Web of Digital Twins

ALESSANDRO RICCI, Dept. of Computer Science and Eng., Univ. of Bologna, Italy
ANGELO CROATTI, Dept. of Computer Science and Eng., Univ. of Bologna, Italy
STEFANO MARIANI, Dept. of Sciences and Methods of Eng., Univ. of Modena and Reggio Emilia, Italy
SARA MONTAGNA, Dept. of Pure and Applied Sciences, Univ. of Urbino Carlo Bo, Italy
MARCO PICONE, Dept. of Sciences and Methods of Eng., Univ. of Modena and Reggio Emilia, Italy

In recent years, digital twins have been pervading different application domains – from manufacturing to healthcare – as an approach for virtualising different kinds of physical entities (things, products, machines). The dominant view developed in the literature so far is about the virtualisation of individual physical assets, in a closed-system perspective. In this paper, we introduce and explore a broader perspective that we call *Web of Digital Twins* (WoDT), in which the digital twin paradigm is exploited for the pervasive softwarisation of possibly large-scale interrelated physical realities. A WoDT 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. The paper introduces an abstract model and architecture aimed to capture key aspects of the idea not bound to any specific application domains or implementing technologies, and discusses their adoption in engineering real-world systems. To this purpose, two concrete case studies are considered, in the context of healthcare and smart mobility. Finally, the paper includes a discussion of a selected set of research directions.

CCS Concepts: • **Computing methodologies** → **Multiagent systems**; • **Software and its engineering** → **Designing software**; • **Computer systems organisation** → *Embedded and cyber-physical systems*; • **Information systems** → *World Wide Web*.

Additional Key Words and Phrases: Digital Twins, Web, Agents, MAS, WoDT

1 INTRODUCTION

In the last decade, the digital twin (DT) paradigm has been explored in different domains as an approach to virtualise entities existing in the real world, creating software counterparts that provide smart services upon them [16, 17, 28]. Such services may range from simple tracking of the actual state of the physical entity or device, to smarter forms of monitoring in order to, e.g., detect and predict possible critical situations, optimise performances, up to more general forms of augmentation of the capabilities of the physical counterpart. Relevant examples can be found in the Industry 4.0 context [52], healthcare [24], smart cities [44]—the interested readers can refer to surveys available in the literature [28]. Despite the specific domain and implementation, the models of DT described in the literature share two main characteristics: *(i)* they typically concern virtualisation of *individual*, standalone assets, in a *closed-system* perspective—being them physical objects, products, machines, buildings;

Authors' addresses: Alessandro Ricci, Dept. of Computer Science and Eng., Univ. of Bologna, Cesena, Italy, a.ricci@unibo.it; Angelo Croatti, Dept. of Computer Science and Eng., Univ. of Bologna, Cesena, Italy, a.croatti@unibo.it; Stefano Mariani, Dept. of Sciences and Methods of Eng., Univ. of Modena and Reggio Emilia, Reggio Emilia, Italy, stefano.mariani@unimore.it; Sara Montagna, Dept. of Pure and Applied Sciences, Univ. of Urbino Carlo Bo, Cesena, Italy, sara.montagna@unibo.it; Marco Picone, Dept. of Sciences and Methods of Eng., Univ. of Modena and Reggio Emilia, Reggio Emilia, Italy, marco.picone@unimore.it.

1533-5399/2022/1-ART1 $15.00
https://doi.org/10.1145/3507909

(ii) they are used for *vertical* applications, designed for specific purposes. Beyond this view, the DT principles and paradigm can be extended to the virtualisation of complex realities composed of interrelated assets, possibly belonging to different domains and different organisations, in a more *open-system* perspective [27, 39]. Such a stance, besides enabling technologies, calls for a proper conceptual model and framework, abstract enough to capture key aspects in spite of concrete application domains and technologies, and yet expressive enough to be a reference for the development of concrete architectures and technologies.

To this purpose, in this paper we introduce and discuss an approach in which the DT paradigm is meant to be pervasively applied to virtualise large-scale, dynamic, possibly cross-domain physical realities of an organisation and across different organisations, resulting in an open distributed ecosystem of connected DTs. We refer to such an ecosystem as *Web of Digital Twins* (WoDT), being inspired by the main conceptual and architectural principles of the Web, and considering the Web, and related technology stack and standards, as a natural underlying deployment architecture and platform—although not necessarily the only one. In this view, a DT is not (necessarily) a vertical application: conversely, the WoDT of an organisation defines a service-oriented software layer on top of which smart applications can be designed and integrated, exploiting functionalities to access and interact with the interrelated physical assets *as-a-service*.

At the application level, a main kind of systems that can take advantage of WoDT as a service are intelligent agents and multiagent systems (MASs) [20], that can exploit DTs as a virtual environment (or, application environment [53]) enabling the access and interaction with the physical world. In this view, a DT functions first of all as a shared medium used by agents to perceive/observe and act upon the physical world. Besides, a DT may provide further higher-level functionalities conceptually augmenting the basic ones provided by the physical world, that could be exploited by agents to support their reasoning and decision making.

The remainder of the paper is structured as follows. We first provide a broad overview and background about DTs (Section 2), and their added value for reference contexts such as Internet of Things. Then, the main aim of the paper is to provide a comprehensive account of the WoDT vision and approach. To this purpose, first, we describe an abstract model capturing key concepts and features (Section 3) and the general traits of architectures based on that model (Section 4), including a discussion about their integration with multiagent systems architectures and technologies. Then, we discuss the application of the model to two real-world concrete case studies (Section 5), based on our previous work exploring the application of DTs in specific domains, in particular healthcare, for major trauma management [29], and in smart mobility scenarios [36]. Finally, we provide an overview of the main research directions for the development of the WoDT vision and approach (Section 6).

2 BACKGROUND AND STATE OF THE ART

The scientific literature has referred to DTs since 2003 when Michael Grieves introduced this concept with an initial formulation in the aerospace field by the National Aeronautics and Space Administration (NASA) [16, 17]. As reported in [28, 51], from there the concept has evolved and attracted growing attention, from manufacturing industries to the Internet of Things and Cyber-Physical Systems contexts. In particular, [28] surveys and analyses the state-of-the-art definitions also investigating the common characteristics of a DT and the domains in which they are currently being developed and adopted.

The original definition introduced the core concepts associated with a DT, namely it is composed by three dimensions: *physical*, *virtual*, and *connection* parts, where the virtual space represents the digital or software representation and replication of the physical asset, and it is mapped to the physical space through the connection part that exchanges information. Moreover, DT possibly includes models of the structure, functionalities, and behaviour of the real counterpart [17, 32]. It can persist for the whole system life-cycle, and it is tightly linked with the physical entity: a *shadowing* process enables the continuous update of DT's internal state in near-real-time

with data acquired on the physical system by different devices – mainly sensors or other sources such as existing IT systems (e.g. ERP, PLM) – and transferred digitally [7].

Recent advancements in IoT, big data, and machine learning have also significantly contributed to the improvements in DTs regarding their real-time capabilities and forecasting properties. Collected data constitute the so-called *digital threads* and are the grounding information on which simulation or machine learning algorithms rely to make predictions, enabling failures to be anticipated, to optimise the system, to design novel features, to ease and accelerate decision making, and to improve productivity— to mention some [43, 52]. According to this definition, the DT is not only a model of the physical asset, but it can autonomously evolve through simulation and AI-enabled algorithms to understand the world, learn, reason, and answer to *what-if* questions. Furthermore, whenever DTs encapsulate reasoning capabilities, the concept of DT has evolved into Cognitive Digital Twin (CDT) [1, 13] that has been introduced in the literature to refer to those DTs that autonomously perform some intelligent task within the context of the physical asset, related to e.g. smart management, maintenance, and optimisation of performances. This corresponds to stage 4 DTs envisioned in [43], as extended DTs delivering additional capabilities besides the physical asset ones, possibly including an autonomous part flanking the basic DT ones. To support cognitive and analytical solutions, some works in literature propose the adoption of semantic models and technologies to extract knowledge from data, building on specific domain-driven ontologies. Semantic relations among data may then be represented as knowledge graphs [42], enabling the exploitation of a set of models and theories to enhance the DT with cognitive capabilities. As such, DTs attracted a multitude of specific approaches related to data analytics [41], behavioural modelling [45], ontology definition [49], or specific device mirroring [46] and networking.

As clearly reviewed and pointed out also in [28], the literature is conceptually aligned on an idea and the importance of DT in multiple fields, but there is not yet a shared set of properties and behaviours that can help to create common background, language, and a unifying model for representing and properly work with DT across multiple application domains. The fragmentation of existing solutions is mostly related to their specificity for a target sector and the missing detailed definition of how DTs should be represented and operate. On the one hand, the resulting trend generates innovative approaches in disparate fields. However, on the other hand, it limits the real potential of uniformed DTs by creating an unnecessary substrate of heterogeneous proposals [50]. Currently, it is almost impossible to create an ecosystem where devices, services, and users can efficiently cooperate through a shared and interoperable DT vision. From this analysis, the definition of a model that introduces concepts and principles on top of which building a DT, crosscutting the different application domains and independent of specific technologies, emerges as an open issue.

The industrial world, particularly the Industrial Internet of Things Consortium, is proposing a shared reference architecture [25, 47] taking into account DTs relationships, composition, and main services (e.g., prediction, maintenance, safety). In the networking research field, DTs are also recently adopted to support interoperability, reduce heterogeneity by providing a dynamic application-driven layer on top of physical equipment [3]. Furthermore, the concept of network DTs appears also for the first time as an informational draft [56] trying to define their role and main responsibilities to mirror network assets. Similar works have been done by the Robotics and Robot Operating System communities [22, 23]. In this context, a set of platforms and solutions has been developed by major industries. It is worth mentioning the vision of GE Digital[1], Siemens[2], and Azure Digital Twin[3]. The latter, in particular, provides a comprehensive approach for designing and developing cross-domain digital twins, including – among the other features – a language called DTDL (Digital Twin Definition Language), which makes it possible to describe graphs of DTs, representing both their properties and their relationships.

[1]https://www.ge.com/digital/applications/digital-twin
[2]https://new.siemens.com/global/en/company/stories/research-technologies/digitaltwin/digital-twin.html
[3]https://azure.microsoft.com/en-gb/services/digital-twins/

This feature is an essential one also in the WoDT proposal, in which – however – a more open-system oriented perspective is explicitly adopted, taking the Web and Semantic Web as main references.

The oneM2M organisation[4] and the World Wide Web Consortium with Web of Things (WoT)[5] are actively working to provide uniform access and description of physical assets to achieve practical interoperability across multiple application domains and deployments. Unfortunately, within these fundamental standardisation activities, the definition of the role of DTs is at an early stage, e.g. the WoT tries to introduce the concept mainly as a cloud-driven interaction pattern instead of a fundamental tool to digitise and model physical assets. Market ready DTs approaches are also mainly focused on legacy systems design, and providers like Amazon[6], Google[7], and Bosch[8] already proposed their siloed implementations and DT services.

The broader perspective brought by the WoDT proposal shares many points with the *Gemini Principles* vision [27], on which the National Digital Twin (NTD) Programme developed in the UK is based. The NDT programme is nationwide, but focused mainly on the built environment. The perspective of WoDT is even larger, considering the opportunity of virtualising physical assets not limited to buildings or related physical objects. In the literature, this pervasive vision has strong affinities with the idea of *mirror worlds* as introduced by D. Gelernter in [15], and further explored and developed in the context of agents and multiagent systems in [40]. Following Gelernter, mirror worlds are "*software models of some chunk of reality*" [15], that is: "*a true-to-life mirror image trapped inside a computer*", which can be then viewed, zoomed, analysed by real-world inhabitants with the help of proper software (autonomous) assistant agents. Following [15], the primary objective of a mirror world is to strongly impact the lives of the citizens of the real world, offering them the possibility to exploit software tools and functionalities provided by the mirror world, generically, to tackle the increasing life complexity. The same vision applies to Web of Digital Twins, which could be considered a concrete approach to design and develop mirror worlds under this perspective.

Finally, the literature already accounts for a few works that apply agents for modelling, designing, implementing, or even exploiting DTs. In [2] BDI agents – being BDI (Belief-Desire-Intention) a main model/architecture adopted to implement knowledge-based intelligent agents [38] – are proposed to represent DTs of real-life organisations claiming that beliefs, desires, and intentions are suitable abstractions for characterising mental attitudes of anthropomorphic organisations. A similar approach is proposed in [48] where agents are adopted as a metaphor to revise the structure of a DT in an autonomous, behaviour-centred perspective encapsulating the inherent agent's perception–decision–action cycle and intelligence. Compared to these works, WoDT is more focused on exploring intelligent agents and MAS at the application layer, modelling a digital twins connected ecosystem as an agent application environment [53].

3 THE WODT MODEL

In this section, we provide a description of the main concepts and principles defining the WoDT idea, abstracting from specific application domains and technologies. Nevertheless, to clarify the concepts, we will use examples from concrete domains, the healthcare scenario in particular.

3.1 Overview

The WoDT idea is based on a background principle introduced in [39] to broaden the perspective about the application of the DT paradigm:

[4]https://www.onem2m.org/
[5]https://www.w3.org/TR/wot-architecture/
[6]https://aws.amazon.com/it/iot/
[7]https://cloud.google.com/solutions/iot
[8]https://www.bosch-iot-suite.com/

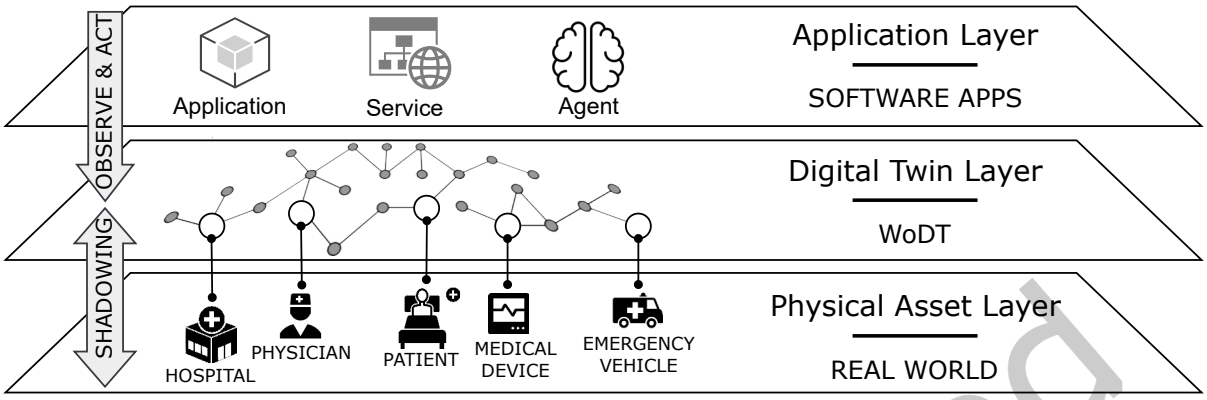


Fig. 1. The WoDT Layered View.

