applied sciences

MDPI

Article

Semantic Microservice Framework for Digital Twins

Gernot Steindl * and Wolfgang Kastner

Institute of Computer Engineering, TU Wien, 1040 Vienna, Austria; wolfgang.kastner@tuwien.ac.at
* Correspondence: gernot.steindl@tuwien.ac.at

Abstract: Digital Twins (DT) in industrial cyber-physical systems are the key enabling technology for Industry 4.0. Services are an essential part of almost every DT concept, but their interaction is usually implementation-specific since no common guidelines are available. This work identifies some fundamental requirements for a DT service framework based on applications identified in corresponding literature. Based on these requirements, a service framework architecture is proposed. The architecture utilizes Semantic Web technology and a workflow engine for service orchestration to support the fulfilment of the identified requirements. As a case study for sensor data evaluation of an industrial process, a proof-of-concept implementation is presented, showing the feasibility and suitability of the proposed DT service framework architecture.

Keywords: Digital Twin; microservice; workflow engine; service architecture; knowlege graph; cyber-physical system; Semantic Web

Citation: Steindl, G.; Kastner, W. Semantic Microservice Framework for Digital Twins. *Appl. Sci.* **2021**, *11*, 5633. https://doi.org/10.3390/app11125633

Academic Editor: Youngchul Bae

Received: 20 May 2021
Accepted: 15 June 2021
Published: 18 June 2021

1. Introduction

The fourth industrial revolution or Industry 4.0 is changing business within the industry, caused by the rapid development of Information and Communication Technology (ICT) [1]. One of the most promising technologies that enables us to achieve the vision of Industry 4.0 is the Digital Twin (DT) [2]. A DT can be defined as "a formal digital representation of some asset, process or system that captures attributes and behaviors of that entity suitable for communication, storage, interpretation or processing within a certain context" [3]. DTs can be used within Industrial Cyber-Physical Systems (ICPS) for monitoring, diagnostic, prediction, and control [4].

During the last few years, a number of research projects have addressed DT architectures and frameworks [5–10]. A very interesting concept is the Five-Dimensional Digital Twin (5D-DT) presented in [8], where five elements or dimensions for a DT are specified: "Physical Entity", "Virtual Entity", "Connection", "Data" and "Services". In this conceptual model, the aspect of services is emphasized as an important part of the DT. This 5D-DT approach is used in [10] as the basis for the Generic Digital Twin Architecture (GDTA), which is aligned with the six Information Technology (IT) layers of the Reference Architecture Model Industry 4.0 (RAMI 4.0) [11] to structure the elements of the GDTA model. However, services are only considered in a more general and abstract way in the GDTA.

In ICT, the microservice architectural style has become more relevant for building distributed software applications with improved scalability and maintainability in recent years [12]. The idea is to build service-based applications by composing small, loosely coupled software services [13]. The actual size of services depends on the application, but the attribute "small" targets their functionality rather than the lines of code. The service composition can be performed via orchestration or choreography. Orchestration is a centralized approach, whereas choreography follows a decentralized method. Sometimes, it even can be beneficial to apply both of those two concepts, e.g., in automation systems [14]. Another design decision that has to be made when utilizing microservices is inter-service communication. The communication can be established using an asynchronous or synchronous request–reply pattern or event-driven asynchronous message

exchange. In microservice architectures, event-driven communication is preferred as it supports decoupling of services from each other [15]. A common technology for synchronous request–reply communication is Hypertext Transfer Protocol (HTTP), which is often used in combination with Representational State Transfer (REST). For event-driven asynchronous communication, many message-oriented middleware solutions exist. One such solution that is widely used is Apache Kafka, which provides fault-tolerant, scalable, and stream-based messaging [16].

As described before, services play a key role in building DTs, but in most DT frameworks or architectures, they are only mentioned or described at a high level of abstraction. As most DT implementations are realized following a specific goal without any architectural template [7], the same holds for the implemented services. There are many design choices for how these services can be composed for later interaction. Their final design and implementation should be based on distinct requirements. Still, there is a significant gap in DT research, regarding how to offer a higher number of services in the same environment to support complex decision making [17]. Missing architectural guidelines are resulting in application-specific solutions which are barely reusable, and increase development time and costs. Thus, our research focuses on the services infrastructure of a DT and explicitly addresses requirements and a service framework architecture, which can later be applied for various DT applications.

The main contributions of our work are the specification of functional and non-functional requirements for a DT service framework derived from a literature review. These requirements were clustered based on three RAMI 4.0 IT layers (Information Layer, Functional Layer, and Business Layer) which are relevant for the proposed service framework. Resting upon the identified requirements, a novel microservice architecture for DTs is proposed. This architecture uses a federated knowledge graph to provide semantic interoperability between services and enables both choreography and orchestration by incorporating event-based messaging in combination with a workflow engine. A case study for a DT of a thermal heating process is used to investigate and evaluate the proposed service architecture. Therefore, microservices for an automatic sensor data evaluation were implemented and the design artifacts are checked against the identified requirements.

The remainder of the paper is structured as follows: Section 2 gives a short overview of related work in the area of DT service framework as well as DT architectures and applications which are used to derive requirements for a DT service framework from. Next, the identified requirements and the derived service framework are presented in Section 3. The proposed service framework is implemented as a proof-of-concept for a defined use-case in Section 4. In the end, the presented service framework and future research directions are discussed.

2. Related Work

A short overview of available DT service frameworks is given, and literature that is used to derive requirements for such a service framework is presented.

2.1. Digital Twin Service Frameworks

In this section, service framework architectures dedicated to DTs and related areas, such as Internet of Things (IoT) applications and smart manufacturing, are presented to find commonalities and shortcomings, which influence the design of our proposed DT service framework.

The design and implementation of a DT in smart manufacturing is presented in [18]. The authors investigated available open-source tools and technology for the implementation of the DT. In this context, they proposed a DT concept in alignment with the Industrial Internet Reference Architecture (IIRA) and the RAMI 4.0. They also introduced a microservice framework and defined 36 services, clustered in groups related to their components in their conceptual DT model, such as monitoring services, things and event management services, simulation management services, decision-making and control services. The

import role of semantic interoperability and a life cycle-oriented knowledge base of DTs is also conceptually addressed by their architecture but not fully explored in their prototype. A Service Manager component is responsible for the composition and orchestration of the services, but further details of how this is performed are not given.

A more concrete example of a service framework and the service interaction inside a DT is provided in [19]. The authors proposed a microservice approach in combination with an event-based architecture. They argue that a DT should follow an event-driven architecture style to parallelize processing and provide near real-time capabilities. Their solution to the problem is the use of Apache Kafka for state tracking, since most of the stream processing in DT implies stateful operations, but microservices should be stateless in principle. Thus, stateful stream processing can be supported in a DT. They carried out performance analysis and showed that Apache Kafka is suitable for managing the states with some restrictions. Such tracking of the current system state is relevant for stream data and human–machine interaction.

Similar to the previously presented architecture, a service-oriented and event-driven manufacturing information system architecture was proposed in [20]. The event-driven architecture is used to avoid point-to-point device and service integration and ensures loose coupling of the services. Apache ActiveMQ [21] is used as Enterprise Service Bus (ESB). The presented use-case is targeting discrete manufacturing, in which the authors showed the integration of devices and services on all hierarchy levels. The service composition in this system is performed using choreography to avoid a central orchestrator. The benefit they explored using their microservices in combination with event-based communication is that services can be developed and tested in isolation, as mock-ups may serve as temporary replacement for other applications.

A microservice architecture is also used in [22]. Rather than targeting a DT architecture, the authors present a framework for predictive analytics of IoT applications. However, some concepts are similar and helpful in the context of DTs. The authors used a containerized microservice architecture to build the data pipeline. Their implementation is based on Docker [23] and Apache Kafka. A central orchestrator is used to combine multiple operations provided by microservices. Additionally, data modeling is used, based on the Web of Objects framework [24] to achieve semantic interoperability.

In most of the presented frameworks, microservices combined with event-based messaging are used, as they can provide benefits regarding separation of concerns, flexibility in choosing technology, scalability, etc. However, their underlying requirements are mostly not stated. Therefore, the next section analyzes DT frameworks and applications to identify some fundamental requirements regarding a DT framework.

2.2. Requirements for a Digital Twin Service Framework

Requirements found in the literature primarily target the overall concept of a DT, rather than only focusing on the service infrastructure. Nevertheless, some general requirements are also relevant for the proposed service framework. Here, the main literature is presented, which is used to derive the requirements presented in Section 3.1.

