A Four-Layer Architecture Pattern for Constructing and Managing Digital Twins

Somayeh Malakuti(✉), Johannes Schmitt, Marie Platenius-Mohr, Sten Grüner, Ralf Gitzel, and Prerna Bihani

ABB Corporate Research Center, Ladenburg, Germany
{somayeh.malakuti,johannes.o.schmitt,marie.platenius-mohr,
sten.gruener,ralf.gitzel,prerna.bihani}@de.abb.com

Abstract. The promise of a digital twin is to make asset lifecycle information accessible by providing a single access point to the information. Thereby, it reduces the required time and effort and enables new data-intensive use cases. This paper provides an abstract four-layer architecture pattern to construct digital twins and to incorporate information from various kinds of sources. The pattern is designed to be flexibly extensible with new information sources and can flexibly support new kinds of proprietary or standard information. We discuss various alternatives to implement the pattern and provide an example realization based on microservices and OPC UA.

Keywords: Digital twin · Microservice · OPC UA · Information model

1 Introduction

A major problem of industrial systems are *information silos*. The information related to different lifecycle phases of an asset (e.g, a device, a production cell) is scattered across multiple information sources. These information sources are often maintained by different internal and external organizations and are interfaced by various applications. This leads to a broken information flow across the lifecycle of the assets because these information sources do not properly exchange information; some information may be duplicated or inconsistent while some others may be missing. As a result, significant amounts of time are usually required to find the relevant information, to convert the information to a suitable format, to pass the information to different tools, etc.

Digital twins can be seen as a promising solution for providing access to the lifecycle information of their assets. The underlying definition of the trend has evolved over time. Initially, a digital twin was considered to be a high fidelity mathematical model of a physical device that could simulate the device as closely as possible. This definition has been enriched over the time to be an evolving

The authors were partially supported by German Federal Ministry of Education and Research in the scope of the BaSys 4.0 project (01IS16022).

T. Bures et al. (Eds.): ECSA 2019, LNCS 11681, pp. 231–246, 2019.
https://doi.org/10.1007/978-3-030-29983-5_16

digital profile of the historical and current behavior of an asset together with all of its properties, where an asset is anything of value for an organization such as a physical device, a subsystem, a plant, or a software entity[1]. The information fragments contained in a digital twin are use case specific and often originate from different lifecycle phases. Typically, it encompasses elements such as ordering details, engineering parameters, operational information, and maintenance information.

Depending on the use case, differences arise in what kind of information must be collected, where it is found, and what it is used for. To have a unified solution, this paper proposes a four-layer architecture pattern to collect information about an asset from various information sources, to construct the digital twin of the asset and to aggregate the information in it. The pattern is designed to be flexibly extensible with new information sources and can support new kinds of proprietary or standardized information. We outline various alternatives to realize this pattern and sketch an example implementation, which is validated using our industrial use case, based on microservices and OPC UA [1] technologies.

In the next section, we describe an industrial use case for digital twins from ABB. Section 3 lists requirements for a solution to create and manage digital twins. Section 4 introduces our four-layer architecture pattern for such a solution, while Sect. 5 discusses a concrete realization of this architecture. Section 6 outlines the related work, and Sect. 7 concludes the paper with a discussion about the future directions of our research.

2 An Industrial Use Case in the ABB Company

Variable speed drives are a common asset in industrial plants to control the speed of motors. In our real-world example, we show digital twins for both the drive instance and its type. The latter contains type-design information such as CAD drawings, simulation models, and documentation. The former is the actual device used by a specific customer. While all of this information could be stored in a single digital twin, this would result in a massive amount of information replication as the type information is the same for all instances. Therefore, we envision two digital twins – one for the instance and one for the type. Since they are related to each other, this relation is also established among their digital twins.

The bottom part of Fig. 1 shows the case where the information related to different lifecycle phases of the drive and its type is scattered across multiple information sources. Here, the information about drive types is maintained in the so-called *Product Type Database*. Various proprietary applications are used, for instance the *Device Selection Tool* is used to select a specific drive type based on desired settings of customers. Information about the installations such as associated customers, plants, purchased devices and warranty information is maintained in a so-called *Installed-base Database*. The operational parameter

[1] https://www2.deloitte.com/insights/us/en/focus/industry-4-0/digital-twin-technology-smart-factory.html.

information of each drive is maintained within the drive firmware itself and served via an OPC UA (IEC 62541 [1]) server; hence, the drive is labeled as an *OPC UA-Enabled Drive*. Further information may be provided by *External Platforms*, e.g. production information for drive components manufactured by external suppliers.