Every strategic physical asset of an organisation must have a corresponding digital twin, mirroring and augmenting its functionalities and services at the digital level, resulting in an ecosystem of connected digital twins.

A WoDT is meant to serve as a blueprint to shape that idea of ecosystem from a computational point of view. The term *physical asset* (PA) mentioned in the principle is intentionally used as a broad term, to include any entity that has some kind of manifestation and relevance in the physical world of the organisation, with a well defined temporal lifespan[9]. It can include physical objects/resources, places, persons, but also activities and processes carried on by people in places. For instance, in the context of the healthcare/clinical scenario later described in Section 5, the organisation is a regional public health authority, involving multiple hospitals and structures distributed on a regional land. In that context, examples of PAs range from vehicles and devices (e.g. an ambulance or a vital signs monitor), building and places (e.g. a hospital, an operating room), persons (e.g. a patient), up to activities and processes happening real-time on the field (e.g. the management of a trauma, a surgery in the operating room), as well as logical aggregated entities, such as a department or the global organisation itself (see Figure 1).

In spite of specific cases, a DT is meant to capture and represent at a proper *level of abstraction* the actual state and functionality of the PA, possibly augmented by the digital layer, and what's happening to it (as well as what happened and what will happen). For a proper level of abstraction, here we mean two things:

- the DT defines a *model* of the PA, so abstracting from aspects of the PA that are not relevant for its purpose;
- the representation provided by the DT is about concepts that concern the PA at the *domain level*—not technical aspects related to how its digitisation is being implemented.

Generally speaking, a DT could host multiple concrete models of the same PA, capturing different aspects. In this paper we will refer to a single abstract model, that may be ground then to multiple concrete ones, without losing generality.

WoDT as Open, Dynamic System of Linked Systems. Two characterising points of DTs in WoDT concern *dynamism* and *relationships*. In particular:

[9]The definition of entity used here is analogous to the one defined by the ISO 24760-1:2011: "entity is an item that has recognisably distinct existence, e.g. a person, an organisation, a device, a subsystem, or a group of such items."

- The PAs part of a WoDT could include both entities that are stably part of the organisation, sharing the same lifespan, and entities with a limited temporal existence, beginning to exist or to be part of the organisation at some point in time and possibly ending or exiting the organisation at some other time. Correspondingly, DTs bound to PAs could either be part of a WoDT since the beginning, or dynamically created and possibly disposed.
- The PAs of an organisation are typically interrelated, and this set of domain-based relationships could change dynamically. For instance, in the healthcare scenario, an ambulance belongs to a hospital and could take part in a mission-related to an emergency event. In a WoDT, these relationships are meant to be explicitly captured and represented at the DT level, by means of *links* among the DTs, similarly to link in hypermedia-based environments (like the Web), and with some defined semantics based on domain-level ontologies (like in the case of Semantic Web).

WoDT Distributed Knowledge Graph. The PAs of an organisation could be in relationship with PAs of *other* organisations, or even the same PA could take part in different organisations with different roles. Accordingly, a DT of an asset inside an organisation could be linked/related to DTs that are outside the organisation. This implies the capability in WoDT to deal with the problem of *interoperability*, e.g. allowing to use multiple/different ontologies, possibly cross-domain, in an open-system perspective. The semantic modelling of each virtualised 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. To this purpose, each DT of a WoDT is meant to be described by a *knowledge graph* (KG) [18, 19], interlinking domain knowledge and physical asset data in a uniform graph representation. A WoDT is therefore represented by a Distributed KG (DKG) [21], linking independent KG, possibly based on different domain-specific ontologies ground to the related physical asset contexts. Semantic Web technologies such as RDF and OWL are taken as the main reference for this aspect.

WoDT at the Application Level. From an application model point of view, a WoDT is meant to define a *cross-application* distributed base layer bridging the digital and physical levels and DTs could serve *as-a-service* different applications, running inside or outside the organisation. In the healthcare scenario, for instance, the DT of the ambulance could be useful for different specific applications, e.g. one about maintenance of the vehicles and one about the allocation of vehicles in the management of an emergency. Furthermore, the same DT can serve as a traffic management application prioritising emergency vehicles over private traffic and public transportation. In the most general case, for the same PA multiple and independent DTs can be available, each one with a different model, specialised for different applications.

A main kind of applications that may benefit from the availability of WoDT is given by intelligent agents [54] and multiagent systems [20], i.e. intelligent systems designed to autonomously perform tasks that need a flexible interaction with the physical/socio-technical environments where they are situated. In this view, DTs can be considered as shared and modular services that intelligent agents can exploit to perceive and observe the state and events of PAs, based on the semantic models provided by the DTs in terms of knowledge graphs. Besides the support for perceiving and observing, DTs may provide actions that allow agents to possibly affect, control and manage the corresponding physical twins. In other words, from an intelligent agent perspective, a WoDT would provide a distributed dynamic *application environment* [53] enabling, mediating and empowering the access to the physical reality.

After this broad overview, in the remainder of the section we describe an abstract model for WoDT, to be useful as a reference for designing and developing WoDT platforms and technologies.

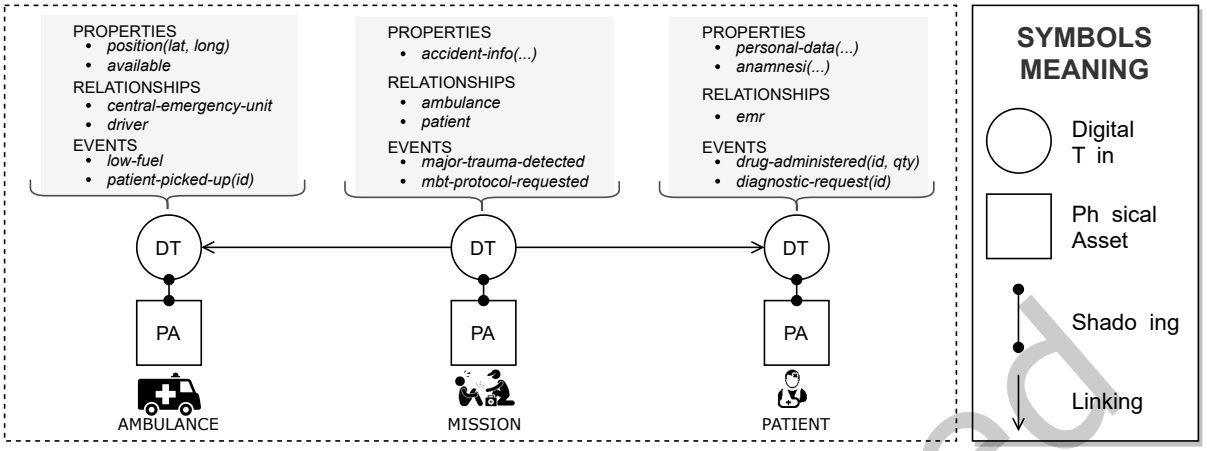


Fig. 2. A representation of three DTs of three different PAs (Ambulance, Mission, Patient), including an excerpt of their models, in terms of properties, relationships and events that can be generated.

3.2 An Abstract Model

Each DT in WoDT is based on a model M of the corresponding PA, defining how the PA is represented at the digital/software/virtual level. Such a representation is defined in terms of *properties*, *relationships* and *events*:

- *Properties* represent the observable attributes of the PA, as labelled data values (variables) that can change dynamically according to the evolution of the PA state.
- *Relationships* represent relationships of the PA with other PAs, as links to other digital twins. Like properties, even relationships can be observable, created dynamically and change over time. Differently from properties, they do not purely concern the local state of the PA, but they allow to refer other PAs, represented by the corresponding DTs.
- *Events* represent relevant observable events that occurred at the PA, at the domain level.

A concrete model defines the actual properties and relationships used by the DT to represent the PA and the events that it can dynamically generate. An example related to the healthcare case study is shown in Figure 2.

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

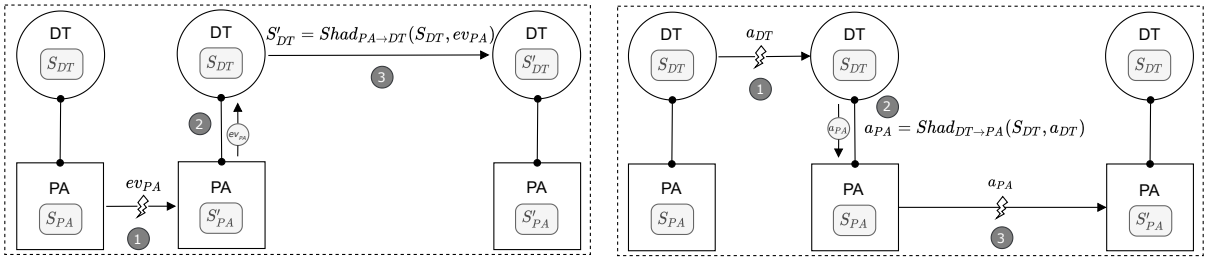
$$S_{DT} = \langle P, R, E, t \rangle$$

where P is the current set of properties (including data values), R is the current set of relationships, E is the sequence of events generated so far, and t is a logical timestamp representing the current time of the PA as modelled by M.

Shadowing is the process to keep the DT state S_{DT} synchronised to the PA state, according to the model M. Any update involves a sequence of three main steps (see Figure 3a):

(1) any relevant change of the state S_{PA} occurring at the PA is captured by an event ev_{PA};
(2) the event ev_{PA} is propagated to the DT;
(3) given a new event ev_{PA}, the state S_{DT} of the DT is updated by means of a shadowing function $Shad_{PA \to DT}$ that depends on the model M: $S'_{DT} = Shad_{PA \to DT}(S_{DT}, ev_{PA})$.

In concrete systems, PAs can be complex entities, with a structured and distributed state. The shadowing process then may involve multiple sources generating information flows and events. Sources can also include other DTs,



(a) Shadowing process from PA to DT: a change of the PA state S_{PA} results in updating the DT state S_{DT}.

(b) Shadowing process from DT to PA: an action request on the DT leads to updating the PA state S_{PA}.

Fig. 3. The Shadowing Process in the WoDT Model.

that is: a DT can shadow a high-level logical PA (e.g. the DT of an organisation) by aggregating information and events provided by other DTs.

Besides mirroring the state, a DT may mirror also *actions* provided by the PA. A simple example is the DT of a lamp, providing actions to switch the light on and off. Accordingly, the shadowing process propagates actions requested on the DT down to the PA, eventually changing its state. This case too involves a sequence of three main steps (see Figure 3b):

(1) an action a_{DT} is requested on the DT, e.g. through the digital twin API;
(2) a new action request a_{PA} for the PA is generated by means of a further shadowing function $Shad_{DT \to PA}$, that is: $a_{PA} = Shad_{DT \to PA}(S_{DT}, a_{DT})$, and propagated to the PA;
(3) the action request a_{PA} is applied to the PA, determining a change of the PA state S_{PA}.

It is worth remarking that an action request a_{DT} does not directly change S_{DT}. Changes to S_{DT} are uniquely caused by shadowing from PA to DT—so, in this case as a result of the PA state change, after applying a_{PA}.

Overall a WoDT is then a dynamic set of independent DTs, each one with its own model and state, linked according to the relationships defined at the PA level. A WoDT is then inherently asynchronous and decentralised—the DTs of a WoDT may have different and independent time models, and evolve independently and asynchronously.

3.3 A Semantic Model based on (Distributed) Knowledge Graphs

The abstract model defined above makes it quite straightforward to semantically describe an instance of DT by a knowledge graph (KG) [19], and a WoDT as a Distributed KG. This is a key aspect of WoDT to enable cross-application/domain interoperability, and support reasoning by intelligent systems running at the application layer.

By using RDF as concrete representation language, the KG of each DT can be represented as an RDF resource characterised by a unique IRI, using RDF triples to represent dynamic state information about properties, relationships, events and time, as well as the static information of the DT. In triples about properties, the predicate is the property name (identifier) and the object is the value of the property—that can be represented either by a literal or the IRI of another resource (in a Linked Data perspective). In triples about relationships, the predicate identifies the relationship name (identifier) and the object is the IRI of the linked DT, corresponding to the target PA which is related to the source PA, mirrored by the linking DT.

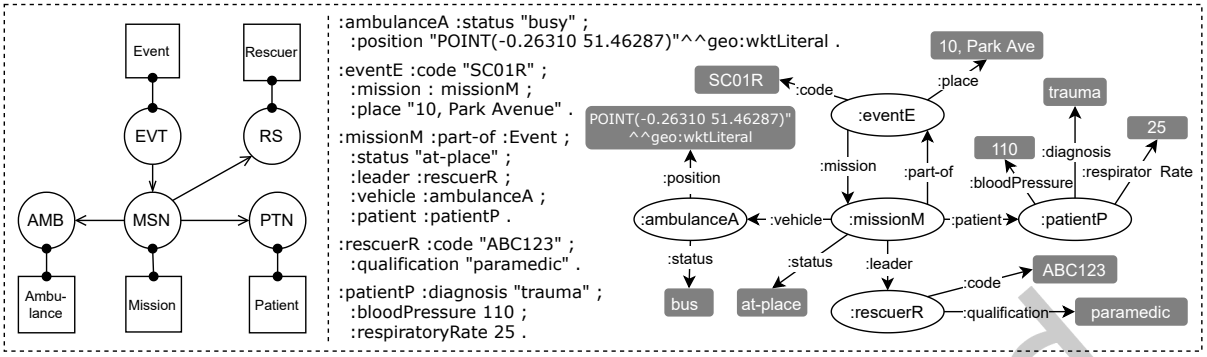


Fig. 4. An example of a RDF-based KG (on the right) for a WoDT (on the left) in the healthcare context.

