information

MDPI

Article

An Approach for Realizing Hybrid Digital Twins Using Asset Administration Shells and Apache StreamPipes

Michael Jacoby [1,*], Branislav Jovicic [2], Ljiljana Stojanovic [1] and Nenad Stojanović [2]

[1] Fraunhofer IOSB, 76131 Karlsruhe, Germany; ljiljana.stojanovic@iosb.fraunhofer.de
[2] Nissatech, 18000 Niš, Serbia; branislav.jovicic@nissatech.com (B.J.); nenad.stojanovic@nissatech.com (N.S.)
* Correspondence: michael.jacoby@iosb.fraunhofer.de; Tel.: +49-721-6091-470

Abstract: Digital twins (DTs) are digital representations of assets, capturing their attributes and behavior. They are one of the cornerstones of Industry 4.0. Current DT standards are still under development, and so far, they typically allow for representing DTs only by attributes. Yet, knowledge about the behavior of assets is essential to properly control and interact with them, especially in the context of industrial production. This behavior is typically represented by multiple different models, making integration and orchestration within a DT difficult to manage. In this paper, we propose a new approach for hybrid DTs by intertwining different DT models. We also show how to realize this approach by combining the Fraunhofer Asset Administration Shell (AAS) Tools for Digital Twins (FAST) to create Industry 4.0-compliant DTs with Apache StreamPipes to implement and manage multiple DT models. Our prototype implementation is limited to a subset of the AAS metamodel and pull-based communication between FAST and an external Apache StreamPipes instance. Future work should provide full support for the AAS metamodel, publish/subscribe-based communication, and other execution environments as well as deployment strategies. We also present how this approach has been applied to a real-world use case in the steel production industry.

Keywords: digital twin; asset administration shell; hybrid digital twin; processing pipelines; interoperability

Citation: Jacoby, M.; Jovicic, B.; Stojanovic, L.; Stojanović, N. An Approach for Realizing Hybrid Digital Twins Using Asset Administration Shells and Apache StreamPipes. *Information* **2021**, *12*, 217. https://doi.org/10.3390/info12060217

Academic Editor: David M. Nichols

Received: 29 April 2021
Accepted: 18 May 2021
Published: 21 May 2021

1. Introduction

In recent years, digital twin (DT) technology has rapidly gained acceptance in various industries. As a digital representation of a physical asset, DTs are able to support the entire lifecycle of an asset. Indeed, the concept of a DT evolved from a model of a physical entity, mostly used in the design phase, to the mirroring of an existing physical entity that can be used as a reference model for a specific physical entity.

DT technology requires that the necessary models of an asset, which mimic the behavior of an asset, are available and accurate [1]. However, this is not always the case, as it may not be possible to create an accurate model, or the process of developing such a model may be time consuming and expensive [2]. This is especially true for assets operating under harsh conditions (e.g., extreme temperatures), where the instability of the process conditions places extremely high demands on the modeling paradigms (numerical modeling as well as data-driven models) [3]. For example, some phenomena (such as thermal conduction) cannot be modeled accurately with first-principle models. On the other hand, some situations are not observable (as there are no sensors for real-time monitoring) and the data needed to build prediction/classification models are missing [4]. In such cases, only models with lower accuracy or based on other phenomena can be developed. Therefore, this paper tackles the problem of representing manufacturing assets using DT technology in cases where the numerical and data-driven models are only partially available and a hybrid approach is requested.

One of the widely used scenarios is tool wear (i.e., equipment degradation), which is highly relevant for assets working in harsh conditions (and, therefore, subject to more

intensive degradation). In such scenarios, data are missing because the equipment is inaccessible and data are incomplete because the equipment is replaced before it is broken. A common solution for such cases is the creation of hybrid digital twins, which allow different combinations of numerical and data-driven models to exploit the full potential of DT models while increasing their transparency and accuracy.

In this paper, we propose a new approach for modeling digital replicas of assets given the fact that accurate models and data are not available and traditional DTs cannot be applied. The approach is based on the hybrid DT paradigm [5], where the process of hybridization goes beyond solely connecting models and data, which usually requires accurate models and complete data [6]. We propose the orchestration of data and models as a new method for connecting (imprecise) models and (incomplete) data. An efficient implementation of orchestration is ensured by using the Apache StreamPipes toolbox [7]. Standardization is achieved by using the Asset Administration Shell [8].

To demonstrate the necessity and applicability of the proposed approach, we have explained how it was applied to realize a complex tool wear scenario in steel production. We argue that such an approach brings several benefits, from creating accurate orchestrated models for the tool wear process to helping operators to gain a deep understanding of the wear process and improving the decision-making process (e.g., optimal decision on when to repair a piece of equipment).

The paper is structured as follows. Section 2 discusses the challenges for realizing hybrid DTs and justifies the decisions made. Section 3 provides a high-level overview of the background technologies and compares them with possible alternatives. Section 4 proposes an architecture for hybrid DTs and explains our approach to realize them. Section 5 shows how the concept of hybrid DTs can be applied to a real-world case from the steel production industry. The paper closes with conclusions and outlooks in Section 6.

2. Challenges and Requirements

According to the Industrial Internet Consortium (IIC) [1], a DT is defined as a digital replica of an asset that captures the attributes and behaviors of that asset. DTs can be developed for different types of assets, such as equipment, facilities, materials, processes, and products but also humans, documents, software systems, etc. The key elements of a DT are its attributes describing its properties, its models describing its behavior, and services that the DT provides to external systems. For more complex cases, a DT may also include or reference other DTs.

The main idea of DTs is to solve an information silo problem by combining isolated data in a semantically consistent manner by ensuring that DT users (either humans or software systems) have an integrated, unified view of the data and information. To provide such an integrated view of the data and ensure a more comprehensive assessment of the data, the source data are fused by mapping them into the DT metamodel. An overview of different DT standards and specifications as well as their comparison can be found in [9].

DT models are of different types, such as 3D models, physical models, data-driven models, etc. As discussed in [10], the presence of different independent models leads to an interoperability problem at the level of digital twin models, which is an even more difficult problem due to many reasons, such as the model type, the DT lifecycle phase for which it was created, etc.

