ISSN 2281-4299

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Realizing Smart Manufacturing Architectures through Digital Twin Frameworks

David Ghedalia
Francesco Leotta
Massimo Mecella

Technical Report n. 2, 2020

Realizing Smart Manufacturing Architectures through Digital Twin Frameworks

David Ghedalia ghedalia.1655277@studenti.uniroma1.it
Francesco Leotta leotta@diag.uniroma1.it
Massimo Mecella mecella@diag.uniroma1.it

June 22, 2020

Abstract

The recent advances in communication and computation technologies and the explosion of the Internet-of-Things (IoT) are the fundamental building blocks of the smart manufacturing approach. Here, "things" are the main actors of production processes and supply chains, i.e., involved machinery and companies accessible through their so called digital twins - DTs. DTs are faithful representations of physical entities in the virtual world, which, by exposing services, can be employed to modify, moni-tor and predict the state of the wrapped objects. The interest in smart manufacturing from industry and institutions led the development of commercial and open source frameworks to implement DTs. In this paper, we will discuss how these frameworks can be employed in consistent architectures for smart manufacturing aiming at coordinating DTs in order to pursue specific production goals.

keywords

Industry 4.0, digital twins, smart manufacturing, development framework, software

1 Introduction

The continuous evolution of technologies in the fields of communication, networking, storage and computing, applied to the more traditional world of industrial automation, in order to increase productivity and quality, to ease workers' lives, and to define new business opportunities, has created the so-called *smart manufacturing* or *Industry 4.0*.

A *digital factory* aims at using digital technologies to promote the integration of product design processes, manufacturing processes, and general collaborative business processes across factories. An important aspect of this integration is to ensure interoperability between machines, products, processes, and services. A digital factory consists of a multi-layered integration of various activities along the factory and related resources (including information). Actors can fall in different categories, being humans (i.e., final users or participants in the production process), information systems or industrial machines. These physical entities must have a faithful representation in the digital world, usually defined *digital twins*.

A digital twin (DT) exposes a set of *services* allowing to execute certain operations and produce data describing its activity. Indeed modern information

systems and industrial machines may natively come out with their digital twin; in other cases, especially when the approach is applied to already established factories and production processes, digital twins are obtained by wrapping actors that are already in place.

Properties of DTs. DTs are commonly known as a key enabler for the digital transformation in manufacturing, however, in the literature, there is no common understanding concerning this term. Different definitions agree on features such as *(i)* connectivity, i.e., the ability to communicate with other entities and digital twins, *(ii)* autonomy, i.e., the possibility for the DT to live independently from other entities, *(iii)* homogeneity, i.e., the capability, strictly connected to the autonomy, that allows to use the same DT regardless of the specific production environment, *(iv)* easiness of customization, i.e., the possibility to modify the behavior of a physical entity by using the functionalities exposed by its DT, and *(v)* traceability, i.e., the fact that a DT leaves traces of the activity of the corresponding physical entity.

Our contribution. Due to the large interest in digital twins, recently some frameworks for their description and development have been proposed as commercial or open source frameworks. In this paper, we review some of those frameworks, by specifically highlighting the interesting features that can be leveraged in order to build real smart factories. We also propose a conceptual architecture in which DTs are dynamically composed in order to achieve specific productions goals, with adaptive features built-in. We finally discuss how the frameworks can be adopted in order to build such an architecture.

Outline. The following of this paper is as it follows. Section 2 presents an overview of DTs in the scientific literature, whereas Section 3 introduces our architecture for smart factories. Section 4 introduces the available frameworks for DTs, by presenting their technical features. Section 5 discusses how to realize our architecture through available frameworks.

2 Digital Twins in Smart Manufacturing

DTs monitor and control physical entities, where physical entities send data to update what are commonly referred to as the *virtual models* [3, 6].

Authors in [1] introduce a DT reference model for a cloud-based cyber-physical system, such as a smart city. The model is based on a smart interaction controller using a Bayesian belief network. They divide the system into three operational modes, namely *(i)* physical level sensors-fusion mode, *(ii)* cyber-level digital twin services-fusion mode and *(iii)* a deep integration of sensor-services fusion mode. They provide a context-based control decision scheme that uses Bayesian networks and fuzzy logic based rules to select any of these system modes for inter-system interactions.

Authors in [12] introduce a novel architecture for large-scale DT platforms including a distributed DT cooperation framework, flexible data-centric communication middleware, and the platform-based DT application to develop a

reliable advanced driver assistance system.

In the the traditional manufacturing process, the production plan is generated based on the new and historical orders. Then the preparation for production is carried out, such as equipment maintenance and material collection. Finally, formal production is executed according to the plan. If some conflicts appear, the plan is modified to adapt to the actual situations. After production, finished products are inspected to ensure whether they meet requirements, and consequently transported into the warehouse or repaired. Information generated during production such as process documents and fault records are kept in files for the next round. The work of [9] provides a conceptual model and specific operation mechanisms for a DT *shop-floor*, that is, a basic unit of manufacturing, where data from both physical and virtual sides as well as the fused data are provided to drive all the steps of the production process.