Figure 4 shows a representation of a portion of Distributed KG related to the DTs of a previous example, represented in RDF. Each KG can be based on different ontologies, expressed in OWL, including both domain-specific ontologies and shared upper ontologies. For instance, the DTs in the healthcare context example and case study can refer to FHIR RDF representation[10] and FHIR OWL Ontology[11].

Dynamically, the KG of a DT instance evolves according to the shadowing process, involving atomic updates of the set of triples. The Distributed KG of a WoDT evolves by virtue of the asynchronous and concurrent evolution of the individual KGs.

3.4 Interaction Model

The interaction model is about the primitives (API) that are provided at the application level to interact with DTs and exploit the functionalities of a WoDT.

The first core functionality is about making observable at the digital level any up-to-date information about the current state of physical assets, as well as events relevant at the domain level. The interaction primitives are then (i) to query and (ii) to track (observe) DTs, at two different levels: (i) individual DTs (ii) and graphs of DTs.

Both querying and tracking (observing) account for getting information about the current state S_{DT} of a DT or a graph of DTs. Querying is about one-shot requests and – given the semantic modelling adopted here – standard semantic query languages like SPARQL[12] can be used as reference to this purpose. Tracking is about subscribing to either a DT or a graph of DTs to receive all observable events (that is, all relevant events occurred in the PA), possibly filtered according to patterns specified with the subscription. Subscription is meant to be dynamic, by means of interaction primitives for start tracking (subscribing) and stop tracking (unsubscribing). Remarks:

- Queries and tracking cannot interfere or block the shadowing process. That is: updates from the physical world have priority and are meant to be performed satisfying the requirements (in terms of e.g. latency, responsiveness, etc) defined by the model. This is an important constraint for concrete architectures and strategies to be adopted to design a platform supporting the WoDT model (Section 3).
- In the most general case, queries (and tracking) that involve a graph of DTs concern Distributed KGs, where it cannot be assumed a-priori neither a unique reference time, nor a common model of time (which is defined by the model M of each DT).

[10] https://www.hl7.org/fhir/rdf.html, accessed in September 2021.
[11] https://w3c.github.io/hcls-fhir-rdf/spec/ontology.html, accessed in September 2021.
[12] https://www.w3.org/TR/sparql11-query/

Besides querying and observing, a DT can mirror also the actions provided by the physical asset to command/control it. Therefore, interaction primitives may include application-specific requests for asynchronously executing actions/commands, shadowing those provided by the PA.

Finally, a last core functionality is about creating (and disposing) DTs. In a WoDT, the creation of a new DT can occur in three conceptually different ways:

- *statically*, when a DT is instantiated and configured by administrators. This typically concerns either standalone systems or root DT of a possibly complex WoDT, where the other DTs are then created dynamically;
- *dynamically by shadowing*, when a DT is created as effect of the shadowing of an existing DT (and PA), possibly linking the new DT by means of some relationship (part of the R set, in the abstract model). In this case the existing DT can be considered the *parent* of the new DT;
- *dynamically by the application level*, when a DT is created as effect of an action requested from the application layer. It could concern the creation of a DT which is either unrelated to any existing DTs, like in the first case, or created in the context of an existing parent DT (linking it by some relationship).

3.5 Modelling Augmentation

A DT can be used not only to virtualise a PA, making its digital shadow accessible and exploitable, but also to extend (*augment*) its functionalities by properly exploiting the digital/software layer [28]. For instance, the DT of a room can provide a property about the number of people inside the room (by exploiting different kind of tracking technologies), even if – at the physical level – there could not be any physical counter. An another example, the DT of a patient can generate a warning event about the health state, given e.g. rules defined by medics, possibly contextualised to the specific situation. Event prediction and simulation – which are main high-level functionalities that are described in the literature for DTs – can be conceptually framed as augmentation, since they are not part of the functionalities mirrored from the PA, but exploit the model M, the state S_{DT} and possibly other available data to generate information about the future states/behaviour of the PAs.

In the abstract model, augmentation can be represented by means of an augmented state S_{AU} including a further set of properties, relationships, and events besides the ones generated by shadowing the PA. The augmentation behaviour can be modelled as an abstract functions *Augm* part of the model M too, like the shadowing functions *Shad*, so that:

- an event ev_{PA} occurring at the PA level may trigger both the update of the state S_{DT}, according to the basic shadowing process, and the update of the augmented state S_{AU} according to the augmentation function: $S'_{AU} = Augm_{PA \to DT}(S_{AU}, S_{DT}, ev_{PA})$;
- an action event a_{DT} part of the augmented behaviour may cause the update of the augmented state into a S'_{AU} and (possibly, not necessarily) generate an event a_{PA} to be propagated to the PA, as in the basic shadowing process: $a_{PA} = Augm_{DT \to PA}(S_{AU}, S_{DT}, a_{DT})$.

In this modeling, the augmented state S_{AU} is kept separated by the state S_{DT} to remark the conceptual difference between them: properties and relationships of the core state are strictly bound to the PA and its evolution, and cannot be (directly) changed by the application layer, which is the case, instead, of the properties and relationships of the augmented state.

4 BRINGING THE WODT MODEL INTO THE REAL WORLD

The WoDT model needs to be supported by an abstract DT's life cycle and an adequate software architecture to be effectively and efficiently deployed as an operational artefact. We thus provide here a plausible *abstract* representation of both.

Figure 5 depicts the envisioned DT life cycle. After startup, the DT moves to the *Operating & Not Bound* state where all the internal modules are active but the DT is not yet associated with the PA. It is the *binding procedure* that connects the two, according to the existing domain-specific requirements. In the *Bound* state the DT is correctly attached to its physical counterpart, hence is able to handle bidirectional events, interact with the PA, and start the shadowing process in order to be effectively synchronised in terms of events and state—reaching the *Shadowed* state.

Any error during synchronisation brings the DT into a new state denoted as *Out of Sync*, where it is unable to handle events, to align its status, or to interact with the external world. Only once synchronisation is correctly recovered (according to the defined model) the DT returns to the *Shadowed* state. During its life cycle, the DT can be also stopped and moved to the *Done* state, where it is still active and accessible from external applications and consumers (maintaining its memory and events log), but it is neither bound or synchronised anymore with the PA. At the end of its life cycle, the DT can be finally dismissed and associated to the *Stop* state.

Given the WoDT model and the DT's life cycle just described, we now present a WoDT blueprint architecture, conceived by *(i)* making explicit the *requirements* put forth by the WoDT model, and *(ii)* devising out the abstract architectural components, as well as their role and relationships, needed to fulfil them.

It is worth remarking that the described architecture is by no means meant to serve as the *unique* reference architecture for implementing an ecosystem of DTs, rather, as an abstract architecture where each component (hence its functional and non-functional responsibilities) may be possibly realised by a slew of different existing models and technologies. We tackled the problem of both platform and DTs' complexity by decomposing them into a set of manageable event-driven modules along with suitable adapters, with the aim of simplifying development, maintenance, and scalability. This approach also reduces the barriers to adoption by not being bound to a specific target platform and domain-specific solution. Finally, this flexibility and modularity also enable each component to be deployed independently and dynamically according to the application scenario or the run-time context. For example, DTs can be executed both on the Edge and in the Cloud, or can migrate among multiple processing nodes. At the same time, the platform is responsible for maintaining the event-driven communication and the distributed knowledge available.

4.1 An Architectural Perspective

By reflecting on the model described in Section 3 we can devise out the following architectural requirements: *(i)* it should be possible to *create* DTs and *bind* them to PAs dynamically, which in turn requires to dynamically associate DTs and PAs with an *addressable* and *discoverable* unique identifier, and to provide the means to resolve and discover such address; *(ii)* changes in the state of a PA, that is, *PA events*, should be captured and reified

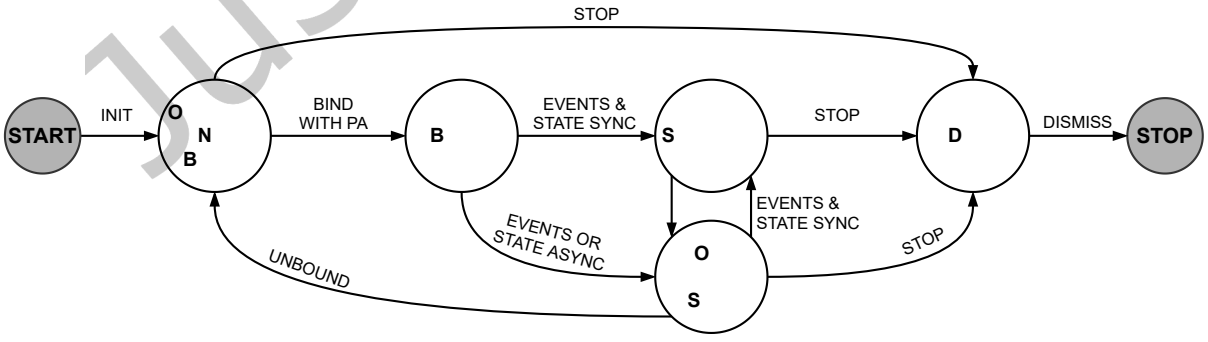


Fig. 5. Abstract representation of the state and transitions of a DT's life cycle.

in a *uniform representation*, regardless of the *heterogeneity* of the source PAs, by the bound DT, whose state may change in response as defined by the DT model(s); *(iii)* a labelled multi-graph (the knowledge graph) is needed to track dynamic *linking* amongst DTs, and means to navigate and query such labelled multi-graph should be provided; *(iv)* the shadowing process must guarantee proper *synchronisation* between the PA and the DT, according to the constraints put forth by the DT model(s)—e.g. in terms of quality of service metrics, amongst which timing constraints; *(v)* an *operational specification* of the DT model(s) must be available for execution at all times, to drive the processes of event capturing, state update, shadowing itself, linking, namely the whole inner functioning of the DT; *(vi)* some services must be available independently of any DT, such as for DT creation and querying; *(vii) observation* of DTs' current and past state and augmented state, model(s), thread of captured events, context in terms of linking sub-graph it participates to, and any other relevant data related to DTs functioning must be available at all times, to external entities, regardless of their heterogeneity (e.g. web service vs. cognitive agent); and *(viii) interaction* with the DT, and consequently the PA, in order to trigger actions and functions, also regarding augmentation, must be possible at all times, and will spawn events giving feedback about the action itself (e.g. results), and possibly generate state updates in the PA or the DT.

Based on these requirements, we define the abstract architecture depicted in Figure 6 as the minimal architecture fulfilling all the requirements described above. Such architecture exposes *(i)* elements which are part of each DT (e.g. Model Execution Engine), hence are conceptually "inside" of a DT, *(ii)* elements which are at the boundary between DTs (e.g. Management Interface), *(iii)* elements which are the boundary between DTs and client entities (e.g. Digital Adapter), and *(iv)* infrastructural elements which are not part of any DT (e.g. DT Manager), hence are conceptually "outside" of a DT.

First we first focus on the inside of a DT, that is, what the components providing the functions that each DT should be able to deliver autonomously are:

Physical Asset Adapter (PAA) The component in charge of *capturing events* e_{PA} coming from the PA(s) associated to the DT (due to, e.g., a change to the PA(s) state), as well as of delivering events e_{PA} to the PA(s), corresponding to actions to be carried out, as requested by applications through the Digital Adapter. Despite *heterogeneity* of PAs, in terms of structure, purpose, behaviour, communication protocols, data representation formats, and so on, such events must be mapped to a *uniform representation* e_I so as to be

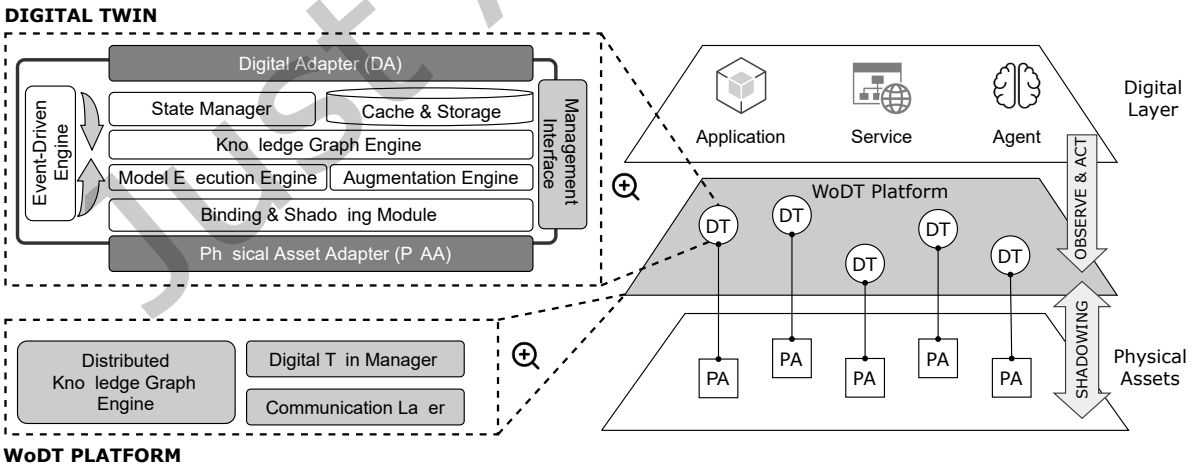


Fig. 6. Overall abstract Architecture of WoDT vision (inner DT and WoDT platform).

seamlessly exchanged amongst different DTs, filtered, manipulated, aggregated, and possibly dispatched to external entities. A fundamental aspect of this mapping is the preservation of some *temporal information* about e_{PA}, such that, for instance, *causal relationships* could be established amongst resulting e_I, and *synchronisation status* can be assessed. Such e_I will be processed within the DT by the Binding & Shadowing Module and the Model Execution Engine, as described below, thanks to the Event-driven Engine, which drives the internal behaviour of each DT. In the case that the PA is mapped into another DT, events e_{PA} are already represented as e_I, hence no further translation is required.

