ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Dipartimento di Informatica - Scienza e Ingegneria
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

A Web-based approach for ecosystems of heterogeneous Digital Twins

Tesi di laurea in
Pervasive Computing

Relatore
Prof. Alessandro Ricci

Correlatori
Prof. Andrei Ciortea
Dott. Samuele Burattini

Candidato
Andrea Giulianelli

Quarta Sessione di Laurea
Anno Accademico 2022-2023

Abstract

The Digital Twin paradigm is growing in popularity in both academia and industry as an approach that can virtualize entities existing in the real world. In particular, Digital Twins refer to the ability to clone and mirror a Physical Asset during its life cycle through a software counterpart that consumers can exploit. Over the years, its wide applicability in different application domains resulted in the loss of uniformity in modeling and development, leading to the creation of vertical and closed silos. In recent years, some proposals from academia have tried to conceptualize an open and interoperable vision for the creation of Digital Twins ecosystems that mirror entire portions of reality. In particular, the Web of Digital Twins (WoDT) vision aims at the creation of open, distributed, and dynamic ecosystems of connected Digital Twins creating a cross-domain and cross-organization service-oriented layer. This thesis proposes a Web-based design of the Web of Digital Twins vision that, exploiting a uniform interface that leverages Web technologies, aspires to the creation of ecosystems of heterogeneous Digital Twins, implemented with possibly different technologies. Moreover, part of the objective is the creation of a compatibility layer towards other popular paradigms, such as the Web of Things. The Web-based Web of Digital Twins supports the composition of discoverable and navigable ecosystems of cross-domain and cross-organizational Digital Twins that can serve as an open and interoperable service layer for applications on top. The idea resulted in the drafting of two specifications, for the obtainment of WoDT-compliant ecosystems, and in the creation of an abstract architecture that supports their implementation. The feasibility of the design has been proved through a prototypical implementation and subsequently exemplified through a use-case scenario.

To my family.

Acknowledgements

First of all, I would like to sincerely thank the supervisor of this thesis, Prof. Alessandro Ricci. You have been a source of inspiration for me during my university career, and working with you during my thesis has been an honor for me. You always made me feel part of the working group and always gave voice to my doubts and ideas, supporting me, especially in the most difficult moments. I also thank you for making my experience at the University of St. Gallen possible and allowing me to go through one of the most challenging experiences of my life.

Then, I would like to thank Dott. Samuele Burattini for likewise being essential throughout this thesis. I am so grateful for all the support and guidance, especially during the first months when the idea was still rough. I wish you the best in your career, wherever it takes you.

I would like to thank all the people I met during my experience at the University of St. Gallen. First of all, Prof. Andrei Ciortea for welcoming me into the research group by providing me with everything I needed to work while feeling as much at home as possible. Thank you for all the interesting discussions and for sharing your knowledge and experience with me. Also, I would like to thank Prof. Simon Mayer and all the other people with whom I had the pleasure to work or chat during the working days, Jan, Ivan, Jessie, Lukas, Jannis, and Matteo.

This thesis is just the last piece of the puzzle of these university years, which I will always remember. They have been years full of fulfillment and tears, joys and sorrows, but I would do it all over again, the same way. A key part of this journey was my family, who supported and spurred me every day, believing in me under all circumstances. It is hard to find the right words to describe what it all means to me, but I want to thank you for teaching me what it means to love and be loved, to pursue a passion, to face difficulties and come out on top, to know how to fall and then get back up, and to always be there for each other. You were incredible during my experience abroad, and it is also thanks to you that I was able to carry it through, only you know what it meant to me. Thank you, Michi, for teaching me to face my fears by seeing them from a different point of view.

Finally, I would like to thank all the people who have been present during these years. First of all, I would like to thank Beatrice, for being by my side during this past year supporting me in everything, and giving me a place of peace. Thank you to Marco, my best friend, for being there after all these years. I wish you the best, you are the living example of never giving up, in front of anything. Finally, thank you to all my friends, including Maria, Davide, Giacomo, Andrea, and Angela.

Thank you all, it was great, and I hope more joy like this will come in the future.

Contents

List of Figures

Listings

Chapter 1

Introduction

In the last decade, the Digital Twin paradigm has been one of the most active research fields in both academia and industry, with already several systems and technologies based on this vision. Digital Twins refer to the ability to clone a Physical Asset through a software counterpart for its entire life cycle. This creates a digitalized version that can be exploited at the application level, mirroring and augmenting its capabilities.

The Digital Twin paradigm has been applied in various domains as an approach that can virtualize entities existing in the real world, creating software counterparts that provide services over them. Several Digital Twin-based solutions have been tested and adopted in many domains including Healthcare, Manufacturing, and Smart Cities, finding interesting approaches and contributing to the advancement of the supporting technologies.

However, the wide applicability of the paradigm, over the years, has led to the proliferation of different definitions and different supporting technologies, resulting in the creation of vertical silos that prevent the possibility of representing portions of reality digitalized and navigable by the applications.

Despite this, some visions coming from academia try to fully exploit the Digital Twin paradigm, seeing its value not only in the single Digital Twin but in the creation of entire ecosystems that represent and mirror entire portions of reality offering an interoperable and navigable interface. In particular, the *Web of Digital Twins* [44] vision aims at the creation of open, distributed, and dynamic ecosystems of connected Digital Twins creating a cross-domain and cross-organization service-oriented layer. Therefore, openness and interoperability are suggested as two fundamental building blocks of the modern Digital Twins.

This thesis aims to implement and contribute to the *Web of Digital Twins* vision through an approach that addresses heterogeneity in Digital Twins ecosystems, aspiring for ecosystems where Digital Twins are not created with custom technologies built around the Web of Digital Twins vision, but where they can use any existing technology suited for their use case and fit equally into the ecosystem. In this scenario, consumers can navigate ecosystems regardless of the underlying technologies by leveraging a uniform interface, an idea drawing from the design rationale behind the Web architecture. This thesis proposes an approach based on the Web architecture, standards, protocols, paradigms, and technologies for the creation of dynamic, open, and long-lived systems that foster a uniform interface in the ecosystems.

This thesis work is the result of a collaboration with the University of St. Gallen, specifically with the Interaction and Communication based Systems research group, experts in hypermedia systems. The thesis work was carried out entirely at the University of St. Gallen, which hosted me for the thesis period.

Thesis Structure Chapter 2 provides an overview of the state of the art on Digital Twins and an in-depth analysis of the *Web of Digital Twins* vision. Afterward, Chapter 3 describes in depth the objective and the idea that characterize the thesis work, and defines the requirements of the system. Chapter 4 provides an overview of the subset of the Web standards, protocols, and paradigms that have been selected, and compares the Digital Twin paradigm to the Web of Things. Subsequently, Chapter 5 presents the proposed design, for which a prototypical implementation and an example use case are provided in Chapter 6. Finally, Chapter 7 concludes this thesis by summarizing and discussing the main contributions and the future directions that can be studied and explored.

Chapter 2

Background

This chapter introduces the Digital Twins paradigm with an overview of the state of the art of the concept and with a more detailed description of the *Web of Digital Twins* vision, which is the one followed in this thesis.

2.1 State of the art on Digital Twins

This section provides the definition and the characteristics of the Digital Twin paradigm. Subsequently, the most popular technologies that implement the paradigm, and an overview of their limitations are presented.

2.1.1 Definitions and characteristics

The Digital Twin concept was created by Michael Grieves and presented at the University of Michigan. The idea goes back to 2002 as part of his ideal vision of Product Lifecycle Management systems [24] presented to the industry as "Conceptual Ideal for PLM". This conceptual model was used in the first PLM courses at the University of Michigan in early 2003, where it was referred to as the *Mirrored Spaces Model* [26]. Then, Michael Grieves in [25] and [26] formalizes the term Digital Twin, providing the first definition:

> *The Digital Twin is a set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level to the macro geometrical level. At its optimum, any information that could be obtained from inspecting a physical manufactured product can be obtained from its Digital Twin.* [26]

In the Grieves's vision, Digital Twins are seen as a concept and set of technologies used to aid manufacturing systems throughout the product life-cycle, even before the creation of the physical product. The manufacturing sector played a crucial role in shaping the definition of many aspects and characteristics of Digital Twins. It was also one of the first fields to test and verify the real applicability and advantages of Digital Twins in a large-scale and highly complex context. In fact, before Digital Twins and IoT, the only way to know the state of a physical object, a machine, or the company in general was to be in proximity to it in order to carry out analyses. So, the information was inseparable from the physical object. The objective of Digital Twins was already from the beginning to allow the representation of such information at the Digital level to enable Digital copies of real-world objects [25][26].

Initially, when the concept was presented by Grieves in 2003, the available technologies for modeling and mirroring physical objects were limited and immature. This slowed down the evolution and the research around Digital Twins.

The quick rise and development of communication technologies, simulation techniques, the spread of other paradigms like the Internet of Things (IoT), and the advances in Artificial Intelligence, allowed a renewed interest in Digital Twins with a vast set of implementations and definitions.

The initial vision was based on Grieves's experience in manufacturing. After applying the Digital Twin paradigm in various domains, including Healthcare, and Smart Cities, it started to seem possible to virtualize any physical object present in the real world. In addition to that, also Augmented and Virtual Reality, Virtualization, and Artificial Intelligence influenced and enriched the Digital Twin concept as we know it today [39].

This renewed interest allowed expanding the Digital Twin concept as a software entity that can model, represent, mirror, and augment the behavior, the state, and the functionalities of a Physical Asset [39].

As a result, it was needed a more general vision and a definition of the Digital Twin concept that was able to include all these new visions and ideas. *Minerva et al.* in [39], proposed a new definition:

> *A Digital Twin is a comprehensive software representation of an individual physical object. It includes the properties, conditions, and behavior(s) of the real-life object through models and data. A Digital Twin is a set of realistic models that can simulate an object's behavior in the deployed environment. The Digital Twin represents and reflects its physical twin and remains its virtual counterpart across the object's entire lifecycle.* [39]

In addition to that, *Minerva et al.* in [39] individuated the state-of-the-art characteristics of a Digital Twin.

Identity The mirrored Physical Assets must be univocally identifiable, and so also the associated Digital Twins need to have a unique identifier to make them addressable in the software space. Each Digital Twin has an identifier, which allows it to be distinguished from other Digital Twins, and a pointer to the associated Physical Asset. Following this idea, multiple Digital Twins, with different identifiers, can be associated with the same Physical Asset. In addition, *Minerva et al.* consider the addition of a time and a space dimension to separate and exactly identify the Digital Twins in a specific environment at a specific period. The time could be used to determine exactly what instance of the Physical Asset is represented by the Digital Twin.

Representativeness and Contextualization The Digital Twin should be able to represent, at the Digital level, the mirrored Physical Asset. At the same time, creating a model that can fully describe it accurately is a very difficult task and not always the goal.

A Digital Twin must be described by a model designed and developed with the main goal of representing all the elements that are necessary and sufficient to qualify the Digital Twin as representative of the interested Physical Asset within the target context.

Contextualizing a Digital Twin means that it is described by the model elements that are necessary and sufficient to represent the Physical Asset in the specific context under consideration.

Reflection Reflection is a property that complements those of Representativeness and Contextualization. This characteristic imposes that a Digital Twin must be able to adequately and promptly reflect the mirrored Physical Asset state, events, and actions. The state does not have to be reflected exactly in the Digital Twins. There may be several transformation functions that relate the values of the Physical Asset to the values stored in the Digital Twin replica.

Replication This characteristic emphasizes the potential for physical objects to be virtualized and replicated multiple times in their Digital Twins. A Digital Twin is a software entity that can be scaled and replicated based on demand using modern virtualization techniques.

Entanglement The Digital Twin must always maintain an up-to-date view of the current state of the associated Physical Asset.

To this end, a connection between the Physical Asset and the associated Digital Twins is required to ensure the transfer of data in real-time or near-real-time.

Minerva et al. considered three characteristics of the *Entanglement* process:

- *Connectivity*: there should be a direct or indirect means to communicate the status changes and related data between the Physical Asset and the Digital Twins. Direct communication relies on a direct communication link between the Physical Asset and the Digital Twin. Indirect communication relies on a third party – acting as a gateway or as an observer – for sending and receiving information.

- *Promptness*: each type of Physical Asset or Application domain has different time requirements. The Digital Twin must be able to reflect the current state of its Physical counterpart. To be able to do this, the Physical Asset must be able to support an exchange of information that allows the Digital Twin not to miss any event. In particular, the Digital Twin update time must be less than the average state change time of the Physical Asset.

- *Association*: it represents the communication type between the Physical Asset and the Digital Twin. There are two types: unidirectional and bidirectional. In the former, the information is exchanged in only one direction, so from the Physical Asset to the Digital Twin or vice versa. In bidirectional communication instead, the information can be exchanged freely in both directions, enabling the execution of actions dependent on the current state of the Digital Twin.

Persistency Applications rely exclusively on Digital Twins to obtain information. Therefore, the Digital Twin, being a software entity separated from its Physical Asset, must be made persistent over time storing its data and maintaining its availability over time.

Memorization Digital Twins should have the ability to store and represent all the relevant present and past data, taking into account the *Representativeness and Contextualization* property. This characteristic enables consumers to analyze past states, and events, solve queries, understand the behavior of the associated Physical Asset, and make predictions within the same space or in other environments (what-if questions, stress tests, and more).

Composability It represents the ability to group several objects into a composed one and then observe and control the behavior of the composed object as well as the individual components. Composability offers a way to abstract and manage the complexity of large systems and focus only on the relevant parts within the target context.

Accountability/Manageability Digital Twins should not break when Physical Assets do, so they require precise and prompt management that allows them to enter into a recovery state when needed. Moreover, when the Entanglement process has some type of error, there must exist a way to quickly restore and resume operation with minimal loss of state information.

Augmentation Physical Assets usually come with a predefined set of functionalities that cannot be modified during their life-cycle.

Instead, Digital Twins can leverage software dematerialization to offer new functionalities like properties or actions that exist only at the software level and can change over time. This opens up infinite possibilities for developing and providing additional services and functionality through software alone.

Ownership Ownership in Digital Twins can be conceived in two ways: data and Digital Twin ownership. The first one is related to the data produced by the Digital Twin itself, and it is important to determine and regulate the ownership and usage rights of these data. The second one is related to the ownership of the Digital Twin in terms of its software entity. The associated Physical Asset usually has an owner but from it, it is possible to create a set of Digital Twins that do not necessarily share the same ownership.

Servitization The Servitization concept, linked to the Augmentation one, relates to the possibility of offering in the market the association of a product with services, functionalities, processes, and access to data of a Physical Asset by means of software capabilities, tools, and interfaces. All of these characteristics described previously enable this functionality. In fact, through Representativeness, Contextualization, Reflection, and Memorization it is possible to memorize the status information of the Digital Twin. Afterward, thanks to Entanglement, it is possible to maintain a continuous bidirectional channel which can be exploited through Augmentation to provide the interface to recall the various services. Therefore, it is easy to understand how Digital Twins represent a launching pad on the business side. Servitization is laying the foundations for a change in the sale of products, pushing the economy based on product ownership, to an economy based on “pay per usage”.

Predictability Digital Twins generate data about properties and events coming from the Physical Asset.

The data can be used to identify patterns, predict anomalies, or understand the behavior of a Physical Asset in a simulated environment (for example *what-if scenarios* via its Digital Twin). Digital Twins and their data can be integrated with Artificial Intelligence to enable new services and features for predictions and simulations, which can also be provided through forms of Augmentation.

Many initiatives already use Digital Twins for predictions, with NASA [23] being a major example.

Some properties discussed herein are foundational for a Digital Twin, i.e., without them, there is no real DT implementation; while others extend and increase the intrinsic value of the Digital Twin relation. This is the result of the work of *Minerva et al.* to try to obtain a definition for a general-purpose Digital Twin.

However, upon analysis of the results from academia and industry, it becomes apparent that several definitions of Digital Twins exist, each influenced and enriched by the domain in which they are used. This is because, as previously stated, Digital Twins are not only useful in manufacturing but also in any domain where a software replica of a Physical Asset is beneficial in certain scenarios. The absence of a shared definition is a significant issue because it prevents the exploitation of their full potential and the creation of Digital Twins ecosystems due to the lack of a shared definition and standards.

To date, there is not a vision or definition that is enough general and shared among experts which can be referred to implement interoperable solutions. In May 2020, Microsoft and other co-founders created the first consortium to establish standards regarding Digital Twins: the *Digital Twin Consortium*[1]. The *Digital Twin Consortium* aims to establish global standards that facilitate the cross-organizational sharing and use of information generated by Digital Twins to help advance the use of this technology in many sectors, from aerospace to natural resources. The consortium is creating a *glossary*[2] with the intent of providing shared definitions about Digital Twins. Their definition of a Digital Twin is the following:

> *A digital twin is a virtual representation of real-world entities and processes, synchronized at a specified frequency and fidelity*[3].

It is possible to notice that the Digital Twin Consortium introduces the concepts of *frequency* and *fidelity*, which are two fundamental characteristics in the modern vision that it is considered and described later in this thesis.

In addition, *ISO* (*International Organization for Standardization*) published a standard for Digital Twins to formalize the concepts and the terminology [1], and *ETSI* (*European Telecommunications Standards Institute*) is writing a technical report more related to interoperability [2].

[1]`https://www.digitaltwinconsortium.org/`
[2]`https://www.digitaltwinconsortium.org/glossary/glossary/`
[3]`https://www.digitaltwinconsortium.org/glossary/glossary/#digital-twin`

Further relevant initiatives include the *National Digital Twin Programme* [20], the UK initiative to create an ecosystem of interconnected Digital Twins for data exchange [31] based on a set of principles called *"The Gemini Principles"* [27].

2.1.2 Survey on popular technologies

From 2002 several commercial solutions tried to enter the market. The availability of the technologies and the spread of the paradigm to several application domains attracted not only the academic world but also industries worldwide to design and develop solutions for Digital Twins. Most of them are centered on the application domain where they are involved, but some others are born with a more generic point of view, especially the *Azure Digital Twins* service from Microsoft.

The main solutions available on the market are:

- *Eclipse Ditto*[4]: it is an open-source middleware, by the Eclipse team, which helps create Digital Twins of IoT devices.
- *Azure Digital Twins*[5]: Azure Digital Twins is a PaaS (Platform as a Service) cloud service that allows the creation of Digital Twins graphs. They are based on models of entire environments.
- *AWS IoT TwinMaker*[6]: it is the Amazon solution to build Digital Twins of Physical and Digital systems. AWS IoT TwinMaker was born to integrate the Digital Twins paradigm in real-world systems such as buildings, factories, industrial equipment, and production lines, so a more industry-related vision that is reflected in its metamodel and visualization features.
- *Bosh IoT Things*[7]: it is a solution that allows applications to manage Digital Twins for their IoT devices, creating a software layer that abstracts from the Physical layer. It is based internally on *Eclipse Ditto*.
- *XM Pro Digital Twins*[8]: they offer a platform where the Digital Twin paradigm is integrated with Artificial Intelligence to analyze data and provide several services that can support industries and their value chain.

After having provided an overview of the main solutions, the two most popular and complete ones, *Eclipse Ditto* and *Azure Digital Twins*, are described in a more structured way.

[4]`https://eclipse.dev/ditto/index.html`
[5]`https://learn.microsoft.com/en-us/azure/digital-twins/`
[6]`https://aws.amazon.com/it/iot-twinmaker/`
[7]`https://docs.bosch-iot-suite.com/things/getting-started/twin/`
[8]`https://xmpro.com/`

Eclipse Ditto

Eclipse Ditto is an open-source middleware that helps build Digital Twins of IoT Devices. Each device is abstracted into its Digital Twin, following the paradigm *"Device as a Service"*. The Digital Twin paradigm is applied in a domain-agnostic manner to provide an abstraction layer for IoT devices. This layer can be used by the application layer without requiring knowledge of the underlying technologies.

Eclipse Ditto is not a complete Digital Twin Platform. Instead, it manages only the Digital Twins abstractions without implementing any IoT protocols to communicate with the actual devices. For this reason, it relies on other solutions for device connectivity, such as *Eclise Hono*[9], MQTT Brokers like *Eclipse Mosquitto*[10] or *Apache Kafka*[11] broker via custom "connections". A connection represents a communication channel for the exchange of messages between any service and Ditto. Ditto supports one-way and two-way communication over connections. This enables consumer/producer scenarios as well as fully-fledged command and response use cases. An example of its typical Architecture can be seen in Figure 2.1.

Eclipse Ditto follows a model composed of these main elements:

- *Thing*: it represents the Digital Twin concept, and it is defined by a *Thing ID*.

- *Definition*: it links a *Thing* to its model that defines its capabilities. Eclipse Ditto does not have a type system. Therefore, it utilizes either the *Web of Things (WoT) Thing Model* or *Eclipse Vorto*. *Definitions* can be specified both for Ditto *Things* and *Features*. Moreover, when described through a *WoT Thing Model*, the associated *WoT Thing Description* of the *Ditto Thing* is automatically generated.

- *Attribute*: used to model the static properties of a Digital Twin within their *Ditto Thing*.

- *Feature*: used to manage data and functionalities of a Digital Twin within their *Ditto Thing*. Each *Feature* is identified by a *Feature ID*. It can adhere to a *Feature Definition* – modeled with either *WoT Thing Model* or *Eclipse Vorto* – that specifies the structure. The structure is composed of a list of *properties*.

- *Policy*: *Things* and other entities can be configured with fine-grained access control by Developers.

[9]`https://eclipse.dev/hono/docs/`
[10]`https://mosquitto.org/`
[11]`https://kafka.apache.org/`

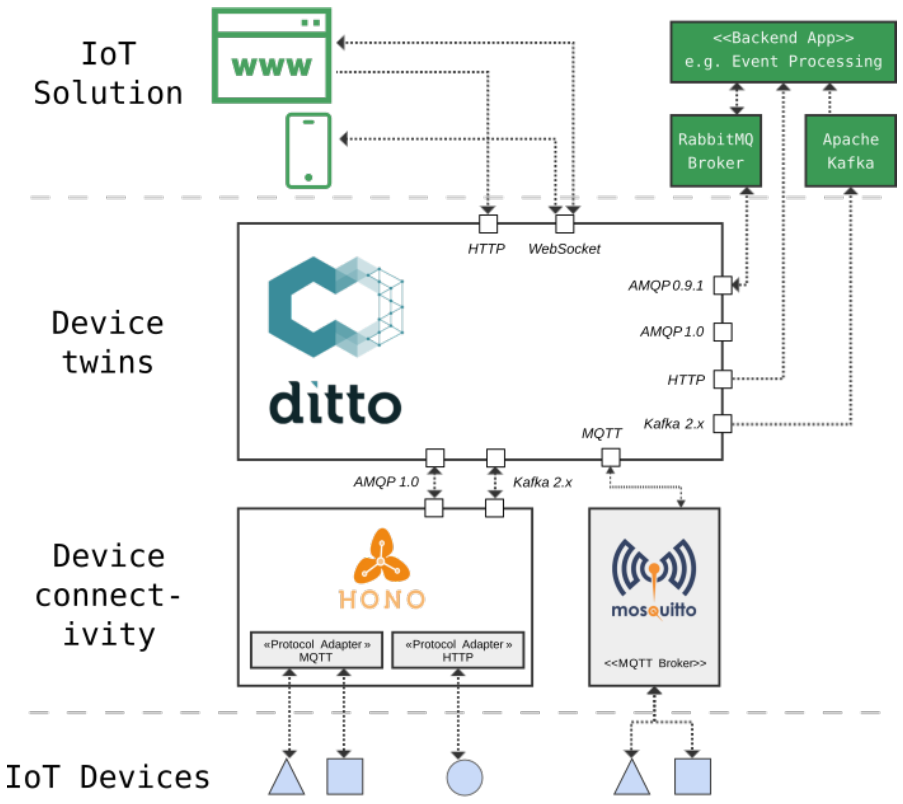

Figure 2.1: Typical Architecture with Eclipse Ditto
Source: `https://eclipse.dev/ditto/index.html`

- *Messages*: Eclipse Ditto allows to route messages to devices. *Messages* can model both *actions* or *events*. Additional control or retention policies are not supported.

In Listing 2.1 there is an example of a *Ditto Thing* with the elements described so far.

Listing 2.1: Example of a Ditto Thing

```
{
  "thingId": "the.namespace:theName",
  "policyId": "the.namespace:thePolicyName",
  "definition": "org.eclipse.ditto:HeatingDevice:2.1.0",
  "attributes": {
      "someAttr": 32,
      "manufacturer": "ACME corp"
  },
  "features": {
      "heating-no1": {
          "properties": {
              "connected": true,
              "complexProperty": {
                    "street": "my street",
                    "house no": 42
                }
            },
            "desiredProperties": {
                "connected": false
            }
        },
        "switchable": {
            "definition": [ "org.eclipse.ditto:Switcher:1.0.0" ],
            "properties": {
                "on": true,
                "lastToggled": "2017-11-15T18:21Z"
            }
        }
    }
}
```

In the following, the weaknesses and the strengths of *Eclipse Ditto* are analyzed.

Weaknesses Currently, Eclipse Ditto does not ensure that *Properties* and *Features* follow the types and the structure defined in the *Definitions*. This is because Eclipse Ditto does not have an internal type system and it does not perform any additional check on the provided *Definitions*.

The metamodel focuses on IoT devices, making it challenging to use as a general-purpose solution. Moreover, *relationships* between Digital Twins are not modeled explicitly.

In addition, actions are not modeled explicitly. They are obtained via *Messages*, but Eclipse Ditto does not perform any type of check, so it is not possible to enforce actions or payloads directly within the Model.

Strengths Eclipse Ditto's *Live Channel* enables bidirectional communication, simplifying the process of performing actions on Physical Devices.

The *Thing* history – `Eclipse Ditto >= 3.2.0` – can be retrieved and/or streamed to other services. So, there are plenty of opportunities for data analysis, predictions, and simulation techniques also based on AI.

Policies offer a very granular way to manage permissions and privileges that allow Eclipse Ditto to be used also in critical and industrial applications – for example in *Bosh IoT Things* deployments.

Events and *Change notifications*, using the *Web Socket API*, can be exploited to enable the Digital Twins observation pattern.

In addition, *connections* allow services such as *Apache Kafka* and *RabbitMQ* to be used in pipelines for observing, aggregating, and analyzing Digital Twins data within deployments.

In conclusion, it is one of the best solutions for Digital Twins in the IoT scenario.

Azure Digital Twins

One of the most versatile and general-purpose solutions on the market is the one offered by Microsoft with *Azure Digital Twins.* Azure Digital Twins (ADT) is a PaaS (Platform as a Service) cloud service that allows the creation of Digital Twins graphs based on models of entire environments. The Azure Digital Twins metamodel is domain-agnostic, making it suitable for use in any application domain.

Digital Twins in Azure Digital Twins are described using *models.* The *models* are defined in a JSON-LD-based language called *Digital Twins Definition Language (DTDL)* created by Microsoft. The metamodel followed by ADT and DTDL consists of:

- *Properties*: they are used to model the Digital Twin state – both static and dynamic. The Azure Digital Twins instance stores only the last value of each property, so no historical data is retained in the service itself.

- *Relationships*: in Azure Digital Twins, relationships between Digital Twins are part of the metamodel. This enables the creation of *graphs* of Digital Twins (Figure 2.2) that mirror the relationships existing between Physical Assets in the real world. Each *Relationship* has a direction, and it can also have associated properties.

- *Events*: they are enabled by both *Telemetry data* and by the Azure Digital Twins's *Event Notification system* that allows the design of pipelines for the integration with external services.

Once the Developer has defined the Digital Twins models, it is possible to use them to create Digital Twins that represent each specific entity in the environment and exploit *relationships* to form the *graph* of Digital Twins as described in the Figure 2.2.

Other minor elements that are allowed in *DTDL models* are *components* and *telemetry.* In addition, *DTDL models* in Azure Digital Twins support inheritance. Not all the features of *DTDL*[12] are supported by Azure Digital Twins. Examples of not supported features are *DTDL commands* and *multiplicity* of relationships.

DTDL models are identified by an `@id` field. This field only identifies the model itself. Each Digital Twin, instantiated from a model, has another identifier to identify the Digital Twin itself. From a *DTDL model* it is possible to create `n` instances, each one with a different identifier. These identifiers are valid and unique only within the Azure Digital Twins instance, so they are not global. In Listing 2.2 there is an example of a *DTDL model* with some elements described so far.

[12] `https://github.com/Azure/opendigitaltwins-dtdl/tree/master`

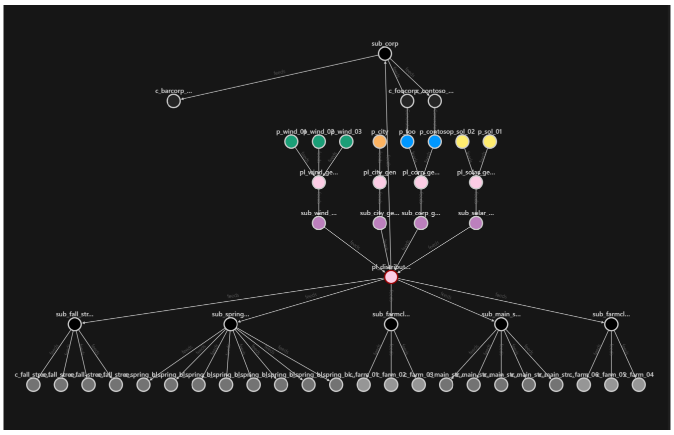

Figure 2.2: Example of an Azure Digital Twins graph

Listing 2.2: Example of a DTDL model

```
{
  "@id": "dtmi:com:adt:dtsample:home;1",
  "@type": "Interface",
  "@context": "dtmi:dtdl:context;3",
  "displayName": "Home",
  "contents": [
    {
      "@type": "Property",
      "name": "name",
      "schema": "string"
    },
    {
      "@type": "Relationship",
      "@id": "dtmi:com:adt:dtsample:home:rel_has_floors;1",
      "name": "rel_has_floors",
      "displayName": "Home has floors",
      "target": "dtmi:com:adt:dtsample:floor;1"
    }
  ]
}
```

The Azure Digital Twins service itself, like Eclipse Ditto, only allows for the creation and management of Digital Twins and the storage of the current status and relationships of the Digital Twins. So it does not store their history or handle the connectivity part with the associated Physical Assets. Microsoft adopts a completely event-driven approach, enabling the creation of pipelines with other services to obtain all the necessary features. This approach is highly scalable because each microservice can scale based on the needs.