Authors in [10] introduce a DT-based system for the waste electrical and electronic equipment (WEEE) recovery to support the manufacturing/remanufacturing operations throughout the product's life cycle. The system is based on a Service Oriented Architecture (SOA), in which both the electronic device and its DT are considered as the services. The DT plays as the bridge between the physical world and the cyber world. As the product moves from the beginning-of-life to middle-of-life and end-of-life, the knowledge and information maintained inside its DT becomes bigger and richer. The digital world is supported and hosted by a cloud computing environment. Computer-aided engineering modules help to simulate the performance to validate and improve the product design. During manufacturing processes, the operations are controlled by the manufacturing execution system. With the help of multiple enablers, e.g., smart monitors, wireless sensors and other IoT devices, the DT is strengthened by the related operation and product performance data collected from the factory. After an end user purchases the product, she can interact with the DT via multiple enablers, e.g., a RFID reader and near-field communication, which are affordable devices for general usage.

Authors in [11] provide an overview of Industry 4.0 features in multiple reference architectures and develops a maturity model for IT architectures for data-driven manufacturing. The group around Reference Architecture Model Industry 4.0 – RAMI – developed a detailed conceptual architecture and a first implementation of the digital twin, the open Asset Administrative Shell. It consists of an operating system and can be integrated into a SOA. RAMI is explicitly defined as a SOA, relying on functionalities encapsulated into services. As an example, RAMI provides access to an administrative shell through a service interface and a common data format. More in details, the digital twin in the Industrial Internet Reference Architecture is located in the entity abstraction layer of the control domain. It directly builds upon of the communication layer and provides an abstraction of sensor values and actuators.

Authors in [13] introduce a semantic schema representation that categorises industrial data streams in DTs, allowing subscribers to consume similar data from multiple assets within a single data analytics pipeline, and paving the way for a more intuitive management of digital twin representations from industrial

assets.

In order to execute operations between products, processes and resources, authors in [4] propose the usage of the popular Automation Markup Language (AutomationML), for modelling a structural parts machining cell. AutomationML stores engineering information following an object-oriented paradigm and allows to model physical and logical components as data objects. An object may consist of other sub-objects, and may itself be a part of a larger composition or aggregation.

Authors in [2] model digital factories as the DTs of physical facilities, replicating the facility in terms of installed machinery, material handling equipment, and layout. The digital factory is supported by a formal ontology and a mathematical model for quantifying the processing capabilities. Manufacturing Service Description Language (MSDL) is a descriptive ontology developed for representation of capabilities of manufacturing services. MSDL decomposes the manufacturing capabilities into five levels of abstraction, namely, supplier-level, shop-level, machine-level, device-level, and process-level. A unique feature of MSDL is that it is built around a service-oriented paradigm, therefore, it can be used for representing a manufacturing system as a collection of manufacturing services. MSDL was initially designed to enable automated supplier discovery in distributed environments with focus on mechanical machining services. However, to address a wider range of services offered by small and medium-sized suppliers in contract manufacturing industry, the service ontology can always be extended systematically through active involvement of a community of ontology users.

Some problems affecting product data management and application in Product Lifecycle Management (PLM) exist as follows: *(1)* due to the different purposes and tasks, the data generated in various phases of the entire product lifecycle may form information islands between different phases of the product lifecycle; *(2)* there is a lot of duplicate data in different phases of product lifecycle. These duplicate data may cause wasting of resources and data sharing problem; *(3)* the interaction and iteration between the so-called *big data analysis* and various activities in the entire product lifecycle are nowadays relatively absent; *(4)* the current applications of big data prefer to put emphasis on the analysis of physical product data rather than the data from virtual models.

Authors in [7] use AutomationML (analogously to the already mentioned work of [4]) to model attributes of a DT, and discuss its application towards the data exchange between different systems that are connected with the DT itself.

Authors in [8] discuss more in general how to generate and use converged DT data to better serve product lifecycle, so as to drive product design and manufacturing.

3 Architectural Model

In this paper, we will show how existing digital twins development and deployment frameworks can be employed in order to implement the Industry 4.0

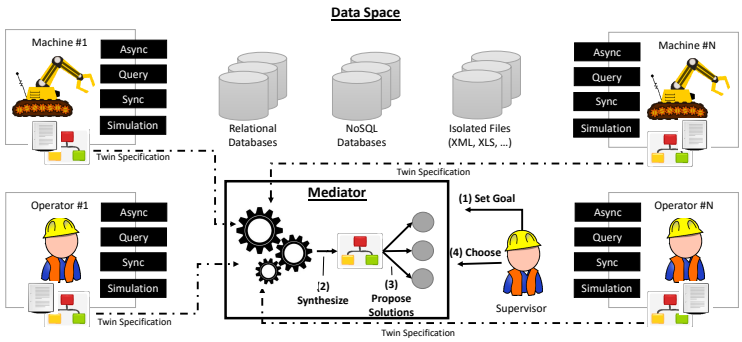

Figure 1: The proposed architecture.

reference architecture introduced in [5]. Here, authors propose an architecture for a smart manufacturing process based on DTs as depicted in Figure 1, where the main components are the DTs, the data space, human supervisors and a mediator.