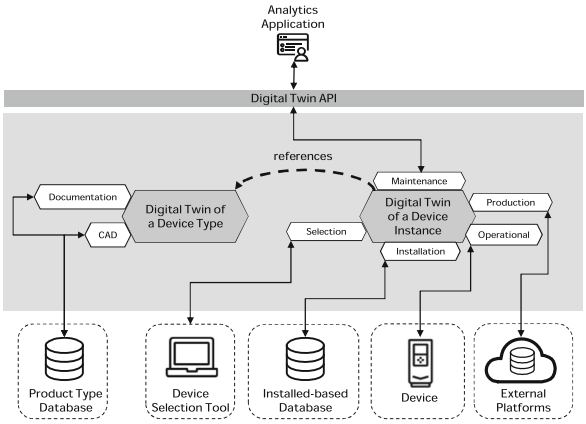

Fig. 1. A digital twin example

As depicted in Fig. 1, the digital twin is considered a means to provide a single entry point to the information of a drive and its type. Each digital twin contains information about the relevant asset. Having a digital twin in place helps to not only easily access the information but can even add further intelligence to the system. For example, there may be an analytics application which receives inputs about a drive's type, initial drive selection parameters, and operational values from the drive's digital twin to calculate its current health status.

3 Requirements

Based on our interviews with internal domain experts and evaluation of existing proposals for digital twins, we identified the following requirements to be fulfilled by a digital twin solution.

R1, Supporting Multiple Information Sources: To make a digital twin solution usable in practice, it must be possible to collect information from diverse sources and feed them to the relevant digital twins. Since the information sources are already operating for many years, only seamless extensions are allowed for enabling them to exchange information with digital twins.

R2, Supporting Modular Extension of Digital Twins: Information pieces are developed and delivered separately during the lifecycle of an asset. For example, when a new drive is installed in a plant, service information is only added once maintenance services take place. Therefore, it must be possible to incrementally extend the relevant digital twin content upon the availability of new information.

R3, Supporting Various Digital Twins Logics: Different digital twins may differ in terms of the information pieces that they enclose, the frequency of information updates, the lifetime of the digital twin, etc. For example, a product type exists before a concrete product is manufactured. Therefore, the digital twin of a product type has a different lifetime than the digital twin of a product itself. We name these the logics of digital twins. A digital twin solution must facilitate defining various digital twins with different logics.

R4, Supporting Information Push and Pull: It must be feasible to actively add information pieces to relevant digital twins upon their availability (push-based), or to query relevant information from passive data sources (pull-based). For example, in our use case, organizations usually have databases in which information related to all devices are stored; for each device, relevant information must be queried and added to the relevant digital twin. Each device may also actively bring more information (e.g., its operational parameters) to its digital twin.

R5, Syncing Information Between Digital Twins and Information Sources: The information enclosed by digital twins may be modified over time. To maintain information consistency, it must be possible to sync information between digital twins and the information sources. A special case is syncing information between a digital twin and the corresponding asset so that the information of the digital twins can be updated based on the actual information of the assets.

R6, Supporting Various Information Formats: Information pieces might be expressed in proprietary formats or based on different standards to facilitate interoperability across organizations. It must be feasible to collect and convert the information pieces and/or an entire digital twin to the desired format in a permanent way or on the fly when needed. The knowledge about which information formats are supported and processable by a certain component needs to be retrievable by the digital twin solution.

R7, Offering Dedicated Interaction Mechanisms for Each Information Piece: Each proprietary or standardized information piece requires a suitable user interface to display information and allow user-interaction.

R8, Identifying Digital Twins and Their Corresponding Asset: Since industrial systems may consist of thousands of assets (e.g. installed devices), it is necessary to provide a means to uniquely identify the assets and their corresponding digital twins on the network. Usually multiple identification schemes are in place within

one organization, meaning that the information pieces collected from various sources might have different identification schemes. Therefore, means are needed to map these identifiers to each other in order to identify the multiple information pieces that are relevant to a given asset.

R9, Offering Digital Twin APIs: Suitable APIs must be offered to applications to access and manipulate information stored within digital twins.

The above-listed requirements mainly cover functional aspects of a digital twin solution. Several non-functional requirements such as security and distribution are also relevant, but out of the scope of this paper.

4 The Architecture Pattern for Digital Twins

Since collecting lifecycle information from various sources and making it accessible via digital twins is a recurring problem in companies, we propose an architecture pattern for managing digital twins and their information sources. This helps various business units or companies to adopt a unified solution for implementing their digital twins.

Figure 2 shows our four-layer architecture pattern for a digital twin solution. The bottom level is the *Information Providers* layer, which consists of various information sources (satisfies *R1*). The *Model Providers* layer is responsible for gathering and processing information pieces from the *Information Providers* layer, and feeding it to the *Digital Twin Providers*. In our pattern, we refer to information pieces as models, since they can be expressed in various proprietary or standard formats. The *Digital Twin Providers* layer creates and manages digital twins. Various applications, e.g. viewing and analytics applications, can be located at the *Applications* layer, which can access and manipulate digital twins.

