JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

Modeling Capabilities of Digital Twin Platforms - Old Wine in New Bottles?

Jérôme Pfeiffer[*], **Daniel Lehner**[†], **Andreas Wortmann**[*], **and Manuel Wimmer**[†]
[*]Institute for Control Engineering of Machine Tools and Manufacturing Units, University of Stuttgart, Germany
[†]CDL-MINT, Institute for Business Informatics - Software Engineering, Johannes Kepler University Linz, Austria

ABSTRACT Digital twins are seen as core technologies to tackle the growing complexity of cyber-physical systems to better understand, monitor, and optimize their behavior. Digital twin platforms aim to facilitate the systematic engineering of digital twins by providing dedicated languages and corresponding tools to describe their abilities. However, with the emergence of these languages for digital twins, the question arises what the nature of these languages is and how they differentiate from existing modeling languages already used in the area of cyber-physical systems. To shed more light on this new modeling area, we study in this paper the modeling capabilities of three industrial digital twin platforms and frame them in existing and well-known modeling concepts provided by UML. In particular, we (i) extract the conceptual metamodels of three industrial digital twin platforms, (ii) compare them with common object-oriented modeling concepts of UML, (iii) and provide first insight about the portability of models between the platforms by performing an experiment. In particular, we use UML class diagrams as an anchor for relating the modeling concepts of digital twin platforms and as pivot for digital twin platform portability. Our investigation summarizes current modeling capabilities of digital twin platforms to provide a better understanding of their shared concepts to developers using such platforms. It also shows that these modeling capabilities often rely on well-known modeling concepts, but also add some new aspects. The performed experiment additionally gives first insights into the portability of different digital twin platform metamodels. To sum up, this work can be see as a starting point for uncovering the nature of digital twin modeling and providing a digital twin language family enabling developers to select appropriate modeling features for describing different aspects of digital twins without having to reinventing the wheel.

KEYWORDS Digital Twins, Language Comparison, Structural Modeling

1. Introduction

Digital twins (DTs) as a digital representation of cyber-physical systems (CPSs) are a core technology of Industry 4.0/5.0 and increasingly deployed in a multitude of domains, e.g., manufacturing (Tao, Cheng, et al. 2018; Tao et al. 2019), avionics (Kraft 2016; Mandolla et al. 2019), building information and energy management (Lu et al. 2019; Jain et al. 2019; Francisco et al. 2020), automotive (Biesinger & Weyrich 2019) or health care (Bruynseels et al. 2018; Laaki et al. 2019; Jimenez et al. 2020). This increasing demand for DTs calls for more efficient development methods (Tao, Zhang, et al. 2018; Rasheed et al. 2020). DTs comprise models, traces, (aggregated) data, and services to represent, monitor, control, or optimize the observed CPS (Kirchhof et al. 2020; Bibow et al. 2020).

DT platforms aim to facilitate the development of DTs by providing tooling for their definition and operations with strong integration into the services of the platform provider. As developing DTs requires close collaboration between software experts and CPS experts, model-driven software development is often employed to overcome the gap between the problem (or business) domain and the software solution (France & Rumpe 2007; Brambilla et al. 2017). Thus, DT platforms increasingly

JOT reference format:
Jérôme Pfeiffer, Daniel Lehner, Andreas Wortmann, and Manuel Wimmer. *Modeling Capabilities of Digital Twin Platforms - : Old Wine in New Bottles?*. Journal of Object Technology. Vol. 21, No. 3, 2022. http://dx.doi.org/10.5381/jot.2022.21.3.a10

provide modeling capabilities for their development.

However, there is currently no common understanding of which modeling techniques the different DT platforms are capable of. As a consequence, it is unclear whether they provide new modeling concepts for DTs, or provide existing concepts (i.e., old wine) wrapped in new textual syntaxes (i.e., in new bottles). Hence, in this paper, we first investigate the metamodels used by platforms and afterwards compare them with existing modeling techniques to investigate to which extent these metamodels offer new concepts. More specifically, we investigate the following research questions:

RQ1 What are the conceptual metamodels used by existing DT platforms?

RQ2 How do they relate to existing modeling concepts and standards?

RQ3 How can existing modeling concepts be leveraged to enable portability of models between these platforms?

