Modeling Digital Twin Data and Architecture: A Building Guide with FIWARE as Enabling Technology

Javier Conde
Departamento de Ingeniería de Sistemas Telemáticos, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid

Andrés Munoz-Arcentales
Departamento de Ingeniería de Sistemas Telemáticos, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid

Álvaro Alonso
Departamento de Ingeniería de Sistemas Telemáticos, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid

Sonsoles López-Pernas
Departamento de Ingeniería de Sistemas Telemáticos, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid

Joaquín Salvachúa
Departamento de Ingeniería de Sistemas Telemáticos, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid

***Abstract*—The use of Digital Twins in the industry has become a growing trend in recent years, allowing to improve the lifecycle of any process by taking advantage of the relationship between the physical and the virtual world. Existing literature formulates several challenges for building Digital Twins, as well as some proposals for overcoming them. However, in the vast majority of the cases, the architectures and technologies presented are strongly bounded to the domain where the Digital Twins are applied. This article proposes the FIWARE Ecosystem, combining its catalog of components and its Smart Data Models, as a solution for the development of any Digital Twin. We also provide a use case to showcase how to use FIWARE for building Digital Twins through a complete example of a Parking Digital Twin. We conclude that the FIWARE Ecosystem constitutes a real reference option for developing DTs in any domain.**

Digital Twins (DTs) have emerged with the aim of improving process productivity. The Industrial Internet Consortium (https://www.iiconsortium.org/) defines a DT as "a formal digital representation of some asset, process or system that captures attributes and behaviors of that entity suitable for communication, storage, interpretation or processing within a certain context" [1]. In other words, DTs constitute the virtual counterpart of physical entities in the collaboration between the virtual and physical world [2].

In order to make the use of DTs effective, it is necessary to overcome a series of challenges [3]. First, DTs require to process and model an immense amount of data in batch and in real-time. For this purpose, distributed systems based on cloud computing are needed. Moreover, data acquisition from heterogeneous sources, each with its own format, and the combination of these data sources constitute a significant challenge, as does defining data models and ontologies that represent any entity regardless of its origin. Therefore, communication and connection protocols among all elements that compound a DT infrastructure must be standardized to ensure interoperability. Furthermore, it is necessary to deal with security in all the systems involved in the data exchange, both physical and virtual. DTs involve a large number of components and, in some cases, the information is very sensitive. Consequently, there must be authentication and authorization systems to protect the data, which must be encrypted in all communications. With the aim of addressing these challenges, new technologies and concepts have emerged, constituting the basis of DTs. Some of these are Big Data, Machine Learning, Internet of Things (IoT), and Linked Data.

Substantial research has been devoted to solving some of the challenges that arise from the requirements of DTs. For instance, [4] compared the most extended proposals regarding data model definition. Asset Administration Shell (AAS), defined by Plattform Industrie 4.0, is a proposal developed to exchange information between real-world assets and digital tools [5]. The Open Data Protocol (OData), developed by Microsoft and standardized in its fourth version by the Organization for the Advancement of Structured Information Standards (OASIS), was proposed for building and consuming RESTful APIs [6]. The Next Generation Service Interfaces-Linked Data (NGSI-LD), standarized by ETSI, was developed for managing context information with support to geo-spatial, temporal and historical data [7]. The Web of Things (WoT) group, at W3C, tries to remove the complexity related to the number of IoT standards and technologies by proposing the WoT Thing Description (DT) for modeling IoT resources, termed Things [8]. Each of the alternatives presented allows to model data using one ore more formats among the most widely used in the industry (e.g., XML, AutomationML, RDF, JSON, and JSON-LD).

Furthermore, a number of works have proposed architectures related to the flow of data in a DT. These architectures are, in most cases, strongly bounded to the field where the DT is applied. However, most of them have one thing in common: they are layered-architectures focused on acquiring, modeling, processing, and consuming data. An example of this four-layer architecture is found in [9]. However, in some approaches, the referenced architectures are segregated into more layers [10], by unfolding the layers related to acquiring and modeling data, resulting into six separate layers. In contrast, some other approaches merge two layers together into one single layer, e.g., the data processing and consuming layers [11]. Other differences lie in how each layer is implemented. There are works that propose architectures in which the communication between the physical world and the virtual one is established through events and actions processed by a Cognitive System [12]. Others, such as [13], are based on a central middleware oriented to the exchange of data in real-time by implementing Publish-Subscribe patterns, or distributed architectures based on microservices that implement different functions of the middleware [14]. Other works, such as [15], show architectures based on cloud computing. In addition, there are many proprietary cloud computing solutions (e.g., Microsoft Azure, IBM, GE Digital, and SAP) [16].