DTs wrap physical entities involved in the process. These physical entities can be manufacturing machines or human operators. A DT exposes a Web API consisting, in general, of three parts: the synchronous one, the query interface and the asynchronous one. The synchronous interface allows to give instructions to the physical entity. These instructions may, for example, produce a state change in a manufacturing machine (in case the twin is over a machine) or ask a human operator to perform a manual task (in case the twin is over a manufacturing worker). The query interface allows for asking information to the physical entity about its state and related information; noteworthy, these latter can be obtained by applying diagnostic and prognostic functions results of machine learning. The asynchronous interface generates events available to subscribers.

It is important to note how manufacturing machines and human actors can be considered identical from the point of view of the offered API, e.g., human actors produce asynchronous events as well, for example generating alarms.

In such contexts, data are usually available through several sources not sharing a common schema and vocabulary, as DTs come from different vendors. It is reasonable to consider an heterogeneous data space where a mediator is present and it also takes the role of translator between different formalisms and access methods.

We consider learning as a fundamental feature of DT. Learned functions include the automatic generation of alarms, but also automatic triggering of actions and status changes. Additionally, twins can be queried on learned functions and, as a result, the data space is far more dynamic than in more traditional scenarios.

As in modern micro-services architectures, notifications based on publish&subscribe is a common architectural pattern, and therefore we provide

for subscriptions to events generated by other DTs.

The data space contains all the data available to the process. These data are heterogeneous in their nature from the access technology point of view, the employed schema (or its absence) and the employed vocabulary. It is important to note how the DTs contribute to the data space with both the query API and the asynchronous one. Other sources for the data space may include relational and no-SQL databases or unstructured sources such as spurious files, which constitute the factory information system.

The human supervisor is the one defining the goals of the process in terms of both final outcomes and key performance indicators to be obtained.

In order to reach the goal defined by the human supervisor, available twins and data must be integrated. This task is fulfilled by the mediator. The mediator acts in two phases: the *synthesis phase* and the *execution phase*. During the synthesis phase, the specifications of the APIs exposed by digital twins and the meta-data (e.g. data source schemas) available in the data space, are composed in order to construct a *mediator process*. During the execution phase, the mediator runs its program by preparing the input messages for the single twins involved in the proper sequencing/interleaving. Indeed, as each twin may potentially adopt a different language and vocabulary, in order to compose required input/output messages, the mediator translates and integrates the data available in the data space to comply with the format requested by the specific called service.

An important aspect of the proposed architecture is that multiple companies can participate in the process (typically those ones involved in the value chain). Again, it is not reasonable to have twins directly communicating with one another. Once again, the role of the mediator is fundamental, being the component that can access the services offered by the twins available in the different companies.

4 Technologies for realizing digital twins

Vendors and open source foundations started providing support to several aspects connected to digital twins including twin specification, implementation, hosting and directory services. Nevertheless there are still missing features, including native support to simulation and data integration.

4.1 Eclipse Ditto

Eclipse Ditto[1] is an open-source platform to implement the interaction patterns of digital twins. In particular, it provides support to *(i)* define APIs abstracting the physical entity behind a twin, *(ii)* route requests between hardware and clients, *(iii)* manage access control policies, *(iv)* caching, and *(v)* notifications.

[1] cfr. `https://www.eclipse.org/ditto`

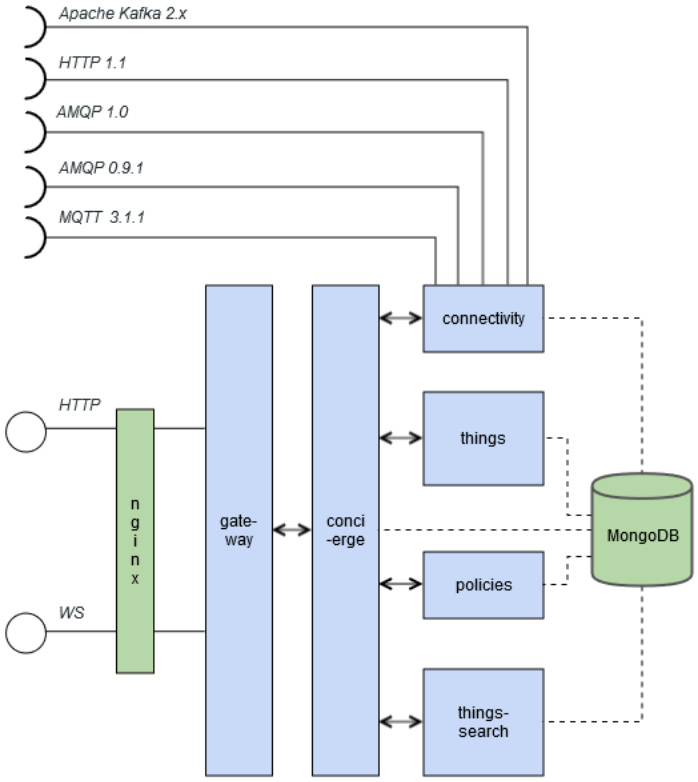

Figure 2: Architecture of Eclipse Ditto

For Eclipse Ditto the digital twin is an abstraction of a real world asset/device with all capabilities and aspects including its digital representation.

