[HKM+23] M. Heithoff, M. Konersmann, J. Michael, B. Rumpe, F. Steinfurth:
Challenges of Integrating Model-Based Digital Twins for Vehicle Diagnosis.
In: International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C),
pp. 470-478, IEEE, Västerås, Sweden, Oct. 2023.

Challenges of Integrating Model-Based Digital Twins for Vehicle Diagnosis

Malte Heithoff, Marco Konersmann, Judith Michael, Bernhard Rumpe, Felix Steinfurth
Software Engineering
RWTH Aachen University
Aachen, Germany
https://se-rwth.de

***Abstract*—The integration of independently developed digital twins for automotive diagnosis in a service-oriented vehicle architecture into a complex systems-of-systems rises various challenges to be handled. These challenges have to be tackled in detail for each particular domain and technical system architecture. Current research lacks to discuss them for service-oriented vehicle architectures. Within the project AUTOtech.agil, we are developing a digital twin for automotive diagnosis. This paper describes the application scenario, discusses integration challenges in detail and identifies possible mitigation strategies for the challenges. This discussion allows us to identify areas where general mitigation techniques have yet to be found and to extract a concrete roadmap for the automotive diagnosis digital twins.**

***Index Terms*—Digital Twin, Model-Driven Engineering, Software Architecture, Automotive, Vehicle Diagnosis, Integration**

I. Introduction

The term software-defined vehicle (SDV) [1] describes the vision of software-driven automotive development, where functions are primarily defined by software. The research project AUTOtech.agil develops a service-oriented architecture for SDV to enable easy updateability and upgradability [2]. We efficiently develop digital twins (DTs) for *vehicle diagnostics* utilizing model-driven engineering (MDE). Model-driven digital twin engineering approaches [3] are applied for creating digital twins of cyber-physical systems [4], e.g., in production [5]–[9], for hospital transportation systems [10], wind turbines [11], robotic arms [12], spacecrafts [13], soccer robots [14] or indoor air quality management systems [15]. MDE approaches are also applied for DTs of software systems [16], cyber-bionical systems [17], or smart cities [18]. Within the automotive domain, up to now, digital twins for, e.g., for safely connected cars [19], vehicle system dynamics [20], smart electric vehicles [21], or electric vehicle battery systems [22], have been developed, however, MDE is rarely used for their development.

As the understanding of DTs often differs from domain to domain, prior to presenting our approach and the resulting challenges, a clarification of our understanding of DTs and digital shadows is required. The following definition is based on [5], [10], [23], [24], and was first published in [25]. It aims to provide a general definition for DTs applicable to various purposes. In our understanding, *"a digital twin of a system consists of a set of models of the system and a set of digital shadows, both of which are purposefully updated on a regular basis, and provides a set of services to use both purposefully with respect to the original system. The digital twin interacts with the original system by providing useful information about the system's context and sending it control commands."* [25] A *digital shadow* includes "a set of contextual data traces and/or their aggregation and abstraction collected concerning a system for a specific purpose with respect to the original system." [24] Thus, a DT might include several digital shadows for different purposes encapsulating the relevant data and relationships to relevant models. Clearly, these digital shadows as well as the digital twin itself are subject to change over time, as the related cyber-physical system can be updated and upgraded.

The requirement for easy updateability and upgradability in SDV requires the constant evolution of our DT. Thus, instead of developing and updating one highly complex DT each time, we develop compositional DTs, such that only the DTs of updated services need modifications. During DT engineering, we use architecture models in an iterative MDE approach [26] to refine the DT. The evolution of DTs is still a challenging research area [27]. While our compositional DT approach focuses on easing the DT engineering and evolution, it requires integrating small DTs. The integration of these model-based DTs in itself comes with many different challenges.

In [28], we identified and described 15 challenges for integrating DTs. In this paper, we discuss how these challenges apply to the application scenario of SDV in AUTOtech.agil and how we intend to handle the challenges. This paper's contributions are:

(a) A description of the application scenario of digital twins for automotive diagnosis in AUTOtech.agil.
(b) The discussion of the challenges for DT integration as stated in [28] in the context of the application scenario.
(c) A discussion of mitigations for these challenges in the given application scenario.

For this specific use case, we have identified 8 of the challenges as not relevant, 5 as relevant, and 2 as very relevant. The remainder of this paper is structured as follows: We present the application scenario and the technical concept of how we build

The work presented in this paper was funded by the German Federal Ministry of Education and Research (BMBF) under the grant no. 01IS22088A.

979-8-3503-2498-3/23/$31.00 ©2023 IEEE
DOI 10.1109/MODELS-C59198.2023.00082

DTs for diagnosis in Sec. II. Sec. III discusses the challenges for DT integration in the context of the application use case, and how we intend to handle the challenges. In Sec. IV we show a research roadmap that emerges from the discussion before we conclude.

II. Application Scenario

The vision of software-defined vehicles (SDV) is that the functions of vehicles are enabled through software. A key role in this vision is the use of a few strong computing resources that orchestrate services in the vehicle. Traditional vehicle manufacturing statically integrates software components to software architecture at design time. SDV instead envisions an easy updateability and upgradability of the software via dynamic architectures.

Fig. 1. Application scenario in the project AUTOtech.agil; Self-driving vehicles of different types (shuttles, transporters, taxis, and personal vehicles) provide mobility services. Multiple vehicles are managed by fleet managers.

