Research Report: A Model-Driven Development Method For Digital Twins

Emilio Carrión[1,2][0000−0002−7026−0495], Pedro Valderas[2][0000−0002−4156−0675], and Óscar Pastor[2][0000−0002−1320−8471]

[1] Mercadona Tech, Mercadona
[2] PROS – VRAIN, Universitat Politècnica de València

1 A MDE Approach to Develop Digital Twins

This model-driven development method for Digital Twins (DTs) has been conceived following the MDE principles that aim to mitigate development challenges by elevating multiple models to primary development artefacts to derive DTs. By adopting an MDE approach, enterprises can use the acquired knowledge and experience to optimise their software development processes. This, in turn, improves the stability and maintainability of the delivered solutions and provides customers with a faster development process [23,24].

Our method allows us to define and instance conceptual specifications into specific software artefacts. This method proposes a PIM (Platform-Independent Model) based on ERDT [16] in order to define DTs, and a model transformation to automatically obtain a preliminary PSM (Platform-Specific Model) based on WoT models. Afterwards, we complement the PSM with specific implementation details and instance it in a runtime platform. In addition, we also give recommendations to which professional roles intervene in each step. We can see a diagram description of this method in Fig. 1.

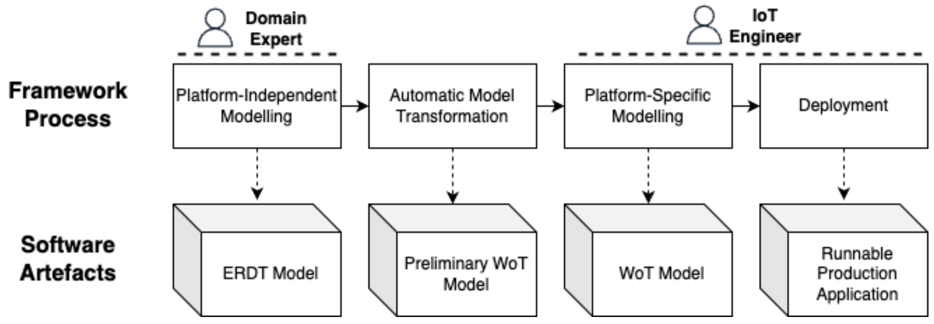

Domain Expert
IoT Engineer
Framework Process
Platform-Independent Modelling
Automatic Model Transformation
Platform-Specific Modelling
Deployment
Software Artefacts
ERDT Model
Preliminary WoT Model
WoT Model
Runnable Production Application

Fig. 1. DT Development Method.

The proposed method comprises the following steps.

Step 1: Platform-Independent Modelling. We instantiate a specific ERDT model of a DT. This model describes at the conceptual level and independently from any technology the main characteristics of the DT that is being developed. This step is meant to be carried out by Domain Experts or Business Analysts that have the domain knowledge to model the DT to comply with its requirements.

Step 2: Platform-Specific Model generation through an automatic Model Transformation. With our DT conceptually modelled, we transform it to a preliminary WoT model.

To ease the process, we propose the implementation of an automatic model transformation using a text template-based approach and established software tools. This implementation allows us to translate instances of ERDT models into preliminary WoT models automatically, thus reducing the complexity in specific technologies such as the WoT definition format.

Step 3: Platform-Specific Modelling completion. We complete our preliminary WoT model with implementation details that have not been defined in the automatic process, due to missing information or not being a structured automatable process. This step and the next one are meant to be done by IoT Engineers that have the implementation detail knowledge to configure the devices as needed.

An example of this is configuring the connection details of the real physical devices our DT represents and specifying them in the WoT Description Models.

Step 4: Deployment. WoT descriptions can be directly interpreted by DT platforms in order to obtain a fully operative implementation. Thus, once the WoT model has been completed, the last step of the proposed method is its deployment in one of this runtime platforms. In this paper, we leverage this step to present a proof-of-concept validation of the proposed MDD method through the implementation of a DT for the running example of this paper in the Ditto platform [3].

With the use of this development method based on a MDE approach, we can define a DT independently of specific domains in the wide variety of fields where DTs are being implemented. We are also able to transfer them to WoT models that allow us to quickly deploy and validate them.

In the following sections, we will review and explain each step in detail.

2 Platform-Independent Modelling: ERDT

The first step of the proposed method is the conceptual modelling of our DT. We support this modelling on the Entity-Relationship Digital Twin (ERDT) model [16], a contribution presented in previous work that extends the well-known Entity-Relationship (ER) model [17] to model DTs of physical systems and the specific characteristics of DTs.

