SoftwareX 31 (2025) 102275

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original Software Publication

HWoDT Framework: A toolchain to build interoperable Digital Twin Ecosystems

Andrea Giulianelli [a], Samuele Burattini [a,*], Andrei Ciortea [b], Alessandro Ricci [a]

[a] Department of Computer Science and Engineering, Alma Mater Studiorum University of Bologna, Via dell'Universita' 50, Cesena, Italy
[b] School of Computer Science, University of St.Gallen, St. Gallen, Switzerland

ARTICLE INFO

Keywords:
Digital Twin Ecosystems
Web architecture
Hypermedia
Interoperability

ABSTRACT

Digital Twins are increasingly being applied as a design paradigm to model complex Cyber-Physical Systems. These systems extend beyond the original Digital Twin concept, which focused on the virtualization of individual standalone physical assets in vertical application stacks. The interconnected nature of physical environments foresees in Digital Twin Ecosystems a novel abstraction to represent multiple connected Digital Twins of heterogeneous assets. In this paper, we present a toolchain to support the development of Digital Twin Ecosystems based on Web standards and principles, to provide a uniform interface and tackle the heterogeneous nature of Digital Twins.

Code metadata

Current code version	v2.3.0 *(wodt-platform)* v1.0.7 *(azuredt-wodt-adapter)* v5.2.1 *(wldt-wodt-adapter)* v1.1.3 *(ditto-wodt-adapter)*
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-25-00150
Permanent link to Reproducible Capsule	*None*
Legal Code License	Apache License 2.0
Code versioning system used	Git
Software code languages, tools, and services used	Java, Kotlin
Compilation requirements, operating environments & dependencies	Java 11 or greater *(wldt-wodt-adapter)*, Java 17 or greater *(others)*, Kotlin 2 or greater
If available Link to developer documentation/manual	https://web-of-digital-twins.github.io/
Support email for questions	andrea.giulianelli7@unibo.it samuele.burattini@unibo.it

1. Motivation and significance

Introduced in manufacturing [1], and popularized by NASA [2], DTs are now considered in several application domains [3–5] as an engineering abstraction for modeling Cyber-Physical Systems (CPS) [6,7]. In this panorama, a Digital Twin (DT) can be defined as a contextualized software replica of a corresponding physical asset (PA) synchronized through a bidirectional connection named *twinning* or *shadowing* process [8,9].

Digital Twin Ecosystems are a novel research direction to improve the modeling of large-scale systems, enabling applications to leverage services that extend beyond the boundaries of a single DT, such as queries, discovery, and monitoring of the aggregated mirrored portion of the physical world [10]. The UK National Digital Twin (NDT) [11] is an important example of this vision, defining design principles and frameworks to streamline the creation of DT ecosystems which inherently exhibit several degrees of heterogeneity as they may connect DTs

* Corresponding author.
E-mail addresses: andrea.giulianelli7@unibo.it (A. Giulianelli), samuele.burattini@unibo.it (S. Burattini), andrei.ciortea@unisg.ch (A. Ciortea), a.ricci@unibo.it (A. Ricci).

https://doi.org/10.1016/j.softx.2025.102275
Received 7 March 2025; Received in revised form 8 July 2025; Accepted 14 July 2025
Available online 12 August 2025

Table 1
Comparison of DT ecosystem feature support in state-of-the-art DT technologies based on the documentation as of May 2025.

	Azure Digital Twins	Amazon IoT TwinMaker	Eclipse Ditto	Twinbase [16]	HWoDT
DT Add/Remove	HTTP API	HTTP API	HTTP API	GitHub API	HTTP API
DT Relationships	Yes	No[a]	No	Yes	Yes
DT Description	DTDL +JSON	No[a]	WoT TD +Ditto Thing	JSON/YAML	WoT TD+KG
Self-described API	No	No	WoT TD	No	WoT TD
Query Support	ADT Query Language[b]	PartiQL[c]	No	No	SPARQL
Observe DTs	Req. Integration	Req. Integration	WebSocket	No	WebSocket

[a] Amazon IoT TwinMaker uses *entities* as the main concept, defined as parts of a DT so there is no clear description of a DT.
[b] https://learn.microsoft.com/en-us/azure/digital-twins/concepts-query-language.
[c] https://partiql.org/.

developed by different stakeholders using different technologies [6,12,13].

Similarly to how Internet of Things (IoT) systems and CPS often deal with heterogeneous interconnected physical assets – leading to interoperability issues – interoperability is being envisioned as a necessity for maturity in the DT context [14,15], especially when envisioning DT ecosystems [10,11]: dynamic sets of DTs that individually represent PAs and their relationships and are possibly based on heterogeneous technologies and platforms.

The technological landscape to create DTs is, in fact, very rich. We limit our analysis to related works in the context of DT platforms acting as middleware collecting and exposing data of several different DTs and allowing users to interact with them. Table 1 presents a high level feature comparison. Namely, we report how different platforms support the management of multiple DTs, if they provide off-the-shelf support to track relationships between DTs and whether they allow querying information across DTs.

The main limitations of the presented approaches are that they require compliance of the DT to platform-specific constraints in terms of: *(i)* protocols and data-formats for the DT to communicate with the platform *(ii)* APIs used by consumers to interact with either the DTs or the ecosystem as a whole (when supported) . This makes DT implementations often tailored to the specific target platform, and limits interaction across multiple platforms since they often propose custom solutions rather than adopting open standards.

Bridging heterogeneous technologies and platforms is a general challenge even outside of the DT domain. Their integration can be tackled through several strategies [17]. Among these, using *integration platforms* which enable service discovery and uniform interfaces is a common architectural choice [18] e.g., to integrate different cloud solutions [19], or even IoT services, such as in the case of the Eclipse Arrowhead framework [20].

For these reasons,following the WoDT idea [10], we propose to tackle the engineering of heterogeneous DT ecosystems exploiting Web architectural principles and technologies [21]. Rather than having *all* DTs conforming to the requirements of a single platform, our proposal for a *HWoDT* advocates for *each* DT within an ecosystem to:

- expose its functionalities as-a-service under a self-described RESTful interface [22] following the WoDT metamodel based on properties, actions, events, and relationships [10];
- expose its current state as a KG to encode semantics into the DT data using a standardized approach [23,24].