To answer these research questions, we select representative DT platforms to investigate. DT platforms originate often from former IoT platforms. Microsoft Azure was a pioneer in this area with the IoT Hub[1] and provides a DT adaption of it with Azure DTs[2]. Besides Microsoft Azure, Amazon Web Service features a DT service with IoT TwinMaker[3]. In contrast to the Azure service with its DT Definition Language (DTDL)[4], they do not provide a metamodel of the language. As a third platform, we consider an open-source initiative by the Eclipse Foundation[5,6,7]. With their DT suite consisting of Hono, Vorto, and Ditto they offer dedicated modeling languages for defining models of DTs. While there exist other DT platforms from other vendors, e.g., Oracle[8], Siemens[9], ScaleOut[10], or IBM[11], we limit the study of DT platforms for this paper to three platforms covering different approaches for their language definition and leave the investigation of additional platforms as future work as our aim is not completeness, but shedding some light in this new modeling area.

To this end, we extract the conceptual metamodel for each of the considered platforms from existing documentation and available artifacts. The correctness of the extraction result is validated by (i) instantiating each metamodel using a common case, and (ii) peer-reviewing the metamodels and the case implementation among each author of this paper.

[1] https://azure.microsoft.com/en-us/services/iot-hub/
[2] https://docs.microsoft.com/en-us/azure/digital-twins/overview
[3] https://docs.aws.amazon.com/iot-twinmaker/latest/guide/what-is-twinmaker.html
[4] https://github.com/Azure/opendigitaltwins-dtdl/blob/master/DTDL/v2/dtdlv2.md
[5] https://projects.eclipse.org/projects/iot.ditto
[6] https://www.eclipse.org/hono/
[7] https://www.eclipse.org/vorto/
[8] https://docs.oracle.com/en/cloud/paas/iot-cloud/iotgs/oracle-iot-digital-twin-implementation.html
[9] https://siemens.mindsphere.io/content/dam/cloudcraze-mindsphere-assets/03-catalog-section/05-solution-packages/solution-packages/digitalize-and-transform/Siemens-MindSphere-Digitalize-and-Transform-sb-72224-A8.pdf
[10] https://www.scaleoutsoftware.com/products/digital-twin-builder/
[11] https://digitaltwinexchange.ibm.com/

Since domain-specific languages, such as the considered DT platform languages, often derive concepts from general-purpose modeling languages, we align the platforms' metamodels with common well-known object-oriented modeling concepts and compare them in detail with the Unified Modeling Language (UML) standard, and in particular, with UML class diagrams (CD).

By bridging the identified gaps between UML and the metamodels offered by DT platforms, we develop a UML CD extension which can act as pivot for transforming DT models from one platform-specific metamodel to another. We demonstrate the feasibility of this idea by an experiment considering the transformation from the metamodel offered by the Azure platform to the metamodel of the AWS platform.

Summarized, to answer the stated research questions we present the following three contributions:

1. A structured discussion and comparison of the underlying modeling concepts of three selected DT platforms. To achieve this, we extract conceptual metamodels from the modeling capabilities offered by the platforms.

2. An estimation of the degree to which the metamodels provided by the respective platforms present novel concepts as well as a characterization of these languages based on their modeling concerns. For this, we align the extracted metamodels with existing, standardized modeling concepts.

3. An assessment of the compatibility of the identified concepts between different metamodels, using an UML extension as pivot for bridging the gap between standardized modeling concepts of UML and the metamodels provided by DT platforms.

In the following, Sec. 2 introduces preliminaries and discusses related work. Then Sec. 3 presents the conceptual metamodels of the languages provided by the considered DT platforms. Sec. 4 compares the metamodels with existing modeling concepts. Sec. 5 demonstrates the use of our findings to extend UML with DT modeling capabilities to act as a pivot for exchanging DT models. Finally, Sec. 6 concludes this paper with an outlook on future perspectives.

2. Background

This section introduces the term digital twin, describes each of the digital twin platforms we consider, and debates related work that is relevant to the topics investigated by this paper.