Figure 2 shows the architecture of Eclipse Ditto. A single instance of Ditto can host different twins, which are representations of physical entities called *things*. Things communicate with Ditto by using different kinds of protocols including asynchronous ones (Kafka, AMQP and MQTT) and synchronous ones (HTTP). Same protocols can be also used by clients to query things and execute operations. A MongoDB server instance stores not only data coming from things but also things definitions and access policies. The orchestration of different components is provided by a specific component called *concierge*. Access to Ditto functionalities is provided to clients through HTTP and WebSockets API exposed by an nginx web server.

Each thing definition in Ditto is identified by a `thingId` and contains an access policy, a definition identifier (multiple things can comply to the same definition) and a set of static (or infrequently changed) attributes (things metadata). According to the specific definition, a twin can expose different features. Differently from attributes, features represents state data. Attributes and fea-

tures must be coherent with the twin definition. Ditto supports definition informally, and it does not provide any compliance checking feature. In order to formally define things class, the Eclipse Foundation provides Vorto, which is a twin specification language.

A thing is represented in the MongoDB persistence service as a JSON document. Noteworthy, identifiers in a specific thing JSON document, are organized in namespaces, in order to avoid collisions. An example follows:

```
{
  "thingId": "my.namespace:myFridge",
  "policyId": "my.namespace:myFridgePolicy",
  "definition": "digitaltwin:DtExample:1.0.0",
  "attributes": {
    "location": "Kitchen"
  },
  "features": {
    "temperature": {
       "properties": {
         "cur_temp": 5
       }
     }
  }
}
```

The communication between clients, i.e., external apps and coordination services, can happen following two different modalities that are shown in Figure 4.1.

In the first mode, called *twin channel*, the communication between clients and things is mediated by Ditto. In the second mode, called *live channel*, the communication between the clients and the things is direct, with Ditto only performing authorization checks. In the latter case, the clients must be aware of the specific communication protocols implemented on the thing side.

4.2 Eclipse Vorto

Eclipse Vorto is a platform whose aim is to define twin specifications and to automatically generate code (e.g., based on the Eclipse Ditto framework). It makes use of information models, that are assembled by abstract and technology-agnostic function blocks. In addition, the Vorto project [2] provides the community with a freely available repository of definitions and code. Vorto provides integration libraries and tools to avoid tight coupling of devices in IoT solutions: they allow to consume device descriptions to integrate devices with various IoT platforms and solutions. Therefore, it can serve as an abstraction layer over different devices, allowing uniform communication despite the diverseness of the protocols.

[2] cfr. `https://www.eclipse.org/vorto/`

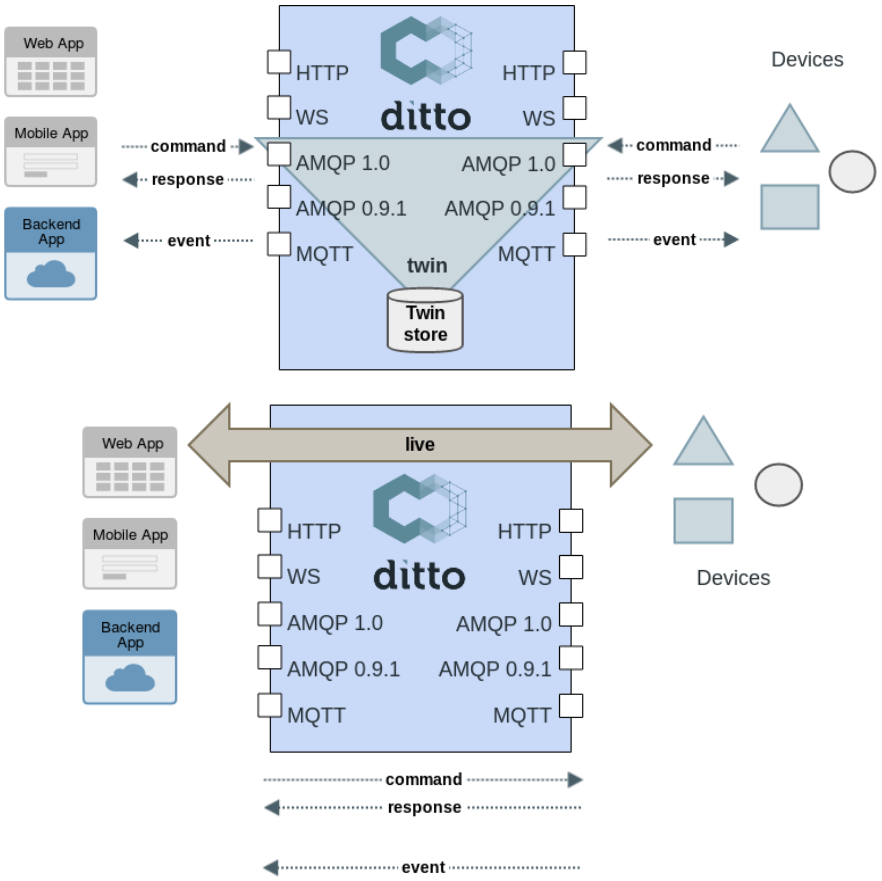

Figure 3: Ditto Twin and Ditto Live channels

Digital twins are described using *vortolang*, which is a domain-specific language. The vortolang is made up of a set of metamodel classes defining the capabilities of digital twins. There are two top-level classes, Information Model and Function Block, that describe a digital twin and the capabilities of digital twins, respectively. There are three metamodel classes that describe capabilities: Properties, Events, and Operations.