This pattern does not make any assumption on the cardinality and distribution of the depicted components. We leave these decisions to the implementations of this pattern based on the quality attributes of the specific use cases. In Sect. 5, we list a set of design alternatives for our use case.

4.1 The Information Providers Layer

We distinguish among the following kinds of information sources.

Applications/Tools: Digital twins must include the output of various tools that exist in organizations. For example, in ABB, dedicated tools exist to select drives suitable for a specific application along with their connected motors, and to parameterize them. The output of these tools is usually stored as files in a specific format, and/or in some kind of databases.

Devices: The operational parameters of devices (e.g., temperature or speed), which are defined within the firmware software of the devices, are another source of information. It is becoming commonly accepted that future Industrial Internet of Things (IIoT) devices will be delivered with an embedded

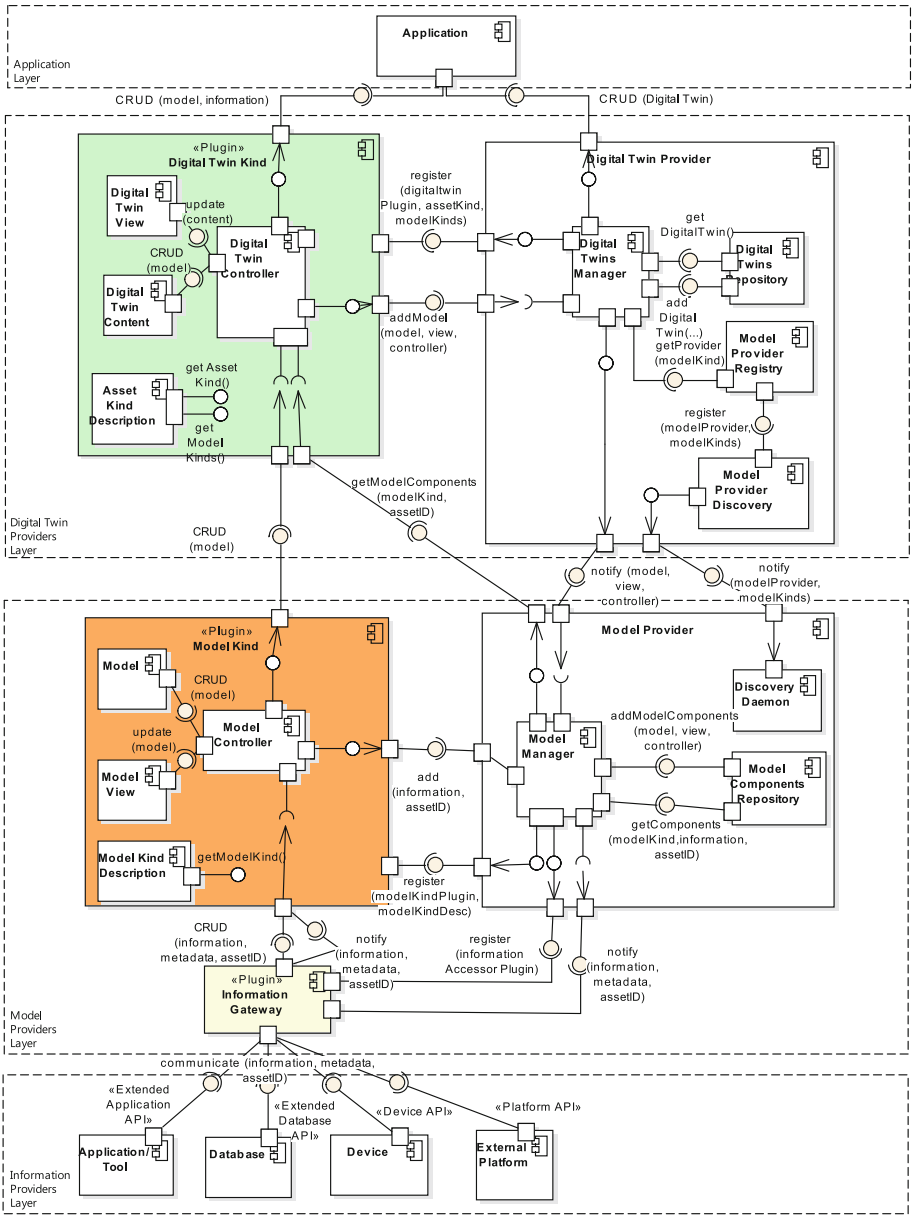

Fig. 2. The architecture pattern

information model in the OPC UA [1] format for defining at least the operational parameters and methods of the devices. Such devices already implement the functionality to sync this information with the relevant information within

their firmware. The operational parameters of the devices can be discovered and added as part of the devices' digital twin.