A requirements-driven DT framework for smart manufacturing or Industry 4.0 is presented in [25]. Some requirements mentioned there are also specifically relevant for a service framework, e.g., a DT's capability should be modular with clear boundaries (RN2). Services should also provide some form of narrow intelligence to solve some special tasks in the application domain (RI3). Furthermore, expertise should be incorporated into services to realize intelligence in solutions (RI4). Another aspect is the interoperability with DT clients, which must be realized by providing an appropriate service interface (RI5). DT should also be extensible, which means incorporating data or services from outside (RN4). An important aspect is that a DT framework must support an evolution rather than a revolution of capabilities, helping us to introduce and further develop the DT over its whole life-cycle (RN3). New services are supporting the evolution of existing capabilities.

As DT should provide an added value over lifetime, the DT services have to be integrated into the business process at the enterprise level in some way (RB1).

In [26], a method for DT-driven product design, manufacturing, and service is introduced. In this context, nine service categories are presented. Some of these service categories demand some requirements regarding the service communication infrastructure. For instance, a service for real-time state monitoring requires a certain data acquisition infrastructure (RF1) and certain communication patterns (RF2). The same holds for a service within the category of energy consumption analysis and forecast, such as processing heterogeneous data from various sources (RI1, RF3), dealing with historical data (RI2) and supporting request–response communication with other services (RF4). These two examples show that a service framework has to be able to support different communication patterns.

A DT architecture reference model for the cloud-based CPS (C2PS) is presented in [27], in which the key properties of computation, control, and communication are described. The cloud-based approach seems reasonable, based on requirements such as computational power and scalability. On the other hand, this can cause a problem, as the communication over the Internet is critical, regarding availability [28] and the communication delay [29] for real-time control applications. Thus, a combination of an edge, fog, and cloud-based approach is feasible for a DT to provide services with their needed resources, as presented in [30]. However, in some use-cases with a high demand for reliability, security, and privacy, the hosting of the DT has to be done on-premise (RN1). To enable this flexibility, a containerized solution for DT services, as proposed in [31], seems to be suitable.

Human–machine interaction is also essential in the context of DTs, which includes social and technical aspects [32]. Maintenance scenarios are one example of such a human–machine interaction. These scenarios can be quite complex, and their states have to be stored during this process, as shown in [33] for gas turbine maintenance. Thus, the state must also be stored at the involved services in such a human–machine interaction (RB2).

Services of a DT often have to process a large amount of data. Such data-intensive applications have some basic non-functional requirements which should be met, such as reliability (RN5), scalability (RN3), and maintainability (RN2, RN4) [34]. Unfortunately, there are no easy solutions to meet these requirements, but specific architectural patterns and techniques can be applied during implementation to fulfil them.

3. Microservice Framework Architecture

In this section, the identified functional and non-functional requirements are presented, which are used to design the proposed DT service framework architecture. A detailed description of this service framework architecture can be found in Section 3.2.

3.1. Identified Requirements

Based on the literature review, some essential fundamental requirements are identified to implement a DT service framework. They are grouped into functional and non-functional requirements. The functional requirements are again grouped by the IT layers of the RAMI 4.0. The list is not comprehensive but gives a solid base for a service framework architecture and its implementation.

3.1.1. Non-Functional Requirements

Table 1 shows the identified non-functional requirements. The possibility to host the services on-premise is, in some use-cases, a prerequisite because of data ownership issues or response time (RN1). The other requirements are mainly concerned with reliability (RN5), scalability (RN3), and maintainability (RN2, RN4), which are relatively common for data-intensive systems.

Table 1. Non-functional requirements.

ID	Requirement	Origin
RN1	The DT and its services should be able to be hosted at the cloud as well as on-premises for data ownership and performance reasons.	[27,31]
RN2	Services of a DT should be loosely coupled to add or remove new services without influencing each other.	[25,26]
RN3	Services of a DT should be scalable to handle requests from a single machine up to a whole factory.	[25,34]
RN4	Services of a DT should be maintainable by different development teams (third party integration).	[34]
RN5	The service infrastructure of a DT should tolerate short down times of single services to increase the reliability.	[34]

3.1.2. Functional Requirements

The following functional requirements are clustered by the Information Layer, Functional Layer, and Business Layer defined in RAMI 4.0.

The requirements for the Information Layer are shown in Table 2. They are mainly concerned with interoperability issues, e.g., how information is provided to other services (RI3, RI4) and exchanged between services and other systems (RI5). Furthermore, how heterogeneous data can be integrated (RI1) and how these data can be interlinked with context information (RI2) to provide further semantics is targeted.

Table 2. Functional requirements for the Information Layer.

ID	Requirement	Origin
RI1	The DT should be able to process heterogeneous data from different sources.	[26]
RI2	The DT should be able to interlink time series data with context information, to make it interpretable for other services.	[26]
RI3	Services of a DT should have control about the information they provide to other services.	[25]
RI4	The DT should have a service which provides access to the information provided by all services of the DT.	[10,25]
RI5	Services of the DT should exchange information in a semantically meaningful way.	[25]

Requirements for the Functional Layer are shown in Table 3. They define how the continuous stream of data (e.g., sensor data) is handled (RF1) and which communication patterns between the services are needed (RF2, RF3, RF4). The communication patterns are manifold: providing sensor data to many services needs an event-based 1:n communication; a monitoring service receiving data from many sources needs an n:1 communication; prediction services would most likely need a request–reply communication pattern.

The requirements for the Business Layer are shown in Table 4. The integration of the DT capabilities and services into the business processes at the enterprise level is essential to support the value-added chain (RB1). Furthermore, human–machine interaction requires holding the state for a certain period (RB2), e.g., a maintenance assignment to a service technician triggered by a predictive maintenance service of the DT. The state of the involved services has to be stored till the service technician confirms some action.

Table 3. Functional requirements for the Functional Layer.

ID	Requirement	Origin
RF1	Services shall be able to access a continuous stream of (sensor) data to monitor the system in real-time.	[26]
RF2	Data streams should be accessible by multiple services simultaneously to process it in parallel and reduce reaction time of the system.	[26]
RF3	Services of the DT should be able to receive data streams from multiple sources at the same time to fuse and process data.	[26]
RF4	Services of the DT should be able to respond to a specific service request to enable a one-to-one communication for information retrieval.	[26]

Table 4. functional requirements for the Business Layer.

ID	Requirement	Origin
RB1	The functional services of the DT should be able to be integrated into the business processes at enterprise level to support the value-added chain.	[25]
RB2	Service interaction states should be traceable to facilitate human–machine interaction and the identification of service faults.	[32]

3.2. Proposed Service Framework Architecture

Based on the identified requirements presented in Section 3.1, design decisions for the service framework architecture are made and explained in this section. The principal architecture of the proposed service framework for a DT is shown in Figure 1. It is based on the concepts of the GDTA presented in [10], which is also aligned with the RAMI 4.0 IT layers. Relevant for the proposed service framework are the Information Layer, the Functional Layer, and the Business Layer, which are also depicted in Figure 1.

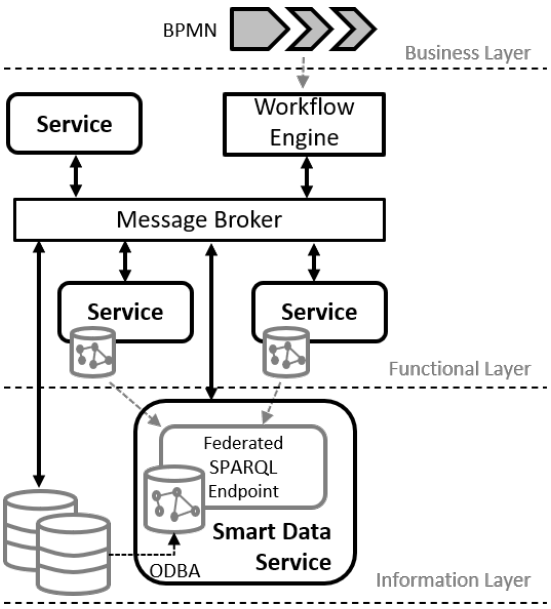

Figure 1. Proposed micorservice framework architecture, in alignment with the RAMI4.0 IT-layers.