This shift of responsibilities from the platform to the DT allows to use the Web itself – with its set of standards – as an integration platform for DT ecosystems. Additionally, as we demonstrate with our prototype, ecosystem-level services can be provided through the composition of such Web-based DTs. As shown in Fig. 1, a DT ecosystem can be composed of existing DTs, implemented with different technologies. Consumers can seamlessly navigate and interact with DTs as they expose the same uniform interface. This paper presents the tools supporting in implementing heterogeneous DT ecosystems following the principles of the HWoDT. This includes:

- *Adapters* which implement the HWoDT uniform interface for a specific DT technology;
- a prototype of a *WoDT Platform* which implements an aggregator for a DT ecosystem and exposes services within the scope boundaries of the ecosystem defined by the registered set of DTs.

2. Software description

Fig. 1 depicts the main HWoDT Framework elements. A WoDT Platform provides support for a DT ecosystem, aggregating data from the DTs that are registered within it to model the portion of interest of the real world.

HWoDT Adapters convert the DT-specific technology APIs to a uniform RESTful interface. They manage *(i)* DT and PA identification, *(ii)* DT semantic representations, *(iii)* and expose interaction patterns to obtain the state of the DT, observe it over time, and modify it through action requests [21].

In particular, DTs are globally identified by a URI and expose two machine-readable representations:

- a *Digital Twin Description (DTD)* which semantically describes the DT model, metadata, and interaction affordances using the Web of Things (WoT) Thing Description (TD) [25]. Its purpose is to provide a description of the DT that platforms and consumers can fetch to understand how to interact with the DT.
- a *Digital Twin Knowledge Graph (DTKG)* which semantically represents the state of the DT through a KG using domain-specific ontologies. The DTKG represents the most recent state of the DT.

The framework is completely open-source (see Code metadata table) and includes:

- a Kotlin prototype of a WoDT Platform that can process WoT TD-based DTD, observe heterogeneous DTs and expose ecosystem services;
- HWoDT Adapters implementing a uniform RESTful interface for three different technologies: Azure Digital Twins, Eclipse Ditto, and White Label Digital Twins (WLDT) [26].

2.1. Software architecture

The overall abstract architecture is depicted in Fig. 2. It shows the internal components and interactions of a WoDT Platform with a DT through its HWoDT Adapter.

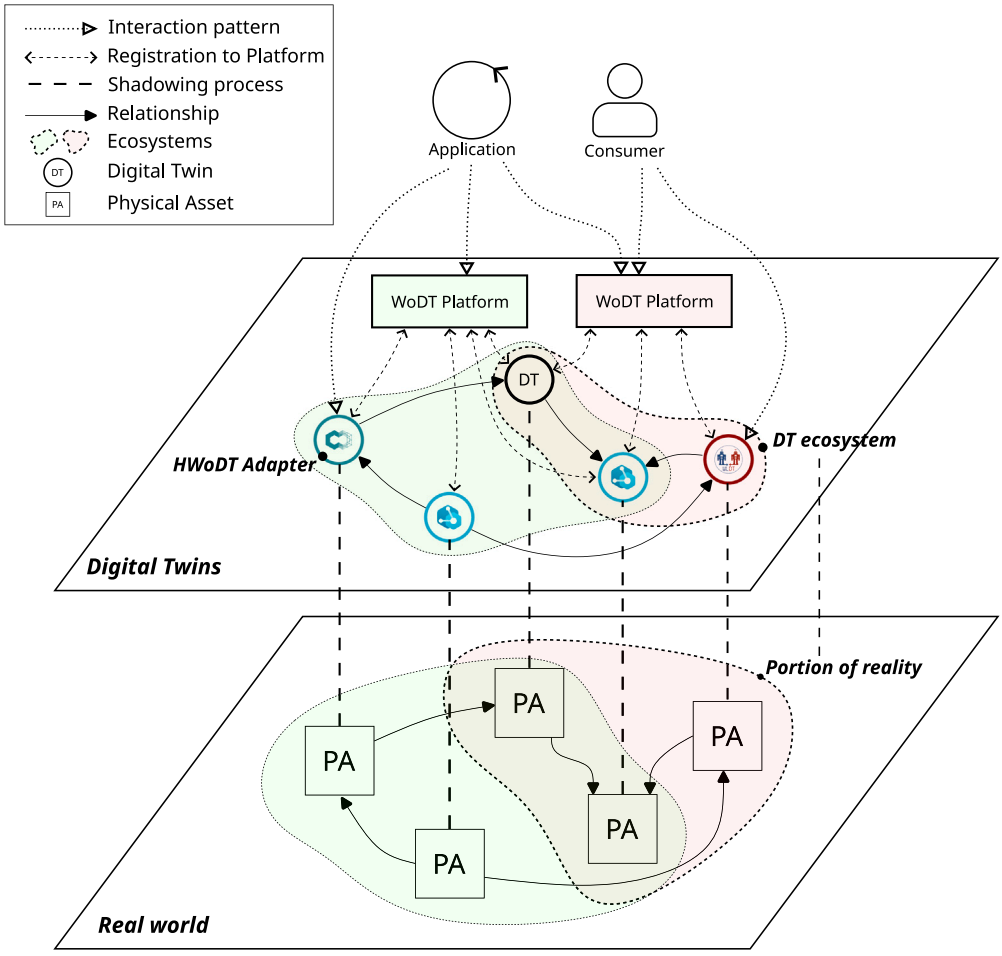

Fig. 1. The HWoDT high-level schema, in the real world PAs belonging to different organizations and domains are connected by relationships. In the digital world, cross-domain ecosystems of DTs mirror different portions of reality, supported by the WoDT Platform.