The main contributions of ERDT models are the definition of historical values (as DTs often rely on historical data for operational functionalities) and the introduction of interfaces and data flows, novel characteristics that ensure real-time synchronisation between the physical and virtual components of the DT. A diagram example of an ERDT model can be seen on Fig. 2.

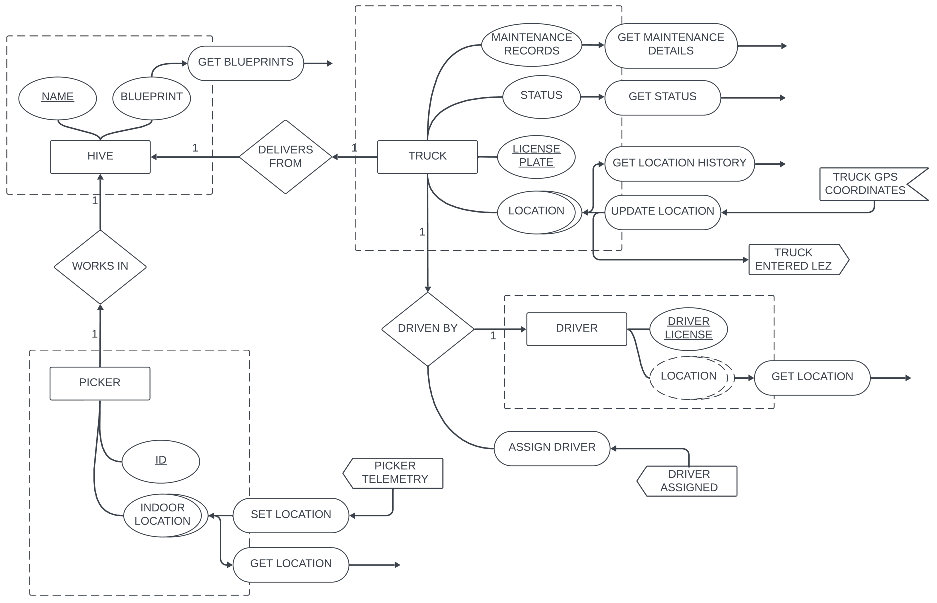


Fig. 2. ERDT Model Diagram Example.

ERDT models are defined with the following components.

Entity Sets and Relationships Entities represent the physical objects we are digitising in our DT. Relationships show how entities are related between them.

Entities, as in the base ER model, have attributes. However, our modelling approach introduces support for features characteristic to DTs, such as attributes that need to keep a historical record of their values. For example, a Truck entity may include a Location History attribute that tracks the past locations of the truck. This property is key for the right representation of the temporal aspects of a DT and enables more complex queries and analyses over time.

Value Sets Value sets are used in ER models to define a set of values that can be assigned to the attribute of one entity. We use them for the same purpose in the context of ERDT.

Interfaces Interfaces act as controlled access points to entities and their attributes, hiding internal complexity and facilitating interaction with the DT.

They are especially useful in systems that manage heterogeneous data sources, as they offer a single access point for fetching or updating information related to an entity, abstracting away the complexity of interacting with multiple systems.

They also allow for the implementation of security and privacy restrictions, controlling which users can access and modify information, providing a first layer of security for data, as they explicitly define what can be done with the data. Any operations not supported or defined by the interface are not allowed, ensuring secured access to sensitive information.

They are represented in the model as capsule-shaped elements that connect attributes with data flows. Some examples shown in Figure 2 are updating the location of a truck or getting the current status of the vehicle.

Data Flows Data flows represent the bidirectional communications between the DT and the physical components. They keep updated the virtual entities with real-time data and allow to provide feedback to the physical entities. They are represented in the model as arrow-like elements that connect the physical world with interfaces and vice versa.