A. Digital Twin Hierarchy

In the research project AUTOtech.agil, we develop digital twins (DT) for SDV for diagnosing issues within vehicles [29]. Fig. 1 sketches the project's vision of different types of self-driving vehicles: shuttles, transporters, taxis, and personal vehicles. Multiple vehicles are managed by fleet managers, e.g., for a transportation company or a shuttle service company. The project employs the automotive service-oriented architecture (ASOA) [2] for vehicles' software components. The ASOA defines multiple layers for service components, as shown in Fig. 2 (bottom to top):

- The **physical layer** shows the single electronic control units (ECU), which are interconnected via Ethernet.
- The **service layer** describes vehicle services, which provide and require interfaces, compute functions, and control the hardware, e.g., window motors, window control buttons, or the trajectory motor.
- The **function layer** describes the vehicle functions that emerge from the interplay of services.
- The **driving mode** layer describes different modes of the vehicle. The car may be in the *automated driving* mode. The *tele-operation* mode means that the vehicle is remotely controlled by a human operator in situations that the automated driving mode cannot handle. The *secure stop* mode means that the car securely comes to a stop in the case of irresolvable incidents.

The ASOA orchestrator controls how the vehicle services are interconnected via a mode automaton. The automaton switches between service interconnection modes based on triggers from the service meta data. E.g., when a laser scanner service for environment detection becomes unavailable, it can switch to a camera-based detection for safely stopping the car for maintenance. Fig. 2 shows active modes, functions, services, and ECUs as configured by the orchestrator at a given time as an example (blue background).

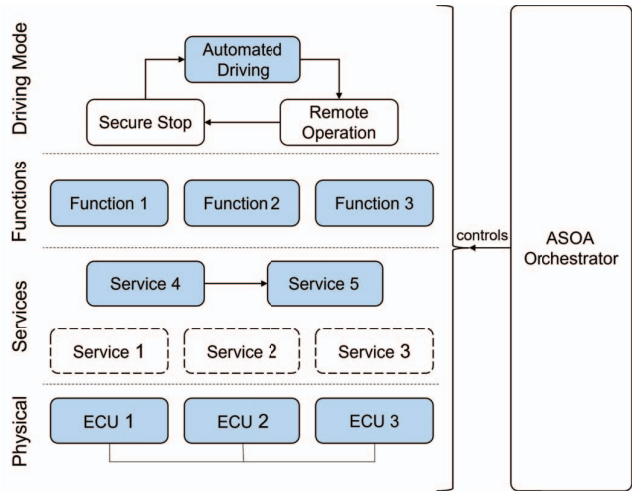

Fig. 2. Layered Architecture of the ASOA (adapted from [2])

The application scenario uses DTs on two levels for diagnosing issues in the vehicle: (1) vehicle services of the ASOA and (2) the vehicle level.

First, for *each vehicle service* we develop *a DT* for single vehicle service diagnosis (vehicle service DT). An example is a *diagnosis DT for the front left wheel* (and one each for the other three wheels).

As part of this *vehicle service DT*, *digital shadows* store time-related data about the wheel's runtime properties, including the pressure and the rotational speed, alongside meta data, including the service availability status, and quality data that describes the quality of the data provision. The latter can decrease over time, e.g., when the pressure sensor's precision decreases over time. A *diagnosis DT service* provides operations for diagnosing the data and meta data. Using an architecture modeling language, we model vehicle services and their behavior according to their specification and generate the respective vehicle service DT, which includes a *simulator DT service*. Thus, we can check whether the observed behavior of an ASOA service conforms to its specification. In the ASOA architecture, both hardware and software components are handled as services, and detectable errors on both hardware- and software side throw error codes. These error codes are explicitly handled by our DTs, which offer predefined diagnosis operations for each error code. As

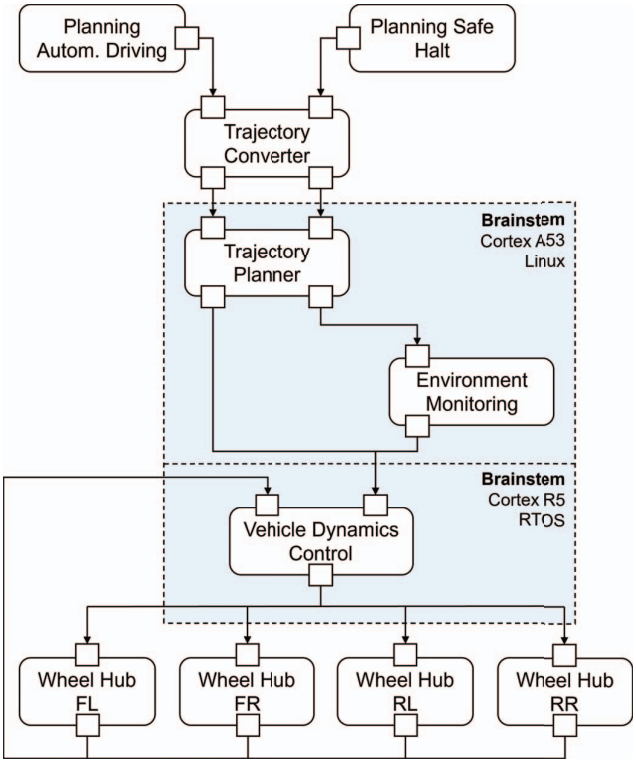

Fig. 3. ASOA ervices involved in planning automated driving and safe halting (adapted from [29]).

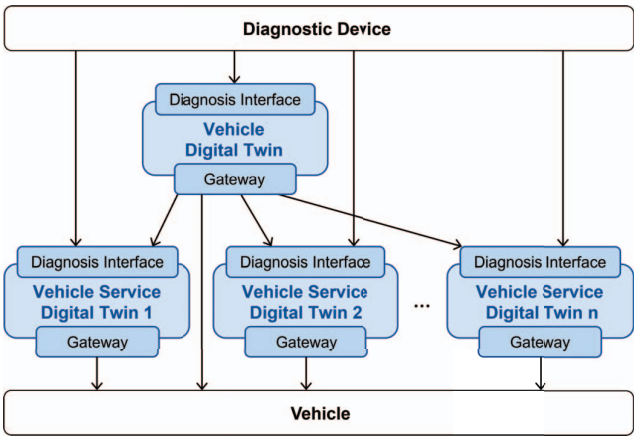

Fig. 4. The hierarchical control flow of the vehicle DT; it can combine the diagnosis operations of the vehicle service DTs to enable effect chain analysis.

both hardware and software components are services, we do not explicitly differentiate between software-induced errors, e.g., receiving an implausible message value, and hardware-induced errors, e.g., faulty memory causing the termination of a service. Instead, each kind of error has its own error code.