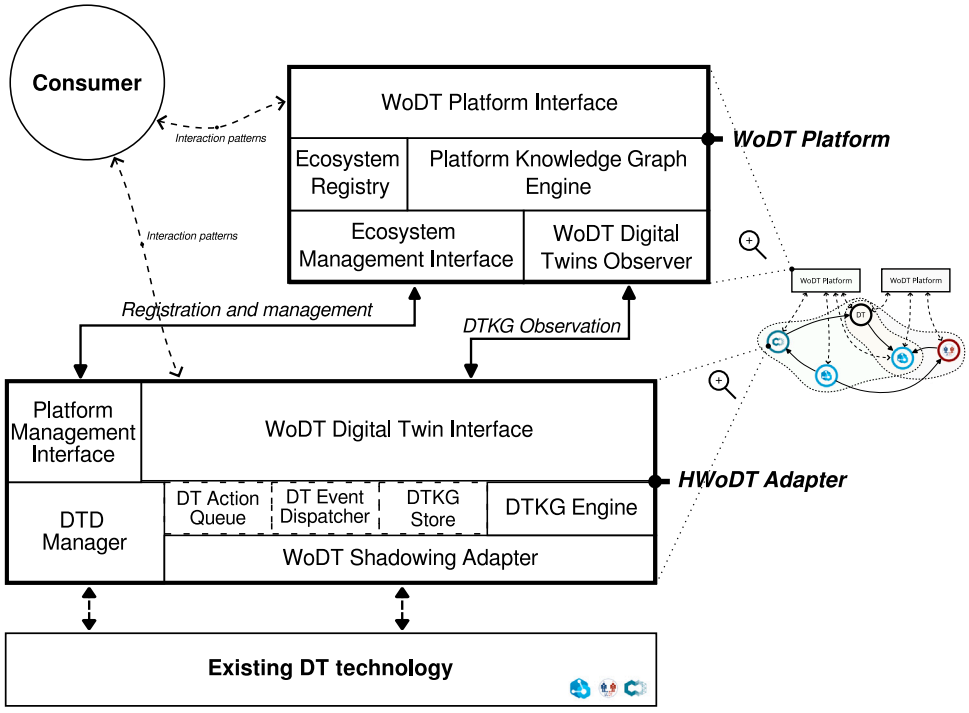

Fig. 2. Overall abstract architecture of the WoDT Platform and HWoDT Adapter, highlighting the main modules and interactions. Modules on the top depends on modules on the bottom.

2.1.1. HWoDT adapters

Adapters implement the uniform interface of the HWoDT on top of DTs developed with any existing (supported) technology. We here describe the generic modular architecture of an adapter, that serves as the reference for ours – and future – implementations.

The *WoDT Shadowing Adapter* maps the existing DT shadowing process, translating the model to the WoDT metamodel [10]. It is further in charge of updating the DTKG. The *DTKG Engine* manages and stores the latest DTKG, delegating memorization duties to the *DTKG Store*. Additionally, the *DT Action Queue* and *DT Event Dispatcher* are optional components that decouple action enqueuing and event dispatching from the *WoDT Shadowing Adapter*. The DTD is instead created and managed by the *DTD Manager*.

The *Platform Management Interface* handles the registration process to WoDT Platforms, while the *WoDT Digital Twin Interface* exposes the APIs to interact with the DT.

2.1.2. WoDT Platform

The WoDT Platform exposes APIs for DTs validation, registration, update, and deletion through the *Ecosystem Management Interface*. DTs inside the ecosystem are registered on the *Ecosystem Registry*, which is also responsible for mapping DTs URI to local URIs accessible within the platform. This forms the foundation for providing a cache of the DT ecosystem that can be navigated at the platform level.

To observe each DT, the *WoDT Digital Twins Observer* handles the DTKG observation process interacting with DTs through the APIs of the DTs uniform interface. A continuous merging process, managed by the *Platform Knowledge Graph Engine*, creates the DT ecosystem KG from the latest DTKG and DTD of the registered DTs. This is an aggregation of the latest available data coming from all the DTs.

Finally, the *WoDT Platform Interface* exposes the platform APIs for the ecosystem-level services.

2.2. Software functionalities

After outlining the high-level architecture of the framework, we now examine the functionalities provided by the components that developers wishing to implement a DT ecosystem can use.

2.2.1. HWoDT adapters

Identification. HWoDT Adapters handle the identification of both DTs and PAs. Specifically, if the underlying technology supports a form of DT identification, the existing identifier is mapped to a URI; otherwise, identification support is added.

Metamodel conversion. To ensure uniformity of modeling in the whole DT ecosystem, DTs models are converted to the WoDT metamodel, composed of properties, relationships, actions, and events [10].

DTD & DTKG generation. HWoDT Adapters generate and expose the DTD based on the underlying representation of the DT model. The *DTD Manager* is responsible for the generation process following the *DTD Conceptual Model*.[1] Our adapters implement a WoT Thing Description-based DTD implementation for compatibility with existing standards.

The DT live state is continuously mapped into the DTKG by the *WoDT Shadowing Adapter* using an RDF-based KG aligned with the domain ontology. Each adapter can be configured to use any ontology, allowing developers to choose the most representative ones for the application domain.

Platforms management. HWoDT Adapters can be configured to automatically register to default WoDT Platforms when deployed. Alternatively, the WoDT Platform can notify existing DTs which are manually registered. HWoDT Adapters hence support a registration notification endpoint.

[1] https://github.com/Web-of-Digital-Twins/dtd-conceptual-model

Interaction patterns. HWoDT Adapters support the following interaction patterns:

- *DTKG & DTD Access*: an HTTP GET request on the DT URI redirects to the DTKG serialization, following Linked Data Principles [27] and ensuring navigability in the resulting DT ecosystem. As the DT qualifies as a *non-information resource*[28,29] we follow the standard Web practice of returning 303 (See Other) status code with the *Location* HTTP header set to the URL of the DTKG information resource.
  In the DTKG response header, an HTTP Link Header with custom relation type `dtd` points to the URI of the DTD. An HTTP GET request on that URI retrieves the corresponding DTD.
- *DTKG Observation*: adapters provide a WebSocket endpoint to observe DTKG updates.
- *Action invocation*: adapters support consumers in invoking DT actions. Consumers can use the DTKG to verify the availability of actions based on the current state. Moreover, the DTD can be exploited to obtain the interaction affordance required to execute the request.