Databases: There are different databases such as *Product Type Database* and an *Installed-base Database*, which contain various information about device types and their installations. These databases usually offer APIs, which can be used in a seamless manner to exchange information with digital twins.

External platforms: The information about a device and its type may be scattered in multiple IoT platforms, possibly owned by different companies. Such platforms are also sources of information that is described in a standard/agreed format or in the proprietary format of the source platform.

In addition to these, we have two special kinds of information providers:

Digital twin applications: The applications using information about digital twins (the top layer of Fig. 2) may also modify the content of digital twins. For example, an application may suggest new maintenance services to be performed on a device and this information can be included in the digital twin of the device. These modifications take place using the digital twin APIs and, if needed, are communicated to the relevant information providers.

Digital twins: Other digital twins provided by the same or different IoT platforms may also be a source of information. For digital twins within one platform, we assume that the information exchange among them takes place within the context of specific applications located at the *Applications* layer. These applications make use of the APIs offered at the *Digital Twin Provider* layer to access digital twins and establish information links among them.

4.2 The Model Providers Layer

We adopt a plugin-based architecture to make the *Model Providers* layer extensible with new kinds of information sources and new kinds of models at runtime (satisfies *R2*). The *Information Gateway* plugin at the bottom part of this layer interacts with information sources to retrieve and/or update information. Dedicated plugins can be defined for each kind of information source.

The *Model Kind* plugin defines necessary components to provide and manage different kinds of models (satisfies *R6*). There is the so-called *Model Kind Description*, describing which kind of model is supported by particular plugins; example model kinds are documentation, CAD drawing, operational parameters, and maintenance. Dedicated plugins may be provided for each model kind. The model descriptions will later on be used at the *Digital Twin Provider* layer to match the models to the relevant digital twin.

Within the *Model Kind* plugin, we adopt the Model View Controller (MVC) pattern [2] to manage each model that is to be included in a digital twin. The information content corresponds to the *Model* component of this pattern. Since each model is defined based on a specific standard or in a proprietary format, there is a dedicated *View* associated to each model to visualize its content in a suitable format. Each model is also associated with a *Controller*, which communicates with the respective view and also populates the model by interacting

with *Information Gateway* (view and controller satisfy *R7*). The interactions with *Information Gateway* may be for creating, reading, updating or deleting (CRUD) information, as well as receiving notifications about the changes in the source information.

The *Model Provider* component is the core component, in which other components are plugged in. Here, *Model Manager* instantiates the MVC components when needed, and keeps a reference to them in *Model Components Repository*. *Model Manager* interacts with the *Digital Twin Provider* layer through its APIs; for example, it receives commands to construct a model, or to provide a model to *Digital Twin Manager*. The information exchange with the *Digital Twin Provider* layer can be push- or pull-based; i.e. the information is actively pushed to, or can be queried by the *Digital Twin Provider* layer, respectively (satisfies *R4*).

Once the models are constructed and added to the relevant digital twins, the associated *Controller* component may also be the recipient of the commands from the *Digital Twin Provider* layer; for example, to update the content of the model in a special frequency.

The availability and multiplicity of model providers may change during the lifetime of a digital twin. To be able to cope with such changes, our architecture opts for a mechanism to discover model providers. The necessary information to perform such a discovery is provided as *Discovery Daemon*, which notifies the *Model Provider Discovery* component in the upper layer.

4.3 The Digital Twin Providers Layer

The core part of this layer is the *Digital Twin Manager*, which is responsible for managing the lifecycle of digital twins, i.e. construction and destruction. It discovers model providers and their supported model kinds via the *Model Provider Discovery* component, which registers the providers at the *Model Provider Registry*. This registry allows *Digital Twin Manager* to retrieve the matching model provider for a given model kind.

There can be different kinds of digital twins within one system. For example, in our use case, we distinguish between the digital twin of a drive type and that of a drive instance. Each contains different kinds of models collected from different information sources. We adopt a plugin-based architecture to make the *Digital Twin Providers* layer extensible with new kinds of digital twins (satisfies *R3*). Here, *Asset Kind Description* specifies desired model kinds for each kind of digital twin. For example, for the digital twin of a drive type, documentation models and CAD drawings are model kinds of interest.

Similarly to each individual model, we also adopt the MVC pattern to manage each kind of digital twins. Here, the actual *Digital Twin Content* forms the model component of the pattern. The content of a digital twin is the aggregation of the models collected from the *Model Providers* layer, as well as an identification mechanism to bind these models together (satisfies *R8*, see Sect. 5 for details). A *Digital Twin Controller* component associated with each digital twin instance updates the model and interacts with the *Digital Twin View* component.

The point of time at which a digital twin must be constructed is use case specific. For example, users may explicitly construct a digital twin via a dedicated *Application* and start collecting information from model providers; another way is to automatically construct a digital twin upon the presence of its first constituent model. Either way, the *Digital Twin Manager* instantiates a *Digital Twin Controller* and passes the control to it to proceed with content acquisition from model providers.