There are three main types:

– Incoming Events: Sent by the physical entities to notify changes in their state. Examples: Driver assigned, Picker telemetry.
– Data Requests: Sent from the DT to the physical entities to query their current status. They are executed periodically or on demand. Example: Truck GPS Coordinates.
– Outgoing Events: Sent from the DT to the physical components to execute actions or send commands. They let the DT to influence the physical system. Examples: sending a warning to the truck driver. Example: Truck Entered LEZ (Low-Emissions Zone)

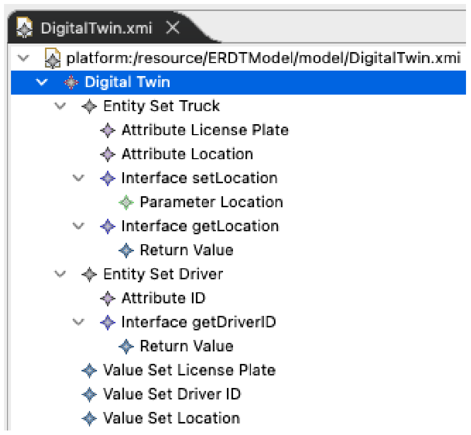
In short, interfaces and data flows enable secure and structured information management in the ERDT model. They facilitate bidirectional communication between the DT and the physical components, enabling real-time synchronisation and the ability to perform analysis and actions based on the collected data.

2.1 Tool Support

To instance the ERDT model, we have used the Ecore standard, which is part of the Eclipse Modelling Framework (EMF) [4]. We have used the Obeo Designer [9] software that is based on Eclipse Sirius [5].

Using these tools we have defined the metamodel which is composed of the elements described in this section. The complete implementation can be found in a Github repository [18].

As a representative example, we have modelled a real scenario of Mercadona Tech's productive logistics environment, where a fleet of delivery trucks, their drivers and locations are represented. To do so, we have used an standard tree editor for Ecore models. As an ongoing work, we are working on the development of a visual editor for ERDT models. A partial view of the resulting model can be seen in Fig. 3.



```
DigitalTwin.xmi
platform:/resource/ERDTModel/model/DigitalTwin.xmi
  Digital Twin
    Entity Set Truck
      Attribute License Plate
      Attribute Location
      Interface setLocation
        Parameter Location
      Interface getLocation
        Return Value
    Entity Set Driver
      Attribute ID
      Interface getDriverID
        Return Value
    Value Set License Plate
    Value Set Driver ID
    Value Set Location
```

Fig. 3. ERDT Model of a delivery truck fleet.

As shown, this step involves conceptual modeling of our DT. In the following steps, we transform it into WoT models and instantiate them in a runtime environment.

3 Platform-Specific Model generation through an automatic Model Transformation

The second step of the proposed method is to automatically transform ERDT models into WoT-based descriptions. To present this, we first introduce the WoT standard and then explain how WoT descriptions are automatically derived from ERDT models.

3.1 WoT Models

WoT is an abstraction about IoT that seeks to solve interoperability problems with the use of standard web technologies. In practice, IoT focuses on connecting devices through different network protocols and WoT focuses on the application layer by providing web interfaces to access and manage these devices.

It was first proposed in 2008 [19] with the aim of implementing interoperability in IoT-based scenarios by creating web technologies that represent these systems, also referred to as Web Things (WTs). It is also included in the World Wide Web Consortium (W3C) with the WoT standard [13] and, despite the recent appearance, there is a growing interest towards it in the field of DTs, as demonstrated by the increasing number of implementations [21], [25], tools

[15], [20], and interest groups [12]. Also, the W3C identifies DTs as one of the WoT use cases [14] and important companies such as Microsoft[3] and Siemens [8] participated in its definition.

The main concept in WoT is the Web Thing (WT) that represents the abstraction of a physical IoT device. Each WT exposes the characteristics of the device through standard web interfaces using RESTful APIs, which allows IoT devices to be accessible and controllable regardless of their communication protocols or their specific characteristics.

WTs are defined by Thing Descriptions (TD), a document that describes the characteristics, capabilities, and methods of interaction of a WT in a specific format, typically JSON-LD (an example is shown in Listing 1.1). The basic elements of a TD are its *properties*, which are interactions that expose state of the Thing, *actions*, that allow to invoke a function which manipulates state or triggers a process, and *events*, interactions that describe an event source which asynchronously pushes event data to consumers.

```json
{
  "name": "Truck",
  "@id": "urn:dev:wot:com:example:Truck",
  "properties": {
    "speed": {
      "writable": true,
      "type": { "type": "integer" },
    }
  },
  "actions": {
    "setSpeed": {
      "inputSchema": { ... },
    }
  },
  "events": {
    "overheatedEngine": {
      "schema": { ... },
    }
  }
}
```

Listing 1.1. WoT Thing Description Example.