One of the main components present in the GDTA is the shared knowledge as part of the Smart Data Service on the Information Layer. This Smart Data Service provides contextual information about data and resources, which can be facilitated by the services of the DT. Its shared knowledge can be realized using Semantic Web technologies to build a so-called knowledge graph. A knowledge graph is a knowledge-based system, which consists of an ontology and reasoner, but also is capable of integrating external

information sources [35]. In this context, an ontology means a formal, explicit specification of a shared conceptualization [36]. Ontologies, based on Web Ontology Language (OWL) can provide the semantics for the data processed by the DT and can be used to enable automatic reasoning if needed. The data can be accessed via SPARQL Protocol and RDF Query Language (SPARQL) endpoints. A SPARQL endpoint is a web service that is capable of receiving and processing SPARQL protocol requests using HTTP. Fundamental information about the physical entity of the DT, such as plant equipment, topology, and the instrumentation, is directly stored in a triplestore, which is managed by the Smart Data Service itself.

Services can add relevant information to the knowledge graph by facilitating a federated SPARQL query engine, within the Smart Data Service. A federated query engine is able to integrate distributed data sources virtually. That means a query is sent to several SPARQL endpoints, and the results are merged. This process is fully transparent for the user, as it seems that only one triplestore is queried [37]. Services can include graphs from their local triplestore that are managed by the services themselves. Thus, information can stay private or can be included into the shared knowledge.

To provide access to historical data and add context and semantics to it, Ontology-Based Data Access (OBDA) is used. Data from a relational database is mapped to ontological concepts and can be accessed through the knowledge graph. The loading from the data is only performed when the data is accessed. Thus, the data stay in their original database and do not have to be copied into the shared knowledge graph, which usually enhances the data access performance [38].

With the help of the presented concepts for building a Smart Data Service, existing and also newly created ontologies can be used within a DT to provide a description in terms of classes, properties, and their interrelation for a specific domain (TBox). Various services can instantiate the individuals (ABox). As they can refer to TBox concepts, clear semantics to the data is provided. This approach has the advantage that data integration can be performed with less effort, as the knowledge graph acts as an abstract semantic integration layer [39]. Thus, data from relational databases and also from other sources, e.g., OPC Unified Architecture (OPC UA) can be included in the knowledge graph and made accessible for all services. Furthermore, the interoperability between services is enhanced, as the exchanged data can be referred to concepts defined in the TBox, providing precise semantics to the data. This is additionally supported by formats such as JSON for Linking Data (JSON-LD), which allow stating such references. Another advantage of using a knowledge graph based on Semantic Web technology is the support for knowledge discovery. Data from various services can be connected, and new insights into the DT can be gained.

The Functional Layer contains the actual services of the DT. Thus, the requirements targeting inter-service communication are relevant, but also the non-functional requirements have influence on the design.

A microservice architecture facilitates maintainability because services can be realized and deployed by independent development teams. Another advantage of microservices is that they can be containerized. Thus, they can be deployed and orchestrated in a virtualized environment hosted in the cloud or on-premise if needed.

For the inter-service communication, a Message-oriented Middleware (MOM) with a message broker is used. This MOM allows the realization of various communication patterns, such as a 1:n or an n:1 communication, which are useful for building data pipelines. A typical request–reply communication is also often required for certain services, which can be implemented on top of such an MOM. The use of an MOM can help to decouple services and support reliability and scalability by operating a cluster of brokers. Suppose the broker is able to store messages; the reliability can be further increased because, if a service is not available for a short period, it can retrieve the message from the broker once it is reconnected.

Service composition is fundamental at the Business Layer to provide certain functionality of the DT. This composition can be implemented by choreography or by orchestration. In some cases, choreography might have advantages because it is a more decentralized approach. However, if states have to be handled, e.g., for error handling or user interaction, this can be laborious [40]. Orchestration, on the other hand, uses a central component, where the composition logic is located. This can lead to a tight coupling of services and integrating service logic into the orchestrator [15]. The workflows in which a DT service is involved can be located on the enterprise or production level [41]. At the enterprise level, typically Business Process Model and Notation (BPMN) is used to describe these workflows, and sometimes workflow engines are used to automate them. Such workflow engines can also be used for microservice orchestration [42]. Using orchestration combined with a workflow engine to manage the flow between various microservices can help to visualize these flows and handle long-lived transactions. As workflow engines support BPMN, this notation can be used to communicate with a non-software developer and seamlessly integrate the DT capabilities into existing business processes on the enterprise level. In Figure 1, only a central workflow engine is depicted, but it is also possible to use multiple decentralized engines.

4. Proof-of-Concept: Automatic Sensor Data Evaluation

For evaluating the proposed service architecture, a proof-of-concept for a DT of a thermal heating process is implemented. The service interaction is demonstrated with a composite sensor data evaluation service, which analyzes sensor data from the plant and detects anomalies. The anomaly is classified as a sensor fault or abnormal behavior of the plant caused by a malfunction of the equipment.

For the presented use-case, three different services are implemented, which are orchestrated by the workflow engine Zeebe [43], as shown in Figure 2. The communication between the services takes place over the stream-based MOM Apache Kafka [16]. Every service holds its own database or triple store, which are connected by the federated query engine FedX [44] to build a distributed knowledge graph as a shared knowledge base for the services. The information exchange between the services is based on JSON-LD. This format is used because it provides semantics by referencing specific contexts with Uniform Resource Identifier (URI). Thus, ontological concepts defined in the shared knowledge graph can be used to unambiguously specify the meaning of the data, which facilitates interoperability.

The services are implemented in Python 3.9.1 and use several libraries to communicate with Zeebe, Kafka, the triplestores, or the databases. Every service and its infrastructure (database, triplestore, etc.) are virtualized in Docker containers.

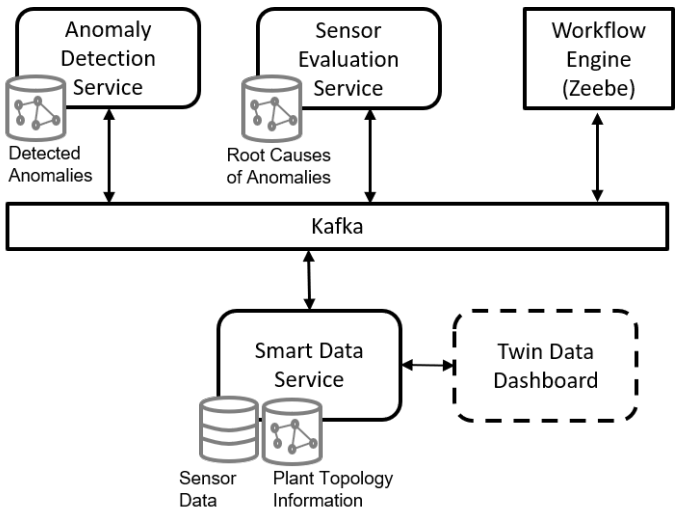

Figure 2. Overview about implemented services for sensor data evaluation.

After introducing the use-case, the implemented services are explained in more detail in the following subsections. The source code and a Docker-compose file to set up the service infrastructure can be found on GitHub [45].

4.1. Use-Case: Thermal Heating Process

As depicted in the pipe- and instrumentation diagram in Figure 3, a simple thermal heating process is used to evaluate the proposed service architecture. The ventilation unit "F1" blows ambient air through an electric heater "H1" into a vessel called "SiPro" where a thermal process takes place. The temperature of the process is controlled by modifying the power of the heater. The air is retrieved from the vessel by the ventilation unit "F2". The heat exchanger "HE1" is used for heat recovery of the exhaust air. Five temperature and two mass flow sensors are placed in the supply and return ducts. The ambient temperature surrounding the vessel causes heat loss of the vessel. The temperature after the heat exchanger T_{hx}, the supply temperature T_{sup}, and the process temperature T_p are measured by sensors with the IDs 102, 105, 106.1. The mass flow into the process is measured by the sensor m_{in}. Those are the most relevant data points for the presented sensor evaluation showcase.

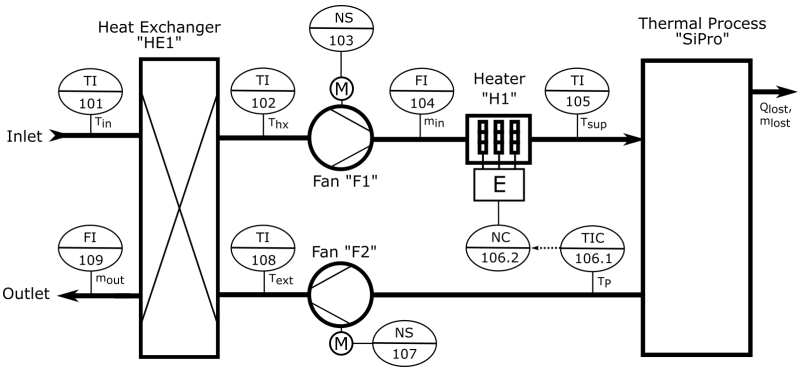

Figure 3. Use-case: Pipe and instrumentation diagram of the heating process.