Diagnosis does not only operate on single-vehicle services. Multiple vehicle services may contribute to a diagnosis. Therefore, *secondly*, we develop a *vehicle DT* by *integrating the DTs of the vehicle services*. This vehicle DT also implements the orchestrator as a DT service that tracks vehicle service interconnections. A subset of the orchestrator automaton represents a single vehicle function, from which we can derive a DT service for function diagnosis. Fig. 3 gives an example of services in the ASOA for automated driving. A vehicle function in this context is given by the driving dynamics function, which controls the steering and the speed of the vehicle according to the trajectories calculated from the automated driving and safe halting planners. These trajectories then consider additional information from the environment monitoring service, before being sent to the vehicle dynamics control service. This service then executes the resulting trajectories by sending control commands to the wheel hubs. As each wheel hub in the AUTOtech.agil vehicles is steered and powered individually [30], the driving dynamics function coordinates the maneuvers by sending steering and motor signals from the vehicle dynamics control to each individual wheel hub, which in return sends status data back to the vehicle dynamics control. Thus, we introduce a diagnosis DT service for the driving dynamics function and compose it with the DTs of all-wheel hub services. The diagnosis service for the function in the vehicle DT therefore provides read/write access to the data and meta data required for the driving dynamics function and all wheel hub sensors. This approach creates a hierarchical composition of DTs for diagnosis with a vehicle DT integrating all vehicle service DTs. This hierarchical integration can be seen in Fig. 4.

A DT operator can run diagnostic operations using a diagnostic device. This device is able to connect to the vehicle DT and single vehicle service DTs and send *diagnosis requests* to both. The diagnostic device can also utilize the DTs to send *control commands* to the vehicle and its services via the service-oriented vehicle diagnostics (SOVD) [31] compliant interface. As an example, when diagnosing a defect driver window, we first check whether using the window lifter switch leads to respective data in the DT of the switch service. Second, we check whether sending a control command to the window lifter motor DT triggers the motor of the original system. Third, we check via the data in the DTs whether using the switch triggers a command from the switch ASOA service to the motor ASOA service. In case of any failures, we can diagnose the respective parts as defective.

The vehicle DT contains a DT service for the orchestration of the different vehicle services and their functions, which reflects the orchestrator in the ASOA, to diagnose the state of the orchestrator. It includes a database collecting the orchestration-relevant signals as well as the state of the orchestrator. On changes in the ASOA by the orchestrator, the orchestrator DT service in the vehicle DT updates the integration of the underlying DTs on the service and function layer. Thus, diagnostic operations on the vehicle DT can be relayed to the correct vehicle service DTs.

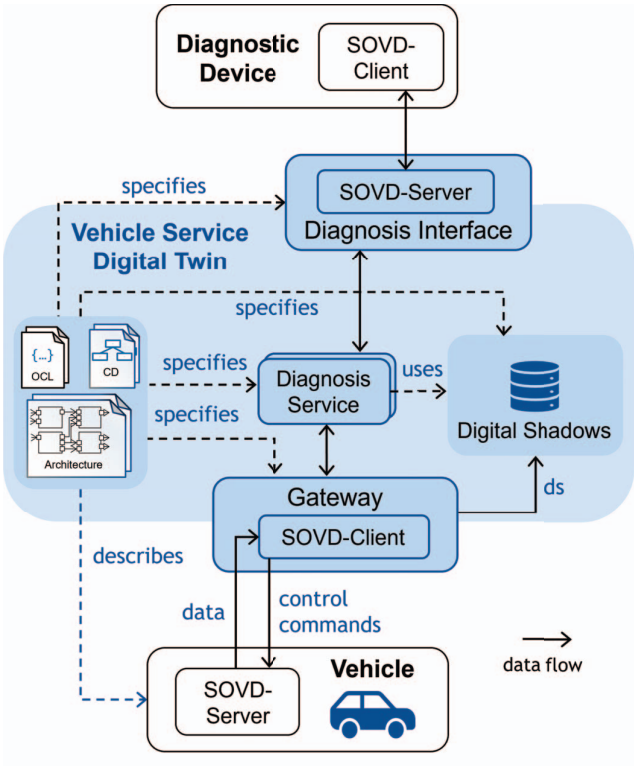

Fig. 5. The DT of a single vehicle service; model-driven engineering is utilized to generate digital twin parts from architecture diagrams, class diagrams, and OCL.

B. Digital Twin Engineering

The structure of our DT for a single vehicle service is shown in Fig. 5. Utilizing model-driven engineering, we generate parts of our DTs from models in the architecture modeling language MontiArc [32], [33], UML class diagrams, and OCL. Architecture models are used to describe service interfaces and the orchestrator. Additionally, the architecture models utilize automata to model the service behavior according to the ASOA service specification to detect service behavior that does not comply with the specification. Hereby, the degree of behavior modeling varies from service to service and can be underspecified or iteratively refined.

Additional service-specific operations can be generated by utilizing class diagrams and OCL to model classes for software errors of services. In the case of diagnosing the behavior of a single-vehicle service, these operations can be utilized directly by the diagnostic device. They contain suitable diagnosis queries for error analysis. Additionally, a diagnostic device can connect to the DTs by utilizing the diagnosis interface.

DT operations for diagnosis can be accessed from the diagnosis interface via SOVD. SOVD is a standard developed by ASAM e.V. It defines a tree structure for retrieving data and allows the execution of queries on the data. It is the basis of communication for our DTs with the diagnostic device as well as with the original vehicle and its services.