The W3C's standardization of WoT has been key to facilitating the integration of IoT devices into web applications. This has enabled the development of interoperable ecosystems where devices from different manufacturers can communicate and collaborate, reducing the fragmentation typical in the IoT space.

[3] Microsoft created its Digital Twin Description Language (DTDL) [2] which is very similar to WoT and is used in Azure platforms, however, today work is being done on its integration back to WoT [11].

For our method, great part of this specification is automatically generated with the model transformation presented in the following section. Afterwards, in the third step, the model is completed with specific details as the devices' connection settings.

3.2 From ERDT to WoT

In this section, we present a model transformation to automatically generate WoT models from ERDT models. It has been implemented as an unidirectional exogenous transformation [22] between concepts defined in the ERDT metamodel (and presented in Section 2) and concepts defined in the WoT metamodel (presented in the previous subsection). To do this, we have defined the transformation mappings between ERDT and WoT concepts that are explained in detail below. Also, we have implemented them by using the Acceleo [1] technology in order to automate their application.

The transformation rules from ERDT and WoT are the following.

- *Entity is a Thing.* The *entities* in ERDT represent the physical elements that make up our DT, the classes that characterize each of the different entities such as the example of a truck. This concept maps to that of *Thing* in WoT which refers to an abstraction of a physical IoT device.
- *Attributes are Properties. Attributes* are the different data associated with an entity set. They represent internal values such as the coordinates of a truck. In WoT this concept fits well with *properties*, which exposes state of the Thing.
- *Value Set is a Type.* In ERDT, *value sets* are the different types of data that are associated with entity attributes and interface parameters. In WoT this concept is mapped to that of *type*, which defines the set of values valid for a specific property.
- *Interfaces are Actions.* In ERDT, we communicate with and update entities by using *interfaces* that receive or return values to query or modify entities. In WoT we have the *actions*, they allow to invoke a function of the Thing, which manipulates state or triggers a process on the Thing.
- *Data Flows are Events.* Finally, in ERDT, *data flows* are the bidirectional data communications between the virtual and physical entities, which allow them to be interconnected in real time to stay updated and thus be able to offer feedback to the physical entity based on the information we have in the DT. In WoT this concept can be mapped to that of *events*, they describe an event source, which asynchronously pushes event data to consumers.

Implementation To facilitate the transformation between ERDT-based conceptual modelling and WoT-based specific modelling, we have created a system of templates that allow us to generate WoT models automatically.

For this, we have used standard and open-source tools that allow us to define and build the different pieces necessary to apply the model transformation. We have used Eclipse Acceleo [1], a code generator from the Eclipse Foundation that implements the standard MOF Model to Text Transformation Language (MOFM2T) [7].

Acceleo allows us to define templates to transform EMF-based models like the one we have obtained in Section 2 and thus generate code from the conceptual definitions of the model. To transform from ERDT to WoT we have defined templates based on the mappings presented above. We can see a snippet of them in the Listing 1.2. The complete implementation can be found in a Github repository [18].

```
[template public generateWoT(dt : DigitalTwin)]

[for (entitySet : EntitySet | dt.entitysets )]
{
  "@context": [’[’/]
    "https://www.w3.org/2022/wot/td/v1.1"
  ],
  "@type": "tm:ThingModel",
  "title": "[entitySet.name/]",
  "properties": {
    [for (attribute : Attribute | entitySet.attributes)]
      "[attribute.name/]": { ... }
    [/for]
  },
  "actions": {
    [for (interface : Interface | entitySet.interfaces)]
      "[interface.name/]": { ... }
    [/for]
  },
  "events": { ... }
}
[/for]

[/template]
```

Listing 1.2. ERDT to WoT models transformation template.

With the use of Acceleo and the DT model obtained using the Obeo Designer suite we obtain a preliminary Thing Model represented in WoT, such as the one presented in Listing 1.1.

4 Platform-Specific Modelling Completion: WoT Models

Once we have the preliminary WoT model, the last phase in the modelling of our DT is to complete it with specific details of the physical devices that can’t be implemented in an automatic way.

These final details are the base URI that is used to reference the Thing and the device connection data. In the WoT standard, this data connection is defined through the "forms" property, which defines the elements required by the DT platform to communicate and run commands on the physical devices. They are defined with the target IRI and the operation that indicates the semantic intention of performing the operation described by the form. The model completion is highlighted in the Listing 1.3 through the completion of properties. Actions and events are completed in the same manner.

```
{
  "name": "Truck",
  "@id": "urn:dev:wot:com:example:Truck",
  "base": "https://example.com/",
  "properties": {
    "speed": {
      "forms": [{ "href": "/truck/speed", ... }]
    }
  },
}
```

Listing 1.3. WoT Thing Description Completion.