To gain comprehensive insight, it is necessary to support the collaboration of multiple models and corresponding data. This strengthens each individual model. A typical scenario would be to use a simulation model to generate training data for machine learning models, as there are usually not enough training data available.

In [5], we proposed extending the concept of DTs to hybrid DTs by interconnecting different models in order to achieve higher predictive capabilities and greater potential for industrial applications. In this paper, we follow this approach and apply the same definition of a hybrid DT: ”A Hybrid Digital Twin is an extension of DT in which the

isolated DT models are intertwined to detect, predict, and communicate less optimal (but predictable) behavior of the physical counterpart long before that behavior occurs"[5].

Based on the previous discussion, the following key requirements can be derived to realize the vision of hybrid DTs:

- Interoperability with external systems: A DT should be easily integrated with other software systems. This requires standards for the DT metamodel as well as standardized interfaces to not only exchange messages with the external systems but also to understand their meaning.
- Interoperability within a DT: DT models should be made interoperable wherever it makes sense. There are no predefined pipelines; i.e., the way models are combined is application specific.
- Extensibility: A DT should be able to handle different types of models. These models could be developed with different technologies, in different programming languages, and using different protocols and require different environments to run them. New models for DT behavior can be created, and new types of models can be considered.
- Reusability: A DT should not be developed from scratch, but reusing existing models/components should be considered.

In the following, we describe our approach to solve the above challenges. Since DTs should not be proprietary solutions but rather be developed based on standards, we propose to develop DTs based on the Asset Administration Shell (AAS) [8]. To our knowledge, this is the only available DT standard with a strong focus on industry, which we believe is an essential prerequisite to be recognized and become well accepted and adapted in real-world industrial plants in the future.

To be able to reuse and orchestrate different DT models and, at the same time, reduce the number of interfaces to be implemented, we decided to use the StreamPipes toolbox [7]. By expressing the DT models, e.g., ML based or physics based, as StreamPipes processors, syntactic and semantic interoperability between DT models is achieved. Model reusability is accomplished by registering the models and their types in the StreamPipes toolbox.

The benefits of using Industry 4.0 technologies in general and AASs in particular for DTs are discussed in several papers. In [11], the authors explored the technical aspects of DTs. One of the key findings was that AASs can support digital twins not only for manufacturing but also in use cases beyond. In [12], the authors identified suitable Industry 4.0 technologies and a holistic reference architecture model to realize the most sophisticated DT-enabled applications. While [12] states that Industry 4.0 technologies advance DTs to drive industrial transformation, in this paper, we explain how this vision can be realized by using the Industry 4.0 AAS metamodel and AAS APIs as the backbone of the proposed approach.

According to [13], a DT is a key enabler in implementing Industry 4.0 in the areas of construction and smart cities. The authors claim that DTs should offer much more than digital representation and should include features such as bi-directional data exchange and real-time self-management. The approach proposed in this work not only enables synchronization between the physical entity and its digital representation with a certain frequency and fidelity, but also provides support for real-time self-management by realizing business logic via StreamPipes.

3. Background Technologies

In this section, we present the most important technologies relevant to our approach for realizing hybrid DTs.

3.1. Asset Administration Shell Specification

The success and usefulness of DTs depends heavily on standardized interfaces to interact with the DTs. We investigated and compared the following state-of-the-art digital twin and IoT standards and specifications: Asset Administration Shell, W3C Web of Things, Digital Twin Definition Language, NGSI-LD, OData, and OGC SensorThings API. The

detailed analysis is published as open access [9]. Based on this work, we decided to use the AAS specification to implement DTs. Although the AAS specification is (partially) still a work in progress, we deemed it best suited because of its strong focus on and background in industry. To our knowledge, the AAS specification is the only DT standard or specification that explicitly supports industry-typical protocols and data formats such as OPC UA and AutomationML. The fact that it is still a work in progress seems acceptable considering that this applies to all DT standards—i.e., there is no complete and "ready-to-use" standard available at this time, as the domain of DT (in industry) is still rather new.

The AAS is a concept developed by the Plattform Industrie 4.0, a network of companies, associations, trade unions, science, and politics in Germany [8]. It is part of the Reference Architectural Model Industry 4.0 (RAMI4.0) [14] and is driven by the idea of manifesting the concept of DTs in factories and industrial production plants. The specification is currently published in two parts. Part 1 [15] introduces the AAS metamodel, including different serialization formats (JSON, XML, RDF, AutomationML, and OPC UA node set), and Part 2 [16] defines different APIs in a protocol- and technology-agnostic way that can be used to interact with an AAS.

Figure 1 shows a simplified version of the AAS metamodel. The main idea is that AssetAdministrationShell represents an Asset (via AssetInformation) and may reference multiple Submodels, which are comprised of a set of SubmodelElements. There are multiple concrete implementations of the abstract SubmodelElement class, of which only Property and Operation are explicitly shown in Figure 1. The rationale behind having Submodel as an intermediate element between AssetAdministrationShell and the SubmodelElements is that submodels group elements in a useful way—e.g., according to functionality or type of connected/represented device—and will be standardized in the future.

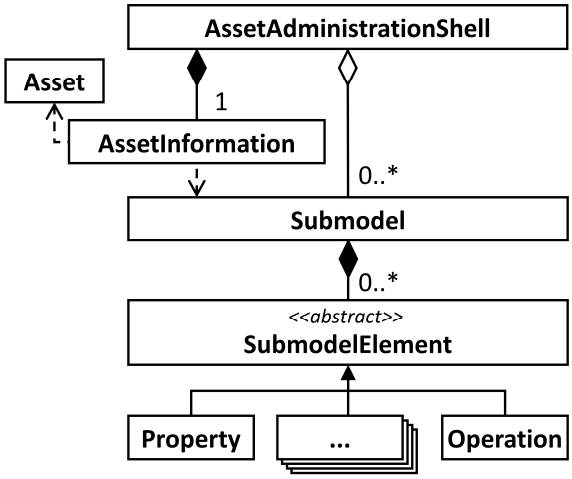

Figure 1. Simplified metamodel of the Asset Administration Shell (v3.0RC01) based on [15].