The transfer of time-related data from the vehicle to the DT is performed by a server that provides a SOVD interface inside the vehicle. This server pre-processes the data before sending it to the DT to reduce the amount of data while ensuring high quality. Therefore, data points of services that do not contain changing data get approximated while changing data gets recorded with a higher level of precision. For service data that influences other services as part of an effect chain, such a change in data requires more precise data of all affected services during the time frame of interest.

For basic diagnosis of services, identifiers and measured values can be read and control commands can be sent to vehicle service DTs via the SOVD interface. SOVD also provides suitable functions for the analysis of log files. Thus, the concept of errors can be mapped directly to SOVD by filtering the log files for error codes and calling the appropriate generated diagnosis operations of the DT according to the error code. These suitable diagnosis operations are modeled in our DT per error kind as a UML class. Thus, the error code itself does not require to contain additional diagnosis-relevant information, as the means to extract this information from the log files is either fully modeled in the respective software error class itself or can be extracted from our orchestrator log data in case of involvement of multiple services. This ensures a fixed size of the error code, which is important to satisfy time constraints regarding in-vehicle communication.

The general structure of our composite vehicle DT is the same as for the single service DTs. The difference is that architecture diagrams for the vehicle DT model the orchestration of the composed DTs and vehicle functions. Herein, the term *vehicle function* describes a vehicle functionality that can involve multiple vehicle services. Furthermore, class diagrams and OCL in our composite DT model errors on the vehicle or vehicle function level, meaning errors that involve multiple vehicle services for diagnosis, which therefore require access to multiple vehicle service DTs. An example of such an error would be a timeout of an expected input. As the vehicle services themselves do not have information about their interconnection, the vehicle is required to provide information on which service should have delivered said input. The interconnection of these vehicle services at the point in time that is observed by the diagnosis, can be derived from the orchestrator DT service. Thus, our composite DT controls the data flow between our single-service DTs and provides diagnostic operations dependent on information from multiple vehicle service diagnoses.

The *orchestrator DT service* allows to query (over time) and control the vehicle service composition. Hence, the vehicle DT can also analyze the behavior of the ASOA orchestrator. This is relevant, as the ASOA orchestrator does not only control service interconnection but also functionality that is dependent on multiple services. An example of such a vehicle function is given by the permission to open the door of a shuttle vehicle from the outside [2]. Three services contribute to the permission to open the door via four different rules:

1. The vehicle is stationary.

2. The vehicle is in one of the defined areas for boarding.
3. A passenger is within two meters of the door with his or her registered transceiver (e.g., a smartphone with an app).
4. Technical staff requests the opening.

The door may be opened from the outside if either rules 1, 2 and 3 are satisfied, or if rules 1 and 4 are satisfied. The vehicle's movement status is detected by the service for the traction motors. Whether the vehicle is in a specified area for boarding and disembarking is detected by a service that compares the position signal with regularly updated maps. The presence of an approaching passenger is detected by a service with a radio transceiver. The opening request of technical personnel is detected by a service that uses the same radio transceiver.

The part of the orchestrator that controls a vehicle function is modeled as its own automaton from which a function diagnosis service is generated. Thus, when diagnosing a specific function, the function diagnosis service with the respective sub-automaton that handles the function orchestration is used to access the relevant vehicle service DTs. The main advantage of splitting vehicle function diagnosis from vehicle diagnosis via an orchestrator subautomaton lies in the possibility to model additional function-specific diagnostic operations.

In the remainder of this paper, we will use similar terms associated with the word "service" or "DT" with very different meanings. To avoid confusion, we briefly describe these terms:

- **Vehicle Service**: A concept of the service-oriented vehicle architecture. Vehicle services are responsible for a specific task in the context of the use case, such as controlling the window motors.
- **ASOA Service**: A service in the ASOA architecture implements a vehicle service. Not every vehicle service must have an ASOA implementation, e.g., some vehicle services have a ROS implementation and an ASOA wrapper. For reference see Fig. 2.
- **Vehicle Service DT**: A digital twin of one vehicle service. It has a connection to the corresponding ASOA service or ASOA wrapper. For reference see Fig. 5.
- **Vehicle DT**: The digital twin of the entire vehicle. For reference see Fig. 4.
- **DT Service**: An active software component within a vehicle service DT or a vehicle DT, consisting of a state, functionality algorithms, and optionally also visualizations in a graphical user interface for human operators. Thus, other digital systems or humans may be users of DT services. For reference see Fig. 5.

III. Challenges

In this section, we discuss how the challenges mentioned in [28] apply to the application scenario above, and how we intend to mitigate them.

1. Horizontal integration of digital twin parts: Multiple views of the original systems, such as the driver's, the maintainer's, or the insurance's view upon an original system may lead to the necessity to integrate multiple DTs into one. In our application scenario, diagnosis is considered one of these views. Therefore horizontal integration on this level is not necessary.

We do, however, integrate multiple DTs for the diagnosis of multiple different original systems – the vehicle services – to build the vehicle DT. The service DTs can be used for isolated diagnostics of single services. These atomic parts are integrated horizontally to enable effect chain analysis for analyzing more complex diagnostic use cases. Additionally, we need to provide multiple visualizations and DT cockpits for the different DT parts.

This is a special case of horizontal integration because we can use the same modeling language for the DT models, which utilize composable languages, i.e., MontiArc [33] for composable architecture models, class diagrams for data structures, and OCL for data constraints. Therefore, on the modeling level, the integration uses well-understood composition mechanisms. On the architectural level, we use SOVD as a unified interface technology to technically integrate the DTs. We automatically generate code for these interfaces from the models.

Still, our DTs are provided with a single perspective: the vehicle diagnosis. Other perspectives on the vehicle or its services might be subject to DTs later on. This showcases the mentioned challenge: a DT, that is produced for one specific purpose, must consider its future integration into other contexts. From software engineering, we have successful methods to handle this: Modularization with clearly defined interfaces in standardized formats.

Main Finding: Horizontal integration is very relevant as we need to combine multiple DTs (for vehicle services) to build the vehicle DT.