Prior works propose FIWARE as a middleware for managing data in a DT [17], [18]. FIWARE (https://www.fiware.org/) is a framework of Open Source components, known as Generic

Enablers (GEs), for facilitating the development and implementation of smart solutions. FIWARE provides all the necessary tools for building DTs. First, it allows for the definition and management of core context data in real-time. It is containerized, highly scalable, and easy to implement and replicate. Moreover, it is supported by the European Commission and the Context Broker GE belongs to the Connecting Europe Facility (CEF) Building Blocks, connecting the so-called "Powered by FIWARE" projects with other recognized European projects. However, the works that propose FIWARE as a middleware, do not delve into how to implement a complete solution using its GEs, but rather limit the role of FIWARE to the integration of the FIWARE Context Broker (for managing context data) and the IoT Agents (for connecting IoT devices to FIWARE infrastructure). The objective of the present article is to extend the definition of a DT reference architecture based on the FIWARE GEs. As a proof of concept, and also as a guide to develop a DT using the FIWARE GEs, we present a use case to demonstrate how architectures and data models based on FIWARE meet the requirements defined in the literature that need to be met in order to implement DTs.

The article is structured as follows. In the next section we present the data modeling based on FIWARE NGSI, followed by a reference architecture based on the FIWARE GEs. Thereafter, these proposals are applied in a use case consisting of a Parking DT. Lastly, the conclusions of the article are presented as well as an outlook of future work.

MODELING DT DATA IN FIWARE

As we have established, one of the cornerstones of designing and developing a DT is modeling data. Data originate from heterogeneous sources, use various protocols and include their own data attributes, attribute types and relationships. Not only is it necessary to standardize the communication among DT components, but standardizing the data format that flows through these components is also required to ensure interoperability. In this regard, FIWARE provides mechanisms for modeling and managing data. It introduces the Next Generation Service Interface (NGSI) and the FIWARE RESTful Application Programmer Interface (API), allowing systems to access the current state of context and perform updates using the NGSI API (https://fiware.github.io/specifications/ngsiv2/stable/).

The main elements of NGSI data are context entities: representations of physical or logical objects. Each of the context entities has a set of attributes and metadata of attributes to represent its real-life properties. Based on context entities, NGSI provides a data model for context information, an interface for exchanging data, and an interface for exchanging information on how to obtain such data. All these core elements are encoded in JSON format.

There is also a version of the NGSI interface that supports Linked Data: the NGSI Linked Data (NGSI-LD). NGSI-LD is an evolution of NGSI that supports formal relationships, semantics and property graphs using JSON-LD as the encoding method. The main goal of Linked Data is creating a machine-readable web of data in which each entity and relationship has a Uniform Resource Name (URN), allowing to access new sources of information by forming graphs of knowledge. NGSI API interactions are made using HTTP/HTTPs requests. Clients use HTTP methods for creating, updating, deleting and querying context information.

The FIWARE community has already standardized many data models, from different domains, known as FIWARE Smart Data Models (https://github.com/smart-data-models/), with the objective of being portable for different solutions. Nowadays, there are six domains related to smart cities, smart environment, smart sensoring, smart agrifood, smart energy, smart water, and cross sector. Other domains, such as robotics, will be added in near future. Smart Data Models are compatible with both NGSIv2 and NGSI-LD. They are proposed and maintained by the community, and users are able to extend them for their specific use case.

MODELING DT ARCHITECTURE IN FIWARE

The centerpiece of any platform or solution "Powered by FIWARE" is the FIWARE Context Broker, bringing a cornerstone function in any smart solution: the need to manage, update and access context information. In the specific scope of DTs, the Context Broker provides a liaison between the physical and virtual world, acting as an intermediary in any exchange of information between the two domains or even within the same domain. Building around the FIWARE Context Broker, a rich suite of complementary components (the FIWARE Generic Enablers) are available, structured in three layers (**Figure 1**): (1) Interface with the Internet of Things, Robots and third-party systems, for capturing updates on context information in the physical world and translating required actuations; (2) Context Data/API management, enabling to perform updates, queries or subscribe to changes on context information; and (3) Processing, analysis, and visualization of context information, implementing the expected smart behavior of applications in the virtual world. The remaining of this section describes the proposed DT Reference Architecture, presenting all the FIWARE GEs involved and the relationship among them.