An Information Model describes a complete digital twin, such as a physical device and defines the set of interfaces as Function Blocks, implemented by the digital twin.

A Function Block describes related capabilities that are implemented by a digital twin. Function Blocks are reusable and can be reused across different Information Models. A Function Block is composed of properties, events and operations.

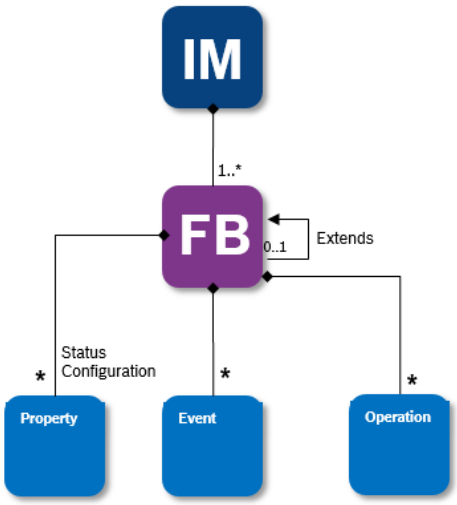

Figure 4: Metamodel classes

- Properties describe both read-only and read-write state of a digital twin. Read-only properties are defined as status properties, whereas read-write properties as configuration properties
- Events define data that is emitted by the device or entity (from device to solution)
- Operations represent a function that can be performed on a Digital Twin (from solution to device). Operations may have request parameters as well as a return type

For example, a board equipped with an altitude sensor could be represented by the following information model:

```
vortolang 1.0
namespace vorto.private.myThings
version 1.0.0
using com.bosch.iot.suite:Altitude ; 1.1.0
infomodel myBoard{
  functionblocks{
    mandatory altitude as Altitude
  }
}
```

Here, the information model refers to the following function block from the Vorto repository:

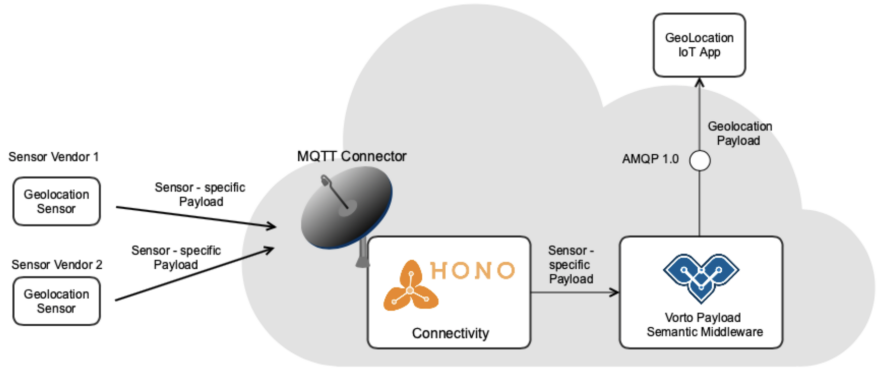

Figure 5: Example of the architecture implementing the Semantic Normalizer Middleware

```
vortolang 1.0
namespace com.bosch.iot.suite
version 1.1.0
functionblock Altitude {
    configuration {
        optional currentCalibration as string
        with {readable: true, writable: true}
        ...
    }
    status {
        mandatory sensorValue as float
        with {readable: true, writable: false}
        optional sensorUnits as string
        with {readable: true, writable: false}
        ...
    }
    operations {
        resetMinandMaxMeasuredValues()
    }}
```

As can be seen, a full set of primitive data types, along with support for a variety of complex types (in the form of enumerations, dictionaries, and entities), are provided and can be specified directly as a type value of a model property.

Eclipse Vorto includes a service that performs payload mapping, that allows IoT solutions to become de-coupled from the plethora of different device data formats. In the example shown in Figure 5[3], there two distinct geolocation sensors transmit data in different formats annd the Vorto Payload Semantic Middleware will convert the received data to a unified format and forward it to

[3]cfr. `https://github.com/eclipse/vorto-examples/blob/master/vorto-middleware/Readme.md`

a back-end.

4.3 Bosch IoT Things

Bosch IoT Things[4], is a cloud-based solution to host Ditto-based digital twins. Anyway, Bosch IoT Things is not a mere cloud wrapper for Ditto, as it provides additional out of the box features such as:

- Built-in integration with other Eclipse IoT projects (e.g., Eclipse Hono, Eclipse hawkBit)
- A web user interface for monitoring and managing twins
- The possibility to extend the functionalities of Eclipse Ditto with other services belonging to the Bosch IoT suite (for example, Bosch IoT Permissions to manage users and permissions)

Figure 6 shows an example deployment of the multiple actors which can access the digital twin (in the example, an ebike). Connectivity to devices can be handled by Bosch IoT Hub, and then bridged to Bosch IoT Things. Another possibility is to build a microservice, which wraps one of the Ditto supported protocols, to directly communicate with the thing.

