which makes it possible to represent different aspects of an asset. If a submodel is available within the body of an AAS, it can be assumed that some relevant properties are available [BBE+17]. There are multiple types of submodels. They can be a data model, a group of functions, or represent another component, thus allowing nesting of components. Every submodel contains an own header and a body. The header is standardized and contains information about the specifics and capabilities

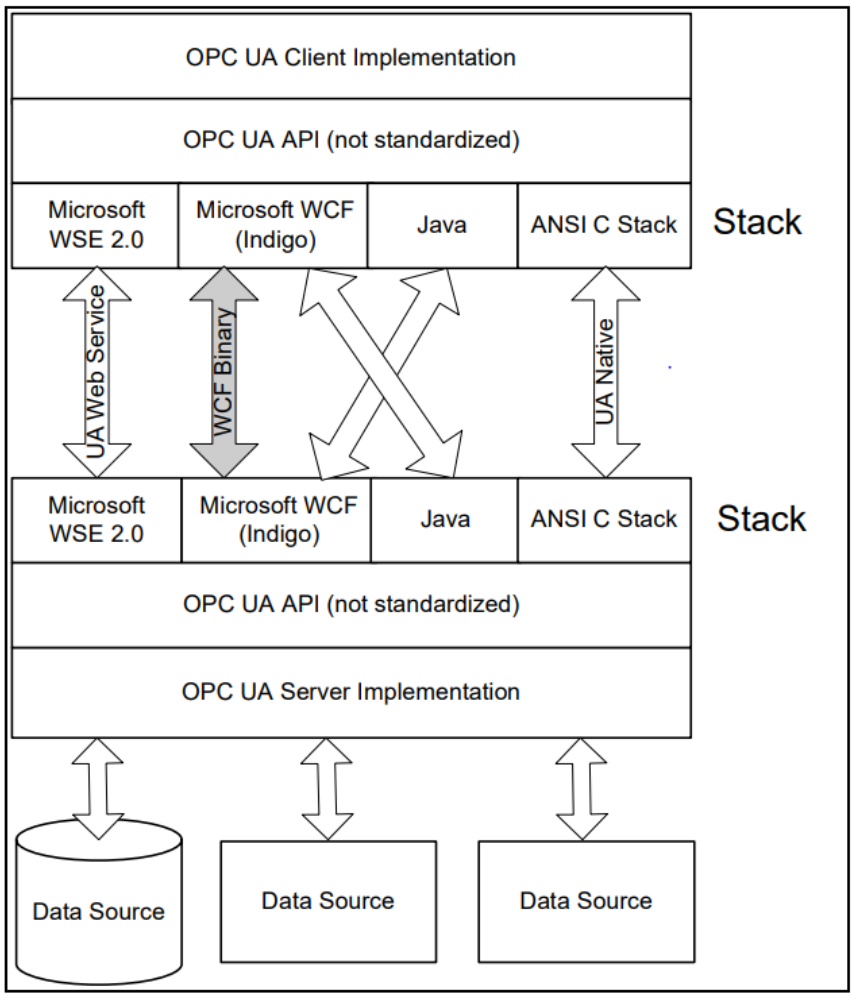

Figure 2.1: OPC UA Architecture [LM06]

of the submodel. All headers of submodels are included in another manifest, which is part of the body of the AAS. The body of a submodel contains the payload, which stores information and is capable of executing methods [TA17].

AAS itself specifies an information model in UML that is technology-neutral. From this model, formats for different technologies can be derived and used in practice [BML+20]. One such technology is OPC Unified Architecture (OPC UA), which will be explained in more detail in the next section. This is done in an effort to show an example of what kind of technology can be used to realize the concept of AAS in practice.

2.1.1 OPC UA

The Open Platform Communications (OPC) specification was used as a basis for OPC Unified Architecture (OPC UA) by the OPC Foundation. OPC UA is a platform-independent standard with the goal of "*provid[ing] interoperability in process automation*" [LM06]. Using the standard, it is possible for different kinds of systems or devices to send request and response messages between

clients and servers. Alternatively, the publish-subscribe model is also supported by OPC UA. In addition to data access, OPC UA is capable of offering functionality regarding other aspects like access control or alarming [Fou17a; SGBP16].

The architecture of OPC UA is illustrated in Figure 2.1. It is based on servers and clients, which communicate using their respective communication stacks. OPC UA defines abstract services in order to provide a Service-Oriented Architecture (SOA). Based on these definitions, a server defines which services it supports. Using the services provided by a server, it is possible to access its *AddressSpace*. The AddressSpace contains the objects and related information made available for clients [Fou17a; LM06]. Clients are able to access servers using their own communication stack using request messages, which are based on OPC UA service definitions. It is important to note that for communication stacks to be able to work together, they have to use the same technology mapping. A compatible server-side communication stack is able to receive messages from clients. Requests included in them are delivered to the implementation of the functionality provided by the server. This is where the logic needed to return the response is implemented. The data accessed by the server implementation can be provided by another underlying system [LM06].

In an effort to enable everybody to create their own communication stack with its own API, the OPC Foundation standardized only the communication between different communication stacks [LM06]. Consequently, a large part of the OPC UA specification is written independent of technologies used to implement it. However, all communication is based on messages with a defined content [Fou17b]. Furthermore, part six of the OPC UA specification [Fou17b] defines mappings, which can be used to implement OPC UA using specific technologies.

Using the AddressSpace model of OPC UA, it is also possible for clients to query the server about metadata. This metadata contains information about the format of data provided by the server. The motivation for this is the possibility to make clients "*able to determine the format at runtime and properly utilize the data*" [Fou17a]. In this model, data is displayed as connected nodes. Nodes contain different kinds of meta information, for example, used for identification or description, and are connected using relationships [LM06]. Relationships are realized using references and can be used by a server to "*present data in a variety of hierarchies tailored to the way a set of Clients would typically like to view the data*" [Fou17a]. This is done in order to allow OPC UA to be used for different domains and provide interoperability. For example, an information model allowing OPC UA to be used in the domain of AAS is described in [4020; Fou17a].

2.2 C2PS

C2PS takes its name from cloud-based cyber-physical systems. It is an architecture reference model for a digital twin within this context that was developed by Alam and Saddik [AE17]. An important principle of the concept is physical *things*, which are represented by associated digital twins (also called cyber things) in the cloud. In the environment in which the system is assumed to be operating, multiple "*independent systems connect together to perform a common goal*" [AE17]. It is also expected that there is always a network connection available.

The general architecture model proposed by C2PS is divided into five layers. The lowest layer contains the *Physical Things*. For every one of these things, there exists a one-to-one connection to a digital twin. Every physical or digital thing can be identified by a unique ID. Digital twins make

up the second layer of the reference model, which is called the *Cyber Things Layer*. Changes to the physical thing are observed using sensors and cause a data store located within the physical thing to be updated. Similarly, the digital twin also contains a data store, which should be updated whenever a change to the physical object takes place. Should information relevant to the physical object be received by the digital twin, it should also be included in the data store of the physical thing. Whenever communication takes place, this is taken into account to make sure important information is stored by the physical thing as well as the digital twin [AE17].

Things can communicate using different layers. "*[D]irect ad-hoc communication*" [AE17] is possible within the physical layer. For communication between digital twins, the *Peer-to-peer Relation Layer* is used. This layer is located on top of the Cyber Things Layer and contains communication groups. Such a group can be identified by a relation ID and communication within it is sent to all members. It is realized as a networking groups of the digital twins on the cloud. A digital twin can be a member of multiple groups at the same time. Hybrid use of both types of communication is possible using this architecture. This makes it feasible for smart things to use their current state to choose the preferred type of communication [AE17].

Within in the cloud, C2PS contains the *Intelligent Service Layer*. It acts as middleware by coupling cyber things, their relations, and related ontologies together. By using ontologies and knowledge of the smart things, the Intelligent Service Layer can initiate a reconfiguration of the system. Another capability of this layer is grouping services together. In order to do this, a *Service Integrator* is used to form what is called a *mashup service* [AE17].

Finally, the data center providing the cloud used in C2PS can offer additional information. This is the highest layer within the proposed architecture and is composed of *System Administration* and *System Usage*. It includes data mining or monitoring functionalities, for example, by providing services used to create summaries [AE17].

By using this architecture, three different types of cloud setup can be distinguished. The one is formed by ad-hoc communication in the physical layer. In it, physical things communicate using their communication infrastructure and (near) real time services can be provided. Services that can tolerate delays and can use cloud infrastructure are located in the second kind of cloud setup. On this level, it is possible to upgrade digital twin processes or use them to improve functionalities. The physical layer is capable of accessing the services provided at this level. In order to achieve this, it is assumed that physical things will communicate with the digital twin using cellular networks. Peer-to-peer connections between digital twins may also be used to access services within this setup. The last proposed setup is provided by the data center. Various data mining applications can use cloud services and provide information to the physical layer or monitoring authorities [AE17].

Another interesting concept included in this proposal is the *Services Manager*. It can be used to manage data privacy and visibility of things. For every smart thing, the owner is able to choose who can use the data generated by it. It is also possible to cut off access for the digital twin entirely, causing the physical thing to be cut off from the cloud [AE17].

2.3 Conclusion

The concept of digital twins has been explored often in the context of the manufacturing industry, with a very important initiative being the German Industrie 4.0. It introduced the concept of AAS, with which it is possible to manage different submodels providing information relating to different aspects of an asset. Within the domain of general Internet of Things (IoT), there are also some interesting concepts. A notable one is the previously discussed C2PS. It illustrates very well how digital twins can interact among themselves and with physical things. Additionally, it shows how communication between devices is possible on different levels.

Within the domain of connected autonomous cars, only few concepts regarding architecture of a digital twin are available. Instead, it is important to note ways in which cars may communicate in the future and what information needs may arise. Two important concepts are the Vehicular ad hoc Network (VANET) and the *Vehicular Cloud*, which are discussed by Gerla et al. [GLPL14]. This shows that connected cars may, in theory, be capable of collaborating with other nearby vehicles and acting as intelligent agents. A different concept illustrating possibilities of this domain is proposed by Mariani and Zambonelli [MZ20]. In it, the authors discuss using different levels of autonomy depending on the current traffic situation. When using this concept, connected cars may have different information needs during operation, depending on the currently chosen level of autonomy. Therefore, a digital twin has to take into account changing information and communication needs, which are not known at the beginning of a route.

This thesis aims to expand upon these concepts of connected cars, which presents a very dynamic domain. The goal is to provide a reference architecture for a digital twin designed specifically to handle the relevant complexities. In this architecture, elements of AAS and C2PS will be included. This is done to combine the advantages of being able to offer multiple submodels with the design for communication on different levels. A prototype will be developed to validate this approach and gain a better understanding of the architecture when applied in practice.

3 Challenges and Goals

In this chapter, some definitions necessary in the context of this thesis are introduced. Furthermore, the expected domain in which the proposed digital twin will operate is characterized. This includes challenges that have to be taken into account by the concept and architecture. Based on these factors, goals for the project are formulated.

3.1 Definitions

In order to better understand the ideas presented in this thesis, some common concepts have to be defined. These will be used as a basis for later sections. This section will also contain motivation for the definitions, since digital twins are defined in multiple ways in other publications.

3.1.1 Connected Car

According to Riccardo and Morisio [CM16], there are different definitions of a connected car. These definitions highlight contrasting subsets of elements, as modern cars are equipped with different types of connections. They go on to define a connected car as a vehicle "*capable of accessing the Internet at any time*", which is "*equipped with a set of modern applications and dynamic contextual functionalities*". Furthermore, a connected car is able to interact with other smart devices or other vehicles. This definition is used as a basis for the one of connected cars used in this thesis. In contrast to Riccardo and Morisio's definition, the connection to the Internet is not necessary. The motivation for this is that cars, which are only able to communicate with vehicles or other devices in their vicinity, but do not support Internet connectivity, would otherwise not be considered connected. Furthermore, the functionality included in connected cars is not limited to Infotainment features. This makes it possible to include autonomous cars in this definition, as they could also use communication to coordinate traffic.

Definition 3.1.1 (Connected Car)
A connected car is a car capable of communicating with other devices or vehicles in its vicinity. It is equipped with modern applications providing features like Infotainment or enabling decision-making by the car. Furthermore, a connected car may be capable of accessing the Internet. Using this connection, it can provide other features or upload data, which can be used by various stakeholders.

3.1.2 Digital Twin

The concept of a digital twin was first presented in 2003. However, today, it is still not fully established, despite increasingly attracting attention [MLC20; SKZ+20]. A digital twin is dependent on the domain it is used in and on the technologies used to realize it. As a consequence, "*[t]here*

is no universally accepted definition of [digital twin]" [SKZ+20]. Instead, the vision of a digital twin evolves constantly and digital twins have to be tailored to fit the domain [SKZ+20]. Minerva et al. [MLC20] mention that the definition has moved away from industrial artifacts or products, instead becoming a more generic notion, which can be applied to different physical objects.

Still, some attempts at defining a digital twin have been made. In a survey by Minerva et al. [MLC20], a simple definition of the concept is used as a starting point. According to it, a digital twin is a "*comprehensive software representation of an individual [physical object]*". In it, properties, conditions and behavior of the object are included using models and data. They also point out that the digital twin "*is a set of realistic models that can be used to simulate an object's behavior*" and it "*represents and reflects its physical twin and remains its virtual counterpart across the object's entire lifecycle*". Later on, they remark that a digital twin links a real entity to a softwareized one. The real entity in this context is relevant to the physical world and the softwareized one is executed in a virtualization space.

A different approach to defining a digital twin is chosen by Sharma et al. [SKZ+20]. They focus on the properties of a digital twin as a way of defining it. As necessary properties, they mention a real-time connection with a physical asset, self-evolution, continuous machine learning analysis, availablity of data over time (to be used by machine learning), and domain dependence. Other possible properties, that are not considered necessary but can be used to create a hierarchy of digital twins, are autonomy and synchronization.

A notable commonality between both approaches is the connection to a physical object, about which data should be made available. Furthermore, Minerva et al. and Sharma et al. agree that a digital twin is dependent on the domain it is used in. The inclusion of autonomy as a possible property of digital twins by Sharma et al. is especially interesting in the domain of connected cars. According to them, a digital twin could be able to change a physical asset.

Within this thesis, the focus is on connected cars in use. Due to this, only the use as a car will be regarded. Any other phase of the lifecycle of the product, for example, conception and design, will be disregarded. Furthermore, as connected cars are closely linked to autonomous cars, a digital twin is not only considered a representation of a car, but also an endpoint for communication and coordination with it. This was also a motivation to increase the focus on the software used to make decisions for the connected car. With this in mind, a digital twin is defined as follows:

Definition 3.1.2 (Digital Twin)
A digital twin is a software representation of a physical object. It contains data about the digital twin and is the endpoint for communication with the object. Depending on the capabilities of the object, possible communication can range from data requests to complex coordination of behavior. Behavior of the object is controlled by the digital twin, making the software used to make decisions for the object part of it. Some additional functionalities, for example, simulations, can also be provided by the digital twin. If the digital twin requires information not concerning its associated object, it is able to communicate with other systems in order to acquire this information. While a digital twin may be operated on hardware of the physical object, this is not necessary and a digital twin is geographically independent of its physical twin.

3.2 Connected Cars and Their Expected Use in the Environment

Currently, many new cars are delivered with the ability to use an Internet connection in order to provide data to users. Some cars are even able to react to simple commands issued by owners using special apps or similar interfaces. We expect the amount of data and functionality made available to become both more numerous and more complex in the future.