The data for the service evaluation is generated by a simulation, and implemented in Open Modelica [46]. The simulation was used to produce data for a sensor fault as well as abnormal behavior of the plant caused by a clogged duct. During a regular operation, the temperature inside the process T_p is controlled and is following the setpoint trajectory shown in Figure 4. As shown, there are three operating points for T_p: 50 °C, 70 °C, and a standby mode, in which the temperature is held at 30 °C. During standby, the pressure setpoint of the ventilation unit "F1" is reduced. Thus, the mass flow m_{in} is also decreased.

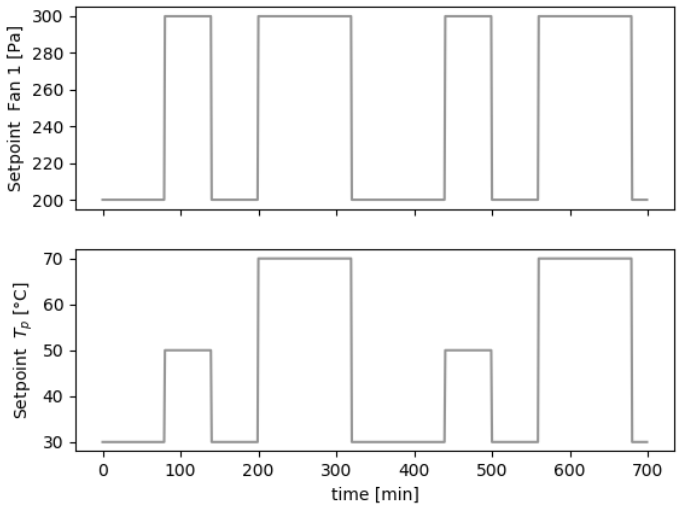

Figure 4. Setpoint for the ventilation unit "Fan 1" and the process temperature T_p.

To emulate the behaviour of real sensors, random noise is added to the simulated values, which is modeled as a normal distribution with zero mean and a standard deviation of 0.2 °C and 0.01 kg/s, respectively. The Open Modelica simulation model is available at the GitHub repository [47].

4.2. Smart Data Service and Communication Infrastructure

Figure 5 shows more details of the Smart Data Service as well as the needed communication infrastructure. The separated processes that build the Smart Data Service, their encapsulation into Docker containers, and the communication between them are shown.

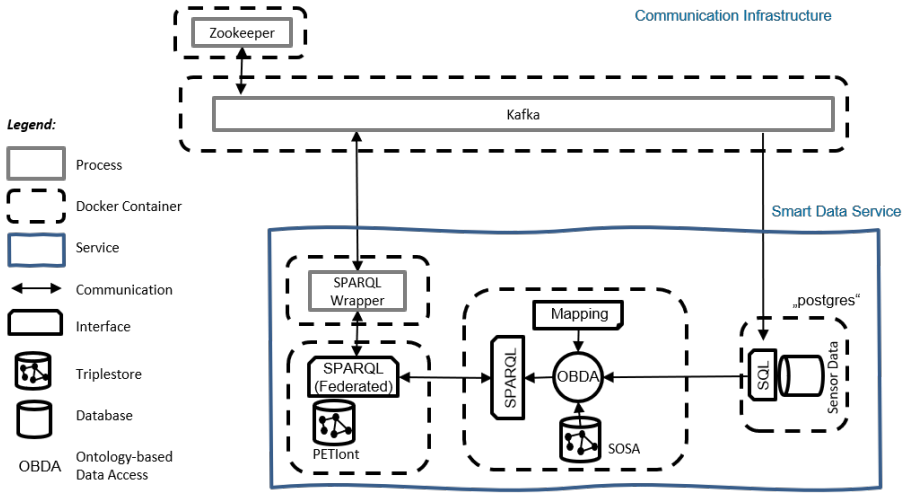

Figure 5. Smart Data Service and Communication Infrastructure.

The Smart Data Service provides access to the knowledge graph formed by interlinking various SPARQL endpoints. Essential plant information, such as equipment and topology information, is directly stored at the Smart Data Service. For the presented proof-of-concept, the Plant, Equipment, Topology and Instrumentation Ontology (PETIont) [48] was used to describe the heating process use-case. Information provided by other services is stored and managed by the services themselves but included in the knowledge graph by the federated SPARQL engine FedX. Thus, this information is interlinked and accessible through the Smart Data Service.

The time-series data from the sensors are stored in a PostgreSQL [49] database. To make these data accessible through the Smart Data Service and interlink them with context information, OBDA is utilized. The Ontop Framework [50] is used to map the data into the Sensor, Observation, Sample, and Actuator (SOSA) ontology [51]. Therefore, mappings are defined to populate the SOSA ontology with individuals based on the data stored in the PostgreSQL database. As an advantage of this approach, the data remain in the relational database and are only loaded in a virtual knowledge graph if needed by a SPARQL query request.

The communication between the services is handled by the stream-based MOM Apache Kafka. Kafka is designed as a distributed system, running on a server cluster. Apache Zookeeper provides a centralized service to manage the nodes inside this cluster. Kafka topics are used to send data to services inside the DT. The request–reply communication pattern is implemented on top of the Kafka infrastructure. The requesting service can add a reply topic to its request, to which the reply will be sent. Such a pattern is implemented to communicate with the Smart Data Service.

As the Smart Data Service communicates with SPARQL internally, a wrapper is implemented to enable requests from other services over Kafka. Services can send queries to the Kafka topic, to which the Smart Data Service is listing to. The answer is sent to the reply-topic specified in the request formatted in JSON-LD

4.3. Anomaly Detection Service

The Anomaly Detection Service is used to find deviation in the data of single sensors. Therefore, a data-driven approach was chosen. A linear autoregressive with exogenous input (ARX) model is trained based on data which is provided by the Smart Data Service. Details about how these models can be derived and identified based on the information stored in the knowledge graph can be found in [48].

The service infrastructure is shown in Figure 6. The "Data-driven Anomaly Detection Process" listens to a specified Kafka topic and starts the anomaly detection if a request arrives. The trained ARX models are executed in a serial–parallel fashion, which means the models are used to predict certain time-steps, and the results are compared with the actual measurements. If the error is larger than a certain threshold, which is three times the standard deviation of the model error, the time is marked as an anomaly. Then, the prediction window is shifted by one time step and actual measured values are used as input for the next prediction.

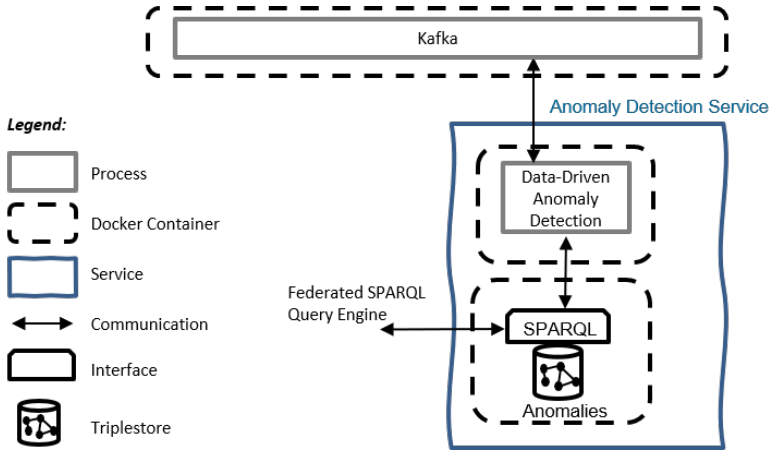

Figure 6. Anomaly Detection Service.

If an anomaly for a sensor value is detected, OWL-Time [52] is used to store that information in the knowledge graph. As depicted in Figure 7, an anomaly is defined in PETIont and described as OWL-Time "time:Interval". A relation is set between the sensor and the anomaly with the property "peti:hasAnomaly". This information is stored in the local triplestore of the service. As the triplestore of this service is connected to the federated query engine, the information is also instantly accessible through the Smart Data Service.