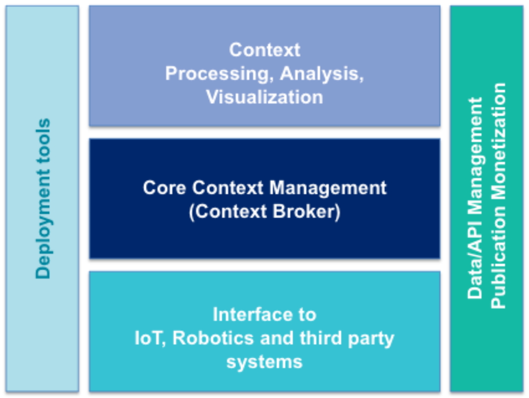
Deployment tools
Context Processing, Analysis, Visualization
Core Context Management (Context Broker)
Interface to IoT, Robotics and third party systems
Data/API Management Publication Monetization

Figure 1. FIWARE GE catalogue. From: https://www.fiware.org/developers/catalogue/, with permission.

Management of context data

As we have stated before, in any "Powered by FIWARE" platform or solution, the Context Broker Generic Enabler is the core and mandatory component. The Context Broker GE enables to manage context information in a highly decentralized and large-scale manner. Currently, FIWARE offers four implementations of the Context Broker: Orion Context Broker, Orion-LD Context Broker, Scorpio Broker, and Stellio Context Broker. These GEs are capable of managing the different versions of the NGSI standard (https://www.fiware.org/developers/catalogue/). In this article, we focus our architecture proposal around the Orion Context Broker, the most extended GE, and the one endorsed by the European Commission as a CEF Building Block.

The **Orion Context Broker** (https://fiware-orion.readthedocs.io/) is the GE that manages context information through the implementation of a publish-subscribe pattern providing a NGSI interface. Clients can create context elements, query and update them, and subscribe to changes in context information for receiving notifications. Other elements interact with Orion through HTTP/HTTPs requests. Orion only saves the latest information state. Thus, if it is required to store the information history, other GEs have to be used.

Data acquisition and persistence

A DT needs to deal with multiple sources of data coming form physical devices. Hence, the data need to be transformed before they can be sent to the Context Broker. Fortunately, FIWARE provides the IOT Agents GE to take care of this task.

The **IoT Agents** (https://github.com/FIWARE/catalogue/blob/master/iot-agents/README.md) GE is a set of software modules handling South IoT Specific protocols and North OMA NGSI interaction. These agents allow to work with the IoT devices that use communication protocols like LWM2M over CoaP, JSON or UltraLight over HTTP/MQTT, OPC-UA, Sigfox, or LoRaWAN. By using this component, the IoT devices will be represented in a FIWARE platform as NGSI entities in a Context Broker.

Additionally, the IoT agents GE allows to trigger commands to actuation devices just by updating specific command-related attributes in their NGSI entities representation at the Context Broker.

Furthermore, other sources of data can feed DTs. On the one hand, there are REST APIs, or Database systems that need to be queried periodically. On the other hand, there are systems like TCP or HTTP servers which send a continuous flow of data. These data, possibly coming from external applications, also need to be converted into NGSI and posted in the Context Broker. In order to deal with this type of systems that are not covered by the IoT Agents, FIWARE provides the Draco GE.

Draco (https://fiware-draco.readthedocs.io/) is a dataflow management system based on Apache NiFi (https://nifi.apache.org/) that supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic using a set of processors and controllers. The suite of processors and controllers provided in this GE allow to convert the incoming data into NGSI entities and attributes, required for their publication in the Context Broker.

Moreover, the Draco GE is used not only in the data acquisition block in the DT architecture, but also in the persistence part. Draco includes a set of processors in charge of persisting NGSI context data in third-party storage systems, allowing to create a historical view of the data posted in the CB.