4.4 AWS IoT: Device Shadow Service

The Device Shadow Service[5] is provided as part of the AWS IoT platform. It is a JSON document that is used to store and retrieve current state information for a device. The Device Shadow service maintains a shadow for each device connected to AWS IoT, where a shadow is used to get and set the state of a device, regardless of whether the device is connected to the Internet.

The Device Shadow stores a state, which is composed by two properties:

- Desired: the desired state of the thing. Applications can write to this portion of the document to update the state of a thing without having to directly connect to a thing.
- Reported: the reported state of the thing. Things write to this portion of the document to report their new state. Applications read this portion of the document to determine the state of a thing.

It is possible to interact with the Device Shadow with:

- MQTT protocol: there are reserved topics for each possible interaction (update, get, delete). It is adopted for the counterpart of the device's shadow
- HTTPS REST API (GET, POST, DELETE methods): it is used by the web or mobile application

[4] cfr. `https://things.eu-1.bosch-iot-suite.com`

[5] cfr. `https://docs.aws.amazon.com/iot/latest/developerguide/iot-device-shadows.html`

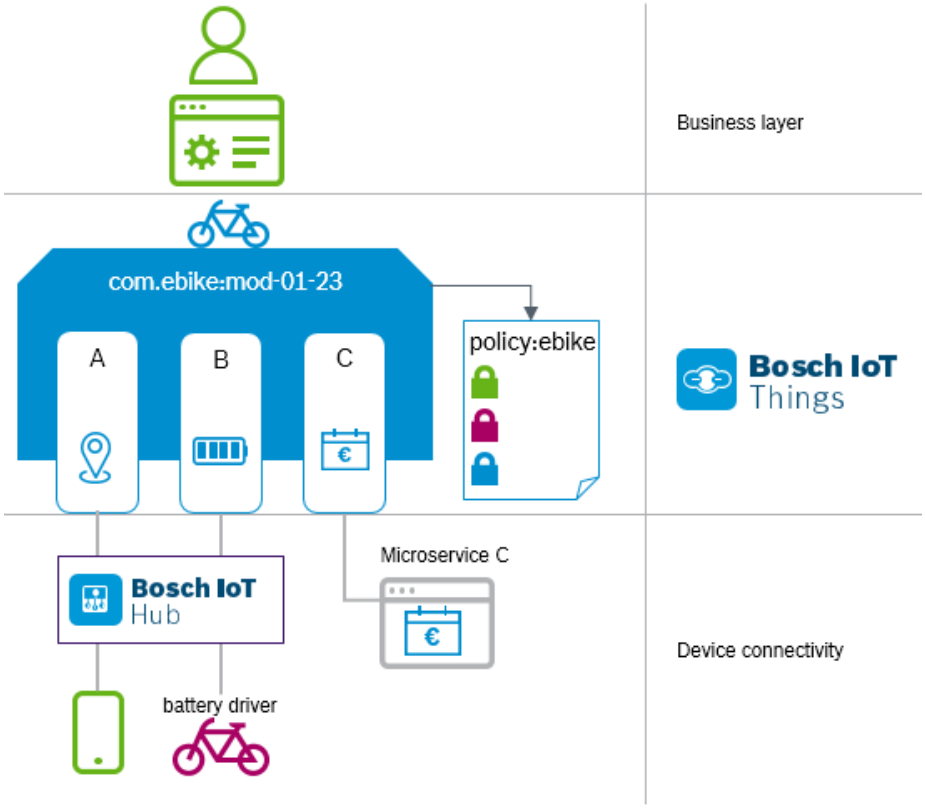

Figure 6: Bosch IoT Things architecture

4.5 Azure Digital Twins

Azure Digital Twins[6] is an IoT service (part of the Azure cloud services) whose focus is to create models of physical environments with spatial intelligence graphs, a technique to represent the relationship between people, places and devices in a hierarchical manner (see Figure 7).

Spatial intelligence graphs are populated by instances of object models, which describe domain-specific concepts, categories, and properties. The most relevant object models are the following:

- Spaces: virtual or physical locations (for example, Tenant, Customer, Region)
- Devices: virtual or physical pieces of equipment (for example, a Raspberry Pi)
- Sensors: objects that detect events