2.2.2. WoDT platform

DT registration and management. The *Ecosystem Management Interface* manages the DT registration, update, and deletion invoked either manually by an administrator or automatically by the DT adapter. The DT DTD is submitted so that the Platform can validate it to fetch the required information, in particular, the DTKG Observation endpoint used to observe the DTKG evolution and maintain the DT ecosystem KG updated.

The *Ecosystem Registry* tracks the registered DTs and the mapping of their URIs to local ones.

Dt ecosystem KG creation. The *Platform KG Engine* manages the DT ecosystem KG by continuously aggregating the latest DTKGs and DTDs from the registered DTs. Storing the most recent snapshot of registered DTs and leveraging DT URI mapping, it offers a local cache that consumers can navigate.

Interaction patterns. The WoDT Platform supports the following interaction patterns:

- *DT ecosystem KG Snapshot*: an HTTP GET request on the WoDT Platform URI redirects to the DT ecosystem KG representation, following a similar approach to DTKG access described above.
- *DT ecosystem KG Observation*: the platform provides a WebSocket endpoint for DT ecosystem KG observation.
- *DT ecosystem KG SPARQL Query*: the platform exposes a SPARQL endpoint [30] to make read-only queries on the DT ecosystem KG.
- *Cached DT Snapshot and Navigation*: an HTTP GET request on the DT mapped URL enables consumers to exploit local caches. Since each registered DT URI is mapped, consumers are able to navigate within the local cache. An HTTP Link Header with relation type `original` allows consumers to navigate to the original DT.
- *Catalog Service*: it is possible to obtain the DT ecosystem catalog, that lists the currently registered DTs.
- *Directory Service*: this service efficiently retrieves the DTs associated with a specific PA ID, enabling consumers to access different representations of the same PA within the ecosystem.

2.3. Performance evaluation of the prototype

The architecture has been designed to foster modularity, openness, and consistency, enabling the integration of heterogeneous DT technologies within the same ecosystem, and enabling querying and observation based on Web and Semantic Web reference technologies. Given this architecture, the performance overhead introduced is almost neg-

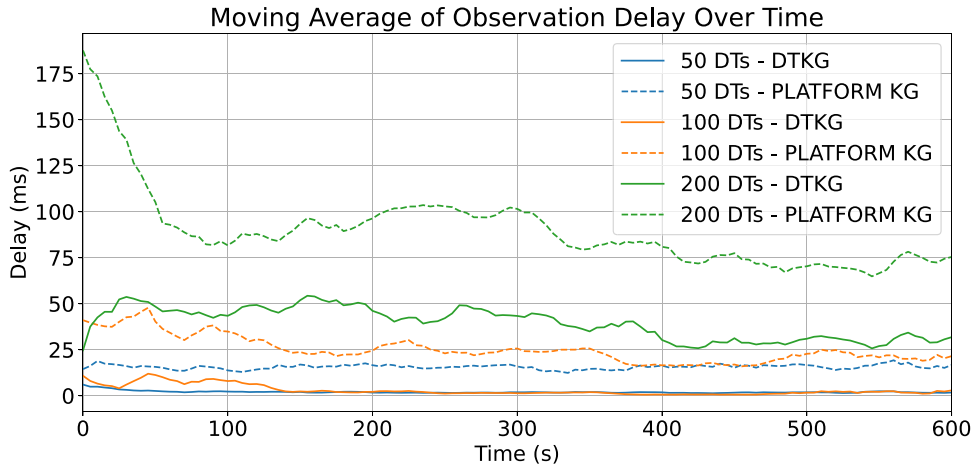

Fig. 3. Average observation delay over time with one client observing each DT and one observing the Platform KG.

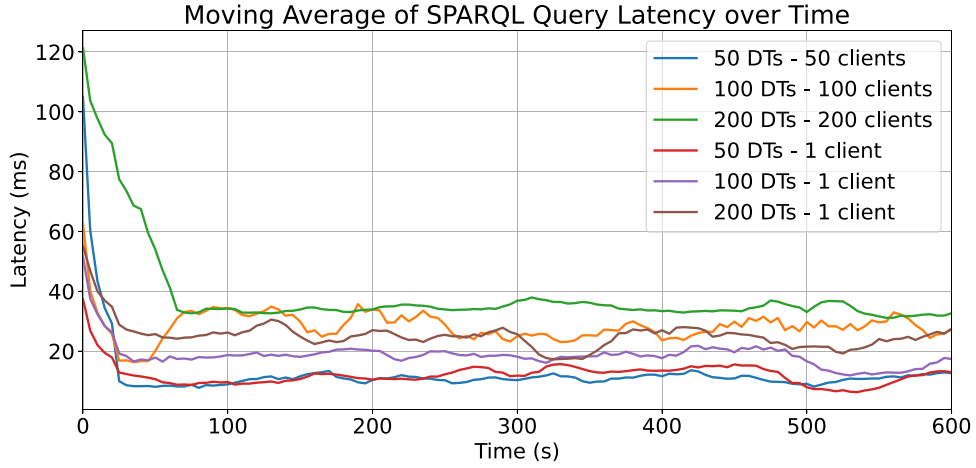

Fig. 4. Average observation delay over time with either one client or one client per DT performing SPARQL queries every second.

ligible when interacting with an individual DTs. The overhead grows whenever clients make queries or observations involving multiple DTs through the platform KG.

To capture the impact of such overhead, we performed tests on the current prototype emulating a resource-intensive DT ecosystem composed of a variable number of temperature sensors sending updates for ten minutes (see Appendix for details).

We measure the delay introduced by the processing of DT updates by the platform: *(i)* when observing the cache of a DTKG; *(i)* when observing the platform KG. Results in Fig. 3 show that observing the platform KG is generally always more costly than observing one DT as the updates of each DT is merged in the platform KG.