The information about digital twins and their MVC components are maintained in the *Digital Twin Repository*. Suitable APIs are offered to applications to access and manipulate digital twin information.

4.4 The Applications Layer

Various use case specific applications may be developed to work with digital twins. These applications may work at the level of digital twins to create, read, update, and delete (CRUD) digital twins; or, they may work with the information contained within digital twins. For the latter case, the applications may directly interact with the respective *Digital Twin Controller*, and for the former case with *Digital Twin Manager*, both of which provide appropriate APIs (satisfies *R9*).

5 A Concrete Architecture Example

This section discusses an example implementation of the architecture pattern for our use case in Sect. 2. We have implemented our example in OPC UA [1] technology. OPC UA is becoming the de facto machine-to-machine communication protocol for industrial automation systems, which also offers an object-oriented information modeling mechanism as well as a service-oriented architecture to access the information. One faces various alternatives in implementing our abstract architecture. Figure 3 summarizes some alternatives and our decisions for our use case. We will return to them after we explain the concrete architecture, as depicted in Fig. 4, in the following subsection.

5.1 The Model Providers Layer

Since various pieces of information are delivered by different organizations, we define a separate model provider for each information provider, so that we can flexibly extend our implementation with new information providers. One may consider different cardinalities in other use cases; for example, a use case may require that one model provider accesses many information providers, or vice versa.

In our use case, we have the following kinds of information providers: (a) Two existing databases, i.e., *Product Type Database* and *Installed-base Database*, (b) the *Device Selection Tool* that generates information about the initial parameterization of drives, (c) a drive with embedded OPC UA server, and (d) an external platform providing production information.

Category	Decision Point	Alternatives	Adoption in the Prototype
Information Provider Layer	Cardinality of Model Providers w.r.t Information Providers	1:1, 1:n, m:1, m:n	1:1, one model provider per information provider
	APIs to Information Providers	Standard vs. Proprietary	OPC UA standard API to access device, proprietary APIs to access other information providers
Model Provider Layer	Cardinality of Model Providers w.r.t Models	1:1, 1:n, m:1, m:n	1:1, one model provider provides one type of model, except for type-specific models, which is 1:n
	Model Content w.r.t Information Providers	Copy vs. Reference	Referencing the operational parameters of the drive, copying the rest
	Model Content Format	Proprietary vs. Standard	Proprietary models
	View Execution	Server-side vs Client-side	Client-side execution of HTML pages
	View Kind	Desktop-based vs. Web-based vs. Native Mobile-based	Web-based views
	Model Provider Distribution	Distributed vs. Centralized	Distributed microservices
Digital Twin Provider Layer	Distribution	Centralized vs. Distributed	One Digital Twin Manager centralized instance
	Cardinality of Model Providers w.r.t Digital Twin Providers	1:1, 1:n, m:1, m:n	m:1, multiple Model Provider microservices, one centralized Digital Twin provider
	Cardinality of Digital Twin w.r.t. Asset	1:1, 1:n, m:1, m:n	1:1, one digital twin for each device
	Communication Kind	Pull-based vs. Push-based	Pulling installation and type information, Pushing drive operational model and maintenance model
	Information Storage	Any storage such as OPC UA servers, MongoDB, etc.	OPC UA servers
	Digital Twin Content w.r.t Model Providers	Copy vs. Reference	Proxy objects in OPC UA server referencing original model objects; using REST URIs to access controllers
	Digital Twin Content Format	Proprietary vs. Standard	Proprietary by default with the possibility to be transformed to a standard format on demand
	Model Provider Discovery	Automatic vs. Manual	Automatic via mDNS discovery mechanism, manual by configuration files
	Distribution	Centralized vs. Distributed	One Digital Twin Manager centralized instance
	Deployment	Device vs. Edge vs. Cloud	On the edge
	Construction and Destruction	Manual vs. Automatic	Automatic upon the appearance of the device on the network, also manual on user requests
	APIs to Applications	Standard vs. Proprietary	Standard OPC UA APIs, REST API
Application Layer	Deployment	Device vs. Edge vs. Cloud	On-premise

Fig. 3. Example alternatives for implementing the pattern

For the sake of extensibility, we adopt microservices to implement each model provider. We use the web-based API of the existing databases to access their information. The model provider that accesses the *Product Type Database* has two plugins for the MVC components because it provides two kinds of models: documentation and CAD drawing. Other model providers offer one type of model; hence, they have one MVC plugin.

The model provider that accesses the *Device Selection Tool* must read the output files of this tool. To access the files on the file system, we define a so-called *Digital Twin Monitored Folder*, where the files must be stored. The existing tool is extended with an add-in to store their results in this folder.