Knowing this current trend, it is easy to see how car manufacturers in the future may also be able to differentiate themselves using software. By providing different functionalities than their competitors, they may be able to provide a unique selling point to customers. An advantage of this approach is that theoretically, software can be upgraded after the sale of a car. While this may also be possible with hardware, it takes a lot of effort and can inconvenience the user of the car. In contrast, installing new software using updates over the Internet may cause only little to no disruption for users.

One factor that has to be taken into consideration when implementing new features is the available data. Not all required information of the environment may be available to the car by itself. This is where interaction offers opportunities. By exchanging data, connected cars can obtain more information, which can be used for new features. This data exchange may take place not only between different connected cars, but also between a connected car and another device. One example for such a case would be a traffic light communicating with the stopped cars to evaluate waiting times and priorities.

Another opportunity of connected cars is linked with the trend to increasingly autonomous cars. In order for cars to be able to drive by themselves or offer advanced drivers assistance systems, large amounts of data have to be acquired and processed [Wol16]. Cars that are able to communicate with each other may be able to exchange relevant information. This can reduce the amount of data collection that has to be performed by a single car. If the shared data is already processed to conform to some common standard, the overall needed processing power decreases even further. Safety can also profit immensely from connected cars. When cars are able to communicate safety-critical information, assistance systems and autonomous cars can react to dangerous situations before they could perceive them without cooperation. In some cases, it might even be possible to avoid a dangerous situation entirely.

These opportunities and scenarios illustrate why a user would prefer a connected car. They also show that the environment in which such a car would operate is a very dynamic one. Large amounts of data are shared by multiple cars and other different devices in order to provide features and improve safety. Furthermore, a car in this domain will have to be able to communicate with multiple other devices at the same time. A crucial factor that cannot be overlooked is the priority of different features supported by this concept. For example, while a connected car may be able to use communication to provide an enhanced Infotainment system, messages regarding safety should always be prioritized.

As more cars will be able to communicate, it seems likely that regulation and standards will be established. Especially for safety-critical functionalities, regulation requiring certain types of data to be provided or cars to be able to react to specific signals may be passed. Another aspect which could influence the communication and information needs of connected cars is the physical environment. This includes the current context in which it operates. For example, when driving on a highway, different information may be needed compared to inner city traffic.

Finally, as connected cars provide increasingly complex software, we expect update cycles to become shorter. One of the reasons for this is the necessity to react to regulations with new software. Apart from that, car manufacturers may also market the idea of paying for additional features after the sale. This could lead to customers upgrading the functionalities provided by their car based on new preferences. An example for this might be if a user wants to integrate new smart devices into a car. For this case, a car manufacturer could provide an interface capable of communicating with this device, which the user could then order.

3.2.1 Fleet Management With Connected Cars

As mentioned in Section 3.2, connected cars are capable of collecting data and acquiring it from other devices. This data can then also be uploaded to the Internet or cloud infrastructure, where it may be accessed by other systems. When this is done by multiple connected cars that are part of a fleet, it is possible to aggregate this information. It can then be used to analyze the state of the fleet as a whole and to make decisions on this scale. Different stakeholders may have an interest in data regarding a fleet of vehicles that is their concern. For example, rental companies would be able to keep track of the current state of their cars. Another use case might be for municipalities looking at traffic patterns in order to improve road infrastructure.

In addition to data collection, managing a car fleet by influencing the state and behavior of its members is also possible. Using a connection to the Internet, it may be possible to send connected cars commands resulting in a change of behavior or available functionality. This allows stakeholders with sufficient authority to influence single cars or even their fleet as a whole. An example for a use case of this system may be remote troubleshooting of a car. Especially if a problem occurs due to a problem in its software, it may be possible for technicians to remotely access the car and reconfigure it in an attempt to fix the problem. On a larger scale, other systems could be coordinated. Mariani and Zambonelli's [MZ20] idea to use degrees of autonomy is a good example of how this can be utilized. According to them, such a system would require meta-coordination. As a possible solution for this problem, they mention the use of a centralized controller [MZ20].

This is a dynamic use case that is based on a relatively short-term decision. However, it is also possible to manage longer-term aspects of a fleet. Especially the management of available functionality could be interesting for entities, such as car manufacturers or rental companies. By employing connected cars, they enable the modification of functionality. A use case for this may arise when be a car manufacturer develops a new feature. In this case, it would be possible to enable the feature free of charge for all customers for a short period of time. This could be done in order to let users test the new functionality and try to convince them to buy it after the trial.

3.3 Challenges

When developing a digital twin for connected cars, many different aspects have to be taken into consideration. As can be seen in the above mentioned expectations regarding the role these cars will play in their environment, the domain is highly complex. In addition, a single digital twin is a part of a large distributed system composed of all communicating devices. This also has an effect on the design, since challenges associated with distributed systems also have to be anticipated.

In this section, challenges facing the digital twin will be introduced. As a starting point, aspects mentioned by Mishra and Tripathi [MT14] were considered. Based on this, expectations of the domain were evaluated and the following selection of important challenges was made.

3.3.1 Scalability

In a distributed system, communication capacity is an important issue regarding scaling [MT14]. Within the domain of connected cars, we expect scalability to be influenced mainly by two factors: The amount of connections a digital twin has to maintain at the same time and the amount of data.

As mentioned in Section 3.2, connected cars will have to be able to connect to multiple other devices. This is especially important in dense traffic environments like city centers or in congestions. In these scenarios, many cars are close to each other. If a communication with every car within a certain distance is desired, an appropriate amount of connections have to be managed by the digital twin.

Another factor impacting scalability is the amount of data handled by a connected car. This can be both data that is available locally and requires processing or information sent to and received from other participants. Due to this, fast data handling should be available for the digital twin. Otherwise, a degradation of functionality might be caused by a backlog of unprocessed information. Communicating large amounts of information might also put stress on a digital twin. In order to prevent this from affecting features, it should be possible to send or receive large data packets without having an intolerable negative influence on other connections.

3.3.2 Heterogeneity

As connected cars are able to communicate with different devices in different contexts, there is a variety of possible technologies and interfaces which might be encountered. Based on multiple factors, such as local regulation, the car manufacturer, and other configurations, we expect cars to support different interfaces and technologies. With this in mind, establishing a single interface for all use cases seems unrealistic. Therefore, a digital twin has to be able to communicate with other heterogeneous digital twins.

Concerns regarding this heterogeneity may arise depending on communication technologies. Each car may be able to communicate using different methods of communication. This has to be taken into account when deciding how different cars are supposed to communicate with each other. A different factor of heterogeneity relates to how diverse systems represent their environment. Depending on this representation, a system may present its information in a specific way or prefer to interact with its environment in a way supporting its architecture. Small changes might be caused by different versions of the same system interacting, for example, when a new version implements an additional data field. However, large disparities may also occur. An example for this may be systems with entirely different low-level concepts. In this domain, it may be useful for a logic-based connected car to communicate with an object-oriented one. A consequence of these distinctions may be that in order to work together, connected cars have to be able to solve differences in their respective environment models. In other words, there has to be a way for cars to not misunderstand each other based on abstract concepts.

The importance for a concept in this domain to manage heterogeneity is emphasized by expected future development. As the proposed architecture should still be usable when future technologies are used by developers, these have to be taken into account. This means that the architecture as a whole has to be able to support currently unknown technologies. Consequently, definitions have to be chosen in a way that makes it possible to include them later on.

3.3.3 Extensibility and Development of Future Changes

Apart from motivating a concept for handling heterogeneity, the domain of connected cars also necessitates good extensibility of a chosen architecture. There are many different factors which might influence functionality of a connected car (and with it, its digital twin) to change. There may also be situations where changing only the digital twin, but leaving the functionalities of a car unchanged, can prove advantageous. An example for this may be a change of data format of the digital twin, which would not affect the functionality provided by the car.

As a result of these expected changes and upgrades in the future, a digital twin should be extensible. If done correctly, this allows the digital twin to support new technology without a large development overhead. It also makes it possible to update the digital twin regularly, therefore, making it possible to react to changing demands and requirements quickly.

Since updates to the digital twin may be necessary regularly, the architecture should be conducive of a short development cycle. In order to achieve this, changes should not affect all parts of the software. Instead, it is ideal if large parts of the digital twin can be reused with only few changes required to add new functionality. Ideally, updates of limited scope for which all required data is already available within the digital twin (for example, adding a new way to access data) would only require adjustments in a single component.

An added advantage of such an approach could be that developers do not require extensive knowledge of the digital twin as a whole. If a feature is to be developed, they could, therefore, focus on the modules that have to be adjusted. Only a few architects may be sufficient to focus on higher-level design choices. This can lead to increases in productivity by decreasing the amount of time new developers have to spend understanding the relevant parts.

3.3.4 Concurrency and Scheduling

Concurrency is another aspect that has to be considered in the domain of connected cars. A digital twin is expected to communicate with multiple other devices at the same time. In order to be able to do this, it has to support concurrent handling of the connections required. Furthermore, handling of its state has to be done in a consistent manner. This means that a digital twin has to take into account how concurrent changes to its state and communication may affect consistency. Transactional databases may be useful to be included in the digital twin in order to prevent associated problems.

A related topic is the scheduling of functions within the digital twin. Since a digital twin can manage different aspects of its associated car, functions may have to be prioritized in some scenarios. As was remarked in Section 3.2, safety-critical messages should always be processed before other

information. Consequently, a digital twin supporting different features has to be able to prioritize them according to at least static rules. Dynamic priorities based on current traffic context and configurations is also conceivable, though not as important.

3.3.5 Robustness

The digital twin as a data source and decision-making tool for a connected car is crucial for the safety of not only passengers, but also other traffic participants in the vehicle's vicinity. Due to this, the robustness of the system has to be guaranteed. This means that errors can be detected, contained and (ideally) recovered from. A modular concept may help with this requirement, as it allows for single modules to be disabled. This way, a failed module may reduce the available functionality, but does not cause the connected car to become unresponsive. If recovery is not possible, the digital twin has to guide the car to a safe fail state, for example, by stopping on the side of a street.

An area where errors have to be expected is the communication. It can be interrupted at any time due to a multitude of reasons. Because of this, a digital twin can not rely on a persistent connection to another device. Instead, it should be assumed that this connection may be severed and this should not cause any errors for either device.

3.3.6 Real-time Capabilities

When developing a concept to be used in the domain of connected cars, the requirement for real-time functionality is very important. In order to be useful in this context, a digital twin has to be able to communicate relevant information in a timely manner. This also includes the ability to quickly establish a connection. Data required for real-time functionalities may become outdated quickly. Due to this, the digital twin should support communication without a large overhead.

However, not all communication supported has to be optimized for real-time use. Functionalities which do not require information to be transmitted as quickly may use other protocols or technologies, which may offer other benefits.

3.3.7 Dynamic Management of the Environment

A connected car is capable of providing a wide variety of functionality. These may all be relevant in different constellations and contexts. Since the digital twin is tasked with the decision-making and communication, it has to be able to manage which functionalities are relevant in a given situation. Depending on this assessment, it then has to communicate with other devices in order to share information or coordinate behavior. As there are many different possible constellations, it seems impractical to configure a digital twin for every possible situation. Instead, it should be able to dynamically decide on an optimal course of action and execute it.

When making these decisions, the digital twin also has to be able to coordinate its current status and information needs to other twins. As they too should make their decisions based on their current view of the environment, a single digital twin will have to cooperate with others. However, these other twins are not known before a car starts on a route. Instead, they are encountered as a dynamic

part of the environment with which the digital twin can interact. When communicating with another device, a common way of exchanging information, like data format and communication channel, has to be agreed upon. Only then are two devices able to exchange information.

To summarize, a digital twin has to be able to manage connections with other twins dynamically based on its current state. These other twins also try to optimize their connections according to their own state and goals. This creates a dynamic environment of digital twins with different interfaces to connect to, which have to be dynamically managed by every participant.

3.3.8 Discovery of Devices

In order for connected cars to be able to communicate with each other or other devices, they have to be able to identify potential partners. This also includes the ability to determine if a connection to a device has already been established. Otherwise, there might be errors where a digital twin connects twice to the same partner using different communication channels. Processes designed to discover and identify a digital twin may depend on the technologies and protocols employed for transmissions. Consequently, discovery may have to be defined separately for every case. Due to this complexity and the fact that the proposed concept should be extensible to new technologies, discovery of devices was considered out of scope for this project. Therefore, it is assumed that there is a way for digital twins to discover and identify one another.

Apart from devices, functionality and data also have to be discovered. A digital twin should be able to find out what types of communication is supported by other twins it is connected to. This challenge is also closely related to which interfaces and data is implemented in the first place. Difficulties arising from this aspect have been discussed in more detail in Section 3.3.2. For this problem a solution will be proposed.

3.3.9 Privacy and Security

Both privacy and security are important aspects of the domain of connected cars. As a lot of information can be collected by a car, they may present an attractive target for malicious actors. Furthermore, compromised security of a connected car can also present a serious safety risk. Due to this, measures to prevent dangerous failures should be built in and it is important to consider security before use in practice.

Security is a large and complex aspect of the digital twin. It might include topics, such as access control and identification. Due to this, security is also considered out of scope.

Similarly, privacy is another complex topic for which a comprehensive concept is necessary. If a connected car is used in practice with insufficient measures to preserve privacy, it may be possible for outside actors to acquire large amounts of data. However, it is important to strike an appropriate balance. A lack of shared information may cause some functions to become unavailable. For example, a car which is forbidden from sharing its destination could not participate in advanced traffic guidance systems. An additional factor to consider when thinking about privacy are laws. Depending on legal context, different privacy laws may have to be followed. This adds to the already

complex aspect of privacy in the context of connected cars. As presenting a concept for preserving privacy would have been very time-consuming, it too was considered out of scope for this project. The goal of this decision is to be able to better show other aspects of the system.

3.4 Goals of this Project

The goal of this thesis is to propose a concept for a digital twin in the domain of connected cars. This concept should be validated by implementing a prototype based on it.

3.4.1 Goals for the Concept

The concept proposed should be useful in the domain of connected cars. It should contain appropriate ideas on the structure of a digital twin. This also means that challenges of the domain have to be taken into consideration. While some aspects were determined to be out of scope, the concept should be advanced enough to be easily understandable and useful in practice. Ideally, it can be used as a basis for other ideas, especially ones which focus on the aspects previously ignored.

Challenges in the domain should be illustrated and ways to meet them should be included. The digital twin architecture also has the goal to be able to support different technologies and functionalities. Most importantly, the concept has the goal of uniting decision making with other aspects of digital twins. This is done in an effort to create a concept capable of being used for autonomous cars. It is also expected that this measure allows to streamline software used in a connected car, as all relevant parts will work together as a single system.

3.4.2 Goals of the Prototype

As a way to validate the concept proposed in this project, a prototype is to be implemented. Its main task is to show how the architecture of a digital twin can be implemented in practice.

Furthermore, it can be used to demonstrate how communication between digital twins should take place. In order to show how communication in case of an error is supposed to take place, the prototype should include at least a rudimentary form of error handling. Finally, the prototype should be able to communicate both with another digital twin as well as (simulated) cloud infrastructure. This way, it is possible to demonstrate the concept, its advantages and disadvantages. In addition, it also allows a practical assessment of how development of the architecture works. For example, if the concept was too hard to understand, this would become obvious during implementation.