Binding & Shadowing Module (BSM) The "heart" of the DT, that is, the component in charge of both the one-time *binding process* associating a DT with its PA, usually done at DT creation time and dismissed when the PA is disposed, and the perpetual *shadowing process* meant to keep the DT and PA in synch. This module interacts with the Event-driven Engine to dispatch events e_I to the Model Execution Engine, and operates according to the policies put forth by it regarding when and how to update the DT state and the Knowledge Graph. Finally, this module also tracks and governs the *lifecycle stages* of a DT, from creation and binding, to unbinding and disposal.

Event-driven Engine (EE) The "nervous system" of the DT, that is, the component *binding together* all the other internal components of a DT, by enabling their reciprocal interaction through events e_I. It is worth emphasising here that we interpret the DT as an event-driven machinery *conceptually*, at the modelling level; however, as already said, we do not constrain the DT to be actually implemented as such, hence the EE in turn may be not, for instance, an internal or shared event bus, but anything else fitting the job.

Model Execution Engine (MEE) The "brain" of the DT, that is, the component in charge of *governing* other components according to the model(s) defined by the DT designer. As such, it dictates which events to capture, how they influence the DT state and behaviour, the admissible linking operations on the KG, the admissible actions on the corresponding PA, etc. Conceptually, through the MEE, the DT designer has the means to make the model(s) she defined *operational*, that is, capable of affecting the behaviour of the DT at run-time.

State Manager (SM) The component responsible for managing DT state updates, according to the policies put forth by the MEE, the events dynamically captured, and the contextual conditions defined by the linking relationships with other DTs. While doing so, particular attention should be devoted in leaving the DT in a *consistent state*, in relation to the constraints possibly defined by the MEE model(s), at all times.

Knowledge Graph Engine (KGE) The component responsible for managing the KG of the DT, including the links to the other DTs, as tracked by the relationships attribute. Also, it is in charge of serving

Cache & Storage (CS) The component providing to the DT those basic functionalities related to storage and caching of data, for instance regarding state updates, KG updates, events caching, and so on.

Management Interface (MI) The set of functions a DT exposes to other DTs, the WoDT platform, and external entities such as platform/administration tools and services, too. Queries about the lifecycle state of a DT, and requests for linking to and creation of other DTs are all expected to be served through the MI.

Digital Adapter (DA) The component complementary to the PAA, that is, in charge of translating events e_I into *DT events* e_{DT}, namely, events generated by the DT towards some external entity in response to invocation of some MI function—e.g. the *track* operation to be notified upon changing of DT (PA) properties and the generation of observable events.

Augmentation Engine (AE) The module that allows the DT to extend its original capabilities (inherited from the PA with the shadowing process) through the activation and execution of one or multiple functional modules. Each AE module can operate with both e_{DT} and e_{PA} (e.g. exposing a prediction action and making available predicted values of a property) according to the implemented augmentation function, and enables the definition of an additional set of properties, relationships, actions, and events exposed to the digital world and accessible by external applications and services.

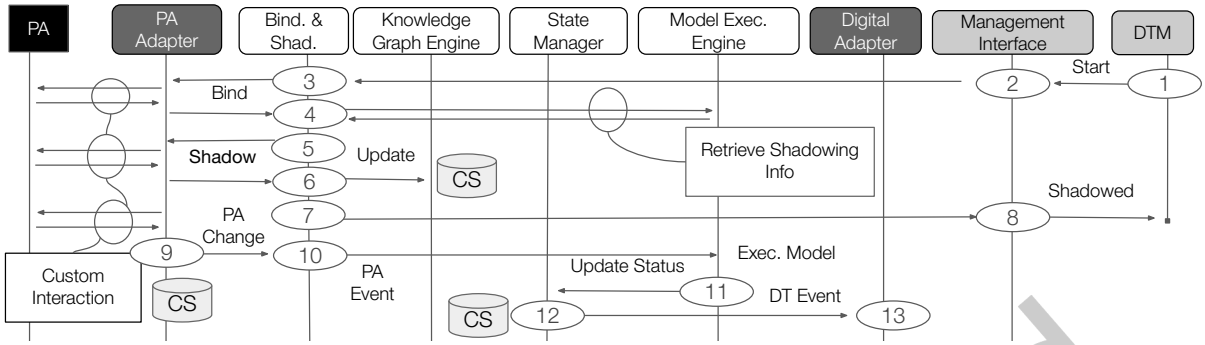


Fig. 7. DT binding and shadowing, and status update.

Then we can consider the outside of a DT, that is, the components providing those functions that cannot be delivered by an individual DT alone but should be supported by platform modules, or that may be anyway practically convenient to have independently of individual DTs. The **Distributed Knowledge Graph Engine (DKGE)** is in charge of providing the means to navigate the entire "web of DTs", that is, the labelled multi-graph where all the linking relationships between DTs are tracked, without necessarily having prior knowledge about the DTs already in the system. By doing so, the DKGE grants access to any of the DT of the platform, and to any of its properties, events e_I thread, and so on – namely, to the whole DT tuple as seen in the model (Section 3)—by routing and forwarding requests for data access to the DTs involved in any given query. The **DT Manager (DTM)** is responsible instead to manage the DT lifecycle, from creation, and hence unique ID and discoverable address generation, to DT disposal. The component also offers typical lookup services such as white and yellow pages to lookup DTs based on ID or specific properties. The **Communication Layer (CL)** enables interaction with external entities, whatever they are. Such a component represents the entry point to the WoDT platform, by providing the API to interact with the DKGE, the DTF, and the DTs themselves, through the DA.

It is worth clarifying that the architecture just presented is purely *logical*, in the sense that we do not pose any restriction on *(i)* implementation of its components, e.g. whether the DT Factory is a centralised registry or a distributed hashtable, or whether the EE is fragmented in each DT, a globally available event bus, or a combination thereof, and *(ii)* deployment of such components, e.g. whether the DT resides on the same host as the PA or even the PAA, or where a DT should be placed across the Edge-Fog-Cloud spectrum. The reason for not doing so is that such choices likely depend on many factors, such as *(i)* the target application domain, *(ii)* the available computational resources and communication infrastructure, and *(iii)* the preferred technologies to actually implement the components. Also, some of the components may be dynamically moved, or replicated, or fragmented (akin to "sharding" for database technology) at *run-time*, thus imposing design-time restrictions seems unnecessary and potentially limiting.

4.2 Interaction Flows

With the aim of clarifying the relationships between the DT's architectural components we identified in the previous section, and provide further insights on DTs functioning with respect to external applications, we here comment a few selected sequence diagrams explaining some of the main operational phases of a DT.

Figure 7 reports the internal DT operations related to PA's binding and shadowing. Interactions unfold as follows. *(1)* an external application (e.g. an agent) interacts with the DTM through the CL to create a new DT The DTM then starts the DT, through the MI module, to shadow a target PA. *(2)* The MI initiates the procedure by requesting to the BSM to perform coupling with the PA. *(3)* The BSM then interacts with the PAA to perform

binding according to PA's nature, communication protocols, and data formats. *(4)* The BSM also interacts with the MEE to retrieve the shadowing information as a function of the DT's model and the received PA's state. For example, a PA can expose multiple properties but only a subset will be shadowed through the defined DT's model. *(5)* The BSM keeps interacting with the PAA to complete the shadowing process, and finally *(6)* updates the KG to keep track of the DT's local view in terms of linking, which in turn uses the CS module to store the update and, if required, update its thread. *(7,8)* Notification of completion of the shadowing process is forwarded to the MI and the DTM. The remaining steps in Figure 7 describe how the DT handles state changes coming from the PA: *(9)* the PA generates an event associated with an internal change (or the PAA detects a variation on the PA), hence the PAA notifies the BSM about the change and tracks it through the CS module; *(10)* the BSM generates a new e_{PA}; *(11)* the MEE executes the DT's model according to the e_{PA}, the current state, and any other relevant information in the DT. Once the new state is correctly computed the MEE notifies the SM about the variation. Finally, *(12,13)* the SM validates the state, updates the CS, and possibly generates a new e_{DT} for the DA—possibly delivered to external applications tracking the DT.

Figure 8 depicts how the query and track operations of the interaction model described in Section 3.4 are carried out by the DT. In particular, in the case of a query operation *(1)*, the DA takes care of transforming such a request into a DT event *(2)*, which is then forwarded to the KGE through the MEE *(3)*. Once the query request reaches the KGE, it is its sole responsibility to appropriately forward the query to the (possibly) linked DTs (not shown). Then, once the query results are available, they are propagated back to the requesting application *(4-6)*. In the case of tracking *(7)*, the flow of interactions stops at the MEE *(9)*, that simply keeps track that a new observer should be associated with the updates coming from the shadowed PA. In fact, steps *(10-14)* are akin to steps *(9-13)* already described for Figure 7, representing the shadowing process. What's new here, is step *(12)* in which the PA state change is notified to the tracking application.

Finally, Figure 9 illustrates the internal DT operations necessary to trigger a specific action on the PA as requested by an application. If the action involves also a status change on the PA, a new e_{PA} will be generated to notify the variation. In particular: *(1,2)* the application acts on the DT to, e.g., modify the status of, or trigger an action on, the PA through the DA; *(3)* the MEE analyses the action event, applies the DT's model, and triggers synchronisation with the BSM; *(4)* the BSM uses the PAA to forward to the PA the action; *(5)* the BSM generates an e_{PA} associated to the action; *(6,7)* the MEE analyses the event, applies the model, and then sends a new e_{DT} for action completion to the DA. Then, if the action on the PA causes a status change, the PAA notifies the modification with the same interactions already seen in previous figures.

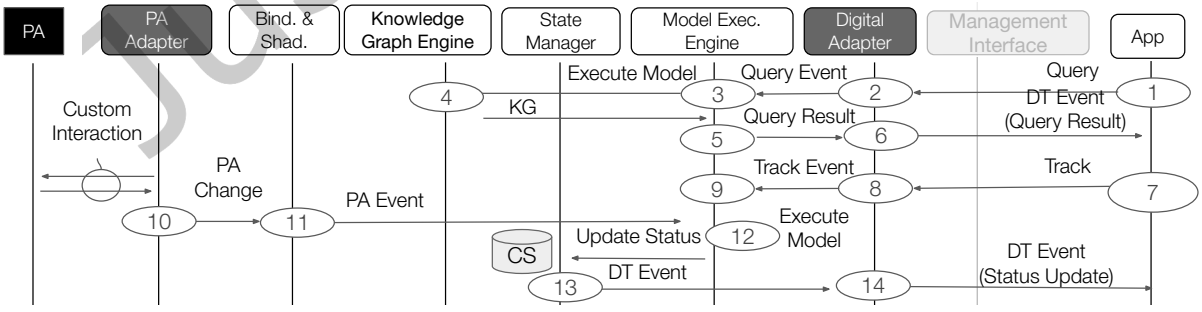


Fig. 8. Interaction of an external application with the DT: query and track.

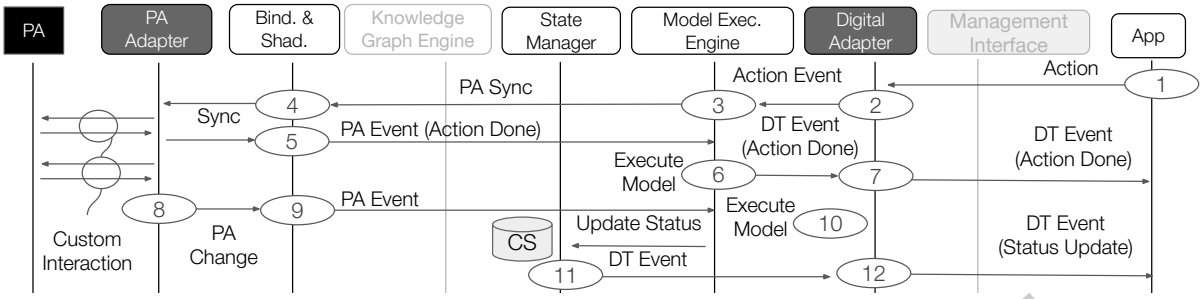
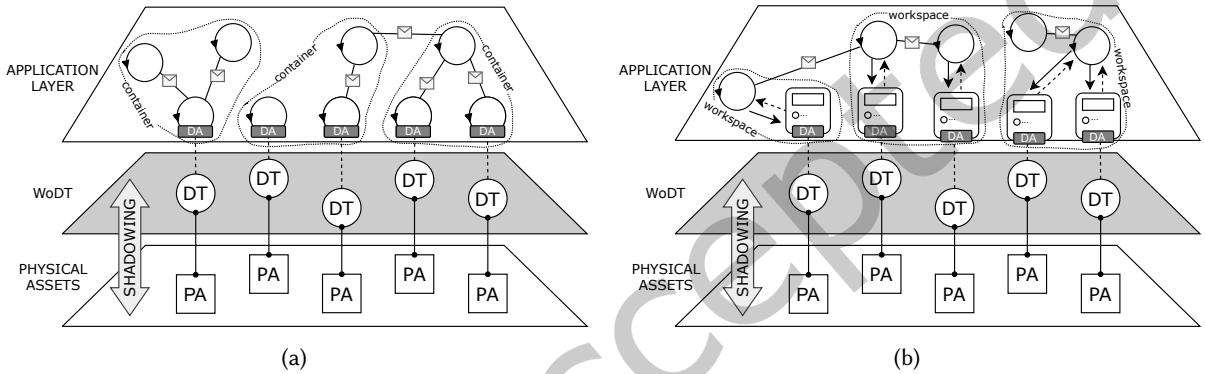


Fig. 9. Interaction of an external application with the DT: action on the PA.



(a) (b)

Fig. 10. Examples of integration approaches using JADE (a) and JaCaMo (b) platforms. For the former, agents wrap DAs to mediate interaction with the DTs; for the latter, DAs become artifacts.