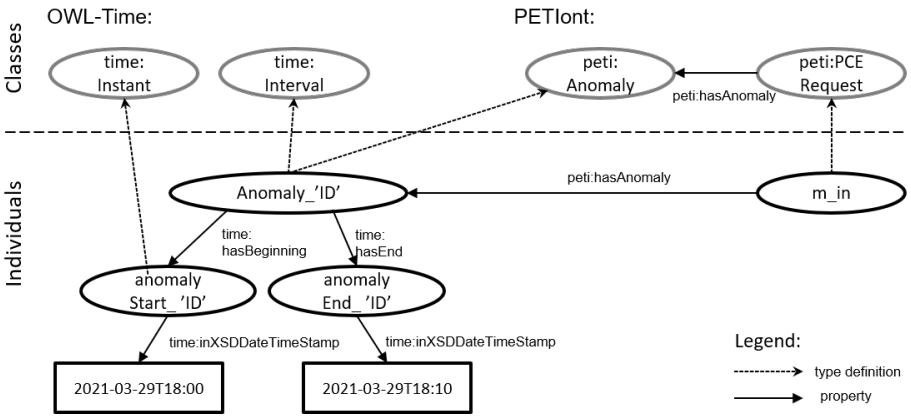

Figure 7. Anomaly modeled inside the knowledge graph.

4.4. Sensor Evaluation Service

The Sensor Evaluation Service receives detected anomalies and classifies these anomalies as a sensor fault or abnormal behavior of the plant. Therefore, the service has infor-

mation about the causal relations between sensors and actuators stored in its triplestore, connected to the federated query engine (Figure 8). A detailed description of how such relations can be automatically derived based on topology and equipment information stored in the knowledge graph can be found in [48].

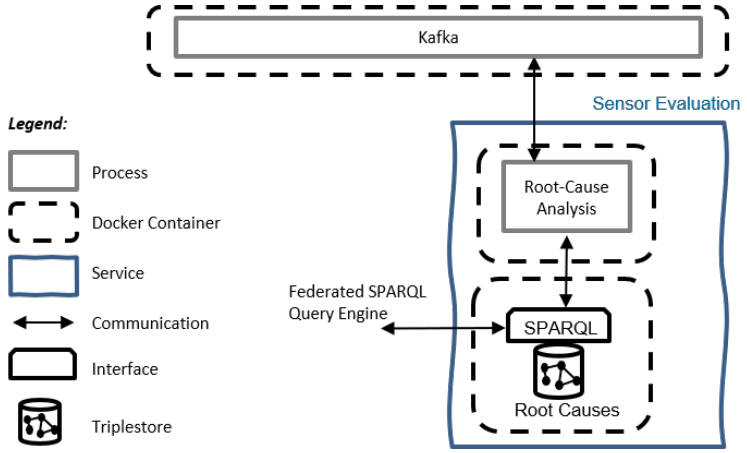

Figure 8. Smart Data Service and Communication Infrastructure.

The first step the service performs is clustering the anomalies based on their timely occurrence. For a cluster of anomalies, the root sensor is detected based on the causal relations retrieved from the knowledge graph. For every cluster of anomalies, a "peti:Incident" is created in the knowledge graph related to the anomalies in the cluster via the "peti:hasRelatedAnomaly" property. Implicit redundant sensors are searched, starting at the identified root sensor. The implicit redundant sensors are checked if they are also detected with an anomaly. If so, a simple majority voting is performed to decide if it is a single sensor fault or an abnormal behavior of the plant is detected. The classification is performed by specifying the "peti:Incident" as a subclass of "peti:AbnormalBehavior" or "peti:SensorFault", as shown in Figure 9. A sensor fault is also related to the identified faulty sensor, which is a subtype of "peti:PCE-Request" in PETIont. The result is stored at the local triplestore, which is connected to the federated query engine. Thus, this information is also accessible by the Smart Data Service and also sent to a Kafka reply-topic encoded as JSON-LD.

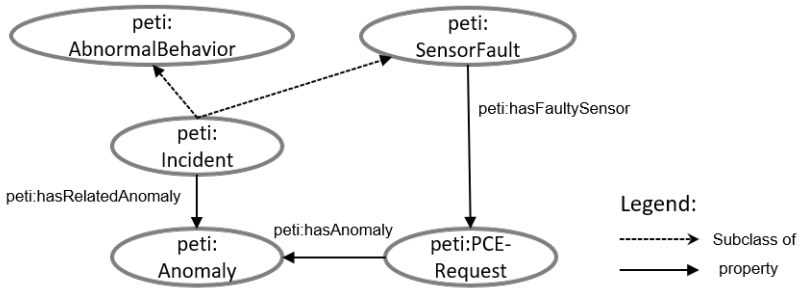

Figure 9. Incident classification.

4.5. Service Orchestration

The microservice orchestration is performed using the open-source workflow engine Zeebe. This workflow engine allows defining the processes visually in BPMN version 2.0. The communication between Zeebe and the services is based on gRPC Remote Procedure Calls. It is also possible to react to messages from Kafka or other MOMs. For the presented use-case, Kafka Connect is used to establish the connection between Zeebe and the services, as shown in Figure 10. Elasticsearch [53] is used by the Zebee workflow engine to store the execution states of the workflows with their internal messages. The Zebee Operate tool is for user interaction, such as visualizing the workflow state of the current execution.

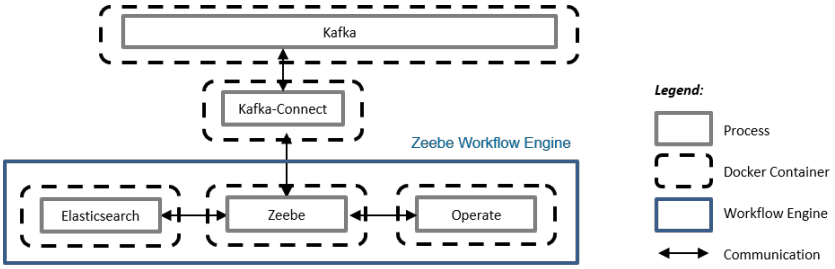

Figure 10. Zeebe workflow engine infrastructure.

The workflow in BPMN for the sensor evaluation of the presented use-case is depicted in Figure 11. The first "service task" is sending a Kafka message to the Anomaly Detection Service, described in Section 4.3. Message parameters, such as a correlation ID, or a reply topic, are defined. The "message catch event" is used afterwards to wait for the response of the service. The results of the Anomaly Detection Service are encapsulated in the Zebee message. If no anomalies are found, the process is finished. Otherwise, the results are handed over to the Sensor Evaluation Service, described in Section 4.4. Again, the catch message event is used to receive the response from Kafka. Depending on the result, a faulty behavior is logged for further analysis, or the maintenance of the faulty sensor is commissioned. Therefore, a service notification can be sent to a service technician. The workflow engine holds the state for maintenance till the service technician confirms the replacement of the sensor.

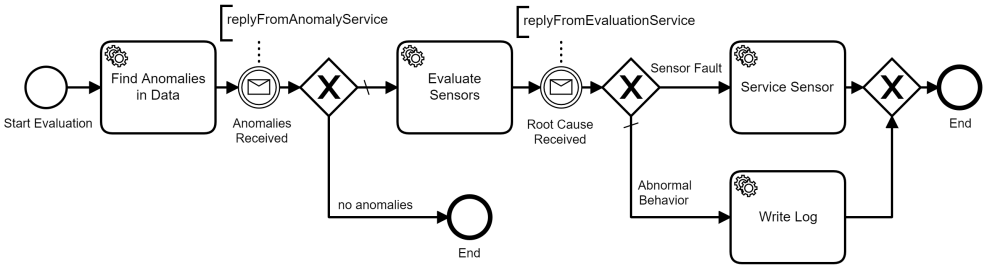

Figure 11. Service orchestration for sensor data evaluation.

For the implementation of the logging of abnormal behavior, a simple Zeebe Worker is used. The same applies to the Sensor Service task. The confirmation by a service technician is emulated via the command-line interface of Zeebe.

4.6. Results of the Sensor Evaluation Process

Two scenarios, "A" and "B", are introduced for testing the sensor evaluation service. In both scenarios the mass flow, which is measured by the sensor m_{in}, shows an anomaly. The anomalies of m_{in} are depicted in Figure 12 and marked with "A" and "B".

The first scenario ("A") is caused by a clogged duct, which results in a reduced mass flow. Therefore, an additional flow resistor is used in the simulation, which is rapidly increased 220 min after the start. As the fan has no closed-loop control, the mass flow decreases rapidly. After 17 min, the resistance is removed so that the mass flow can increase to its normal value again.