3.2. Apache StreamPipes

Apache StreamPipes is an open-source Internet of Things (IoT) toolbox that enables users to collect, analyze, and explore IoT data in a streaming manner [7]. One of the key concepts in Apache StreamPipes is pipelines. A pipeline is composed of a set of pipeline elements, typically data sources, data processors, and data sinks, that are combined in a workflow-like manner. Apache StreamPipes offers an intuitive and easy-to-use visual editor that enables non-expert users to create and execute such pipelines.

In Apache StreamPipes, pipeline elements are separated in three groups based on their role within a pipeline:

- Data Sets/Streams—They represent the start of a pipeline and cover data acquisition from an external data source, such as sensors, databases, etc. In essence, they are responsible for data fetching and forwarding data to other elements within the pipeline. They can either be implemented as concrete data sets/streams allowing no further

configuration, or instantiated with generic templates, called adapters, that offer further configuration options.

- Data Processors—Used for defining a pipeline's main logic, i.e., defining what a pipeline does with the data fetched from the external source. Processing steps include filtering, aggregating, trend detection, image manipulation, deep and machine learning, and many more.
- Data Sinks—They represent an ending point of a pipeline and, as such, are used to store data, send data to visualization systems, send notifications/alerts, forward data to third-party systems for further processing, etc.

For easy and fast development of pipelines, Apache StreamPipes is delivered with lots of ready-to-use implementations of these elements, allowing out-of-the-box connection to various databases, brokers, and sensors as well as providing support for frequently used protocols, such as HTTP, Kafka, MQTT, and others.

Data being exchanged between pipeline elements are organized into events, with each element having specified the structure of events that it receives and forwards, called event schemas. This way, Apache StreamPipes manages the connections of various pipeline elements and does not allow connection of incompatible ones.

Furthermore, the pool of available elements can easily be extended with custom implementations, e.g., for specific deep and machine learning models or model types, statistical analysis, complex event processing, or custom data sources/sinks.

3.3. Comparing StreamPipes and Node-RED

Node-RED is an open-source platform developed by IBM to easily create IoT applications with very minimal programming skills [17]. Although both Node-RED and Apache StreamPipes enable the graphical creation of processing pipelines in the sense of a "boxes and arrows" approach, a closer look reveals various differences between them, both in terms of the target user group and technical level.

On a technical level, Node-RED is described as a solution for "low-code programming" [18]. The focus is on the development of data-driven applications by means of a graphical user interface for users with low programming skills or little programming effort. This means that during creation, there must be a basic understanding of underlying data structures, e.g., the structure of the JSON objects used for transmission. In contrast, Apache StreamPipes is aimed at business users without programming skills. The interface for creating pipelines abstracts from the underlying data structure and thus requires less technical understanding. While Apache StreamPipes uses a semantic description level—for example, to automatically hide non-compatible data fields of an input data stream in a data processor and thus reduce user errors—Node-RED uses a purely syntactic manipulation of the parameters at the level of JSON documents.

The differences at the technical level are most apparent in the implementation of the runtime environment. Node-RED relies on a Node.js runtime environment, i.e., a so-called "single host runtime" processing the incoming data of a processor, whereas Apache StreamPipes uses a wrapper architecture, which allows the development of a pipeline element in different execution environments and easy scalability. In addition to lightweight wrappers for edge nodes with low processing power, Apache StreamPipes offers wrappers for scalable big data systems such as Apache Flink or Apache Kafka. One advantage of this architecture is that the execution layer can also be realized in different programming languages. A Python wrapper for Apache StreamPipes is also available as a prototype, which, e.g., simplifies the use of ML-based processors.

Furthermore, StreamPipes can be seen as more of an end-to-end toolbox, with the pipeline editor being just one module out of several that aim to support non-technical users in analyzing Industrial IoT (IIoT) data.

4. Approach to Hybrid Twins

In this section, we present our approach on how to create real-world hybrid DTs. We start with a technology-agnostic architecture and show how to implement executable DTs using Fraunhofer AAS Tools for Digital Twins (FAST) based on a minimal DT example. Additionally, a concrete implementation of the technology-agnostic architecture using Apache StreamPipes is presented.

4.1. Architecture

To realize a hybrid DT based on the combination of multiple models, we propose the general architecture presented in Figure 2.

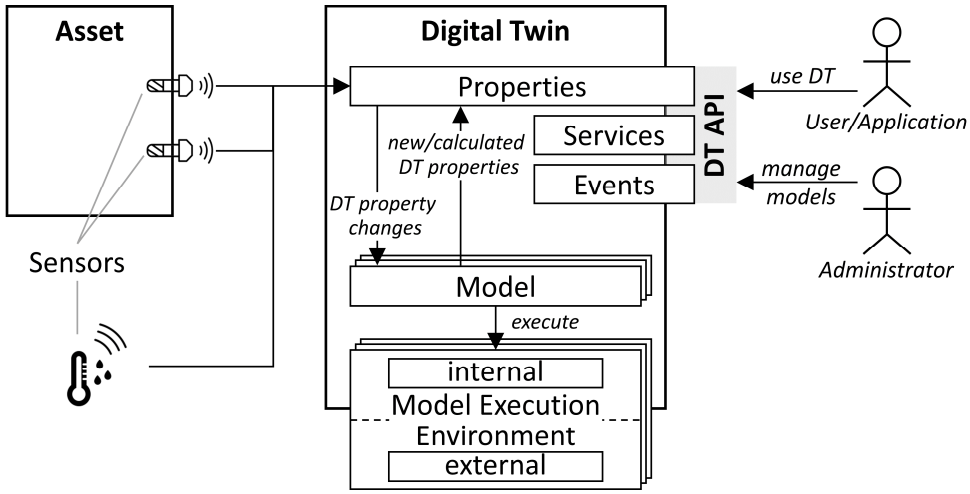

Figure 2. General architecture for realizing hybrid digital twins.

A DT is a digital representation of an asset. It comprises relevant information from sensors attached to the asset as well as from relevant external sensors. To enable interoperability within and across factories, DTs should implement a common and standardized application programming interface (API). Current existing DT standards typically represent a DT as a set of properties, services, and events and provide an HTTP/REST-based API to interact with the DT [9]. This is reflected in Figure 2 in the upper right part of the DT and by the user/application that is using the DT API.