We additionally measure the latency of SPARQL queries on the platform KG every second, with variable numbers of DTs and clients. Results in Fig. 4 show that with large numbers of DTs, queries are a more effective way of observing the evolution of the ecosystem over time, albeit clients may not receive all the updates from all DTs, but only the last available state at the time of querying.

Notably, both figures show that the system suffers a cold-start problem as all the DTs register to the platform when booting up. In our analysis, we additionally found that the most significant performance bottleneck is caused by the serialization of the Platform KG. In future works we will explore possible optimizations for these issues, such as sending only incremental changes (e.g., [31]) of the KGs over WebSockets, rather than the full state.

3. Illustrative example

We showcase the HWoDT Framework with an implementation of a trauma management scenario, originally described in [10]. The goal of this example is to describe the steps developers need to follow to realize a DT ecosystem starting from an existing heterogeneous deployment. Interested readers can access the DT ecosystem implementation on the GitHub organization (see Code metadata table) in the `major-trauma-management-case-study` repository.

In a complex and fragmented environment such as the healthcare domain, we imagine having three legacy systems leveraging different DT technologies: Azure Digital Twins (ADT), WLDT, and Eclipse Ditto (see Fig. 5).

Through HWoDT adapters, we can implement the uniform interfaces on top of the existing DTs. The ADT adapter acts as a middleware, providing support for all the DTs on the same ADT instance. The Ditto adapter is deployed as a middleware and configured to observe an individual DT on a Ditto instance. WLDT DTs instead, must include and configure the WLDT adapter as a custom *Digital Adapter*, adding it as a Java dependency.

To promote semantic interoperability, the adapters have been configured to use the HL7 FHIR ontology,[2] a standard for medical data representation.

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

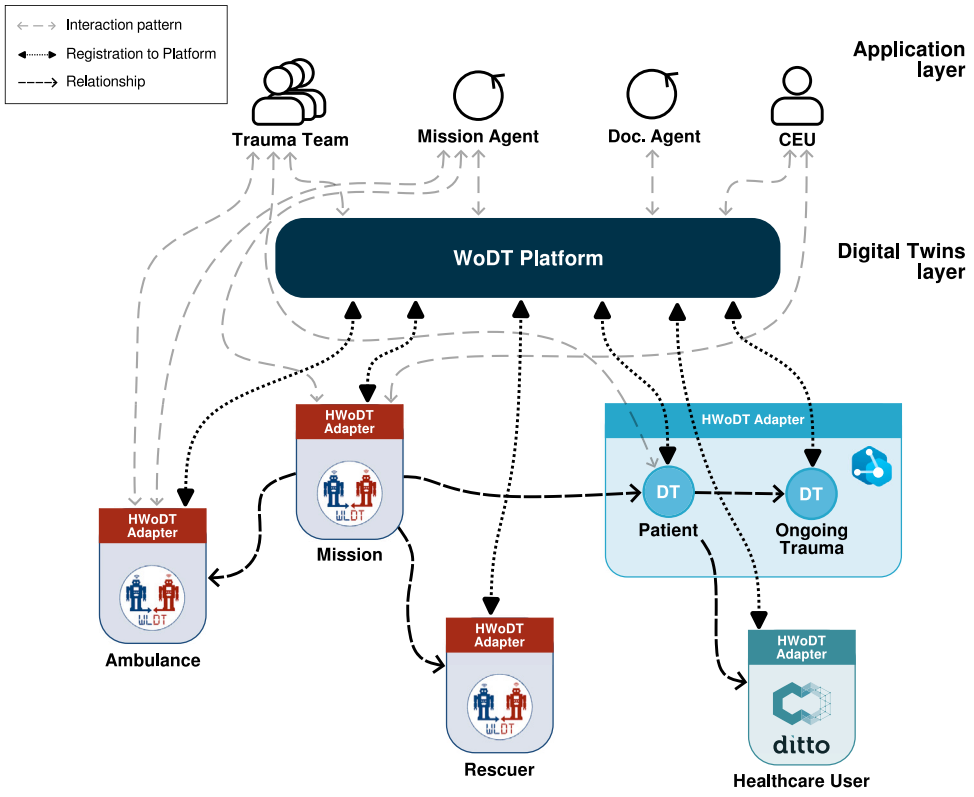

Fig. 5. The HWoDT-based DT ecosystem for the trauma management case study. The image also depicts the interactions present in each layer.

At this stage, DTs are already fully interoperable and can be used individually as-a-service through the REST-based HWoDT uniform interface to exploit the interaction patterns described in Section 2.2.1.

Deploying the WoDT Platform – and registering the DTs to it – provides additional services at the ecosystem level to support application development. For instance, Listing 1 is an example of a SPARQL query that could be used to implement the *Mission Agent* to identify an inactive ambulance and an available rescuer with the desired qualification.

Listing 1 SPARQL Query performed by the Mission Agent to obtain the available ambulances and rescuers.

```
PREFIX fhir: <http://www.hl7.org/fhir/>

SELECT ?ambulance ?rescuer
WHERE {
    ?ambulance a fhir:Location ;
                fhir:status "inactive" .

    ?rescuer a fhir:Practitioner ;
              fhir:qualification ?qualification .
    ?qualification fhir:code ?qualificationObj .
    ?qualificationObj fhir:coding ?qualificationCoding .
    ?qualificationCoding fhir:code ?qualificationCode .
    ?qualificationCode fhir:v "397897005" .

    FILTER NOT EXISTS { ?mission fhir:participant ?
        rescuer }
    FILTER NOT EXISTS { ?mission fhir:location ?ambulance
        }
}
```

The HWoDT-based design eliminates the need for systems to individually query each interested DT against their custom technological-dependent interfaces and subsequently merge the derived heterogeneous information. Instead, they can exploit a uniform interface that enables seamless navigation.

4. Impact

The HWoDT Framework represents the first proposal and open-source software toolkit for implementing ecosystems of heterogeneous DTs based on open Web standards. Heterogeneity is considered an advantage that enables the reuse of existing resources that are already effective within the original domains. We here describe its main benefits, and the impact these can have in the community.