Azure Digital Twins can be driven with data and events from any service like *Azure IoT Hub*, *Logic Apps*, custom services, and more – see Figure 2.3. These kinds of data flow enable the collection of telemetry from physical devices in the environment and the processing of this data using the Azure Digital Twins graph

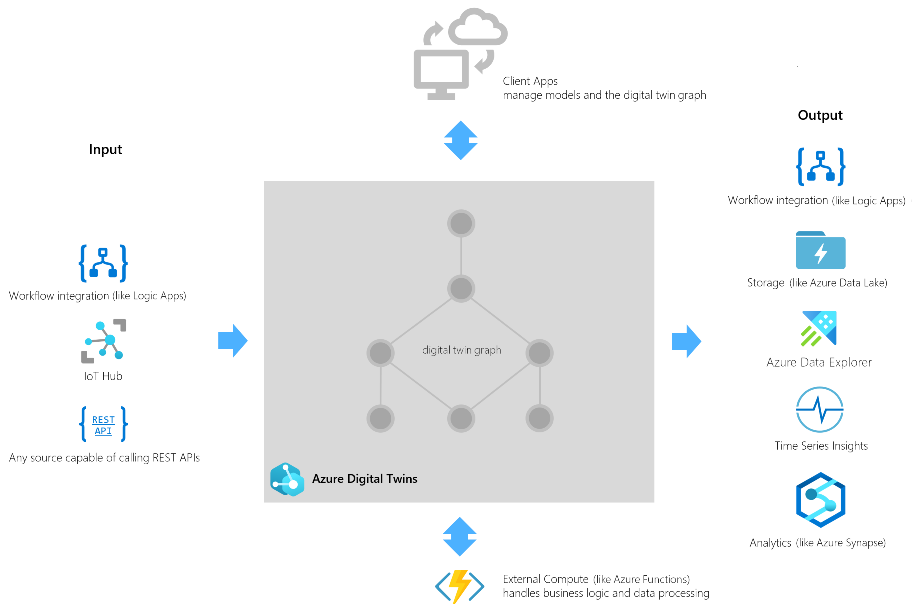
Client Apps
manage models and the digital twin graph
Input
Output
Workflow integration (like Logic Apps)
IoT Hub
REST API
Any source capable of calling REST APIs
digital twin graph
Azure Digital Twins
Workflow integration (like Logic Apps)
Storage (like Azure Data Lake)
Azure Data Explorer
Time Series Insights
Analytics (like Azure Synapse)
External Compute (like Azure Functions)
handles business logic and data processing

Figure 2.3: Scheme of Azure Digital Twins possibilities of integration
Source:
`https://learn.microsoft.com/en-us/azure/digital-twins/overview`

in the cloud.

Azure Digital Twins can be configured to send events and data to other downstream services for storage or additional services (Figure 2.3). To achieve this, it is possible to configure an *event-route* with custom data filters to expose events and send them to an *endpoint* that can be connected to the external service of interest. Azure Digital Twins supports three types of endpoints: *Event Hubs*[13], *Event Grid*[14], *Service Bus*[15].

The following is a brief description of the main Azure services that can be used in a pipeline with Azure Digital Twins:

- *Azure IoT Hub*[16]: handles the connections to Physical Devices and all the technologies and protocols. If it is part of a pipeline with *Azure Functions* it is possible to implement *bidirectional channels* with Digital Twins. This would allow obtaining data from Physical Devices through Azure IoT Hub. The data would be sent to Azure Digital Twins to update the Digital replicas, which will generate events observed by the services in the pipeline. These

[13] `https://learn.microsoft.com/en-us/azure/event-hubs/`
[14] `https://learn.microsoft.com/en-us/azure/event-grid/`
[15] `https://learn.microsoft.com/en-us/azure/service-bus-messaging/`
[16] `https://learn.microsoft.com/en-us/azure/iot-hub/`

services can execute analysis, decide actions, and invoke them on the Digital Twins, which are then reflected on the Physical Assets via IoT Hub.

- *Azure Functions*[17]: a serverless FaaS (Function as a Service) solution that allows portions of code to be specified with the possibility of automatic scaling. It is useful for business logic and data processing, data ingress, mapping, or as a simple bridge between Azure services.

- *Azure SignalR*[18]: provides real-time functionality to applications over HTTP. This service can be used in combination with events from Azure Digital Twins that relate to graph updates to implement *observation* of Digital Twins.

- *Azure Data Explorer*[19]: it can be used to store, retrieve and analyze the history of Digital Twins.

In addition, the Azure Digital Twins graph can be queried via the *Azure Digital Twins Query system* using an SQL-like query language called *Azure Digital Twins query language* to obtain all the stored information.

In the following, the weaknesses and the strengths of *Azure Digital Twins* are analyzed.

Weaknesses The main weakness of the Azure Digital Twins service is that it is mostly a closed system:

- It is not possible to create relationships between Digital Twins of different Azure Digital Twins instances, or with Digital Twins external to Azure.

- Ad-hoc, even if powerful, query language.

- Ad-hoc, even if expressive, language to model Digital Twins.

In addition, the metamodel of Azure Digital Twins does not support *DTDL commands*, preventing the modeling of Digital Twins' actions. In the same way, *Augmentation* is complex and requires several steps to be implemented and managed effectively.

Finally, Authentication and Authorization in Azure Digital Twins are implemented at a coarse-grained level, with RBAC permissions granted only at the instance level, without any options at the Digital Twin level.

[17]`https://learn.microsoft.com/en-us/azure/azure-functions/`
[18]`https://learn.microsoft.com/en-us/azure/azure-signalr/`
[19]`https://learn.microsoft.com/en-us/azure/data-explorer/`

Strengths Azure Digital Twins can leverage the Microsoft experience in cloud solutions, and this is visible in the scalability and flexibility of the offered services.

The *DTDL*, despite being an ad-hoc language, is highly flexible and expressive. In addition, the associated metamodel includes *relationships*, allowing the creation of graphs.

Azure Digital Twins has a very powerful query system and great tool support. Some related tools are: *Azure Digital Twins Explorer* to view and manage the Digital Twin graph and models, *Azure 3D Scene Studio* to offer 3D Visualization opportunities, *DTDL Extensions* for *VS Code* to help Developers during their modeling tasks and more.

Currently, it is one of the best solutions on the market for general-purpose Digital Twins.

2.1.3 Limitations of the current approaches

The concept of Digital Twin extends beyond the current proposals. Some scenarios require the digitalization of portions of reality that can be exploited at the application level, and so it is needed an approach that is able to reflect the dynamicity and the relationships that exist between the Physical Assets in their corresponding Digital Twins. This requires the removal of silos in favor of interoperability for the creation of Digital Twins ecosystems that can be exploited and navigated by consumers. There are several examples of this need, from manufacturing, where it is important to connect the different entities involved and be able to observe their dynamicity in terms of both state and relationships, to healthcare or smart cities, where several software solutions to real-world problems can be supported by digitalized and contextualized portions of reality that can be exploited at the application level.

In the analyzed commercial solutions, there is a tendency to ease the creation of vertical applications specialized in their application domain. They generally use custom protocols and data types to access Digital Twins in a closed-system manner. Furthermore, metamodels and features often rely on the definition and domain of interest in which the technology was developed. This can create challenges when attempting to establish relationships with Digital Twins implemented using different technologies, making integration impossible.

However, the situation is not better from an academic perspective either. The literature so far is full of closed-system proposals for the virtualization of *individual* Physical Assets.

In this situation, it is impossible to support a shared and interoperable *Digital Twin ecosystem* on which services and consumers can reason.

These problems limit the advantages and the opportunities that can be exploited. The section below analyzes a modern vision, followed by this thesis, based on the creation of *Web of Digital Twins* [44].

2.2 Web of Digital Twins

The previous section has already outlined the main limitations with today's Digital Twins, specifically the *siloing* of current vision and solutions that prevent the modeling of ecosystems of Digital Twins, increasingly useful in today's problems. The dominant view developed in the literature so far is about the virtualization of individual physical assets, in a closed-system perspective.

An open-system perspective that allows multiple organizations to participate in open and interoperable Digital Twin ecosystems is useful [45] to support the described needs. The Digital Twin principles and paradigm can be extended to the virtualization of complex realities composed of interrelated assets, possibly belonging to different domains and different organizations, in a more open-system perspective [44] [45]. To achieve this, standards and/or agreements in Digital Twin design and development and an abstract, expressive, and domain-independent conceptual model are fundamental.

The *Web of Digital Twin (WoDT)* vision [44] was born with the intent of creating open, distributed, and dynamic ecosystems of connected Digital Twins that are cross-domain and cross-organizational, enabling interoperability of information. This results in the *Digital Twin as a Service* approach, where Digital Twins are no more vertical silos associated with their applications.

In this new vision, it is possible to refine the definition of a Digital Twin as:

> *Digital Twins refer to the ability to clone a Physical Asset (PA) through a software counterpart during its life cycle. The Digital Twin has a model that reflects all the properties, relationships, and characteristics of the physical asset that are important for the analyzed context.*

This vision is broader than any vision analyzed before, considering the opportunity of virtualizing Physical Assets not limited to objects. Every strategic Physical Asset of an organization must have a corresponding Digital Twin, mirroring its state, relationships, and services at the digital level, resulting in an ecosystem of connected Digital Twins that reflect, and optionally augment, the physical world. The term *Physical Asset* includes Physical objects, places, and people, but also *activities* and *processes* as shown in the Figure 2.4

In the literature, this pervasive vision has strong affinities with the idea of mirror worlds as introduced by D. Gelernter in [22]. Mirror Worlds are *"software models of some chunk of reality"*, like a *"true-to-life mirror image trapped inside a computer"*.

The definition of *Web of Digital Twins* is:

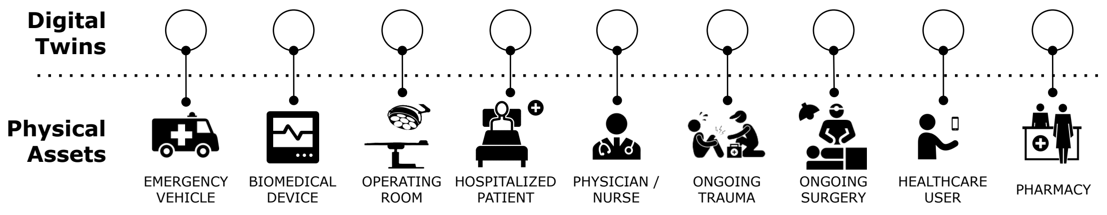

Figure 2.4: Physical Assets and corresponding Digital Twins — Source: [44]

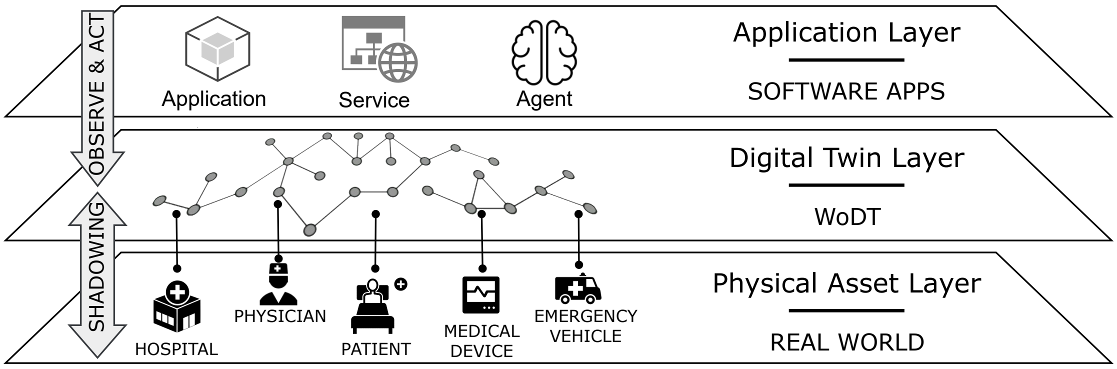

Figure 2.5: The WoDT Layered View — Source: [44]

> *A Web of Digital Twins can be conceived as an open, distributed, and dynamic ecosystem of connected Digital Twins, functioning as an interoperable service-oriented layer for applications running on top, especially smart applications and multiagent systems.* [44]

The *Web of Digital Twins* aims to create an interoperable vision where Digital Twins can be created and have relationships across multiple application domains and multiple organizations. In this view, the ecosystem of Digital Twins acts as a *shared medium* used by consumers (e.g., agents, MAS) to perceive, observe, and act upon the Physical World (Figure 2.5).

Each Digital Twin in WoDT is based on a *model M* of the corresponding *Physical Asset (PA)*. The model of Digital Twins in *Web of Digital Twins* is defined by:

- *Properties*: represent the observable attributes of the PA, as variables that can change dynamically according to the evolution of the PA state.
- *Relationships*: allow to mirror real-world relationships between the PAs in the "digital world" as relationships or links between their associated Digital

Twins to form the Digital Twins ecosystem, the *Web of Digital Twins*.

- *Events*: represent PA's domain events that can be observed via its Digital Twin.

Given the model *M*, the dynamic state S_{DT} of a DT can be defined by a tuple:

$$S_{DT} = < P, R, E, t >$$

where P is the current set of *properties*, R is the current set of *relationships*, E is the current sequence of *events* generated so far, and t is a logical timestamp representing the current time. The representation provided by the Digital Twin is about concepts that concern the Physical Asset at the *domain level* enabling agents and applications on top to reason on the Digital Twins as if they were directly interacting with the Physical Assets.

Ideally, a Digital Twin in a WoDT could host or be implemented with multiple concrete models of the same Physical Asset, capturing different aspects and contextualizing different application domains. Depending on the clients, appropriate models are then used.

Digital Twins must be *dynamic* to be able to correctly represent the current state of the real world. The creation and the disposal of Digital Twins must reflect the dynamism of the associated Physical Asset. Like properties, to correctly represent the PAs, relationships must be created dynamically, and explicitly represented in the WoDT at the DT level. These relationships should change over time to reflect the PAs' activities.

The process that allows to keep the Digital Twin state S_{DT} synchronized with the associated Physical Asset – according to the model M – is the *shadowing* process. The *shadowing* process is a fundamental building block of a Digital Twin in a WoDT, and it consists of a *bidirectional* channel that allows the exchange of data. A Digital Twin model also provides *actions* that consumers can use to control and manage the Physical Assets directly from their Digital Twins. This is of interest for the *shadowing* process because *actions* are propagated from the Digital Twin to the Physical Asset. The Digital Twin state is not automatically updated after actions invocations as it still depends on the *shadowing* process and so in case on the consequences of the action on the Physical Asset as shown in the Figure 2.6.

Not all the *shadowing processes* and *models* are equal. Digital Twins' *shadowing process* and *model* are described in terms of *Fidelity*. *Fidelity* meta-data for a Digital Twin is like an assurance that it makes respect to its ability to mirror the Physical Asset of interest. Different consumers may be interested in different levels of Fidelity, and so they may choose different Digital Twins based on that.

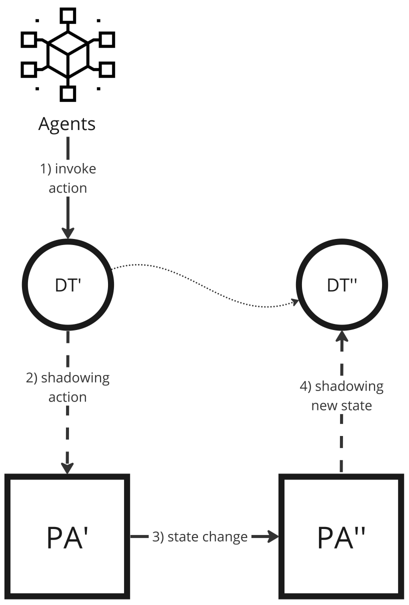

Figure 2.6: The WoDT bidirectional shadowing process

The data collected from the *shadowing* process – apart from updating the current state of the DT – constitute the so-called *digital thread* and pose the base for the implementation of the *Memorization* characteristic of Digital Twins as described by *Minerva et al.* in [39].

As previously described for the metamodel, Digital Twins in a WoDT reflect the relationships of Physical Assets in the real world. Considering that Physical Assets can have relationships with Physical Assets from other organizations, accordingly Digital Twins can have cross-organizational relationships. An approach that can cope with different ontologies to form *cross-domain* and *cross-organizational* Web of Digital Twins is needed. The *semantic modeling* of each virtualized physical asset into a corresponding DT is an aspect of primary importance to foster *interoperability* and *openness*, as well as the development of intelligent applications on top [44]. For this reason, each Digital Twin of a Web of Digital Twins is described by a Knowledge Graph that follows the domain representation of the Physical Asset and its model *M*. A Web of Digital Twins is therefore represented by a *Distributed Knowledge Graph* that links independent Knowledge Graphs, which may be based on different domain-specific ontologies, ground to the related physical asset contexts – see Figure 2.7.

As seen in the Figure 2.5 a *Web of Digital Twins* is meant to define a *cross-application distributed base layer* continuously shadowing the real world bridging the digital and physical levels [44]. Each Digital Twin could serve *as-a-service* for

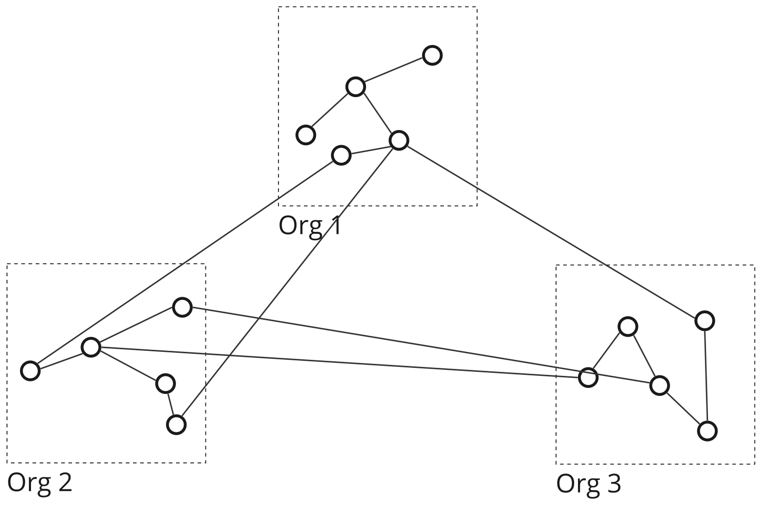

Figure 2.7: The WoDT Distributed Knowledge Graph
Each node needs to be considered an independent Knowledge Graph for a Digital Twin in the WoDT

different applications at the same time. In addition to that, for the same Physical Asset multiple and independent Digital Twins can be available, each one with a different model, specialized for different applications.

In this scenario, Applications, Agents, and MAS can be *situated* in the real world through the layer composed by the Digital Twins.

A Web of Digital Twins exposes to its consumers (e.g., Applications, Agents, and MAS) the following *interaction primitives*:

- *Action invocation*: a Digital Twin can mirror also the actions provided by the Physical Asset to command/control it. A consumer can request the invocation of an action/command to the Digital Twin that, via bidirectional shadowing, will redirect the request to the associated Physical Asset. The action invocation cannot change automatically the Digital Twin state, because it will always only depend on the shadowing process.

- *Query*: it allows performing a *one-shot* request to query the current state. Queries can be performed at the Digital Twin level (e.g., obtain the value of a property), or at the Web of Digital Twins graph level (e.g., get all the lamps in a room).

- *Observation*: as the query primitive, it can be seen at two different levels: Digital Twin level, and Web of Digital Twin graph level. It allows a consumer to subscribe and receive all the events and updates from either the interested Digital Twin or from the entire ecosystem graph.

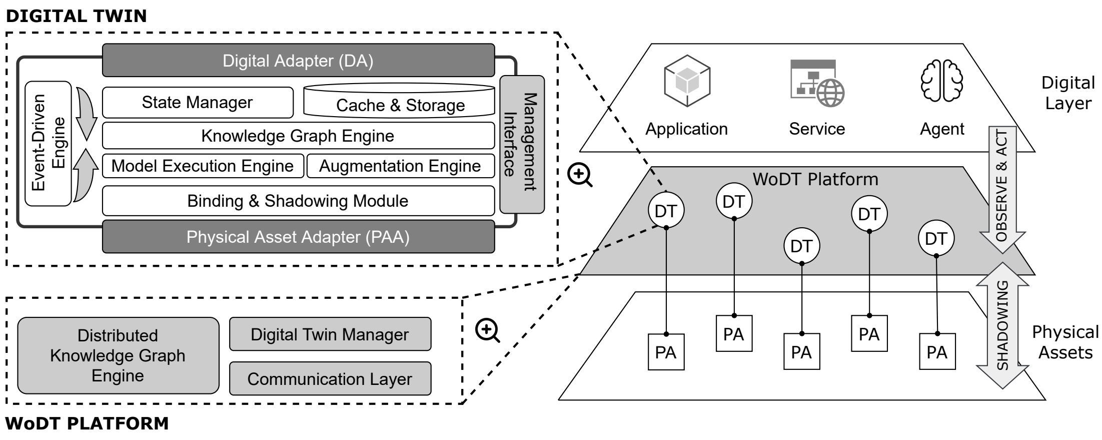

Figure 2.8: Abstract Architecture defined in the Web of Digital Twins vision
Source: [44]

Each interaction primitive cannot interfere with or block the shadowing process. So, updates from the Physical World are the priority. This is fundamental to designing Digital Twins and a platform that supports the Web of Digital Twins model.

In [44] an Abstract Architecture fulfilling all the requirements of the Web of Digital Twins model is defined. The architecture, described in Figure 2.8, is completely *event-driven* and is based on three main types of events:

- e_{PA}: these are events from the Physical Assets derived from changes in real-world state.
- e_{DT}: these are events that represent changes in the Digital Twin state.
- e_i: these are internal events that guide the functioning of the Digital Twin.

The architecture is composed of components that are internal to each Digital Twin in the ecosystem and components that are outside, part of a Web of Digital Twins "platform".

The components inside each single Digital Twin are:

- *Physical Asset Adapter*: handles the connection with the Physical Asset, and it manages the e_{PA} events from and to the associated Physical Asset.
- *Building & Shadowing Module*: handles the *binding* process that associates the Physical Asset to the Digital Twin and the perpetual *shadowing process* to keep the Physical Asset and the Digital Twin in synch.

- *Event-driven Engine*: it is the engine that binds together all the internal components of a Digital Twin.
- *Model Execution Engine*: allows the Digital Twin Developer to make the Digital Twin *model* operational and defines how the model influences the Digital Twin state and behavior.
- *State Manager*: it has the responsibility of managing the Digital Twin state consistently.
- *Knowledge Graph Engine*: it is the engine that manages the Knowledge Graph of the Digital Twin, including the links to other Digital Twins.
- *Cache & Storage*: it implements the storage and caching functionalities.
- *Management Interface*: it is the interface exposed to other Digital Twins, to the WoDT Platform, and external entities for administration and services.
- *Digital Adapter*: it allows consumers to interact with the Digital Twins and use the provided Interaction primitives.
- *Augmentation Engine*: it is the component that implements the augmentation within the Digital Twin.

The components of the WoDT Platform are:

- *Distributed Knowledge Graph Engine*: it provides the means to navigate the entire ecosystem (WoDT) Knowledge Graph.
- *Digital Twin Manager*: it manages the Digital Twins lifecycle, offering typical lookup services such as white and yellow pages.
- *Communication Layer*: it allows consumers to use the services provided by the WoDT Platform.

Chapter 3

Contribution: Web-based WoDT

3.1 Objective

After having given a brief overview of the motivations of this thesis and having provided the essentials of the necessary background, it is appropriate to understand more deeply what the objective of this thesis is and the contributions that are proposed.

As described above, nowadays, we are assisting in the spreading of closed-system proposals for the virtualization of individual Physical Assets used for vertical applications. This is not a universally bad thing, surely there are use cases where even individual and vertical Digital Twins are very useful, but these types of proposals do not allow us to fully exploit the advantages offered by the Digital Twin paradigm that could be used in modeling general and open ecosystems of Digital Twins where relationships reflect the ones between the associated Physical Assets, enriching the data available to consumers. As stated, the *Web of Digital Twins* vision [44], the one followed by this thesis, aims to create an interoperable service layer where Digital Twins can be created and have relationships across multiple application domains and multiple organizations.

The objective of the thesis is to try to propose an implementation of the *Web of Digital Twins* vision [44] to obtain the possibility for interoperability and openness between Digital Twins of different domains, organizations that use different existent technologies, and create an open, discoverable, and navigable ecosystem of Digital Twins that can serve as a service layer for applications on top.

So the objective is to propose an implementation that is aligned with *Web of Digital Twins* but at the same time allows the creation of ecosystems composed of heterogeneous Digital Twins in a way that is similar to the *National Digital Twin program* [20]. This need comes from the fact that it is challenging to create a single technology or a single platform and expect that everybody will use it. Therefore, there is the need to cope with existing technologies, standards, and affirmed similar paradigms and try to understand the similarities, and differences to make them interoperable and enable the advantages of the *Web of Digital Twins* vision.

3.2 Idea

The idea for this thesis is inspired by some recent works that try to exploit the Web as an application architecture for the creation of dynamic, open, and long-lived systems [11][12]. These systems adopt the architectural style of the Web — REST [18] — to inherit its properties. Central to the Web architecture, the *hypermedia*, is now increasingly used for designing highly scalable, dynamic, open, and interoperable systems such as the Linked Data Platform [47] and the Web of Things [35]. These systems have some requirements and characteristics that are similar to the Digital Twins ones, so the Web is an interesting approach to explore.

Therefore, in this thesis, there is the will to research and explore the use of the Web as the application architecture, the REST architectural style, and the Web standards and protocols for the creation of WoDT ecosystems composed of heterogeneous Digital Twins supported by existent technologies. Specifically, there is the will to understand what is required and redact some form of specifications that *WoDT-complaint Digital Twins and Platforms* must follow to be able to create ecosystems of Digital Twins that follow the *Web of Digital Twins* vision.

The characteristics of the Web and REST can be exploited for the creation of a description and management layer for Digital Twins that allows the creation of interoperable and open ecosystems. One of the main constraints of the REST architectural style that helps here is the *uniform interface*, which allows information hiding.

In addition to the use of existing technologies, popular paradigms must also be considered. Specifically, the *Web of Things* paradigm is very popular in the Web scenario and has already explored some ideas to integrate devices as first-class entities into the Web, experimenting with the digitalization of Physical Assets. Moreover, it has a renewed interest in Digital Twins, proposing several use cases and deployment scenarios where Digital Twins are involved [35]. Digital Twins and WoT Things, from an external point of view, seem not so distant. For this reason, to provide a solution that may also be considered by the *Web of Things* community or at least to be as open as possible, a compatibility layer between the two could be researched and if possible provided. Hence, a deeper study and analysis is needed on the Web standards, protocols, paradigms, and on the REST architectural style to be able to propose a Web-based design.

3.3 Contributions overview

Following the objective and the idea, the contributions of this thesis can be summed in the following ones:

- *Analysis and alignment*: a subset of the Web standards, protocols, and paradigms has been selected to support the requirements of the proposed idea. In particular, for the *Web of Things* paradigm, a comparison and an alignment, with Digital Twins in the *Web of Digital Twins* vision is provided.

- *Web-based Web of Digital Twins*: this is the main contribution of the thesis and it is related to the actual design and prototypical implementation of the *Web of Digital Twins* vision using the Web as an application architecture to enable interoperable, open and heterogeneous ecosystems.

3.4 Requirements

After the description of the main idea of the *Web-based Web of Digital Twins*, it is necessary to state its high-level requirements. In this section, the functional, non-functional, and implementation requirements will be briefly discussed and analyzed, paving the way for the analysis of the Web standards, protocols, and paradigms to support the idea. Obviously, the requirements and the characteristics from the *Web of Digital Twins* vision, described in the section 2.2, are implicit and the base for the ones presented in this section. It is worth remembering that in the *Web of Digital Twins* vision, the ecosystems are enabled by two main elements: the *Digital Twins* (in the following referred to as *WoDT Digital Twins* to highlight their compatibility with the WoDT vision) and the *Web of Digital Twins Platform* (*WoDT Platform* or *WoDT Digital Twins Platform*). In addition, by *Consumers* are meant any entity – person, agent, service, and so on – that wants to interact with any WoDT Digital Twins or any WoDT Digital Twins Platform.

3.4.1 Functional requirements

Metamodel WoDT Digital Twins must offer the *Web of Digital Twins* metamodel as described in [44] and in section 2.2. Considering the objective of the thesis, i.e. having heterogeneous ecosystems composed of Digital Twins developed with different technologies and platforms, each WoDT Digital Twin must map its internal metamodel – used by the technology at hand – to that of the Web of Digital Twins.

Shadowing WoDT Digital Twins must implement a single, bidirectional, and consistent shadowing process as defined in the Web of Digital Twins vision.

Descriptions WoDT Digital Twins must be described in terms of:

- *Metadata*: it includes DT's high-level metadata and the description of all the exposed interfaces for the provided interaction patterns. The DT's metadata is necessary for the interaction with Consumers and to enable the WoDT Digital Twins to be able to join the WoDT ecosystem. This is the intersection point that can enable the compatibility layer with *Web of Things*.
- *Current state*: it describes the current state – snapshot – of the WoDT Digital Twin as a *navigable Knowledge Graph*. The current state includes each element of the metamodel:
  - Properties' current values
  - Current relationships' instances: the target must be the global identifier of the target WoDT Digital Twin
  - Current set of available actions

  The Knowledge Graph must be generated by the WoDT Digital Twin itself following its reference ontologies based on its domain of interest. All of this allows you to work with a WoDT Digital Twin as if you were working directly with the Physical Asset it mirrors.

A Consumer must be able to navigate between and within both types of descriptions.

Memorization WoDT Digital Twins may optionally store their old data to offer old versions of their descriptions, especially their state in a specific moment in the past, to Consumers. The memorization service, when available, is intended as another Interaction pattern, and as such must be adequately described.

Augmentation WoDT Digital Twins may optionally offer augmented properties, events, or actions that must be differentiated from the Physical Assets ones. Moreover, their interfaces must be adequately described inside the descriptions.

Creation of a Web of Digital Twins The Web of Digital Twins paper [44] suggests the use of a WoDT Digital Twins Platform to support the creation of the WoDT ecosystems. The WoDT Platform should provide all the services that a single WoDT Digital Twin alone is not capable of supplying – at least efficiently.

The Platform must manage the Web of Digital Twins – so the WoDT ecosystem – of an organization or even of multiple organizations, spanning different domains and act as the service layer described in [44]. The ecosystem should be built by the merging of the registered WoDT Digital Twins' data. It should be possible to register a WoDT Digital Twin to a WoDT Platform in at least two ways:

- *Automatically by the DT*: WoDT Digital Twins must be able to register/add themselves to a Platform (or multiple ones).

- *Manually by the administrator of the WoDT Platform*: The WoDT Platform administrator must be able to manually register/add a WoDT Digital Twin — discovered in any way — to the WoDT Platform.

Within the same WoDT Platform, more WoDT Digital Twins can be associated with the same Physical Asset.

The flexibility in the registration process and the possibility of having cross-organizational and cross-domain ecosystems enables the creation of custom views of the real world that are contextualized and contain only the relevant information for the problem at hand. Digital Twins are exploited as a way to access and reason over the reality of interest. So, the same WoDT Digital Twin can be part of different ecosystems to create a contextualized view of reality composed of only the necessary blocks.