4 Digital Twin Architecture

In this chapter, the architecture of the proposed digital twin is explained. This includes components and their inner structure based on sub-components. Furthermore, descriptions of messages exchanged by digital twins and some other notable characteristics of the concept are included.

4.1 General Concept

A diagram depicting an instance of the architecture of the proposed digital twin can be found in Figure 4.1. The general structure for it is influenced by the AAS. A digital twin should be able to support multiple interfaces at the same time. These have to be managed according to current information needs and the state of the environment. Finally, the digital twin also has to include components capable of providing the functionality used to control the car.

In order to provide various interfaces, the digital twin uses *Adapters*. An Adapter has a defined interface and is identified by a unique ID. Any functionality or data provided by Adapters is located within another component, the *Internal Digital Twin (IDT)*. As a way to make the IDT compatible

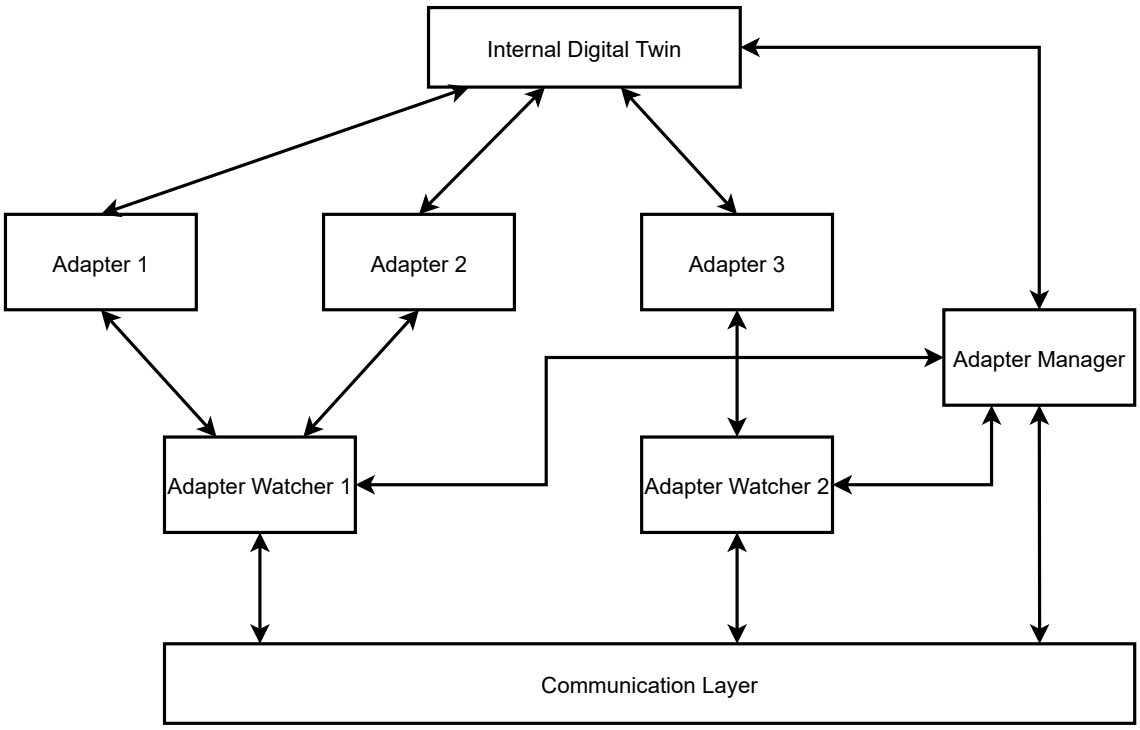

Figure 4.1: Sample instance showing all components of the architecture and their interactions

with as many devices, data formats and protocols as possible, Adapters contain code which is used to transform calls and requests. These are sent by communication partners based on a well-defined interface. An Adapter is tasked with transforming them into calls compatible with the interface of the IDT. If a response is expected, it is generated by the Adapter based on the internal response of the IDT. The result is an IDT whose communication is delegated to a layer, similar to AAS.

All communication between Adapters of digital twins is based on messages. This is done in an effort to keep the concept simple while still allowing for all methods of remote communication to be applicable. For example, RPC is based on messages [TS08]. An Adapter implementing this concept would have to transform messages into the provided calls. Then, it would use IDT calls to receive a result and return it using RPC messages.

In Order to manage Adapters of a digital twin, the *Adapter Manager (AM)* and *Adapter Watchers (AWs)* are used. Every Adapter is connected to an AW. This AW is tasked with overseeing the messages sent to and from the Adapter, aggregating data about it, and communicating with the AM and *Communication Layer (CL)* on behalf of it. AWs are used as a way to reduce the amount of unnecessary information that has to be processed by the AM. They achieve this by communicating information only if it is deemed relevant. Unnecessary details are omitted, though they can be tracked and used to inform the AM of trends.

The AM is responsible for managing the connections a digital twin engages in. This includes decisions, such as with which devices a connection should be established and which Adapters should be used. All coordination with other digital twins takes place using the messages defined for the AM. As a way to react to changes of the environment, the AM is able to communicate with the IDT. Since the AM does not contain information on the state of the environment, this connection is used to assess the need for different connections and connection partners. Vice versa, it is possible for the IDT to request data on the aspects managed by the AM, if this is needed for its operation. Apart from the IDT, the AM is also in contact with all AWs and the CL. This enables it to receive information on other aspects of the digital twin, which can be considered in its decision-making.

Communication is possible on two different levels using the proposed architecture. First, a connected car should be able to exchange messages with other devices in its vicinity. This is possible using the CL. It contains communication channels, which support different technologies used to communicate to other devices. The use of these channels is managed by the AM, which matches channels and Adapters together based on compatibility and other factors.

Second, communication with cloud infrastructure should also be possible when using the architecture. This is done by allowing for communication with a system located in a cloud as part of the IDT. Internally, it can contain *Cloud Fragments (CFs)*, which are used to model a connection to an outside system related to the connected car.

The main goal of this concept is to isolate business logic in the IDT and allow access using different interfaces. An advantage of this approach is that it is possible to support different concepts using Adapters. The IDT can be implemented using an independent concept, while Adapters include logic to transform its logical view into one compatible with their respective interfaces. In addition, different communication channels and CFs allow for different modes of communication. Both of these measures are aimed at improving the longevity of both the architecture as well as digital twins using it. Furthermore, it is possible to develop Adapters and communication channels independently

of one another. This facilitates separate development of different interfaces and technologies in short cycles. In the context of this domain, car manufacturers could use this to their advantage by being the first to bring new features on the market or react to a change in demand.

4.2 Components

Figure 4.1 shows how an instance of the proposed architecture could look like. It also includes which components communicate with each other. In the displayed instance, three Adapters and two AWs are used. Additionally, the IDT, AM and CL can be seen. Using an instance to illustrate the concept has the advantage of showing how components interact. A more general diagram would not be as useful, since the connections of Adapters and AWs would be harder to depict.

4.2.1 Internal Digital Twin

The *Internal Digital Twin (IDT)* is the part of the digital twin containing data about the car and its environment. Any logic required to obtain or process and store this data (like sensor handlers) is part of the IDT. If decision-making is available for the car (for example, in autonomous cars), any logic used for it is encapsulated within this component. Furthermore, software used to control the car itself, for example, cruise control or high beam control, is also part of the IDT.

The only way for it to communicate with other devices are the implemented Adapters. As a single IDT has to be able to fulfill requests for multiple Adapters, a well thought-out interface for these interactions is important. Furthermore, as Adapters themselves are unable to communicate with each other, sometimes information about Adapters in use has to be stored in the IDT. This can be done by an Adapter changing the state of it. If another Adapter's behavior might be affected by the first one, it can check the IDT for this change of state and conclude if the first Adapter is connected. Due to this, the design of the IDT has to be verified thoroughly in order to rule out problems, which might be caused by simultaneous operation of some Adapters. Alternatively, the AM could be configured to never connect problematic Adapters at the same time.

Apart from Adapters, the IDT is also connected to the AM. This connection is used to inform the AM about the current traffic situation. Based on this information, different Adapters may be chosen or communication technologies may be more appropriate. A possible example for such an interaction might be, if the IDT informs the AM that the car just exited a city, causing an Adapter used to share the location of pedestrians to be turned off.

As the IDT contains all handlers of hardware, it controls the CL indirectly. It is able to turn Hardware like transceivers on or off. This can change the availability of different communication channels. However, during normal operation, the IDT should not affect the CL at all, except to ensure smooth operation. The IDT should only command the CL if the state of the car as a whole necessitates actions. An example for such a situation might be if some hardware is in the process of overheating and, therefore, has to be turned off. Apart from these special cases, the IDT is unable to communicate with any AW or the CL.

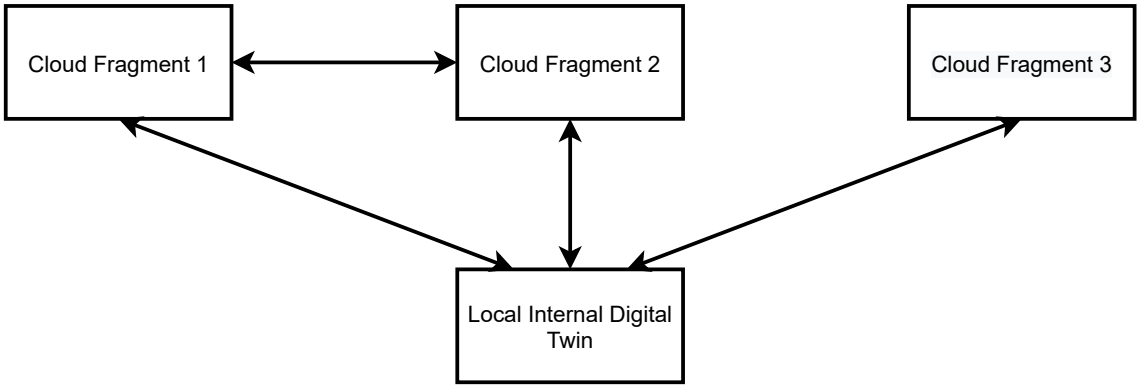

Figure 4.2: Sample instance of an Internal Digital Twin

Internal Structure

The internal structure of the IDT is mainly a black box. This is done in order to allow for a variety of different approaches to be chosen. Inside the IDT, any technology or architecture can be used by the developer. As long as the interface is able to satisfy the requirements of all Adapters, they will be able to mask the internal structure.

However, as approaches which use cloud infrastructure to process information should be possible using this architecture, a simple partition of the IDT is defined. According to this definition, the IDT contains one *Local Internal Digital Twin (LIDT)* and any number of *Cloud Fragments (CFs)*. How many CFs are included is chosen by the developer based on multiple factors, for example, required functionality or cost of maintaining a cloud service.

The LIDT is the software which is part of the IDT and runs on hardware located within the connected car. At least functionality like sensor handlers or other hardware management has to be included in it. When a CF is used, a module able to communicate with it must be incorporated. If no cloud infrastructure is used, the LIDT contains all data and behavior of the connected car.

Unlike the LIDT, CFs are not required. A CF is a part of the digital twin that is executed in a cloud environment. It includes any component running on the cloud that stores or processes information regarding the connected car. In practice, these components will most likely handle data for multiple connected cars. However, from the point of view of a digital twin, this will be abstracted and instead assumed that the components are used only by itself. This is done in an effort to make the concept simple yet compatible with different technologies and approaches. Developers implementing components other than the CFs will not have to account for this use by multiple cars. Only those working on the CF itself will have to find ways to realize it efficiently. Communication between different CFs is allowed. This can be useful in case two cloud solutions require the same data. For example, if a CF was previously inactive and now requires a large amount of data, a different CF may be able to provide it. Therefore, the LIDT would not have to retransmit information, which reduces strain on communication networks used by it.

This internal structure and different communication options can be seen in Figure 4.2. It illustrates how an instance of the internal digital twin can be structured. In this example, three different CFs are used. Two of them are also connected with each other, meaning they can communicate. However, such connections are not necessary. This becomes evident due to *Cloud Fragment 3*, which does not have connections to any other fragment.

4.2.2 Adapters

An *Adapter* is a component that enables different devices to communicate using a specific interface. This interface is unambiguously defined and characterizes how devices can interact with each other. Different kinds of Adapters can be used by the digital twin. They can be capable of connecting not only to nearby connected cars, but also other types of devices. For example, an interface may be defined for communication with a smart traffic light. If the digital twin of a connected car implements an Adapter based on this interface, it can be used to cooperate with the traffic light.

An Adapter always offers exactly one interface and transforms calls to it into instructions for the IDT. This means that an Adapter bridges the black-box implementation of the IDT and a known interface, thereby enabling cooperation with other participants.

The interface provided by an Adapter is encoded using an ID. This ID is unique, meaning that any two devices that have an Adapter implementing the same interface can communicate with each other. They can do so by using the Adapter, regardless of internally used concepts or technologies, which are encapsulated in the internal digital twin.

One of the core ideas of the proposed concept is that a digital twin can provide multiple Adapters. These can be active at the same time, provided the IDT is capable of resolving possible conflicts caused by this. As a consequence, the connected car will be able to exchange information and coordinate with different devices in different ways. Furthermore, it is possible to communicate with the same device using multiple Adapters in order to use different interfaces at the same time. An example for such a use case may be that two cars use one Adapter for coordinating their relative speed, while simultaneously sharing information on traffic conditions using another.

4.2.3 Adapter Watcher

The function of an *Adapter Watcher (AW)* is to oversee the operation of one or more Adapters. Additionally, it is tasked with communicating with associated Adapters and managing messages to and from them. It can store information about connections and the Adapters, but nothing else.

While an Adapter is active, its AW collects information on the ongoing connection. This is done by aggregating information the Adapter sends to its AW directly. In addition, an AW analyzes messages addressed to and sent from its associated Adapters. When a message addressed to a connected Adapter is received by the CL, it is passed to the AW. Before the message is transferred to the Adapter, the AW can evaluate it and its contents. Furthermore, it is possible for the AW to modify them. For example, if an Adapter expects messages to only contain application level content, but the AW receives a message containing other layers, it can remove them in order to match the format expected by the Adapter.

Some of the possible modifications may be based on commands from the AM. An example for such a case may be if the AW is commanded to delay messages to its Adapters in order to reduce CPU load. In order to make such orders possible, the AW is connected to the AM.

When an Adapter wants to send a message, the process also involves the AW. It receives a message from the Adapter with additional information specifying the type of connection it should be sent over. Based on those requirements, the AW chooses an appropriate communication endpoint and passes the message to it. In order to do this, the AW is capable of storing information about what connections may be used by which Adapter. Additionally, outgoing messages can be analyzed. Similar to incoming messages, this data may be used to draw conclusions on the state of the Adapter and its ongoing connections.

Another way for the AW to acquire information is by the Adapter notifying it. Adapters can send information about their status or connection to their associated AW at any time. However, it is important to note that Adapters cannot store their own data. This means that such notifications can only be caused by the state of the IDT or events. Consequently, it is advisable for the AW to keep track of trends, as the Adapter is not capable of this.

If the AM requires information about an Adapter or its connections, it can request it from the AW. In addition, the AW can send the AM important information as soon as it becomes available. An example for such a case is when errors are noticed. These can be passed on to the AM immediately. Alternatively, the error can be noted but not yet reacted to. This may be done for errors with a low priority, for which there might be a rule that they have to appear multiple times before the AW passes their occurrence on to the AM.