Since we adopt OPC UA as the technology to implement digital twins, the OPC UA drive can be seen as a *native* model provider: An information provider

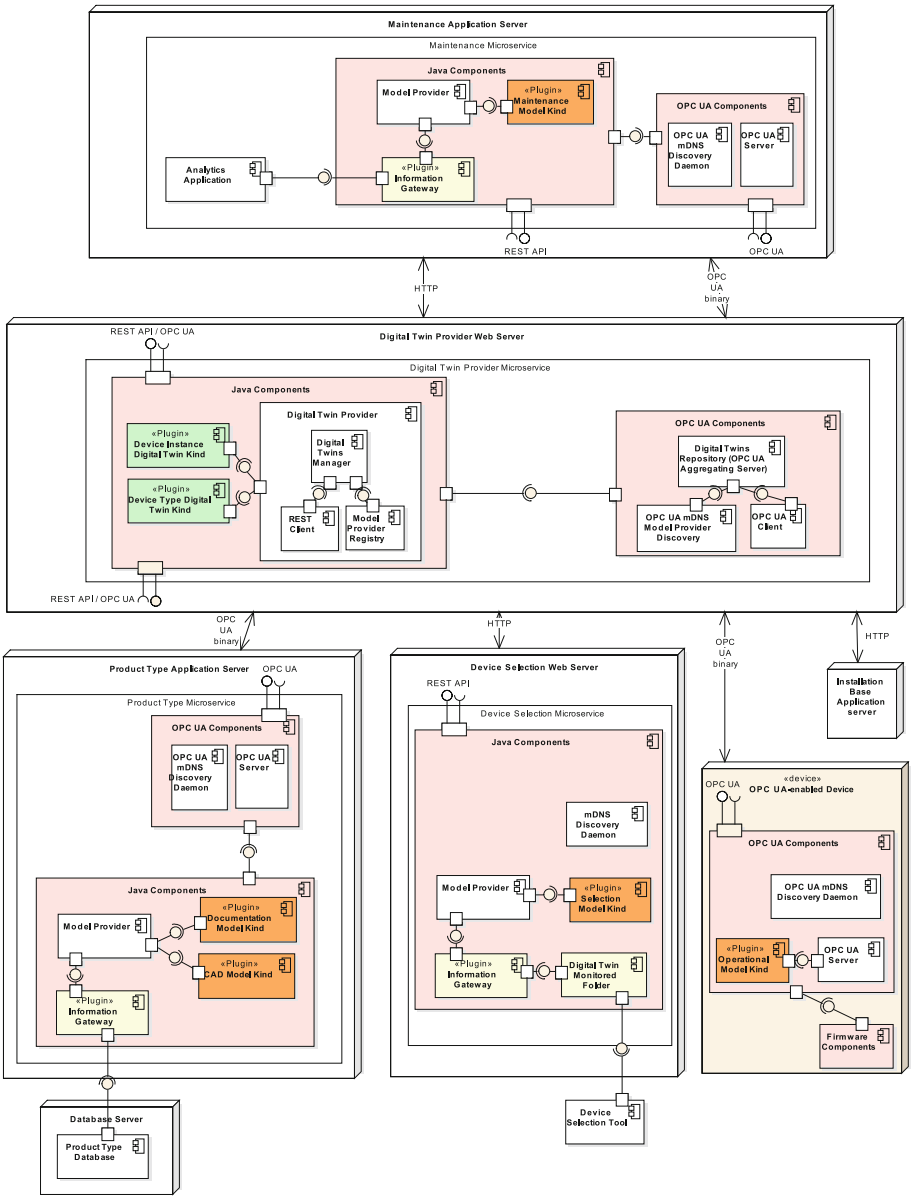

Fig. 4. An example concrete architecture

that provides the necessary MVC components, which can be directly integrated into a digital twin, is regarded in our architecture as a native model provider. In terms of the necessary MVC components, the embedded OPC UA server of the drive provides the operational parameters of the drive as an OPC UA

information model. There is a controller object associated with the model to enable access to the models. The HTML view description is included within the model and is passed to the upper layers to be rendered at the client-side.

The communication between the OPC UA server and digital twins uses OPC UA client/server sessions using the OPC UA binary protocol that is based on TCP/IP.

Different technologies can be adopted to define the APIs of other model providers to the *Digital Twin Provider* layer. We have experimented with OPC UA-based and REST APIs with JSON. The OPC UA-based APIs are used for the OPC UA-enabled drive; however, they can also be adopted by other model providers so that their models can easily be integrated with the corresponding digital twins using native OPC UA services. In this case, the model provider must include an OPC UA server in which models are maintained, and there is an OPC UA client at the *Digital Twin Provider* layer to communicate with this server. REST APIs with JSON representation of information can be adopted, where the JSON-based models are translated to OPC UA objects via digital twin controllers.