Data analysis

In the scope of a DT, the data gathered from physical entities can be processed to extract relevant information or to draw conclusions that can influence, optimize and/or control any of the components of the DT without the need of human intervention. A DT equipped with such capabilities is known as an Intelligent DT [19]. In order to implement an Intelligent DT, Big data engines are required so as to handle large amounts of data.

The **Cosmos GE** (https://fiware-cosmos-flink.readthedocs.io) provides an interface for integrating two of the most well-known Big Data processing engines (namely, Apache Flink and Apache Spark) with the rest of the components of the FIWARE Ecosystem. This GE consists of a set of connectors that enable the reception of data from the Context Broker (through its subscription/notification feature) directly within data processing jobs running in these engines, allowing to process context data in real-time, as well as to send the result from this processing back to a given entity in the Context Broker so other components in the architecture can subscribe to changes in its attributes.

Security architecture

In the FIWARE Ecosystem, there are a number of GEs that manage authorization and authentication with regard to users and devices. In general, every request sent to a FIWARE GE has to be authenticated using a token. The **Keyrok GE** (https://fiware-idm.readthedocs.io/) is an Identity Management component based on OAuth 2.0. It is in charge of issuing the tokens for previously registered and authenticated subjects (users and devices). The **Wilma PEP Proxy GE** (https://fiware-pep-proxy.readthedocs.io/) acts as a Policy Enforcement Point (PEP), intercepting the aforementioned requests and validating the tokens to ensure the fulfillment of access control policies based on previously defined roles and permissions for subjects. Roles and permissions are defined and assigned by services' owners. In complex scenarios, where advanced policy definition is needed, the XACML-based Policy Decision Point (PDP) **AuthZForce GE** (https://authzforce-ce-fiware.readthedocs.io/) should be used.

USE CASE OF A DT MODELED WITH FIWARE

In this section, we present a use case that depicts the guidelines to build a DT based on the FIWARE Ecosystem through an example of a Parking DT. This DT will be used to improve traffic in a parking lot. It will help customers find their vehicle through a 3D mobile application and predict the parking expected occupancy throughout a given day. The system collects real-time data from several sensors distributed around the parking space and data from an external API that provides weather information used in predictions.

The process of building the DT will be divided into four phases: data modeling, data acquisition, data consumption and security aspects. In **Figure 2** the general architecture of the solution is presented.

Data modeling

For our system, we selected four different models, from the Smart Data Models catalog, to represent entities of type `OffStreetParking`, `ParkingSpot`, `Vehicle` and `WeatherForecast`. The first one models the whole parking; the second, a

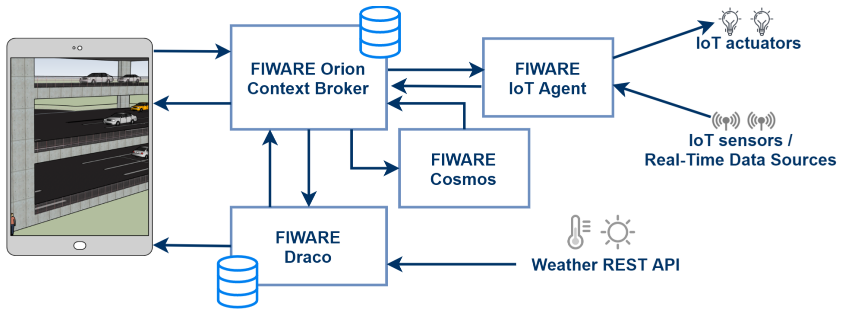

Figure 2. Simplified architecture of the Parking DT in which security GEs have been omitted for the sake of simplicity

parking place; the third, a customer's vehicle; and the fourth, the weather forecast for the next hour.