2. Vertical composition of digital twins: The challenge of vertical composition means that the i) data, ii) service, and iii) models of a DT need to be composed when their respective DTs are composed. The challenges here include the different data frequencies, service operations required, or model granularity.

ASOA architecture specifications are not hierarchical, but all services are defined on the same layer. However, the vertical composition of DTs is still present in the form of vehicle DTs. Diagnostics of vehicles and their functions include the diagnosis of the composed services. An example is the function of automated door controls: allowing the doors to be opened from outside (a vehicle function) depends on multiple services. The speed of the vehicle has to be 0 and the current autonomous driving mission has to be completed [2]. For the purpose of diagnosis, we introduce a hierarchy of the vehicle and its service orchestration.

Composing the data of the vehicle (i) service DTs and the vehicle DT is simple due to the purpose of the DT and the composed nature of the vehicle functions and vehicle services. All data required for the functions require a respective definition in a DT service of the vehicle DT. Therefore the data is available in the required format and granularity. All vehicle service DTs provide uniform DT services (ii) for the purpose

of diagnostics. The SOVD standard describes composable interfaces via a tree structure, which propagates to the DT services and their queries and control commands. For the composition of models (iii), concepts and tools already exist (see challenge 1).

Main Finding: Vertical composition is very relevant as we need to compose different vehicle service DTs to enable vehicle function diagnosis.

3. Composition of DTs for different perspectives: This challenge tackles different perspectives upon an original system, e.g., DTs of the machines and of the employees on the shop floor of a factory. Vehicle diagnostics is the only purpose for the DT at hand, and the application scenario only provides this one, technical perspective on the issue. Thus, multiple perspectives are not relevant. One could argue that the distinction between software-induced and hardware-induced errors in the architecture can be handled from different perspectives. However, hard- and software components are considered services in the context of ASOA, with error types only distinguished by the respective error codes.

Main Finding: Composition of DTs for different perspectives onto the vehicle is not relevant to our scenario, as we just have the diagnosis as a singular perspective.

4. Connection of independently developed systems to a system-of-system: The services that the DTs are based on are developed independently with different technologies by different developers. E.g., services such as the automated driving component are implemented in Robot Operating System (ROS) instead of ASOA services. Therefore the connection of independently developed systems to a system-of-system comes as a challenge in this scenario. To mitigate this challenge, communication interfaces with the DTs are unified via the SOVD standard. We, thus, implement diagnosis interfaces for the DTs with SOVD to ensure a unified communication, independent of the vehicle service implementation. This also affects the virtualization, as the SOVD standard provides API composition (see challenge 2).

Main Finding: The connection of independently developed systems to a system-of-system is relevant in the context of vehicle services with different technology stacks.

5. Different lifecycle representations of the original system: Different lifecycle representations of an original system, such as *in-manufacturing* and *in-operation* are not the focus of our project. Therefore this challenge does not apply. However, diagnostic data could be used in the context of different lifecycle representations. One example of this would be a quality assurance twin for car producers, as one DT per vehicle or as one DT for all vehicles (or a respective DT composition). During manufacturing, diagnosis has another role than after production. Diagnosis information during the manufacturing phase can increase the effectiveness and efficiency of issue handling on single cars. A DT of all cars in manufacturing can help identify common hardware issues such as degrading sensor quality and fixing it in the production process for future cars, while identified software issues can be patched.

Main Finding: This challenge is not relevant for us, as different lifecycle representations of an original system are not the focus of our application scenario.

6. Protection of intellectual property: The DT for diagnosis collects data about the input, state, and output of services in the car and the orchestrator state. Assuming that multiple companies are involved in producing parts of these cars, the data can provide insights into the intellectual property of these companies. Hence it has to be considered who can access the data and execute a diagnosis based on the data. In the course of the project, we assume that the services of each part provide only the diagnosis information that the respective companies are willing (or obliged) to share with their partners and potentially the customer.

Main Finding: Protection of intellectual property is not a challenge in our scenario as we assume that the diagnosis information shared by the vehicle consists only of information the respective companies are willing to share. However, this might become a relevant topic in the future.

7. Privacy aspects of data and 8. Rights and roles in the integrated DT: A DT for vehicle diagnostics can define certain rights and roles for different diagnostic use cases. For a private vehicle owner, a role with access to a subset of simplified diagnostic information might be of interest, while the car repair shop needs access to the complete diagnostic interface. As already mentioned, in terms of lifecycle representation, whole product lines might be of interest to the car manufacturer. To address privacy concerns such as those arising from the EU's general data protection regulation (GDPR), any DT for a vehicle must respect the respective regulations. Privacy and security aspects with associated rights and roles need to be integrated. The SOVD standard specifies an authentication concept, that can be used to limit access to SOVD data, but no implementation technology or composition mechanism is defined. As the implementation is in our control, we can use technology with a common configuration of rights and roles. When the development of vehicle services is distributed over multiple stakeholders, standardization of this technology stack and configuration must be enforced.

Main Finding: Data privacy and rights and roles in the integrated DT are both relevant to our application scenario, as the result has to comply with data protection regulations. Thus, certain roles such as car manufacturers require disidentification of vehicle diagnosis data.

9. Composition of heterogeneous twin implementations: Within the presented application scenario the DT for diagnosis is developed by one single organization and is expected to be homogeneously developed in a model-based approach with code generation. As we expect the DT to be at least as long-living as the vehicle itself, we must also expect considerable software updates of the DT. The model-based approach provides the benefit of abstracting potential heterogeneity on the code level, leaving the generation of clear interfaces as a requirement. This provides a composition of heterogeneous implementations at run time. We also expect modeling techniques to evolve. This poses a challenge with respect to the composition of models [34], potentially built

with heterogeneous modeling languages and even language workbenches.