The second scenario ("B") is caused by a faulty mass flow sensor m_{in}, which delivers the wrong values. To make things more difficult for the sensor evaluation service and show its capability, the faulty sensor values for the mass flow sensor m_{in} are exactly the same as for the anomaly "A", but in that case, the mass flow is not reduced in the simulation. Thus, the sensor evaluation service has to use context information to classify the occurred anomalies correctly. The faulty sensor data starts about 600 min after the start.

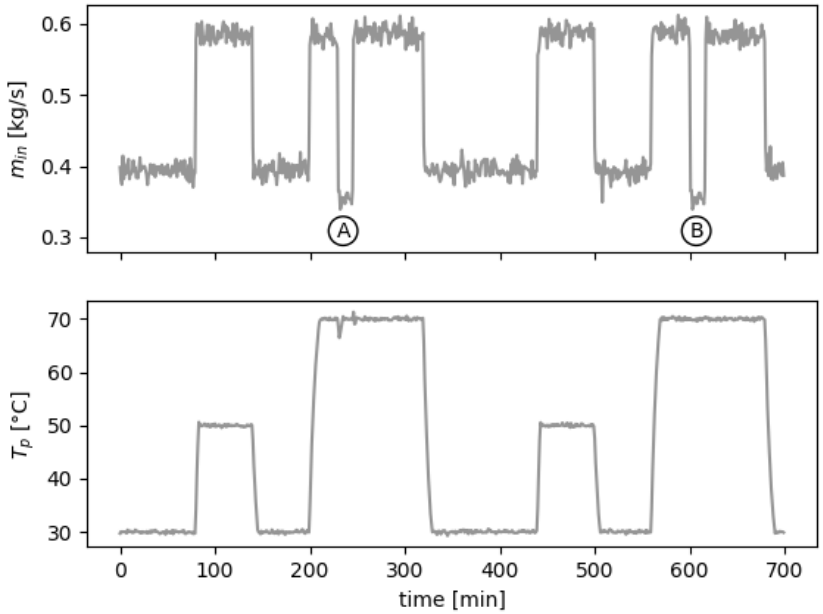

Figure 12. Scenarios: sensor fault ("A") and clogged duct ("B").

The workflow shown in Figure 10 is invoked with a unique correlation ID, to assign messages to the workflow instance. With that correlation ID, the Data Anomaly Detection service is started. The deviations between the internal ARX model results and the measured values are visualized in Figure 13. The periods in time where the deviation is exceeding the threshold are marked as red area. As the internal models are executed in a serial–parallel manner, the unexpected values are propagating through the various models, as shown for the anomaly "B". The simulated values are used as input values within the simulation window. Thus, a clogged duct, which results in a reduced mass flow, shown by the discrepancy between the measured and simulated values of m_{in}, has also influence on the simulated values of T_{hx}, T_{sup}, and T_p during the parallel operation.

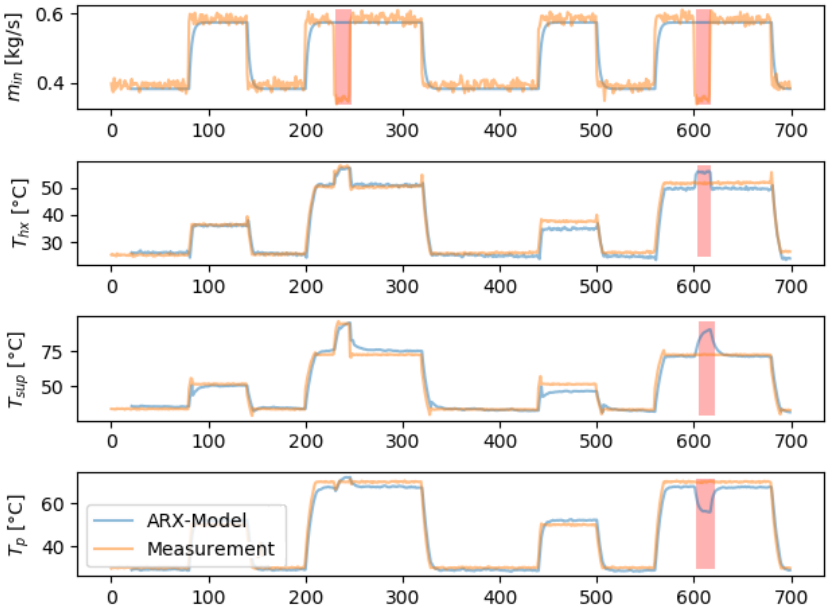

Figure 13. Anomalies detected by the Sensor Anomaly Detection Service.

The detected anomalies are encoded in JSON-LD, based on the parts of PETIont and OWL-Time, as shown in Figure 7, and sent as response. The response is handed over to

the Sensor Evaluation Service by the workflow engine. The service analyses the detected anomalies based on causal relations derived from the topology information. The service clusters all identified anomalies into two groups, based on their timely occurrence.

The first cluster or incidence contains only one anomaly (Scenario A). The other sensor values which are influenced by m_{in} are in line with the simulated output of the ARX models. Thus, the measurements are correct and the incident is classified as abnormal behavior of the plant, causing a reduced mass flow m_{in}.

For scenario B, a deviation between the measurement and the simulation is visible also at sensors which are influenced by m_{in} (Figure 13). This means there is a discrepancy between the measurements and the simulated ARX model. The causality information from the knowledge graph is used to identify the root cause of the deviations—in this case, a faulty sensor value of m_{in}. A simplified visualization of the causal relations is shown in Figure 14. Based on this information, the root sensor m_{in} is identified and the incident is classified as a sensor fault.

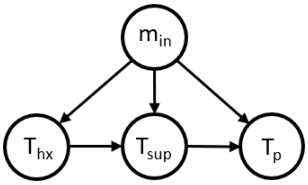

Figure 14. Causal relations between sensors.

The identified incidences are classified based on the parts of PETIont, which is shown in Figure 9. In the presented use-case scenario, "A" is correctly classified as abnormal behavior and "B" as sensor fault. The result is encoded as JSON-LD and returned as a reply to the workflow engine over Kafka.

As mentioned before, the other services for logging and interaction with a service technician are only implemented as mock-up.

5. Discussion

The proposed DT service framework architecture was evaluated with the proof-of-concept implementation for the presented use-case. The service interaction combined with the shared and federated knowledge graph and the service orchestration with the workflow engine has been shown.

The presented functional and non-functional requirements for a DT service framework do not form a comprehensive list, but they can be used as a starting point for the implementation. In particular, non-functional requirements depend on the concrete implementation and cannot be fulfilled by a general architecture. However, the architecture can support their achievement. An essential non-functional requirement was not targeted by the proposed architecture, as it has to be considered inherently—security. Nevertheless, security has to be considered in DT applications during the whole life-cycle.

The proposed DT service framework architecture can be extended for particular use-cases. The proof-of-concept implementation for the sensor data evaluation for a DT used open-source tools to fulfill the identified requirements. Table 5 shows design artifacts and the requirements which were supported to be met by the specific artifact.

The microservice architectural style supports the maintainability of the whole framework (RN2, RN4), as they can be extended and developed by individual teams. Furthermore, microservice can be easily containerized, which facilitates the hosting in a cloud infrastructure as well as on-premise (RN1).

Inter-service communication is a crucial part of the whole service framework. Using a MOM with event-based messaging is also beneficial for decoupling microservices (RN2) as well as enable 1:n and n:1 communication (RF2, RF3). Apache Kafka, used in the proof-of-concept implementation, further supports the data streams (RF1), which are stored persistently in Kafka (RN5). Thus, connecting services can also access previously sent

data. Kafka topics are used to enable request–reply communication between services (RF4). Furthermore, it supports scalability by running multiple brokers on a server cluster (RN3).

Table 5. Requirements supported by design artifacts.

Design Artifact	Supported Requirements
Microservice Architecture	RN2, RN4
Containerization	RN2, RN4
MOM (Apache Kafka)	RN2, RN3, RN5, RF1, RF2, RF3, RF4
Shared Knowledge graph	RI1, RI5
Federated Query Engine	RI4, RI3
OBDA	RI2
Workflow Engine	RB1, RB2

Semantic Web technology can be utilized to perform data integration (RI1) and build up the shared knowledge graph. Based on the ontologies used in the knowledge graph, data can be exchanged between services and external clients using JSON-LD (RI5). Using standard ontological models is vital to provide interoperability. A hierarchical structure of the used ontology, with an upper, a domain, and task ontology, can help to provide a common understanding of concepts. Several ontology integration methods exist for such cases as described in [54]. As there will be hardly a consensus about the ontologies to be used, topics such as ontology alignment and mapping will become more important. Additionally, proper ontology design methods, such as those presented in [55], facilitate the reusability of these ontologies.