The WoDT Platform must represent and expose the ecosystem via a Knowledge Graph – the *WoDT Digital Twins Platform Knowledge Graph.*

Interaction patterns The WoDT Digital Twins and the WoDT Platforms must offer the interaction patterns described in the Web of Digital Twins vision. In addition to them, the WoDT Platforms should provide the *multi-model directory service* that from a Physical Asset identifier returns all the registered WoDT Digital Twins that are associated with it within the Platform ecosystem.

Another important interaction pattern is the *Navigation*:

- It should be possible to easily navigate within and between the ecosystems exploiting the relationships that exist between WoDT Digital Twins

- It should be possible to navigate from a WoDT Digital Twin – that acts as an ecosystem entry-point for a Consumer – to all the WoDT Platforms where it is registered (or it has been registered) allowing the navigation, and the request of services, in each ecosystem where it is part of.

Compatibility layer Web of Digital Twins does not have the objective of competing with the other paradigms. For this reason, the idea of interoperability and openness can be extended to the possibility of compatibility with other paradigms.

As described in the section 3.2, an interesting paradigm that can be compatible with the Web of Digital Twins is *Web of Things*. Hence, each WoDT Digital Twin should offer a compatibility layer towards the Web of Things that allows Consumers to reason on a WoDT Digital Twin as a WoT Thing – if they want or have the necessity.

The proposed design must be general enough to support compatibility with additional or different paradigms and technologies.

3.4.2 Non-functional requirements

Identification In the Digital Twin paradigm, the *identification* is a fundamental element. A Digital Twin mirrors a Physical Asset, so it is important to precisely state how they are both identified:

- WoDT Digital Twins must be identified uniquely globally. A WoDT Digital Twin identifier must not change during its lifecycle.
- Physical Assets must be identified with the best-suited identifier that reflects their domain, e.g., a license plate for a car, a health card number for a person, and so on. It may not be global, but it must be unique within its domain of interest.

Heterogeneity This requirement copes with the need to create Web of Digital Twins ecosystems composed of heterogeneous Digital Twins. WoDT Digital Twins data, metadata, and interaction patterns must be exposed via a *Uniform Interface* to provide information and technological hiding. This enables consumers to abstract away from the particular technology or platform – Digital Twin Builders e.g., Azure Digital Twins, Eclipse Ditto, and so on – used under the hood by the WoDT Digital Twin Developers, allowing the seamless creation of heterogeneous ecosystems.

Independence WoDT Digital Twins should be able to exist alone and within the Platform, enabling the *DT-as-a-Service* vision. Following this requirement, consumers interact:

- with Digital Twins for the services that are offered at the Digital Twin level – obtainment of the WoDT Digital Twin Knowledge Graph, observation, and so on – using the descriptions stated above.

- with the Platform to have access to the application service layer enabled by the Platform Knowledge Graph on the ecosystem of WoDT Digital Twins.

Openness and Interoperability WoDT Digital Twins must have the possibility to create cross-organization and cross-domain relationships, resulting in ecosystems composed of more organizations that deal with multiple domains.

Moreover, the openness is related also to the support of different protocols for the implementation of the interaction patterns. The protocols used in the exposed Interaction patterns must not be enforced by design. Interaction patterns must be described semantically through the use of Hypermedia controls. In this way, the resulting descriptions are independent of the specific protocols needed by the WoDT Digital Twin Developers. For example, in some constrained scenarios HTTP is too heavy and CoAP is a necessity.

Dynamicity WoDT Digital Twins should be able to cope with completely dynamic updates in their state and their model. In the same way, WoDT Platforms should be able to cope with the dynamicity of the registered WoDT Digital Twins and provide an updated view of the ecosystem.

Deployment WoDT Digital Twins should be able to be deployed in the best-suited network node i.e., on the Cloud, the Fog, or the Edge, without additional constraints. Hence, they should always be able to join the Web of Digital Twins ecosystem regardless of their deployment type.

Fidelity As stated previously, how to express Fidelity is still an open problem. Hence, the goal for this thesis is to specify *where* there could be the need to put Fidelity metadata.

3.4.3 Implementation requirements

REST architectural style Considering the necessity for a dynamic, open, and long-lived system that exposes an interoperable Uniform Interface, the WoDT Digital Twins and the WoDT Digital Twins Platforms must be designed following the REST architectural style – to inherit all the properties – and exploit Web technologies, standards, and protocols.

Chapter 4

Analysis of the Web for a Web-based WoDT

In the previous chapter, the contributions of this thesis were defined, identifying a Web-based implementation of the *Web of Digital Twins* vision. Following the requirements stated in the section 3.4, the subset of the Web standards, protocols, and paradigms have been selected, and this chapter provides an overview of the results. Subsequently, the comparison between *Web of Things* and the *Digital Twins* paradigm is provided to feed the ensuing design of the idea.

4.1 Web and REST

The first step is certainly the study of the *Web* and its *REST* architectural style for the creation of dynamic, open, and long-lived systems.

The World Wide Web was created by Tim Berners-Lee while working at CERN [4] as an attempt to persuade CERN management that a global hypertext system was in CERN's interests for the management of general information about accelerators and experiments conducted at CERN. [4] discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext system.

From there was born the World Wide Web [5] as a shared information space through which people and machines could communicate [6]. The World Wide Web was intended as a distributed hypermedia system. *Hypermedia* is defined by the presence of application control information embedded within, or as a layer above, the presentation of information [18]. *Distributed Hypermedia* allows the presentation and control information to be stored at remote locations [18].

The *REST Architectural Style* was developed as an abstract model of the Web Architecture. It was used to guide the redesign and definition of the *Hypertext*

Transfer Protocol (HTTP) and *Uniform Resource Identifier (URI)* [18].

The important lesson from the World Wide Web design is that the World Wide Web has succeeded in large part because its software architecture has been designed to meet the needs of an Internet-scale distributed hypermedia application [18]. One radical design decision by Tim Berners-Lee, with respect to the vision of Hypertext created by Ted Nelson [40], was the possibility of having broken links, allowing to refer to non-existent or not available resources.

In the following, the *REST Architectural Style*, the protocols, and the paradigm of interest for this thesis are presented. The provided descriptions are not intended as exhaustive because they will only cover the necessary and sufficient parts to be able to understand the contents of this thesis and the design of the *Web-based Web of Digital Twins*.

4.1.1 REST Architectural Style

First, it is needed a definition of *architectural style*:

> *An architectural style is a coordinated set of architectural constraints that restricts the roles and the features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style.* [18]

The first version of *REST* was developed by Roy Thomas Fielding between October 1994 and August 1995, primarily as a means for communicating Web concepts while developing the HTTP/1.0 specification and the initial HTTP/1.1 proposal. REST was ultimately defined by Roy Thomas Fielding in his doctoral dissertation in 2000 [19].

To define the architecture of the Web and so its architectural style, it was necessary to understand the requirements of the Web. The objective was to build a system capable of providing a universally consistent interface to this amount of structured information in a way that can be referenced and used by others without having a local copy, in an interoperable and scalable way.

The requirements were the following ones, from [18].

Low Entry-Barrier Considering the objective of creating a shared information space based on the voluntary participation of people and machines, a low entry barrier was needed.

The chosen user interface was the *Hypermedia* due to its simplicity, generality, and the possibility of freely structure information through the use of hypermedia relationships (links). This choice allowed authors to reference information that may be temporarily or permanently unavailable or that did not exist yet, allowing for partial availability. This was a difficult decision in terms of style, as it resulted in links no longer being globally consistent, which contrasted with the definitions at the time.

Simplicity was also a goal for the sake of application developers since all the protocols were text-based, so easy to test and analyze.

Extensibility Each system that is intended to be long-lived must be designed considering the possibility of changes in the requirements. The Web is one of these systems.

Distributed Hypermedia — Latency The typical use case of the Web is the transfer of large amounts of data, so large-grained data transfers. The Web is based on hypermedia interaction which usability is highly sensitive to the user-perceived latency, i.e., the time between selecting a link and the rendering of a usable result. For these reasons, the Word-Wide Web architecture needs to minimize network interactions.

Internet-Scale The Web is intended to be an Internet-scale distributed hypermedia system. Moreover, the entire system is not under the control of a single entity.

Then, architectural elements need to be scalable and robust to handle unanticipated loads or malformed/malicious data. Each architectural element has some constraints to provide anarchic scalability:

- Clients cannot be expected to maintain knowledge of all servers
- Servers cannot be expected to retain the knowledge of the state across requests
- *Back-pointers* in hypermedia data are not allowed

In addition to that, the deployment of architectural elements must be completely independent to allow old and new implementations to co-exist without preventing the use of the new features and capabilities of newer implementations.

The requirements just described were used to derive the abstract model of the World-Wide Web creating the *REST* architectural style:

> *REST (Representational State Transfer) is a coordinated set of architectural constraints that attempts to minimize latency and network communication, while at the same time maximizing the independence*

> *and scalability of component implementations. This is achieved by placing constraints on connector semantics, where other styles have focused on component semantics. REST enables the caching and reuse of interactions, dynamic substitutability of components, and processing of actions by intermediaries, in order to meet the needs of an Internet-scale distributed hypermedia system.* [18]

The name *"Representational State Transfer"* is intended to evoke an image of how a well-designed Web application behaves: a network of Web pages forms a virtual state machine, allowing a user to progress through the application by selecting a link or submitting a short data-entry form, with each action resulting in a transition to the next state of the application by transferring a representation of that state to the user [18].

The REST architectural style was defined by a set of constraints taken from the following existent architectural styles:

- *Client-server*: the separation of concerns between the architectural elements improves the portability of the user interface and the scalability of the server. In addition to that, this enhances the deployment because components, being separated, can evolve independently.

- *Stateless*: means that each request should contain all the necessary data without the possibility to refer to an old request. This has the pros of improving the visibility (easy to monitor), reliability (easy to recover from partial failures), and scalability (server components don't need to store state and can free the resources quicker) of the system with the tradeoff of decreasing network performances due to the increase in payload dimension (for the repetitive data) and reducing the control of the server on the application state.

- *Cache*: added as a way to improve network efficiency by implicitly or explicitly labeling data as cacheable or not. This architectural style, which tries to mitigate the effects of the stateless architectural style, has the advantage of reducing the necessary round trips increasing efficiency, scalability, and user-perceived performance at the cost of a reduction in reliability due to stale data.

- *Uniform Interface*: to obtain generality, the REST architectural style emphasizes a uniform interface between components. This increases the visibility of interactions, simplifies the overall system architecture, and enables *information hiding.* The cost of having a uniform interface is that it was optimized for the common case of the Web – large-grain hypermedia data transfer –

eliminating the possibility of being specific to application needs. It is formed by the union of four constraints: *identification of resources*, *manipulation of resources through representations*, *self-descriptive messages* and *hypermedia as the engine of the application state (HATEOAS).*

- *Layered System*: it is possible to compose an architecture of hierarchical layers that cannot see beyond the immediate layer with which they are interacting. In this way, the overall system is simpler because each layer only knows about a single layer. Additionally, it increases the system scalability and security because intermediaries can be: load balancers, shared caches, or reverse proxies. The stateless constraint is essential for the presence of this constraint. Components that are between client and servers are called *intermediary components*, and they act as both a client and a server to forward, with possible translation, requests and responses. The two main types of intermediary components are: *proxy* and *gateway (reverse proxy).*

- *Code-On-Demand*: it is an optional constraint that extends client functionalities by downloading and executing code on the client. It simplifies clients and improves the system's extensibility, reducing visibility.

The base concepts of REST are *resources* and *representations.* REST components communicate by transferring a representation of a resource in a format matching one of an evolving set of standard data types, selected dynamically based on the capabilities or desires of the recipient and the nature of the resource [18]. So, the *uniform interface* enables *encapsulation.*

Resources are the key abstraction of information in REST. Any information that can be named and is important enough to be referenced as a thing itself can be a resource, even non-virtual objects (physical objects). They allow authors to reference the concept rather than the specific representation. A resource R is a temporally varying membership function $M_R(t)$, which for time t maps to a set of values: *resource representations* and/or *resource identifiers.* As said previously, references to a resource can be made even before its existence – in that case, the function returns an empty set. Each resource has a *resource identifier.* The identifier used on the Web is the URI [9].

Resource's current or intended state is captured by *representations.* A *representation* is a sequence of bytes, plus *representation metadata* to describe those bytes [18]. The message control data defines how the receiving component will use the representation. Moreover, when a resource can be described with multiple representations, *content negotiation* can be used to select the best one. Usually, content negotiation happens for the negotiation of the representation *media type.*

Hence, resources identify concepts, so possibly Digital Twins, instead representations are used to manipulate resources. This enables the information-hiding principle because a client is restricted to the manipulation of representations rather than directly accessing the implementation of a resource. The latter constraint will be used to enable the uniform interface and the information hiding in the *Web-based Web of Digital Twins*.

REST concepts of "request" and "response" may have the appearance of a *remote invocation style*, but REST messages are targeted at a conceptual resource rather than an implementation identifier for a remote procedure or method.

RESTful APIs

Nowadays, the term *HTTP RESTful API* is very common but if we consider the pure version of REST most of the APIs are not RESTful, and they instead implement it only at certain levels.

It is worth considering that the *Hypertext Transfer Protocol (HTTP)*, and the *Constrained Application Protocol (CoAP)* for constrained devices, are the only protocols designed specifically for the transfer of resource representations. Back in the day, REST was used to identify problems in old versions of HTTP and to create the HTTP/1.0 and the HTTP/1.1 proposals. However, creating an HTTP API does not directly mean that we are creating an API that follows and respects all the REST constraints and principles.

About this, the creator of REST, Roy Thomas Fielding, wrote an article [16] where he states that all the pure REST API – so the one that can be called RESTful – must be hypertext-driven. To be "hypertext-driven" – and not merely RPC over HTTP – means that the uniform interface constraint must be implemented completely. The part that usually is missing is the *"Hypermedia as the Engine of Application State" (HATEOAS)* that, instead, it is fundamental to enable a uniform interface with self-descriptive messages and so to be able to manipulate resources through their representations without accessing out-of-band information. In addition, it is fundamental to use the right media type or an ontology to give semantics to representations and so to enable the "self-descriptive messages" constraint. To respect the "self-descriptive messages" constraint, each resource representation should carry enough information to describe how to process the message.

The REST architectural style does not impose any semantics to HTTP methods – also because REST is independent of HTTP – but instead in [16] is stated that at most are *media types* that tells the client either what method to use or how to determine the method to use, or it is obtained following the ontology used in the representation.

In conclusion, a REST API should be used with no prior knowledge beyond the initial URI and the set of standardized media types that are appropriate for the intended audience [16].

In 2008, Leonard Richardson proposed the *Richardson Maturity Model* that breaks down the principal elements of a REST approach in levels of maturity where the higher your API is the more it is aligned with the REST principles. It is composed of four levels:

- *Level 0*: this is the starting point where HTTP is used merely as a transport system for remote interaction without using the mechanisms of the Web. This is usually similar to Remote Procedure Invocation, with a singular endpoint for all the requests.

- *Level 1*: in this level, *resources* are introduced, and different URIs are used for the requests.

- *Level 2*: this level is characterized by the use of HTTP methods to define operations on resources. This is where most of the public REST APIs stand.

- *Level 3*: in this level, the use of hypermedia is introduced following HATEOAS.

Level 3 of the *Richardson Maturity Model* (RMM) is not enough for defining a RESTful API because it misses the concept of *self-descriptive messages*. Following this confusion, the *Hypermedia Maturity Model*[1] (HMM) was created to make Hypermedia API more clear, taking the RMM Level 3 and splitting it into four additional levels. The more an API goes up in these levels, the more it is self-descriptive, satisfying both HATEOAS and self-descriptive messages constraints.

- *HMM Level 0*: this is the RMM Level 3, where hypermedia can be encoded in an ad-hoc way with no semantics that allows a client to recognize it as a link to process it. This solution requires out-of-band documentation.

- *HMM Level 1*: at this level, media types that model links as first-class features are used, and so they are recognizable and usable directly from the message. This is perfectly fine with read-only APIs. An example of HMM Level 1 media type is *JSON-HAL* [33].

- *HMM Level 2*: at this level, also forms are introduced as first-class entities in the media type. This level allows describing and representing form-like hypermedia controls. An example is the *Siren* media type [49].

- *HMM Level 3*: a representation of resources at this level does not just describe the actions that are possible to take, but also describes the data itself. Automated interactions can occur when data is self-descriptive and adheres to the self-descriptive messages constraint. An example is *Hydra* [36].

[1] `https://8thlight.com/insights/the-hypermedia-maturity-model`

4.1.2 Web resources: identification and versioning

This section presents the best practices and the protocols for identifying and versioning Web resources. The knowledge presented here will be used in the *Web-based Web of Digital Twins*.

Information vs. Non-information resources

Considering that *Digital Twins* can be seen as Web resources, the comparison between *information* and *non-information resources* must be analyzed to be able to accurately design the identifiers and the interaction patterns.

A *Uniform Resource Identifier* is a compact string of characters for identifying an abstract or physical resource [9]. As previously stated, a resource can be anything from virtual entities to non-virtual ones. Then, URIs need to identify not just virtual resources but also real-world objects like people, cars, or abstract entities. In [46] these entities are called *real-world objects* or *things*.

When dealing with real-world objects it is necessary to be able to identify the object itself and its representations otherwise this overlapping can lead to confusion when used in semantic descriptions [46] with questions like: *"Are we referring to the object itself or its representation?"*.

Sauermann et al. address this problem in [46] providing two different solutions that remain compatible with Semantic Web and Linked Data principles and allow retrieving a representation of real-world objects based on their URI.

> *There should be no confusion between identifiers for Web documents and identifiers for other resources. URIs are meant to identify only one of them, so one URI can't stand for both a Web document and a real-world object.* [46]

A real-world object is also called a *non-information resource*, instead a Web document is called an *information resource*. To be called *information resource* all its essential characteristics must be able to be conveyed in a message and this is not the case for real-world objects. Considering Mario as a person, we may like him, but we may not like his homepage. Here it is possible to understand the need for two different URIs and also the need to be able to "jump" from the resource Mario to its representation.

Hash URIs One possible component of a URI is the *fragment*. A URI that contains a fragment cannot be retrieved directly, its root resource – where there is the *information resource* – must be retrieved before. For this reason, the first solution to the problem is to use *hash URI* for *non-information resource.* This permits to refer to non-information resources without creating ambiguity.

303 URIs The second solution derives from the *httpRange-14 resolution* [15] and it consists in the use of the `303 See Other` HTTP Status code to indicate that the requested resource is a *non-information resource*. To complete the response, the `Location` HTTP Header must point to the document that contains the representation of the resource, and so the *information resource*.

Memento protocol

When navigating the Web, usually we obtain only the current version and representation of a resource. In some cases, it is interesting the possibility of obtaining the historical representation of a resource for several use cases, such as analysis.

In [13][50] is proposed the *Memento protocol* as a straightforward extension of HTTP that adds a time dimension to the Web, allowing the navigation of older representations and the negotiation in the time dimension. This idea is directly linked to the Memorization requirement described previously, and for this reason, the *Memento* protocol was explored as a possible candidate to implement it.

The protocol defines four main concepts which interact as defined in Figure 4.1:

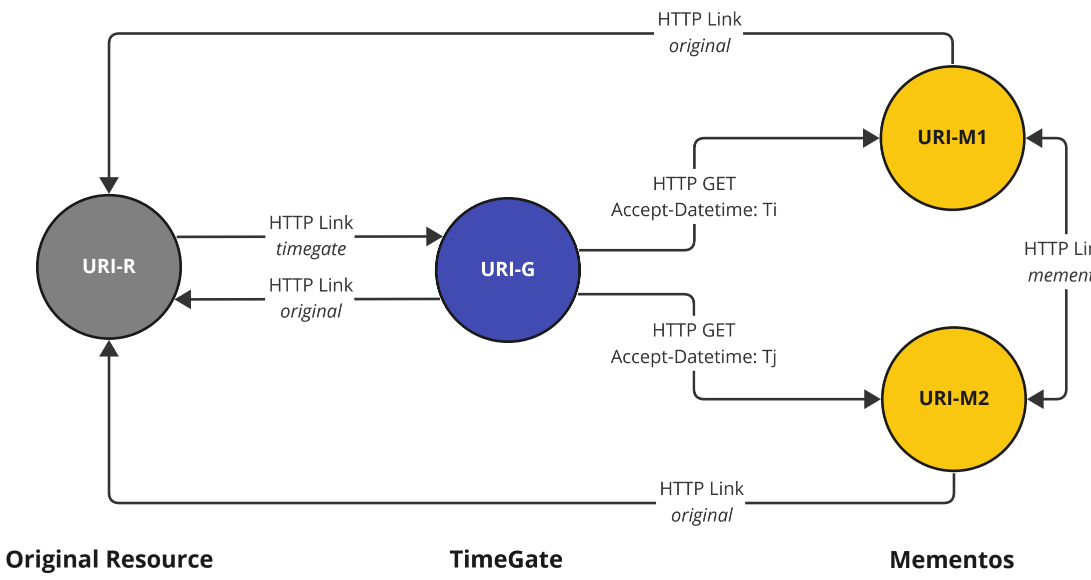

Figure 4.1: Architectural overview of the Memento protocol

- *Original resource*: the original resource for which we want to access prior versions.

- *Memento*: it is a web resource that consists of a prior version of the *Original resource*. So it is the historicized version of the *Original resource* in some past point in time.

- *TimeGate*: it is the Web resource that implements the *datetime negotiation* and helps clients in finding the right Memento, of the *Original resource*, for their preferred datetime.

- *TimeMap*: it is the Web resource that offers the list of URIs of Mementos for an *Original resource*.

In [13] are specified all the HTTP Headers required to correctly implement the protocol and enable the uniform interface over the resource versioning.

4.1.3 Semantic Web

The *Web-based Web of Digital Twins* has the objective of creating a navigable Knowledge Graph of the ecosystem that can contain domain data. For this reason, a technology that links semantic data is needed. The *Semantic Web* is the Web-based solution to this problem.

The Semantic Web was born in 2001 from an idea of *Tim Berners-Lee et al.* [8] as a collection of standard technologies to realize the *Web of Data*.

At the time, the issue was that web content was only intended for human consumption, making it impossible for computers to automatically process its semantics. The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, facilitating collaboration between computers and humans [8]. Being able to pass from the Web of documents to the Web of Data enabled the definition of rules for making inferences on data, allowing for logical data augmentation based on imposed properties.

To implement the Semantic Web, it is needed a model of data that allows information to be distributed over the Web [3]. The *Resource Description Framework (RDF)* provides the model for data and a syntax so that independent parties can exchange and use it. RDF is designed mainly to be read and understood by computers. RDF can be serialized in different syntaxes such as Turtle [43], XML [21], and JSON-LD [48]. The model is based on triples [8] composed by: subject, predicate, and object. The Semantic Web model enables the creation of Knowledge Graphs where triples are connected to form the Web of Data.

The Semantic Web technology stack includes a vast set of technologies such as:

RDFS *RDF Schema (RDFS)* provides a data-modeling vocabulary for RDF data. RDF Schema is an extension of the basic RDF vocabulary [10].

OWL The W3C *Web Ontology Language (OWL)* is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things. OWL is a computational logic-based language such that knowledge expressed in OWL can be exploited by computer programs, e.g., to verify the consistency of that knowledge or to make implicit knowledge explicit. OWL documents, known as ontologies, can be published on the World Wide Web and may refer to or be referred from other OWL ontologies [28].

SPARQL The *SPARQL Protocol and RDF Query Language (SPARQL)* is a query language [30] and protocol [14] for RDF. The SPARQL Protocol [14] describes a means for conveying SPARQL Queries and updates to a SPARQL processing service via HTTP.

4.1.4 Linked Data

In Tim Berners-Lee's vision, the Semantic Web was not just a way to have semantics in a piece of data that is published somewhere, but it was a lot more, it represented the possibility of enabling a semantic layer composed by data – expressed with Semantic Web technologies – linked together that can be navigated and queried to find other, related data [7].

In [7] Tim Berners-Lee proposed the term *Linked Data* to name this possibility, enabled by the Semantic Web, to have data linked and expressed in RDF. To be called *Linked data* data must respect these four principles [7]:

- Use URIs as names for things.
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).
- Include links to other URIs so that they can discover more things.

By following Linked Data principles, datasets become interconnected, creating a network of knowledge. This interconnectedness allows for a richer understanding of data by traversing relationships.

Linked Data principles prescribe a way to build a uniform interface for the Web of Data, so they are essential to connect the Semantic Web. These principles are exploited to enable the navigation requirement that characterizes the *WoDT Digital Twins Platform Knowledge Graph.*

Once defined the principles under the concept of Linked Data, it was necessary to clarify and extend them in order to provide a protocol for managing Linked Data using the Web.

The *Linked Data Platform (LDP)* [47] specifies an HTTP-based protocol for reading/writing Linked Data on the Web. It does so by:

- imposing further constraints on HTTP requests and responses used to manipulate Linked Data – the HTTP protocol is too under-specified for interoperability

- imposing constraints on the lifecycle and representation of resources hosted on an LDP-compliant server

In particular, it is a Linked Data specification defining a set of integration patterns for building RESTful HTTP services that are capable of reading/writing RDF data. A *Linked Data Platform* is any client, server, or client/server combination that conforms in whole or in sufficient part to the LDP specification, which defines techniques for working with Linked Data Platform Resources over HTTP [47].

LDP-compliant servers support two types of *Linked Data Platform Resources*: *LDP RDF Sources* that are resources represented in RDF, and *LDP Non-RDF Sources* that are resources using other formats, like images, HTML, and so on.

The LDP recommendation [47] describes a special type of *LDP RDF Source* called *Container* which acts as a container of Linked Data Platform Resources. For example, a client could access the URI of a container on an LDP at some university to list all the curriculums and for each one all the courses and the teachers. Moreover, Containers respond to all the read/write requests for a resource. There are three different types of Containers: *Basic Container*, *Direct Container*, and *Indirect Container*. The difference between the three is in terms of the triples that are generated when a resource is added to the Container itself.

Basic Container It is the most basic variant of LDP Container providing generic storage in an LDP Server that generates only *containment triples* (Figure 4.2). *Containment triples* are a set of triples, maintained by the LDP Container (LDPC), that lists documents created by the LDPC but not yet deleted. These triples always have the form:

```
<LDPC URI> ldp:contains <document-URI>
```

Direct Container It adds flexibility on top of Basic Container, providing the possibility of managing *membership triples* (Figure 4.3). *Membership triples* use a domain-specific vocabulary and allow the creation of custom relationships to the new LDP Resource created.

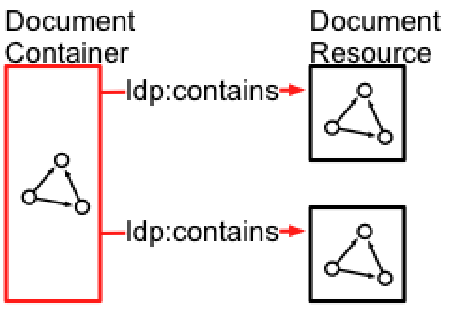

Figure 4.2: Example of Basic Container
Source: [38]

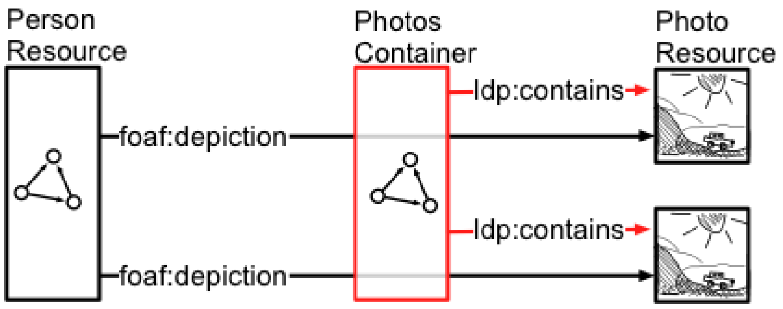

Figure 4.3: Example of Direct Container with a non-Container subject
Source: [38]

Indirect Container It is very similar to Direct Container, except that it is capable of having members whose URIs can be any resources rather than fixing it to the original URIs. The member URI of the membership triple could be based on the content of the newly inserted documents. Considering the example in Figure 4.3, the member URI is the one associated with the `foaf:depiction` membership triple that can be associated with any other URI, based on the inserted photo document or any information or non-information resource.

The recommendation in [47] defines the set of rules for HTTP operations to work compatibly with the *Linked Data Platform*.

The *Linked Data Platform* was studied and analyzed but not adopted in the *Web-based Web of Digital Twins* due to its strict HTTP-based design and because the current requirements are not aligned with the advantages that the *Linked Data Platform* provides and the problems that it solves. Its adoption would have resulted only in additional constraints, complexity, and verbosity.

4.2 Web of Things: vision and comparison

The requirements in section 3.4 set the need for a compatibility layer towards *Web of Things*. In this section, the *Web of Things (WoT)* paradigm is described and analyzed, comparing it with respect to the *Digital Twin* paradigm to establish their affinities and differences.

4.2.1 Overview of the WoT paradigm

Internet of Things was created to promote the networking and interoperability of devices by enabling devices to connect at the network layer via the *Internet Protocol (IP)*. However, at some point, it became apparent that devices also need to interoperate at the application layer to fully exploit the advantages of the Internet of Things vision.

Following this need, starting in 2007 the *Web of Things (WoT)* [51][29] paradigm was created exploiting the World Wide Web as a middleware for enabling application-layer interoperability. The Web of Things is the ability to use modern Web standards and the REST architectural style on embedded devices as an application layer over Internet of Things. In this way, any application can use and create mashups of different IoT devices (so WoT Things) no matter what technology, network protocol, or standard is used under the hood. This vision, starting in 2017, was standardized by the *W3C* [35][32][34].

The paradigm is based on a simple abstraction called *Thing*. The definition of *Thing* is:

> *An abstraction of a physical or a virtual entity whose metadata and interfaces are described by a WoT Thing Description, whereas a virtual entity is the composition of one or more Things.* [35]

Following this definition, everything can be abstracted in terms of a *WoT Thing* described by its *WoT Thing Description* [32]. Then, the WoT Thing Description is no more than a way to describe an interface, the “entry point” of an IoT instance or a virtual entity. The WoT Things are used in the Web of Things paradigm to represent at the conceptual level the abstractions that form the Web of Things and that allow the interoperability layer on top of IoT devices.

The metamodel offered by Web of Things [35] used to model WoT Things – via their WoT Thing Descriptions – consists of three elements, also called *Interaction affordances*: *Properties*, *Events* and *Actions*. An *Interaction affordance* contains the metadata of a Thing that shows and describes the possible choices to Consumers, thereby suggesting how they may interact with the Thing [35]. In particular:

- *Property*: it exposes the state of a WoT Thing. Properties can be read, and optionally they may be writable and observable.