4.1. Benefits for digital twin ecosystem development

The HWoDT uniform interface, implemented via the HWoDT Adapters for each specific DT technology, allows each DT to be used as-a-service and expose its data and services uniformly through a RESTful interface following the idea of DT *servitization* [8]. As a result, the WoDT Platform can implement the DT ecosystem abstraction and services by aggregating the interested DTs independently of the underlying technologies. The HWoDT Framework enables application developers to shift the focus only on the application logic rather than on the diverse DT technologies API and SDKs.

Without the HWoDT applications would need to

1. get data from different DTs of interest with their specific API
2. transform and merge data in a coherent model
3. derive the necessary actions to take
4. act on the selected DTs with their specific API

Differently, with the HWoDT applications can

1. query or observe the ecosystem as a whole through the platform KG
2. derive the necessary actions to take
3. find the API exposed by the DT in its DTD and act on it.

This results in: *(i)* a more stable application layer that requires no modification when a new DT technology is introduced, simplifying the integration of new organizations and stakeholders, and *(ii)* a more stable DT layer that can change the underlying technologies without affecting the upper layers benefiting from information and technology hiding. Furthermore, compatibility with the Linked Data Principles ensures HWoDT DT ecosystems navigability. Of course, this comes at the cost of producing the necessary semantic descriptions for the DTs in terms of DTD and DTKG, configuring the adapters for the specific technology and registering DTs to a platform.

4.2. Community impact

Among the software paradigms that stand to benefit from a digital representation of the real world, agents and Multiagent Systems (MAS) are of particular significance [32,33]. The HWoDT Framework promotes the design of DT ecosystems that can serve as virtual environments for agents [34,35] to reason and interact with the physical world. By leveraging their hypermedia nature and distributed architecture across the Web, DT ecosystems created with the HWoDT Framework are aligned with the *hypermedia MAS* vision [36], providing a robust foundation for autonomous agent-based systems. Additionally, the HWoDT Framework strategically aligns with WoT standards [25,37], ensuring compatibility without introducing a novel technology for developing WoT-compliant DTs. A notable strength of this approach is that a DTD can be seamlessly processed by WoT consumers, thereby enabling the DT to be integrated as a valid WoT Thing and included in WoT mashups.

The work on the HWoDT Framework is significant for interoperability, seen as a necessary condition for the DT maturity [14], where the proposed framework can represent a building brick to achieve interoperability even in pre-existing cross-domain and cross-organization systems. Envisioning REST as the prominent architectural style, the HWoDT Framework represents a strong proposal to design open systems – like the National Digital Twin [11] – contributing to research activities in architecting DT systems [38]. The HWoDT Framework has already been used in collaboration with other researchers and universities,[3] and experiments have been carried out to be used as the middleware for the *Digital Twin Continuum* project [39].

5. Conclusions

The HWoDT Framework is a novel toolchain designed to support the creation of heterogeneous DT ecosystems. The REST-based uniform interface and the compatibility with the Linked Data Principles implement the DT-as-a-service paradigm in a Web-oriented fashion. This enables information and technological hiding, supporting interoperability and navigability.

We consider DT ecosystems a long-term research effort, and plan to evolve the available tools to enhance the support for their development. In the near future we plan to address limitations of the current prototype, improving the performance of observation with RDF event streams and continuous querying, exploring distributed approaches for querying the ecosystem without relying on a centralized middleware and supporting a larger number of protocols and DT platforms through adapters.

[3] https://twinkle-project.github.io/

CRediT authorship contribution statement

Andrea Giulianelli: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Samuele Burattini:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Andrei Ciortea:** Writing – review & editing, Validation, Supervision, Project administration, Methodology, Funding acquisition, Formal analysis, Conceptualization. **Alessandro Ricci:** Writing – review & editing, Validation, Supervision, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the Italian MUR in the framework of the CrossLab and the FoReLab projects (Departments of Excellence), by the European Union - NextGenerationEU - under PRIN 2022 project TWINKLE (project ID: 20223N7WCJ and CUP: E53D23007770001), and by the European Union - NextGenerationEU and Azienda Unità Sanitaria Locale (AUSL) della Romagna under the project “Digital Twins Ecosystems for the clinical, strategic and process governance in Healthcare” (DM 352/2022) - CUP J33C22001400009. This research received funding from the Swiss National Science Foundation under grant No. 189474 (*HyperAgents*).

Appendix. Experimental setup

To measure the performances of our prototype in a variety of settings, we emulate having multiple DTs connected to the same platform. We imagine an application scenario with a sensor network of temperature sensors. We implement and run DTs of the sensors with the WLDT framework and run all DTs on the same process. As realism of the temperature values is not relevant for this experiment, the DTs are programmed to simply generating a random temperature value between 0 and 100 every second and publishing a new state update. To avoid having perfectly synchronized sensors, we further start them with a random offset of at most one second.

We run both the DTs and the platform on the same machine with a 13th Gen Intel(R) Core(TM) i7-13700H CPU and 32 GB of RAM using the OpenJDK 21.0.7 Java Runtime Environment.

As the main features of the platform are observing changes through WebSockets or querying the Platform KG with SPARQL, we measured the delay induced by the platform when processing data and requests for both interactions by observing logs of the platform.

Websocket observation. To measure the delay when observing the DT ecosystem through the WebSockets, we collect logs by printing when incoming data is received from a DT and when the corresponding update is processed by the platform updating the representation of each cached DTKG and to the global Platform KG. We test the system by having one client for each DT plus one observing the whole Platform KG. We repeat the measurements with 50, 100 and 200 DTs registered to the platform. Table A.2 shows statistics for both the observation of the DTKG and the Platform KG. As expected, the average delay increases with the number of simultaneously connected DTs, furthermore, the average delay of observing the Platform KG is higher than observing individual DTs. In our analysis we verified that this is caused by the serialization of the whole KG to stream it at every update, which is computationally intensive. Fig. 3 shows the average delay across all DTKG observation events, and the average delay of Platform KG observation over time. We make a bucketed average with 3 s buckets and then compute the moving average with a window of size five.