The content of models may be defined in a proprietary or standard format. Even if the content is defined in a proprietary format of the original information providers, the controller part of MVC may translate the content into a standard format (satisfies *R6*). The content of the models may be a copy—accompanied with a caching mechanism—of the original information in the underlying information providers, or a reference to it (satisfies *R5*). Which of these options is preferable depends on the desired performance, memory consumption, availability of information, and required frequency of information update. For example, if the original information/model provider becomes unavailable, it is still possible to work with the latest cached information in case information is copied/cached. In our use case, we adopt referencing for the OPC UA-based parameters of the underlying drive and copying for other information.

Our architecture is web-based, meaning that the view part of the MVC components is defined via a set of HTML files, and they are rendered at the client side, i.e. in the *Applications* layer of the architecture.

5.2 The Digital Twin Providers Layer

In the *Digital Twin Providers Layer*, there is only one instance of the *Digital Twin Manager* which can centrally manage multiple digital twins. There will be one digital twin instance for each device and one digital twin for each device type. Hence, there are two *Digital Twin Kind* plugins in our realization.

The choice of technology for storing digital twins depends on the scale of the system and desired non-functional requirements such as performance. We make use of a so-called OPC UA aggregating server to store digital twins, and to aggregate underlying models in digital twins. An OPC UA aggregating server is a special OPC UA server, which concentrates the information of underlying servers and may add more logics on top, e.g. to compute historic information.

In our implementation of the aggregating server, only a reference to the actual information in the underlying model providers is maintained, and actual information is retrieved from the underlying model providers on demand.

Since the underlying model providers can communicate via OPC UA or REST APIs, there are dedicated clients in the *Digital Twin Provider* to facilitate the communications. We adopt the multicast DNS (mDNS) [3] discovery mechanism to automatically discover the appearance/disappearance of microservices. Each model provider microservice makes use of its *mDNS Discovery Daemon* to announce its presence as well as its list of supported models to the *Model Provider Discovery* component. The microservices keep announcing themselves on a regular basis while they are alive. The *Model Provider Discovery* component will update the *Model Provider Registry* as soon as new microservices announce themselves or previously present microservices disappear. In an OPC UA realization, the disappearance of a microservice will be noted by a closed client connection, which could also trigger an update of the registry.

The *Digital Twin Manager* can interact with multiple model providers. The interactions can be push-based or pull-based (satisfies *R4*). In the latter case, the *Digital Twin Manager* requests models from the available model providers; in the former case, model providers proactively push the models upon their availability.

Digital twins may be constructed either manually upon user requests in the *Application* layer or automatically upon the availability of a specific model. The latter is implemented in our use case when a drive is installed in the network, its presence is detected via mDNS, and its embedded OPC UA information model is considered as the first model to be included in the digital twin of the drive. Afterwards, the corresponding digital twin controller pulls other models from the available model providers and adds them to the corresponding digital twin.

5.3 The Applications Layer

There is one microservice in this layer, which contains an analytics application. There are Java components that interact with the *Digital Twin Provider* microservice via REST APIs to receive necessary information from within the digital twin of a device and its type. This information is provided to a Python application for further analysis, and the results of the analysis are stored back in the so-called maintenance model. This means that, in addition to consuming digital twin content, the application is also a model provider for the digital twin. To be able to easily aggregate the maintenance model into the digital twin of the corresponding device, we have an OPC UA server within the *Analytics* microservice, which stores the maintenance model.

5.4 A Common Identification Mechanism

It is usually the case that each information provider adopts its own specific scheme to uniquely identify information pieces. For example, in our use case, within the *Product Type Database* there is a unique alphanumeric ID to designate

each type, but there are several aliases for one type ID; within the *Installed-base Database* there is a numeric identifier for each installation, all its contained devices, services performed on the devices, etc.; each manufactured device has a serial number and type information included, which follows a specific format. Since the information providers are usually developed and managed by separate organizations, it is not practical to enforce one unique way of identification on all of them. Instead, we need to define how the various identifiers can be mapped to each other.

Since in our use case we assume that there is one digital twin per device, we take the device serial number as the key identifier for the digital twin to which various identifiers must be mapped. Each information provider must be extended to provide a mapping between its internal identifier and the associated serial number, and this mapping must be communicated to the digital twin when a new model is provided to the *Digital Twin Providers* layer. This approach enables us to communicate with the underlying information provider using its own identifier and yet aggregate multiple models together within one digital twin.

Each model that should become a part of a digital twin must have dedicated fields defining its identifiers. Figure 5 shows the generic structure for defining identifiers in different models. Here, an abstract class *Model* contains identifiers that are mandatory for every model, since all models inherit from this class. The property *modelID* is used to define the unique identifier of the model itself. The property *modelKindID* is used to keep the identifier of the model kind, if any. If there is, for example, any standardized model kind description, a reference to that description could be stored in this property. The key identifier (serial number in our use case) and further optional local asset identifiers within each information provider can be stored in *assetIDs* collection.