- *Event*: it exposes the events of the WoT Thing to consumers. Events can be generated at different levels: at the software level by the Thing itself (e.g., update of a Property) or by the associated IoT device (e.g., engine problem in a car).

- *Action*: it allows Consumers to invoke a function of the WoT Thing. Actions can be used for different objectives: manipulate the state, invoke a physical action on the associated IoT device, and so on.

As stated previously, the Thing Description is used to describe a WoT Thing, and it is composed of five parts:

- *Metadata*: that describes the Thing itself such as its ID, type, title, description, and so on.

- *Interaction Affordances*: they describe how the Thing can be used in terms of *Properties*, *Events* and *Actions*, so implicitly its model.

- *Schemas*: data schemas exchanged in input and output.

- *Security Definitions*: they provide metadata about the security mechanism necessary to use the Thing.

- *Web links*: they represent relationships with other Web resources (including other WoT Things as well).

An example of a Thing Description can be seen in Listing 4.1.

The Thing Description, thanks to its general and domain-independent metamodel, can be used to describe any interface on the Web. Every Thing must have its Thing Description. In the provided example, it is possible to notice that each *Interaction Affordance* has its own *Protocol Binding*. A *Protocol Binding* is the mapping from an *Interaction Affordance* to concrete messages of a specific protocol that enable Consumers to use the parent affordance [32]. The Thing is an abstraction, so an entity that exists independently of the Protocol Bindings specified in its Thing Description, so they are completely optional.

WoT Thing Descriptions follow the *Thing Description Information Model* [32] that consists of a set of class definitions that define the "abstract model" of a compliant Thing Description. In addition, to be able to serialize and deserialize them, in [32] it is provided the mapping of the *TD Information Model* to JSON-LD [48]. Mapping to JSON-LD also allows Consumers who do not understand semantics to consume the Thing Descriptions through classic JSON libraries. Moreover, to

Listing 4.1: Example of a WoT Thing Description – Source: [32]

```json
{
    "@context": [
        "https://www.w3.org/2022/wot/td/v1.1",
        { "saref": "https://w3id.org/saref#" }
    ],
    "id": "urn:uuid:300f4c4b-ca6b-484a-88cf-fd5224a9a61d",
    "title": "MyLampThing",
    "@type": "saref:LightSwitch",
    "securityDefinitions": {
        "basic_sc": {"scheme": "basic", "in": "header"}
    },
    "security": "basic_sc",
    "properties": {
        "status": {
            "@type": "saref:OnOffState",
            "type": "string",
            "forms": [{
                "href": "https://mylamp.example.com/status"
            }]
        }
    },
    "actions": {
        "toggle": {
            "@type": "saref:ToggleCommand",
            "forms": [{
                "href": "https://mylamp.example.com/toggle"
            }]
        }
    },
    "events": {
        "overheating": {
            "data": {"type": "string"},
            "forms": [{
                "href": "https://mylamp.example.com/oh"
            }]
        }
    }
}
```

support also all the Consumers that do not support the JSON representation format, the *Thing Description Ontology*[2] was created allowing the implementation of the *TD Information Model* in RDF.

Considering the REST architectural style, these approaches allow the exchange of Thing Descriptions to respect the REST *self-descriptive messages* constraint. In addition, also the *HATEOAS* constraint is fulfilled – to enable the *Uniform Interface* constraint of REST – by the use of *hypermedia controls* as serialization of Protocol Bindings. A *hypermedia control* is the machine-readable description of how to activate an affordance [35]. Thing Descriptions support two types of hypermedia controls:

- *Links*: they allow having relationships with other Web resources – including other WoT Things – and they are used for discovery and navigation. To be self-descriptive, W3C WoT Links follows the *Web Linking* specification [41], so they are composed of:
  - a link context
  - a relation type
  - a link target
  - some target attributes (optional)
- *Forms*: they are used to describe the Protocol Bindings of the specified Interaction Affordances available to consumers. They operate as "request templates" for consumers to submit the request for an affordance. To allow the support of IoT protocols and the creation of self-descriptive messages, *WoT Binding Templates* [34] can be used.

The Thing Descriptions are used in a lot of different domains. For this reason, they provide a way to enable the possibility of adding domain-related knowledge through the *TD Context Extensions* [35]. *TD Context Extensions* are used to integrate additional semantics to the content of the Thing Description. For example, in the Listing 4.1 the SAREF ontology is used. It is the mechanism that is used to import additional Protocol Bindings and Binding Templates [34].

In addition, the *W3C Web of Things* recommendation [35] specify the *Thing Model* as a way to describe classes of Things that have the same capabilities, allowing the creation of templates for Thing Descriptions. Their definition is useful only to describe the model of a class of Things, but it is not enough to enable consumers to interact with the single instances.

In *Web of Things* deployments, various topologies are possible. The main two are, that are described in [35] are:

[2]`https://www.w3.org/2019/wot/td`

Direct interaction It is the simplest topology where the consumer has direct access to the real Thing and so to the real entity behind it (Figure 4.4). The Thing Description can be directly exposed by the IoT devices or by software service on top of the entity.

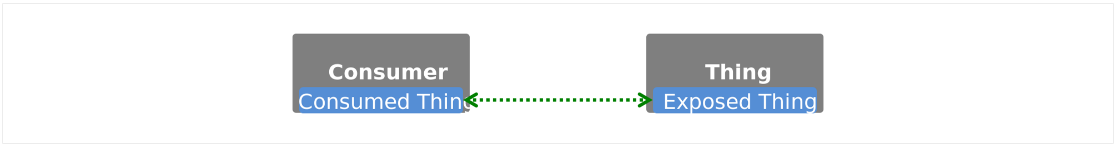

Figure 4.4: Direct interaction in Web of Things
Source: [35]

Intermediaries Several patterns introduce the need to have an *Intermediary* between the consumer and the real Thing (Figure 4.5). *Intermediaries* are software entities that can be used for proxying between devices and networks or to enable the *WoT Digital Twin* paradigm. In the former case, the Intermediary is usually used to implement additional security mechanisms (such as TLS), to provide additional Protocol Bindings, or to generally augment Things capabilities. In the latter case instead, they are used to providing state caching, deferred updates, or services like predictions or simulation.

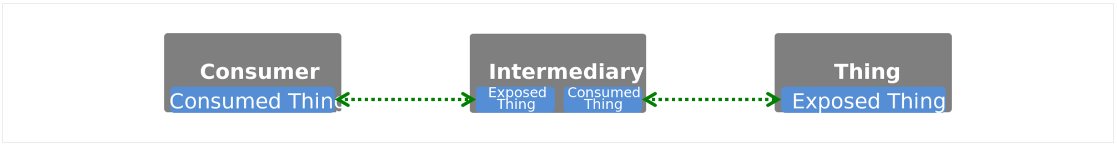

Figure 4.5: Intermediaries in Web of Things
Source: [35]

4.2.2 Digital Twins in Web of Things

Recently, the Web of Things community has developed a renewed interest in Digital Twins. In the WoT Recommendation [35], Digital Twins are defined in the following way:

> *A digital twin is a type of Virtual Thing that resides on a cloud or edge node. Digital Twins may be used to represent and provide a network*

> *interface for real-world devices which may not be continuously online, may be able to run simulations of new applications and services before they get deployed to the real devices, may be able to maintain a history of past state or behaviour, and may be able to predict future state or behaviour.* [35]

Digital Twins in Web of Things are enabled by a topology that includes *Intermediaries*. The task of the Intermediary is to provide state caching, deferred updates, and all the services that the Digital Twin may expose.

The definitions and the examples provided offer a vision that seems similar to the modern ones. However, the W3C WoT Recommendation [35] does not elaborate on the requirements and the typical characteristics of a WoT Digital Twin nor does it provide details on Digital Twin implementation. They define Digital Twins only as a general concept that could be represented as a WoT Thing and so described by a Thing Description. This is not necessarily bad, as it opens up opportunities for comparison and possible alignments with the vision followed by this thesis, Web of Digital Twins.

4.2.3 Comparison between Digital Twins and WoT Things

Following the idea described in the section 3.2 and considering the popularity of the *Web of Things* paradigm [35] is necessary to analyze it in depth and try to provide a first – personal – comparison between *Digital Twins* and *WoT Things* that permits to understand if, and in case how, it is possible to provide an integration of the two paradigms.

In the following, the comparison is described from different points of view to analyze the differences and the similarities in a structured way. Consider that the comparison is performed on the versions of the recommendations available at the time of the analysis, so the one referenced [35] [32]. New features that emerged after are not considered.

Definition

It is interesting to start the comparison from the definitions of the two concepts. We already defined both of them, but to ease the reader the definitions are proposed again here.

Digital Twin

> *Digital Twins refer to the ability to clone a Physical Asset (PA) through a software counterpart during its life cycle. The Digital Twin has a*

> *model that reflects all the properties, relationships, and characteristics of the physical asset that are important for the analyzed context.* [39][44]

WoT Thing

> *An abstraction of a physical or a virtual entity whose metadata and interfaces are described by a WoT Thing Description, whereas a virtual entity is the composition of one or more Things.* [35]

One of the points that outstands among the others in the Digital Twin definition is the fact that it is a *real software entity* that stores the Digital Twin's state and enables different types of services and computations. So, when Digital Twins are involved it means that the corresponding software entities are deployed somewhere and mirror the associated Physical Assets – objects, people, processes, and so on. One advantage of being a software entity is that it can exist even before the Physical Asset exists, and so can mirror it for the entire life-cycle [39].

Instead, WoT Things are "abstractions" that do not consider an additional software entity as a mandatory element in the architecture. So, WoT Things are a more general concept that is useful as an abstraction to represent and describe something that *may* use a software entity. The Thing – and so its Description – could be:

- *directly exposed by the Physical Asset*: this case is more common on IoT devices. This possibility is in contrast with the Digital Twin vision.
- *exposed by an Intermediary*: exploiting the layered system constraint of the REST architectural style. This one provides a software entity used for state caching and related services, that is placed on top of the Physical Asset, abstracting it at the digital level. These *Intermediaries* could act as the "software entities" that are missing from the *Things* to be able to represent a *Digital Twin*.

Metamodel

The metamodel defines the modeling capabilities of the paradigm. In the following, the possibilities of WoT Things are compared to the Digital Twin metamodel.

Properties In *Digital Twins*, properties are read-only because they can be updated only by the *shadowing process*. In addition, Digital Twins' properties are observable by default.

In *WoT Things* properties can always be read and optionally be writable. In addition, their observability is not mandatory.

As it is possible to see, in both paradigms it is possible to model both static and dynamic properties. The difference here is focused on the possibility of writing the value of a property – not allowed on Digital Twins, optional in WoT Things – and on the observability optionality.

Relationships In *Digital Twins*, relationships are a first-class concept that is fundamental to enable the Web of Digital Twins. Considering that they model the relationships between Physical Assets in the real world, relationships between Digital Twins are dynamic, domain-oriented, observable, read-only and so managed via the *shadowing process*.

In *WoT Things* it is possible to create generic links with other Web resources – including other WoT Things, but not restricted to them. Links follow the Web Linking specification [41], so they can also be domain-oriented, but they are not treated as a first-class element of the Thing's metamodel so they can not be observed by consumers – it is not possible to specify an *Interaction Affordance* and a *Protocol Binding* for them. Finally, they can be dynamic, but they are not read-only.

Events Events are very similar in both paradigms, with the possibility of modeling Physical Asset's or augmented events.

Actions Actions are very similar in both paradigms. Consumers can request the invocation of an action that is executed directly on the Physical Asset or augmented.

Interaction patterns

The Digital Twin paradigm, and in particular the Web of Digital Twins vision, provides several interaction patterns at the individual and at the ecosystem level that can be exploited by Consumers. It is needed to understand if WoT Things can support them and at which level.

Snapshot Digital Twins' current state must be available to Consumers. This allows the Consumer to obtain the complete state of the associated Physical Asset as if he is directly interacting with it. Moreover, if *Memorization* property is considered, it should be possible to request a Snapshot of a past instant in time.

In *WoT Things* we cannot specify a form at the Thing level to obtain the complete view over a Thing, and so there is no concept of a Snapshot of the whole Thing[3]. However, there is the possibility to specify additional properties to represent the complete state of a Thing – but only in the latest state.

Observation In *Digital Twins*, Consumers can observe the state evolution – so the snapshot evolution.

Instead, *WoT Things* does not provide a way to observe the whole Thing because it does not allow observation forms at the Thing level[4]. It is possible to observe properties and events independently, but not links.

Action invocation As defined previously, Digital Twins can have actions that can be exploited by consumers in their logic.

Action invocation is natively supported by *WoT Things.* Consumers can request the invocation of an action described in the *Thing Description.*

Ecosystem interactions At the ecosystem level, Digital Twins offer two main interaction patterns: *query* and *observation.* Both of them require Platform support to be enabled.

The same applies to *WoT Things* that does not specify anything about interaction at the ecosystem level. However, following the REST architectural style, it is possible to build layered systems providing Intermediaries that can expose these services.

Features

Bidirectional shadowing In *Digital Twins* the *shadowing process* is a central feature because it allows the Digital Twins to mirror in real-time the associated Physical Assets. As it is bidirectional, it also enables the mirroring of actions towards the Physical Assets. This results in a software counterpart that can fully replicate the Physical Assets.

Generic *WoT Things*, being abstract concepts not necessarily supported by a software entity that maintains the state, do not have a well-defined and specific shadowing process. Instead, if we consider the more specific view over *WoT Digital Twins* [35], where *Intermediaries* are involved, then there are possibilities to represent the current state of the Physical Asset and, at the same time, mirror the actions to it.

[3]From the Web of Things (WoT) Thing Description 1.1, recently published as the new recommendation, additional support is provided and could be exploited in the future.

[4]Also for the Observation Interaction pattern, the newly released Web of Things (WoT) Thing Description 1.1 recommendation provides support to be exploited in the future.

Memorization Being able to represent a Physical Asset means not only to be able to represent its present state but also to retain its past data which can be used to offer services like predictions, simulation, and so on. This functionality is important in *Digital Twins* because Consumers can obtain the Digital Twin state in a specific point in the past.

The WoT recommendation does not address data historicization. This is evident in the WoT model as it lacks a mechanism for correctly specifying versioning metadata, leaving the responsibility to external services or ontologies. One possible solution at the implementation level is to use *Intermediaries* to provide Memorization functionalities. However, this solution does not solve the expressivity problem of the WoT model.

Augmentation *Digital Twins* can offer new properties, actions, or events totally at the software level, so not originally supported by the Physical Asset, resulting in an augmentation of its capabilities.

Things that are directly exposed by the IoT devices cannot expose augmented properties, actions, or events because the augmentation layer is missed – i.e., the Physical Asset directly exposes the elements. Instead, if *Intermediaries* are involved, even *WoT Things* can expose augmented elements.

Quality attributes

Openness and Interoperability For the objective of this thesis, in *Digital Twins* openness is more related to the possibility of having relationships in an ecosystem composed of more organizations that deal with different domains. It is important that Physical Assets and Consumers can communicate with the technology or the protocol most fitted to their needs, but the main goal here is to support multi-organizational and multi-domain scenarios.

Instead, in *WoT Things*, considering that the Web of Things paradigm is born to cope with heterogeneity in the IoT scenario, it is more related to the technology and protocol openness.

Fidelity One of the new and emerging concepts in Digital Twins is *Fidelity*. Although Fidelity is yet an open problem – representation, control, and so on – it is one of the concepts that differentiate Digital Twins from the other paradigms.

Web of Things does not describe anything that is related to the *Fidelity* concept, and *WoT Things* are Things despite their ability to mirror the associated Physical Asset. This is because the Thing abstraction simply represents an interoperable and open interface.

Discussion

At first glance, *Web of Things* and *Digital Twins* seem not so different, but analyzing them more deeply it is possible to note some main problems in the *Web of Things* view compared to the Digital Twins one:

- *Things are not necessarily supported by a software entity*: in the generic vision – so without considering *WoT Digital Twins* – *WoT Things* are only mere abstractions, so they may not be supported by any additional software layer. This poses several problems in their ability to cope with some *Digital Twins* features such as *Bidirectional Shadowing*, *Memorization*, and *Augmentation.*
- *Not complete and writable metamodel*: Digital Twins have strong requirements in terms of metamodel, and the *WoT Things* one has some pitfalls when dealing with Digital Twins one:
  - *Properties may be writable and are not observable by default*: a Thing may be modified by a Consumer, instead a Digital Twin must not – it only relies on the shadowing process.
  - *Weak concept of Relationship*: Things have only the concept of generic link. Moreover, they are writable and cannot be observed by Consumers.
- *Missing Interactions at the Thing level*: the *Web of Things* recommendation does not provide a way to get and observe the complete status – *Snapshot* – of a Thing.
- *Absence of the Fidelity concept*: the *Web of Things* recommendation does not contain anything related to the *Fidelity* concept with respect to Things.

Some of the issues, such as the weakness of the Relationship concept, the Memorization feature, and the Interaction patterns at the Thing level, stem from the expressive limitations of the current WoT model which prevent them from being accurately discerned and specified. However, in some cases, it can be solved through the "specialization" of the *Web of Thing* recommendation, which means adding constraints to Things to be able to model what is mandatory for a Digital Twin. For this reason, it is possible to exploit the features made available by the *Web of Things* recommendation to create Things that respect the principles of Digital Twins. In particular, the integration point can be obtained through the *WoT Thing Description* that could serve as a descriptor of a Digital Twin.

Therefore, this thesis work tries to provide a compatibility layer towards *Web of Things* proposing that:

A Digital Twin can be considered as a WoT Thing, but not every WoT Thing can be considered a Digital Twin.

Referring to what has been said, an example of a Thing that cannot be considered a Digital Twin is a sensor that directly exposes its Thing Description to describe the interfaces to access its latest value without any additional software layer. This one cannot be considered a Digital Twin because it does not have the required software entity to provide the basic functionalities of a Digital Twin. Finally, a WoT Digital Twin that does not offer the basic interaction patterns described above is only considered a Digital Twin in the WoT vision, and not in the Web of Digital Twins one – followed in this thesis.

Chapter 5

WWoDT: the Web-based WoDT

Following the requirements, and the analysis of the Web standards, protocols, and paradigms, this chapter presents the proposed design for the *Web-based Web of Digital Twins (WWoDT)*. The chapter begins with a high-level overview of the proposed idea and architecture, highlighting the need for the provided Specifications for creating WoDT-compliant ecosystems of Digital Twins. Afterward, the Abstract Architecture of the system is described.

5.1 High-level description

The creation of ecosystems of heterogeneous Digital Twins requires the support of *dynamic*, *open*, and *long-lived* systems. The *Web* and its *REST* architectural style, as stated in the section 3.2, were used in several systems [11][12] to inherit their properties, and in the same way they are exploited in this thesis for the design of the *Web-based Web of Digital Twins*.

The base idea of the work is centered over the need for *WoDT-compliant Digital Twins* and *WoDT-compliant Platforms* that can cooperate and co-exist within the same ecosystem despite their technology, domain, or organization in a seamless and interoperable way. To achieve this, the responsibilities, the metadata, the features, the interaction patterns, and so on need to be specified. This thesis provides two Specifications that are the base for the proposed work:

- *WoDT Digital Twin Specification*
- *WoDT Digital Twins Platform Specification*

The details of these Specifications will be described afterward, for the moment it is only needed to understand that any Digital Twin that wants to join a WoDT ecosystem must follow the *WoDT Digital Twin Specification*, becoming a *WoDT*

Digital Twin. Moreover, any WoDT ecosystem is enabled by a software platform that follows the *WoDT Digital Twins Platform Specification* – also called *WoDT Digital Twins Platform.*

WoDT Digital Twins are identified by a URI and they are described by:

- *Digital Twin Knowledge Graph (DTKG)*: it models the current state of a WoDT Digital Twin as a navigable Knowledge Graph. This Knowledge Graph is generated by the WoDT Digital Twin itself using its reference ontologies – based on its domain of interest. The DTKG enables Consumers to obtain the current state of the associated Physical Asset directly from its Digital replica, with all the advantages described before.

- *Digital Twin Descriptor (DTD)*: it allows a WoDT Digital Twin to present itself to generic Consumers and WoDT Digital Twins Platforms. It is similar to an API Specification for a Web service or a Thing Description for a WoT Thing, and it contains the exposed interfaces and the metadata about a WoDT Digital Twin. In addition to that, it enables a compatibility layer towards the technology or protocol used to implement it (such as *Web of Things*).

Both descriptions are formalized compatibly with the Linked Data principles. The separation is only at the *RDF-document* level, both of them are Knowledge Graphs on which it is possible to reason altogether.

The high-level architecture that describes the interactions between the elements of a WoDT ecosystem is shown in Figure 5.1.

Specifically, the WoDT Digital Twins Platform is a software entity that manages the creation of the *WoDT Digital Twins Platform Knowledge Graph* by merging the Digital Twins Knowledge Graphs obtained from the registered WoDT Digital Twins. Based on the WoDT Digital Twins Platform Knowledge Graph, the WoDT Digital Twins Platform offers services at the application level and acts as a service layer to enable applications on top of it.

As per requirement, the WoDT Digital Twins can register themselves to the WoDT Digital Twins Platforms, or they can be added by Platform administrators so that everyone can compose their *Web of Digital Twins.* In this vision, each WoDT Digital Twins can be part of different Platforms (as shown in the Figure 5.1). Indeed, a single Digital Twin can be of interest for different realities. For example, a Digital Twin of a person may be of interest to the national health system, to the company for which he works, to the gym he goes to every day, and so on. This precisely enables the creation of contextualized Web of Digital Twins, such as the Cesena's WoDT, the National Health System's WoDT, or the Andrea's WoDT, that enable the creation of digital and contextualized representation of the

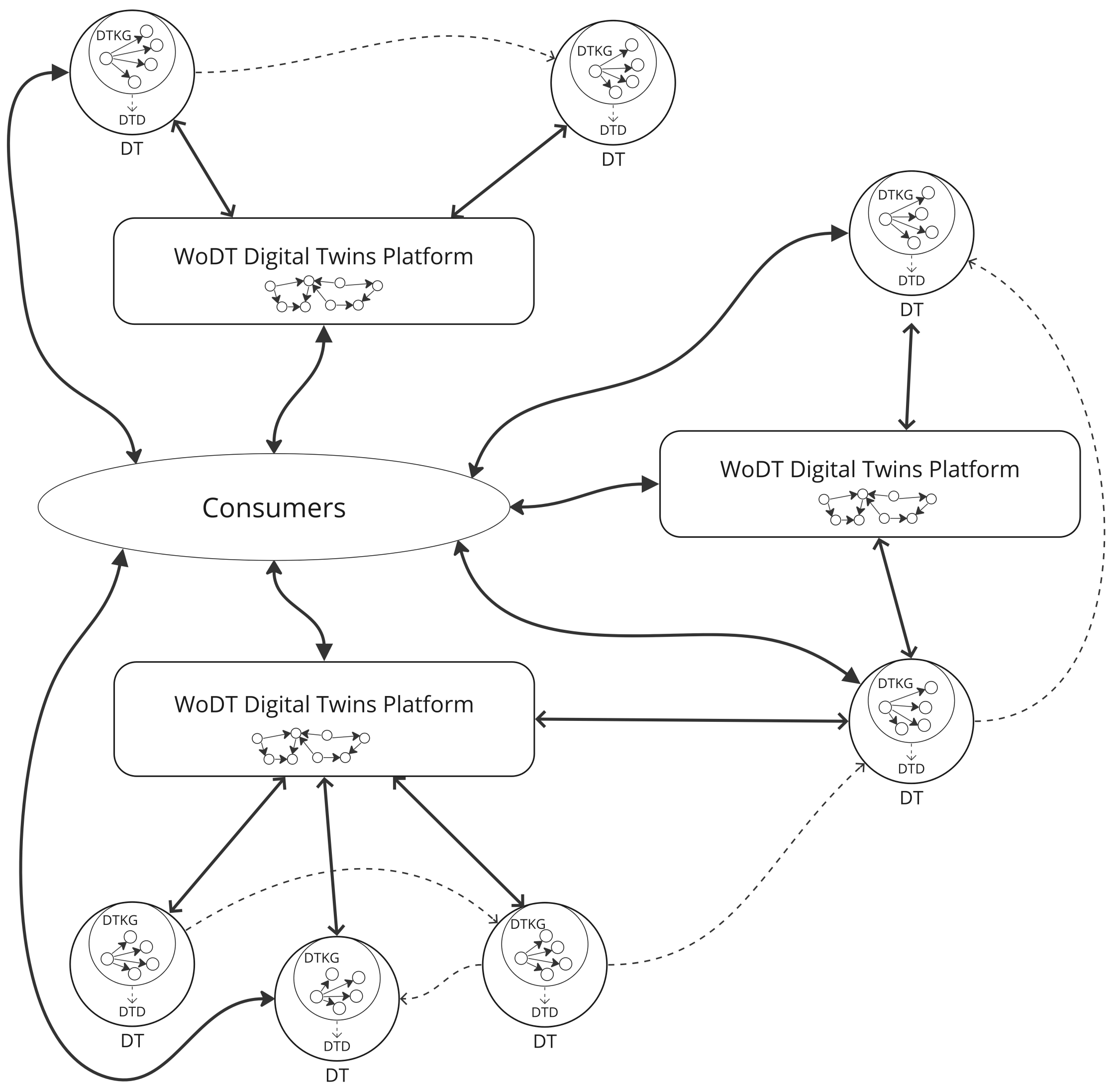

Figure 5.1: High-level architecture of a WoDT ecosystem

reality of interest exploiting Digital Twins – implemented with the different technologies available – which are in relation among themselves in a cross-domain and cross-organizational vision. The proposed requirements and design serve to go in this direction.

In the registration process, the *Digital Twin Descriptor (DTD)* is used to present the WoDT Digital Twin to the WoDT Digital Twins Platform. The Platform uses the DTD to understand how to observe the WoDT Digital Twin in order to receive its *Digital Twin Knowledge Graph (DTKG)* snapshots. Then, the

Platform merges the received DTKGs and DTDs altogether, building the *WoDT Digital Twins Platform Knowledge Graph* used to provide services to Consumers and applications on top. The WoDT Digital Twins Platform Knowledge Graph is an overall view of all or part of the ecosystem, a navigable Knowledge Graph that contains both domain data and metadata of all the registered WoDT Digital Twins. It is necessary to have a centralized view supporting the services that reason at the ecosystem level or need to aggregate data. This is a different approach with respect to distributed queries that simplifies the consistency of queries with the tradeoff of having a centralized view.

In addition, the WoDT Digital Twins use the DTD also to present themselves to Consumers who want to interact directly with them – for data retrieval, actions, and so on.

The DTD does not impose constraints over communication protocols and technologies, and the DTKG represents the current state of the Digital Twin as if it came directly from the Physical Asset. That, in addition to the *REST* architectural style and the use of vocabularies and ontologies within messages, enables the interoperability and the Uniform Interface all over the WoDT ecosystem. The obtained Uniform Interface hides the technologies used internally by the Digital Twins and Platforms developers, enabling heterogeneity.

As shown in the Figure 5.1, Consumers can interact either with the WoDT Digital Twins Platform or with the WoDT Digital Twins directly – *DT-as-a-Service* –, respecting the independence requirement described previously (3.4.2).

5.2 Specifications

In this section, the most prominent concepts of the *WoDT Digital Twin Specification* and the *WoDT Digital Twins Platform Specification* are described. The Specifications are the basis for all the work done and proposed in this thesis. However, they are not complete, so they are still in a draft state.

All the details about the requirements and the terminology definitions are not proposed again, as they have already been described in the previous sections.

5.2.1 WoDT Digital Twin Specification

The *WoDT Digital Twin Specification* serves as an additional layer built on top of existing Digital Twins, allowing them to join WoDT ecosystems despite the technologies or protocols used internally. Therefore, it attaches itself to the existing shadowing process. A *WoDT-compliant Digital Twin* is designed exploiting existing Web technologies and standards and following the REST architectural style.

Identification: WoDT Digital Twin

A WoDT Digital Twin must be identified by a URI, that allows it to be identified uniquely globally.

A WoDT Digital Twin URI must not change, therefore it must remain the same for the whole lifecycle (*PURL*-like services). Hence, different URIs mean different Digital Twins and vice versa:

$$URI_1 \neq URI_2 <=> DT_1 \neq DT_2$$

Identification: Physical Asset

As stated in the requirements (section 3.4.2), a Physical Asset must be identified with the best-suited identifier that reflects its domain, e.g., a license plate for a car, a health card number for a person, and so on. The *Physical Asset Identifier (PHID)* may not be global, but it must be unique within its domain of interest. The reason is that the identifier, reflecting the domain, depends on the specific Physical Asset at hand and on the domain in which it is inserted. Moreover, the Physical Asset Identifier – in the usual case – cannot be hosted, so it cannot be a URI or a URL for example, and hence it must be a generic ID.

Metamodel

A WoDT Digital Twin must follow the *Web of Digital Twins* metamodel. Specifically, in this specification the metamodel is considered not only composed of Properties, Relationships, and Events but also of *Actions* that represent the actions that can be invoked on the Physical Asset by interacting with its Digital Twin (one of them).

In the following, the constraints for each element of the metamodel are presented:

- *Properties*: they must be read-only, dynamic, and observable. Properties must be read-only because they must be modified dynamically through the shadowing process and no other external entity can modify them. For example, if a Digital Twin of a light bulb is considered, no one except the light bulb itself can set its current state (through shadowing).

- *Relationships*: relationships must be read-only (updated only through the shadowing process), dynamic, observable, and unidirectional. In addition, they must reflect an existing domain-oriented relationship between the associated Physical Assets.

- *Events*: events must be observable and domain-oriented.

- *Actions*: they must be explicitly modeled to reflect the complete state of the associated Physical Asset. A Digital Twin may expose different protocols to invoke each action. In addition, as stated previously, Digital Twin's actions do not change directly the state of the Digital Twin, but they can generate a change in the Physical Asset state.

Interaction patterns

After having described what are the offered Interaction patterns, it is necessary to start to describe more specifically how they must be provided. Each WoDT Digital Twin must be completely autonomous, in the sense that it should not rely on external entities to carry out its work.

Snapshot A WoDT Digital Twin must provide an affordance to access its current snapshot, its Digital Twin Knowledge Graph. A WoDT Digital Twin is a *non-information resource*, and it is identified by a URI. An HTTP GET request on the WoDT Digital Twin URI, following [46] and [15], must respond with a `303 (See Other)` status code that must have the `Location` HTTP header set to the URL of the current representation of the Digital Twin Knowledge Graph, offering the Snapshot interaction pattern.