Internal Structure

The way in which the AW is organized internally can be seen in Figure 4.3. In addition to the internal structure, connections from AW subcomponents to other components are included. Relevant higher-level components can be recognized in the diagram based on their gray color.

One of the most important components of the AW is the *Information Aggregator (IA)*. In this component, information on connections and their state as well as the state of the Adapter is stored. Any interaction between AW and AM takes place over this component, making it critical for operation of an AW. It can receive data from all other components, apart from message queues, of the AW. This makes it possible to acquire information which should be provided by AWs. In addition to collecting data, the IA communicates with the *Message Distributor (MD)* and the *Adapter Commander (AdC)*. This way, commands can be given to influence incoming or outgoing messages of associated Adapters. It is also possible for the MD and AdC to request information from the IA, which is required when checking which connections are valid for which Adapter.

For every Adapter which is handled by an AW, one incoming and one outgoing Message Queue should be provided. This makes it possible to prioritize Adapters by prioritizing processing of their respective queues. Components used to manage incoming and outgoing messages are the MD and the AdC. The MD handles outgoing messages and passes them on to appropriate communication endpoints. Similarly, the AdC is used to process incoming messages before handing them over to

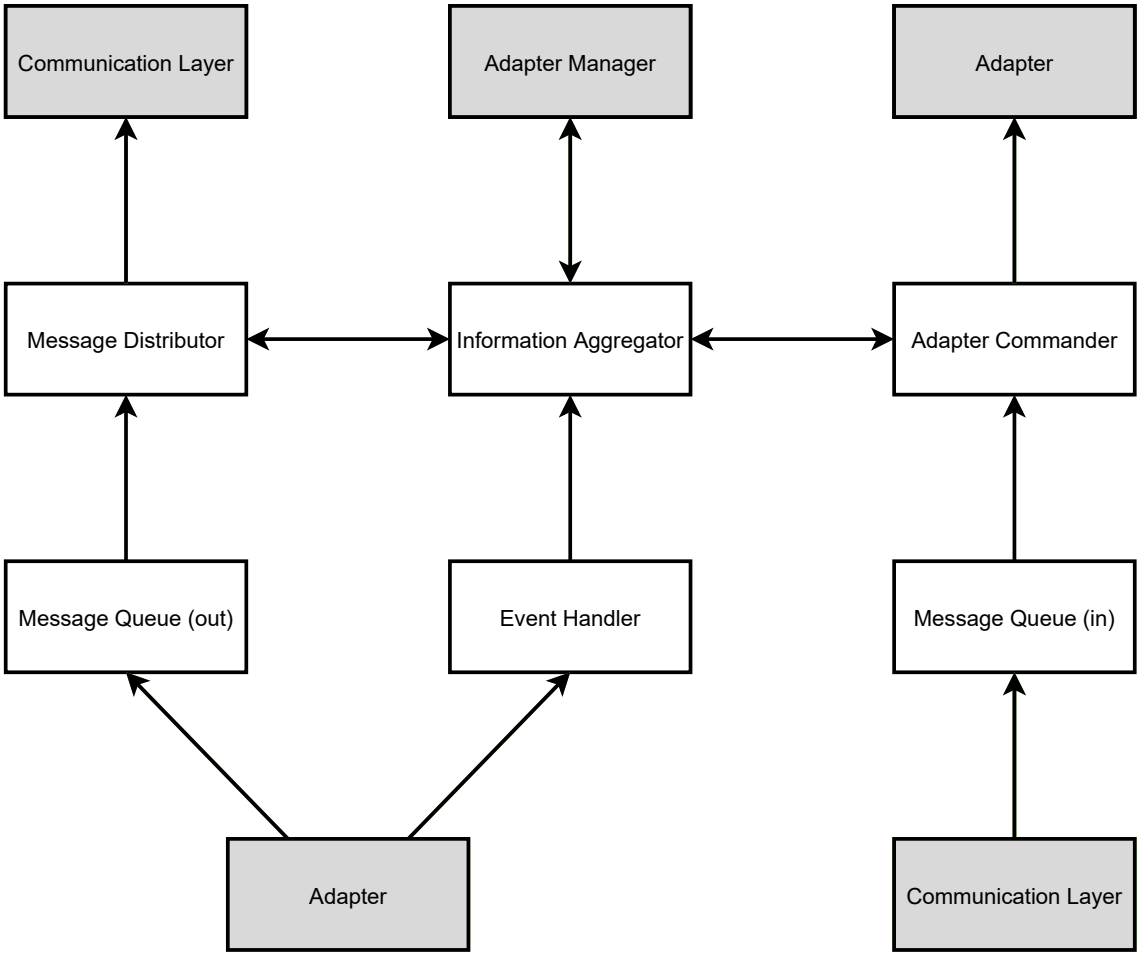

Figure 4.3: Internal structure of the Adapter Watcher

their specified Adapter. While both of these components may modify parts of the messages, this may not be needed. However, this does not make them obsolete. While in these cases they will not alter messages, they still have to collect information on messages to pass on the the IA.

Furthermore, the AdC is used to pass commands onto Adapters. As Adapters do not contain state and should only be based on the state of the IDT, these controls are very limited. An example of a use case for them is turning the Adapter on and off.

The final component in this diagram is the *Event Handler (EH)*. It is in contact with the Adapter and is informed about important events, for example, errors occurring. The EH collects data on these events and can notify the IA if conditions set by the developer are met. This is done with the aim of reducing strain on the IA, which may be caused by too many unimportant event notifications.

4.2.4 Communication Layer

The *CL* is the component of the digital twin that makes it possible to exchange messages with other devices. Any software component used to transmit or receive information is part of it. Internally, the CL is a collection of various communication channels. Every one of those channels is capable of sending or receiving messages using a specific protocol and technology. For example, there may be a communication channel capable of exchanging information via Bluetooth. In order to identify different channels, a unique character sequence containing no digits or special characters is used. When digital twins exchange information about communication channels, these identifiers are used to make intentions and data unambiguous.

A communication channel can be based on connections or datagrams. Both of those approaches are possible, though handled slightly different when sending messages. This will be described in more detail in Section 4.4.2. Apart from this differentiation, any type of technology or protocol can be used by a communication channel. As a consequence, the concept can include even future technologies without having to define new special cases.

When receiving a message, the CL is tasked with forwarding it to a specified destination. If no connection for the recipient exists, the AM is informed. In order to decide which message should be passed to which component, the CL is informed about existing connections by the AM. This includes information on which communication channels have been assigned to which Adapters and partners. A scenario illustrating this idea can occur when a digital twin communicates with a partner using multiple Adapters. Depending on existing connections, a communication channel may pass on messages sent by the partner and addressed to a specific Adapter. However, those specifying a different destination may not be forwarded if the CL was not informed that the channel can also be used for this other Adapter.

This makes it possible for the AM to manage connections in a detailed manner. The advantage of this is the ability to very precisely react to priorities for different Adapter and communication technologies. Furthermore, it is possible to control the utilization of different channels. For example, the AM may choose to distribute connections over multiple different channels in order to reduce strain on a single channel. In addition to this management, the CL can react to other commands given by the AM. An example for such a command is to ignore messages from a specific digital twin. It is also possible for the AM to turn communication channels on or off.

When managing connections, the AM may need information about the state of the CL and its various channels. This information can be acquired by requesting it or by the CL proactively notifying the AM of changes to its state. A case in which the state of the CL might change is if the availability of hardware components changes due to input by the IDT. However, this case should be unlikely. During normal operation, the IDT should only interact with the CL indirectly. This is the case as the AM assesses the current situation based on data of the IDT. If this indicates a need to change the current configuration of the CL, the AM will take appropriate action using its authority.

4.2.5 Adapter Manager

The component controlling Adapters and connections of the digital twin is the *Adapter Manager (AM)*. Its main task is the communication with AMs of other digital twins. Together with the other AMs it decides which Adapters and communication channels to use. In order to do this, the AM is

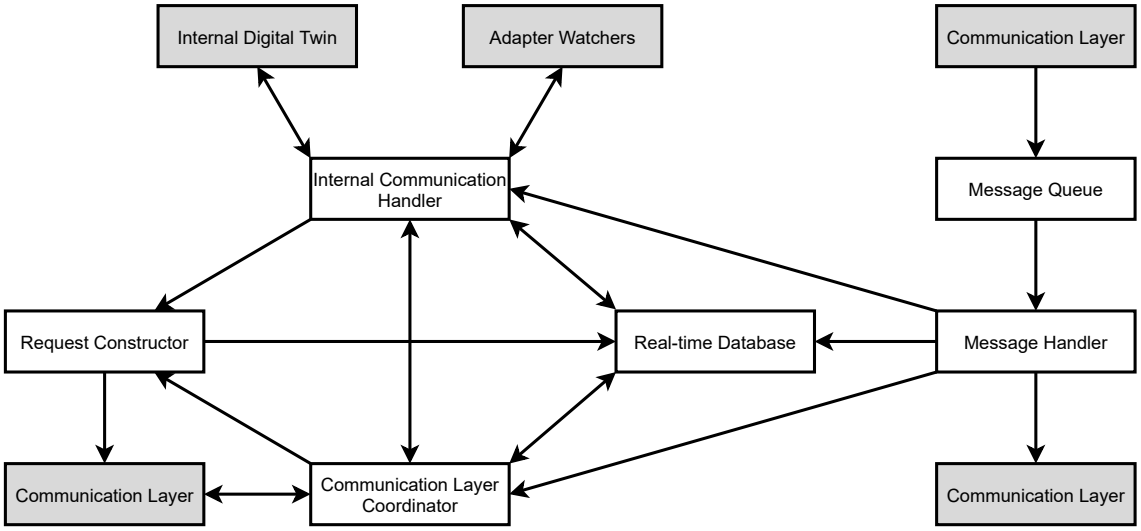

Figure 4.4: Internal structure of the Adapter Manager

aware of the available Adapters. It is able to prioritize them based on static knowledge, such as their optimal use case as well as the current state of the digital twin, for example, the number of ongoing connections using a specific Adapter.

During operation, the AM keeps track of connections and can react to events that might arise. For example, if it is alerted to a high number of errors in an Adapter by the corresponding Adapter Watcher (AW), it can request the connection partner to switch to a different Adapter. In order to be able to command other parts of the digital twin, the AM is connected to the CL and all AWs. Adapters can be controlled indirectly via their associated AW.

Furthermore, the AM is connected to the IDT. This connection is bidirectional, meaning the AM can request information from the IDT and vice versa. The AM uses this connection to gain information on the state of the environment and the car. It uses this knowledge to determine the optimal configuration regarding the connections of the digital twin. This includes both the Adapters and communication channels used as well as with which digital twins connections should be established. In addition, the IDT is able to request information from the AM. This may be used during operation of the connected car. For example, the amount of ongoing connections may be a relevant data point when trying to assess traffic density.

As the AM has to manage large parts of the digital twin, it has to be able to store information. However, all knowledge of the connected car and its environment not related to connections and communication is stored in the IDT. This separation should also be kept regarding behavior. While the AM is able to make decisions concerning the connections used by the digital twin, all decisions regarding the behavior of the physical car are made by the IDT.

Internal Structure

Similarly to the AW, the AM is also subdivided into an internal structure. The organization can be seen in Figure 4.4. It includes the subcomponents of the AM and connections between them. If a component of the high-level architecture communicates with the AM, it is displayed in gray and its connections to subcomponents are shown.

An integral part of the AM is a database. As state has to be stored and is accessed by multiple subcomponents, a robust and highly available database should be chosen for this task. Furthermore, due to the concurrent nature of the AM caused by communicating with multiple communication partners at the same time, transactions can be used to ensure a consistent state. Otherwise, phenomena like lost updates might cause problems at runtime.

Exchange of information with the IDT and AWs takes place using the *Internal Communication Handler (ICH)*. It contains the interface for these components to request information, report changes of state, and declare their priorities. In order to provide this interface, the ICH is able to communicate with the *Request Constructor (RqC)* and the *Communication Layer Coordinator (CLC)*. Additionally, the ICH is tasked with informing the IDT and AWs of changes that are relevant to them. For example, if an Adapter is disconnected, the ICH will inform the associated AW.

If the ICH determines that a message should be sent to a communication partner, the RqC will be called. This component is used to generate messages for interactions that are initiated by the digital twin. When a message is sent, it is stored in the database. Upon receiving a response, the *Message Handler (MH)* will then be able to look up the initial message and react based on this knowledge.

Adding to the functionality of the RqC, the MH is used to handle incoming messages. This can be both new messages from other devices or responses to previously sent ones of the digital twin. In order to be able to react to communication, the message handler is connected to the ICH and the CLC. Therefore, handling a message can also include calls to other components via these subcomponents and sending responses. As the concept expects AMs to communicate based on messages, a message queue is also included in its subdivision. This queue is used as input by the MH. When the MH or RqC need to send a message, they pass it to an appropriate communication channel in the CL, from where it is sent to the specified partner.

Apart from using the CL to send messages, the AM is tasked with its management. This function is fulfilled by the CLC. It is able to send commands to the CL and receive status updates by it. The knowledge gained from these updates, together with the information stored in the database, and communication with other subcomponents, is used by the CLC to manage the CL. This includes decisions, such as which communication channels should be used or which ones can be temporarily turned off. In some cases, communication channels may be configurable regarding the quality of service they provide. For these channels, the CLC has to determine the optimal configurations based on the current situation.

4.3 Messages

The interaction of digital twins is based on messages. This allows use of a variety of technologies and approaches for Adapters. AMs also communicate based on messages. Every digital twin receives messages using its CL, which distributes them to the appropriate component.

4.3.1 Message Format

When a digital twin receives a message, it has to determine both the sender and the addressed component. This is necessary in order to check if the use of the communication channel is allowed or if an error message should be sent in response.

The first 128 bits of a message are an ID used to identify the sender. Every device has a unique ID enabling the digital twin to quickly recognize how a message should be handled based on sender. After the sender ID, another 64 bits are used to identify the target Adapter. For every Adapter, a unique ID is defined. This makes it possible to pass messages on to the interface they are designated for. If the target is the AM, the Adapter ID is 0. This also means that no Adapter should be assigned the ID 0, as it would not be capable of receiving messages.

After these two IDs, the actual message content begins. This can include any number of bits in any format, though this may depend on the communication channel. The goal of this approach is to give extensive freedom to developers regarding the structure and content of messages they use. Consequently, many different data formats can be used for interfaces implemented by Adapters.

4.3.2 Adapter Manager Messages

Messages that can be processed by the AM are called *Adapter Manager Message (AMM)*. They are encoded in JSON using UTF-8. The reason for this is that a human-readable message format makes logging and bugfixing easier. Especially in the domain of connected cars, this is very valuable, as identifiers can be understood across developer teams. Two widely used human-readable formats are XML and JSON. Of those two, XML offers more functionality but at the cost of performance. As the AMMs are only used to transfer name-value pairs, we chose JSON as the format to use.

Every message contains the ID of the sender, a sequence count, and a string denoting the type of the message. While the sender ID is also included in the full message, it is also added in the payload of it. This is done so that libraries for using JSON can also easily access the sender ID. Since some libraries may have problems with displaying an integer with a large number of bits, the ID is displayed as a string of numbers. This string encodes the number in decimal and does not include leading zeros. The goal of this design is to reduce the number of characters needed to display it.