Table A.2
Delay (ms) for DTKG and Platform KG observation across different numbers of Digital Twins, with one client for each DTKG and one for the Platform KG.

Digital Twins	DTKG			Platform KG		
	50	100	200	50	100	200
Mean	1.90	2.88	39.36	15.65	24.97	87.85
Std Dev	1.97	6.37	38.47	9.02	14.98	40.80
Min	0	0	0	3	6	16
25%	1	0	2	10	14	56
50%	2	1	31	12	20	83
75%	2	2	65	17	32	114
Max	30	69	259	60	103	371

Table A.3
Latency (ms) for SPARQL queries with clients performing queries every second.

Digital Twins	One client			N Clients (N = DTs)		
	50	100	200	50	100	200
Mean	11.98	18.16	25.16	11.01	28.01	37.36
Std Dev	6.26	7.64	8.54	12.97	27.31	27.72
Min	4	7	15	3	6	14
25%	9	13	19	5	10	25
50%	12	18	26	8	18	36
75%	15	22	28	15	36	40
Max	122	119	148	451	583	881

SPARQL queries. To measure latency in performing SPARQL queries, we log the time required by the platform to process each request. We implement a simple client performing a query every second in line with the application scenario of the experimental setup in which DTs produce an update every second. Listing 2 shows the SPARQL query used in the experiment, which selects all the sensors whose current measured temperature value is over 50 degrees.

Listing 2 SPARQL used in the experimental setting, searching for DTs of sensors measuring more than 50 degrees

```
PREFIX ex: <http://example.org/ontology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?sensor ?temperature
WHERE {
  ?sensor a ex:TemperatureSensor ;
          ex:hasTemperature ?temperature .
  FILTER(xsd:double(?temperature) > 50)
}
```

We measure latency having either only one client or one client per DT with 50, 100, and 200 DTs registered to the platform. Table A.3 shows statistics for the latency for both one client and N clients where N is the number of DTs. Results confirm that performances of queries is influenced by both having more DTs as the KG is bigger and more dynamic and by having more clients connected to the platform as the platform needs to answer more requests in the same amount of time. Fig. 4 shows the average latency of queries over time. We make a bucketed average with 3 s buckets and then compute the moving average with a window of size five.

Data availability

Data used in the experiments is available upon request.

References

[1] Grieves M. Digital twin: manufacturing excellence through virtual factory replication. White Pap 2014;1(2014):1–7.

[2] Glaessgen E, Stargel D. The digital twin paradigm for future NASA and US Air Force vehicles. In: 53rd AIAA/aSME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA. 2012, p. 1818.

[3] Ricci A, Croatti A, Montagna S. Pervasive and connected Digital Twins - A vision for digital health. IEEE Internet Comput 2022;26(5):26–32. http://dx.doi.org/10.1109/MIC.2021.3052039.

[4] Mohammadi N, Taylor JE. Smart city Digital Twins. In: 2017 IEEE symposium series on computational intelligence. IEEE; 2017, p. 1–5. http://dx.doi.org/10.1109/SSCI.2017.8285439.

[5] Rodrigo MS, Rivera D, Moreno JI, Alvarez-Campana M, López DR. Digital Twins for 5G networks: A modeling and deployment methodology. IEEE Access 2023;11:38112–26. http://dx.doi.org/10.1109/ACCESS.2023.3267548.

[6] Qi Q, Tao F, Hu T, Anwer N, Liu A, Wei Y, et al. Enabling technologies and tools for Digital Twin. J Manuf Syst 2021;58:3–21. http://dx.doi.org/10.1016/j.jmsy.2019.10.001.

[7] Tao F, Zhang H, Liu A, Nee AYC. Digital twin in industry: State-of-the-art. IEEE Trans Ind Informatics 2019;15(4):2405–15. http://dx.doi.org/10.1109/TII.2018.2873186.

[8] Minerva R, Lee GM, Crespi N. Digital twin in the IoT context: A survey on technical features, scenarios, and architectural models. Proc IEEE 2020;108(10):1785–824. http://dx.doi.org/10.1109/JPROC.2020.2998530.

[9] Saracco R. Digital twins: Bridging physical space and cyberspace. Computer 2019;52(12):58–64. http://dx.doi.org/10.1109/MC.2019.2942803.

[10] Ricci A, Croatti A, Mariani S, Montagna S, Picone M. Web of digital twins. ACM Trans Internet Techn. 2022;22(4):101:1–30. http://dx.doi.org/10.1145/3507909.

[11] Kendall J. National digital twin: Integration architecture pattern and principles. UK: CDBB; 2021, http://dx.doi.org/10.17863/CAM.68207.

[12] Tao F, Qi Q. Make more digital twins. Nature 2019;573(7775):490–1.

[13] Bolton A, Butler L, Dabson I, Enzer M, Evans M, Fenemore T, et al. Gemini Principles. Centre for Digital Built Britain; 2018, http://dx.doi.org/10.17863/CAM.32260.

[14] Klar R, Arvidsson N, Angelakis V. Digital twins' maturity: The need for interoperability. IEEE Syst J 2024;18(1):713–24. http://dx.doi.org/10.1109/JSYST.2023.3340422.

[15] ETSI Specialist Task Forces (STF) 628. SmartM2M; digital twins communication requirements. TS-103-845-V1.1.1, European Telecommunications Standards Institute (ETSI); 2024.

[16] Autiosalo J, Siegel J, Tammi K. Twinbase: Open-source server software for the Digital Twin Web. IEEE Access 2021;9:140779–98. http://dx.doi.org/10.1109/ACCESS.2021.3119487.

[17] Schreieck M, Ondrus J, Wiesche M, Krcmar H. A typology of multi-platform integration strategies. Inf Syst J 2024;34(3):828–53. http://dx.doi.org/10.1111/isj.12450, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/isj.12450, URL https://onlinelibrary.wiley.com/doi/abs/10.1111/isj.12450.