For this initial specification, the *Memorization* functionality applies only to the Digital Twin Knowledge Graph. Therefore, a WoDT Digital Twin may offer its Consumers older versions of the DTKG. Considering that the DTKG is a web resource and that it is needed to provide historicization at the resource level, the *Memento protocol* was chosen. Hence, each WoDT Digital Twin that wants to implement Memorization must follow the *Memento protocol* [13][50]. Each different historical version of a Digital Twin Knowledge Graph becomes a *Memento* identified and located by a URL. A WoDT Digital Twin, when implementing the Memento protocol, must offer a *Memento TimeGate* that guides the Consumers to the right Memento and it must include the required HTTP Link Headers in the responses (defined by the protocol).

Observation A WoDT Digital Twin must provide an affordance to observe its Digital Twin Knowledge Graph evolution. Each time a Digital Twin changes, the Consumer must be notified with the new consistent snapshot. The observation pattern can be offered with multiple and different protocols like WebSockets or WebSub.

Action invocation Consumers must be able to invoke the actions provided by a WoDT Digital Twin. The protocol bindings and the required inputs must be described within the Digital Twin Descriptor, where multiple protocols may be supported.

Usually, the available actions change depending on the state of the Physical Asset and the corresponding WoDT Digital Twin. For this reason, the Digital Twin Knowledge Graph must list all the invokeable actions in the current state. If a Consumer tries to invoke an action when it is not available – considering its state – the WoDT Digital Twin must deny the execution and inform the Consumer using the best-suited strategy with respect to the protocol used for the request.

Descriptions: Digital Twin Descriptor

WoDT Digital Twins have one and only one Digital Twin Descriptor. Therefore, the multimodel idea [44], where multiple Digital Twin models can be defined for the same Physical Asset to capture different aspects, is obtained through different WoDT Digital Twins, with different URIs, that are associated with the same Physical Asset.

Physical Assets during their lifecycle could change their model. For this reason, the Digital Twin Descriptor, apart from being always aligned with the Physical Asset model, must be versioned accordingly. In this way, each change is reflected in its version, and Consumers – and caches – can understand if the resource is stale or not.

The contents of a Digital Twin Descriptor depend only on the WoDT Digital Twin model, so they do not depend on its current state. Therefore, no property values, relationship instances, or available actions are described in the Digital Twin Descriptor. Instead, it contains a complete description of the WoDT Digital Twin in terms of metadata and affordances – with all the supported protocols – to access its state and interfaces.

At the specification level, to enable interoperability and create a Uniform Interface, there are two main choices: a custom *media-type* or the use of existing media types in addition to a *vocabulary* or an *ontology*. The first solution is not ideal for the Digital Twin Descriptor because the goal is to also be aligned with the *Semantic Web* community and use the Descriptor as an enabling element for compatibility with other paradigms, such as the Web of Things. For this reason, the strategy followed and proposed in this thesis is to have an *RDF-based* Digital Twin Descriptor to enable the Uniform Interface. RDF-based representations provide more flexibility in this context because they essentially move the interoperability problem one layer above – from media types to vocabularies. This added flexibility is what would allow the use of automated inference with formal alignments (via `owl:sameAs`, subclasses, and so on) that enable compatibility. Therefore, the specification only provides an *Abstract conceptual model* – following a similar approach to the WoT recommendation [32] – for the Digital Twin Descriptor that can be implemented with the preferred and compatible paradigm. In this scenario, the WoDT Digital Twins Platforms need to describe the supported implementation of the Digital Twin Descriptor – so the ones that they support in the registration process.

In the following, it is presented the Digital Twin Descriptor's *Abstract conceptual model* using a tabular representation:

- *Element*: the name of the element of the *Abstract conceptual model.*
- *Mandatory*: it indicates if the element presence is mandatory or not (abbreviated with *Mand.* in the following tables).
- *Value type*: the data type used to represent the element value.
- *Description*: a short description summarizing the rationale for the element.

Digital Twin Descriptor root The content of a Digital Twin Descriptor is described in table 5.1. Any Digital Twin Descriptor implementation must respect the contents described in table 5.1.

Table 5.1: Structure of the Digital Twin Descriptor

Element	**Mand.**	**Value type**	**Description**
Version	yes	string	The version of the Digital Twin Descriptor of the WoDT Digital Twin.
WoDT Digital Twin URI	yes	URI	The URI of the WoDT Digital Twin.
Physical Asset Identifier	yes	string	The ID of the associated Physical Asset (PHID).
WoDT Digital Twin type	yes	URI	The value must be the class – in the domain ontology – that represents the Digital Twin type.
Shadowing latency	no	number	The latency in the shadowing process. The value is expressed in milliseconds.
Deployment country	no	string	The country where the WoDT Digital Twin is deployed. The value is expressed according to *ISO 3166-1 alpha-3* standard.

continues on next page

Memento TimeGate	no*	URL	The Memento TimeGate HTTP URL. *It is mandatory only when Memorization is available on the WoDT Digital Twin.
WoDT Digital Twins Platforms	yes	List of URL	The WoDT Digital Twins Platforms where the WoDT Digital Twin is present (registered or added). The list may be empty – when it is not registered to any platform.
Observation affordance	yes	List of *Form*	The affordances to observe the evolution of the Digital Twin Knowledge Graph. It is a list because it is possible to specify different protocols.
Properties	yes	List of *Property*	All the possible properties of a WoDT Digital Twin. The list may be empty – when it does not have any property relevant to the context.
Events	yes	List of *Event*	All the events emitted by a WoDT Digital Twin. The list may be empty – when it does not emit any event relevant to the context.
Relationships	yes	List of *Relationship*	All the possible relationships of a WoDT Digital Twin. The list may be empty – when it does not have any relationship type relevant to the context.
Actions	yes	List of *Action*	All the possible actions offered by a WoDT Digital Twin. The list may be empty – when it does not offer any action relevant to the context.

Property Each *Property* of a WoDT Digital Twin represented in the Digital Twin Descriptor must respect the contents described in Table 5.2.

Table 5.2: Structure of a Property in the Digital Twin Descriptor

Element	**Mand.**	**Value type**	**Description**
Property value type	yes	URI	The type of the property value.
Domain predicate	yes	URI	The URI that identifies the domain-related predicate used to describe the property in the Knowledge Graph.
Is Augmented	yes	boolean	It states if the property is an augmented one or not. If not, it means that it is mirrored from the associated Physical Asset.
Tolerance	no	number	The tolerance in the value, with respect to the real value, presented by the WoDT Digital Twin.
Read affordance	yes	List of *Form*	The affordance to read the current value of the property. It is a list because it is possible to specify different protocols.
Observation affordance	yes	List of *Form*	The affordance to observe the evolution of the value of the property. It is a list because it is possible to specify different protocols.

Event Each *Event* of a WoDT Digital Twin represented in the Digital Twin Descriptor must respect the contents described in Table 5.3.

Table 5.3: Structure of an Event in the Digital Twin Descriptor

Element	**Mand.**	**Value type**	**Description**
Event data type	yes	*Data Schema*	The data structure of the data sent within the event.

continues on next page

Is Augmented	yes	boolean	It states if the event is an augmented one or not. If not, it means that it is mirrored from the associated Physical Asset.
Observation affordance	yes	List of *Form*	The affordance to observe the event. It is a list because it is possible to specify different protocols.

Relationship Each *Relationship* of a WoDT Digital Twin represented in the Digital Twin Descriptor must respect the contents described in Table 5.4.

Table 5.4: Structure of a Relationship in the Digital Twin Descriptor

Element	**Mand.**	**Value type**	**Description**
Target type	yes	URI	The type of the target WoDT Digital Twin.
Domain predicate	yes	URI	The URI that identifies the domain-related predicate used to describe the relationship in the Knowledge Graph.
Read affordance	yes	List of *Form*	The affordance to read the active instances of the relationship. It is a list because it is possible to specify different protocols.
Observation affordance	yes	List of *Form*	The affordance to observe the evolution of the instances of the relationship. It is a list because it is possible to specify different protocols.

Action Each *Action* of a WoDT Digital Twin represented in the Digital Twin Descriptor must respect the contents described in Table 5.5.

Table 5.5: Structure of an Action in the Digital Twin Descriptor

Element	**Mand.**	**Value type**	**Description**
Action type	yes	URI	The URI that identifies the type of the action at the domain level. Consumers use this type to select and recognize the needed action (e.g., `saref:ToggleCommand`).
Is Augmented	yes	boolean	It states if the action is an augmented one or not. If not, it means that it is mirrored from the associated Physical Asset.
Required input	no	*Data Schema*	The input data structure that the action takes as input.
Action ID	yes	string	The Action ID used to refer to the action inside the Digital Twin Knowledge Graph.
Action invocation affordance	yes	List of *Form*	The affordance to invoke the action. It is a list because it is possible to specify different protocols.

Data Schema It describes the structure of the data being sent. Its serialization depends on the specific implementation. For example, a *Thing Description-based* implementation could use the *Data Schema* structure [32].

Form It contains all the protocol information that is necessary for automatic handling – without out-of-band data. In addition to that, the form must contain the *media type*[1] that describes the serialization used for sending and receiving the data with the protocol defined in the form. The bindings between the media type and the *Data Schema* are defined by the specific implementation of the Digital Twin Descriptor. For example, a *Thing Description-based* implementation could use *Web of Things Binding Templates* [34].

The Digital Twin Descriptor enables the WoDT Digital Twins to present themselves to Consumers without the need for *out-of-band* information or fixed interfaces. This is aligned with the constraints to obtain a Uniform Interface in the

[1]`https://www.iana.org/assignments/media-types/`

REST architectural style, in particular to the *self-descriptive messages* and *hypermedia as the engine of application state (HATEOAS)* ones.

Fidelity As said previously, for the Fidelity concept the objective is not to provide a comprehensive definition, but instead to start to think about the places where it is useful. Fidelity meta-data acts as an assurance that the Digital Twin makes respect to its ability to mirror the Physical Asset of interest. For this reason, we can immediately see that a Consumer must be able to understand how good the WoDT Digital Twin is in mirroring the Physical Asset. As it is possible to note in the *Abstract conceptual model*, Fidelity metadata is useful even in the Digital Twin Descriptor. In fact, there is Fidelity metadata that needs to be specified independently of the particular state of the WoDT Digital Twin. In the proposed *Abstract conceptual model*, Fidelity is composed of two different concepts: *Latency* and *Tolerance.*

Latency is described by the latency in the shadowing process, and by the country of deployment of the WoDT Digital Twin itself. The first one is useful to understand the latency in the shadowing process itself, so the time needed for the WoDT Digital Twin to be in sync. The second one is needed by Consumers to be able to estimate the possible network delay in communications with the WoDT Digital Twin.

Tolerance refers to the tolerance in the property's value concerning the actual value sensed in the real world. This is because properties' values usually come from data acquired by sensors, so for Consumers tolerance may be necessary. The tolerance, in the Digital Twin Descriptor, is considered with the same unit of measurement in which the property value is represented.

HTTP Link Headers The Digital Twin Descriptor can be hosted by the WoDT Digital Twin itself, or it may be hosted by an external entity. This flexibility comes from the fact that the Consumers are always able to navigate to it via the HTTP Link Header within the Digital Twin Knowledge Graph – Snapshot – response, as it will be described later. However, any request for the Digital Twin Descriptor has to include the following HTTP Link Headers in its response:

- HTTP Link Header to the WoDT Digital Twin Platforms where the associated WoDT Digital Twin is registered
  - *optional*: it is optional because a WoDT Digital Twin may not be registered on any Platform. In addition, there could also be a case where there are multiple links because, as described earlier, a WoDT Digital Twin may register itself or be registered to different WoDT Digital Twins Platforms at the same time.

  - the *relation type* is `registeredToPlatform`. It must be formalized in an ontology and provided as an *extension relation type*, as described in the *Web Linking* specification [41].
  - A consumer can use the links to navigate in the WoDT ecosystems where the WoDT Digital Twin is registered, and perform queries or access the provided services. These are the same links that are included in the *"WoDT Digital Twins Platforms"* field in the Digital Twin Descriptor – Table 5.1.

- HTTP Link Header to the Digital Twin Knowledge Graph of the associated WoDT Digital Twin
  - *mandatory*
  - the relation type is `currentStatus`. It must be formalized in an ontology and provided as an *extension relation type*, as described in the *Web Linking* specification [41].
  - It allows jumping easily from the DTD to the DTKG.

Caching In addition, Digital Twin Descriptors may be cached, reducing network latencies. To enable the caching mechanisms, a WoDT Digital Twin can include the following HTTP Headers in the Digital Twin Descriptor's response:

- *Cache-control*: the type of `Cache-control` can be chosen by the DT Developer.
- *ETag*: it needs to identify the specific version of the resource.
- *Last-modified*: to be compatible with the RFC 9110 [17], the `Last-modified` HTTP Header is recommended.

Finally, a WoDT Digital Twin must be able to respond to conditional GET requests for its Digital Twin Descriptor. If the resource has not changed, the response has the `304 (Not Modified)` status code, instead, if the resource has changed, the response has the `200 (OK)` status code and returns the current version of the Digital Twin Descriptor.

Descriptions: Digital Twin Knowledge Graph

The *Digital Twin Knowledge Graph (DTKG)* is generated by the WoDT Digital Twin based on the current state and domain ontology. It must be aligned with the metadata within the Digital Twin Descriptor, resulting in a domain-oriented description of the current state of the WoDT Digital Twin.

Following its name, the Digital Twin Knowledge Graph is a Knowledge Graph where the subject of each triple is the WoDT Digital Twin URI itself. This Knowledge Graph contains:

- *Properties values*: the current values of the properties. There cannot be links to access properties in the current state, but there must be values that Consumers can directly read. Properties in the Digital Twin Knowledge Graph are represented as *triples*: the predicate is the property's *domain predicate* specified in the Digital Twin Descriptor, and the object is the current value of the property.
- *Current relationships*: the current instances of relationships in which the WoDT Digital Twin – so the associated Physical Asset – is involved. In the current state only the current set of relationships are of interest, not all the possible ones. This allows us to represent the current situation of the associated Physical Asset more realistically. Relationships in the Digital Twin Knowledge Graph are represented as *triples*: the predicate is the relationship's *domain predicate* specified in the Digital Twin Descriptor, and the object is the URI of the linked WoDT Digital Twin. As per requirement, the linked WoDT Digital Twin may be owned by a different organization (openness and interoperability).
- *Current set of possible actions*: current set of actions that can be invoked on the WoDT Digital Twin based on its current state. Hence, it is a subset of all the actions. In the Digital Twin Knowledge Graph, for each possible action there is a triple that has as predicate `availableActionId` (to be defined in the WoDT ontology) and as object the *Action ID* associated with the action in the Digital Twin Descriptor. Therefore, a Consumer can understand which actions can be executed from the DTKG and then, following the HTTP Link Header, navigate to its DTD to get the affordances to invoke them.

Consumers obtain the Digital Twin Knowledge Graph invoking the Snapshot interaction pattern. As stated before, an HTTP GET request on the WoDT Digital Twin URI responds with the `303 (See Other)` status code indicating in the `Location` HTTP Header the URL of the Digital Twin Knowledge Graph. This enhances the navigation between different WoDT Digital Twins (by their relationships) and allows it to be compatible with the *Linked Data* principles.

Fidelity Regarding Fidelity, a possible idea – that needs further exploration – is the option of having a *Fidelity score* in the Digital Twin Knowledge Graph. The *Fidelity score* could be obtained by a formula designed by the DT Developer and based on both the Fidelity metadata within the Digital Twin Descriptor – to weight the different elements depending on the objective of the Digital Twin – and the current status of the Digital Twin to be able to understand how the Digital Twin is performing with respect to the motivation for which it was designed originally. The objective is to provide Consumers a way to understand quantitatively how much a WoDT Digital Twin can reflect accurately its associated Physical Asset, and how much it can trust the data for high-risk decisions. Moreover, only the WoDT Digital Twin Developer or Designer can extract the formula because it is the only one that can weigh the Fidelity characteristics of the DT itself.

HTTP Link Headers As specified in the requirements, it must be possible to jump from the Digital Twin Knowledge Graph to the Digital Twin Descriptor. Therefore, in the Snapshot response, an HTTP Link Header with relation type `hasDescriptor` (to be defined in the WoDT ontology) that points to the URL of the DTD must be set.

Caching The Digital Twin Knowledge Graph aims to provide a domain-oriented representation for the WoDT Digital Twins and so of their associated Physical Assets. It is needed always an up-to-date version, therefore it must not be cached by general intermediaries, only the WoDT Digital Twins Platforms can. To avoid intermediaries caching the DTKG, the HTTP `Cache-control` Header can be set to the `no-store` value. In addition, in the DTKG response, there should be the `Last-Modified` HTTP Header to state the date and time it was last updated.

5.2.2 WoDT Digital Twins Platform Specification

The *WoDT Digital Twins Platform Specification* defines the responsibilities, the functionalities, and the constraints that a Platform must respect to form ecosystems of heterogeneous WoDT Digital Twins, compatibly with the Web of Digital Twins vision.

As stated, the Platform is needed to provide all the services that a single WoDT Digital Twin alone is not capable of supplying, or supplying efficiently. The main goal is the creation of the *WoDT Digital Twins Platform Knowledge Graph* obtained by merging the DTKGs, and optionally the DTDs, of the registered WoDT Digital Twins. The resulting Knowledge Graph describes the entire ecosystem of an organization or even multiple organizations, spanning different domains and acting as a navigable and discoverable service layer for the applications on top. Based on the Platform Knowledge Graph, the service layer is enriched with additional services that are necessary for a Web of Digital Twins.

A *WoDT-compliant Digital Twins Platform* is designed and developed following the REST architectural style.

Management of the Platform ecosystem

The *WoDT Digital Twins Platform Knowledge Graph* is a living representation of the ecosystem that Consumers can use to access and observe the real world, from a Digital and immaterial replica. The ecosystem is formed by the registered WoDT Digital Twins that are linked together in a graph, in particular a Knowledge Graph. The following management services are necessary to manage the ecosystem and each one can be implemented, compatibly with the REST architectural style, with the preferred technology or protocol.

Registration and observation Following requirements, to be part of the Platform ecosystem, a WoDT Digital Twin can register itself or it can be added to a WoDT Digital Twins Platform providing the Digital Twin Descriptor.

WoDT Digital Twins Platforms provide an endpoint for the registration process:

- The registration process takes the WoDT Digital Twin's DTD as input. A *WoDT-compliant Digital Twins Platform* should support all the compliant implementations of the Digital Twin Descriptor.
- The registration can be finalized only if at least one protocol for the observation of the WoDT Digital Twin is supported by the Platform, otherwise it must fail safely and compatibly with the protocol or the technology used.
- If the registration is performed by a Platform administrator and not by the WoDT Digital Twin itself, then the Platform notifies the WoDT Digital Twin of the registration by sending a request to the appropriate endpoint. Each WoDT Digital Twin, that can be added externally, must offer a notification endpoint.

The Platform, once the WoDT Digital Twin is registered, uses its Digital Twin Descriptor to obtain the Observation interaction pattern endpoint and start observing its DTKG evolution. The received Digital Twin Knowledge Graphs are merged with the other WoDT Digital Twins' DTKG, building the *WoDT Digital Twins Platform Knowledge Graph*. Moreover, also the Digital Twin Descriptors may be merged in the WoDT Digital Twins Platform Knowledge Graph. This enables more powerful queries that involve also Digital Twins metadata (e.g., deployment country or fidelity or protocol constraints). Hence, both the Digital Twin Knowledge Graph and the Digital Twin Descriptor are Knowledge Graphs. The separation is just at the RDF-document level, it is only a design choice for different reasons such as caching, usage, and so on, but it does not mean that they need to be separated in the WoDT Digital Twins Platform Knowledge Graph.

Therefore, the Platform does not connect directly to the registered WoDT Digital Twins' shadowing process, but they observe them like general Consumers. This is because the shadowing process is the responsibility of each WoDT Digital Twin which must be able to live alone or within an ecosystem.

The WoDT Digital Twins Platform Knowledge Graph, storing the latest snapshot of the registered WoDT Digital Twins, creates a local cache that Consumers can exploit to ask for Digital Twins data. The Platform may be closer, from a network-latency perspective, than the WoDT Digital Twin, so it could help in reducing network latencies when essential for the Consumer. However, the local cache may be temporarily outdated when the Consumer requests it, so it is a trade-off. If the Consumer requires the most up-to-date state, he should request it directly from the WoDT Digital Twin. Alternatively, if latency is an issue, it may use the local cache on the Platform if it is more convenient. Following this possibility, each registered WoDT Digital Twins URI, in the whole WoDT Digital Twins Platform Knowledge Graph, is mapped to a local URL so built:

```
{platform url}/wodt/{wodt digital twin uri}
```

Mapping each registered WoDT Digital Twin, so even relationship targets when registered, enables navigation directly at the Platform level in a centralized view without the need to jump between different servers all over the world. At this URL, a Consumer can get the local snapshot of the associated WoDT Digital Twin, as described afterward. A way to get the original URI is offered and described afterward.

WoDT Digital Twin's Model update Any changes in the Physical Asset model are reflected in the WoDT Digital Twin, specifically updating its Digital Twin Descriptor (including its version). When this happens, the WoDT Digital Twin must notify the WoDT Digital Twins Platforms where it is registered. A WoDT Digital Twins Platform provides an endpoint for the Digital Twin Descriptor update:

- The update process takes the updated Digital Twin Descriptor as input.
- The update can be finalized only if at least one protocol for the observation of the WoDT Digital Twin is supported by the Platform, otherwise it must fail safely and compatibly with the protocol or the technology used.

The update request is adequately processed, and it may result in the restart of the observation process or the deletion of Digital Twin's data.

WoDT Digital Twin deletion When a registered WoDT Digital Twin is deleted, it notifies the interested WoDT Digital Twins Platforms. A WoDT Digital Twins Platform provides an endpoint for the deletion notification that takes as input only the WoDT Digital Twin URI. The request results in the deletion of all the parts related to the WoDT Digital Twin from the WoDT Digital Twins Platform Knowledge Graph and in the removal of the local snapshot endpoint.

Interaction patterns

The WoDT Digital Twins Platform Knowledge Graph is the base for all the services provided at the Platform level. In the following, the Interaction patterns that represent the service layer offered by the WoDT Digital Twins Platform to Consumers are described.

Platform ecosystem snapshot An HTTP GET request on the WoDT Digital Twins Platform URL, following [46] and [15], must respond with a `303 (See Other)` status code that must have the `Location` HTTP header set to the current representation of the WoDT Digital Twins Platform Knowledge Graph, offering the Platform ecosystem snapshot interaction pattern. Considering HATEOAS and self-descriptive messages constraints of REST, it is possible to insert in the response the hypermedia controls to register a new WoDT Digital Twin to the Platform. This would allow agents or Digital Twins to automate the registration process.

Local WoDT Digital Twin snapshot As stated before, the WoDT Digital Twins Platform Knowledge Graph creates a local cache that Consumers can exploit to ask for Digital Twins data. The *Local WoDT Digital Twin snapshot* interaction pattern is meant to offer Consumers a way to access local snapshots. An HTTP GET request on the mapped local URL returns the local data about a WoDT Digital Twin. In addition, in the response, an HTTP Link Header with relation type `original` is specified to point to the original WoDT Digital Twin URI. Considering HATEOAS and self-descriptive messages constraints of REST, it is possible to insert in the response the hypermedia controls to update the WoDT Digital Twin DTD and to delete the WoDT Digital Twin from the Platform. This would allow Digital Twins to automate lifecycle management operations. Appropriate authentication and authorization systems are necessary.

Query on the Platform Knowledge Graph A WoDT Digital Twins Platform must provide an affordance to make SPARQL queries over the WoDT Digital Twins Platform Knowledge Graph. A WoDT Digital Twins Platform exposes a SPARQL endpoint, compatible with the SPARQL 1.1 Protocol [14] (only *query operations*, not updates), at the following URL:

```
{platform url}/wodt/sparql
```

The queries are performed only at the Platform level. However, following the REST architectural style – and in particular the *layered system* constraint – it is possible to create intermediaries that act as aggregators creating a complete or a partial view over different ecosystems composed by the interested WoDT Digital Twins Platforms allowing queries at a bigger scale.

Observation of the Platform Knowledge Graph A WoDT Digital Twins Platform must provide an affordance to observe the evolution of the WoDT Digital Twins Platform Knowledge Graph. The observation is only at the Platform level. However, following the REST architectural style – and in particular the *layered system* constraint – it is possible to create intermediaries that act as aggregators creating a complete or a partial view over different ecosystems composed by the interested WoDT Digital Twins Platforms allowing observation at a bigger scale.

Multi-model directory service A WoDT Digital Twins Platform must provide the *multi-model directory service* interaction pattern that from a Physical Asset Identifier returns the URIs of all the associated WoDT Digital Twins that are registered to the Platform. This interaction pattern is necessary because for the same Physical Asset multiple and independent WoDT Digital Twins can be available, each one with a different model, specialized for different applications.

5.3 Abstract Architecture

The Specifications are the basis for the creation of *Web-based WoDT-compliant ecosystems of heterogeneous Digital Twins*. Starting from them, there is the need to describe the internal responsibilities of each role, defining an *Abstract Architecture* that Developers can follow for the implementation of a Web of Digital Twins. In this section, the proposal for the Abstract Architecture is presented. It is not a mandatory architecture that needs to be implemented to be compliant, but it is a purely logical one to help describe components and responsibilities.

The proposed Abstract Architecture comes, apart from Specifications, from the objective of realizing Web of Digital Twins ecosystems of heterogeneous Digital Twins where Digital Twins use different technologies under the hood, can be deployed in the best-suited network node, and are useful alone and as part of an ecosystem – *DT-as-a-Service*.

The Abstract Architecture is shown in Figure 5.2.

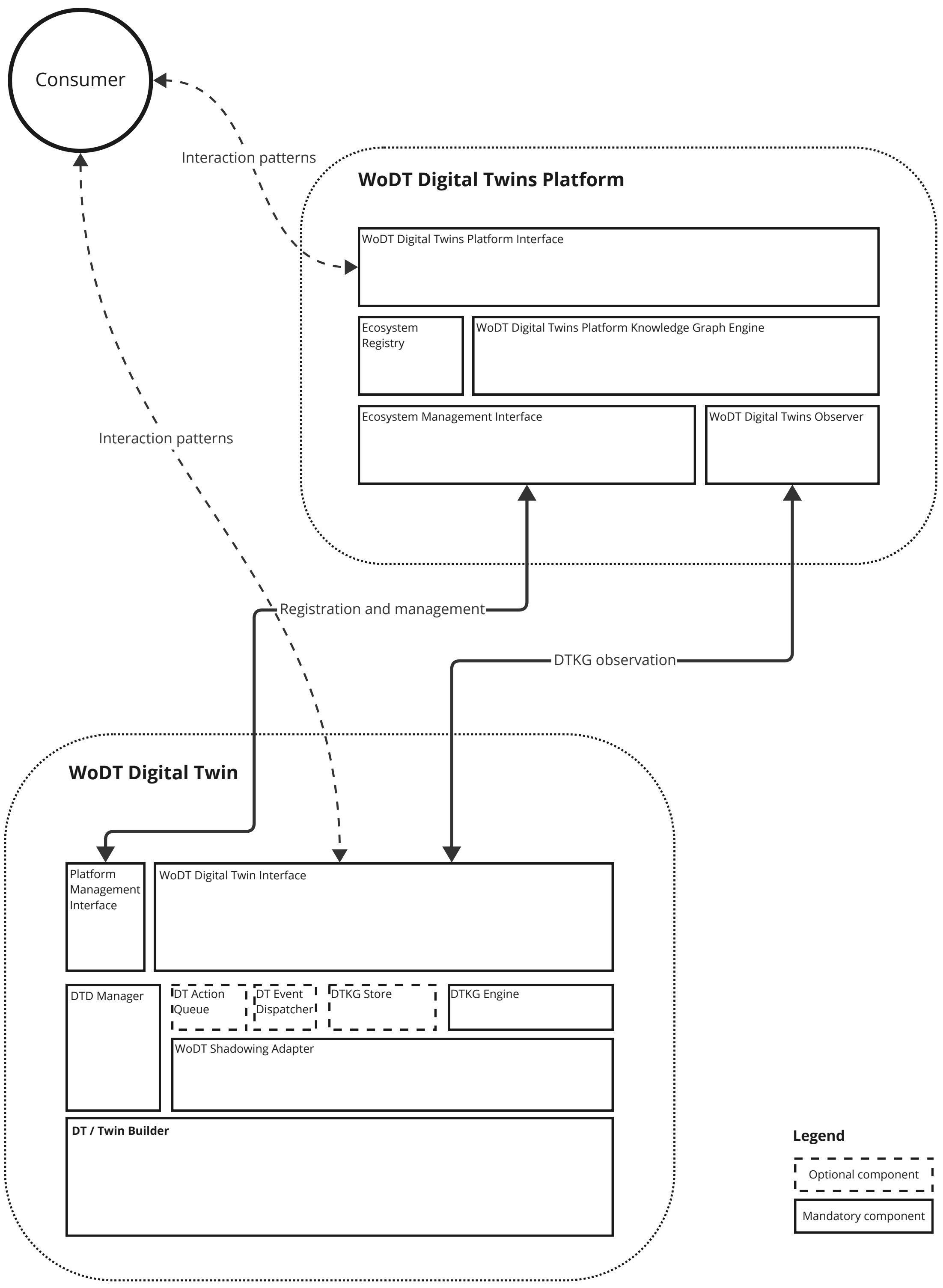
Consumer
Interaction patterns
Interaction patterns
WoDT Digital Twins Platform
WoDT Digital Twins Platform Interface
Ecosystem Registry
WoDT Digital Twins Platform Knowledge Graph Engine
Ecosystem Management Interface
WoDT Digital Twins Observer
Registration and management
DTKG observation
WoDT Digital Twin
Platform Management Interface
WoDT Digital Twin Interface
DTD Manager
DT Action Queue
DT Event Dispatcher
DTKG Store
DTKG Engine
WoDT Shadowing Adapter
DT / Twin Builder
Legend
Optional component
Mandatory component

Figure 5.2: Proposed Abstract Architecture

The figure (Figure 5.2) shows both the internal components and the interactions between the roles. In the figure, there is only one WoDT Digital Twin for simplicity, but every WoDT Digital Twin is connected in the same way to the Platform. The interactions are the ones described before in the Specifications:

- A WoDT Digital Twin can register, update, and delete itself to the WoDT Digital Twins Platform. In addition, it can be notified about the registration to a Platform.
- A WoDT Digital Twins Platform observes the WoDT Digital Twins' DTKG to build the WoDT Digital Twins Platform Knowledge Graph.
- Consumers can interact with both the WoDT Digital Twins Platform, for services at the ecosystem level, and directly with the WoDT Digital Twins, for services at the Digital Twin level. The interactions are based on the *Interaction patterns* presented above.

5.3.1 WoDT Digital Twin

Starting from the *WoDT Digital Twin*, the identified components are the following:

Digital Twin or Digital Twin Builder It represents the technology used to implement the Digital Twin. It could be a technology that allows the creation of a single Digital Twin, e.g., Eclipse Ditto, or it could be a *Digital Twin Builder* like Azure Digital Twins where multiple Digital Twins are managed. Both possibilities must be considered to enable ecosystem heterogeneity. Based on the technologies used here, the components on top could be more or less complex, considering their gaps with respect to the requirements.