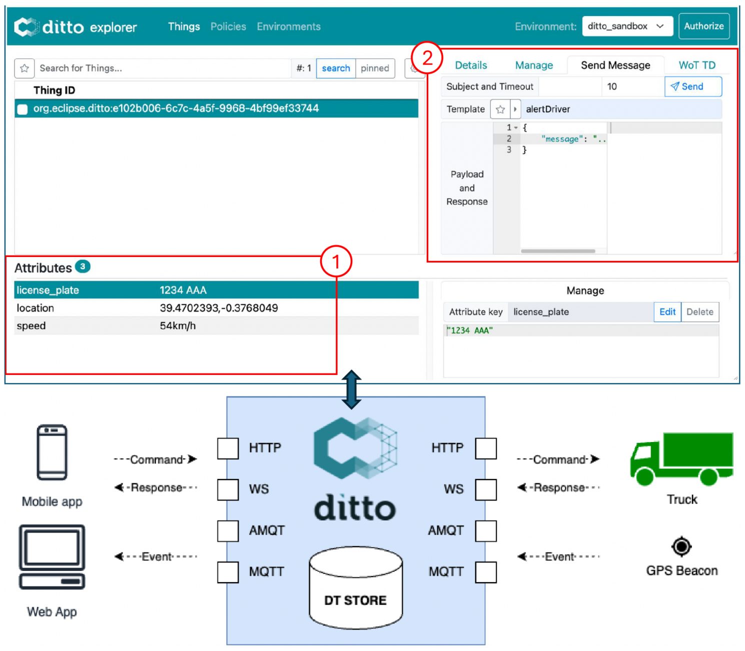
Finally, with this platform-specific detail completion, we end up with a complete WoT model that we can instance in runtime platforms. We can also validate the resulting model using open tools of the WoT standard such as EdiTDor [6].

5 Deployment and Proof of Concept Validation

To validate and demonstrate the value that this model-driven development method brings, the last step is to instantiate the resulting WoT model on a runtime platform. As WoT is a standard defined by the W3C it is directly interpretable by DT platforms such as Eclipse Ditto [3] or the Onesite platform [10]. These platforms allow us to instantiate DTs from model specifications like the one obtained in the previous sections, they also offer interfaces and UI applications to view and manage our DT internals. We have chosen Eclipse Ditto because it is an open-source framework. It allows us to instantiate our truck fleet model into an executable and productive environment with ease. This also allows us to perform a proof-of-concept validation in such a way we can demonstrate how a fully operative DT is obtained from the ERDT model created in the first step.

Ditto offers us a dashboard-like UI interface where we can define devices based on definition models. As mentioned before, Ditto can also integrate with WoT models making the implementation an easy and streamlined process.

Figure 4 shows the user interface that the Ditto platform generates to interact with the Truck entity defined in the ERDT model. As we can see, it shows data of its properties (license plate, location, speed, see numbered area 1) in real time



Thing ID
org.eclipse.ditto:e102b006-6c7c-4a5f-9968-4bf99ef33744

Subject and Timeout	10
Template	alertDriver

```
{
    "message": "..
}
```

Attributes 3

license_plate	1234 AAA
location	39.4702393,-0.3768049
speed	54km/h

Attribute key: license_plate

```
"1234 AAA"
```

Fig. 4. DT Architecture (bottom) and Ditto UI (Top).

and also allows to trigger actions such as "alertDriver" on the physical device (see numbered area 2).

Once our DT has been instantiated in Eclipse Ditto, we can create and access the different entities that exists in our real scenario via that visual UI. In addition, it also offers us a RESTful interface that allows us to access and update the entities and attributes inherent to our DT. This not only helps us to instance our DT in a real digital environment but also gives us all the tools needed to build a productive application, allowing us to better manage our fleet of vehicles.

The architecture of the DT solution using Eclipse Ditto (or any runtime platform compatible with WoT) is shown in Fig. 4. As we can see, Ditto acts as a base infrastructure platform between our physical devices and the access points of our users to our DT. Ditto RESTful interfaces can also be used to build custom applications that allow a better and more suited control of our use case, leveraging on the platform to connect and update our virtual devices and act as a communication channel to provide feedback to our physical devices.