4.3 Integration with Agent-based Architectures and Platforms

As mentioned in Section 3.1, agent-based approaches are a main reference for modelling and engineering smart applications and intelligent situated systems running on top of WoDT. In an agent-based view, a WoDT defines a virtual environment where agents are logically situated, exploiting the WoDT interaction model to perceive and act upon the PAs through the DTs. On the agent side, the integration with such a virtual environment can be designed using two main conceptually different approaches:

- The first one is based on *agentification* of the virtual environment, that is: every DT is represented inside the MAS by an agent functioning as its representative (or proxy), and this kind of agents provide an interface based on the Agent Communication Language (ACL) – e.g. FIPA ACL, based on speech acts – for the other agents of the MAS to interact with the DT. A concrete example of this approach is shown in Figure 10a, based on JADE [4], a well known FIPA compliant platform [5]. In this case, the agent representing the agentified DT would encapsulate and hide the DA machinery to interact with the DT.
- The second one is based on modelling the WoDT virtual environment in terms of first-class environment abstractions on the agent side. For instance, in the A&A metamodel [34], each DT could be represented as an artifact, being *artifacts* the basic entities used to modularise computational environments in A&A, organised in workspaces. In this case, agents would interact with DTs perceiving their observable states, events and performing actions by interacting with artifacts, i.e. perceiving/observing their observable

properties (mapping the DT state) and executing operations (representing actions). Figure 10b shows a concrete example based on JaCaMo [6], a MAS platform that supports artifact-based environments and BDI agents, that can be programmed in Jason.

These two simple approaches are useful just to enable the integration at a technical/platform level between agents and DTs. A deeper form of integration can be explored by considering that a WoDT is possibly a *distributed hypermedia-based environment*, being based on an open distributed dynamic KG, which can be represented in terms of Semantic Web technologies. In the literature, research on Hypermedia-based MAS [8] is exactly about agents that are situated in a distributed hypermedia environment that they can navigate and use in pursuit of their goals. Accordingly, a further way to understand and explore the design of agent-based applications running on a WoDT is to view them as a special kind of Hypermedia based MASs, where the distributed hypermedia environments in this case are meant to be virtualisations of physical assets.

Finally, architectures for designing intelligent agents – such as the BDI one – can be a relevant reference not only for designing intelligent systems at the application level, but also at the Digital Twin Layer, as constituting elements of Cognitive Digital Twins. As a specific example, a CDT based on the BDI architecture can exploit the sense-plan-act reasoning cycle to realise the shadowing process of a DT as well as the augmentation counterpart: through the event-driven sensing, changes in the PA are mapped into beliefs, that can trigger the execution of reactive plans realising the augmented behaviour, including pro-active tasks toward the achievement of goals as defined by the stage 4 DT vision [43]. Vice versa, requests for action coming from applications may be encapsulated in messages sent to the agent-based DT, that through appropriate plans triggers the needed actions on its associated PA. Indeed the opportunities about exploiting intelligent agent architectures like BDI to design CDTs are manifold, and although some early research activities are already started on the topic, much more will be needed to comprehensively understand the extent and limits of agent-based DT modelling and engineering.

5 APPLYING WODT TO REAL WORLD CASE STUDIES

The vision described in Section 3 has been devised by generalising our experience in the design of real-world systems in specific domains, namely *healthcare* and *smart city*. In this section, we briefly introduce these cases and discuss their modelling using the WoDT vision.

5.1 The Case of Major Trauma Management

Major Trauma Management is one of the most challenging scenarios where physicians can be involved in the healthcare context. Like other time-dependent pathologies, major traumas ask for a team of physicians with strong heterogeneous expertise (called trauma team), to promptly identify a diagnosis and quickly provide medical aid. In fact, patient health outcomes strongly depend on the first hour of treatment. In broad terms, the whole trauma management process can be conceptually split into three main stages:

Stage #1 – Emergency Call Management Following an emergency call to the Central Emergency Unit (CEU), an operator collects information about the occurred event then plans and starts a first-aid emergency mission, involving a particular rescuer and a specific vehicle;

Stage #2 – Pre-Hospital Management The rescuer reaches the patient with the aim of administering him/her first aid basic life support, deciding the severity of the trauma and, finally, transferring the patient to the trauma centre;

Stage #3 – Trauma Management At the emergency room of the trauma centre, the patient is taken in charge by a team of expert physicians called trauma team, led by its trauma leader, with the aim to do everything is required to save patient's life according to the severity of the occurred trauma.

Besides procedures that physicians have to accomplish to save patient's life, this process requires in every stage some collateral activities in order to *(i)* document the overall evolution of the ongoing trauma – e.g., time tracks

of procedures implementation and drugs administration, diagnostics results, and so on – and, *(ii)* have continuous monitoring of the real-time evolving state of the trauma process—in particular, of the patient and other assets, including the trauma teams members.

To support this scenario, and in particular these two latter collateral needs, a research project called TraumaTracker [10, 29] has been carried out in cooperation with an Italian Trauma Centre[13] since 2017. Briefly, TraumaTracker has been designed and developed to support the trauma team at Stage #3 of the major trauma management process. In particular, it acts as a personal assistant agent of the trauma leader to produce the trauma documentation of the in-hospital stage and monitor the evolution of the ongoing medical procedures, possibly producing alerts for the trauma leader. The TraumaTracker prototype is currently being used: to date, over 1600 trauma reports have been collected. Since its first release, TraumaTracker has been constantly refactored and updated according to a domain-driven design process. Recently it has also been extended to Stages #1 and #2, refactoring its design towards a DT-oriented architecture [9, 39].

In this paper, we demonstrate how the mission-critical scenario of major trauma management can be designed according to the WoDT approach (and model). Table 1 describes the details of a major trauma management scenario, considering its evolution both in the physical and the digital worlds. To provide a better comprehension of how the WoDT can be designed to support the scenario of this case study, in Figure 11 a graphical notation is used to model relations among involved DTs and related PAs. This figure represents a kind of "architectural view" of the relations among the PAs composing the scenario and the DTs modelling them. Moving from the top to the bottom of this figure, for each stage a snapshot of DTs in execution in that stage is reported. In particular, some of them are conveniently created with the purpose of shadowing emerging PAs in the evolution of the scenario (e.g., the `MissionDT` at Stage #1, the `TraumaTeamDT` at Stage #2 or the `ShockRoomDT` at Stage #3). Other DTs, instead, are in execution regardless of the specific stage and scenario (e.g., the `CentralEmergencyUnitDT` at Stage #1 or the `HospitalDT` at Stage #3) because they are part of the broader WoDT of the whole Local Health Department. In other words, these latter DTs have been previously created in the context of other DTs, and they are in continuous execution to support heterogeneous scenarios beyond the major trauma management one.

It is worth noting that some of them – e.g. the `PhysicianDT` and `RescuerDT` at the Stage #1 – can be coupled to the same PA. This means that both DTs represent the same PA with different levels of specialisation. In this case, the rationale behind this design choice is given by the specific domain, that is: a physician working at the local health department has its own DT created by the time when he/she was hired, representing his/her digital counterpart as individual (the `PhysicianDT`); vice versa, when the physician acts as a rescuer in the specific context of a rescue mission he/she has to be coupled to a dedicated DT (the `RescuerDT`) conceived as a specialisation of the previous DT having specific properties (e.g., the identifier of the rescuer for that specific mission) according to the role played at that moment. This latter DT's life span is limited to the duration of the mission in which he/she is involved.

Software agents are not explicitly represented in Figure 11, to avoid cluttering. Nevertheless, they are the proactive actors observing and acting upon DTs. For instance, the personal assistant agent of the trauma leader – as identified in the TraumaTracker system – is an agent which comes into play mainly at Stage #3 of this scenario, when the trauma leader starts to coordinate the trauma team in performing medical procedures to save patient's life. In particular, considering, for instance the aim to producing relevant alerts related to the ongoing trauma, such personal agent observes all the DTs shadowing the in-hospital macro phase (`OngoingTraumaDT`, `VitalSignsMonitorDT`, `ShockRoomDT`, ...) and, potentially, it exploits the `DisplayDT` API to show the alert (e.g., about the fact that the patient heart rate is decreasing rapidly).

Figure 12 shows instead (a portion of) the KG of the WoDT and how it evolves, from stage to stage, according to the evolution of the case study as described in Table 1. The KG is represented in RDF using the Turtle notation

[13]The "M. Bufalini" Hospital Trauma Centre, AUSL della Romagna, Cesena, Italy.

Table 1. The evolution of a Major Trauma Management Scenario both in the physical and in the digital world.

	Physical World	**Digital World**
Stage #1	An emergency call is taken by the unique health number, and the *Central Emergency Unit* (CEU) operator collects the first-contact information of the occurred *event* (e.g., the place of the event, its status, the number of involved people, and their rough health status). Then, for each involved victim, a new *mission* is started with a specific *vehicle* (e.g., an ambulance) and a particular *physician* qualified to act as *rescuer*.	A new DT for the event (`EventDT`) is created in the context of the running DT coupled to the CEU. This new DT includes the information collected by the operator about the event. According to the rescue process, also a DT for each mission (`MissionDT`) is instantiated and linked to DTs of both the vehicle (`AmbulanceDT`) and the rescuer (`RescuerDT`), dynamically discovered exploiting the CEU DT.
Stage #2	The emergency crew arrives at the event place and interact with the *patient*, possibly identified as a qualified *healthcare user* with his health insurance card. Here, the rescuer giving the first-aid evaluates the patient medical condition to establish a *diagnosis* (a major/severe trauma, in this example). Accordingly, a destination for the patient is decided (in this case, the *emergency department* of the nearest *hospital* acting as *trauma centre*). So, the patient is moved to the destination by the emergency crew and, in the meanwhile, at the notified trauma centre a new *trauma team* (led by its *trauma leader*, typically an anaesthetist-resuscitator) is dynamically composed and informed about the incoming patient health conditions.	A new DT for the patient (`PatientDT`) tracking the triage data collected by the rescuer is created and linked to the `MissionDT`. In the case that the patient is properly identified (using his/her health id), the corresponding DT (`HealthcareUserDT`) is discovered and linked by the `PatientDT`. When the diagnosis is a major trauma, a new DT for the trauma management process (`OngoingTraumaDT`) is created in the context of the DT of the selected trauma centre (`TraumaCentreDT`), and it is linked to the DT of the patient, to start to collect information of the incoming patient. Finally, the DTs of the physician of the trauma team (`TraumaTeamDT` and `TraumaLeaderDT`) are properly created and linked.
Stage #3	The emergency crew arrives at the emergency department and entrusts the *patient* to the *trauma team*. So, the mission of the emergency crew ends as well as the involvement of the rescuer. The trauma management in-hospital process starts, possibly involving multiple *rooms* and *facilities* of the hospital. For instance, the main room where the trauma is managed, called *shock-room*, is equipped with adequate facilities to support physician's work, among them most relevant are a *display* to refer to tracked information and diagnostics' results dynamically, and the *vital signs monitor* to collect and observe the patient's vital signs trace. Other rooms of the trauma path are, e.g., the computer-aided tomography (CT) room or a dedicated operating room. Finally, when the trauma management process ends, the patient is generally hospitalised according to his/her new health condition, e.g. into the intensive-care unit.	The `OngoingTraumaDT` tracks all the relevant events happening during the trauma management. In a sense, this DT replaces the `PatientDT` in the in-hospital phase: no more updates are reported to this latter DT. The `OngoingTraumaDT` tracks the current room where the patient (and trauma team) are, by linking the corresponding DT (e.g. the `ShockRoomDT`). The DT of the current room provides the links to the DTs of the physical facilities in the room (e.g., the `DisplayDT` and the `VitalSignsMonitorDT`). These facilities can be exploited by, e.g., the personal assistant agent of the trauma team/leader, implementing context-aware support. When the trauma management ends, the `OngoingTraumaDT` is no longer updated, and a DT referring to the hospitalised patient is created by the DT of the designed hospital ward. Likewise, the DTs of the trauma team and trauma leader ends their work.

and is built according to the specific domain glossary of terms. For instance, the `:ambulanceA` instance at the Stage #1 is represented by two properties (`:position` and `:status`) and it is related to the `:CEU` instance, with the `:from` relation. Note that the `:ambulanceA` is also indirectly related to the `:missionM` concept because this latter has a relation `:vehicle` toward `:ambulanceA`. An agent which would like to track the position of an ambulance of a specific mission in the context of a specific event, could observe changes of the `:position` propriety of the `:ambulanceA` instance in the graph.