WoDT Shadowing Adapter The *WoDT Shadowing Adapter* has the objective of adapting the Digital Twin metamodel to the Web of Digital Twins metamodel to cope with the gap present between the technology used and Web of Digital Twins. Depending on the Digital Twin technology, this component may be more or less complex. The WoDT Shadowing Adapter has the following responsibilities:

- Continuous observation of the *Digital Twin* or the *Digital Twin Builder* to extend data to the upper components.
- Digital Twin ID mapping to its WoDT Digital Twin URI.
- Dispatch events from the Digital Twin via the *DT Event Dispatcher* to be exposed by the *WoDT Digital Twin Interface*. This is needed only when the

Digital Twin itself is not able to expose events to Consumers – allowing, at the same time, their description in the Digital Twin Descriptor.

- Map the Digital Twin metamodel to the Web of Digital Twins metamodel.
- Map the data about its current status to a semantic representation that follows its domain ontology. The predicates are the ones described in the Digital Twin Descriptor. Therefore, it extends the Digital Twin shadowing process, providing a domain-oriented representation that can be stored on the *DTKG Engine*.
- Bridge the action invocation requests that are queued in the *DT Action Queue* to the Digital Twin to execute them. This is needed only when the Digital Twin itself is not able to expose actions to Consumers – allowing, at the same time, their description in the Digital Twin Descriptor.

DTD Manager It is the component dedicated to the management of the Digital Twin Descriptor. The DTD could be:

- Generated automatically by the component, processing the Digital Twin model and obtaining the necessary information from the other components of the architecture. The essential data needed is:
  - *Domain ontology*: needed to associate the data from the Digital Twin to domain predicates. Generally, it is a piece of information coded by the Digital Twin Developer.
  - *Digital Twin model*: obtained automatically from the Digital Twin technology used.
  - *Exposed interfaces*: they are necessary to describe the affordances in the Digital Twin Descriptor. Usually obtained from the *WoDT Digital Twin Interface* or coded.
  - *Platforms to which is registered*: from the *Platform Management Interface.*

  If the technology used is a *Digital Twin Builder*, then the *DTD Manager* manages the DTD of each single Digital Twin. Finally, following a DTD update, the DTD Manager must notify the Platform Management Interface that spreads the update to the interested Platforms.
- Hard-coded by the Digital Twin Developer.
- Managed by a management API by the Digital Twin administrator.

DTKG Engine It manages the Digital Twin Knowledge Graph of the WoDT Digital Twin. If the technology used is a *Digital Twin Builder*, then it manages the DTKG of each single Digital Twin. In addition, if the WoDT Digital Twin supports Memorization, then when it is updated it will store the previous DTKG in the *DTKG Store*, saving its history.

DTKG Store It stores the historical versions of the Digital Twin Knowledge Graph. It is an optional component because Memorization is not mandatory for a WoDT Digital Twin.

DT Event Dispatcher It manages events from the Digital Twin when the event observation is not supported natively by the technology used. The *WoDT Digital Twin Interface* exposes the events to the interested Consumers.

DT Action Queue It manages the action invocation request queue to be shadowed by the *WoDT Shadowing Adapter* when the action invocation from Consumers is not natively supported by the technology used. The choice to have a separate component is because it can buffer the action requests, leaving the *WoDT Shadowing Adapter* the only responsibility to act as a bridge.

Platform Management Interface It manages the registration to the WoDT Digital Twins Platforms. Specifically, its responsibilities are:

- Handling the Digital Twin Descriptor update and the deletion with notifications to the interested Platforms.
- Management of the registration notification endpoint for the registration to a WoDT Digital Twins Platform by an external entity.
- Automatic registration to a specific set of WoDT Digital Twins Platform, if configured to do so.
- Management of the list of WoDT Digital Twins Platforms to which the WoDT Digital Twin is registered. If the technology used is a *Digital Twin Builder*, then it manages a separate list for each Digital Twin.

WoDT Digital Twin Interface It offers the Interaction patterns described in the *WoDT Digital Twin Specification.* To satisfy the requests, it interacts with the other components of the architecture.

5.3.2 WoDT Digital Twins Platform

For what concerns the *WoDT Digital Twins Platform*, the identified components are the following:

Ecosystem Management Interface It handles the registration, the update, and the deletion of the WoDT Digital Twins to the Platform. The *Ecosystem Management Interface* is the responsible one for the validation of the Digital Twin Descriptors and the registration notification to externally added WoDT Digital Twins.

WoDT Digital Twins Observer The *WoDT Digital Twins Observer* observes the registered WoDT Digital Twins to get their updated DTKG.

Ecosystem Registry The *Ecosystem Registry* contains the registry of all the registered WoDT Digital Twins. Moreover, it contains the logic for the URI mapping.

WoDT Digital Twins Platform Knowledge Graph Engine It is the engine that manages the *WoDT Digital Twins Platform Knowledge Graph*, handling the continuous merging process of the DTKGs – and optionally of the DTDs – and the deletion of deleted WoDT Digital Twins. It offers an interface to get the whole Knowledge Graph, to get the local cached view over a registered WoDT Digital Twin, and to perform a SPARQL Query on the Platform Knowledge Graph.

WoDT Digital Twins Platform Interface It offers the Interaction patterns described in the *WoDT Digital Twins Platform Specification*. To satisfy the requests, it interacts with the other components of the architecture.

5.3.3 Interaction flows

To clarify the relationships between the different components of the *Abstract Architecture*, three base flows are discussed here, using UML Sequence Diagrams.

Registration process

The first one, shown in Figure 5.3, describes the interactions in the registration process.

Assuming that the WoDT Digital Twin already knows the URL of the WoDT Digital Twins Platform where it wants to register, the flow starts from the *Platform Management Interface* that obtains the updated Digital Twin Descriptor

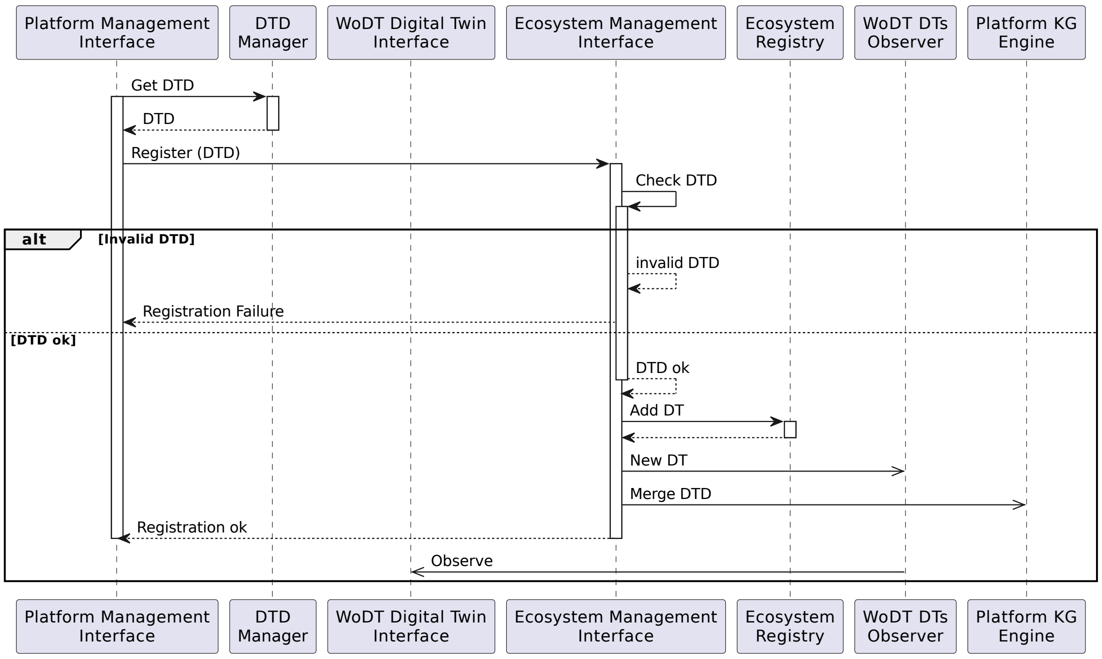

Figure 5.3: UML Sequence diagram for WoDT Digital Twin registration

from the *DTD Manager*. After that, it sends the registration request, with the DTD in the body, to the *Ecosystem Management Interface* of the Platform. The *Ecosystem Management Interface* validates and checks the received DTD. If the DTD is invalid, then it cannot proceed and returns the error to the Digital Twin *Platform Management Interface*. Otherwise, if the DTD is valid, it can finalize the registration process by adding it to the *Ecosystem Registry* and notifying the *WoDT Digital Twins Observer*, to start observing the new WoDT Digital Twin, and the *WoDT Digital Twins Platform Knowledge Graph Engine*, to merge the new Digital Twin Descriptor to the Platform Knowledge Graph. Finally, the registration is confirmed to the WoDT Digital Twin, and the *WoDT Digital Twins Observer*, as soon as it processes its Digital Twin Descriptor, starts observing the newly registered WoDT Digital Twin.

Query on the WoDT Digital Twins Platform Knowledge Graph

The second one, shown in Figure 5.4, describes the interactions that are necessary to query the WoDT Digital Twins Platform Knowledge Graph.

The Consumer, for simplicity, already knows the URL of the Platform. The flow starts from the Consumer that performs the SPARQL query on the *WoDT Digital Twins Platform Interface*. The latter requests the resolution of the query

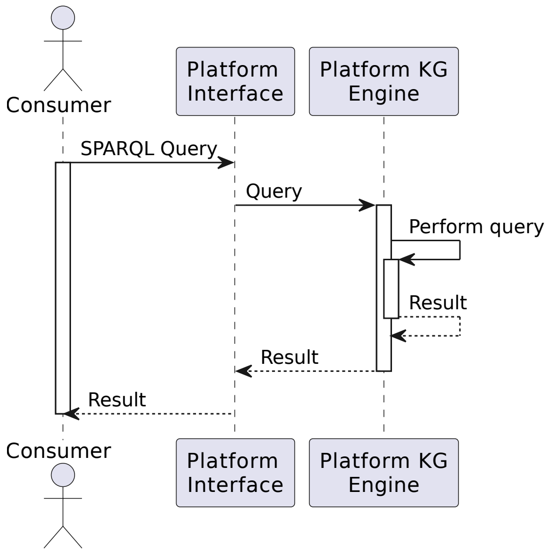

Figure 5.4: UML Sequence diagram for ecosystem query

to the *WoDT Digital Twins Platform Knowledge Graph Engine* that returns the results to the *WoDT Digital Twins Platform Interface*, immediately forwarded to the Consumer. The Consumer now can process the results of the requested query.

It is interesting to note that in this case the Consumer is already aware of the URL of the Platform, but it could also be the case that he is not. Thanks to the consistency of the Specification, the Consumer can use any WoDT Digital Twin, which he already knows, as an *entry point* for the ecosystem to be able to execute queries on it. In fact, by obtaining the DTD of the entry point, the Consumer will also obtain all the Platforms to which it is registered. This allows the Consumer to be able to perform queries on all the available Platforms.

Observation of a WoDT Digital Twin

The third one, shown in Figure 5.5, describes the interactions that are necessary for a Consumer to start to observe a WoDT Digital Twin.

After the Consumer obtains the URI of the interested WoDT Digital Twin, maybe performing a query on the ecosystem, he can request its Digital Twin Descriptor to the *WoDT Digital Twin Interface* to get its Observation affordance. The *WoDT Digital Twin Interface* obtains the latest Digital Twin Descriptor from the *DTD Manager* and returns it to the Consumer. The Consumer processes the DTD and uses the Observation affordance, against the *WoDT Digital Twin Interface*, to start observing the WoDT Digital Twin. When the *Physical Asset* updates, it shadows the event to the *Digital Twin* that it is reflected to the *WoDT*

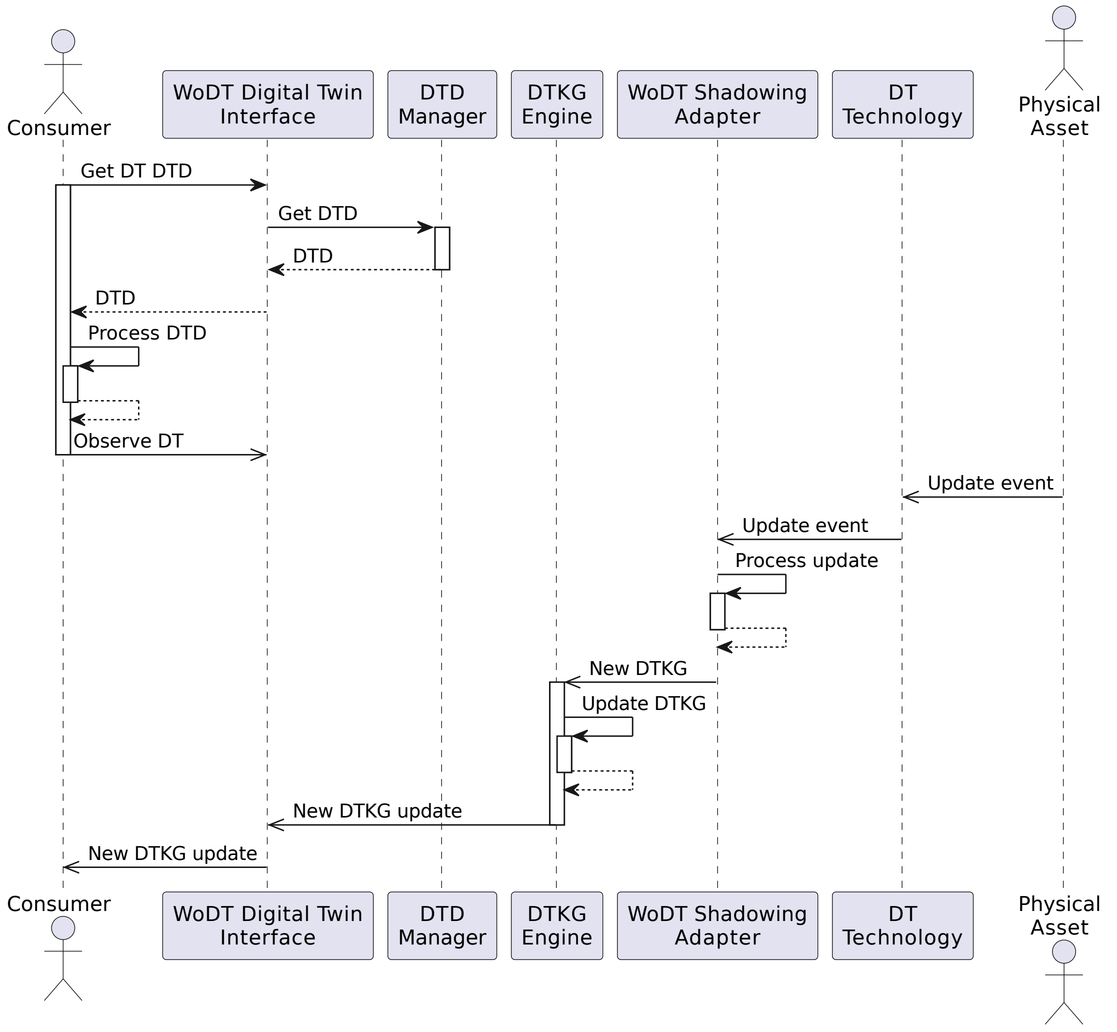

Figure 5.5: UML Sequence diagram for WoDT Digital Twin observation

Shadowing Adapter. The *WoDT Shadowing Adapter* processes the event and notifies the *DTKG Engine* that updates its internal Knowledge Graph and notifies the *WoDT Digital Twin Interface*. The *WoDT Digital Twin Interface* forward the event to all the observers, including the Consumer.

Chapter 6

Prototype

In this chapter, it is described the prototype for the *Web-based Web of Digital Twins* created to demonstrate the feasibility of the design proposed in the previous chapter. The chapter begins with a description of the prototype design that concretizes the proposed *Abstract Architecture*. Afterward, a description of the most prominent details of the prototype implementation is provided. Finally, to demonstrate the effectiveness of the proposed design and the realized prototype, an example use case is presented and described.

6.1 Prototype design

The objective of the thesis is to create ecosystems of heterogeneous Digital Twins using the Web. So, following this need, the prototype exploits the proposed design and proves the possibility of using different technologies for Digital Twins development in the creation of ecosystems of heterogeneous Digital Twins. Moreover, it is interesting to try different levels of generalization in the creation of *WoDT-compliant Digital Twins*: the creation of a library (or an extension), and ad-hoc approach (but easily generalizable). The first approach may be followed by Digital Twin developers or companies like Azure, Eclipse, and so on that want to extend the WoDT support to all their instances, instead, the second one may be the occasion for individual developers to integrate their creations into a WoDT ecosystem.

The chosen technologies for Digital Twins are *Azure Digital Twins* and *White Label Digital Twins Framework (WLDT Framework).* The first one was already described in the Background chapter (2), and the second one is a framework proposed at the academic level [42]. In addition, the Digital Twin Descriptor is implemented using the *Web of Things Thing Description* enriched with an additional vocabulary that fills the gap with the Abstract Conceptual Model proposed in the *WoDT Digital Twin Specification*.

The proposed prototype aims to test the feasibility of the design, so only basic functionalities are needed. For this reason, the *excluded* features are:

- Digital Twins domain or augmented events
- Augmentation
- Usage of Digital Twins Builder: only single Digital Twins are managed. This means that in the use of *Azure Digital Twins*, only one Digital Twin is exposed as a *WoDT-compliant Digital Twin*.
- Memorization

6.1.1 WoDT Digital Twins Platform

The *WoDT Digital Twins Platform* prototype is designed following exactly the *Abstract Architecture* described in the previous chapter.

6.1.2 WoDT Digital Twin: Azure Digital Twins

The *WoDT Digital Twin* based on the *Azure Digital Twins* service, shown in Figure 6.1, uses the *Azure* stack to provide the Digital Twin technology.

As it is possible to note, the architecture is completely based and follows the *Abstract Architecture*, leaving only the optional components outside. Hence, the interesting part here is the pipeline design that acts as the *Digital Twin technology*. The Azure-based pipeline is designed to be simple and cheap to deploy, so we know that other solutions are possible – maybe better – but this one allowed us to experiment with what we need simply and cheaply. The pipeline is composed of the following services:

- *Azure Digital Twins*: the Azure Digital Twins service is, as described in the Background, a PaaS (Platform as a Service) cloud service that allows the creation of Digital Twins graphs based on models of entire environments. In this case, only one of the Digital Twins in the graph is exposed as WoDT Digital Twin.

  The metamodel, except actions, followed by Azure Digital Twins is similar to the Web of Digital Twins one and a direct mapping can be implemented.

  So, this service was used to create and model the Digital Twin. Furthermore, being all the Azure suite event-driven, the graph update events are sent along the pipeline to be further processed and ready for the *WoDT Shadowing Adapter*.

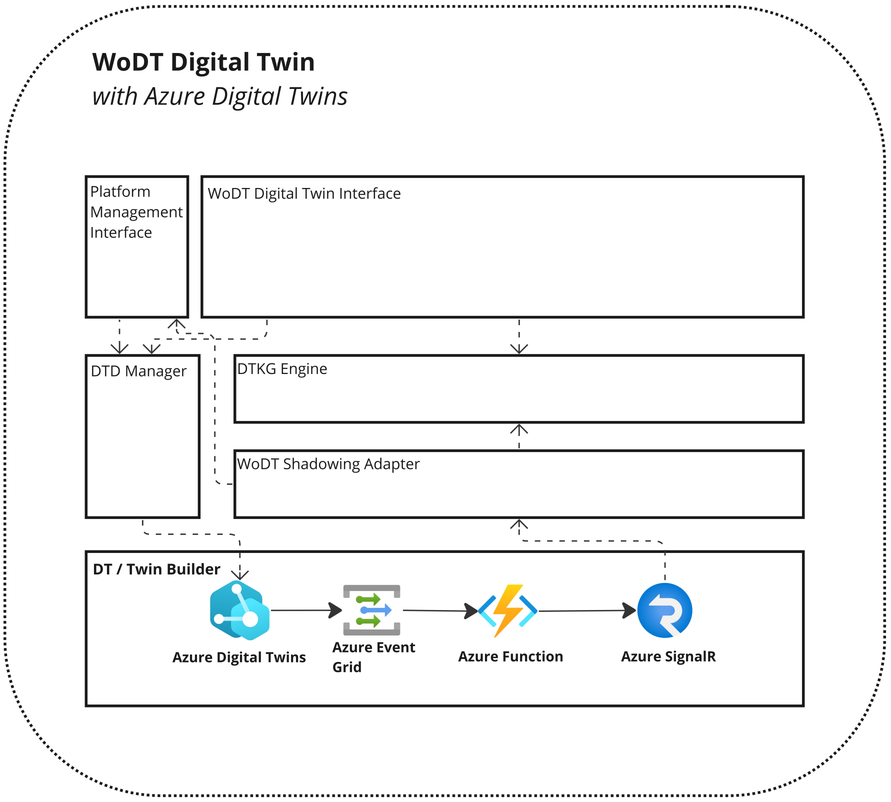

Figure 6.1: Prototype design of the WoDT Digital Twin based on Azure Digital Twins

- *Azure Event Grid*: events from Azure Digital Twins are sent to an *event route* configured with an *Azure Event Grid endpoint*, a fully managed Pub Sub message distribution service. This service allows exposing events from Azure Digital Twins, which as described in the Background supports a limited amount of *endpoints*, and sending them along the preferred pipeline or service.

- *Azure Function*: Azure Digital Twins send only patch events, so only the changes that happened. To ease the *WoDT Shadowing Adapter*'s work, an Azure Function is used to obtain the full updated snapshot of the Digital Twin, augmented with the data needed, to be sent to the WoDT Shadowing Adapter via the *Azure SignalR* service.

- *Azure SignalR*: the Azure SignalR service is used to expose the snapshot update events. In particular, this is the service used by the *WoDT Shadowing Adapter* to extend the shadowing process of the internal Digital Twin to become a WoDT-compliant Digital Twin.

The arrows in Figure 6.1 show the main dependencies among components. Particularly important here are the two with Azure Services. The first one, as already described, is between Azure SignalR and the WoDT Shadowing Adapter. It is needed to extend the shadowing process and receive the Digital Twin snapshots, in the Azure Digital Twins metamodel, to be adapted to the Web of Digital Twins metamodel. The second one, between the DTD Manager and the Azure Digital Twins instance, is needed to automatically create the Digital Twin Descriptor. Specifically, the DTD Manager component retrieves the Digital Twin model stored on Azure Digital Twins and converts it to a *WoDT-compliant Thing Description* that is further enriched with the remaining elements needed.

The remaining part of the architecture follows the Abstract Architecture.

6.1.3 WoDT Digital Twin: WLDT Framework

The second type of WoDT Digital Twin proposed in the prototype is based on the *White Label Digital Twins Framework (WLDT Framework).*

Before focusing on the proposed architecture, to better understand the components, it is necessary to make a brief overview of the *White Label Digital Twins Framework*. The *White Label Digital Twins* [42] is a framework that supports the design and development of Digital Twins. It intends to maximize modularity, re-usability, and flexibility to mirror any type of Physical Asset. The offered Digital Twin metamodel is aligned with the Web of Digital Twins one, allowing the modeling of *Properties*, *Events*, *Relationships*, and *Actions*. The main idea for a Digital Twin implemented with the framework is shown in Figure 6.2.

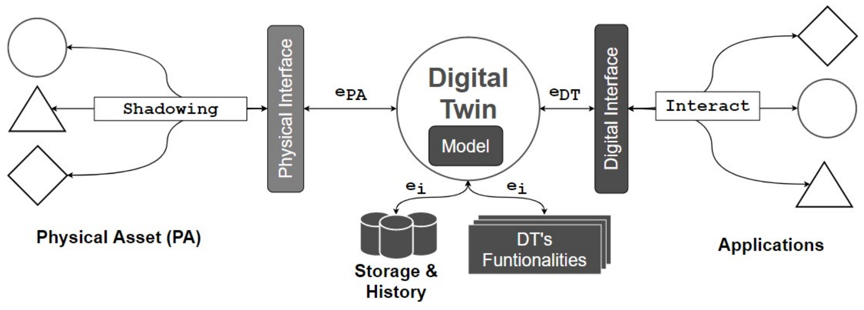

Figure 6.2: Abstract architecture of a Digital Twin with the WLDT Framework
Source: `https://github.com/wldt/wldt-core-java`

A WLDT instance provides a Digital Twin as a software entity that can be run in the cloud or on the edge. The *Physical Interface* is the component in charge of the shadowing process, while the *Digital Interface* handles the communication with the Application layer.

More specifically, in Figure 6.3, the main components that make up the architecture of the WLDT Framework and that allow the implementation of a Digital Twin are presented.

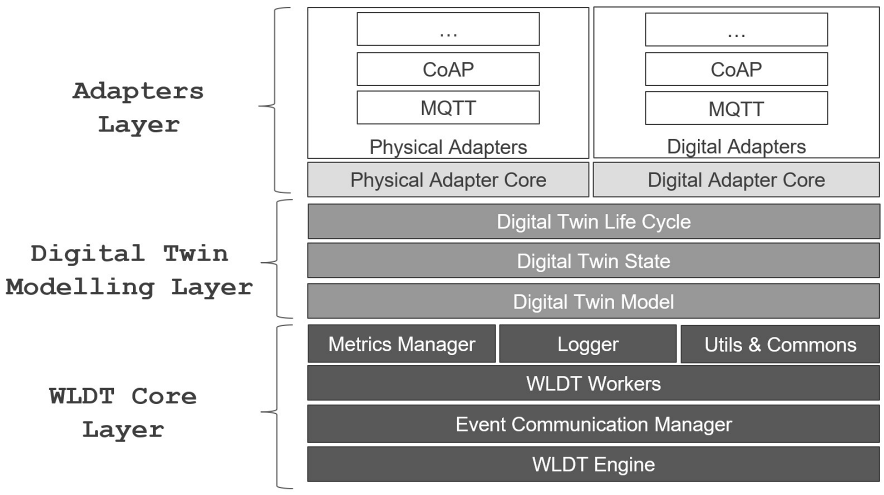

Figure 6.3: Components of the WLDT Framework
Source: `https://github.com/wldt/wldt-core-java`

- *WLDT Engine*: it is the core of the Digital Twin, and it orchestrates the internal modules of the architecture. The component is defined by a multi-thread engine that allows the execution of multiple *workers* simultaneously.

- *WLDT Event Bus*: it is the internal Event Bus that supports the exchange of data between the components.

- *WLDT Workers*: a *worker* is the basic executable entity of the *WLDT Engine*. Each active component of the architecture is implemented as a *WLDT Worker*.

- *Digital Twin State*: it is the component that manages the mirrored and the augmented state of the Digital Twin. The *Digital Twin State* is maintained in sync with the associated Physical Asset through the *Shadowing Model Function* that implements the shadowing process, receiving data from the specified *Physical Adapters* (for state updates) and *Digital Adapters* (for actions) and following the mappings specified by the Digital Twin Developer to concretize the Digital Twin model.

- *Physical Adapter*: the *Physical Interface*, that communicates with the Physical Asset, uses several *Physical Adapters* to support different communication types and protocols. Each *Physical Adapter*, apart from supporting a specific protocol, is dedicated to a specific subset of properties, relationships, events, and actions exposed by the Physical Asset through the specific protocol.

- *Digital Adapter*: the *Digital Adapters*, within the *Digital Interface*, are used to expose the Digital Twin state and functionalities to the Application layer. A Digital Twin Developer can define multiple *Digital Adapters* to support and offer different protocols.

The WLDT Framework already implements all the generic components, leaving the Developer only the duty to define the *Shadowing Model Function*, to concretize the Digital Twin model, and the necessary *Physical* and *Digital Adapters* to enable communication respectively with the Physical Asset and with Consumers.

Returning to the prototype, the proposed architecture of the WoDT Digital Twin prototype based on the WLDT Framework is shown in Figure 6.4.

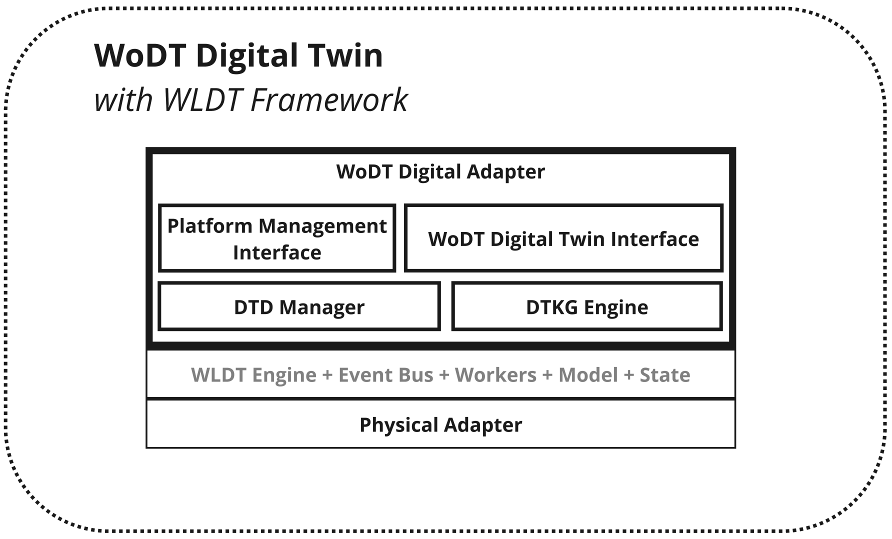
WoDT Digital Twin
with WLDT Framework
WoDT Digital Adapter
Platform Management Interface
WoDT Digital Twin Interface
DTD Manager
DTKG Engine
WLDT Engine + Event Bus + Workers + Model + State
Physical Adapter

Figure 6.4: Prototype design of the WoDT Digital Twin based on WLDT Framework

As shown, the components of the WLDT Framework are exploited to position the necessary elements of the WoDT Digital Twin Abstract Architecture. Considering that the WLDT Framework metamodel is equal to the WoDT metamodel, the *WoDT Shadowing Adapter* is not needed and all the remaining components of the Abstract Architecture are provided at the *Digital Interface* level. In particular, to enable the creation of WoDT-compliant Digital Twins a specific *Digital Adapter* – the *WoDT Digital Adapter* – is proposed. The *WoDT Digital Adapter* includes the components of the Abstract Architecture, so the *WoDT Digital Twin*

Interface, the *Platform Management Interface*, the *DTD Manager*, and the *DTKG Engine* as subcomponents. In this way, the proposed Digital Adapter can sit on the WLDT Framework architecture transparently, enabling any Digital Twin implemented with the framework to become a WoDT Digital Twin. Hence, the proposed design deals only with the necessary elements, maintaining the development of the Physical Adapter and the Shadowing Model Function a responsibility of the Digital Twin Developer.