A DT may also comprise a set of models. These models can be of different types, e.g., physical or data-driven. A DT is called a hybrid DT when it comprises multiple interconnected models instead of any number of isolated models [5].

All models have in common the fact that they take the changing values of properties of the DT as input, apply some logic, and report back their result to the DT by either updating the existing properties or creating new ones. Different models are implicitly interconnected or combined by using the output of one model as the input for another one. As models represent a logic-processing step, they need to be executed. For each type of model, this requires a model execution environment that can be deployed inside and/or outside of a DT. The decision on where to deploy the execution environments is a trade-off between the ability of the DT to run as a stand-alone application and the resource requirements, especially because using external model execution environments can be shared between multiple DTs. We also propose to "extend" the DT API (by adding predefined services) to allow administrative users to create, read, update and delete models as well as to start and stop their execution.

4.2. AAS Model Example

In the remainder of this section, we will illustrate how to prototypically create a hybrid DT. This will be based on a minimalistic artificial use case where we want to supervise

a machine using a temperature sensor and trigger an overheating alarm when certain conditions are met.

The first step is to model the DT according to the AAS metamodel (see Figure 1). Figure 3 shows a simplified version of the DT model for our use case. Basically, the model comprises an asset, Machine, and an AAS, MachineAAS, for that asset. The MachineAAS has a single submodel called Status which contains two properties: Temperature and OverheatingAlarm, whose types are float and boolean, respectively.

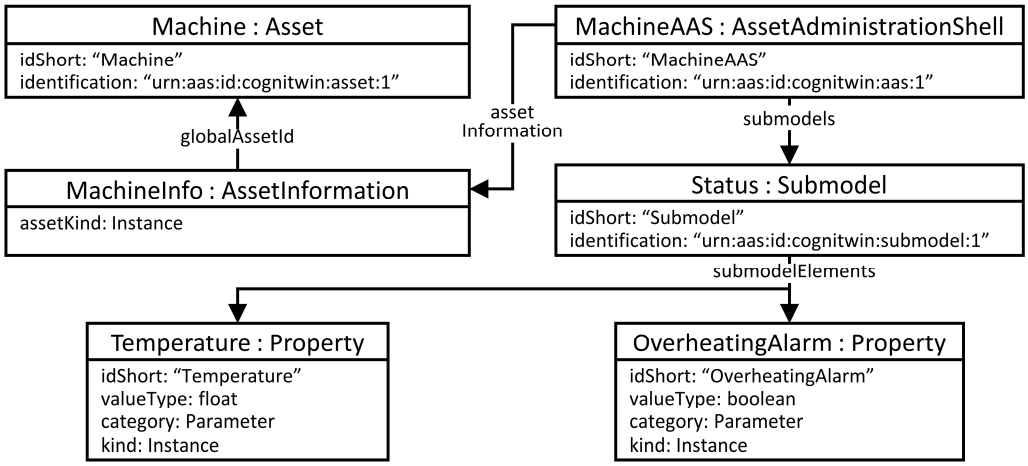

Figure 3. Simplified UML Object Diagram of the demonstrator Asset Administration Shell.

4.3. I4.0 Conform Digital Twins

In the previously introduced general architecture for hybrid DTs (Figure 2), models are defined as a set of instructions that operate on and produce (new) properties of a DT on a technology-agnostic level that can be executed in a model execution environment. To implement this, a concrete technology to define and execute such models has to be chosen. For this purpose, we selected Apache StreamPipes [7]. Figure 4 shows how this choice influences the architecture. As StreamPipes is a stream processing framework, models are realized using the pipeline concept. As the model execution environment, we selected an external StreamPipes deployment. It also provides an easy-to-use visual editor for creating and managing pipelines. As StreamPipes is a generic tool for creating processing pipelines using many different kinds of data sources and sinks, e.g., different network protocols, files, or databases, we created custom adapters for AAS-based DTs called DT Source and DT Sink. Additionally, read, start, and stop operations are supported on the pipelines via the DT API. The adoption of other pipeline operations (create, update, and delete) via the DT is in progress as well as deployment of the Apache StreamPipes runtime within the DT.

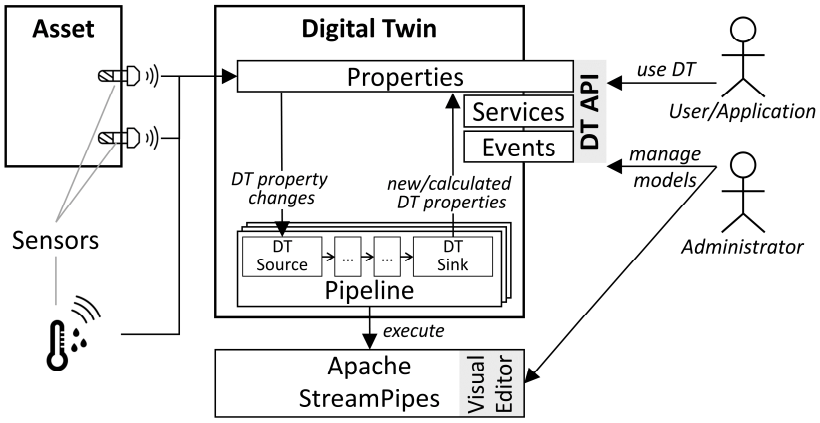

Figure 4. Adaptation of the general hybrid digital twin architecture to our implementation.

Another important aspect that will be addressed in the future is how historical data can be integrated in a hybrid DT. Current standards typically model DTs only as a representation of the current state of the asset, ignoring the fact that historical data are a relevant element, especially when it comes to applying, generating, or training models.

To implement our architecture, we developed a Java library called Fraunhofer AAS Tools for Digital Twins (FAST), which enables easy creation and execution of a DT. Figure 5 shows the high-level architecture of FAST. The library is designed to be easily extendible and customizable by offering a variety of extension points realized via Java interfaces, e.g., for different de-/serialization formats (JSON, XML, RDF, and AutomationML), persistence implementations, asset connections, and protocol bindings as well as custom I4.0 languages.