2.1. Digital Twins

Digital twins (DTs) are software systems comprising data, models, and services to interact with a cyber-physical system for a specific purpose (Kirchhof et al. 2020; Bibow et al. 2020; Kritzinger et al. 2018). As such, they serve to better understand, monitor and optimize the behavior of their counterpart. They are "twins" in the sense that changes to the counterpart are automatically reflected in the DT and changes in the DT are automatically reflected in the counterpart. To this end, DTs need

to yield a representation of their counterpart, as well as a connection to it that supports communicating changes between both systems (Eramo et al. 2022). Through this, DTs enable a variety of value-adding services, such as detailed design-space exploration, real-time monitoring, predictive maintenance, process optimization, and more (Tao et al. 2019; Xu et al. 2021).

Digital shadows (Brauner et al. 2022; Bibow et al. 2020; Kritzinger et al. 2018) are tailored data structures that represent a specific system for a specific purpose. They are shadows in following the changes of the represented system under a specific illumination. Hence, they neither need to be complete nor can they change the represented system directly.

2.2. Digital Twin Platforms

To support developers in creating DTs and shadows, big vendors already provide tooling support (Qi et al. 2021), for different aspects of a DT. A combination of several tools from one supplier is often referred to as a DT platform (Lehner et al. 2022). In the following, the DT platforms examined in this paper are described in more detail.

Microsoft Azure supports modeling in form of JSON files corresponding to the Digital Twin Definition Language (DTDL). DTDL models are created using the Azure Digital Twins (ADT) service, which offers a graphical model editor, and a webservice for reating, reading, adapting, and deleting DTs of a certain system. Based on JSON DT definitions, devices can be connected to the Azure platform using the Azure IoTHub and several tools provided by the platform (e.g., a time series insights service to store temporal data and the Azure machine learning service supports training and applying machine learning models on observed data) can make use of this interaction with the devices.

Eclipse offers Vorto as a dedicated textual modeling tool that also serves as a repository and marketplace for created models in the Vortolang modeling language. Based on these models, a RESTful webservice can be set up using Eclipse Ditto, and connect various devices using Eclipse Hono or the Bosch IoTHub. All mentioned tools are provided as open-source software on GitHub, together with extensive documentation and docker-compose files for running these tools on a self-hosted server. Additionally, the Ecore source code of the Vortolang metamodel is provided on GitHub.

Amazon Web Service (AWS) provides an IoT service, called IoT TwinMaker dedicated to building DTs. It enables users to model their devices together with the data they exchange with the DT. For this, IoT TwinMaker offers a RESTful webservice and an API reference[12] to send appropriate models to. Additionally, the IoT TwinMaker has a service scene composer that can be employed for creating 3D scenes of the physical space and overlay data received from devices in the scene. Furthermore, the IoT TwinMaker provides plug-ins for building dashboards for end-users. For connecting devices through the edge, AWS provides the Greengrass[13] service, which provides connectors and connectivity services for edge devices.

[12] https://docs.aws.amazon.com/iot-twinmaker/latest/apireference/Welcome.html

[13] https://aws.amazon.com/de/greengrass/

2.3. Related Work

In the context of this paper, there is already literature that summarizes relevant modeling concepts for domains that DTs usually build on, such as the Internet of Things (Wortmann et al. 2020; Petrasch & Hentschke 2016), cyber-physical systems (Derler et al. 2012; Broman et al. 2012), or specific domains such as building automation (Tang et al. 2019). For the area of DTs, there is already existing work that aims to conceptualize DTs (Yue et al. 2021; Becker et al. 2021), or reuse existing modeling languages such as UML (Munoz et al. 2021; Azangoo et al. 2020) or AutomationML (Lehner et al. 2021; Schroeder et al. 2016) to implement these concepts. However, to implement DTs, different tools are required. Existing tools and platforms have already been examined in the literature (Qi et al. 2021; Lehner et al. 2021). However, there is still little knowledge about the modeling capabilities of these tools and platforms, as well as their alignment with theoretical concepts often discussed in the context of DTs or related domains. Also, the use of existing modeling concepts as pivot language for these DT platform languages is not yet investigated.

In this work, we aim to bridge this gap between the implementation of DTs and their model-based representation, by investigating the modeling capabilities of existing DT platforms, especially in the context of existing modeling standards, and to which extent these modeling capabilities overlap, e.g., to provide solutions which enable portability between different DT platforms and other future developments.