Data acquisition

Since the data originate from different sources, it is necessary to connect them through different protocols and adapt the information to the corresponding data models. We receive data from IoT sensors that report information of the vehicles and the place that they occupy in the parking. In this use case, sensors communicate through the Ultralight 2.0 protocol. An example of the payload when a vehicle parks is `id|123456|t|car|p|51`. This payload represents a **Vehicle** entity of type `car` with plate number `123456` that is stationed in the parking spot number `51`. The respective IoT Agent receives this message, transforms it into the necessary entities in the NGSIv2 format and updates the context information sending three HTTP update requests to Orion. For the sake of simplicity, the samples presented in this use case do not include all the required attributes of the model (e.g. `location`). However, a complete solution must include them. The first request creates the entity of type `Vehicle` that represents this car:

```
{
    "id": "vehicle:501",
    "type": "Vehicle",
    "vehicleType": "car",
    "vehiclePlateIdentifier": "123456"
}
```

The second one, updates the parking place 51, entity of model `ParkingSpot`, with a `status` occupied:

```
{
    "id": "spot:51",
    "type": "ParkingSpot",
    "name": "51",
    "status": "occupied",
    "refVehicle": "vehicle:501",
    "refOffStreetParking": "parking:1"
}
```

Lastly, the third one decreases the `availableSpotNumber` of the `OffStreetParking` entity:

```
{
    "id": "parking:1",
    "type": "OffStreetParking",
    "availableSpotNumber": 1449
}
```

The remaining data source is the weather API, which provides the weather forecast. Below is an example of these data:

```
{
    "temp":27.50,
    "tempmin":27.08,
    "tempmax":27.60,
    "precipitation": 0.56,
    "wind":{
        "speed":1.5
    },
}
```

In this case, Draco retrieves the data periodically from the API and transforms them to fit the `WeatherForecast` model:

```
{
    "id": "weatherForecast:2020-08-03T09",
    "type": "WeatherForecast",
    "validFrom": "2020-08-03T09:00:00.00Z",
    "validTo": "2020-08-03T10:00:00.00Z",
    "temperature": 27.50,
    "precipitationProbability": 0.56,
    "dayMaximum": {
        "temperature": 27.60
    },
    "dayMinimum": {
        "temperature": 27.08
    },
    "windSpeed": 1.5,
}
```

For this purpose, Draco uses two types of processors: one called InvokeHTTP, that makes the polling to the API, with a period of one minute, and another called JoltTransformJSON, which transforms the incoming JSON into the NGSIv2 format. Lastly, another InvokeHTTP processor sends an HTTP request to Orion and updates the context information.

Data consumption

Orion provides the latest version of context information for the entities described and the other components subscribe to the changes in the entities they are interested in. Therefore, the 3D application will send an HTTP POST request to Orion in order to subscribe to changes in the status attribute of `ParkingSpot` entities. Consequently, when there is any change in the application the 3D frontend will receive HTTP requests from the Context Broker and it will be updated by adding or deleting the respective vehicles. Cosmos is subscribed to changes in the Context Broker too, specifically to the `WeatherForecast` entity and the `ParkingSpot` status attribute. With these data and some additional information (e.g., the hour of the day, the day of the week) Machine Learning algorithms can be applied to make useful real-time predictions for the parking such as the expected number of vehicles.

Finally, when changes in `ParkingSpot` entities take place, the IoT Agent receives a notification, processes it, and sends an action to update the light color of the smart bulb situated in the parking place. The chosen color code is red for closed parking place, yellow for occupied, and green for free. In this case, the bulb changes from green to yellow, and consequently, the payload of the Ultralight 2.0 sent message is `bulb:0051@light|yellow`.

Storing historical data is of the utmost importance in our DT because historical data are used to train Machine Learning algorithms and periodically generate reports. Consequently, Draco subscribes to any changes in the context information and saves all data in a Mongo database using the ListenHTTP and NGSIToMongo processors.

Security aspects

Thus far, we have avoided talking about security in our Parking DT for the sake of simplicity, but it is indeed a topic that needs to be addressed. Confidentiality and integrity of data are fundamental aspects of DTs. Hence, it is necessary to secure the access to the Context Broker by managing roles and permissions. For that purpose, FIWARE provides two GEs that make this task easier. These are FIWARE KeyRock and FIWARE Wilma. We are going to focus on the access control of the 3D mobile application, but this proposal could be extended to the rest of the components in the architecture. **Figure 3** represents the extended architecture including security GEs.