This ease of integration achieved thanks to the definition of a DT at a conceptual level and its transformation to specific technologies makes the development of DTs and their application a more structured task, reducing the complexity that the fragmentation of technologies in the field can cause.

References

1. Acceleo, `https://eclipse.dev/acceleo/`
2. Digital twins definition language (dtdl), `https://azure.github.io/opendigitaltwins-dtdl/DTDL/v3/DTDL.v3.html`
3. Eclipse ditto - open source framework for digital twins in the iot, `https://eclipse.dev/ditto/`
4. Eclipse modeling project: The eclipse foundation, `https://eclipse.dev/modeling/emf/`
5. Eclipse sirius, `https://eclipse.dev/sirius/`
6. Editdor, `https://eclipse.github.io/editdor/`
7. Mof model to text transformation language, `https://www.omg.org/spec/MOFM2T/1.0/About-MOFM2T`
8. A new era for the internet of things, `https://www.siemens.com/global/en/company/stories/research-technologies/open-innovation/new-era-internet-of-things.html`
9. Obeo designer, `https://www.obeodesigner.com/en/`
10. Onesite platform, `https://www.onesait.com/technology/platform`
11. Siemens and microsoft to converge digital twin definition language ..., `https://press.siemens.com/global/en/pressrelease/siemens-and-microsoft-converge-digital-twin-definition-language-w3c-thing-description`
12. Web-based digital twins for smart cities interest group charter, `https://www.w3.org/2024/06/smart-cities/`
13. Web of Things (WoT) — w3.org. `https://www.w3.org/WoT/`
14. Web of things (wot): Use cases and requirements (Mar 2022), `https://www.w3.org/TR/wot-usecases/`
15. Blank, M., Lahbaiel, H., Kaebisch, S., Kosch, H.: Role models and lifecycles in iot and their impact on the w3c wot thing description. In: Proceedings of the 8th International Conference on the Internet of Things. p. 1–4. IOT '18, Association for Computing Machinery, New York, NY, USA (Oct 2018). `https://doi.org/10.1145/3277593.3277908`, `https://doi.org/10.1145/3277593.3277908`
16. Carrión, E., Pastor, , Valderas, P.: Conceptual modelling method for digital twins. In: Maass, W., Han, H., Yasar, H., Multari, N. (eds.) Conceptual Modeling. p. 417–435. Springer Nature Switzerland, Cham (2025). `https://doi.org/10.1007/978-3-031-75872-0_22`
17. Chen, P.P.S.: The entity-relationship model—toward a unified view of data. ACM Transactions on Database Systems **1**(1), 9–36 (Mar 1976). `https://doi.org/10.1145/320434.320440`
18. EmilioCarrion: Github - emiliocarrion/erdttowot, `https://github.com/EmilioCarrion/ERDTtoWoT`
19. Guinard, D., Trifa, V.: (2008), `https://webofthings.org/2017/04/08/what-is-the-web-of-things/`
20. Korkan, E., Kaebisch, S., Kovatsch, M., Steinhorst, S.: Safe Interoperability for Web of Things Devices and Systems, p. 47–69. Springer International Publishing, Cham (2020). `https://doi.org/10.1007/978-3-030-31585-6_3`, `https://doi.org/10.1007/978-3-030-31585-6_3`
21. Lopez-Arevalo, I., Gonzalez-Compean, J.L., Hinojosa-Tijerina, M., Martinez-Rendon, C., Montella, R., Martinez-Rodriguez, J.L.: A wot-based method for creating digital sentinel twins of iot devices. Sensors **21**(1616), 5531 (Jan 2021). `https://doi.org/10.3390/s21165531`

22. Mens, T., Van Gorp, P.: A taxonomy of model transformation. Electronic Notes in Theoretical Computer Science **152**, 125–142 (Mar 2006). https://doi.org/10.1016/j.entcs.2005.10.021
23. Pastor, O., Molina, J.C.: Model-Driven Architecture in Practice. Springer, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71868-0
24. Schmidt, D.C.: Model-driven engineering. IEEE Computer **39**(2) (2006)
25. Sciullo, L., Trotta, A., Montori, F., Bononi, L., Di Felice, M.: Wotwins: Automatic digital twin generator for the web of things. In: 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM). p. 607–612 (Jun 2022). https://doi.org/10.1109/WoWMoM54355.2022.00095, https://ieeexplore.ieee.org/abstract/document/9842775