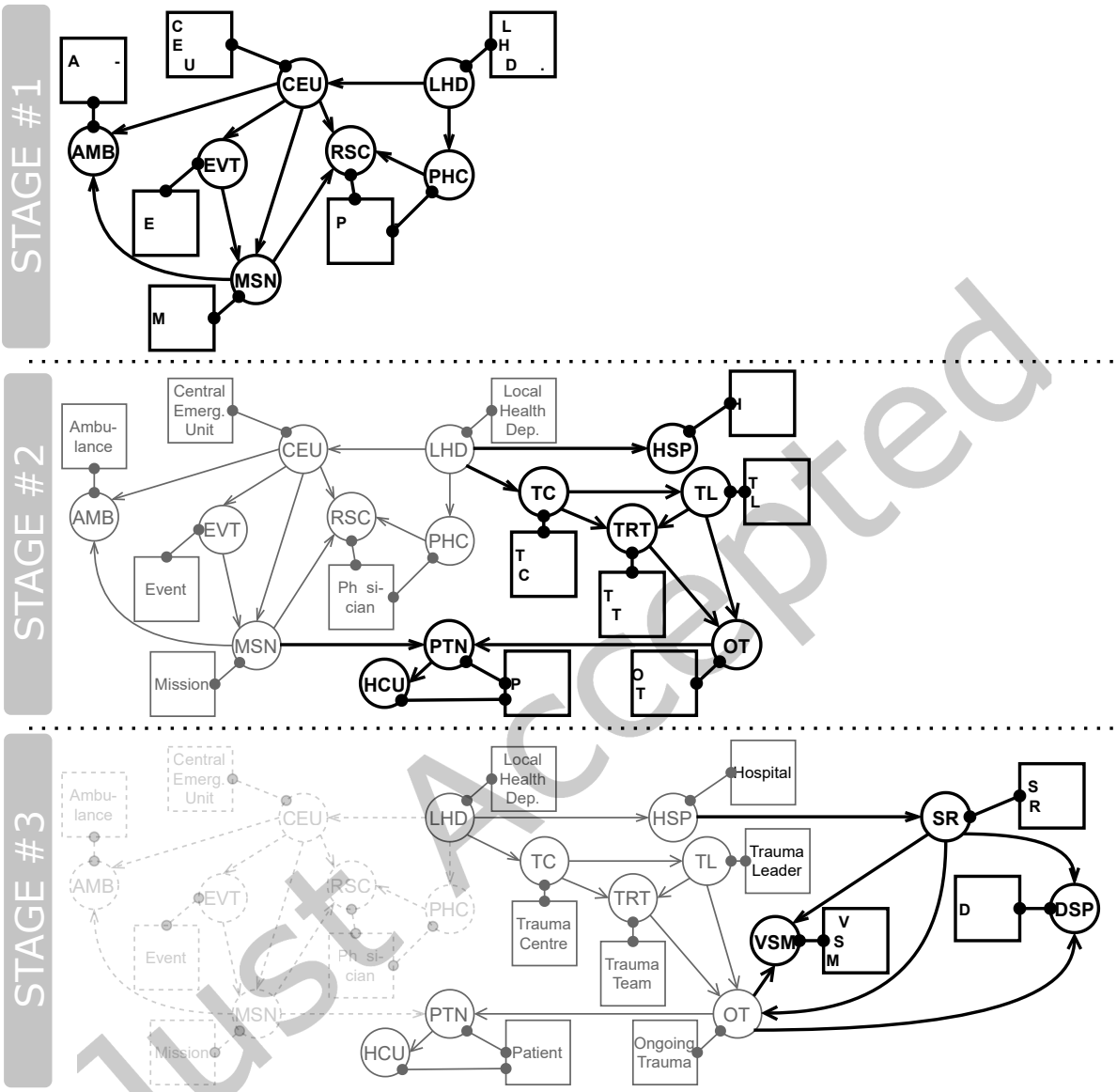


Fig. 11. The evolution of a Major Trauma Management Scenario in terms of relations among DTs and PAs. Entities in bold represent new additions to the KG, entities in grey represent old entities but still active, faded entities represent currently inactive entities.

To further support agent reasoning, a KG may include not only ABox assertions, i.e. facts associated with the actual state and situation of the PAs mirrored by the WoDT, but also TBox statements, about classes and properties of the ontologies [14]. As an example, Figure 13a shows a refinement of the KG at Stage #1 in which the subject of each RDF triple has been qualified, introducing a proper domain-oriented parent concept to enhance the

semantics and the reasoning upon it. For instance, we added the information about the fact that the `:ambulanceA` is a `:Vehicle` (note that in the Turtle syntax the keyword "a" can be used to express an `:is-a` relationship).

When considering concrete real-world domains, the knowledge graph of a WoDT may profitably refer to existing standard ontologies available for those domains. A main example in the healthcare context is given

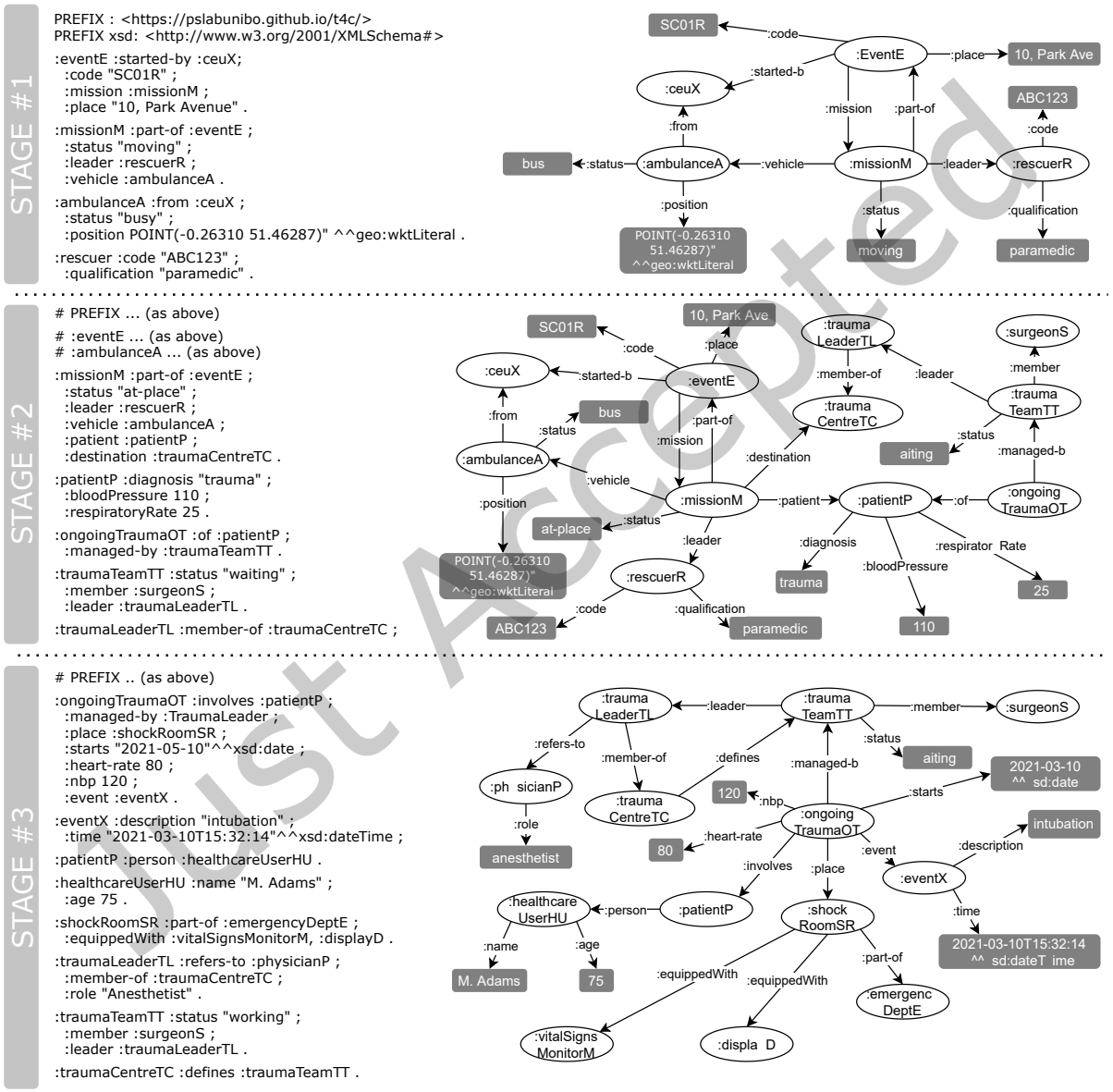


Fig. 12. The Knowledge Graphs related to the Major Trauma Management Scenario Evolution.

```
PREFIX : <https://pslabunibo.github.io/t4c/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

:eventE a :Accident ;
  :started-by :ceuX ;
  :code "SC01R" ;
  :mission :missionM ;
  :place "10, Park Avenue" ;
  :victims 1 .

:missionM a :FirstAidAction ;
  :part-of :eventE ;
  :status "moving" ;
  :leader :rescuerR ;
  :vehicle :ambulanceA .

:ambulanceA a :Vehicle ;
  :from :ceuX ;
  :status "busy" ;
  :position "POINT(-0.263 51.462)"^^geo:wktLiteral .

:rescuerR a :Rescuer ;
  :code "ABC123" .

:rescuerR a :TraumaCentreEmployee ;
  :name "J. Smith" ;
  :qualification "paramedic" .
```

(a) Generic RDF description with unspecified domain-specific ontology.

```
PREFIX : <https://pslabunibo.github.io/t4c/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX fhir: <https://www.hl7.org/fhir#>

:eventE a fhir:Encounter;
  fhir:identifier "SC01R" ;
  fhir:location :eventLocationEL ;
  fhir:status "active" ;
  :victims 1 .

:eventLocationEL a fhir:Location ;
  fhir:address "10, Park Avenue" .

:missionM a fhir:Encounter ;
  fhir:partOf :eventE ;
  fhir:status "moving" ;
  fhir:location :ambulanceA ;
  fhir:participant :rescuerR .

:ambulanceA a fhir:Location ;
  fhir:status "active" ;
  fhir:position "POINT(-0.263 51.462)"^^geo:wktLiteral .

:rescuerR a fhir:Practitioner ;
  fhir:identifier "ABC123" ;
  fhir:name "J. Smith" ;
  fhir:qualification "paramedic" .
```

(b) RDF description according to the domain-specific FHIR ontology.

Fig. 13. Refinements to the semantics of the Major Trauma Management Scenario Stage #1.

```
SELECT ?t ?v1 ?v2
WHERE {
   ?t a :OngoingTrauma .
   ?p a :Patient .
   ?t :involves ?p .
}
```

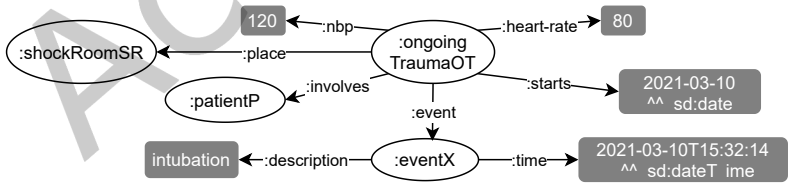


Fig. 14. An Example of a SPARQL Query on the KG.

by FHIR[14], the standard for healthcare data exchange. Figure 13b shows a refinement of the representation of the WoDT at Stage #1, in which domain concepts have been rearranged according to the FHIR ontology. For instance, FHIR uses the concept `fhir:Location` to identify all the heterogeneous set of healthcare locations (e.g., buildings, rooms, streets, vehicles): for this reason, both `:ambulanceA` and `:eventLocationEL` are now instances of the `fhir:Location` concept. Moreover, both `:eventE` and `:missionM` must be qualified as `fhir:Encounter` instances, although they have a very different meaning in the specific major trauma management scenario. Nonetheless, in FHIR every non-planned occurring event, involving practitioners and patients must be defined as encounters, with specific properties as, i.e., `fhir:participant`, `fhir:status`, `fhir:identifier` as shown in this example.

[14]https://www.hl7.org/fhir/

Finally, Figure 14 reports an example of a simplified SPARQL Query performed on a portion of the KG of the Stage #3 related to the described Trauma Management scenario.

5.2 The Case of Mobility Intelligence

In a future cooperative driving scenario with both autonomous and non-autonomous vehicles sharing the road infrastructure, basic services such as intersection crossing, parking, and ride-sharing will need to be re-designed [26]. A WoDT can play the role of the enabling coordination infrastructure: *(i)* each vehicle has its own DT created and bound when the vehicle is registered in the municipality's dedicated registry; *(ii)* each citizen may also have her own DT, whose creation and binding could happen at birth upon registration in public health registries; *(iii)* each relevant road infrastructure element – such as Road Side Units (RSUs) working as intersection managers, parking lots managers, etc. – is also digitally represented as a DT; *(iv)* some of these, such as those representing RSUs, are statically connected at design-time to provide application developers with the means to shape the computational environment of intelligent mobility and traffic management applications; *(v)* some others, such as those representing vehicles, become connected dynamically, at run-time, based on users' adoption and the applications' needs. Several use cases may be described with the WoDT vision we defined in Section 3. Here we focus on intersection crossing as the most challenging urban task for both autonomous and non-autonomous vehicles, but other application scenarios may as well target smart parking, ride-sharing, and overall traffic flow management.

Currently, most intersections are regulated by right of way signals placed on the road, traffic lights, or roundabouts. These regulations means are suitable for human-driven vehicles but largely inadequate (e.g. sub-optimal) for autonomous ones, which could leverage cooperative driving to cross intersections more efficiently. Literature about autonomous intersection crossing is abundant and features many different approaches, such as reservation-based, negotiation-based, distributed constrained optimisation, solutions based on game-theoretic approaches, etc. [26]. Common to all approaches is the assumption that either an intersection manager is available, as the computational component of the intersection road infrastructure in charge of coordinating vehicles, or that vehicles are able to communicate with each other and reach an agreement about the crossing order in a fully decentralised way. In Table 2, we take as a reference a reservation-based approach to intersection crossing, one of the most successful and studied approaches [12], and describe the relationships between what happens in the physical world and what happens in the digital world, that is, in the WoDT representing the domain (the problem as well as the solution).

There, it is worth noting that the signalling operation mentioned in stage "incoming" can be realised according to two approaches: the one described in Table 2 needs the `Vehicle` DT to be *pro-active*, as it is the one who informs the intersection about its intention to cross; an alternative would be to let the `Intersection` DT devise out the `Vehicle` DTs intentions through *observation* (e.g. if both turning lights are off, the vehicle is going straight). Preference of either approach is a design choice whose discussion is out of scope here.

The many links established throughout the scenario lifespan are depicted in Figure 15, which shows the temporal evolution (from top to bottom) of the knowledge graph of the WoDT, assumed to be defined by RDF in Turtle notation, exploiting the SAREF4Auto ontology[15] currently defined by the ETSI [16]. In stage "setup" the basic road infrastructure is setup, and vehicles registered to the municipality are bound to their DT. Links in this stage are rather static, as they resemble (mostly) persistent relationships. In stage "incoming" the dynamic links tracking the status of the intersection and of the crossing process begin to be established towards all vehicles approaching the intersection area (e.g. within a 100m radius). In stage "outgoing" further links are established to track the highly dynamic process of intersection crossing. For instance, links resembling waiting queues are

[15]https://forge.etsi.org/rep/SAREF/saref4auto
[16]https://www.etsi.org

Table 2. Description of the intersection crossing scenario using the Web of DTs model.