The federated query engine enables the distribution of the knowledge graph. Every service of the DT manages its information and can decide which part should be shared with other services by including it in the knowledge graph. The parts which are not included are only available locally by the service itself (RI3, RI4). However, reasoning on the shared knowledge graph can introduce problems because of inconsistency. As services can add relevant information by themselves to the shared knowledge graph, this problem becomes more likely. To avoid this problem, reasoning should only be applied to private parts of the knowledge graph, where consistency can be guaranteed to some extent. If reasoning should be performed on the whole graph, other techniques have to be applied which can deal with uncertainty, e.g., fuzzy reasoning methods. Even if reasoning capabilities can not be provided, the knowledge graph can still be used for data retrieval and knowledge discovery.

Using a workflow engine for service orchestration is beneficial for holding states during human–machine interaction. Using BPMN also facilitates the integration DT services into workflows at enterprises level. For the production level planning, BPMN is not always sufficient and other notations such as Coloured Petri Nets might be more suitable [56]. Extensions of the workflow engine could support this multi-level workflow execution in a more convenient way.

Future work will investigate the real-time performance of the proposed service architecture for various use-cases. In this context, the integration of OPC UA is planned. How OPC UA information can be included in the shared knowledge graph has already been shown in [57]. Next, OPC UA integration into the service framework will be investigated. In this context, the execution time of SPARQL endpoints can become relevant, which will be further investigated.

Author Contributions: Conceptualization, G.S.; methodology, G.S.; software, G.S.; validation, G.S.; writing—original draft preparation, G.S.; writing—review and editing, W.K.; visualization, G.S.; supervision, W.K.; project administration, W.K. Both authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding, but was supported by the doctoral school SIC!—Smart Industrial Concept! at the TU Wien.

Acknowledgments: The authors acknowledge TU Wien Bibliothek for financial support through its Open Access Funding Programme. Open Access Funding by TU Wien.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

5D-DT	Five-Dimensional Digital Twin
ARX	Autoregressive with exogenous input
BPMN	Business Process Model and Notation
DT	Digital Twin
ESB	Enterprise Service Bus
GDTA	Generic Digital Twin Architecture
HTTP	Hypertext Transfer Protocol
ICPS	Industrial Cyber-Physical Systems
ICT	Information and Communication Technology
IIRA	Industrial Internet Reference Architecture
IoT	Internet of Things
IT	Information Technology
JSON-LD	JSON for Linking Data
MOM	Message-oriented Middleware
OBDA	Ontology-Based Data Access
OPC UA	OPC Unified Architecture
OWL	Web Ontology Language
RAMI 4.0	Reference Architecture Model Industry 4.0
REST	Representational State Transfer
SPARQL	SPARQL Protocol and RDF Query Language

References

1. Parida, V.; Sjödin, D.; Reim, W. Reviewing literature on digitalization, business model innovation, and sustainable industry: Past achievements and future promises. *Sustainability* **2019**, *11*, 391. [CrossRef]
2. Tao, F.; Zhang, H.; Liu, A.; Nee, A.Y. Digital Twin in Industry: State-of-the-Art. *IEEE Trans. Ind. Inform.* **2019**, *15*, 2405–2415. [CrossRef]
3. Malakuti, S.; van Schalkwyk, P.; Boss, B.; Ram Sastry, C.; Runkana, V.; Lin, S.W.; Rix, S.; Green, G.; Baechle, K.; Varan Nath, C. *Digital Twins for Industrial Applications. Definition, Business Values, Design Aspects, Standards and Use Cases*; White Paper; Industrial Internet Consortium: Milford, MA, USA, 2020; pp. 1–19.
4. Lu, Y.; Liu, C.; Wang, K.I.; Huang, H.; Xu, X. Digital Twin-driven smart manufacturing: Connotation, reference model, applications and research issues. *Robot. Comput. Integr. Manuf.* **2020**, *61*, 101837. [CrossRef]
5. Grieves, M. *Digital Twin: Manufacturing Excellence through Virtual Factory Replication this paper Introduces the Concept of a A Whitepaper by Dr. Michael Grieves*; White Paper; Florida Institute of Technology: Melbourne, FL, USA, 2014.
6. Ashtari Talkhestani, B.; Jung, T.; Lindemann, B.; Sahlab, N.; Jazdi, N.; Schloegl, W.; Weyrich, M. An architecture of an Intelligent Digital Twin in a Cyber-Physical Production System. *Automatisierungstechnik* **2019**, *67*, 762–782. [CrossRef]
7. Josifovska, K.; Yigitbas, E.; Engels, G. Reference Framework for Digital Twins within Cyber-Physical Systems. In Proceedings of the 2019 IEEE/ACM 5th International Workshop on Software Engineering for Smart Cyber-Physical Systems, SEsCPS 2019, Montreal, QC, Canada, 28 May 2019; pp. 25–31. [CrossRef]
8. Tao, F.; Zhang, M.; Nee, A. Five-Dimension Digital Twin Modeling and Its Key Technologies. *Digit. Twin Driven Smart Manuf.* **2019**, 63–81. [CrossRef]
9. Abburu, S.; Berre, A.J.; Jacoby, M.; Roman, D.; Stojanovic, L.; Stojanovic, N. COGNITWIN—Hybrid and Cognitive Digital Twins for the Process Industry. In Proceedings of the 2020 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC), Cardiff, UK, 15–17 June 2020; pp. 1–8. [CrossRef]
10. Steindl, G.; Stagl, M.; Kasper, L.; Kastner, W.; Hofmann, R. Generic digital twin architecture for industrial energy systems. *Appl. Sci.* **2020**, *10*, 8903. [CrossRef]
11. Adolphs, P.; Bedenbender, H.; Dirzus, D.; Martin, E. *Reference Architecture Model Industrie 4.0 (RAMI4.0)*; Technical Report; VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik/ZVEI: Düsseldorf, Germany, July 2015. Available online: https://www.vdi.de/ueber-uns/presse/publikationen/details/reference-architecture-model-industrie-40-rami40-english-version (accessed on 17 June 2021).