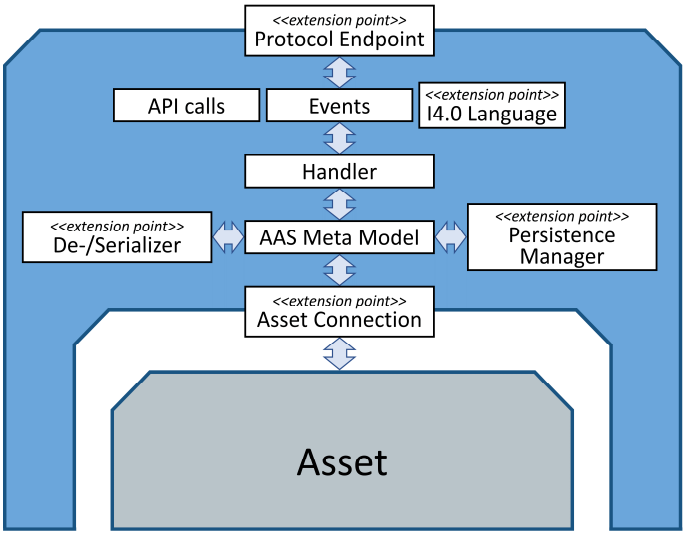

Figure 5. High-level architecture of the Fraunhofer AAS Tools for Digital Twin (FAST) library.

A core element of FAST is the AAS metamodel, which represents the DT with all of its properties and services. By introducing the asset connection extension point on the south side, we support connecting the AAS to any kind of resource or legacy system independently of the connection or protocol type. On the north side, providing an API to interact with the DT, we provide the protocol endpoint extension point, allowing to implement endpoints using multiple different communication protocols as envisioned by the AAS specification. FAST currently provides an HTTP/REST-based and an OPC UA endpoint. Each endpoint can provide functionality defined by the API calls in [16] and support events. Additionally, endpoints may support any I4.0 language [19]. I4.0 languages are a concept of Industry 4.0 providing (potentially) complex standardized communication protocols for special purposes.

Listing 1 shows how to create a DT using the fluent API of FAST. First, a previously defined AAS model is loaded from a *.json file (lines 1–2). In the second step, a DT service is created using the previously defined model (see Section 4.2) (line 4), including in-memory data persistence implementation (line 5) and an HTTP-based endpoint (lines 6–10). Additionally, the DT is connected to an external running StreamPipes environment by passing the HTTP endpoint and an access token (lines 11–17). Finally, the DT is started (lines 19–20).

4.4. Apache StreamPipes Extensions for Digital Twins

Connecting an AAS-based DT to Apache StreamPipes can be achieved by using the included data stream adapter, e.g., via HTTP. As this is rather cumbersome and requires the user to have lots of background knowledge about the URL schemes and payload formats of the AAS specification, we decided to develop two custom adapters, called DT Source and DT Sink, that support non-expert users with connecting to an AAS-based DT.

Listing 1 Example code showing how to start an AAS with an HTTP endpoint from a *.json file

```
AssetAdministrationShellEnvironment environment =
        new JsonFileReader().readFile("model.json");
AasService service = AasService.builder()
            .environment(environment)
            .dataAccess(new InMemoryDataAccess(environment))
            .endpoint(HttpEndpoint.builder()
                .port(8080)
                .mapping(IAssetAdministrationShellHttp.withPath("myAAS"))
                .mapping(ISubmodellHttp.withPath("myAAS"))
                .build())
            .extension(StreampipesExtension.builder()
               .environment(environment)
               .runtime(StreampipesRuntime.builder()
                  .endpoint("http://localhost/streampipes")
                  .apiKey("xxxxxxxxxxxxxxx")
                  .build())
            .build())
        .build();
AasServiceManager.Instance.setAasService(service);
service.start();
```

4.4.1. DT Source

In Apache StreamPipes, a custom data source/stream component can be implemented either as a separate data stream pipeline element or as an adapter template. Both approaches are valid and only differ in the degree of reusability of the component. A data stream represents a single concrete instance of a data source, and therefore, it cannot be further configured by the user and can only be used as is. In essence, DT Source could only be used for exactly the DT it is implemented for.

In contrast, adapters can be used to instantiate multiple data streams with different configurations. On a higher level, these data streams would do the same thing, i.e., connect to the DT, but with different connection configurations, thus allowing the user to connect to any AAS-based DT.

For this reason, we implemented the DT Source as an adapter template, ensuring interoperability between AAS and StreamPipes, as well as its reusability.

Figure 6 shows the configuration dialog when creating a DT Source in Apache StreamPipes. Currently, the user must provide the root URL of the DT API and a polling interval. As the AAS specification currently only defines a protocol-agnostic API, the DT Source only works with the FAST-specific mapping of the protocol-agnostic API to HTTP. For this reason, we currently only support pull-based integration.

Once this configuration step is finished, the DT Source fetches all SubmodelElements of type Property of the DT according to the metamodel shown in Figure 1. In the next step, all submodel elements are displayed in the form *[Submodel.idShort]_[Property.idShort]* as shown in Figure 7. In this dialog, the user can choose which properties are going to be part of the generated events and perform additional property configuration, such as basic value or unit transformations that are included in Apache StreamPipes.

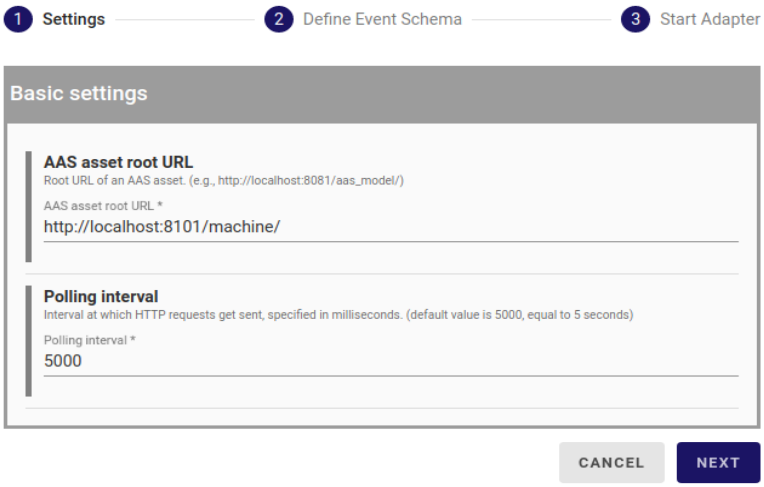

Figure 6. Configuration dialog when creating a DT Source.