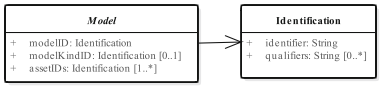

Fig. 5. Identifiers as part of any model

All these identifiers are defined in the class Identification, which has a property to keep the value of the identifier, accompanied with zero or more qualifiers in the qualifiers collection. The qualifiers provide additional information about the identifier, e.g., whether it is a serial number. This mechanism allows using different formats for various identifiers while preserving their semantics. The object-oriented definition of model identifiers presented in Fig. 5 allows a straightforward mapping to the selected implementation technology used by model providers, e.g., OPC UA or REST APIs with JSON.

6 Related Work

In [4] we discussed various architectural aspects of digital twins. The pattern proposed in this paper illustrates a concrete realization of these architectural aspects. Alam and Saddik describe C2PS [5], a "digital twin architecture reference model" for cloud-based cyber-physical systems. However, they focus on

network communication aspects and controller design. They do not address how to create a digital twin from the information modeling perspective, nor do they consider various information sources. Gabor et al. [6] present the definition of an architectural framework for digital twins as well. In contrast to our approach, their digital twin is mainly about the simulation of real asset, and their framework is limited to it. Likewise, Delbrügger et al. [7] focus on a digital twin that simulates a factory and introduce a navigation framework that aims to improve movement paths.

In [8], a service oriented application for knowledge navigation is presented. The architecture of the application linking different data sources is outlined briefly without mentioning the approach of how to link those data source together. In [9], a twin platform based on a data-centric middleware is defined, whose architecture mostly focuses on communication and data transfer between the physical assets and simulation and not the general management of digital twins.

General cloud providers also started supporting digital twins and include them as a service into their IoT solutions. For example, Microsoft's Azure Digital Twins platform[2] enables creating digital twins and populating them with data. However, all information models to be included must follow a specific data format; they do not yet provide the flexibility to extend the system with new information sources and different kinds of information models. In general, Azure Digital Twins comes with one concrete solution and does not provide many degrees of freedom regarding different architectural alternatives. Asset Administration Shell (AAS) is the digital twin for Industrie 4.0 systems [10]. Our pattern can be adopted to implement AAS by flexibly collecting information and generating the so-called submodels for AAS.

7 Conclusions and Future Work

In this paper, we proposed an architecture pattern to construct the digital twin of an asset considering its various information sources. The pattern is designed for flexible extension with new information sources and supports new kinds of proprietary or standard information. We outlined the alternatives to realize this pattern and described a microservices- and OPC UA-based example implementation, which was validated based on an industrial use case from ABB.

In the future, digital twin solutions have to take into account composite structures, e.g., devices that consist of multiple other devices, each having its own digital twin, leading to composite digital twins. Our architecture pattern needs to be validated with respect to such scenarios.

[2] https://azure.microsoft.com/en-us/services/digital-twins/.

References

1. OPC Foundation: IEC 62541–1: OPC Unified Architecture - Part 1: Overview and concepts (2016). https://webstore.iec.ch/publication/25997
2. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P.: Pattern-Oriented Software Architecture: A System of Patterns. Wiley, New York (1996)
3. Internet Engineering Task Force (IETF): Multicast DNS. http://www.ietf.org/rfc/rfc6762.txt
4. Malakuti, S., Grüner, S.: Architectural aspects of Digital Twins in IIoT systems. In: Proceedings of the 12th European Conference on Software Architecture (ECSA 2018): Companion Proceedings, p. 12. ACM (2018)
5. Alam, K.M., El Saddik, A.: C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems. IEEE Access **5**, 2050–2062 (2017)
6. Gabor, T., Belzner, L., Kiermeier, M., Beck, M.T., Neitz, A.: A simulation-based architecture for smart cyber-physical systems. In: IEEE International Conference on Autonomic Computing (ICAC), pp. 374–379. IEEE (2016)
7. Delbrügger, T., Lenz, L.T., Losch, D., Roßmann, J.: A navigation framework for digital twins of factories based on building information modeling. In: 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1–4. IEEE (2017)
8. Padovano, A., Longo, F., Nicoletti, L., Mirabelli, G.: A digital twin based service oriented application for a 4.0 knowledge navigation in the smart factory. IFAC-PapersOnLine **51**(11), 631–636 (2018)
9. Yun, S., Park, J., Kim, W.: Data-centric middleware based digital twin platform for dependable cyber-physical systems. In: 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN), July, pp. 922–926 (2017)
10. Plattform Industrie 4.0: Details of the Asset Administration Shell – Part 1: The exchange of information between partners in the value chain of Industrie 4.0 (2018). https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/2018-details-of-the-asset-administration-shell.html