Main Finding: Composition of heterogeneous twin implementations is not relevant to our application scenario, as our model-based approach mitigates potential heterogeneity on code level by generating standardized interfaces and homogeneous DT implementations. However, heterogeneous DT models and DTs developed by different organizations pose a challenge for composition in the long term.

10. Conflicting constraints and requirements: Conflicting requirements in the application scenario are not DT-specific, but rather diagnosis-specific. An example of this is given by the fact that the DT requires exact data for diagnostics, while the DT operator wants to send less data to minimize cost. In terms of conflicting requirements between DTs, all of our DTs in the project are based on the domain of vehicle diagnostics.

A conflict in requirements arising in the scenario is the data granularity. Some services require fine-grained data, others avoid that due to data rate limits for the transfer to the DT implementation in the cloud. Effect chain analysis requires this fine-grained data. In our example scenario, we use a technique employed in vehicle diagnosis: we allow for configuring the granularity (e.g. frequency) of data at run time by control commands. During correct operation, only some information is transmitted. When a reason for a deeper look exists, we trigger a more fine-grained data transfer until a specified event takes place.

Main Finding: On DT-level, this challenge is not relevant to our application scenario, as our different DTs are diagnosis specific and therefore held to the same constraints and requirements.

11. Hierarchical functional abstraction: Vehicle service diagnosis requires all data provided by the vehicle services, while the diagnosis of vehicle functions only requires parts of that. This is a functional abstraction. The ASOA orchestrator defines the functions and their interconnection to services. The function services in the vehicle DT reuses them. Vehicle services in our application scenario are designed to provide the necessary data and control commands, albeit more than necessary. Therefore, functional abstraction is not a challenge in the context of our project.

Main Finding: Hierarchical functional abstraction is already defined by the ASOA orchestrator and therefore not a relevant challenge for our DT, since the existing concept can be reused.

12. Composition of interfaces DT2DT and DT2CPS The challenge of composing DT2DT and DT2CPS interfaces is mitigated in our application scenario due to standardized communication via SOVD, both for the real-world system. This holds for both DT2DT communication as well as DT2CPS communication.

However, standards are sometimes vague and allow for different implementations, which can cause issues. Therefore, the vagueness of the standard has to be mitigated by exact documentation of certain design decisions. Additionally, certain vehicle services such as the automated driving component are implemented in Robot Operating System (ROS) instead of ASOA services (see challenge 4). This also impacts the composition of interfaces for our vehicle DT, as ROS services wrapped as ASOA services are not as introspective as regular ASOA services and do not support traceable effect chains. Therefore, information on whether a given vehicle service is a ROS service or an ASOA service has to be included in the model to ensure that only valid diagnostic queries are offered by our composed vehicle DT depending on the service kind.

Main Finding: DT2DT and DT2CPS interface composition is relevant in the context of vehicle services with different technology stacks and vagueness in the SOVD standard.

13. Interoperability of models and simulation environments: The interoperability of models and simulation environments for multiple vehicles is not a challenge in the given application scenario. We define the environment of our DTs ourselves, and having multiple interoperable environments is not planned. However, on the level of vehicle service DTs interoperability comes into play. We simulate the behavior of the services and the interconnections in the DT for analysis. This is possible because the DTs are all based on the same family of modeling languages, which can be composed for the simulation.

Additionally, the nature of a service-oriented architecture facilitates a simpler composition, as each vehicle service can be modeled as a single, self-contained component, and their connections can be controlled during the simulation according to the orchestrator service of the vehicle DT.

Main Finding: Interoperability of models and simulation environment is relevant in the context of interoperability between vehicle service DTs.

14. Integration of graphical user interfaces: The integration of graphical user interfaces in our DT is a challenge, as the possibility to diagnose single services in our DT requires a well-thought-out solution for displaying requested information in a clear and concise manner. Displaying all information about every single service at once is clearly the opposite of that. As information concerning diagnostics is accessed manually when an issue arises, the graphical interface should highlight services that logged errors or showed unusual behavior. Additionally, a keyword-based filtering option should show just the services and service compositions tagged with those keywords. In the given application scenario a standardized interface exists towards the user's devices: the SOVD standard defines a tree structure for retrieving data and allows to execute queries on the data. The graphical representation is left to the devices. Thereby in the given use case, user interface integration is limited to the integration of multiple SOVD interfaces.

Main Finding: Integration of graphical user interfaces is not relevant in our application scenario, as the graphical representation of information is left to the diagnostic devices connecting to our DTs via SOVD.

15. Heterogeneous technology-stack and different distribution patterns of DTs: A heterogeneous technology stack and different distribution of our DT are not an issue in our application scenario. We define a homogeneous technology stack

and distribution scenario for the DT in our project. The main part of the DT exists in the cloud. However, parts of the DT for pre-processing data might be in-vehicle. The standardization of communication over SOVD solves possible issues concerning this distribution by defining respective SOVD queries.

Main Finding: A heterogeneous technology stack is not an aspect of our application scenario, and therefore not a challenge. Additionally, standardization of communication over SOVD mitigates different distribution patterns of our DTs.

Summary. The composition of vehicle service DTs and the vehicle DT is the main challenge in the application scenario. We identified three mitigations: Composition on the model level, unified interfaces, and uniform code generation.

Composition on the model level is possible because the languages in use are defined composably. By nature of a service-oriented architecture, unified interfaces already exist in the original vehicle, further easing interface construction between our vehicle service DTs. Unified interfaces via the SOVD standard are composable due to the data, query, and control command structure of SOVD. The uniform code generation helps us with the integration of the DTs via SOVD servers and interfaces, and uniform diagnosis services in the DTs.

In the application scenario, challenges regarding intellectual property and privacy and similar qualities are less important. Therefore, the mitigation of these challenges is not in focus for our project.