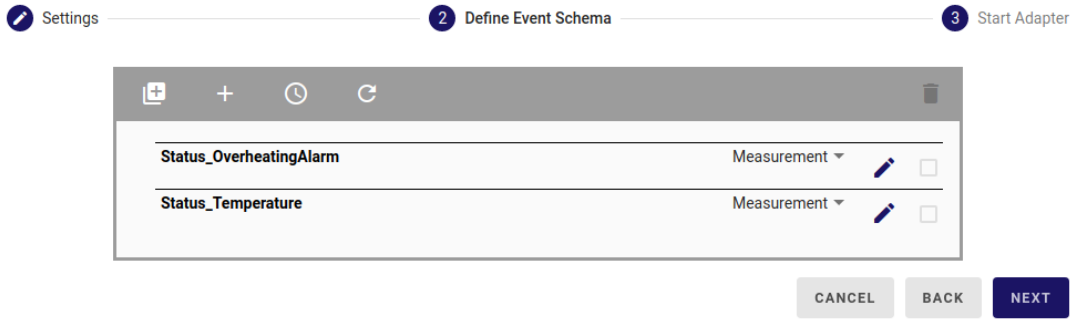

Figure 7. Dialog to modify the event schema of a DT Source showing all properties of an AAS-based DT according to the example AAS shown in Figure 3.

Once the user finishes configuration and starts the adapter, a corresponding data stream pipeline element is created, which will show up in the visual editor and can be placed in any pipeline via drag and drop. When placed in a pipeline, the DT Source will continuously (with the specified polling interval) generate events containing the latest values of the selected properties of the DT and forward them to following elements in the pipeline.

4.4.2. DT Sink

To update the DT with values calculated in a pipeline, we provide the DT Sink component. This component can be used to publish data to any AAS-based DT. Figure 8 shows the configuration dialog for the DT Sink. First, the user has to provide the URL of the submodel that the data should be written to. The second part of the dialog shows all available properties coming from the pipeline and allows to select which to send to the DT. While the pipeline is running, this element sends an HTTP PUT request for each of the selected event properties for every incoming event, thus updating the AAS-based DT.

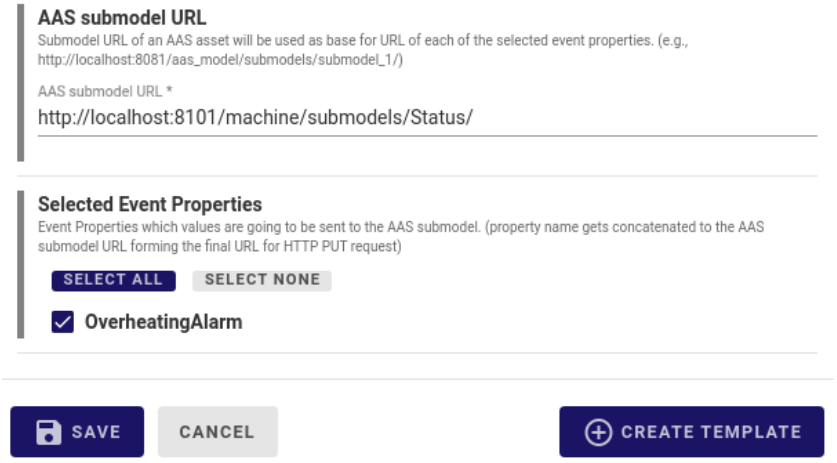

Figure 8. Configuration dialog of the DT Sink.

4.4.3. Future Extensions

The current implementation of DT Source and DT Sink is merely a prototype. The main reason for this is the fact that the AAS specification is still in development and currently only specifies a protocol-agnostic API and no mapping to a concrete communication protocol. Mappings to different communication protocols, such as MQTT or OPC UA, would ensure interoperability and easier integration of AAS into any software and should be taken into consideration. Regarding Apache StreamPipes, this would allow users to choose which communication protocol and serialization format to use and would enable publish/subscribe-based protocols as well.

5. Steel Production Use Case

Steel production is a complex process that usually consists of three stages. First, scrap steel is collected and melted in an electric arc furnace. In the second stage, the molten melt is transferred to ladles for secondary metallurgy. In the third and final step, the casting process, the molten steel is molded to a desired shape. The secondary metallurgy process is carried out in specially developed ladles lined with heat-resistant bricks that are designed to withstand the prevailing extreme temperatures and conditions for a sustained period. However, the inner wall of the ladles is worn out gradually with every heat. After a certain number of heats, typically between 50 and 100, these brick walls are so thinned down that the brick lining needs to be completely demolished, and a fresh batch of bricks has to be placed along the inner walls.

If the brick lining fails, the liquid steel might melt through the ladle and be released into the workshop, causing lethal danger to the workers and physical damage to the production site. Replacing the brick lining early, i.e., when it could potentially withstand more heat without failure, results in increased production overheads and costs. Therefore, when to repair a ladle is a critical decision. Up to now, it has been taken by a technician or an engineer based on visual inspection of the brick conditions and process parameters. However, due to the enormous risk to the health and safety of the workers in the production plant, the technician usually makes a "safe" decision to repair the ladle slightly early. A detailed description of the use case challenges can be found in [5].

Our goal was to develop an AAS-compliant hybrid DT for a ladle that gives a reliable and accurate recommendation on when to repair the ladle. This could be achieved by combining multiple models of different types and realizing them using the architecture and software introduced in Section 4.