In order to check which messages should be processed, a sequence count is used. Since no guarantees regarding the order of arrival of messages has to be made for AMMs, the AM has to account for them arriving out of order. In these cases, the sequence number in conjunction with the type and content of a message can be used to find out if it should still be processed. For example, if a message arrived with a request contradicting a message with a higher sequence count, the earlier message would be ignored.

The final field included in every AMM is the type of message. It is used by the AM to determine what the format of the rest of the message is. Depending on the type, different fields are included containing the parameters necessary. Furthermore, depending on the state of the connection, an AM may be able to immediately recognize some invalid requests based on their type.

In many types of messages, a priority is included as another field. This is done in order for digital twins to convey how urgent a request is. While functionality should be provided whenever feasible, distinct priorities can be used to assess which requests can be declined without disrupting the system

Priority Level	Example Use Cases
1	technical crashes make a change necessary, regulation requires a change, action is necessary for digital twins to stay connected
2	provides large functionality improvement, is required for Adapters to stay connected, change necessary due to context change (for example going from highway to city traffic)
3	minor technical issues (for example range issues in communication channel)
4	provides small functionality improvement, causes improvement to passenger comfort
5	causes slight improvement in technical efficiency (for example data transmitted with marginally lower overhead)

Table 4.1: Different priorities and examples for cases in which they should be used

significantly. There are five different priorities. A lower number indicates a higher priority of the request in a message. Some examples for situations in which certain priorities should be included when communicating are listed in Table 4.1

A different field included in many messages is used to convey additional information. It is used due to the possibility that different communication channels or Adapters may need more information. For example, when an Adapter using a TCP channel is connected, the port number of the channel could be included in this field. In order to match this data and the component it is relevant to, an internal structure has been defined for additional data. Information relevant for the AM is included in a "general" field. Data regarding communication channels or Adapter is stored in an array called "specific". It contains objects which encompass a string "ref". As Adapters use IDs which can be represented as numbers using digits and the names of communication channels do not contain digits, no further clarification is needed and every viable content of "ref" can be matched.

When two digital twins communicate, a very important part of their coordination takes place using AMMs. This is due to the fact that their respective AMs coordinate on which Adapters and connections to use. During this process, some information may also be exchanged about the capabilities of the digital twins. Most requests can be both accepted and declined by the recipient. An example for a not rejectable one is a disconnect request. However, if possible, digital twins should aim to provide as much functionality as possible to their environment, as this may result in a better overall system. The following functions have been devised to be used by AMs when communicating. A thorough list of messages used to realize the needed communication can be found in Appendix A.1 on page 63.

1. Quick connect: When two digital twins want to establish a connection, they can do so using this functionality. When it is used, a connection with an Adapter is established immediately. Due to this, it is recommended to use quick connect, as domain related data can be exchanged without delay.
2. Temporary connect: In contrast to quick connect, a temporary connect can only be used for a short amount of time. After this, it automatically disconnects. The goal of this functionality is to allow AMs to exchange information about the properties of their digital twins. Due to

this, only a limited number of request types are processed during this connection state. If a connection using an Adapter is established during the temporary connection, it becomes a normal one and all other functionality becomes available.

3. Disconnect: Used to disconnect an Adapter connection. If the connection as a whole should be terminated, this message with an Adapter ID of 0 can be used.
4. Add additional connection: This is used to establish another connection using a different Adapter. This way, multiple connections between two digital twins are possible at the same time.
5. Swap Adapter: This function can be used to disconnect one Adapter while simultaneously establishing a connection using another one.
6. Add additional channel to Adapter: If an Adapter is capable of using different communication channels, this function can be used to add a communication channel over which it can send or receive messages. This may be used for redundancy, but can also enable the Adapter to send messages using different guarantees provided by the channels.
7. Remove channel from Adapter: Can be used to remove a communication channel connection from an Adapter. It is important to note that this may reduce the efficiency and functionality of the Adapter. This is the case if requirements regarding message delivery can no longer be met. In some cases, this may even cause the Adapter to become inactive.
8. Swap communication channel: Similar to swapping Adapters, but used to swap communication channels used by an Adapter.
9. Manage QOS message: Some communication channels may offer different guarantees for delivery, for example, regarding delivery time. If coordination is necessary to configure these aspects, this functionality can be used.
10. Error (response) message: AMs have to be able to react to errors. This is done using error messages. If the processing of an error message requires a response, this is possible using an error response message. A more detailed explanation regarding error handling can be found in the section “Error Handling and Types” below.
11. Update ID: It is assumed that two digital twins never have the same ID. However, should this not always be the case in practice, this function can be used to update it. There are also other cases in which this function may be useful. For example, it is possible to update IDs regularly during trips in order to avoid being tracked based on a constant ID.
12. Check connection: If little data is transmitted, an AM may want to check if communication with a partner has been interrupted. In these cases, this functionality can be used.
13. Get version: Since AMs are intended to be expanded in the future, it is important for them to be able to request each others version.
14. Swap manager version: If two AMs implement the same versions, they are able to swap between them. This can be used to switch to a version providing more functionality, but may also be utilized as a way to swap to a less powerful and less computation-intensive one.
15. List supported Adapters: An AM can be requested to list the Adapters implemented by the digital twin.
16. List supported communication channels: An AM can be requested to list the communication channels available. These can be used in order to exchange messages with the digital twin.
17. Send information on a communication channel: Some communication channels may require additional data in order to be used. For example, a technology may use a specific frequency in a range, which has to be agreed upon. This functionality makes it possible to exchange this kind of information.

Error Handling and Types

Errors are handled using error messages sent between affected AMs. This type of message contains an error info object and may include a log message. The error info object is used to show what kind of error occurred and can be used by the communication partners to convey information. Based on this data, digital twins can try to fix problems as they appear.

In contrast to the error info object, the log message is not meant to be analyzed by software or used to react to problems. Instead, its purpose is to convey in natural language what error occurred and why. When a developer reads this log message, it may be possible to find the cause of the problem and fix it. This leads to a better overall system in which fewer errors occur. However, as natural language may be ambiguous and it is hard to define rules that make it possible to describe every possible failure, this part of an error message should not be used by the digital twin.

In order to allow an AM to efficiently process errors, types have been defined which make it possible to handle them due to their fields being known. The types of errors which can be used by the digital twin are as follows (a specification of their contents is available in Appendix A.2 on page 68):

1. Unequal connection state: If two AMs disagree about their connection state, this error can be used. When it occurs, the connection is considered to be terminated and has to be reestablished in order to be used.
2. Invalid request: If a request cannot be executed due to the state of the digital twin, this error will be sent. For example, a request to disconnect an Adapter that is not connected will result in this error. This only applies to AM requests, Adapters have to check the validity of requests addressed to them on their own.
3. Invalid message format: If a message is unreadable, the AM will send an error of this type in response.
4. Invalid message content: If a message to the AM can be read but does not contain the expected information, this error will be returned. Examples for causes of this error are wrong or missing JSON fields or an unknown type of message.
5. Invalid communication channel: If a message is received by a communication channel which is not currently assigned to the target of the message, this error is sent as a reply.
6. No such Adapter: If a message is addressed to an Adapter that is not implemented by a digital twin, this error is sent in response. It indicates that the target Adapter is not available and messages to this Adapter will have no effect.
7. Unnecessary reply: This error type can be used, if a reply is received for which no request was sent.
8. Adapter crashed: If an Adapter or required systems in the IDT crashed, this error is sent to all communication partners. In addition, the error can contain additional information based on the Adapter that crashed.
9. Communication channel crashed: This error is similar to the Adapter crashed type. Similarly, it too can contain additional information based on the problematic channel.
10. Request ignored: As described in Section 4.3.2, AMMs contain a sequence count in order to determine if they should be processed. If based on this number an AM decides not to process a message, this error is sent in response.

11. Same ID detected: If two digital twins use the same ID, this error type can be used to draw attention to this problem. While it is unlikely that this happens, we felt it is important to include a way to resolve the issue, as the huge number of cars on the road increases the chance.
12. Other: This error type signals to an AM that the sender has encountered an error that does not fit another category. The receiver can store the included log message for later analysis by a developer and continues operating normally. If an error that cannot be classified and is severe enough to warrant connection termination arises, an unequal connection state error should be sent instead. In this case, the log message should also be used to communicate to the receiver what sort of problem caused the connection to fail.

4.4 Other Notable Characteristics

In this section, some other notable characteristics of the proposed concept are presented. One of them is, how cloud infrastructure can be included in the digital twin. Furthermore, a description of message handling and requirements regarding delivery of messages is included.

4.4.1 Coordination Using Cloud Infrastructure

A connected car may be able to communicate with other traffic participants using a cloud connection. For example, cloud infrastructure could be used to determine an efficient speed for vehicles on a road. These types of connection are possible regardless of the distance of the cars themselves, as long as they are able to connect to the cloud.

If they are close to each other, it is still recommended to connect their respective AMs. This leads to them being aware that a connection has been established. An advantage of this is that additional Adapters can be added easily and information can be exchanged using AMMs. Furthermore, it becomes possible to switch Adapters immediately should the connection to the cloud be disrupted. As a consequence, the delay until the digital twins can communicate again is significantly lowered due to their AMs already being connected.

In order for cloud solutions to be used, a special type of Adapter should be provided. An Adapter of this type is purely used to indicate a connection via a cloud and should not provide any actual interface apart from its Adapter ID. Instead, it is used to signal to AMs that a connection to a partner is established using cloud infrastrucutre. For every cloud implementation, a specific Adapter is provided. Its ID is used to identify different cloud solutions unambiguously. When two AMs agree on using this Adapter, their respective digital twins do not exchange messages using the Adapter itself. In this case, the IDTs of the twins are informed that they can communicate with the other participant using the cloud. As no messages are exchanged between cloud pseudo Adapters, no AW is needed to observe and control it. These functions are fulfilled by the IDT.

4.4.2 Quality of Service Handling and Connections

The digital twin is able to support multiple communication channels, which can provide different levels of Quality of Service (QoS). Furthermore, a channel can be connection or datagram based.

All of these factors have to be taken into account when assigning ways to communicate to Adapters. This also includes the requirements Adapters have regarding their communication. Examples of such specifications may be that some Adapters only work with a specific channel or require a short delivery time for messages. Due to this, the AM has to be able to determine an optimal strategy to match Adapters and communication channels. Information on connections, in case of connection based channels, or available channels is passed to AWs. There, this information is stored and later used to pass messages from the Adapter to appropriate communication endpoints. Additionally, AWs inform Adapters capable of using different channels of available communication options. The way in which this is done depends on the Adapter and its implementation. For example, it is possible for the Adapter to store all available connections. Alternatively, an Adapter could also be informed of available QoS which can be fulfilled, but not how this is done. During operation, the Adapter then decides on a channel or QoS for messages and passes both pieces of information to its AW.

Should the required QoS not be available, an Adapter notifies its AW of this situation. The AM is then informed by the AW and should resolve the problem. This can be done multiple ways, for example, by adding another communication channel, changing the QoS configuration of the current one, or swapping the Adapter. During this process, the Adapter behaves in a defined way. This depends on the Adapter and what kind of requirements are violated. Examples for behavior are using an inferior communication channel, reducing available functionality, or becoming inactive.

4.5 Definition of Components

As the proposed architecture and concept are made to be extensible, some definitions are necessary to include new components. An AM is defined using a version identifier. It contains three numbers in the format "1.2.3". The first number indicates the general interface of the AM, the second one small changes to the basic interface (for example minor developer-specific changes), and the third one is used to differentiate versions of these changes. When developing a digital twin, it is important for the AM to implement not only the interface of its version, but also older ones. In order to guarantee interoperability with older versions, every AM implementing versions “*.*.x” has to also implement version “*.*.y”, where y<x and the first two numbers are equal. The same goes for larger version changes, meaning if “x.*.*” is implemented, “y.0.0” for y<x also has to be included. This version identifier is used to define all messages an AM can handle and possible ways it may react to them. Included in this specification are all error messages and their responses, when they are used, and what behavior is expected from digital twins upon sending or receiving them.

Another important type of component to specify are Adapters. They are identified by an ID of 64 bits. In addition, their definition contains what kind of messages they can receive and how they behave. Furthermore, an Adapter specification has to establish what kind of communication channels they can use and what QoS they require. This also includes how behavior is affected if their requirements regarding communication are not met.

In addition to Adapters, communication channels are used to enable developers to implement different technologies in the digital twin. The identification of communication channels is possible due to unique strings. From an operational perspective, they have to define technical details of how messages are transmitted as well as hardware required. In order for the AM to make decisions regarding QoS aspects and for Adapters to consider which channels are appropriate, information regarding the QoS available for this channel has to be included.

5 Prototype

In this chapter, we describe the prototype. This includes its structure and our experiences gained during development and testing of the prototype.

5.1 Implementations and Features

This section contains a description of the structure and capabilities of the prototype. An important aspect of this is the inclusion of a CF, whose integration will be covered in detail.

5.1.1 Technologies Used

A goal for the prototype was to show how it is possible to communicate with a cloud environment. In order to achieve this, the MBP was used to simulate a cloud environment. The MBP is a platform introduced by Franco da Silva et al. [FHS+20]. The reason for choosing this platform is its ability to display complex JSON objects. As the AMMs are formatted in JSON, this makes it simpler to include MBP and reduces the overall complexity of the digital twin.

The digital twin itself was implemented using the programming language Python. It was chosen due to a native JSON support and ease of integration with MBP. Due to both the AM and MBP using JSON, Python's support of the format was very useful during development. The fact that it was available without having to install an additional library made it a very attractive choice. Another advantage of using Python for the digital twin is that MBP uses scripts written in Python to extract data. Choosing the same language for the rest of the digital twin further reduces complexity and ensures good integration of MBP.

A VM running Ubuntu was chosen as the environment in which the digital twin instances were run. In order to show how digital twins interact, multiple instances of the same code were executed and communicated with each other. It is possible to configure the prototype instances to react differently to certain events. This makes it possible to show interactions between digital twins with varying behavior. The configuration and other commands are controlled using a command line interface. As a way to illustrate how CFs work, the MBP instance runs on a different system than the rest of the digital twin. In our implementation, a Windows computer was used to install and execute MBP.