IV. Roadmap

The next steps in our endeavor to model-based engineering of DTs for vehicle diagnosis include *defining reference models* used for describing the services and *building code generators* for diagnosis using SOVD interfaces. We will build respective models for the application scenario in the research project. *Defining and implementing composition operators* for DT models play an important role in our work. We will use the work of Broy and Rumpe [35] for the composition of system engineering models as a reference. We enrich our architecture models with error classes that define specific diagnostic operations based on error codes thrown by the original system, which aids in providing assistance for error diagnosis. As we analyzed the required diagnostic information for certain errors, our next step in that regard is generating suitable diagnosis queries to collect said information. For the integration on the code level, we will define a *uniform code generator* and *unified interfaces* for the DTs.

We will *evaluate* the approach on increasingly complex service composition based on the project's reference use cases. Currently, we model the orchestrator DT service based on the original system implementation for the vehicle DT.

The solutions to the challenges discussed in this paper are specific to the given use case. It is interesting to investigate how these specific solutions can contribute to a *generic solution* or how they can serve as guidance to find similar solutions in other use cases, e.g., by describing them as *patterns*. In addition, these reference solutions can be connected to the reference models which describe the services.

V. Conclusion

Since automotive functions in SDV are updatable and upgradable, DTs of SDV must reflect these adaptations. For not adapting large complex DTs, we develop multiple smaller DTs of vehicle services, which we integrate into vehicle DTs. We use MDE to generate DTs from architecture models, data models, and data constraints. In this paper, we presented the application scenario of diagnosis for SDV in the research project AUTOtech.agil and the associated DT engineering method. We discussed the challenges of DT integration in the application scenario and measures to mitigate the challenges.

We identified 7 of the challenges of DT integration presented in [28] as relevant in the given scenario. While 8 challenges do not apply yet, we found that 2 out of the 7 relevant challenges are very relevant and have a large impact on DT engineering in the application scenario. However, for all identified challenges, we are able to conceptualize mitigations. The two most important challenges to overcome in our application scenario are the horizontal and vertical integration of DTs. Our mitigation techniques include model-based system design with composition on the model level, code generation from the MDE, and standardized interfaces. In future work, we will develop the model-driven DT generator for the application scenario for SDV in AUTOtech.agil.

References

[1] Z. Liu, W. Zhang, and F. Zhao, "Impact, Challenges and Prospect of Software-Defined Vehicles," *Automotive Innovation*, vol. 5, pp. 180–194, mar 2022.

[2] A. Kampmann, B. Alrifaee, M. Kohout, A. Wüstenberg, T. Woopen, M. Nolte, L. Eckstein, and S. Kowalewski, "A dynamic service-oriented software architecture for highly automated vehicles," in *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019.

[3] F. Bordeleau, B. Combemale, R. Eramo, M. van den Brand, and M. Wimmer, "Towards model-driven digital twin engineering: Current opportunities and future challenges," in *Systems Modelling and Management* (Ö. Babur, J. Denil, and B. Vogel-Heuser, eds.), Springer, 2020.

[4] K. Feichtinger, K. Meixner, F. Rinker, I. Koren, H. Eichelberger, T. Heinemann, J. Holtmann, M. Konersmann, J. Michael, E.-M. Neumann, J. Pfeiffer, R. Rabiser, M. Riebisch, and K. Schmid, "Industry voices on software engineering challenges in cyber-physical production systems engineering," in *IEEE 27th Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, IEEE, September 2022.

[5] M. Dalibor, J. Michael, B. Rumpe, S. Varga, and A. Wortmann, "Towards a Model-Driven Architecture for Interactive Digital Twin Cockpits," in *Conceptual Modeling*, Springer, 2020.

[6] T. Brockhoff, M. Heithoff, I. Koren, J. Michael, J. Pfeiffer, B. Rumpe, M. S. Uysal, W. M. P. van der Aalst, and A. Wortmann, "Process Prediction with Digital Twins," in *Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pp. 182–187, ACM/IEEE, October 2021.

[7] S. Fur, M. Heithoff, J. Michael, L. Netz, J. Pfeiffer, B. Rumpe, and A. Wortmann, *Sustainable Digital Twin Engineering for the Internet of Production*, pp. 101–121. Springer Nature Singapore, April 2023.

[8] D. Lehner, S. Sint, M. Vierhauser, W. Narzt, and M. Wimmer, "Aml4dt: A model-driven framework for developing and maintaining digital twins with automationml," in *26th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2021.

[9] A. Niati, C. Selma, D. Tamzalit, H. Bruneliere, N. Mebarki, and O. Cardin, "Towards a digital twin for cyber-physical production systems: A multi-paradigm modeling approach in the postal industry," in *23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[10] D. Bano, J. Michael, B. Rumpe, S. Varga, and M. Weske, "Process-Aware Digital Twin Cockpit Synthesis from Event Logs," *Journal of Computer Languages (COLA)*, vol. 70, June 2022.

[11] J. Michael, I. Nachmann, L. Netz, B. Rumpe, and S. Stüber, "Generating Digital Twin Cockpits for Parameter Management in the Engineering of Wind Turbines," in *Modellierung 2022*, pp. 33–48, Gesellschaft für Informatik, June 2022.

[12] P. Muñoz, M. Wimmer, J. Troya, and A. Vallecillo, "Using trace alignments for measuring the similarity between a physical and its digital twin," in *25th Int. Conf. on Model Driven Engineering Languages and Systems: Comp.*, ACM, 2022.

[13] N. Christofi and X. Pucel, "A novel methodology to construct digital twin models for spacecraft operations using fault and behaviour trees," in *25th Int. Conf. on Model Driven Engineering Languages and Systems: Comp.*, ACM, 2022.

[14] G. Walravens, H. M. Muctadir, and L. Cleophas, "Virtual soccer champions: A case study on artifact reuse in soccer robot digital twin construction," in *25th Int. Conf. on Model Driven Engineering Languages and Systems: Comp.*, ACM, 2022.

[15] H. S. Govindasamy, R. Jayaraman, B. Taspinar, D. Lehner, and M. Wimmer, "Air quality management: An exemplar for model-driven digital twin engineering," in *ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2021.