5.1. Description of Available Data

There are three types of data relevant to steel production, each describing a particular part of the secondary metallurgy process:

- Low-frequency data—Single measurement per parameter, collected at the end of each ladle usage. Monitored parameters include the percentage of sulfur after vacuum, total electrical consumption, total amount of lime added, amount of time of steel being in the ladle, amount of alumina added, percentage of sulfur at tapping, whether burners are used, amount of time for vacuum, amount of gas used for stirring, amount of fluorspar added, amount of steel, continuous casting format, desulfurization speed, amount of time the ladle was powered on, percentage of EAF slag at tapping, and percentage of manganese at tapping. Historical acyclic data are used for the training of machine learning and deep learning models, while real-time acyclic data are used for inference.
- High-frequency data—Time-series measurements per parameter, collected every second during each ladle usage. Monitored parameters are nitrogen or argon pressure, gas flow rate, back pressure, type of used gas, gas instruction, vacuum pressure, temperature, and electrical consumption. Real-time cyclic data are used for outlier detection.
- Manually collected data—Measurements of brick thickness after each cycle of ladle usage. The ladle is vertically split in half, and for each half and for each layer of bricks in it, the minimum value of brick thickness is recorded. This type of data is not relevant for real-time processing. It is used for deep learning model training instead.

For the decision-making process, the following four low-frequency parameters are usually described as the most important ones: percentage of sulfur after vacuum, electrical consumption, amount of lime added, and amount of time of steel being in the ladle.

5.2. Implemented Pipelines

In order to gain better insight into the state of the tool (i.e., ladle) and take appropriate and timely actions when it comes to managing it, we developed pipeline elements for the following models:

- Deep Learning—Fully connected neural network model that takes low-frequency data as input and outputs the predicted tool state. Input data pass through a preprocessing step that is integrated into separate pipeline elements and is executed before the neural network.
- Statistical Analysis—Model that applies change point detection and a multivariate exponentially weighted moving average control chart on high-frequency data, detecting possible outliers and their root causes.
- Complex Event Processing—Model that applies complex event processing using Siddhi [20].
- Machine Learning—Partial least squares regression model that uses low-frequency data to infer whether or not the tool can be used without any incidents. This model requires a preprocessing step that should be integrated into a separate pipeline element.
- Physics based—Model that uses the laws of physics to calculate the values of parameters that are not being monitored, thus providing other models with additional information about the process.

The pipeline and the interactions of its elements are shown in Figure 9. Since these models have to conform to StreamPipes' internal mechanisms, they can easily be combined within pipelines. StreamPipes internally manages all data transfers and is transparent when it comes to the logic of individual components. This way, it is possible to realize complex hybrid DT models from different independent DT models.

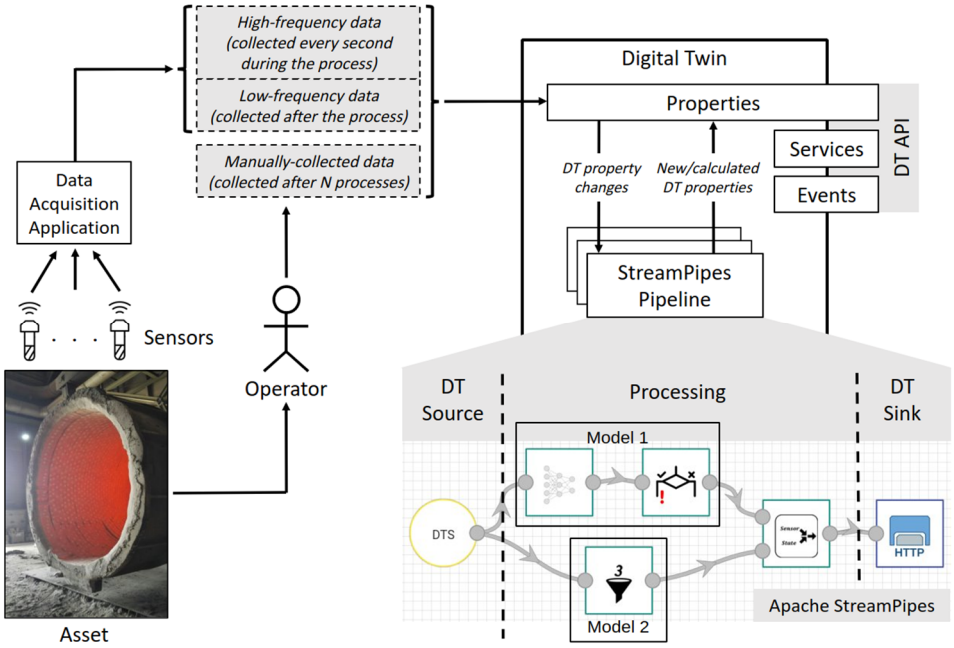

Figure 9. Hybrid digital twin for the steel production use case.

The role of hybrid modeling is to improve the accuracy of predictions made by separate models, as follows:

- The physics-based model calculates the thickness of the brick walls in each heat. We note here that there is no such value in the original dataset.
- The deep learning model uses this new information provided by the physics-based model as an additional input for the training process and generates more accurate predictions on whether the ladle can be used in the next run. It should be emphasized that without this additional data, the algorithm was not able to make an accurate generalization of the prediction model, since the model was under-fitted.

Figure 9 shows a high-level overview of the steel production use case including the required components and their interaction. The state of the asset is monitored with sensors located at the facility. This state is represented by high-frequency data (collected every second during the process), low-frequency data (collected after the process), and manually collected data (collected after N processes). Sensor readings are collected using the Data Acquisition Application, which then updates the state of the DT using the DT API (DT property update).

These data can be polled by an operator who wants to see the process information or used by other StreamPipes pipeline services (DT services). These services represent processing components that use available data to calculate/infer new information regarding ladle usage. Access to the raw sensor data and new information is provided by the DT and can be used by the operator. These DT properties answer the operator's questions regarding the process, such as "What is the ladle/brick state?", "Should the ladle be repaired?", "How many more heats can the ladle last?", etc., and provide general information about the ongoing heat process—all through the DT API.

Figure 9 also shows an example pipeline that combines different independent models into a complex hybrid one. Pipeline elements can be added or removed, thus creating new relations and changing the pipeline logic, as long as the new pipeline conforms to the semantics of the used elements. A greater number of different models, along with their proper integration, provides better insight into the monitored processes. This, in turn, provides a better understanding and appropriate actions when it comes to their management.