Moreover, the metamodel alignment allows having direct support for all the metamodel elements, in particular actions and events, unlike the solution proposed with Azure Digital Twins (where actions have been left out for the moment).

6.2 Prototype implementation

Following the design proposed in the previous section, the description of the most important details of the prototype implementation is provided. For each software artifact, the technologies and, when needed, the prominent aspects of the code are described.

Regarding the prototype, for simplicity and to develop only what was needed for the example use case, the WoDT Digital Twins and the WoDT Digital Twins Platform follow the REST architectural style without the HATEOAS constraint, remaining at a lower maturity model.

6.2.1 WoT-based DTD and WoDT Vocabulary

As stated before, the *WoT Thing Description* is used to implement the Digital Twin Descriptor. The Thing Description does not have the whole semantics needed to represent the concepts of the Abstract conceptual model of the Digital Twin Descriptor so a vocabulary to fill the gap, called the *WoDT vocabulary*, is provided. The *WoDT vocabulary* is written using the common Semantic Web technology stack composed of *OWL (Web Ontology Language)*, *RDFS (RDF Schema)*, and *RDF (Resource Description Framework)*.

Firstly, it is necessary to describe the mapping between the Abstract conceptual model and the Thing Description model and then, based on that, define the *WoDT vocabulary*.

In Table 6.1, the implementation of the Digital Twin Descriptor via the WoT Thing Description is described. Consider that the mapping is performed on the versions of the recommendations available at the time of the design, so the one referenced [35] [32]. New features that emerged after are not considered.

Table 6.1: Mapping of the Abstract conceptual model to the WoT Thing Description

Element	**Description**
Version	Data property – `version` – in the WoDT vocabulary.
WoDT Digital Twin URI	Set as the ID of the Thing, using the `id` field.
Physical Asset Identifier	Data property – `physicalAssetId` – in the WoDT vocabulary.
WoDT Digital Twin type	Set as the `@type` field at the Thing level. This allows to specify the type of Digital Twin using the domain ontology.
Shadowing latency	Object property in the WoDT vocabulary – `shadowingLatency` – with a `Duration` range from the *OWL-time* ontology.
Deployment country	Data property – `deploymentCountry` – in the WoDT vocabulary.
Memento TimeGate	Defined exploiting the *Hypermedia Controls* vocabulary definitions of the Thing Description. In particular, it is set as a *Link* with the `timegate` relation type, following the *Web Linking* Specification and the *Memento* protocol.
WoDT Digital Twins Platforms	Defined exploiting the *Hypermedia Controls* vocabulary definitions of the Thing Description. In particular, it is set as a *Link* with a custom relation type defined in the WoDT vocabulary as an Object property – `registeredToPlatform` –, following the *Web Linking* Specification.
Observation affordance	At the time of the design, the Thing Description does not provide a way to observe the whole Thing. For this reason, the chosen strategy is to enforce the creation of a property named `snapshot` that contains an observation form. The `snapshot` property has the `observable` and `readOnly` fields set to `true`.

continues on next page

Properties	Set using the *Property* interaction affordance of the Thing Description. Each property has the `observable` and `readOnly` fields set to `true`.
Events	Set using the *Event* interaction affordance of the Thing Description.
Relationships	At the time of the design, the Thing Description does not provide a way to observe or add metadata to *Links*. For this reason, the chosen strategy is to model relationships with the *Property* interaction affordance of the Thing Description. Each property, that corresponds to a Digital Twin relationship, has the `observable` and `readOnly` fields set to `true`.
Actions	Set using the *Action* interaction affordance of the Thing Description.

In the tables 6.2, 6.3, 6.4, and 6.5 are described the mappings for each element of the metamodel, respectively for *Properties*, *Events*, *Relationships* and *Actions*.

Table 6.2: Mapping of the Abstract conceptual model Property to the WoT Thing Description

Element	**Description**
Property value type	Set via the `@type` field at the Property level.
Domain predicate	Custom field that uses the `domainPredicate` Object property defined in the WoDT vocabulary as a field name. The value of the field is the URI of the predicate associated with that property using the domain ontology.
Is Augmented	Custom field that uses the `augmentedInteraction` Data property defined in the WoDT vocabulary as a field name. The value of the field is `true` in case the property is an augmented one, `false` otherwise.

continues on next page

Tolerance	Custom field that uses the `propertyTolerance` Data property defined in the WoDT vocabulary as a field name. The value is the tolerance of the property as previously defined in the *WoDT Digital Twin Specification*.
Read affordance	Set as a *Form* instance specified with the `readproperty op` field.
Observation affordance	Set as a *Form* instance specified with the `observeproperty op` field.

Table 6.3: Mapping of the Abstract conceptual model Event to the WoT Thing Description

Element	**Description**
Event data type	Set with the `data` field of the Thing Description, using the WoT `DataSchema`.
Is Augmented	Custom field that uses the `augmentedInteraction` Data property defined in the WoDT vocabulary as a field name. The value of the field is `true` in case the event is an augmented one, `false` otherwise.
Observation affordance	Set as a *Form* instance specified with the `subscribeevent op` field.

Table 6.4: Mapping of the Abstract conceptual model Relationship to the WoT Thing Description

Element	**Description**
Target type	Set via the `@type` field at the Property (relationship) level. This allows to specify the target type using the domain ontology.
Domain predicate	Custom field that uses the `domainPredicate` Object property defined in the WoDT vocabulary as a field name. The value of the field is the URI of the predicate associated with that relationship using the domain ontology.

continues on next page

Read affordance	Set as a *Form* instance specified with the `readproperty op` field.
Observation affordance	Set as a *Form* instance specified with the `observeproperty op` field.

Table 6.5: Mapping of the Abstract conceptual model Action to the WoT Thing Description

Element	**Description**
Action type	Set via the `@type` field at the Action level. This allows to specify the action type using the domain ontology.
Is Augmented	Custom field that uses the `augmentedInteraction` Data property defined in the WoDT vocabulary as a field name. The value of the field is `true` in case the action is an augmented one, `false` otherwise.
Required input	Set with the `input` field of the Thing Description, using the WoT `DataSchema`.
Action ID	Set as the name of the action in the Thing Description Action array.
Action invocation affordance	Set as a *Form* instance specified with the `invokeaction op` field.

The *Data Schema* and the *Form* specified in the *Abstract conceptual model* are mapped respectively to the `DataSchema` and the `Form` concept in the Thing Description model.

All the Data and Object properties identified above have been modeled in the WoDT vocabulary. In addition, the WoDT vocabulary contains the remaining properties defined in the Specifications, so the predicates needed for the *Digital Twin Knowledge Graph* (`availableActionId`) and the additional relation types to be used in the HTTP Link Headers.

In conclusion, it is possible to note that the WoT Thing Description, appropriately supported by the WoDT vocabulary, can implement a Digital Twin Descriptor offering at the same time a compatibility layer towards the Web of Things. A WoDT Digital Twin described by a Digital Twin Descriptor implemented with a Thing Description can also be used by a Consumer as a WoT Thing.

6.2.2 Azure Digital Twins based WoDT Digital Twin

The *WoDT Digital Twin* based on the *Azure* stack, in particular on *Azure Digital Twins*, was implemented simulating an *ad-hoc* approach that allows a Digital Twin Developer to expose a Digital Twin, defined in its Azure Digital Twins instance, as a WoDT Digital Twin allowing its integration in WoDT ecosystems. Therefore, it is related to a specific Digital Twin, particularly the Ambulance Digital Twin that will be described in the example use case. However, the proposed implementation is easily generalizable to become a library and for this reason, it is described without any reference to the use case.

Regarding the Azure services, one main issue with Azure Digital Twins is the impossibility of modeling relationships with Digital Twins that live outside the Azure Digital Twins instance itself. The targets in Azure Digital Twins relationships must be valid IDs of Digital Twins that are managed under the same instance. The strategy followed in the prototype was to create a new Digital Twin inside the instance for each target WoDT Digital Twin with which the original one has a relationship in the ecosystem. Hence, the Digital Twin created is an internal representation of the external WoDT Digital Twin that can be linked to the original one. Moreover, the URI of the interested WoDT Digital Twin is set as a property due to the impossibility of setting a URI as the ID of a Digital Twin inside an Azure Digital Twins instance.

As described in the prototype design, the created *Azure Digital Twins* instance sends all the events to the specified *Azure Event Grid topic* that redirects them to the *Azure Function* described before. The *Azure Function* was developed in *C#* and has the objective of adapting the *patch events* (specified in the *Cloud Events* specification, Listing 6.1), sent by Azure Digital Twins, to complete and consistent Digital Twin snapshots event to be sent to the *WoDT Shadowing Adapter*.

Listing 6.1: Example of a patch event from Azure Digital Twins

```json
{
    "specversion": "1.0",
    "id": "39d4abb9-e3ee-4ed5-ad17-2243a9784946",
    "type": "Microsoft.DigitalTwins.Twin.Update",
    "source": ...,
    "data": {
        "modelId": "dtmi:io:github:webbasedwodt:Ambulance;1",
        "patch": [
            {
                "value": 33,
                "path": "/fuelLevel",
                "op": "replace"
            }
        ]
    },
    "subject": "ambulance",
    "time": "2024-02-14T15:12:14.5044983+00:00",
    "datacontenttype": "application/json",
    "traceparent": ...
}
```

The Azure Function maps the patch event to a complete snapshot of the interested Digital Twin, retrieving data from the Azure Digital Twin instance and mapping the data to an internal JSON representation that the WoDT Shadowing Adapter can easily parse. Here, no semantics is involved because data is only exchanged between internal components, so not publicly exposed. An example, derived from the event in Listing 6.1, is shown in Listing 6.2.

Listing 6.2: Example of a mapped event by the Azure Function

```
{
  "dtId": "ambulance",
  "eventType": "UPDATE",
  "eventDateTime": "2024-02-14T15:12:14.5044983+00:00",
  "properties": {
    "fuelLevel": 33,
    "busy": true
  },
  "relationships": [
    {
      "$sourceId": "ambulance",
      "$relationshipName": "rel_is_approaching",
      "$targetId": "http://localhost:3001/"
    }
  ]
}
```

The event, shown in Listing 6.2, is sent to the *WoDT Shadowing Adapter* via the *Azure SignalR* service.

The remaining components of the architecture were developed as a *Kotlin* service on top of the Azure pipeline, as described in the prototype design. The service was created following the principles of the *Clean Architecture* [37] which allowed the correct separation of domain modeling, use cases, application logic, and everything related to technologies and infrastructure. The use of this type of architecture has made it possible to obtain excellent testability, extensibility, and maintainability. Moreover, to better handle the flow of data between the various components and to manage asynchronous computations, *Kotlin Coroutines* are used allowing an event-driven communication orchestrated by an engine, called `WoDTEngine`. In the following, an overview of the implementation of each component is provided.

Domain ontology mapping The DTKG and the DTD contain domain-related data, so it is essential to be able to convert data from the Azure DTDL model and from update events to the domain ontology that the Digital Twin is supposed to offer to Consumers. For this reason, a concept of *ontology* (a Kotlin interface), independent of any specific technology, was developed inside the service allowing Digital Twin Developers to easily specify the mappings via a new class. To allow that, an internal model of a Knowledge Graph was created to support the process.

WoDT Shadowing Adapter This component connects to the Azure SignalR service subscribing to the update events from the Azure Digital Twins instance – enhanced by the Azure Function. Internally, it uses the *Microsoft SignalR client*, specifically its asynchronous Consumer. The received events are converted to a Knowledge Graph, that is modeled using the previously defined ontology mapper, and exposed to the other components via a `SharedFlow`. The read-only and public `Flow` allows the component to expose the processed updates as events that can be easily consumed.

DTD Manager This component automatically connects to the *Azure Digital Twins* instance using the *Azure Digital Twins SDK*, retrieves the DTDL model of the Digital Twin, and starting from it obtains the Digital Twin Descriptor implemented via the WoT Thing Description to be provided to Consumers. To support the creation of the Thing Description, the *Sane City WoT Servient* library was used.

DTKG Engine It manages the current Digital Twin Knowledge Graph using the *Apache Jena* library and exposes the flow of DTKGs through a Kotlin `Flow` to be easily consumed. The Digital Twin Knowledge Graph is stored only in memory for simplicity.

WoDT Digital Twin Interface It provides the Interaction patterns of the WoDT Digital Twin, following the *WoDT Digital Twin Specification*. The *Ktor* framework was used to implement the HTTP Web Server and the REST APIs. The observation affordance was provided via the *WebSocket* protocol and implemented with *Ktor*.

Platform Management Interface It manages the registrations to the WoDT Digital Twins Platforms. It offers the endpoint used by the Platforms to signal its registration as an HTTP REST API using the *Ktor server* framework, and it manages the communications with the WoDT Digital Twins Platforms using the HTTP client offered by the *Ktor client* framework.

WoDT Engine An internal engine was developed to orchestrate the various components of the architecture. Thanks to *Kotlin Coroutines* its development was very straightforward and allowed to easily set up the launcher of the service that has the only responsibility to inject the dependencies and start the engine.

6.2.3 White Label Digital Twins based WoDT Adapter

The proposed design of the WoDT Digital Twin based on the *WLDT Framework* eases the development of a library (*wldt-wodt-adapter*) that sitting on top of existing software enables the creation of WoDT-compliant Digital Twins. The library was developed as an extension of the *WLDT Framework* providing a way to define the domain ontology and, based on that, enabling the creation of WoDT Digital Twins through the usage of a custom Digital Adapter – the `WoDT Digital Adapter`. Moreover, the library is published on the *GitHub Packages Registry* to be easily imported by Digital Twin Developers.

Being based on the WLDT Framework, it was developed in *Java*. In the following, an overview of the implementation of each component is provided.

Domain ontology mapping The domain ontology mapping follows the same strategy chosen for the Azure-based WoDT Digital Twin. The library offers Digital Twin Developers an interface – `DTOntology` – to implement to be able to map internal data from the defined WLDT model to the domain ontology that the WoDT Digital Twin has to offer to Consumers. It is completely independent of any specific technology, so it is written in plain *Java*.

DTKG Engine It manages the current Digital Twin Knowledge Graph using the *Apache Jena* library. It provides all the methods to update the current DTKG based on the callback methods called on the `WoDT Digital Adapter` and to observe the evolution of the DTKG (based on the *Observer* pattern). The Digital Twin Knowledge Graph is stored only in memory for simplicity.

DTD Manager This component automatically creates the Digital Twin Descriptor, implemented with the WoT Thing Description, based on the callback methods called on the `WoDT Digital Adapter`. To support the creation of the Thing Description, the *Sane City WoT Servient* library was used.

WoDT Digital Twin Interface It provides the Interaction patterns of the WoDT Digital Twin, following the *WoDT Digital Twin Specification*. The HTTP Web Server and the WebSocket protocol were implemented using the *Javalin* framework.

Platform Management Interface It simply manages the registrations to the WoDT Digital Twins Platforms. The Digital Twin, when starts, can automatically register to the interested WoDT Digital Twins Platforms. The communications with the Platforms are handled by the *Java HTTP client*.

The resulting `WoDT Digital Adapter` exploits the callback methods of the `Digital Adapter` library class to orchestrate the components, offering the required features for a WoDT Digital Twin. A Digital Twin Developer that wants to extend its WLDT Digital Twin to be WoDT-compliant needs only to add the *wldt-wodt-adapter* library to its dependencies and use the `WoDT Digital Adapter` passing a valid implementation of the domain ontology mapper. In Listing 6.3 a simple example of how a Digital Twin Developer can set up a WoDT-compliant Digital Twin using the WLDT Framework and the *wldt-wodt-adapter* library.

Listing 6.3: Usage of the WoDT Digital Adapter to create WoDT-compliant Digital Twins

```
// Create the WLDT Engine
final WldtEngine digitalTwinEngine = new WldtEngine(
    new MirrorShadowingFunction(),
    "example-dt"
);
// Add the Physical Adapter
digitalTwinEngine.addPhysicalAdapter(new ExamplePhysicalAdapter());
// Add the WoDT Digital Adapter -- here the contribution
digitalTwinEngine.addDigitalAdapter(new WoDTDigitalAdapter(
    "wodt-dt-adapter",
    new WoDTDigitalAdapterConfiguration(
        ...
        new ExampleOntology(),
        ...
    )
));
// Start the WLDT Engine
digitalTwinEngine.startLifeCycle();
```

6.2.4 WoDT Digital Twins Platform

The *WoDT Digital Twins Platform* was developed from scratch, without any base library or platform, in *Kotlin*. The Platform was created following the principles of the *Clean Architecture* [37] which allowed, as stated before, the correct separation of domain modeling, use cases, application logic, and everything related to technologies and infrastructure, in addition to the obtainment of excellent testability, extensibility, and maintainability. The asynchronous computations and the event-driven core of the Platform are managed through the use of *Kotlin Coroutines*.

For this thesis, the developed Platform supports the Digital Twin Descriptor only implemented with the WoT Thing Description, and for the observation of WoDT Digital Twins only supports the WebSocket protocol. The interactions with the WoDT Digital Twins regarding registration and lifecycle activities are HTTP-based.

In the following, an overview of the implementation of each component is provided.

Ecosystem Management Interface This component was developed in two parts. The first part is the controller of the HTTP REST APIs used for managing the WoDT Digital Twins registration and lifecycle. It is implemented using the *Ktor server* framework. The second part includes the logic for checking the validity and support of the provided Digital Twin Descriptors. It also signals the registration of added WoDT Digital Twins (using the *Ktor client* framework) and exposes the flow of WoDT Digital Twins that register or delete through a Kotlin `Flow` for easy consumption.

WoDT Digital Twins Observer As soon as a new WoDT Digital Twin is registered to the Platform, this component starts observing it using the WebSocket protocol (with the *Ktor client* framework). The received Digital Twin Knowledge Graphs are exposed to the interested components through a Kotlin `Flow` to be easily consumed. Moreover, when a WoDT Digital Twin is deleted, it closes the associated socket accordingly.

Ecosystem Registry This component manages the registry of all the registered WoDT Digital Twins and the URI/URL mappings, as described by the WoDT Digital Twins Platform Specification. It is a plain *Kotlin* service without infrastructure dependencies. It is used by several components of the Platform for different purposes so, to avoid wrong usage of the component, different interfaces for the different clients were created, following the *ISP (Interface Segregation Principle)*. It is not the only component where this principle is applied, but it is one of the most prominent ones.

WoDT Digital Twins Platform Knowledge Graph Engine It implements all the services on the *WoDT Digital Twins Platform Knowledge Graph* that other components of the Platform need, compatibly with the WoDT Digital Twins Platform Specification. In this implementation both the Digital Twin Knowledge Graphs and the Digital Twin Descriptors are stored (in memory), experimenting with the possibility of more powerful queries, as described before. The flow of the updated version of the Platform Knowledge Graph is exposed through a Kotlin *Flow* to be easily consumed. The Knowledge Graph is managed using the *Apache Jena* library. This library is particularly useful for managing the query resolution task. In particular, this component performs queries on the Platform Knowledge Graph using an OWL Reasoner for the inference and following the *SPARQL Protocol* thanks to the support of *Apache Jena*. Specifically, it handles all the query types such as *SELECT*, *ASK*, *CONSTRUCT*, and so on, and all the response content types such as *CSV* (Comma-separated values), *TSV* (Tab-separated values), *JSON*, and so on. Even in this component is applied the *ISP* principle, separating the writing and the reading responsibilities.

WoDT Digital Twins Platform Interface It provides the Interaction patterns of the WoDT Digital Twins Platform, following the *WoDT Digital Twins Platform Specification*. The HTTP Web Server and the WebSocket protocol (the only protocol supported for observation) were implemented using the *Ktor* framework.

WoDT Platform Engine The various components of the Platform and the events they generate are orchestrated by an internal engine, called `WoDT Platform Engine`. Its implementation, based on *Kotlin Coroutines*, is shown in Listing 6.4.

Listing 6.4: Implementation of the WoDT Platform Engine

```kotlin
suspend fun start() = coroutineScope {
    launch {
        ecosystemManagementInterface.ecosystemEvents.collect {
            if (it is NewDigitalTwinRegistered) {
                launch {
                    woDTDigitalTwinsObserver.observeDigitalTwin(it.dtd)
                }
                platformKnowledgeGraphEngine.mergeDigitalTwinDescriptor(it.dtd)
            } else if (it is DigitalTwinDeleted) {
                woDTDigitalTwinsObserver.stopObservationOfDigitalTwin(it.dtURI)
                platformKnowledgeGraphEngine.deleteDigitalTwin(it.dtURI)
            }
        }
    }
    launch {
        woDTDigitalTwinsObserver.dtkgRawEvents.collect {
            platformKnowledgeGraphEngine.mergeDigitalTwinKnowledgeGraphUpdate(
                it.first,
                it.second
            )
        }
    }
    platformWebServer.start()
}
```

As it is possible to note, the HTTP client and server, implemented using the *Ktor* framework, are used by different components. To better organize the code, free the components from infrastructure dependencies, and to follow the *Clean Architecture* principles, the classes `WoDTPlatformHttpClient` and `WoDTPlatformHttpServer` have been created to contain all the related logic and have been properly injected to the interested components. In this way, the actual components of the Platform remain at a higher level of abstraction and are independent of the specific protocols or implementation of the infrastructural concepts increasing their testability and maintainability.

6.2.5 DevOps technologies and practices

During the development of the prototype, modern *DevOps* technologies and practices were used. Their usage promotes small, incremental changes that are integrated continuously, reducing the probability of failure. The several *DevOps* tools available automatize as much as possible, leaving the use of human resources only for the work that matters, not for repetitive tasks, improving the quality and the organization of the code. The automation is possible because the *DevOps* culture promotes highly tested code to be able to perform automatic operations, such as automatic dependency updates, safely.

In the following, the overview of the used *DevOps* practices and technologies is presented.

Workflow Organization

The prototype software artifacts, whose development is managed using *Git*, are organized in a *GitHub Organization*, leveraging the *GitHub* hosting services.

The repositories that contain *Kotlin* or *Java* projects have been initialized using respectively *Kotlin* and *Java GitHub template repositories*, to speed up the set-up of all the used tools.

For an efficient usage of the *Git branches*, the *Git-Flow branching model* was used. In particular, the exploited types of branches are:

- *main*: main branch that contains the code associated with releases.
- *develop*: development branch, contains the pre-production code integrating the developed features.
- *feature*: support branch, used for the development of a specific feature, which once completed will be integrated into the develop branch.

Moreover, to simplify the usage of the automatic versioning tools and to standardize the commit messages, the *Conventional Commits* specification[1] was used. The main types of commit used are: *build*, *chore*, *ci*, *docs*, *feat*, *fix*, *refactor*, *style*, and *test*.

Compliance with the specification and the code quality standards are ensured before each commit using ad-hoc *git hooks*.

Build Automation

Gradle was used to automate the source code compilation operations, dependency management, and the configuration of the Kotlin and Java-based development

[1] `https://www.conventionalcommits.org/en/v1.0.0/`

environment. *Gradle* is a powerful and flexible tool for the build automation and management of dependencies with a declarative syntax.

The DRY principle applies also to dependency management. For this reason, to manage dependency in a more organized way, enabling automatic dependency updates, the *TOML Gradle Version Catalog* was used.

Testing

To verify correct functioning, thanks to the Clean Architecture principles adopted, it was possible to test each layer independently through the use of Unit-Tests. The following frameworks were used for testing:

- *Kotest*[2]: it is a Kotlin testing framework. In particular, all the tests were written using the *StringSpec* style, providing a declarative syntax.
- *Junit5*[3]: it is a Java testing framework.

In addition, to test the respect of the *Clean Architecture* principles, the *ArchUnit*[4] library was used to make tests over the system's architecture.

Quality Assurance

A good practice of *Build Automation* and DevOps in general is the usage of code quality assurance tools. The projects developed in *Kotlin* and *Java* were configured with tools for the *Static Code Analysis*, and *Style Checking* that automatically perform code quality checks during the build process.

Continuous Integration

One of the most important DevOps practices is the *Continuous Integration.* The *Continuous Integration* practice has the objective of continuously integrating the code with the main development line to promptly identify integration problems and improve the quality of the software, allowing a faster and more reliable development process.

For each repository, a specific *Continuous Integration workflow* was designed and developed using *GitHub Actions*[5]. The designed workflows are used to perform the following tasks: build, code quality assurance, tests, release and delivery, and documentation publication.

[2]`https://kotest.io/`
[3]`https://junit.org/junit5/`
[4]`https://www.archunit.org/`
[5]`https://github.com/features/actions`

Finally, to automate the software dependency updates, the *Renovate*[6] and *Mergify*[7] bots have been used.

Continuous Delivery

The designed workflows for the Continuous Integration also support the *Continuous Delivery* of the software artifacts.

In particular, for the *Azure Digital Twins based WoDT Digital Twin* and for the *WoDT Digital Twins Platform* the respective *Docker* images are built, from their `Dockerfile`, and published on the *GitHub Container Registry*. The published images' versions are automatically aligned with the releases on *GitHub Release*.

In addition, also the *wldt-wodt-adapter* library (for the WLDT-based WoDT Digital Twins) is of interest for Continuous Delivery. The library is automatically versioned and released on *GitHub Release* and delivered on *GitHub Packages* to be easily used just by adding it as a dependency.

Versioning

The process that assigns a unique identifier to a particular software state, allowing one to distinguish between the different states of a software product and to be able to refer to it, is called *Software Versioning*.

The prototype (each software) is versioned according to the *Semantic Versioning* specification. According to it, the software version is composed of three numbers: *Major*, *Minor*, and *Patch*. Each change to the source code causes one of these numbers to increase based on the importance of the changes made. In particular, in the prototype, following the usage of the *Conventional Commits* specification the version is automatically computed by analyzing the commits semantics. The automatically computed version is used in the *Continuous Integration* and *Delivery workflows* to automatically publish releases and software artifacts.

6.3 Example use case

In this section, the example use case used to demonstrate the effectiveness of the proposed design and the realized prototype is described. Firstly, the scenario that was implemented as a use case is defined and thereafter, the obtained results are presented.

[6]`https://www.mend.io/renovate/`
[7]`https://mergify.com/`

6.3.1 Use case description

A cross-domain and cross-organizational perspective must be taken to prove the vision's main characteristics. Hence, the objective is the creation of a scenario of interest for different organizations involved in different domains, showing multi-organization and multi-domain support. In the proposed use case, the Healthcare and the Smart City domains live together and show an important way of using information from a shared ecosystem of Digital Twins.

The scenario takes inspiration from the *Major Trauma Management* defined in [44] taking into account only the effects of the first stage, the *Emergency Call Management*. In particular, the focus is on the ambulance mission to reach the patient and the management of the traffic lights on the ambulance route. Hence, two organizations are involved: the *Healthcare organization*, which handles the ambulance emergency missions, and the *Smart City organization*, which manages the city. Each one has its own WoDT Digital Twins Platform, where they manage their WoDT Digital Twins or add others from the ecosystem, and an application layer, where the business logic and automated agents get executed. In particular, the *Smart City organization* adds all the ambulance's Digital Twins of the city, including them as part of its WoDT Digital Twins Platform Knowledge Graph. Hence, the *Smart City application layer* can observe the Web of Digital Twins ecosystem and understand that an ambulance (whose WoDT Digital Twin is added to the *Smart City organization*) needs to reach a patient or the hospital. Therefore, it can run an automated agent that, reasoning on the current state of the world (accessed through the contextualized replica offered by the WoDT Digital Twins and queried through the WoDT Digital Twins Platform of its organization), can manage the traffic lights to create a *"green route"* for the ambulances, from the start to the endpoint. In addition to that, the *Smart City application layer* could query the ecosystem to understand which vehicles are interested and warn them about the next approaching ambulance. This can be useful for both autonomous and non-autonomous cars. The former can automatically decide the best action to take and so react promptly to the warning, and the latter can warn the driver about the event. In conclusion, this simple scenario can help to reduce road accidents, reduce intervention time, and increase road safety.

This scenario, shown in Figure 6.5, allows us to reason over the majority of the requirements analyzed during this thesis.

In Figure 6.5 it is possible to note the establishment of the typical *WoDT Layered View* (described in the section 2.2) composed of three layers: the *Physical Asset layer*, the *Digital Twin layer*, and the *Application layer*. As already stated, the *Application layer* accesses the *real world* through the *Digital Twin layer*. The *Digital Twin layer*, in this scenario, even if it is composed of the two organizations, contains *WoDT Digital Twins* that are completely independent of the organiza-

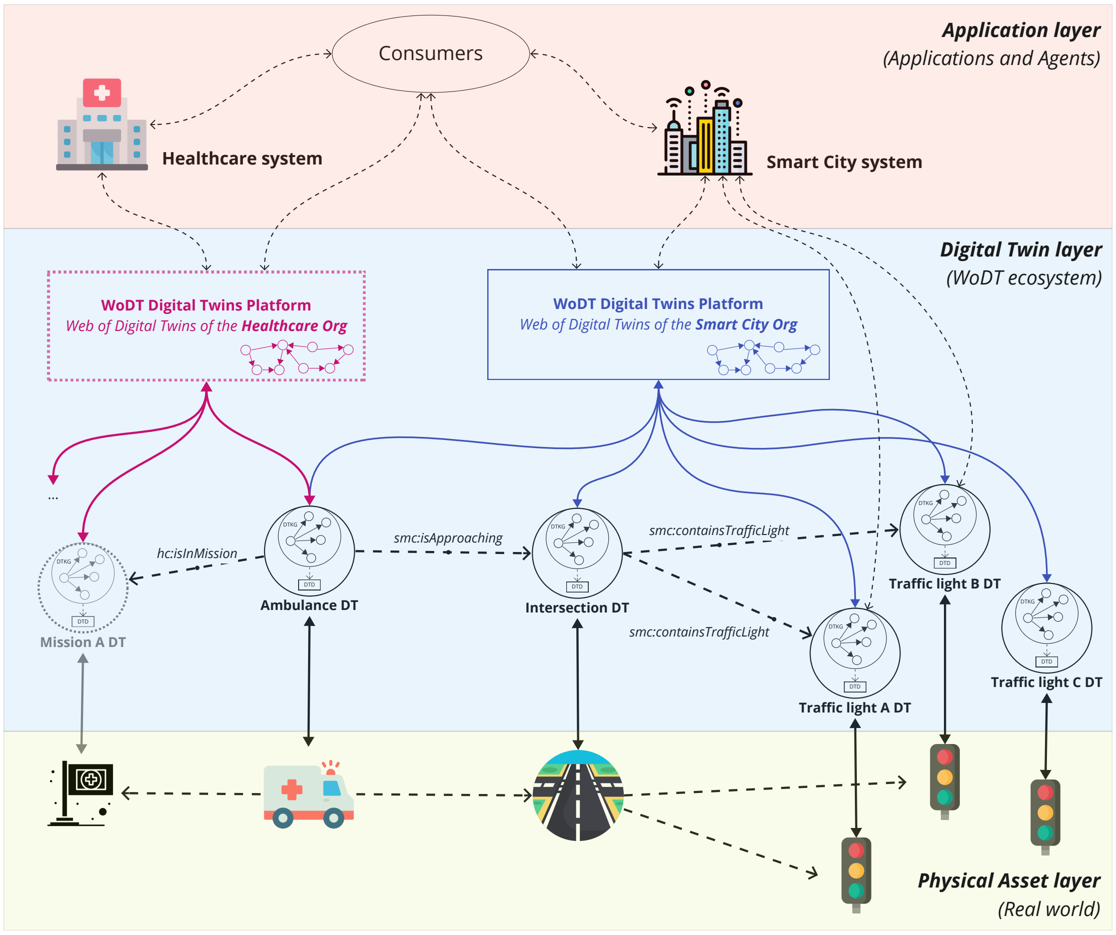

Figure 6.5: Scheme of the proposed use case scenario

tions to which they are registered. For example, the *Ambulance WoDT Digital Twin* is registered to both Platforms. This highlights the respect of the *Digital Twin as a Service* constraint.