12. De Lauretis, L. From monolithic architecture to microservices architecture. In Proceedings of the 2019 IEEE 30th International Symposium on Software Reliability Engineering Workshops (ISSREW 2019), Berlin, German, 27–30 October 2019; pp. 93–96. [CrossRef]
13. Dragoni, N.; Giallorenzo, S.; Lafuente, A.L.; Mazzara, M.; Montesi, F.; Mustafin, R.; Safina, L., Microservices: Yesterday, Today, and Tomorrow. In *Present and Ulterior Software Engineering*; Springer International Publishing: Cham, Switzerland, 2017; pp. 195–216. [CrossRef]
14. Stutz, A.; Fay, A.; Barth, M.; Maurmaier, M. Orchestration vs. Choreography Functional Association for Future Automation Systems. *IFAC-PapersOnLine* **2020**, *53*, 8268–8275. [CrossRef]
15. Newman, S. *Building Microservices—Design Fine Grained Systems*; O'Reilly Media, Inc.: Newton, MA, USA, 2021.
16. Foundation, A.S. Apache Kafka. Available online: https://kafka.apache.org/ (accessed on 12 May 2021).
17. Cimino, C.; Negri, E.; Fumagalli, L. Review of digital twin applications in manufacturing. *Comput. Ind.* **2019**, *113*, 103130. [CrossRef]
18. Damjanovic-Behrendt, V.; Behrendt, W. An open source approach to the design and implementation of Digital Twins for Smart Manufacturing. *Int. J. Comput. Integr. Manuf.* **2019**, *32*, 366–384. [CrossRef]
19. Alaasam, A.B.; Radchenko, G.; Tchernykh, A. Stateful stream processing for digital twins: Microservice-based kafka stream dsl. In Proceedings of the SIBIRCON 2019—International Multi-Conference on Engineering, Computer and Information Sciences, Novosibirsk, Russia, 21–27 October 2019; pp. 804–809. [CrossRef]
20. Theorin, A.; Bengtsson, K.; Provost, J.; Lieder, M.; Johnsson, C.; Lundholm, T.; Lennartson, B. An event-driven manufacturing information system architecture for Industry 4.0. *Int. J. Prod. Res.* **2017**, *55*, 1297–1311. [CrossRef]
21. Apache Software Foundation. Apache ActiveMQ—Flexible & Powerful Open Source Multi-Protocol Messaging. Available online: https://activemq.apache.org/ (accessed on 19 May 2021).
22. Ali, S.; Jarwar, M.A.; Chong, I. Design methodology of microservices to support predictive analytics for IoT applications. *Sensors* **2018**, *18*, 4226. [CrossRef] [PubMed]
23. Docker. Docker—Accelerate How You Build, Share and Run Modern Applications. Available online: https://www.docker.com/ (accessed on 19 May 2021).
24. Fattah, S.; Sung, N.M.; Ahn, I.Y.; Ryu, M.; Yun, J. Building IoT services for aging in place using standard-based IoT platforms and heterogeneous iot products. *Sensors* **2017**, *17*, 2311. [CrossRef]
25. Moyne, J.; Qamsane, Y.; Balta, E.C.; Kovalenko, I.; Faris, J.; Barton, K.; Tilbury, D.M. A Requirements Driven Digital Twin Framework: Specification and Opportunities. *IEEE Access* **2020**, *8*, 107781–107801. [CrossRef]
26. Tao, F.; Cheng, J.; Qi, Q.; Zhang, M.; Zhang, H.; Sui, F. Digital twin-driven product design, manufacturing and service with big data. *Int. J. Adv. Manuf. Technol.* **2018**, *94*, 3563–3576. [CrossRef]
27. Alam, K.M.; El Saddik, A. C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems. *IEEE Access* **2017**, *5*, 2050–2062. [CrossRef]
28. Engelsberger, M.; Greiner, T. Software architecture for cyber-physical control systems with flexible application of the software-as-a-service and on-premises model. In Proceedings of the IEEE International Conference on Industrial Technology, Seville, Spain, 17–19 March 2015; pp. 1544–1549. [CrossRef]
29. Ding, K.; Chan, F.T.; Zhang, X.; Zhou, G.; Zhang, F. Defining a Digital Twin-based Cyber-Physical Production System for autonomous manufacturing in smart shop floors. *Int. J. Prod. Res.* **2019**, *57*, 6315–6334. [CrossRef]
30. Saad, A.; Faddel, S.; Mohammed, O. IoT-based digital twin for energy cyber-physical systems: Design and implementation. *Energies* **2020**, *13*, 4762. [CrossRef]
31. Borodulin, K.; Sokolinsky, L.; Radchenko, G.; Tchernykh, A.; Shestakov, A.; Prodan, R. Towards digital twins cloud platform: Microservices and computational workflows to rule a smart factory. In Proceedings of the 10th International Conference on Utility and Cloud Computing—UCC 2017, Austin, TX, USA, 5–8 December 2017; pp. 205–206. [CrossRef]
32. Gorecky, D.; Schmitt, M.; Loskyll, M.; Zühlke, D. Human–machine-interaction in the industry 4.0 era. In Proceedings of the 2014 12th IEEE International Conference on Industrial Informatics (INDIN 2014), Porto Alegre, Brazil, 27–30 July 2014; pp. 289–294. [CrossRef]
33. Panfilenko, D.; Poller, P.; Sonntag, D.; Zillner, S.; Schneider, M. BPMN for knowledge acquisition and anomaly handling in CPS for smart factories. In Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, Berlin, Germany, 6–9 September 2016; pp. 2–5. [CrossRef]
34. Kleppmann, M. *Designing Data-Intensive Applications*; O'Reilly Media, Inc.: Newton, MA, USA, 2017.
35. Ehrlinger, L.; Wöß, W. Towards a definition of knowledge graphs. In Proceedings of the CEUR Workshop, Leipzig, Germany, 13–14 September 2016; Volume 1695.
36. Studer, R.; Benjamins, V.R.; Fensel, D. Knowledge Engineering: Principles and methods. *Data Knowl. Eng.* **1998**, *25*, 161–197. [CrossRef]
37. Schwarte, A.; Haase, P.; Hose, K.; Schenkel, R.; Schmidt, M. FedX: Optimization techniques for federated query processing on linked data. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer, Berlin/Heidelberg, Germany, 2011. [CrossRef]

38. Steindl, G.; Kastner, W. Query Performance Evaluation of Sensor Data Integration Methods for Knowledge Graphs. In Proceedings of the 6th IEEE International Conference on Big Data, Knowledge and Control Systems Engineering, Sofia, Bulgaria, 21–22 November 2019.
39. Schachinger, D.; Kastner, W.; Gaida, S. Ontology-based abstraction layer for smart grid interaction in building energy management systems. In Proceedings of the 2016 IEEE International Energy Conference (ENERGYCON 2016), Leuven, Belgium, 4–8 April 2016; [CrossRef]
40. Richardson, C. *Microservices Patterns*; Manning Publications: Shelter Island, NY, USA, 2018; p. 520.
41. Kozma, D.; Varga, P.; Larrinaga, F. Dynamic Multilevel Workflow Management Concept for Industrial IoT Systems. *IEEE Trans. Autom. Sci. Eng.* **2020**, 1–13. [CrossRef]
42. Gutiérrez–Fernández, A.M.; Resinas, M.; Ruiz–Cortés, A. Redefining a process engine as a microservice platform. *Lect. Notes Bus. Inf. Process.* **2017**, *281*, 252–263. [CrossRef]
43. Camunda. Zeebe Workflow Engine for Microservices Orchestration. Available online: https://github.com/camunda-cloud/zeebe (accessed on 12 May 2021).
44. RDF4J. Federation with FedX. Available online: https://rdf4j.org/documentation/programming/federation/ (accessed on 12 May 2021).
45. Steindl, G. Digital Twin Service Framework. Available online: https://github.com/Smart-Industrial-Concept/DigitalTwinServiceFramework (accessed on 19 May 2021).
46. Fritzson, P.; Pop, A.; Abdelhak, K.; Ashgar, A.; Bachmann, B.; Braun, W.; Bouskela, D.; Braun, R.; Buffoni, L.; Casella, F.; et al. The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development. *Model. Identif. Control* **2020**, *41*, 241–295. [CrossRef]
47. Steindl, G. Heating Process Simulation. Available online: https://github.com/Smart-Industrial-Concept/HeatingProcessSimulation (accessed on 19 May 2021).
48. Steindl, G.; Kastner, W. Ontology-Based Model Identification of Industrial Energy Systems. *IEEE Int. Symp. Ind. Electron.* **2020**, 1217–1223. [CrossRef]
49. The PostgreSQL Global Development Group. PostgreSQL: The World's Most Advanced Open Source Relational Database. Available online: https://www.postgresql.org/ (accessed on 12 May 2021).
50. Xiao, G.; Lanti, D.; Kontchakov, R.; Komla-Ebri, S.; Güzel-Kalaycı, E.; Ding, L.; Corman, J.; Cogrel, B.; Calvanese, D.; Botoeva, E. The virtual knowledge graph system ontop. *CEUR Workshop Proc.* **2020**, *2663*, 1–16.
51. World Wide Web Consortium. Semantic Sensor Network Ontology. W3C Recommendation. Available online: https://www.w3.org/TR/vocab-ssn/ (accessed on 19 May 2021).
52. Hobbs, J.R.; Little, C. Time Ontology in OWL. W3C Candidate Recommendation. Technical Report, World Wide Web Consortium (W3C). 2020. Available online: https://www.w3.org/TR/owl-time/ (accessed on 17 June 2021).
53. Elastic. Elasticsearch—Distributed, Multitenant-Capable Full-Text Search Engine. Available online: https://www.elastic.co/elasticsearch/ (accessed on 19 May 2021).
54. Ekaputra, F.J.; Sabou, M.; Serral, E.; Kiesling, E.; Biffl, S. Ontology-Based Data Integration in Multi-Disciplinary Engineering Environments: A Review. *Open J. Inf. Systms* **2017**, *4*, 1–26.
55. Frühwirth, T.; Kastner, W.; Krammer, L. A methodology for creating reusable ontologies. In Proceedings of the 2018 IEEE Industrial Cyber-Physical Systems (ICPS 2018), St. Petersburg, Russia, 15–18 May 2018; pp. 65–70. [CrossRef]
56. Kozma, D.; Varga, P.; Larrinaga, F. Data-driven Workflow Management by utilising BPMN and CPN in IIoT Systems with the Arrowhead Framework. In Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, Zaragoza, Spain, 10–13 September 2019; pp. 385–392. [CrossRef]
57. Steindl, G.; Früwirth, T.; Kastner, W. Ontology-Based OPC UA Data Access via Custom Property Functions. In Proceedings of the 24th Interantional Conference on Emerging Technologies and Factory Automation, Zaragoza, Spain, 10–13 September 2019.