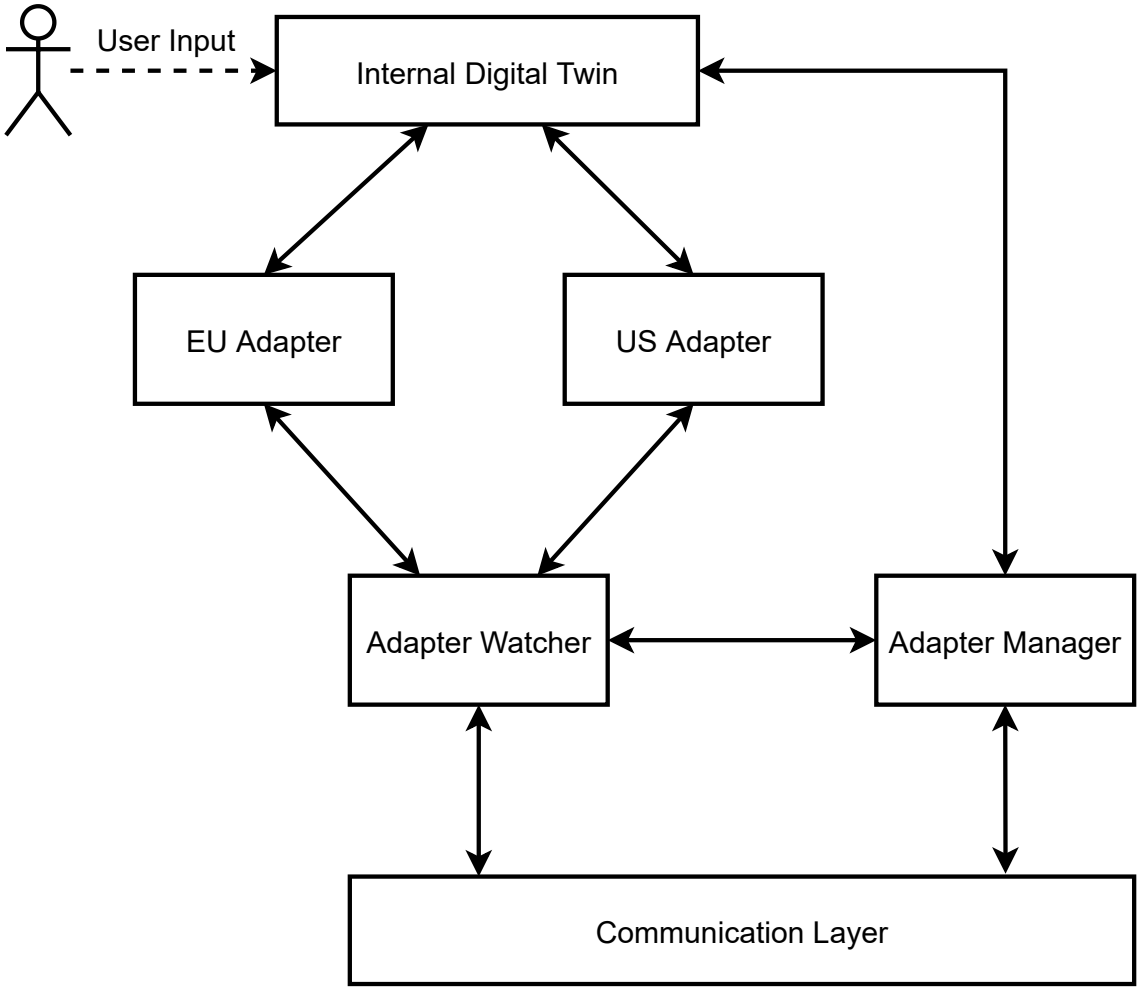

Figure 5.1: Architecture of the prototype

5.1.2 Architecture and Functionality of the Prototype

The architecture of the prototype is outlined in Figure 5.1. It can be seen that two different Adapters are included. Both are assigned to the same AW and communicate with the IDT. As no test environment capable of generating data was available, the IDT generates data by randomly choosing values for set variables. The prototype is controlled by user input, which facilitates demonstrations of the proposed concept. MBP components used for demonstrating cloud support are part of the IDT. This will also be discussed in more detail in Section 5.1.3.

Furthermore, the structure of the communication layer has to be mentioned. It contains two different communication channels. One of them is able to send messages using TCP, the other one utilizes UDP for this purpose. This makes it possible to demonstrate how different QoS aspects are handled. If a message should be sent using at-least-once delivery, the TCP channel has to be used. Otherwise, UDP may be advantageous due to a lower overhead.

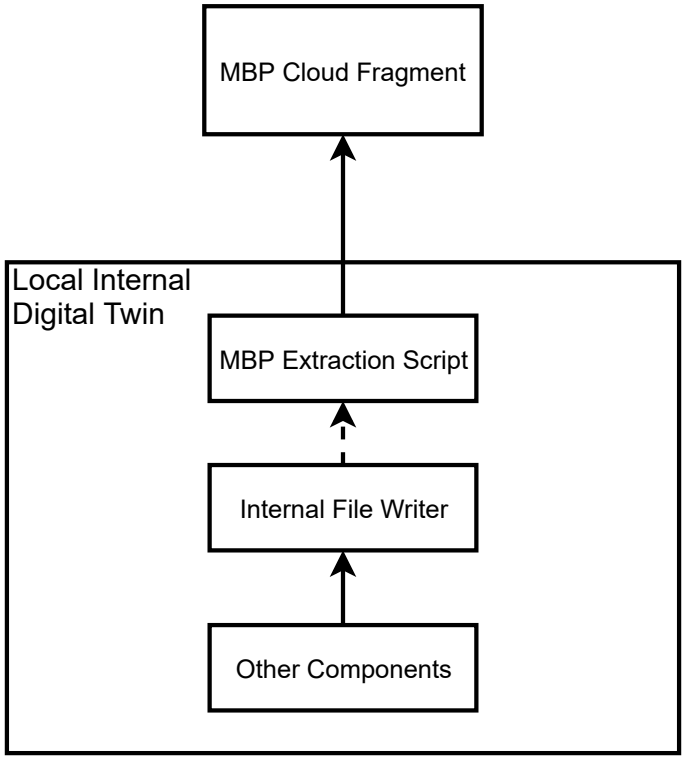

Figure 5.2: MBP integration as Cloud Fragment

5.1.3 Inclusion of MBP

Components used to display data about the digital twin and its connections in MBP are considered part of the IDT. The way in which they are arranged and work together is illustrated in Figure 5.2. Here, the separation of the IDT into a LIDT and CFs can be seen. Furthermore, the way in which data is transmitted to the cloud is outlined.

The LIDT contains information about the car and the environment. In addition, it can find out about the state of connections by communicating with the AM. This data is aggregated from multiple subcomponents of the LIDT to an Internal File Writer, which is part of the LIDT. It is written to a file, which is then read by a script manged by MBP. As this script runs on the same hardware as the rest of the digital twin, it is not considered a part of the CF, despite running in a process separate from non-MBP components.

Finally, the Extraction Script transmits data to MBP. MBP then displays the contained information. An example for how this looks in MBP can be seen in Figure 5.3. Visible are two different aspect of the digital twin, which may be processed in a cloud. In Figure 5.3a, information about the environment and the digital twin itself can be seen. The state of ongoing connections are visible in Figure 5.3b. When CFs are used in practice, further analysis is likely to be conducted or information could be used as input for other applications in the cloud. It would also be possible to send data back to the LIDT. The resulting CF would be used for processing and considered as part of the IDT.

```
▾ Object
  sender_id: 1
  ▾ environment_info: Array [4]
    ▾ 0: Object
      id: "1"
      connected_eu: true
      connected_us: true
      current_speed: 4.009321781451521
      intended_speed: 4.955806596216828
      current_direction: "West"
      intended_direction: "North"
      next_maneuver_in: 89025.11286563633
    ▾ 1: Object
      id: "2"
      connected_eu: true
      connected_us: true
      current_speed: 24.9028176677661
      intended_speed: 9.244807918216331
      current_direction: "East"
      intended_direction: "South"
      next_maneuver_in: 49869.20626480183
    ▾ 2: Object
      id: "3"
      connected_eu: false
      connected_us: true
      current_speed: 16.76559126071863
      intended_speed: 8.209190226524676
      current_direction: "Direction not available"
      intended_direction: "Direction not available"
      next_maneuver_in: 23231.261761702015
    ▾ 3: Object
      id: "Not connected"
      connected_eu: false
      connected_us: false
      current_speed: -1
      intended_speed: -1
      current_direction: "Not connected"
      intended_direction: "Not connected"
      next_maneuver_in: -1
```


(a) Display of the state of the environment


```
▾ Object
  id: "1"
  number_connected_partners: 2
  ▾ connection_infos: Array [4]
    ▾ 0: Object
      partner_id: "2"
      ▾ eu_adapter: Object
        connected_eu: true
        time_last_msg_sent_eu: "27.09.2021, 14:32:25"
        time_last_msg_received_eu: "27.09.2021, 14:32:25"
        connected_using_tcp: true
        secure_mode_enabled: true
      ▾ us_adapter: Object
        connected_us: true
        time_last_msg_sent_us: "27.09.2021, 14:32:25"
        time_last_msg_received_us: "27.09.2021, 14:32:25"
    ▾ 1: Object
      partner_id: "3"
      ▾ eu_adapter: Object
        connected_eu: false
        time_last_msg_sent_eu: "not connected"
        time_last_msg_received_eu: "not connected"
        connected_using_tcp: false
        secure_mode_enabled: false
      ▾ us_adapter: Object
        connected_us: true
        time_last_msg_sent_us: "27.09.2021, 14:32:24"
        time_last_msg_received_us: "27.09.2021, 14:32:24"
    ▾ 2: Object
      partner_id: "not connected"
      ▾ eu_adapter: Object
        connected_eu: false
        time_last_msg_sent_eu: "not connected"
        time_last_msg_received_eu: "not connected"
        connected_using_tcp: false
        secure_mode_enabled: false
      ▾ us_adapter: Object
        connected_us: false
        time_last_msg_sent_us: "not connected"
        time_last_msg_received_us: "not connected"
```


(b) Display of the state of connections of the digital twin

Figure 5.3: Display of information about the digital twin and its connections in MBP

5.1.4 Functionality of the Prototype

The prototype is controlled using the command line interface seen in Figure 5.4. Using it, different functionalities can be accessed. They were chosen in order to be able to simulate the behavior of a real digital twin as accurately as possible.

Due to this, the interface includes functions enabling decisions which would normally be made by the AM. For example, a user can tell the digital twin to connect to another one by entering the partners ID. In addition to commands used to simulate decisions, it is possible to instruct the digital twin to pause sending messages from an Adapter for a short time. While such a decision would not be useful in practice, it makes it possible to demonstrate how a digital twin could react to an interrupted connection. When connecting, instances of the prototype use a convention to calculate their partners port number based on the ID. However, this is only done in order to send the first message. In any case in which a port number has to be exchanged after this, for example, when connecting Adapters, they are included in the additional information section of the AMM.

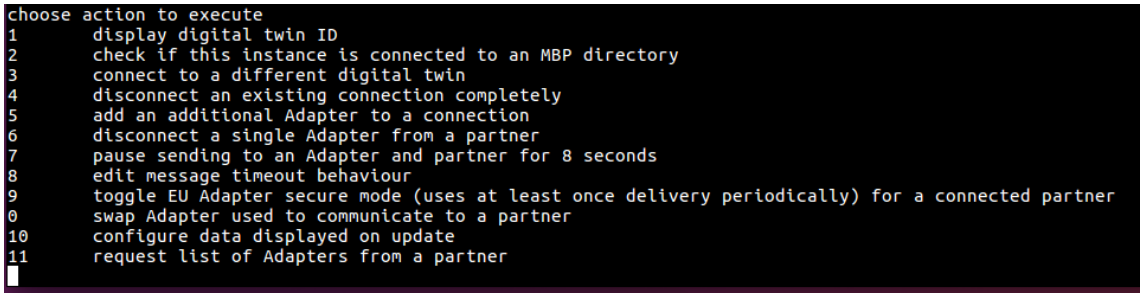

Figure 5.4: Command line interface of an instance of the prototype

Two different Adapters were devised for the prototype. They are called EU and US Adapter. Both are capable of exchanging information about their digital twin with a partner and receiving information about the partner. In order to construct a scenario which shows how the same information can be transmitted in different formats and interfaces, both Adapters support the exchange of an intended speed and a current speed. While in the IDT, both of these values are stored in m/s, the EU Adapter transmits this information in kph and the US Adapter in mph. When working with this information, both Adapters have to use code to transform the data stored in the IDT into the format and units expected by their interface.

Another goal of using two different Adapters was to show how the use of more than one at the same time can be advantageous. In the prototype, this is done by including direction information in the EU Adapters data set and the distance to the next maneuver in the US one. During operation, it can be seen that in order for the digital twin to have access to all available data, both Adapters have to be connected. This becomes apparent in Figure 5.3. As the digital twin with ID 3 is not connected using the EU Adapter, no information on its current or intended direction is available. In contrast, all fields are available for the twin with ID 2, which is connected using both Adapters.

Finally, in an effort to demonstrate QoS handling in the proposed architecture, different communication channels can be used. Normally, both Adapters use UDP for exchanging messages. However, the EU version supports a mode of operation in which every third message is sent via TCP. This approach was chosen to imitate how Adapters may have different QoS requirements for messages.

5.2 Practical Appraisal of Prototype

During implementation of the prototype, we were able to assess how the architecture influences the development of a digital twin. Regarding development time, it became apparent that the initial implementation of general components, such as the AM, can be time-consuming. However, once they are available, adding new Adapters and communication channels can be done quickly and easily. This means that after initial development, the digital twin can be extended without extensive work. One of the reasons for this is that loose coupling was achieved. For example, new Adapters can be implemented independently from many other components like communication channels or other Adapters. This makes it possible to develop interfaces separately, which we expect to be valuable due to the volatile requirements of the domain.

While developing the prototype, some errors had to be fixed. In these situations it became clear that making AMMs human-readable provides an advantage. JSON makes it easy to understand what the AMs are doing during their interaction. If an error occurs, the cause can often be found by considering what kind of messages were exchanged in which order. Furthermore, the log message that can be included in error messages is another helpful tool during bugfixing. It makes it possible to find the source of the error quicker, which makes development as a whole less time-consuming.

Another aspect which has to be considered when using the proposed architecture is the connection between Adapters and the IDT. Every piece of information exchanged between the digital twin and other devices in its vicinity has to pass between these components. Due to this, it is imperative for good performance of the overall system to design their connection with this in mind. Some factors which should be included in this consideration are the amount of data expected, the frequency in which it is updated, or how detailed the environment should be modeled in the IDT.

6 Evaluation

In this chapter, the proposed concept is evaluated. This includes an explanation of how challenges described in Section 3.3 on page 22 were met. In addition, some considerations regarding practical use of the architecture and concept are included.

6.1 Assessment of the Architecture

The proposed architecture appears to support loose coupling between components. This makes it possible to extend a digital twin without a large amount of effort. Furthermore, Adapters and communication channels as used in the architecture make it possible to cope with a highly dynamic environment, such as the one connected cars operate in.

As different interfaces can be used in different situations and a multitude of communication technologies can be supported, heterogeneity may even become an asset for developers using the concept. For example, a digital twin implementing many different Adapters may be able to more appropriately react to a given situation. In addition, different communication technologies can be used based on the current requirements.

Apart from supporting connections with devices in the vicinity of a car, this architecture also allows for parts of a digital twin to be located in the cloud. This supports many different approaches, for example, ones which rely on large processing power of powerful machines in the cloud. Simpler cloud applications can also be included, for example, a cloud solution which receives data from a car and makes it accessible to a mobile app for the owner.

It is also possible to develop CFs which digital twins can use to exchange information. These are considered part of the IDT. Therefore, if a digital twin does not support such a CF, it does not have to be able to consider this factor. Instead, the external interface of every other digital twin is still based on Adapters and communication channels with a unique identifier.

In addition to interoperability, a big advantage of the IDT being used to encapsulate the internal logic and decision-making is the possible stability. As long as the interface provided for Adapters supports their functionalities, little to no changes are expected to be necessary when adding new Adapters. This means that an IDT may be used for a long time without going out of date. Even if other technology for communication or interfaces become popular, other components can be used to bridge old and new interfaces.

Furthermore, the IDT can be upgraded independently of Adapters and communication channels. This enables developers to use new concepts in the internal logic. If older interfaces would normally not be compatible with them, Adapters can be used to transform the data as necessary.

6.1.1 Resolving of Design Challenges

In Section 3.3 on page 22, different challenges facing the concept were described. They were based on both the domain of connected cars and the nature of distributed systems. In the following, we will discuss how the proposed concept and architecture account for these aspects. As privacy and security were considered outside of the scope of this project, they will not be discussed here.