[18] Singh PM, van Sinderen M, Wieringa RJ. Reference architecture for integration platforms. In: Hallé S, Villemaire R, Lagerström R, editors. 21st IEEE international enterprise distributed object computing conference. IEEE Computer Society; 2017, p. 113–22. http://dx.doi.org/10.1109/EDOC.2017.24, URL DOI: 10.1109/EDOC.2017.24.

[19] Hong J, Dreibholz T, Schenkel JA, Hu JA. An overview of multi-cloud computing. In: Barolli L, Takizawa M, Xhafa F, Enokido T, editors. Web, artificial intelligence and network applications - proceedings of the workshops of the 33rd international conference on advanced information networking and applications. Advances in intelligent systems and computing, vol. 927, Springer; 2019, p. 1055–68. http://dx.doi.org/10.1007/978-3-030-15035-8_103.

[20] Delsing J. IoT automation: Arrowhead framework. Crc Press; 2017.

[21] Giulianelli A, Burattini S, Ciortea A, Ricci A. Engineering interoperable ecosystems of Digital Twins: A Web-based approach. In: Wimmer M, Egyed A, Combemale B, Chechik M, editors. Proceedings of the ACM/IEEE 27th international conference on model driven engineering languages and systems. ACM; 2024, p. 476–85. http://dx.doi.org/10.1145/3652620.3688263.

[22] Fielding RT, Taylor RN. Principled design of the modern Web architecture. ACM Trans Internet Techn. 2002;2(2):115–50. http://dx.doi.org/10.1145/514183.514185.

[23] Karabulut E, Pileggi SF, Groth P, Degeler V. Ontologies in digital twins: A systematic literature review. Future Gener Comput Syst 2024;153:442–56. http://dx.doi.org/10.1016/j.future.2023.12.013.

[24] Kharlamov E, Martin-Recuerda F, Perry B, Cameron D, Fjellheim R, Waaler A. Towards semantically enhanced Digital Twins. In: 2018 IEEE international conference on big data. 2018, p. 4189–93. http://dx.doi.org/10.1109/BigData.2018.8622503.

[25] Kaebisch S, McCool M, Korkan E, Kamiya T, Charpenay V, Kovatsch M. Web of things (WoT) thing description. W3C recommendation, World Wide Web Consortium; 2023, URL https://www.w3.org/TR/2023/REC-wot-thing-description11-20231205/.

[26] Picone M, Mamei M, Zambonelli F. WLDT: A general purpose library to build IoT Digital Twins. SoftwareX 2021;13:100661. http://dx.doi.org/10.1016/j.softx.2021.100661.

[27] Heath T, Bizer C. Linked data: Evolving the web into a global data space. Synthesis lectures on the semantic web, Morgan & Claypool Publishers; 2011, http://dx.doi.org/10.2200/S00334ED1V01Y201102WBE001.

[28] Cool URIs for the semantic web — w3.org. 2025, https://www.w3.org/TR/cooluris/. [Accessed 3 July 2025].

[29] [httpRange-14] Resolved from Roy T. Fielding on 2005-06-19 (www-tag@w3.org from June 2005) — lists.w3.org. 2025, https://lists.w3.org/Archives/Public/www-tag/2005Jun/0039.html. [Accessed 3 July 2025.

[30] Feigenbaum, Williams, Clark, Torres. SPARQL 1.1 protocol, W3C recommendation 21 march 2013. W3C recommendation, World Wide Web Consortium; 2013, URL http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/.

[31] Roffia L, Azzoni P, Aguzzi C, Viola F, Antoniazzi F, Cinotti TS. Dynamic linked data: A SPARQL event processing architecture. Futur Internet 2018;10(4):36. http://dx.doi.org/10.3390/FI10040036.

[32] Burattini S, Mariani S, Montagna S, Picone M, Ricci A. Distributing intelligent functionalities in the Internet of Things with agents and Digital Twins. Internet Things 2025;31:101560. http://dx.doi.org/10.1016/J.IOT.2025.101560.

[33] Mariani S, Picone M, Ricci A. About Digital Twins, Agents, and Multiagent Systems: A Cross-Fertilisation Journey. In: Melo FS, Fang F, editors. Autonomous agents and multiagent systems. best and visionary papers - AAMAS 2022 workshops, virtual event, May 9-13, 2022, revised selected papers. Lecture notes in computer science, vol. 13441, Springer; 2022, p. 114–29. http://dx.doi.org/10.1007/978-3-031-20179-0_8.

[34] Weyns D, Omicini A, Odell J. Environment as a first class abstraction in multiagent systems. Auton Agents Multi Agent Syst 2007;14(1):5–30. http://dx.doi.org/10.1007/S10458-006-0012-0.

[35] Ricci A, Piunti M, Viroli M. Environment programming in multi-agent systems: an artifact-based perspective. Auton Agents Multi Agent Syst 2011;23(2):158–92. http://dx.doi.org/10.1007/S10458-010-9140-7.

[36] Ciortea A, Mayer S, Gandon F, Boissier O, Ricci A, Zimmermann A. A decade in hindsight: The missing bridge between multi-agent systems and the world wide web. In: Elkind E, Veloso M, Agmon N, Taylor ME, editors. Proceedings of the 18th international conference on autonomous agents and multiAgent systems. International Foundation for Autonomous Agents and Multiagent Systems; 2019, p. 1659–63, URL http://dl.acm.org/citation.cfm?id=3331893.

[37] Lagally M, Matsukura R, McCool M, Toumura K, Kajimoto K, Kawaguchi T, et al. Web of things (WoT) architecture. W3C Recommendation, World Wide Web Consortium; 2023, URL https://www.w3.org/TR/2023/REC-wot-architecture11-20231205/.

[38] Ferko E, Bucaioni A, Behnam M. Architecting digital twins. IEEE Access 2022;10:50335–50. http://dx.doi.org/10.1109/ACCESS.2022.3172964.

[39] Barbone A, Burattini S, Martinelli M, Picone M, Ricci A, Virdis A. Digital twin continuum: a key enabler for pervasive cyber-physical environments. In: 2024 33rd international conference on computer communications and networks. 2024, p. 1–9. http://dx.doi.org/10.1109/ICCCN61486.2024.10637565.