6. Conclusions and Outlook

The main idea of DTs is to solve an information silo problem by combining isolated data in a semantically consistent manner by ensuring an integrated, unified view of the data and information. In addition to the data, the DTs might contain multiple models that are of different types and usually independent of each other. There are many application scenarios, especially in the process industry domain, where data are not available/reliable or the models are not precise. To gain comprehensive insight, it is necessary to support the collaboration of multiple models and corresponding data.

To be able to deal with imprecise models and incomplete data, in this paper, we propose a new approach for hybrid DTs. The proposed approach is based on I4.0-conforming DTs integrated with Apache StreamPipes. A DT exposes a well-defined DT API offering access to properties, services, and events. The integration of models, e.g., ML based or physics based, is realized by expressing the models as StreamPipes pipelines. Each pipeline can collect relevant data about the state of the DT over time and continuously output new virtual/calculated properties that will be made available via the DT API to the outside world. The proposed solution resolves the interoperability issues at the level of the DT models, as it offers a standardized way of integrating different models. The usage of the AAS specification ensures interoperability of the DT model and API.

The strategy to orchestrate DT models through the use of StreamPipes and I4.0-compliant DTs might not be optimal in all situations. The proposed solution is primarily aimed at supporting (I)IoT and telemetry data and structured data, including APIs for services and events. It is not aimed at supporting media data such as video and camera data (RGB, Infrared, HF laser, etc.). Furthermore, it is also not aimed at supporting natural language processing with speech/audio and/or complex graph structures. Further extensions or alternative solutions may be needed to support such data types.

Author Contributions: Conceptualization, L.S., N.S., and M.J.; data curation, B.J.; software, B.J. and M.J.; supervision, L.S.; visualization, B.J. and M.J.; writing—original draft preparation, B.J., L.S., N.S., and M.J.; writing—review and editing, B.J., L.S., and M.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partly funded by the H2020 COGNITWIN project, which has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 870130.

Acknowledgments: We would like to thank the StreamPipes community and especially Dominik Riemer and Philipp Zehnder for their support.

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

References

1. Industrial Internet Consortium, Digital Twins for Industrial Application, an Industrial Internet Consortium White Paper. Available online: https://www.iiconsortium.org/pdf/IIC_Digital_Twins_Industrial_Apps_White_Paper_2020-02-18.pdf (accessed on 20 March 2021).
2. Antunes do Carmo, J.S. *Physical Modelling vs. Numerical Modelling: Complementarity and Learning*; Preprints: Basel, Switzerland, 2020. [CrossRef]
3. *Hybrid Modeling in Process Industries*; Glassey, J.; von Stosch, M. (Eds.) CRC Press: Boca Raton, FL, USA, 2020; ISBN 9780367572228.
4. Lin, P.; Yuan, X.X.; Tovilla, E. Integrative modeling of performance deterioration and maintenance effectiveness for infrastructure assets with missing condition data. *Comput. Aided Civ. Infrastruct. Eng.* **2019**, *34*, 677–695. [CrossRef]
5. Abburu, S.; Roman, D.; Berre, A.; Stojanovic, L.; Jacoby, M.; Stojanovic, N. COGNITWIN–Hybrid and Cognitive Digital Twins for the Process Industry. In Proceedings of the IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC), Cardiff, UK, 15–17 June 2020. Available online: https://doi.org/10.1109/ICE/ITMC49519.2020.9198403 (accessed on 20 March 2021).
6. Hamilton, F.; Lloyd, A.L.; Flores, K.B. Hybrid modeling and prediction of dynamical systems. *PLoS Comput. Biol.* **2017**, *13*, e1005655. [CrossRef] [PubMed]
7. Apache StreamPipes. Available online: https://streampipes.apache.org/ (accessed on 20 March 2021).
8. Details of the Asset Administration Shell: From Idea to Implementation. Plattform Industrie 4.0, Ed. Available online: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/vws-in-detail-presentation.pdf (accessed on 20 March 2020).

9. Jacoby, M.; Usländer, T. Digital Twin and Internet of Things—Current Standards Landscape. *Appl. Sci.* **2020**, *10*, 6519. [CrossRef]
10. Stojanovic, L.; Bader, S.R. Smart Services in the Physical World: Digital Twins. In *Smart Service Management: Design Guidelines and Best Practices*; Maleshkova, M., Kühl, N., Jussen, P., Eds.; Springer International Publishing: Cham, Germany, 2020; pp. 137–147. [CrossRef]
11. Plattform Industrie 4.0—Digital Twin and Asset Administration Shell Concepts and Application in the Industrial Internet and Industrie 4.0 (Plattform-i40.de). Available online: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Digital-Twin-and-Asset-Administration-Shell-Concepts.html (accessed on 12 May 2021).
12. Shohin, A.; Xun, X.; Ray, Y.; Zhong, Y.L. Digital Twin as a Service (DTaaS) in Industry 4.0: An Architecture Reference Model, Advanced Engineering Informatics. *Adv. Eng. Inform.* **2021**, *47*, 101225. [CrossRef]
13. Sepasgozar, S.M.E. Differentiating Digital Twin from Digital Shadow: Elucidating a Paradigm Shift to Expedite a Smart, Sustainable Built Environment. *Buildings* **2021**, *11*, 151. [CrossRef]
14. DIN SPEC 91345:2016-04, Reference Architecture Model Industrie 4.0 (RAMI4.0). Available online: https://www.beuth.de/en/technical-rule/din-spec-91345/250940128 (accessed on 20 March 2021).
15. Platform Industrie 4.0, Details of the Asset Administration Shell—Part 1. Available online: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.pdf?__blob=publicationFile&v=5 (accessed on 20 March 2021).
16. Platform Industrie 4.0, Details of the Asset Administration Shell—Part 2. Available online: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part2_V1.pdf?__blob=publicationFile&v=6 (accessed on 20 March 2021).
17. Node Red. Available online: https://nodered.org/ (accessed on 8 April 2021).
18. Introducing Node-RED 1.0. Available online: https://nodered.org/ (accessed on 26 April 2021).
19. I4.0 Language. Available online: https://www.i40.ovgu.de/i40/en/I4_0+language-p-48.html (accessed on 8 April 2021).
20. Siddhi—Cloud Native Stream Processor. Available online: https://siddhi.io/ (accessed on 26 April 2021).