	Physical World	**Digital World**
Stage "setup"	The municipality deploys the "intelligent mobility" platform on a target intersection. Relevant RSUs are deployed, such as smart cameras to monitor traffic conditions and a computational node (e.g. a RasberryPi) to govern the crossing process according to a policy given by the municipality. Vehicles are registered to the municipality before hitting the streets, and are supposed to be equipped with a suitable hardware & software stack enabling at least identification.	The `Intersection` DT is created and bound, representing the status of the intersection area. The `CrossingProcess` DT is created and bound, representing the policy of the crossing process and its status. A `:deployed` link is established amongst the two, to represent the fact that the intersection is currently enforcing the given crossing policy on approaching vehicles. In case of a fully autonomous vehicle, the `Vehicle` DT is created and bound, representing the vehicle status and behaviour. In case of a non fully autonomous vehicle, the `Driver` DT is also created and bound, and linked to the `Vehicle` to represent the connection between a vehicle and its driver.
Stage "incoming"	A vehicle approaching the intersection is detected by some RSU (e.g. a smart camera with a given detection radius). The vehicle somehow signals the intention to cross the intersection, e.g. by activating the turning lights in case of a non autonomous vehicle, or by communicating with the intersection manager over wireless networks in case of an autonomous vehicle attempting to reserve a spatio-temporal slot for occupying the intersection area [12].	A `:crossing` link is established between the `Vehicle` DT and the `Intersection` DT, to track the presence of the vehicle within the intersection area. A `:managing` link is established between the `CrossingProcess` DT and the `Vehicle` DT, to track the fact that the vehicle is now being managed by the intersection policy. The `Vehicle` DT signals the intention to cross to the `Intersection` DT, e.g. by raising an appropriate event e_{DT}, or by exploiting the dedicated service interface on the `Intersection` DT. The `CrossingProcess` DT, by observing the `Intersection` DT, becomes aware of the crossing request and includes the `Vehicle` DT in the coordination process aimed at distributing right of ways.
Stage "outgoing"	The intersection manager checks new requests for crossing against pending ones given the current crossing state, and decides which vehicles get the right of way, and which spatio-temporal constraints they should abide to while crossing. The vehicle eventually gets its right of way, then can safely cross the intersection.	The `CrossingProcess` DT establishes `:assigned` and `:waiting` links to `Vehicle` DTs already having a `:managing` link with it, representing the crossing status of the vehicle—respectively: right of way given, or not. The `Intersection` DT establishes `:leading` links with `Vehicle` DTs leading a queue of vehicles—that is, vehicles with a `:waiting` link and an incoming `:queuing` link. The `Vehicle` DTs establish `:queuing` links with the preceding vehicle, if any, tracking the fact that to get the right of way they need the leading `VehicleDT`s to get it first. The `Vehicle` DT loses all of its links related to the `CrossingProcess` DT and the `Intersection` DT, tracking the fact that it is no longer involved in the intersection.

established, as well as links tracking the crossing status of a vehicle, where `:assigned` means that the vehicle got its right of way, whereas `:waiting` the opposite. It is worth noting that *orientation* of links highly depend on their *semantic*: for instance, we decided to let the `:queueing` link go from the queued vehicle to the leading one, but the opposite could be meaningful as well. However, it is crucial to keep in mind that orientation of links has an impact on applications, as which links a DT can participate to as the *subject* of the relationship should be known at *design-time*, whereas those involving the DT as the *object* can be unknown before *run-time*.

To conclude, we emphasise that on top of the WoDT infrastructure, we can envision a wide array of applications. For instance, an application may continuously monitor the knowledge graph linking together all the different intersections of an urban area, e.g. based on ownership of the municipality, to provide to city governance a dashboard with a map charting the traffic flow. The governance can then detect bottlenecks and, for instance,

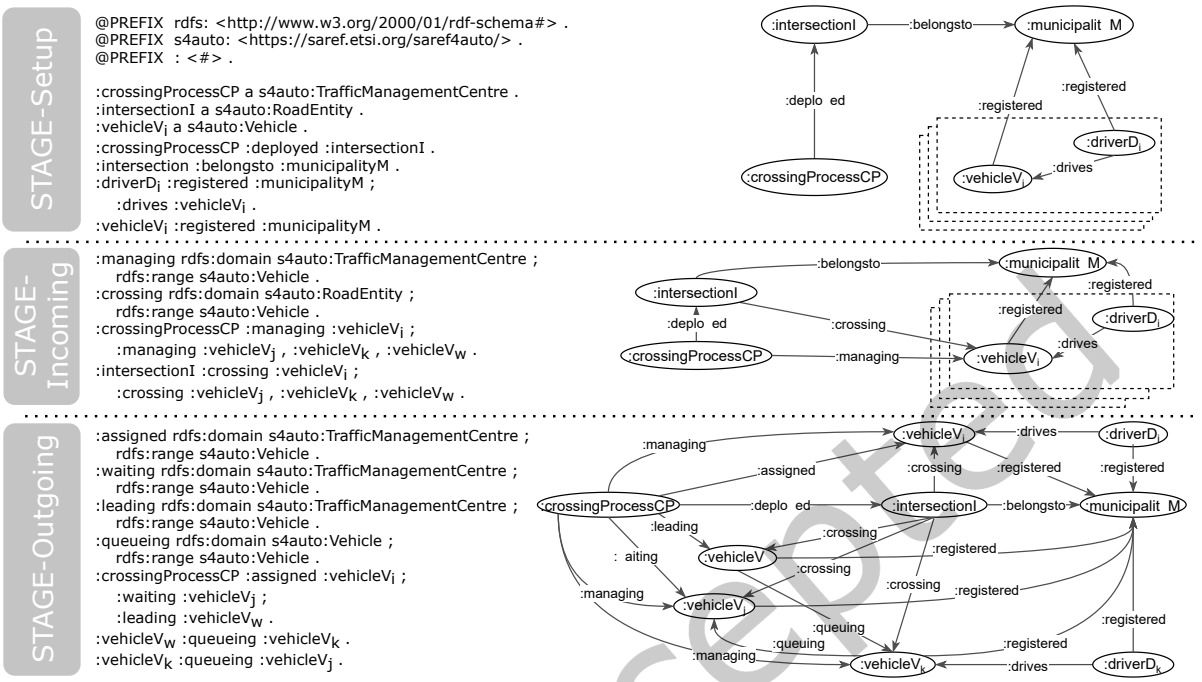


Fig. 15. Evolution of the knowledge graph in the mobility intelligence scenario.

change the crossing policy of selected intersections. A similar application may be given to drivers, or integrated with vehicles' navigation systems, so that they can always choose the least congested route towards their destination. Another kind of application instead may inform drivers about the crossing policy of intersections along their route, so that they can decide which to avoid (e.g. auction-based policies to avoid spending "road credits" to cross). More complex applications, such as for gaining actionable knowledge through traffic flow prediction, may be engineered as well on the common, homogeneous, interoperable substrate provided by the WoDT, depending on the augmentation capabilities of the DT, as discussed in Section 6.

6 RESEARCH DIRECTIONS

The vision and model proposed in this paper, as well as its application in the real world, introduce potential opportunities and open issues that should drive future research and implementation activities, possibly in different areas. In this section, we provide an overview of a selection of them.

Realising Interoperable WoDTs. In this paper, we described the main concepts of WoDT using an abstract conceptual framework and architecture. The possibility to apply and implement it by preserving (cross-domain) interoperability is bound to the definition of shared concrete meta-models and languages. For instance, in the case of WoT, standardised metadata and other re-usable technological building blocks – such as the Thing Description – have been defined by the W3C WoT Working Group, to ease the integration across IoT platforms and application domains. Analogously, standardised metadata and technological building blocks can be devised as well for WoDT, possibly layered upon enabling ones, such as Semantic Web and the ones defined by WoT. Main references

to be considered for exploring this direction include both standardisation initiatives (such as the Digital Twin Consortium[17]), large scale projects (such as the Digital Twin Programme[18] promoted by the CDBB), but also concrete languages and technologies available from specific companies. A main example for this is the digital twin definition language (DTDL) by Microsoft. These technologies can be used to develop specific incarnations of the WoDT model, for instance one specific to the WoT domain exploiting standard Web technologies. Indeed, we deliberately described our reference architecture in abstract terms, without specific constraints on design paradigm or technologies (except for event-driveness), exactly to let the interested communities – be it the (Semantic) Web community, the MAS community, etc. – develop their own technical solutions, depending on reference applications requirements and available technologies.

Design and Implementation of WoDT Middleware and Tools. Recent years clearly showed how the lack of standards or common agreements for DTs design and development has led to the proliferation of several isolated platforms and domain specific solutions. This trend is also emphasised in [50] where the authors highlight how the real DTs' potentials are seriously limited by the existing fragmentation and heterogeneity. Each existing approach or platform is built from scratch with a siloed centralised vision instead of a shared set of methodologies, models, and interaction patterns. The WoDT contribution aims to overcome these limitations by creating an interoperable vision where DTs can seamlessly cooperate within the same application domain and across multiple domains at the same time. As previously illustrated, the WoDT does not impose any implementation specifications or constraints, rather, it aims to operate at a higher layer supporting multiple platforms and tools following the set of shared modelling principles and event-driven design.

An open challenge for DTs and WoDT will be at first related to the definition of open implementations for both the DT's core and the platform in order, on the one hand, to simplify the shadowing and the augmentation management and, on the other hand, to support a distributed and interoperable knowledge system and communication overlay. The natural next step will be to quickly adopt the new implementations and start developing and integrating specific modules and libraries dedicated to target PAs, domains, and use cases in order to exploit existing standards and creating a set of shared features without the need to reinvent the wheel at each deployment, and limit the risk to create siloed ecosystems. The perfect example will be the integration between WoDT with the IoT world, where the benefits of introducing an interoperable and flexible DT's layer will be strategic at different levels and for several uses cases. In that specific domain, where the fragmentation of the physical layer is a massive issue, the integration of WoDT with the standardisation efforts provided by consortia such as oneM2M and W3C WoT represent an appealing and concrete opportunity to quickly reach a standardised version of IoT DTs, capable of providing a scalable digital abstraction on top of the physical layer.

Shadowing and Certified DTs. Shadowing is a key process for DTs, being responsible of making the state of the DT a *correct* digital shadow of the PA, where the semantics of correctness is given by the model M—it may include constraints about fidelity, responsiveness, accuracy, etc. Applications exploiting DTs – especially intelligent agent-based systems reasoning upon DTs and taking autonomously decisions given the observable state of DTs – should have evidence that a DT is working (or not) as promised by its model/specification. Accordingly, a proper level of certification should be useful (or, rather, necessary) to define the quality of service expected from a DT, as part of its Service Level Agreement.

Querying and Observing Graphs of DTs. Querying and observing graphs of DTs and, correspondingly, Distributed KG is a challenging issue, given in particular the semantic constraints specified in Section 3. This issue is strongly related to existing research works in Semantic Web literature that are about querying distributed RDF data stores [37], and, more generally, to research that deals with large-scale semantic integration of linked data [31].

[17]https://www.digitaltwinconsortium.org/index.htm
[18]https://www.cdbb.cam.ac.uk/what-we-do/national-digital-twin-programme

The two aspects of Distributed KG in WoDT that further characterise the open issue is about dynamism and shadowing, so that individual KG continuously evolve, possibly with a high changing frequency, and the stream of updates from the physical world cannot be blocked or be interfered by querying and tracking.

Design of Intelligent Agents Situated in WoDT. The adoption of a semantic model based on knowledge graphs makes it particularly interesting to explore the usage of intelligent agents that adopt an explicit *knowledge level* [33] to represent and reason about their tasks, goals, and environment. In the case of BDI Agents, for instance, this translates to adopting a model for representing beliefs based on KG triples. That is: each triple is represented by a belief and then a WoDT environment observed by an agent is represented by a (dynamic) set of triples tracking the corresponding KGs, properly updated according to the evolution of the WoDT. A main reference for this research investigation is given by existing works in the literature exploring the integration of BDI Agents and Semantic Web [11] and ontology-based agents [30]. In these works, agents are equipped with basic capabilities to access and query OWL-based knowledge based on some ontologies. That knowledge is however almost static. The WoDT calls for agents capable of observing knowledge graphs that could dynamically evolve, not only in terms of values in data properties but also in terms of relationships.

Besides knowledge representation, a further main research issue concerns the opportunity to combine practical reasoning techniques – that are typically adopted on the agent side [54] – with cognitive capabilities provided by DTs, such as predictive ones. Accordingly, in order to decide the course of actions to perform to fulfil some task, the agent could consider not only the current observed state of the WoDT (in terms of knowledge graph), but exploiting the prediction/simulation functionalities provided by the DT as-as-service. This calls for exploring the design of intelligent agents (and MAS) exploiting *anticipatory capabilities* [35] to enhance the overall sense-making process and improve decision-making.

Prediction and Historical Data Analysis based on Knowledge Graphs. In the literature, distinguishing functionalities such as threading and prediction have been explored for individual DTs mirroring specific physical assets [43]. The WoDT approach broadens this view by considering a graph of linked but independent evolving DTs: a semantic model based on knowledge graphs makes it possible to explore these features in terms of the evolution of KGs and distributed KGs. A challenging aspect here is that a WoDT may involve multiple DTs based on different models M, hence abstracting away different facets of the observed reality, which in turn likely need different data being available, following different distributions—features that complicate notably the task of learning patterns that can be generalised. Existing works in the literature have explored such techniques in the case of single models or a single data stream [55]. Nevertheless, these contributions can be taken as a starting point for exploring extensions considering the integration of multiple heterogeneous models, as well as of multiple heterogeneous data sources for data-driven approaches.

7 CONCLUDING REMARKS

The WoDT is an effort to take the lessons learnt from the World Wide Web, the IoT, multiagent systems, and distributed systems, and apply them to the definition of an event-driven, decentralised, interoperable, linkable and discoverable vision of digital twins. The proposed model and abstract architecture define a basic conceptual framework, that can be mapped onto a variety of concrete deployment scenarios and implementation technologies, with the aim to be a unifying horizontal layer on top of the physical assets.

The use cases presented in Section 5 illustrated how the WoDT can be actually shaped into specific application domains with peculiar challenges and constraints related, for example, to the enrolment of heterogeneous physical assets, a structured hierarchical organisation, and dynamic evolution in terms of interactions and knowledge representation. On one hand, the WoDT allowed to model DT's properties, behaviours, and relationships, and consequently to represent large-scale and complex physical environments as an open ecosystem of connected

and interoperable DTs. On the other hand, the proposed vision supports the definition of a new cyber layer where applications, agents, and services can implement and orchestrate new smart and dynamic systems of components by relying on a structured and integrated DT's overlay, without the responsibility to handle the fragmentation and the heterogeneity characterising the physical layer.