[16] M. Heithoff, A. Hellwig, J. Michael, and B. Rumpe, "Digital Twins for Sustainable Software Systems," in *IEEE/ACM 7th Int. Workshop on Green And Sustainable Software (GREENS)* (P. Lago and R. Kazman, eds.), pp. 19–23, IEEE, July 2023.

[17] I. David, P. Archambault, Q. Wolak, V. Vu, T. Lalonde, K. Riaz, E. Syriani, and H. Sahraoui, "Digital Twins for Cyber-Biophysical Systems: Challenges and Lessons Learned," in *ACM/IEEE 26th International Conference on Model-Driven Engineering Languages and Systems (MODELS)*, IEEE, 2023.

[18] J.-S. Sottet, P. Brimont, C. Feltus, B. Gateau, and J.-F. Merche, "Towards a lightweight model-driven smart-city digital twin," in *10th Int. Conf. on Model-Driven Engineering and Software Development MODELSWARD - Volume 1*, pp. 320–327, INSTICC, SciTePress, 2022.

[19] X. Chen, E. Kang, S. Shiraishi, V. M. Preciado, and Z. Jiang, "Digital behavioral twins for safe connected cars," in *21th ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems*, ACM, 2018.

[20] M. Spiryagin, J. Edelmann, F. Klinger, and C. Cole, "Vehicle system dynamics in digital twin studies in rail and road domains," *Vehicle System Dynamics*, vol. 61, no. 7, pp. 1737–1786, 2023.

[21] G. Bhatti, H. Mohan, and R. Raja Singh, "Towards the future of smart electric vehicles: Digital twin technology," *Renewable and Sustainable Energy Reviews*, vol. 141, 2021.

[22] F. Naseri, S. Gil, C. Barbu, E. Cetkin, G. Yarimca, A. Jensen, P. Larsen, and C. Gomes, "Digital twin of electric vehicle battery systems: Comprehensive review of the use cases, requirements, and platforms," *Renewable and Sustainable Energy Reviews*, vol. 179, 2023.

[23] P. Brauner, M. Dalibor, M. Jarke, I. Kunze, I. Koren, G. Lakemeyer, M. Liebenberg, J. Michael, J. Pennekamp, C. Quix, B. Rumpe, W. van der Aalst, K. Wehrle, A. Wortmann, and M. Ziefle, "A Computer Science Perspective on Digital Transformation in Production," *Journal ACM Transactions on Internet of Things*, vol. 3, pp. 1–32, 2022.

[24] F. Becker, P. Bibow, M. Dalibor, A. Gannouni, V. Hahn, C. Hopmann, M. Jarke, I. Koren, M. Kröger, J. Lipp, J. Maibaum, J. Michael, B. Rumpe, P. Sapel, N. Schäfer, G. J. Schmitz, G. Schuh, and A. Wortmann, "A Conceptual Model for Digital Shadows in Industry and its Application," in *Conceptual Modeling, ER 2021* (A. Ghose, J. Horkoff, V. E. Silva Souza, J. Parsons, and J. Evermann, eds.), pp. 271–281, Springer, October 2021.

[25] M. Heithoff, A. Hellwig, J. Michael, and B. Rumpe, "Digital twins for sustainable software systems," in *2023 IEEE/ACM 7th International Workshop on Green And Sustainable Software (GREENS)*, pp. 19–23, IEEE, 2023.

[26] M. Broy, M. Gleirscher, P. Kluge, W. Krenzer, S. Merenda, and D. Wild, "Automotive architecture framework: Towards a holistic and standardised system architecture description," tech. rep., White paper of the IBM Cooperation and technical report of the TUM, 2009.

[27] F. Bordeleau, B. Combemale, R. Eramo, M. van den Brand, and M. Wimmer, "Towards Model-Driven Digital Twin Engineering: Current Opportunities and Future Challenges," in *Systems Modelling and Management* (Ö. Babur, J. Denil, and B. Vogel-Heuser, eds.), pp. 43–54, Springer International Publishing, 2020.

[28] J. Michael, J. Pfeiffer, B. Rumpe, and A. Wortmann, "Integration Challenges for Digital Twin Systems-of-Systems," in *10th IEEE/ACM International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems*, pp. 9–12, ACM, May 2022.

[29] B. Böhlen, O. Meyer, B. Alrifaee, J. Beerwerth, A. Kampmann, S. Kowalewski, M. Konersmann, B. Rumpe, and F. Steinfurth, "Software-Defined Vehicle – Herausforderungen in der Diagnose dienstorientierter Fahrzeugarchitekturen," in *Tagungsband - Diagnose in mechatronischen Fahrzeugsystemen XVI: Software-Defined Vehicle, SOVD, Maschinelles Lernen und KI, Standardisierung, HU und ADAS*, TUDpress, 2023.

[30] T. Martens, L. Pouansi Majiade, M. Li, N. Henkel, L. Eckstein, S. Wielgos, and M. Schlupek, "Unicaragil dynamics module," in *29th Aachen Colloquium Sustainable Mobility*, 2020.

[31] Association for Standardization of Automation and Measuring Systems, "ASAM SOVD - API Specification Version 1.0.0."

[32] A. Haber, J. O. Ringert, and B. Rumpe, "Towards Architectural Programming of Embedded Systems," in *Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme VI*, vol. 2010-01 of *Informatik-Bericht*, pp. 13 – 22, fortiss GmbH, Germany, 2010.

[33] A. Haber, *MontiArc - Architectural Modeling and Simulation of Interactive Distributed Systems*. Aachener Informatik-Berichte, Software Engineering, Band 24, Shaker Verlag, September 2016.

[34] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap," *Future of Software Engineering (FOSE '07)*, pp. 37–54, May 2007.

[35] M. Broy and B. Rumpe, "Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung," *Informatik-Spektrum*, vol. 30, pp. 3–18, Februar 2007.