Furthermore, it is necessary to prove the possibility of creating ecosystems of heterogeneous Digital Twins where existent technologies are used and a compatibility layer towards existent popular paradigms is provided. For this reason, the prototype, as described previously, is based on two different technologies for the development of WoDT Digital Twins, *Azure Digital Twins* and the *White Label Digital Twins* framework and a compatibility layer towards *Web of Things* is offered (implementing the Digital Twin Descriptors with the *WoT Thing Description*).

The Ambulance Digital Twin is based on the *ad-hoc Azure Digital Twins* based prototype, as described before. Being, an Azure-based Digital Twin, its model is described by a *DTDL* file. The model is composed of two Properties and two Relationships. The Properties are:

- *busy*: it states if the ambulance is busy or not.
- *fuel level*: it indicates the percentage of fuel available.

The Relationships are:

- *part of a mission*: it connects the Ambulance Digital Twin to the Mission Digital Twin (not modeled in this scenario) in which the ambulance is involved.
- *is approaching*: it mirrors the fact that the ambulance is approaching an intersection.

Its model is shown in Listing 6.5.

Listing 6.5: DTDL model of the Ambulance Digital Twin

```json
{
 "@id": "dtmi:io:github:webbasedwodt:Ambulance;1",
 "@type": "Interface",
 "@context": "dtmi:dtdl:context;2",
 "displayName": "Ambulance",
 "contents": [
  {
   "@type": "Property",
   "name": "busy",
   "schema": "boolean"
  },
  {
   "@type": "Property",
   "name": "fuelLevel",
   "schema": "double"
  },
  {
   "@type": "Relationship",
   "@id": "dtmi:io:github:webbasedwodt:Ambulance:rel_is_part_of_mission;1",
   "name": "rel_is_part_of_mission",
   "displayName": "The Ambulance is part of the mission",
   "target": "dtmi:io:github:webwodt:ExternalDT;1"
  },
  {
   "@type": "Relationship",
   "@id": "dtmi:io:github:webbasedwodt:Ambulance:rel_is_approaching;1",
   "name": "rel_is_approaching",
   "displayName": "The Ambulance is approaching the intersection",
   "target": "dtmi:io:github:webwodt:ExternalDT;1"
  }
 ]
}
```

The Intersection and the Traffic lights Digital Twins are based on the *WLDT Framework* and made *WoDT-compliant* with the developed library (*wldt-wodt-adapter*). In both the Digital Twin types, the *Physical Adapter* is mocked, and the *Shadowing Model Function* exactly mirrors the data without modifications. The Intersection's Digital Twin model is composed of only one relationship, *contains traffic light*, that mirrors the relationship between the Intersection and the Traffic lights of which is composed. Instead, the Traffic lights' Digital Twin models are composed of one Property and one Action. The Property, *is on*, is a simplified representation of the Traffic light state by considering only the on and off state, which can be switched by the *switch* Action. The Traffic lights *A* and *B* are part of the previously described Intersection, contrary to C. The Traffic light *C* is added to verify the correctness of the queries performed subsequently, introducing data that is not useful.

Each organization creates its personalized view of reality, to observe and contextualize, also in terms of which WoDT Digital Twin to include in its WoDT Digital Twins Platform Knowledge Graph. The *Intersection*, the *Traffic light A*, the *Traffic light B*, and the *Traffic light C* WoDT Digital Twins register themselves to the WoDT Digital Twins Platform of the *Smart City organization*, while the *Ambulance* Digital Twin, apart from registering itself to the WoDT Digital Twins Platform of the *Healthcare organization*, is added by the Platform administrator to the WoDT Digital Twins Platform of the *Smart City organization*, to be able to observe it.

According to the prototype implementations, a model mapping to the domain ontology is specified for all the types of WoDT Digital Twins. In addition, considering that the *Ambulance WoDT Digital Twin*, the WoDT Digital Twins Platform, and each WoDT Digital Twin created with the *wldt-wodt-adapter* library have a Continuous Delivery pipeline that automatically builds and releases the associated *Docker* image, the deployment of the use case has been eased using *Docker compose*.

6.3.2 Results

After the WoDT Digital Twins have all been registered to the WoDT Digital Twins Platforms as described before, the *Smart City application layer* starts observing the *WoDT Digital Twins Platform Knowledge Graph* of its organization using the dedicated Platform's *WebSocket* endpoint and handles every ambulance that wants to approach any intersection. As soon as an ambulance approaches an intersection, it will be mirrored at the *Digital Twin layer* by creating a relationship between the *Ambulance WoDT Digital Twin* and the *Intersection WoDT Digital Twin*. The *Smart City application layer*, observing the WoDT ecosystem, can understand the new current state of the world and it:

- performs a SPARQL Query over its *WoDT Digital Twins Platform Knowledge Graph* to understand which traffic lights are interested and their available actions. For simplicity, in the use case, the query is performed only on the domain data, but with the prototypical implementation provided, it could be possible to obtain directly also the DTD's data to invoke the actions without executing a separate request.
- obtains the Digital Twin Descriptors of the involved traffic lights to understand how to invoke actions, using the returned metadata.
- performs the actions on the involved traffic lights, creating a *"green route"* for the interested ambulance.

Firstly, the *Smart City application layer* performs the SPARQL Query shown in Listing 6.6.

Listing 6.6: SPARQL Query to obtain the traffic lights of the interested intersection

```
PREFIX smc: <https://smartcityontology.com/ontology#>
PREFIX wodt: <https://purl.org/wodt/>

SELECT ?trafficLight ?availableAction
WHERE {
    <http://localhost:4000/wodt/http://localhost:3000/> smc:isApproaching
        ?intersection .
    ?intersection smc:containsTrafficLight ?trafficLight .
    ?trafficLight wodt:availableActionId ?availableAction .
}
```

The SPARQL Query contains only domain-related elements, allowing also Consumers who are unaware of the Digital Twin Descriptor semantics to perform queries over the WoDT ecosystem. In particular, this query can navigate the graph and return the traffic lights of the interested intersection. The obtained result, considering `text/csv` as accepted *mime-type*, is shown in Listing 6.7

Listing 6.7: SPARQL Query result of the Listing 6.6

```
trafficLight,availableAction
http://localhost:4000/wodt/http://localhost:3003/,switch
http://localhost:4000/wodt/http://localhost:3002/,switch
```

Having the local URLs of the interested *Traffic light WoDT Digital Twins*, the *Smart City application layer* can, apart from obtaining the DTD data directly from the Platform as said before, extract the original WoDT Digital Twin URIs and obtain the Digital Twin Descriptors. For example, the *Digital Twin Descriptor* of the *Traffic light A* is shown in Listing 6.8 (*is-on* property affordances are hidden to favor readability).

The *Smart City application layer* analyzes and navigates the Digital Twin Descriptors, implemented with the WoT Thing Description supported by the *WoDT*

Listing 6.8: Digital Twin Descriptor of the Traffic light A

```
{
    "id": "http://localhost:3002/",
    "properties": {
        "snapshot": {
            "forms": [
                {
                    "href": "ws://localhost:3002/dtkg",
                    "op": [
                        "observeproperty"
                    ],
                    "subprotocol": "websocket"
                }
            ],
            "type": "string",
            "observable": true,
            "readOnly": true
        },
        "is-on": {
            "observable": true,
            "readOnly": true,
            "@type": "https://www.w3.org/2001/XMLSchema#boolean",
            "https://purl.org/wodt/domainPredicate": "https://lampontology.com/
                ontology#isOn",
            "https://purl.org/wodt/augmentedInteraction": false
        }
    },
    "actions": {
        "switch": {
            "@type": "https://lampontology.com/ontology#SwitchCommand",
            "https://purl.org/wodt/augmentedInteraction": false,
            "forms": [
                {
                    "href": "http://localhost:3002/action/switch",
                    "op": [
                        "invokeaction"
                    ]
                }
            ]
        }
    },
    "@type": "https://smartcityontology.com/ontology#TrafficLight",
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "https://purl.org/wodt/version": "1.0.0",
    "links": [
        {
            "href": "http://localhost:4000/",
            "rel": "https://purl.org/wodt/registeredToPlatform"
        }
    ],
    "https://purl.org/wodt/physicalAssetId": "trafficLightA"
}
```

vocabulary, searching for action of type `SwitchCommand` and checking, using the SPARQL Query result (Listing 6.7), if the action can be executed in the current state of the Digital Twin. Once, it finds the required actions, it gets all the metadata to request their invocation. Finally, it applies its logic to understand which traffic lights need to be turned on or off depending on the ambulance route and invoke the respective actions.

This simple use case demonstrates that Consumers can reason and act over reality by accessing the *Web of Digital Twins* without caring about technologies and the actual interaction with the Physical Assets. Each *WoDT Digital Twin* can be used by organizations to build their view over reality and solve powerful queries for the application layer. As stated, this use case only proves the main features of the design, mainly the possibility of creating an ecosystem of heterogeneous Digital Twins. All the other designed and implemented features have been tested successfully separately (for example, the *Multi-model directory service*).

Furthermore, it is interesting to analyze the state of the *WoDT Digital Twins Platform Knowledge Graph* of the *Smart City organization* when the ambulance approaches the intersection (so when the SPARQL Query in the Listing 6.6 is executed). A partial view, where only the data about the *Ambulance*, the *Intersection*, and the *Traffic light A WoDT Digital Twins* is reported, is shown in Listing 6.9 and in Figure 6.6.

From the Listing 6.9, it is possible to analyze the personalized and contextualized view over reality created by the *Smart City organization*, where relationships between *WoDT Digital Twins* mirrors the dynamicity of the real world. In addition:

- *DTKGs* and *DTDs* data are mixed, enabling the possibility of powerful queries by Consumers.
- each *WoDT Digital Twin URI* has been mapped to its local URL.
- it is a Knowledge Graph that can be easily navigated and queried.

Finally, even if it is possible to appreciate the *Digital Twin Knowledge Graphs* directly in the Listing 6.9, the *DTKGs* of the *Ambulance* and of *Traffic light A WoDT Digital Twins* are shown respectively in Listing 6.10 and Listing 6.11.

Each *WoDT Digital Twin* exposes the same uniform interface, and so Consumers see them as Web Resources that they can use and navigate compatibly with the *Linked Data principles.*

Listing 6.9: Partial WoDT Digital Twins Platform Knowledge Graph of the Smart City organization

```
@prefix hc: <https://healthcareontology.com/ontology#> .
@prefix smc: <https://smartcityontology.com/ontology#> .
@prefix lp: <https://lampontology.com/ontology#> .
@prefix td: <https://www.w3.org/2019/wot/td#> .
@prefix jsonschema: <https://www.w3.org/2019/wot/json-schema#> .
@prefix hctl: <https://www.w3.org/2019/wot/hypermedia#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix wodt: <https://purl.org/wodt/> .

<http://localhost:4000/wodt/http://localhost:3000/>
        a hc:Ambulance;
        hc:hasFuelLevel "87.0"^^xsd:double;
        hc:isBusy true;
        wodt:physicalAssetId "AM3030T";
        wodt:version "1.0.0";
        smc:isApproaching <http://localhost:4000/wodt/http://localhost:3001/>;
        td:hasPropertyAffordance [
            a smc:Intersection;
            wodt:domainPredicate smc:isApproaching;
            jsonschema:readOnly true;
            td:isObservable true;
            td:name "rel_is_approaching"
        ];
        ...
        td:hasPropertyAffordance [ a
            jsonschema:StringSchema;
            jsonschema:readOnly true;
            td:hasForm [
                hctl:forSubProtocol "websocket";
                hctl:hasOperationType td:observeProperty;
                hctl:hasTarget "ws://localhost:3000/dtkg"^^xsd:anyURI
            ];
            td:isObservable true;
            td:name "snapshot"
        ];
        ...

<http://localhost:4000/wodt/http://localhost:3001/>
        a smc:Intersection;
        wodt:physicalAssetId "intersectionPA";
        wodt:version "1.0.0";
        smc:containsTrafficLight <http://localhost:4000/wodt/http://localhost
            :3003/> , <http://localhost:4000/wodt/http://localhost:3002/>;
        td:hasPropertyAffordance [
            ...
        ];
        ...

<http://localhost:4000/wodt/http://localhost:3002/>
        a smc:TrafficLight;
        lp:isOn true;
        wodt:availableActionId "switch";
        wodt:physicalAssetId "trafficLightA";
        wodt:version "1.0.0";
        td:hasActionAffordance [
            a lp:SwitchCommand;
            td:hasForm [
                hctl:hasOperationType td:invokeAction;
                hctl:hasTarget "http://localhost:3002/action/switch"^^xsd:anyURI
            ];
            td:name "switch"
        ];
        ...
```

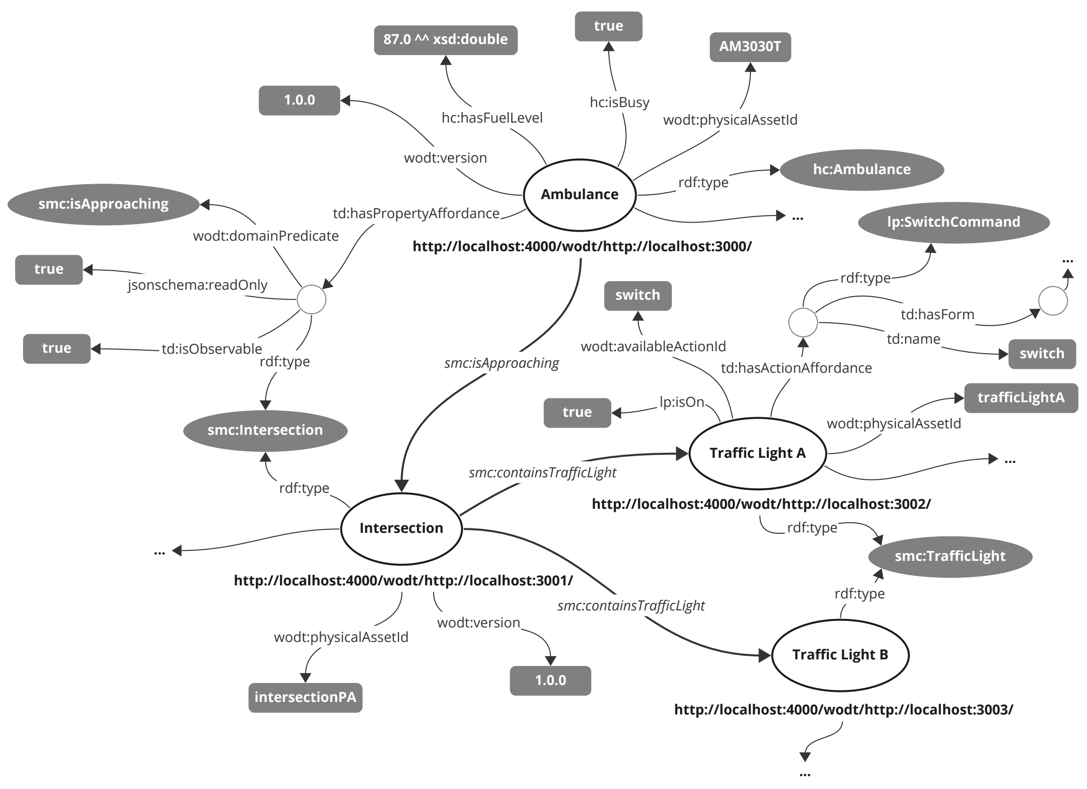

Figure 6.6: Partial visualization of the WoDT Digital Twins Platform Knowledge Graph of the Smart City organization

Listing 6.10: DTKG of the Ambulance WoDT Digital Twin

```
<http://localhost:3000/>
        <https://healthcareontology.com/ontology#hasFuelLevel>
                "87.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
        <https://healthcareontology.com/ontology#isBusy>
                true ;
        <https://smartcityontology.com/ontology#isApproaching>
                <http://localhost:3001/> .
```

Listing 6.11: DTKG of the Traffic Light A WoDT Digital Twin

```
<http://localhost:3002/>
        <https://lampontology.com/ontology#isOn>
                true;
        <https://purl.org/wodt/availableActionId>
                "switch" .
```

Chapter 7

Conclusions

Nowadays, we are assisting in the spreading of closed system proposals for the virtualization of individual Physical Assets used for vertical applications. While it is useful in some domains, the Digital Twin paradigm can be further exploited for the digitalization of entire portions of reality.

The vision followed in this thesis, the Web of Digital Twins, aims to the creation of an interoperable service layer where Digital Twins are created and have relationships across multiple domains and multiple organizations.

The thesis objective was to propose an implementation of the *Web of Digital Twins* vision to obtain the possibility for interoperability and openness between Digital Twins of different domains, organizations that use existent technologies, and create an open, discoverable, and navigable ecosystem of Digital Twins that can serve as a service layer for applications on top.

This thesis proposed an implementation of the Web of Digital Twins vision for ecosystems of heterogeneous Digital Twins, by integrating key concepts and methods from the theory and practice of the Web – Web architecture, standards, protocols, and the REST Architectural style.

The thesis work resulted in the creation of two specifications – the *WoDT Digital Twin Specification* and the *WoDT Digital Twins Platform Specification* – and the design of an Abstract Architecture for the creation of a *Web-based Web of Digital Twins*.

The *Web-based Web of Digital Twins* allows the creation of ecosystems of heterogeneous Digital Twins, where each Digital Twin can be developed with existing technologies and at the same time offers a compatibility layer towards the Web of Things. Digital Twins become independent Web services that are used to reason about the real world by creating personalized and contextualized views of reality.

The feasibility of the proposed design has been proved through the creation of a prototypical implementation that follows both specifications. It involved the usage of two different technologies for the development of Digital Twins, Azure Digital Twins, and the White Label Digital Twins framework, experimenting with the heterogeneity, openness, and interoperability of the envisioned ecosystems. Furthermore, by comparing and aligning with the Web of Things paradigm, it was possible to apply a compatibility layer using the WoT Thing Description.

The prototype demonstrated the successful use of the Web, hypermedia, and the REST architectural style to create a uniform interface that makes interactions independent of specific technologies. Moreover, the compatibility with the Linked Data principles has provided the ecosystem with strong navigability and semantics which allow the construction of the Platform Knowledge Graph for the execution of ecosystem queries. This enables any Consumer of the application layer to be able to navigate, observe, and reason within the Digital Twin layer as if they were directly interacting with the real world, obtaining the information of interest. However, the Knowledge Graph of the ecosystem, in particular the portion of the ecosystem covered by each singular Platform, is built in a centralized way. This represents a trade-off that simplifies query execution by maintaining a consistent ecosystem state with the cost of introducing data redundancy, which may result in querying stale data and creating a single point of failure for ecosystem-level services. Regarding the latter point, the Platform represents a single point of failure, but it is not the single entry point for the WoDT ecosystem. By leveraging the properties of the REST architectural style, the use of hypermedia, and the compliance with the Web Linking specification, Consumers do not need to be aware of the several Platforms of the ecosystem, but they can use any *WoDT Digital Twin* as an entry point.

The proposed idea does not constitute an alternative method of developing Digital Twins. The *WoDT Specifications* do not replace existing Digital Twins technologies, which must independently implement all necessary functionalities such as the shadowing process or the augmentation engine. Instead, the specifications act as an additional layer, either internal or external to the Digital Twin builders, to ensure compliance with *WoDT* ecosystems. Depending on the supporting Digital Twin framework or platform, if it requires the creation of an additional software layer – as for the Azure Digital Twins based WoDT Digital Twin – an additional layer will be added, which in some cases could cause latency issues. Furthermore, when the metamodel is not directly compatible with the *Web of Digital Twins* one, a significant effort is required to develop the *Abstract Architecture* components. This can be a challenging task for Digital Twin Developers. Therefore, for large-scale usage, a greater support for Digital Twin builders and frameworks is certainly crucial.

Additional analyses and evaluations in various scenarios are required to confirm the effectiveness of the Web protocols involved. For instance, the strategy that enables the *Memorization* requirement, so the *Memento* protocol, is the result of the first experiments, and it represents a first proposal that derives from one of the most used protocols on the Web. Then, the concept of queries on history, a well-known idea in literature, is missing from the text and warrants further exploration. Additionally, all the other types of Digital Twins that have emerged in recent years, such as Cognitive Digital Twins, are not considered. However, the proposed design is general and allows additional services to act on *WoDT Digital Twins* using all the information they expose. Other examples of services that in the literature are commonly seen as Digital Twins features are Simulation and Prediction.

Regarding the general definition and functionalities of the system, the provided interaction patterns and all the other details stated in the specifications, further experiments and analysis considering different use case scenarios, domains, Digital Twin technologies, and protocols are needed to evaluate and, in case, validate the proposed design. At the moment, it is not possible to provide a general and formal validity of the work because only a limited amount of tests have been carried out.

7.1 Future Work

Offering multi-domain and multi-organization support, the Digital Twins paradigm is a very broad research field that needs further exploration in indefinite paths.

For what concerns the thesis work, the specifications are in the initial stages and require additional research and formalization. Furthermore, it is essential to offer alternative implementations of the Digital Twin Descriptor by utilizing other established ontologies or paradigms. This will enhance compatibility with other standards and increase the openness and interoperability of the solution.

As for the complete WoDT ecosystem Knowledge Graph, it is currently only achievable by leveraging the layered system constraint of REST and developing custom services that merge data. Additional research is required to enable queries on Knowledge Graphs that involve multiple organizations and platforms.

During the thesis, there was the chance to analyze and study only a subset of the possible Web standards and protocols. Therefore, a valuable contribution would be to research new protocols or standards that could enhance the current design. An alternative approach has been analyzed and designed at a high level following the *Solid* specification[1]. Based on initial results, each WoDT Digital Twin could have a *Solid Pod* to store data. This approach was considered for several reasons, including control over decentralized data, the ability to store multivariate data (which is an important trend in modern Digital Twins), compatibility with the *Solid* ecosystem, and more.

[1]`https://solidproject.org/`

These are the main paths to follow. Further research is necessary to address the various challenges and fully explore the opportunities presented by the Digital Twin paradigm.

Bibliography

[1] ISO/IEC 30173:2023. Digital twin: concepts and terminology, 2023. Available at https://www.iso.org/standard/81442.html. [Last access: 2023-12-31].

[2] ETSI TR 103 827. Saref: Digital twins opportunities for the ontology context, 2023. Available at https://www.etsi.org/deliver/etsi_tr/103800_103899/103827/01.01.01_60/tr_103827v010101p.pdf. [Last access: 2023-12-31].

[3] Dean Allemang, Jim Hendler, and Fabien Gandon. *Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL*, volume 33. Association for Computing Machinery, New York, NY, USA, 3 edition, 2020.

[4] Tim Berners-Lee. Information management: A proposal. *CERN*, 1989.

[5] Tim Berners-Lee. World-wide web: The information universe. *Internet Research*, 1992.

[6] Tim Berners-Lee. Www: Past, present, and future. *Computer*, 29(10):69–77, oct 1996.

[7] Tim Berners-Lee. Linked data. https://www.w3.org/DesignIssues/LinkedData.html, 2006. [Online; Last access: 2023-12-31].

[8] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *ScientificAmerican.com*, 05 2001.

[9] Tim Berners-Lee, Larry M Masinter, and Roy T. Fielding. Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396, August 1998.

[10] Brickley and Guha. RDF Schema 1.1. W3C Recommendation, February 25 2014.

[11] Andrei Ciortea, Olivier Boissier, and Alessandro Ricci. Engineering world-wide multi-agent systems with hypermedia. In Danny Weyns, Viviana Mascardi, and Alessandro Ricci, editors, *Engineering Multi-Agent Systems*, pages 285–301, Cham, 2019. Springer International Publishing.

[12] Andrei Ciortea, Simon Mayer, Fabien Gandon, Olivier Boissier, Alessandro Ricci, and Antoine Zimmermann. A decade in hindsight: The missing bridge between multi-agent systems and the world wide web. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, page 1659–1663, Richland, SC, 2019. International Foundation for Autonomous Agents and Multiagent Systems.

[13] Herbert Van de Sompel, Michael Nelson, and Robert Sanderson. HTTP Framework for Time-Based Access to Resource States – Memento. RFC 7089, December 2013.

[14] Feigenbaum, Williams, Clark, and Torres. SPARQL 1.1 Protocol, W3C Recommendation 21 March 2013. W3C Recommendation, 2013.

[15] Roy T. Fielding. httprange-14. `https://lists.w3.org/Archives/Public/www-tag/2005Jun/0039.html`, 2005. [Online; Last access: 2023-12-31].

[16] Roy T. Fielding. Rest apis must be hypertext-driven. `https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven`, 2008. [Online; Last access: 2023-12-30].

[17] Roy T. Fielding, Mark Nottingham, and Julian Reschke. HTTP Semantics. RFC 9110, June 2022.

[18] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, may 2002.

[19] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures.* University of California, Irvine, 2000.

[20] Department for Business and Trade. National Digital Twin Programme (NDTP). `https://www.gov.uk/government/collections/the-national-digital-twin-programme-ndtp`, 2023. [Online; Last access: 2023-12-28].

[21] Gandon, Schreiber, and Beckett. RDF/XML Syntax, W3C Recommendation 25 February 2014. W3C Recommendation, 2014.

[22] David Gelernter. *Mirror Worlds or the Day Software Puts the Universe in a Shoebox: How Will It Happen and What It Will Mean.* Oxford University Press, Inc., USA, 1991.

[23] Edward Glaessgen and David Stargel. *The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles.* 2012.

[24] Michael Grieves. Product lifecycle management: Driving the next generation of lean thinking. 01 2005.

[25] Michael Grieves. Digital twin: manufacturing excellence through virtual factory replication. *White paper*, 1(2014):1–7, 2014.

[26] Michael Grieves and John Vickers. *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*, pages 85–113. Springer International Publishing, Cham, 2017.

[27] Digital Framework Task Group. *White paper: The Gemini Principles.* Technical report, Centre for Digital Built Britain, 2018. Available at `https://www.cdbb.cam.ac.uk/DFTG/GeminiPrinciples`.

[28] W3C OWL Working Group. OWL Web Ontology Language, W3C Recommendation 11 December 2012. W3C Recommendation, 2012.

[29] Dominique Guinard, Vlad Trifa, and Erik Wilde. A resource oriented architecture for the web of things. pages 1–8, 2010.

[30] Harris, Seaborne, and Prudhommeaux. SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013. W3C Recommendation, 2013.

[31] Hetherington James and West Matthew. *The Pathway Towards an Information Management Framework: A Commons for a Digital Built Britain.* Technical report, Centre for Digital Built Britain, 2020. Available at `https://doi.org/10.17863/CAM.52659`.

[32] Kaebisch, McCool, Kamiya, Charpenay, and Kovatsch. Web of Things (WoT) Thing Description. W3C Recommendation, 2020.

[33] Mike Kelly. JSON Hypertext Application Language. Internet-Draft draft-kelly-json-hal-11, Internet Engineering Task Force, October 2023. Work in Progress.

[34] Koster and Korkan. Web of Things (WoT) Binding Templates. W3C Group Note, 2023.

[35] Kovatsch, Matsukura, Lagally, Kawaguchi, Toumura, and Kajimoto. Web of Things (WoT) Architecture. W3C Recommendation, 2020.

[36] Markus Lanthaler. Hydra core vocabulary. Unofficial draft, 2021.

[37] Robert C. Martin. The Clean Architecture, 2012. Available at: `https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html`. [Last access: 2024-02-14].

[38] Nandana Mihindukulasooriya and Roger Menday. Linked Data Platform 1.0 Primer. W3C Working Group Note, 2015.

[39] Roberto Minerva, Gyu Myoung Lee, and Noël Crespi. Digital twin in the iot context: A survey on technical features, scenarios, and architectural models. *Proceedings of the IEEE*, 108(10):1785–1824, 2020.

[40] T. H. Nelson. Complex information processing: A file structure for the complex, the changing and the indeterminate. In *Proceedings of the 1965 20th National Conference*, ACM '65, page 84–100, New York, NY, USA, 1965. Association for Computing Machinery.

[41] Mark Nottingham. Web Linking. RFC 8288, October 2017.

[42] Marco Picone, Marco Mamei, and Franco Zambonelli. Wldt: A general purpose library to build iot digital twins. *SoftwareX*, 13:100661, 2021.

[43] Prudhommeaux and Carothers. RDF 1.1 Turtle - Terse RDF Triple Language, W3C Recommendation 25 February 2014. W3C Recommendation, February 25 2014.

[44] Alessandro Ricci, Angelo Croatti, Stefano Mariani, Sara Montagna, and Marco Picone. Web of digital twins. *ACM Trans. Internet Technol.*, 12 2021.

[45] Alessandro Ricci, Angelo Croatti, and Sara Montagna. Pervasive and connected digital twins—a vision for digital health. *IEEE Internet Computing*, 26(5):26–32, 2022.

[46] L. Sauermann and R. Cyganiak. Cool uris for the semantic web. W3c interest group note, W3C, 2008.

[47] Speicher, Arwe, and Malhotra. Linked Data Platform 1.0, W3C Recommendation 26 February 2015. W3C Recommendation, February 26 2015.

[48] Sporny, Longley, Kellogg, Lanthaler, Champin, and Lindstrom. JSON-LD 1.1 - A JSON-based Serialization for Linked Data, W3C Recommendation 16 July 2020. W3C Recommendation, 2020.

[49] Kevin Swiber, Tom Howard, Matthew Dobson, Nils Dagsson Moskopp (grand-centrix), Vladimir Tsukur, Adam, Sam Ward, ramirahikkala, Mike Raynham, javascript journal, Dylan Beattie, and Dominic Barnes. kevinswiber/siren: Siren v0.6.2, April 2017.

[50] Herbert Van De Sompel. Memento at the w3c. `https://www.w3.org/blog/2016/memento-at-the-w3c/`, 2016.

[51] Erik Wilde. Putting things to rest. 2007.