Moving forward from the local scope of a single application domain, the possibility to exploit a uniform and interoperable Web of DTs also opens the way to the design of a new generation of cross-domain computational infrastructures, trying to mirror the physical world where existing assets seamlessly move and interact across multiple contexts at the same time. For example, a person can be an employee for a company and a patient for the health system, or an ambulance can be a vehicle on the street and a resource for the trauma management ecosystem. Through the adoption of WoDT, DTs from multiple realms can start cooperating (potentially on demand) to reach a shared goal or to opportunistically implement a new behaviour, that is something quite difficult to achieve in the siloed environments representing the state of the art.

REFERENCES

[1] Sailesh Abburu, Arne J. Berre, Michael Jacoby, Dumitru Roman, Ljiljana Stojanovic, and Nenad Stojanovic. 2020. COGNITWIN – Hybrid and Cognitive Digital Twins for the Process Industry. In *IEEE Int. Conf. on Engineering, Technology and Innovation (ICE/ITMC)*. 1–8.

[2] Ahmad Alelaimat, Aditya Ghose, and Hoa Khanh Dam. 2020. Abductive Design of BDI Agent-Based Digital Twins of Organizations. In *Proc. of the 23rd Int. Conf. on Principles and Practice of Multi-Agent Systems (PRIMA '20) (LNCS, Vol. 12568)*. Springer, 377–385.

[3] Paolo Bellavista, Carlo Giannelli, Marco Mamei, Matteo Mendula, and Marco Picone. 2021. Application-driven Network-aware Digital Twin Management in Industrial Edge Environments. *IEEE Trans. on Industrial Informatics* (2021).

[4] Fabio Bellifemine, Federico Bergenti, Giovanni Caire, and Agostino Poggi. 2005. JADE - A Java Agent Development Framework. In *Multi-Agent Programming: Languages, Platforms and Applications*. Multiagent Systems, Artificial Societies, and Simulated Organizations, Vol. 15. Springer, 125–147.

[5] Federico Bergenti, Giovanni Caire, Stefania Monica, and Agostino Poggi. 2020. The first twenty years of agent-based software development with JADE. *Auton. Agents Multi Agent Syst.* 34, 2 (2020), 36.

[6] Olivier Boissier, Rafael H. Bordini, Jomi Fred Hübner, Alessandro Ricci, and Andrea Santi. 2013. Multi-agent oriented programming with JaCaMo. *Sci. Comput. Program.* 78, 6 (2013), 747–761.

[7] Stefan Boschert and Roland Rosen. 2016. Digital twin—the simulation aspect. In *Mechatronic Futures*. Springer, 59–74.

[8] Andrei Ciortea, Simon Mayer, Fabien Gandon, Olivier Boissier, Alessandro Ricci, and Antoine Zimmermann. 2019. A Decade in Hindsight: The Missing Bridge Between Multi-Agent Systems and the World Wide Web. In *Proc. of the 18th Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS '19)*. IFAAMAS, 1659–1663.

[9] Angelo Croatti, Matteo Gabellini, Sara Montagna, and Alessandro Ricci. 2020. On the Integration of Agents and Digital Twins in Healthcare. *Journal of Medical Systems* 44, 9 (04 Aug 2020), 161.

[10] Angelo Croatti, Sara Montagna, Alessandro Ricci, Emiliano Gamberini, Vittorio Albarello, and Vanni Agnoletti. 2019. BDI personal medical assistant agents: The case of trauma tracking and alerting. *Artificial Intelligence in Medicine* 96 (2019), 187–197.

[11] Ian Dickinson and Michael J. Wooldridge. 2003. Towards practical reasoning agents for the semantic web. In *Proc. of the 2nd Int. Conf. on Autonomous Agents & Multiagent Systems (AAMAS '03)*. ACM, 827–834.

[12] Kurt M. Dresner and Peter Stone. 2008. A Multiagent Approach to Autonomous Intersection Management. *J. Artif. Intell. Res.* 31 (2008), 591–656.

[13] Pavlos Eirinakis, Kostas Kalaboukas, Stavros Lounis, Ioannis Mourtos, Jože M. Rožanec, Nenad Stojanovic, and Georgios Zois. 2020. Enhancing Cognition for Digital Twins. In *2020 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. 1–7.

[14] Dieter Fensel, Umutcan Simsek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, and Alexander Wahler. 2020. *Knowledge Graphs - Methodology, Tools and Selected Use Cases*. Springer.

[15] David Gelernter. 1991. *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., New York, NY, USA.

[16] Edward Glaessgen and David Stargel. 2012. The digital twin paradigm for future NASA and US Air Force vehicles. In *Proc. of the 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*.

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

[18] Claudio Gutierrez and Juan F. Sequeda. 2021. Knowledge Graphs. *Commun. ACM* 64, 3 (Feb. 2021), 96–104.

[19] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, and et al. 2021. Knowledge Graphs. *Comput. Surveys* 54, 4 (Jul 2021), 1–37.
[20] Nicholas R. Jennings. 2001. An Agent-Based Approach for Building Complex Software Systems. *Commun. ACM* 44, 4 (April 2001), 35–41.
[21] Tobias Käfer and Andreas Harth. 2020. Tutorial: Distributed Knowledge Graphs for the Web of Things. In *10th International Conference on the Internet of Things Companion* (Malmö, Sweden) *(IoT '20 Companion)*. Association for Computing Machinery, New York, NY, USA, Article 13, 4 pages.
[22] Niki Kousi, Christos Gkournelos, Sotiris Aivaliotis, Christos Giannoulis, George Michalos, and Sotiris Makris. 2019. Digital twin for adaptation of robots' behavior in flexible robotic assembly lines. *Procedia Manufacturing* 28 (2019), 121 – 126. 7th International conference on Changeable, Agile, Reconfigurable and Virtual Production (CARV2018).
[23] Vladimir Kuts, Tauno Otto, Toivo Tähemaa, and Yevhen Bondarenko. 2019. Digital twin based synchronised control and simulation of the industrial robotic cell using virtual reality. *Journal of Machine Engineering* 19 (02 2019), 128–144.
[24] Ying Liu, Lin Zhang, Yuan Yang, Longfei Zhou, Lei Ren, Fei Wang, Rong Liu, Zhibo Pang, and M. Jamal Deen. 2019. A novel cloud-based framework for the elderly healthcare services using digital twin. *IEEE Access* 7 (2019), 49088–49101.
[25] Somayeh Malakuti and Sten Grüner. 2018. Architectural Aspects of Digital Twins in IIoT Systems. In *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings* (Madrid, Spain) *(ECSA '18)*. Association for Computing Machinery, New York, NY, USA, Article 12, 2 pages.
[26] Stefano Mariani, Giacomo Cabri, and Franco Zambonelli. 2021. Coordination of Autonomous Vehicles: Taxonomy and Survey. *Comput. Surveys* 54, 1, Article 19 (Feb. 2021), 33 pages.
[27] Members of the Digital Framework Task Group. 2018. *White paper: The Gemini Principles.* Technical Report. Centre of Digital Built Britain. Available at https://www.cdbb.cam.ac.uk/DFTG/GeminiPrinciples. Last access: 20210401.
[28] Roberto Minerva, Gyu Myoung Lee, and Noël Crespi. 2020. Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models. *Proc. IEEE* 108, 10 (2020), 1785–1824.
[29] Sara Montagna, Angelo Croatti, Alessandro Ricci, Vanni Agnoletti, Vittorio Albarello, and Emiliano Gamberini. 2020. Real-time tracking and documentation in trauma management. *Health Informatics Journal* 26, 1 (2020), 328–341.
[30] Álvaro F. Moreira, Renata Vieira, Rafael H. Bordini, and Jomi Fred Hübner. 2005. Agent-Oriented Programming with Underlying Ontological Reasoning. In *Declarative Agent Languages and Technologies III, 3rd Int. Workshop, DALT 2005, Utrecht, The Netherlands (Lecture Notes in Computer Science, Vol. 3904)*. Springer, 155–170.
[31] Michalis Mountantonakis and Yannis Tzitzikas. 2019. Large-Scale Semantic Integration of Linked Data: A Survey. *ACM Comput. Surv.* 52, 5, Article 103 (Sept. 2019), 40 pages.
[32] Elisa Negri, Luca Fumagalli, and Marco Macchi. 2017. A review of the roles of digital twin in CPS-based production systems. *Procedia Manufacturing* 11 (2017), 939–948.
[33] Allen Newell. 1982. The Knowledge Level. *Artif. Intell.* 18, 1 (1982), 87–127.
[34] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. 2008. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems* 17, 3 (2008), 432–456.
[35] Giovanni Pezzulo. 2008. Coordinating with the Future: The Anticipatory Nature of Representation. *Minds Mach.* 18, 2 (2008), 179–225.
[36] Marco Picone, Stefano Mariani, Marco Mamei, and Franco Zambonelli. 2021. WIP: Preliminary Evaluation of Digital Twins on MEC Software Architecture. In *IEEE 22nd International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. In press.
[37] Bastian Quilitz and Ulf Leser. 2008. Querying Distributed RDF Data Sources with SPARQL. In *The Semantic Web: Research and Applications*, Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 524–538.
[38] Anand S. Rao and Michael P. Georgeff. 1991. Modeling Rational Agents within a BDI-Architecture. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91). Cambridge, MA, USA, April 22-25, 1991.* Morgan Kaufmann, 473–484.
[39] Alessandro Ricci, Angelo Croatti, and Sara Montagna. 2021. Pervasive and Connected Digital Twins – A Vision for Digital Health. *IEEE Internet Computing* (jan 2021).
[40] Alessandro Ricci, Michele Piunti, Luca Tummolini, and Cristiano Castelfranchi. 2015. The Mirror World: Preparing for Mixed-Reality Living. *IEEE Pervasive Computing* 14, 2 (2015), 60–63.
[41] Dominik Riemer. 2018. Feeding the Digital Twin: Basics, Models and Lessons Learned from Building an IoT Analytics Toolbox (Invited Talk). In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 4212–4212.
[42] Joze M. Rozanec, Lu Jinzhi, Aljaz Kosmerlj, Klemen Kenda, Kiritsis Dimitris, Viktor Jovanoski, Jan Rupnik, Mario Karlovcec, and Blaz Fortuna. 2020. Towards Actionable Cognitive Digital Twins for Manufacturing. In *Proceedings of the International Workshop on Semantic Digital Twins co-located with the 17th Extended Semantic Web Conference, SeDiT@ESWC 2020, Heraklion, Greece, June 3, 2020 (CEUR Workshop Proceedings, Vol. 2615)*. CEUR-WS.org.

[43] Roberto Saracco. 2019. Digital Twins: Bridging Physical Space and Cyberspace. *Computer* 52, 12 (2019), 58–64.
[44] Ehab Shahat, Chang T. Hyun, and Chunho Yeom. 2021. City Digital Twin Potentials: A Review and Research Agenda. *Sustainability* 13, 6 (2021).
[45] Jack Sleuters, Yonghui Li, Jacques Verriet, Marina Velikova, and Richard Doornbos. 2019. A Digital Twin Method for Automated Behavior Analysis of Large-Scale Distributed IoT Systems. In *2019 14th Annual Conference System of Systems Engineering (SoSE)*. IEEE, 7–12.
[46] Eugene Y. Song, Martin Burns, Abhinav Pandey, and Thomas Roth. 2019. IEEE 1451 Smart Sensor Digital Twin Federation for IoT/CPS Research. In *2019 IEEE Sensors Applications Symposium (SAS)*. IEEE, 1–6.
[47] Viniciu Souza, Robson Cruz, Walmir Silva, Sidney Lins, and Vicente Lucena. 2019. A Digital Twin Architecture Based on the Industrial Internet of Things Technologies. In *2019 IEEE Int. Conf. on Consumer Electronics (ICCE)*. 1–2.
[48] Christian Stary. 2021. Digital Twin Generation: Re-Conceptualizing Agent Systems for Behavior-Centered Cyber-Physical System Development. *Sensors* 21, 4 (2021).
[49] Charles Steinmetz, Achim Rettberg, Fabíola Gonçalves C Ribeiro, Greyce Schroeder, and Carlos E. Pereira. 2018. Internet of things ontology for digital twin in cyber physical systems. In *2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*. IEEE, 154–159.
[50] Fei Tao and Qinglin Qi. 2019. Make more digital twins. *Nature* 573, 7775 (2019), 490–491.
[51] Fei Tao, He Zhang, Ang Liu, and A. Y. C. Nee. 2019. Digital Twin in Industry: State-of-the-Art. *IEEE Trans. on Industrial Informatics* 15, 4 (2019), 2405–2415.
[52] Thomas H.-J. Uhlemann, Christian Lehmann, and Rolf Steinhilper. 2017. The digital twin: Realizing the cyber-physical production system for Industry 4.0. *Procedia Cirp* 61 (2017), 335–340.
[53] Danny Weyns, Andrea Omicini, and James J. Odell. 2007. Environment as a First-class Abstraction in Multi-Agent Systems. *Autonomous Agents and Multi-Agent Systems* 14, 1 (Feb. 2007), 5–30.
[54] Michael J. Wooldridge and Nicholas R. Jennings. 1995. Intelligent agents: theory and practice. *Knowl. Eng. Rev.* 10, 2 (1995), 115–152.
[55] Yan Xu, Yanming Sun, Xiaolong Liu, and Yonghua Zheng. 2019. A Digital-Twin-Assisted Fault Diagnosis Using Deep Transfer Learning. *IEEE Access* 7 (2019), 19990–19999.
[56] Cheng Zhou, Hongwei Yang, Xiaodong Duan, Diego Lopez, Antonio Pastor, Qin Wu, Mohamed Boucadair, and Christian Jacquenet. 2021. *Concepts of Digital Twin Network*. Internet-Draft draft-zhou-nmrg-digitaltwin-network-concepts-03. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-zhou-nmrg-digitaltwin-network-concepts-03 Work in Progress.