[6] cfr. `https://docs.microsoft.com/en-us/azure/digital-twins/`

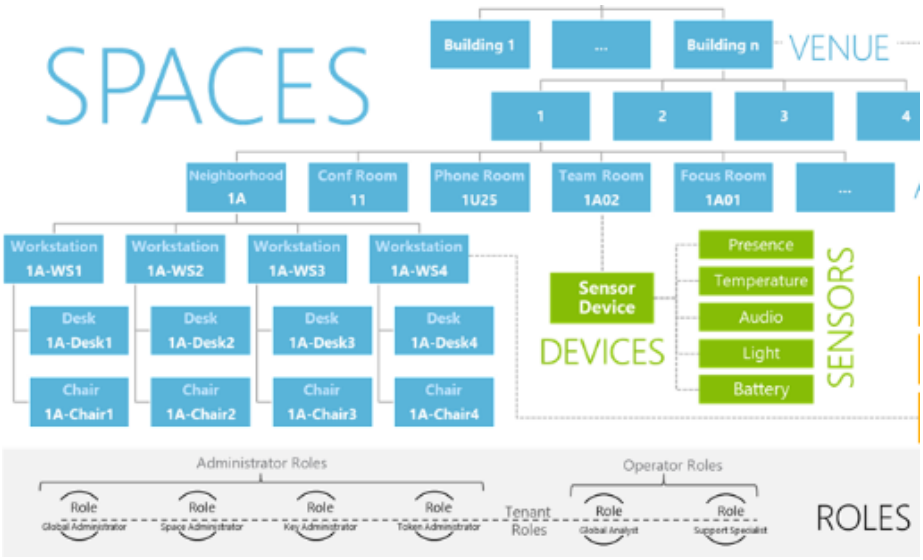

Figure 7: Example of spatial intelligence graph

- Users: they identify occupants and their characteristics
- Roles: sets of permissions assigned to users and devices in the spatial graph (for example, Space Administrator, User Administrator and Device Administrator)
- Role assignments: the associations between roles and an objects in the spatial graph. They enable precise access control over specific data, resources, and actions in the spatial graph
- User-defined functions (UDF): they define custom functions that run against incoming data from devices in order to send signals to predefined endpoints, allowing automation of device tasks
- Matchers: objects that determine a set of conditions for which the user-defined functions are triggered. For each UDF, at least a matcher must be assigned

When a Digital Twins service is deployed in the personal subscription, the subscription's owner becomes the global administrator of the root node and full access to the entire structure is automatically granted. To interact with a spatial graph, a collection of REST APIs is provided. Besides providing basic methods to interact with the graph, other useful features are offered:

- Graph filtering: it is used to narrow down request results. It is possible to filter by IDs, name, types, subtypes, parent space, and associated spaces

- Graph traversing: it is possible to traverse the spatial graph through its depth and breadth
- Graph extensibility: it is possible to customize the underlying Digital Twins object models with new types and ontologies

When designing the spatial graph, graph inheritance must be kept in mind: inheritance applies to the permissions and properties that descend from a parent node to all nodes beneath it. For example, when a role is assigned to a user on a given node, the user has that role's permissions to the given node and every node below it. Each property key and extended type defined for a given node is inherited by all the nodes beneath that node.

4.6 IoT Plug and Play

IoT Plug and Play is part of the Azure platform and provides almost the same services of Eclipse Vorto, that are:

- A language to describe digital twins
- A repository containing IoT Plug and Play ready devices (Azure IoT Central)
- Tools and libraries to integrate devices without writing any embedded code

Digital twins are described using the Digital Twin Definition Language[7], which makes use of a variant of JSON called JSON-LD. JSON-LD is designed to be usable directly as JSON as well as usable in Resource Description Framework (RDF) systems. There are two top-level classes, CapabilityModel and Interface, that describe digital twins and the capabilities of digital twins, respectively (corresponding to Information Model and Function Block in Eclipse Vorto). There are three metamodel classes that describe capabilities: Property, Telemetry and Command (corresponding to Property, Event and Operation in Eclipse Vorto). One of the few substantial differences with Eclipse Vorto is the possibility to provide semantic type annotations of capabilities, so that analytics, machine learning, UIs, and other computation can reason about the semantics of the data and not just the schema of the data.

A capability model describes a device and defines the set of interfaces implemented by the device. For example a board equipped with an altitude sensor could be represented by the following description:

```
{
    "@id": "urn:example.com:myBoard",
    "@type": "CapabilityModel",
    "implements": [
        {
```

[7] cfr. `https://github.com/Azure/IoTPlugandPlay/tree/master/DTDL`

```
            "name": "altitude",
            "schema": "urn:example:altitude:1"
        }
    ],
    "@context": "http://azureiot.com/v1/
            contexts/IoTModel.json"
}
```

An interface describes related capabilities that are implemented by a device or digital twin. Interfaces are reusable and can be reused across different capability models. In the previous capability model, the following Interface has been used:

```
{
    "@id":"urn:example:altitude:1",
    "@type":"Interface",
    "displayName":"Altitude",
    "contents":[
        {"@type":"Property",
        "name":"currentCalibration",
        "writable":true,
        "schema":"string"},
        ...
        {"@type":"Telemetry",
        "name":"sensorValue",
        "schema":"float"},
        {"@type":"Telemetry",
        "name":"sensorUnits",
        "schema":"string"},
        ...
        {"@type":"Command",
        "name":"resetMinandMaxMeasuredValues",
        "commandType":"synchronous"}
    ]
}
```

An Interface is composed of properties, telemetries and commands:

- A Property describes the read-only and read-write state of a device or digital twin
- Telemetry describes the data emitted by a device or digital twin, whether the data is a regular stream of sensor readings or an occasional error or information message (from device to solution)
- A Command describes a function or operation that can be performed on a device or digital twin (from solution to device). Commands can be either synchronous and asynchronous

A full set of primitive data types, along with support for a variety of complex schemas (in the form of Arrays, Enums, Maps, and Objects), are provided and

can be specified directly as the value in a schema statement in a digital twin interface.