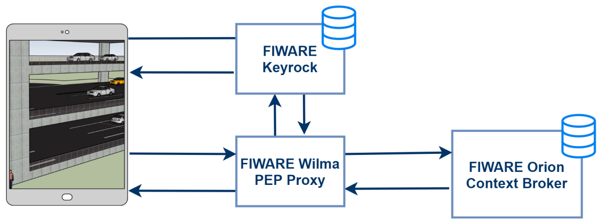

Figure 3. Extended architecture of the Parking DT with security GEs

We can distinguish three types of roles (i.e., admin, supervisor, and general user). Moreover, there are three main permissions (i.e., creating and deleting supervisors, updating the status of `ParkingSpot` entities, and retrieving the `ParkingSpot` entities). The admin has all three permissions, the supervisor can update the status of `ParkingPost` entities and retrieve them. The general user is only allowed to retrieve the `ParkingSpot` entities. These roles and permissions are created and assigned by the admin through the Keyrock interface or Keyrock REST API. Users that are not authenticated are redirected to Keyrock in order to sign in using any of the OAuth 2.0 grant flows available. The permissions are stated as an HTTP action an a resource. For example, the second permission is configured by the combination of action POST and resource */parkingSpot*.

FIWARE Wilma allows to secure the access to Orion, hiding its real address from external users. The parking users do not request information directly to the Context Broker, but instead send messages to Wilma, Wilma checks if the users have the required permissions and, if they do, it redirects the requests to Orion.

CONCLUSIONS AND FUTURE WORK

The success of DTs in today's industry, and consequently, the growth of their application in an increase number of fields, makes it necessary to standardize the way they are implemented, both from the components' side and that of the data flow. In this article we proposed FIWARE as a complete solution for building DTs, not only theoretically, but also practically through a use case of a Parking DT. On the basis of this work, it can be stated that FIWARE GEs and FIWARE Smart Data Models overcome the challenges formulated in the literature related to real-time and batch data consuming and processing, scalability, cloud computing, data modeling, and security. From our research we can conclude that FIWARE constitutes a legitimate reference option for the easy development of DTs of any kind. Moreover, being FIWARE GEs open source software, their evolution and adaptation to new requirements, technologies and standards is ensured thanks to FIWARE Community.

As future work, the reference architecture should be validated with more use cases and the Smart Data Models repository needs to be expanded with additional domains. In the future we will propose the adaptation of this work to the NGSI-LD standard, using another Context Broker rather than Orion, and exploiting the advantages of Linked Data. Finally, as explained before, the proposed architecture is agnostic to the technology used in the DT visualization. However, it could be interesting to explore the benefits of using different approaches with regard to the visualization part.

REFERENCES

1. M. Buchheit, A. Ferraro, C. Lim, S.-W. Lin, J. Morrish, and B. Zarkout, "Digital Transformation in Industry White Paper," Industrial Internet Consortium, 2020. [Online]. Available: https://www.iiconsortium.org/pdf/Digital_Transformation_in_Industry_Whitepaper_2020-07-23.pdf
2. E. Negri, L. Fumagalli, and M. Macchi, "A review of the roles of Digital Twin in CPS-based production systems," *Procedia Manufacturing*, vol. 11, pp. 939–948, 2017.
3. T. H. J. Uhlemann, C. Lehmann, and R. Steinhilper, "The Digital Twin: Realizing the Cyber-Physical Production System for Industry 4.0," in *Proc. 24TH CIRP Conf. on Life Cycle Engineering*, S. Takata, Y. Umeda, and S. Kondoh, Eds., vol. 61, 2017, pp. 335–340.
4. M. Jacoby and T. Usländer, "Digital Twin and Internet of Things—Current Standards Landscape," *Applied Sciences*, vol. 10, no. 18, 2020.
5. "The Asset Administration Shell: Implementing digital twins for use in Industrie 4.0," Plattform Industrie 4.0, 2019. [Online]. Available: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/VWSiDV2.0.pdf
6. M. Pizzo, R. Handl, and M. Zurmuehl, "OData Version 4.01. Part 1: Protocol," Organization for the Ad-vancement of Structured Information Standards, OASIS Standard, 2020. [Online]. Available: https://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-protocol.pdf
7. European Telecommunications Standards Institute, "Context Information Management (CIM); NGSI-LD API," ETSI-GS-CIM-009, 2020. [Online]. Available: https://www.etsi.org
8. V. Charpenay, T. Kamiya, M. McCool, S. Käbisch, and M. Kovatsch, "Web of Things (WoT) Thing Description," W3C, W3C Recommendation, Apr. 2020. [Online]. Available: https://www.w3.org/TR/2020/REC-wot-thing-description-20200409/
9. S. Malakuti, J. Schmitt, M. Platenius-Mohr, S. Grüner, R. Gitzel, and P. Bihani, "A Four-Layer Architecture Pattern for Constructing and Managing Digital Twins," in *Software Architecture, 13th European Conference, ECSA 2019*, ser. Lecture Notes in Computer Science, vol. 11681, 2019, pp. 231–246.
10. A. Redelinghuys, K. Kruger, and A. Basson, "A Six-Layer Architecture for Digital Twins with Aggregation," in *Service oriented Holonic and Multi-Agent Manufacturing*, T. Borangiu, D. Trentesaux, P. Leitão, A. Giret Boggino, and V. Botti Navarro, Eds., 2020, pp. 171–182.
11. V. Souza, R. Cruz, W. Silva, S. Lins, and V. Lucena Jr, "A Digital Twin Architecture Based on the Industrial Internet of Things Technologies," in *Proc. IEEE Int.Conf. Consum. Electron. (ICCE)*, 2019, pp. 1–2.
12. T. Gabor, L. Belzner, M. Kiermeier, M. Beck, and A. Neitz, "A Simulation-Based Architecture for Smart Cyber-Physical Systems," in *Proc. 2016 IEEE Inter. Conf. on Automotic Computing (ICAC)*, 2016, pp. 374–379.
13. S. Yun, J.-H. Park, and W.-T. Kim, "Data-centric Middleware based Digital Twin Platform for Dependable Cyber-Physical Systems," in *Proc. 2017 Ninth Inter. Conf. on Ubiquitous and Future Networks (ICUFN 2017)*, 2017, pp. 922–926.