Scalability

We expect scalability to be influenced by the amount of devices a digital twin will communicate with and the amount of data exchanged. In an effort to make many concurrent connections with partners possible, the AM has to be implemented correctly. It contains a MH, which could be instantiated multiple times. As a result, AMMs could be distributed between these instances, thereby reducing the load on every single one. Furthermore, the use of different Adapters and communication channels makes it possible to support a large amount of connections. If a single Adapter or channel is not able to keep up, the AM can decide to change to a different one that is also appropriate. Another way in which the developers could improve scalability is by instantiating Adapters more than once. Similarly to using multiple instances of the MH, they could be managed in a way that ensures requests to this Adapter are distributed and do not overwhelm one instance.

The scalability in regards to the amount of data exchanged depends mainly on design decisions of the developer. One way in which related problems may be reduced is by providing Adapters which implement an interface that uses only a small amount of data. Furthermore, the interface of the IDT has to be designed in a way compatible with expected information volume. This can also include measures making sure the internal structure of the component is able to handle them, for example, by using efficient databases.

Heterogeneity

Heterogeneity is one of the main motivations for using different Adapters and communication channels. It is approached by having digital twins convey to each other in which ways they are able to communicate. Based on this information, they can decide upon the optimal Adapters and communication channels to use. Environment factors can also influence this decision.

Internally, every digital twin can use whatever concepts and technologies are preferred by the developer. As long as the necessary functionality is available in any way, an Adapter is able to make it accessible using a well-defined interface which is known by other participants. This system makes it possible to mask the complexities caused by heterogeneity. Furthermore, as the digital twin is defined in a technology-neutral way, future developments can be included without having to revisit the basic concept.

Extensibility and Development of Future Changes

The proposed architecture leads to a loose coupling of different components. Especially communication channels and Adapters can be changed without affecting many other aspects of the digital twin. The design is chosen this way because we expect these components to be subject to changes

the most. Due to this, the ability to modify them without having to significantly adjust other parts of the system is very valuable. As a central component, the AM could, in theory, be influenced by many different changes. However, while it is tasked with managing the digital twin as a whole, we expect handling of new components to be similar to existing ones in most cases. Some additional data may be required for considering them in decisions, but the main functionality of coordination with the communication partner's AM stays the same.

As a consequence, the presented digital twin can be extended to support new technologies and interfaces. Its functionality can also be updated, though this may require changes in the IDT. Ideally, only very limited coordination of different Adapters and communication channels should be necessary. This makes it possible to develop multiple ones concurrently, further improving the extensibility of the architecture.

Concurrency and Scheduling

Concurrency within the digital twin has to be considered by the developer. Depending on their goals and requirements, they have to instantiate the architecture in a way that supports them. One way in which concurrent processing could be supported is by using multiple instances of components that handle messages. Furthermore, changing the state of components in transactions may be a way to prevent problems from occurring due to inconsistent information.

In addition to concurrency, scheduling has to be considered. The way in which the AM schedules requests has to be decided by the developer, as this is a detail decision not included in the architecture. Messages addressed to Adapters can be prioritized in two ways. The first one is by including information on current utilization of resources in the IDT. Based on this, the properties of different calls to the behavior and data can be used to prioritize them. The other way in which a prioritization is possible includes the AWs. As all messages addressed to its associated Adapters are passed through this component, it can be commanded by the AM to delay them. The consequence of this is that messages reach an Adapter later. In most cases, this is expected to lead to a reduction in required processing resources. However, as the Adapter may also interpret this as an error, it is recommended to use the first option for delaying processing of less important Adapters.

Robustness

Robustness of the digital twin is included in different aspects of the proposed concept. The first way in which it is included is due to the ability of different components to monitor each other. For example, AWs aggregate information on the operation of their assigned Adapters. In addition, they can be notified if an error occurs by the Adapter itself. If an AW becomes aware of a problem, it notifies the AM, whose task it then is to try to resolve the issue.

In order to increase the robustness of the digital twin, developers may also introduce watchdogs to restart components if they crash. For example, it may be a good idea to add a watchdog to the internal structure of the AM. This idea can be further augmented by storing data in a way that ensures they are not lost in case of a crash. A component especially suited to this approach is the AM, as it contains a database in its internal structure.

Another aspect of the concept improving robustness are the error messages. They can contain log entries which make it possible for developers to find problems in their digital twin. While this does not contribute to immediate resolution of problems, it makes it easier for car manufacturers to improve the digital twins used for their cars. In the long term, this reduces the frequency with which connected cars communicate incorrectly due to software errors. This means that the overall system of communicating digital twins in an environment is improved, thereby contributing to robustness as digital twins will have to handle errors less frequently.

Real-time Capabilities

We expect digital twins in this domain to have to be able to communicate without large delays. In an effort to reduce the time needed to connect two devices, the quick connect functionality was introduced. It makes it possible to establish a connection using an Adapter in a short time.

Apart from establishing a connection quickly, it is important for real-time capabilities that the overhead for messages is as low as possible. In order to achieve this, they are passed through as few components as possible. As messages are processed in the Adapters, the only additional step is an AW. However, this is not expected to delay the message significantly. In both directions (to the Adapter and to the CL), an AW is tasked with passing messages on as quickly as possible. While a slight delay has to be accepted in order to keep statistics on the operation of the Adapter up-to-date, only small amounts of data should be evaluated. This means that the overall process is fast and should, therefore, not introduce any significant overhead.

Dynamic Management of the Environment

The AM is the component mainly responsible for navigating the dynamic environment of this domain. It is tasked with managing the ongoing connections. This includes what partners the digital twin is connected to, what communication channels are used for a connection, and what Adapters are needed to exchange currently relevant information. In order to communicate with other digital twins, the AM communicates with their respective AMs. For these interactions, the interface defined by the AMMs is used.

In order to make these decisions, the AM is well-connected within the proposed architecture. It is informed about the state of the CL and AWs notify it on the state of the Adapters. In addition, a connection with the IDT is available. This makes it possible for the AM to request information on the state of the environment. Using the knowledge available to it, the AM can manage communication with relevant devices. As a result, the digital twin is able to make decisions dynamically and can function in the domain of connected cars.

Discovery of Devices

We separated discovery in this domain into two different categories: discovery of communication partners and discovery of functionality and data provided by a digital twin. As the scope of the project was limited, ways to discover other digital twins able to exchange information in an environment were not included in the concept.

However, discovery of behavior and information is possible. This is achieved by the introduction of Adapters and their respective IDs. As every Adapter implements a defined interface, a digital twin can indicate the availability of the respective functionalities and data by using the Adapters ID. When digital twins communicate, they exchange information regarding their Adapters in the form of these IDs. Due to them being unambiguous, this communicates in a concise manner which interfaces are supported, without a need for expensive negotiation or ontologies.

6.1.2 Main Developer Concerns

As mentioned regarding some aspects in Section 3.3, the developer of a digital twin has to make some considerations about the way in which the architecture is instantiated. One facet of these design decisions regard robustness, which was covered in more detail in Section 6.1.1. In addition, it is important that no bottlenecks are introduced due to developer decisions. For example, using only communication channels with a low data rate may limit how much information a digital twin can exchange with partners. Furthermore, it may often be a good decision to use multiple instances of a component. Apart from the previously mentioned advantages, this facilitates a fast recovery and thereby contributes to the robustness of the digital twin.

CFs are another topic important topic for developers to consider. While they are allowed to communicate with each other if there is more than one, they have to be configured in a way which makes this possible. Depending on the technologies and providers of the underlying cloud infrastructure, this in itself may be a complicated task. Furthermore, it is important to keep consistency of the stored information in mind. The use of CFs should not cause problems due to inconsistent state of the IDT.

Finally, the developer of the digital twin is responsible for all functionality and data provided by the connected car. In order to be usable independently of requirements regarding these factors, the proposed concept is not designed with specific expectations regarding the type of communication and information exchanged. Instead, developers of a digital twin have to decide upon the behavior and data the car should provide. Then, they should use appropriate technologies to realize their goals using the IDT to encapsulate their code.

6.2 Practical Use Considerations

In this section, the proposed concept is evaluated regarding its use in practice. This includes factors, such as development and fleet management. Furthermore, an ideal use scenario is described in order to illustrate the value which can be added by the concept.

6.2.1 Development in Practice

When a digital twin using this architecture should be developed, it is recommended to start with a basic framework. This means that the AM and the IDT should be implemented first. After this, Adapters, their respective AWs, and communication channels of the CL should be developed.

As they should be as independent from each other as possible, they may be implemented at the same time by different teams. In this case, it is important to make sure there are no problematic interactions between different Adapters. This may happen if they access the same fields and functionality in the IDT or they manage similar aspects of the car.

Another decision affecting development is if CFs are used. If that is the case, appropriate ways for the LIDT to communicate with the cloud infrastructure has to be included. However, as a separation into CFs and LIDT is possible, CFs may be developed separately. A requirement for this to be feasible is a previously agreed upon interface for communication inside the IDT.

There are two different ways in which such a digital twin can be implemented. The first one is to start with the development of a new piece of software from the ground up. This may be more expensive at first, but could also have advantages. For example, if the design of the IDT is defined with the expected use case of providing functionality and data to Adapters, adding new ones may be simpler. The second way to implement a digital twin is to use existing software of a connected car. Code should then be separated to fit the tasks of each component of the proposed architecture. Then, the old code is complemented in order to fit into these components. Functionalities not found in the original system will have to be developed from the ground up. For example, code used to communicate with other AMs will most likely not be included in the initial system.

6.2.2 Possible Advantages and Challenges in Practice

When used in practice, a challenge of the proposed concept may arise due to the amount of stakeholders it can affect. In order for the concept to work, they have to use the defined architecture for their digital twins. If a majority of car manufacturers do not adopt this approach, it becomes unlikely that others do so as the benefit of it is limited if only few twins are implemented this way. Because of this, the interoperability provided by the concept requires different stakeholders to agree to use the proposed architecture.

A related problem is linked to the definitions of components. Adapters, communication channels, and versions of the AM all have to be identified and specified. In order to do this, some organization has to be chosen tasked with defining these items. Without this, it would not be possible to identify those aspects of a digital twin unambiguously. For example, it may occur that due to coincidence two car manufacturers use the same ID for their custom Adapters. In this case, the AM may still try to connect them. However, as their interface is not identical, no information could be exchanged. These identifications are only useful, if all of them are agreed upon by every stakeholder.

One of the ways in which these specifications can be used leads to an advantage. As every component is defined, a large amount of work can be saved by reusing these definitions. By using the identification, it is possible to immediately convey exactly which one is meant. Due to all stakeholders having to agree upon specifications, there is a much lower risk of misunderstandings caused by small differences in definitions. For example, if a car manufacturer intends to use a specific communication channel, there is a clear definition of how it should communicate with other digital twins. Similarly, the behavior and data provided by an Adapter is known from the start of implementation. This means that a developer does not have to account for sudden changes regarding these aspects. Instead, they can be assumed to be constant throughout the project.

Another advantage of the concept is that it can be extended to other devices. For example, a smart traffic light may also use this architecture. This way, it is possible for devices to support different interfaces. As cars of various ages will be present in traffic, smart devices can use this to provide older and newer interfaces for communication with all traffic participants. A result of this would be that smart devices can be used for a longer time without having to be replaced. Furthermore, new interfaces and therefore functionalities can be introduced more easily. The reason for this is that older interfaces can still be supported, even if new ones are made available. Consequently, a guarantee that all participants can use the new interface is not required to introduce it, as this does not entail a reduction in service quality for cars using an older standard.

6.2.3 Fleet Management Capabilities

The proposed concept for digital twins can be used to provide fleet management capabilities. Such a feature can be implemented using CFs. For every fleet management system, a single CF can be added. This fragment is used to exchange information between the fleet manager and the digital twin. For example, the connected car can receive commands on how it should behave or can send relevant information to the central authority. Due to the architecture allowing for multiple CFs to be used in a digital twin, it is possible to connect a car to more than one fleet management system. In practice, this may be useful as different stakeholders can communicate with the car. An example for such a situation may be that a vehicle is connected not only to the management system of its rental company, but is also managed by a traffic guidance system at the same time.

If a fleet of cars should be capable of exchanging information relevant only within the fleet, an Adapter can be implemented to fulfill that requirement. Similarly, an Adapter can be used to allow fleet members to communicate with other kinds of devices, which may be used to manage them. An example for such a device may be sensors on a parking lot reserved for cars of an organization.

These options make it possible to include fleet management systems into the digital twin. More than one such system can be included in the architecture. If this is done, the developer has to define priorities of the commands of the different systems. Alternatively, the digital twin can contain CFs only of fleet management systems that cannot cause a contradiction if used together. An example of how this can be achieved is by including systems so that not more than one manages a single aspect. When all of these factors are considered, it is possible to develop a highly connected digital twin that exchanges information with fleet management systems. If enough cars are controlled by such a twin, overall efficiency of traffic is likely to improve as a result of more data aggregation and high-level optimization.

6.2.4 Ideal Use Scenario

In an ideal scenario, traffic includes various connected cars developed by different car manufacturers. They all implement the proposed architecture and can offer diverse functionalities and information. For example, some of these cars may be autonomous while others are not. A connected car is able to react appropriately to a car it can not communicate with, but its digital twin tries to connect with as many relevant devices as possible. What devices are considered relevant may change depending

on the traffic situation and the priorities of the car, but digital twins are able to react accordingly. In order to make this possible, AMs communicate using different AMMs. Additionally, cloud infrastructure can be used to support concepts, such as remote processing and fleet management.

Using these ways of communication, connected traffic participants gain a better understanding of their environment and are able to coordinate actions. It is also possible to share data of different sensors. This can be used by digital twins to complement each others knowledge of their environment. If multiple cars include the same type of sensor, energy may be saved if their respective digital twins agree on using a single one. In this case, data of one sensor can be shared with other cars, which allows them to turn off their devices. This cooperation makes it possible for digital twins to receive data generated by newer and more accurate sensing technologies. As a result, older cars can profit from newer hardware, even if they do not include it themselves.

Updates of the digital twin can be provided remotely by the car manufacturer. These may include new Adapters, new communication protocols or a new CF. This approach is used to regularly update the software and keep the behavior and data of the car up-to-date. As the proposed architecture is based on modules, it is also possible for users to order functionality after the delivery of their car. This makes the software of a connected car into a constantly evolving system adapted to the requirements of various stakeholders.

6.3 Limitations

The main limitations of the concept have to do with the scope of the project. Due to it, the aspects security, privacy, and discovery of devices could not be considered. However, these are important factors when operating in practice. Consequently, the proposed concept may have weaknesses regarding these issues.

While a prototype was implemented to validate the approach, it is important to keep in mind that it was realized by only one developer. This can lead to the experience and assessment of the prototype containing some subjective bias. We tried to counteract this by evaluating how the architecture can be used as a framework to solve challenges of the domain.

A different limitation of the validation of the prototype stems from only one being implemented. Due to efforts to include features which make a demonstration of the overall concept possible, implementing more than one prototype would not have fit the scope of the project. As a result, the concept could only be validated by multiple instances of the same prototype interacting.

7 Summary and Outlook

We expect connected cars to become increasingly more common in the future. This opens up a lot of different opportunities, as they are able to exchange data in various ways. Based on different information needs and the current environment, connections with various devices can be established. Furthermore, it is possible to include cloud infrastructure to complement a digital twin. For example, expensive calculations may be moved to the cloud in order to utilize more powerful machines.