5 Discussion

Tools and frameworks introduced in Section 4 allows to implement most of the architecture described in Section 3. In particular, synchronous and asynchronous interfaces exposed by a digital twin are easily supported as frameworks, such as Eclipse Ditto, allows to communicate with the thing behind the twin in both fashions. Additionally, twin specification languages do exist, such as Eclipse Vorto, that allow to define classes of twins with their respective attributes, state variables and operations. Solutions can be deployed on premise, or in cloud. This latter possibility simplifies the scenario in which the process to be automatized spans across different organizations. One aspect that is usually neglected by available solution is support for simulation and prediction, which is a fundamental part of the architecture proposed in Section 3. If, on the one hand, simulation and prediction can be offered with similar channels as the other interfaces, on the other hand, such functionalities require to support virtual perspectives on the state of the device, which are not usually provided out of the box by available solutions.

To this aim Eclipse Ditto displays key advantages, compared to the other platforms as it *(i)* is an open-source project, *(ii)* offers a wide range of connectivity services, *(iii)* is highly customizable, *(iv)* embraces the concept of "Device-as-a-Service", which is especially relevant in a Digital Twin scenarios, and *(v)* is extensible with Eclipse Vorto seamlessly. However, the price to pay in exchange for the listed benefits is a steep learning curve and, therefore, a high barrier to entry for developers.

Bosch IoT Things alleviates the problem, by bringing together Eclipse Hono, Eclipse Ditto and Eclipse hawkBit. The trade-off for a more user-friendly environment is the subscription cost. Bosch IoT Things sets a limit to the number of API calls and managed data volume to the free version, besides which it is necessary to own a paid subscription.

Amazon's Device Shadow focuses on a specific scenario, in which there is a need of a cloud service that acknowledges the difference between a reported and an actual state and notifies it to the device of interest. Although it is both intuitive and efficient in what it performs, it does not allow a richer and more complex representation of devices, that is sometimes required in the case of digital twins.

As previously stated, Azure Digital Twin service is designed to build automatic tasks whose behaviour depends on the data queried from sensors. It is ideal in case there is a large amount of sensors that can be organized hierarchically, according to their physical location. Another advantage is the possibility to route telemetry messages and events to other Azure services (e.g., analytics, AI, and storage) to further exploit IoT data.

content...

References

[1] K. M. Alam and A. El Saddik, "C2ps: A digital twin architecture reference model for the cloud-based cyber-physical systems," *IEEE Access*, vol. 5, pp. 2050–2062, 2017.

[2] F. Ameri and R. Sabbagh, "Digital factories for capability modeling and visualization," in *IFIP International Conference on Advances in Production Management Systems*. Springer, 2016, pp. 69–78.

[3] R. Baheti and H. Gill, "Cyber-physical systems. the impact of control technology, t. samad and am annaswamy," *IEEE Control Systems Society*, vol. 1, 2011.

[4] J. Bao, D. Guo, J. Li, and J. Zhang, "The modelling and operations for the digital twin in the context of manufacturing," *Enterprise Information Systems*, pp. 1–23, 2018.

[5] T. Catarci, D. Firmani, F. Leotta, F. Mandreoli, M. Mecella, and F. Sapio, "A conceptual architecture and model for smart manufacturing relying on service-based digital twins," in *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, 2019, pp. 229–236.

[6] E. A. Lee, "Cyber physical systems: Design challenges," in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. IEEE, 2008, pp. 363–369.

[7] G. N. Schroeder, C. Steinmetz, C. E. Pereira, and D. B. Espindola, "Digital twin data modeling with automationml and a communication methodology for data exchange," *IFAC-PapersOnLine*, vol. 49, no. 30, pp. 12–17, 2016.

[8] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, and F. Sui, "Digital twin-driven product design, manufacturing and service with big data," *The International Journal of Advanced Manufacturing Technology*, vol. 94, no. 9-12, pp. 3563–3576, 2018.

[9] F. Tao and M. Zhang, "Digital twin shop-floor: a new shop-floor paradigm towards smart manufacturing," *Ieee Access*, vol. 5, pp. 20 418–20 427, 2017.

[10] X. V. Wang and L. Wang, "Digital twin-based WEEE recycling, recovery and remanufacturing in the background of industry 4.0," *International Journal of Production Research*, pp. 1–11, 2018.

[11] C. Weber, J. Königsberger, L. Kassner, and B. Mitschang, "M2DDM – a maturity model for data-driven manufacturing," *Procedia CIRP*, vol. 63, pp. 173–178, 2017.

[12] S. Yun, J.-H. Park, and W.-T. Kim, "Data-centric middleware based digital twin platform for dependable cyber-physical systems," in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 2017, pp. 922–926.

[13] P. Zehnder and D. Riemer, "Representing industrial data streams in digital twins using semantic labeling," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 4223–4226.