14. M. Ciavotta, M. Alge, S. Menato, D. Rover, and P. Pedrazzoli, “A Microservice-based Middleware for the Digital Factory,” *Procedia Manufacturing*, vol. 11, pp. 931–938, 2017.
15. K. M. Alam and A. El Saddik, “C2PS: A Digital Twin Architecture Reference Model for the Cloud-Based Cyber-Physical Systems,” *IEEE Access*, vol. 5, pp. 2050–2062, 2017.
16. R. Minerva, G. M. Lee, and N. Crespi, “Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models,” *Proceedings of the IEEE*, vol. 108, no. 10, pp. 1785–1824, 2020.
17. G. N. Schroeder, C. Steinmetz, C. E. Pereira, and D. B. Espindola, “Digital Twin Data Modeling with AutomationML and a Communication Methodology for Data Exchange,” *IFAC-Papersonline*, vol. 49, no. 30, pp. 12–17, 2016.
18. C. Steinmetz, A. Rettberg, F. Ribeiro, G. Schroeder, and C. Pereira, “Internet of Things Ontology for Digital Twin in Cyber Physical Systems,” in *Proc. of SBESC 2018*, 2018, pp. 154–159.
19. B. Ashtari Talkhestani, T. Jung, B. Lindemann, N. Sahlab, N. Jazdi, W. Schloegl, and M. Weyrich, “An architecture of an Intelligent Digital Twin in a Cyber-Physical Production System,” *at - Automatisierungstechnik*, vol. 67, no. 9, pp. 762–782, 2019.

Javier Conde is currently pursuing a PhD in Telematics Engineering at UPM and a researcher in the Department of Telematics Engineering. His research interests lie in the fields of Open Linked Data, Digital Twins, Big Data and Machine Learning.

Andrés Munoz-Arcentales is currently working as a researcher in the Department of Telematics Engineering at UPM and studying a Ph.D. in Telematics Engineering at the same university, with a major research interests in Data Engineering, Data Usage Control and Machine Learning in Smart Spaces.

Álvaro Alonso is currently an Assistant Professor with the UPM. His research interests include Multi-videoconferencing Systems, Security Management in Smart Context scenarios and Public Open Data.

Sonsoles López-Pernas (GSM'19) is pursuing her PhD in Telematics Engineering at UPM. Since 2015, she has worked as a researcher in the Department of Telematics Engineering at UPM, participating in different projects such as FIWARE and eid4U.

Joaquín Salvachúa is currently an Associated Professor with UPM. His research interests include Advanced Cloud and Edge architectures, Big Data infrastructure, Data Privacy and usage control, NoSql Databases, applications and identity in Blockchain.