While the domain creates new opportunities, it also comes with challenges. Some of these have to be considered as connected cars interacting constitutes a distributed system. Others arise due to domain specific aspects, such as the dynamic management of the domain. Some of the most important problems that have to be solved include heterogeneity and extensibility. The reason for their significance is the amount of different stakeholders. In addition to car manufacturers and users, connected cars may be influenced by other interested parties. These may include regulators and fleet management providers.

In order to accommodate this diverse set of stakeholders and other requirements, a concept for a digital twin is proposed. Its core idea is to use Adapters to implement different interfaces, while encapsulating specialized and potentially proprietary systems of developers in an IDT. Inside this component, communication with cloud infrastructure can be included. Similarly to the Adapters, different approaches to communication can be supported in the CL using communication channels. Using different components to oversee and manage the state of this digital twin, connections with various other devices can be established in order to exchange information. The resulting digital twin can be used in the connected car domain and can be extended in order to meet future requirements.

In an effort to further evaluate the proposed concept, a prototype was implemented. Its assessment showed that while the initial framework may be time-consuming to develop, extending the digital twin afterwards is possible in a short amount of time. An evaluation of the concept as a whole showed that the digital twin addresses challenges of the domain appropriately. The main issue that was identified is how definitions should be decided upon. Some kind of central committee is a possible solution to this problem. For this to be practical, all stakeholders would have to agree upon one, which may prove difficult.

All things considered, the proposed concept appears to be useful for this domain. It brings many advantages, both in the general concept and in ways it might be used in practice. However, for use in a real environment, some aspects still have to be considered.

7.1 Future Work

Some future work options deal with further refinement of the concept. Due to the scope of the project, not all challenges of the domain could be considered. Consequently, designing a security and privacy view for the architecture might be an interesting topic. Another issue which could be added is the discovery of devices by the digital twin.

Furthermore, more work regarding the validation of the concept would be useful. Especially testing how prototypes developed by different teams interact with each other could provide valuable insight. With it, a better assessment of the interface and architecture may be possible.

A different direction which may also be interesting to pursue is if this digital twin could be useful in other phases of the lifecycle of a connected car. For example, the use during early development of the car using Adapters to show possible functionality could be explored. A similar topic may be the use of this digital twin in other domains. As more devices are used in different contexts, the advantages of this concept may be useful in other additional domains.

Bibliography

[21] *Industrie 4.0 Glossary*. https://www.plattform-i40.de/IP/Navigation/EN/Industrie40/Glossary/glossary.html. [Online; accessed 16-September-2021]. 2021 (cit. on p. 13).

[4020] P. I. 4.0. "Details of the Asset Administration Shell Part 1-The exchange of information between partners in the value chain of Industrie 4.0 (Version 2.0. 1)". In: (2020) (cit. on p. 15).

[AE17] K. M. Alam, A. El Saddik. "C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems". In: *IEEE access* 5 (2017), pp. 2050–2062 (cit. on pp. 15, 16).

[AVF20] R. Apanaviciene, A. Vanagas, P. A. Fokaides. "Smart building integration into a smart city (SBISC): Development of a new evaluation framework". In: *Energies* 13.9 (2020), p. 2190 (cit. on p. 11).

[BBE+17] H. Bedenbender, M. Billmann, U. Epple, T. Hadlich, M. Hankel, R. Heidel, O. Hillermeier, M. Hoffmeister, H. Huhle, M. Jochem, et al. "Examples of the asset administration shell for Industrie 4.0 components–basic part". In: *ZVEI white paper* (2017) (cit. on p. 13).

[BML+20] B. Boss, S. Malakuti, S. Lin, T. Usländer, E. Clauer, M. Hoffmeister, L. Stojanovic. "Digital twin and asset administration shell concepts and application in the industrial internet and industrie 4.0". In: *Plattform Industrie* 4 (2020) (cit. on pp. 13, 14).

[CM16] R. Coppola, M. Morisio. "Connected car: technologies, issues, future trends". In: *ACM Computing Surveys (CSUR)* 49.3 (2016), pp. 1–36 (cit. on pp. 11, 19).

[DGH+16] W. Dorst, C. Glohr, T. Han, F. Knafla, U. Loewen, R. Rosen, T. Schiemann, F. Vollmar, C. Winterhalter. "Implementation Strategy Industrie 4.0-Report on the results of the Industrie 4.0 Platform". In: *Bitkom/VDMA/ZVEI* (2016) (cit. on p. 13).

[FHS+20] A. C. Franco da Silva, P. Hirmer, J. Schneider, S. Ulusal, M. T. Frigo. "Mbp: Not just an iot platform". In: *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE. 2020, pp. 1–3 (cit. on p. 45).

[Fou17a] O. Foundation. *OPC Unified Architecture Part 1: Overview and Concepts*. Tech. rep. https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-1-overview-and-concepts/. Nov. 2017 (cit. on p. 15).

[Fou17b] O. Foundation. *OPC Unified Architecture Part 6: Mappings*. Tech. rep. https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-6-mappings/. Nov. 2017 (cit. on p. 15).

[GLPL14] M. Gerla, E.-K. Lee, G. Pau, U. Lee. “Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds”. In: *2014 IEEE world forum on internet of things (WF-IoT)*. IEEE. 2014, pp. 241–246 (cit. on p. 17).

[LM06] S.-H. Leitner, W. Mahnke. “OPC UA–service-oriented architecture for industrial applications”. In: *ABB Corporate Research Center* 48.61-66 (2006), p. 22 (cit. on pp. 14, 15).

[MLC20] R. Minerva, G. M. Lee, N. Crespi. “Digital twin in the IoT context: a survey on technical features, scenarios, and architectural models”. In: *Proceedings of the IEEE* 108.10 (2020), pp. 1785–1824 (cit. on pp. 19, 20).

[MM17] T. Malche, P. Maheshwary. “Internet of Things (IoT) for building smart home system”. In: *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*. IEEE. 2017, pp. 65–70 (cit. on p. 11).

[MT14] K. S. Mishra, A. K. Tripathi. “Some issues, challenges and problems of distributed software system”. In: *International Journal of Computer Science and Information Technologies. Varanasi, India* 7.3 (2014) (cit. on p. 23).

[MZ20] S. Mariani, F. Zambonelli. “Degrees of autonomy in coordinating collectives of self-driving vehicles”. In: *International Symposium on Leveraging Applications of Formal Methods*. Springer. 2020, pp. 189–204 (cit. on pp. 17, 22).

[SGBP16] M. Schleipen, S.-S. Gilani, T. Bischoff, J. Pfrommer. “OPC UA & Industrie 4.0-enabling technology with high diversity and variability”. In: *Procedia Cirp* 57 (2016), pp. 315–320 (cit. on p. 15).

[SKZ+20] A. Sharma, E. Kosasih, J. Zhang, A. Brintrup, A. Calinescu. “Digital Twins: State of the Art Theory and Practice, Challenges, and Open Research Questions”. In: *arXiv preprint arXiv:2011.02833* (2020) (cit. on pp. 11, 19, 20).

[TA17] E. Tantik, R. Anderl. “Integrated data model and structure for the asset administration shell in industrie 4.0”. In: *Procedia Cirp* 60 (2017), pp. 86–91 (cit. on pp. 13, 14).

[TS08] A. S. Tanenbaum, M. van Steen. *Verteilte Systeme: Prinzipien und Paradigmen*. 2nd ed. Pearson Studium, 2008. ISBN: 978-3-8273-7293-2 (cit. on p. 30).

[WF15] F. Wortmann, K. Flüchter. “Internet of things”. In: *Business & Information Systems Engineering* 57.3 (2015), pp. 221–224 (cit. on p. 11).

[Wol16] D. Wollschläger. “Preconditions, requirements & prospects of the connected car”. In: *Auto Tech Review* 5.1 (2016), pp. 30–35 (cit. on p. 21).

A Full Specification of Messages and Errors

A.1 List of Adapter Manager Messages

Please note: In this list, every message contains the JSON fields “id”, “sequence”, and “type”. Due to this, they are not explicitly mentioned every time.

A.1.1 Quick Connect

1. initial message
   a) type: “connect”
   b) JSON fields: version, adapters (Array), add info
2. connection accepted
   a) first response
      i. type: “connect resp”
      ii. JSON fields: version, adapter, channels (Array, only used if adapter can use multiple channels), add info
   b) second response
      i. only used if necessary, can be omitted
      ii. type: “connect accept resp”
      iii. JSON fields: channel (only used if “channel” was included in previous message), add info
3. no Adapters match
   a) contains a counteroffer
   b) can be handled by receiver like new initial connect message
   c) type: “connect resp”
   d) JSON fields: version, adapters (Array)
4. connection rejected
   a) type: “connect resp”
   b) JSON fields: adapter (null value)

A.1.2 Temporary Connect

1. initial message
   a) type: “temp connect”
   b) JSON fields: version, timeframe
2. connection accepted
   a) type: “temp connect resp”
   b) JSON fields: version, timeframe
3. connection declined

a) type: “temp connect resp”
b) JSON fields: version, timeframe (null value)

A.1.3 Disconnect

1. initial message
   a) Adapter 0 if all ongoing connections should be terminated
   b) type: “disconnect”
   c) JSON fields: adapter, add info
2. response
   a) type: “disconnect resp”
   b) JSON fields: adapter, add info

A.1.4 Add Additional Connection

1. initial message
   a) type: “add adapter”
   b) JSON fields: adapter, priority, channels (Array, only used if adapter can use multiple channels), add info
2. accepted
   a) first response
      i. type: “add adapter resp”
      ii. JSON fields: adapter, accepted, channel (only used if “channels” was included in previous message), add info
   b) second response
      i. only used if necessary, can be omitted
      ii. type: “add adapter resp”
      iii. JSON fields: adapter, channel (only used if “channel” was included in previous message), add info
3. connection declined
   a) type: “add adapter resp”
   b) JSON fields: adapter, accepted

A.1.5 Swap Adapter

1. initial message
   a) type: “swap adapter”
   b) JSON fields: old adapter, new adapter, priority, channels (Array, only used if new adapter can use multiple channels), add info
2. accepted
   a) first response
      i. type: “swap adapter resp”
      ii. JSON fields: old adapter, new adapter, accepted, channel (only used if “channels” was included in previous message), add info
   b) second response
      i. only used if necessary, can be omitted

   ii. type: “swap adapter resp”
   iii. JSON fields: old adapter, new adapter, accepted, channel (only used if “channel” was included in previous message), add info
3. rejected
   a) type: “swap adapter resp”
   b) JSON fields: old adapter, new adapter, accepted

A.1.6 Add Additional Channel to Adapter

1. initial message
   a) type: “add channel”
   b) JSON fields: adapter, priority, channel, add info
2. accepted
   a) first response
      i. type: “add channel resp”
      ii. JSON fields: adapter, channel, accepted, add info
   b) second response
      i. only used if necessary, can be omitted
      ii. type: “add channel resp”
      iii. JSON fields: adapter, channel, accepted, add info
3. rejected
   a) type: “add channel resp”
   b) JSON fields: adapter, channel, accepted

A.1.7 Remove Channel from Adapter

1. initial message
   a) type: “remove channel”
   b) JSON fields: adapter, channel, add info
2. response
   a) type: “remove channel resp”
   b) JSON fields: adapter, channel, add info

A.1.8 Swap Communication Channel

1. initial message
   a) type: “swap channel”
   b) JSON fields: adapter, priority, old channel, new channel, add info
2. accepted
   a) first response
      i. type: “swap channel resp”
      ii. JSON fields: adapter, priority, old channel, new channel, accepted, add info
   b) second response
      i. only used if necessary, can be omitted
      ii. type: “swap channel resp”
      iii. JSON fields: adapter, priority, old channel, new channel, accepted, add info

3. rejected
    a) type: “swap channel resp”
    b) JSON fields: adapter, priority, old channel, new channel, accepted

A.1.9 Manage QOS

1. type: “manage qos”
2. JSON fields: channel, content

A.1.10 Error (Response)

1. error
    a) type: “error”
    b) JSON fields: error id, log, error info
2. error response
    a) type: “error resp”
    b) JSON fields: error id, log, error info

A.1.11 Update ID

1. initial message
    a) type: “new id”
    b) JSON fields: new id
2. response
    a) type: “new id resp”
    b) JSON fields: old id, new id
3. confirmation
    a) type: “new id confirm”
    b) no additional JSON fields included

A.1.12 Check Connection

1. initial message
    a) type: “check connection”
    b) JSON fields: msg id
2. response
    a) type: “check connection resp”
    b) JSON fields: msg id

A.1.13 Get Version

1. initial message
    a) type: “request mgr version”
    b) no additional JSON fields included
2. response

a) type: “request mgr version resp”
b) JSON fields: version

A.1.14 Swap Manager Version

1. initial message
   a) type: “swap mgr version”
   b) JSON fields: priority, version
2. accepted
   a) type: “swap mgr version resp”
   b) JSON fields: version, accepted
3. rejected
   a) type: “swap mgr version resp”
   b) JSON fields: version, accepted, counteroffer (optional)

A.1.15 List Supported Adapters

1. initial message
   a) type: “list adapters”
   b) no additional JSON fields included
2. response
   a) multiple responses possible if large amount of data
   b) type: “list adapters resp”
   c) JSON fields: resp num, total num, adapters (Array)

A.1.16 List Supported Communication Channels

1. initial message
   a) type: “list channels”
   b) no additional JSON fields included
2. response
   a) multiple responses possible if large amount of data
   b) type: “list channels resp”
   c) JSON fields: resp num, total num, channels (Array)

A.1.17 Send Information on a Communication Channels

1. initial message
   a) type: “channel info request”
   b) JSON fields: channel
2. response
   a) type: “channel info request resp”
   b) JSON fields: channel, add info (null if channel not supported)

A.2 List of Error Types

A.2.1 Unequal Connection State

1. type: “connection state”
2. JSON fields: adapter (is equal to 0 if general connection is in question)

A.2.2 Invalid Request

1. type: “invalid request”
2. JSON fields: request sequence num

A.2.3 Invalid Message Format

1. as no sequence number may be extractable, a part or the full message can be included as a string of ones and zeros
2. type: “invalid format”
3. JSON fields: partial, message

A.2.4 Invalid Message Content

1. type: “invalid msg content”
2. JSON fields: sequence

A.2.5 Invalid Communication Channel

1. type: “invalid channel”
2. JSON fields: adapter, invalid channel, allowed channels (Array)

A.2.6 No Such Adapter

1. type: “no such adapter”
2. JSON fields: adapter

A.2.7 Unnecessary Reply

1. type: “unnecessary reply”
2. JSON fields: response sequence num

A.2.8 Adapter Crashed

1. can include additional information based on affected adapter
2. type: “adapter crashed”
3. JSON fields: adapter, add info

A.2.9 Communication Channel Crashed

1. can include additional information based on affected channel
2. type: “channel crashed”
3. JSON fields: channel, add info

A.2.10 Request Ignored

1. includes sequence number of message the ignored request conflicted with
2. type: “request ignored”
3. JSON fields: num ignored, conflict num

A.2.11 Same ID Detected

1. can also be triggered by a third digital twin connected to the ones with the same ID
2. type: “same id”
3. JSON fields: id

A.2.12 Other

1. main motivation to send this error is log entry included in error message
2. type: “other”
3. no additional JSON fields included