3. Uncovering DT Platform Metamodels

To enable a structured comparison of the DT languages provided by the selected platforms, we derive the underlying conceptual metamodels consisting of classes, attributes, and relationships. The concrete syntaxes of the considered DT languages are realized and described in different ways. DTDL provides a JSON-based modeling syntax, whereas Vortolang offers a metamodel based on the Eclipse Modeling Framework (EMF) for defining its language, and the AWS IoT TwinMaker is API-based and requires REST calls to this API in an appropriate format. To overcome these heterogeneous language realizations, we extracted their abstract syntaxes to conceptual metamodels. Therefore, we (i) focus on the description of the type level of the DT rather than its instance level, and (ii) simplify namings and modeling concepts to enable a common conceptual view across the chosen platforms. We employed the descriptions and examples given by the platform vendors to extract essential concepts and used peer-reviewing among the authors of this paper to check our results. We implemented the derived metamodels using EMF, i.e., Ecore as metamodeling language. The results together with examples for models of these metamodels are published on GitHub[14].

3.1. Microsoft Azure Digital Twin Definition Language

The Digital Twin Definition Language (DTDL) is developed by Microsoft Azure and used by its DT platform (i.e., the Azure Digital Twins Service, which is connected to physical assets via

[14] https://github.com/derlehner/dt_language_comparison

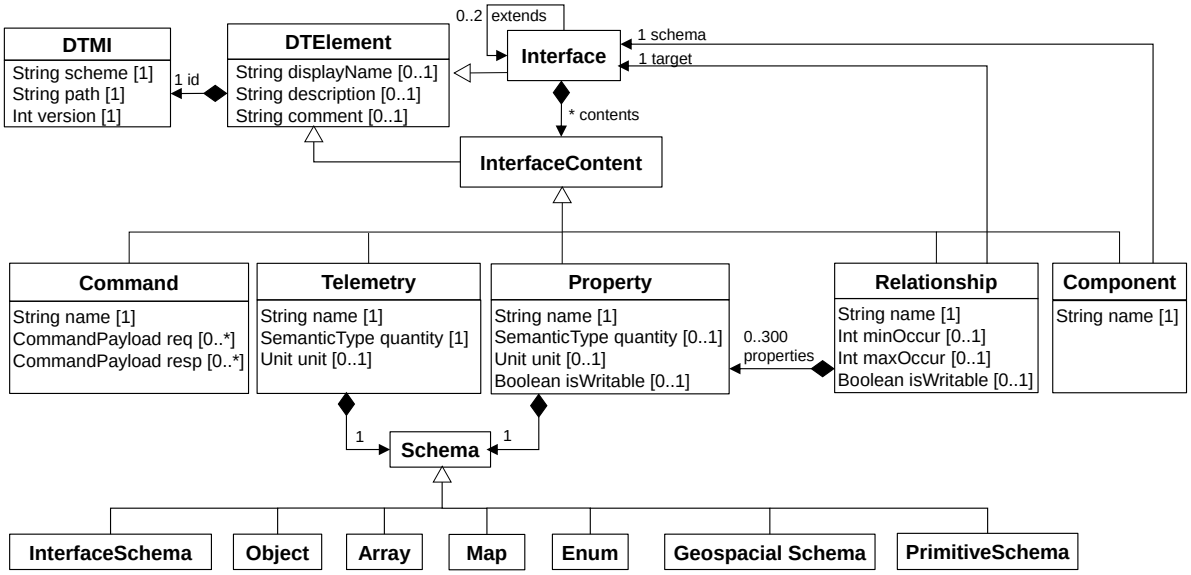

Figure 1 Conceptual metamodel of DTDL in UML CD notation.

the Azure IoT-Hub). We derived the DTDL metamodel (DTDL-MM) shown in Figure 1 from the documentation of DTDL v2, provided on GitHub[15].

In this metamodel, the main element is called `Interface`, representing the type of a physical asset. This interface has a `name`, and a list of `contents`. It can also extend up to two other interfaces, up to an inheritance depth of 10 (the documentation however does not explain these limitations). The contents of the interface is a list of `Properties`, `Telemetries`, `Relationships`, `Components`, and `Commands`.

A relationship specifies a connection that an interface can have with another interface (i.e., the target). Each relationship specifies the min and max multiplicity (i.e., the number of devices conforming to the target interface that a device from the given interface can be connected to for the given relationship). Relationships can also be extended with further properties, although the maximum number of properties contained by a relationship is 300. The writable property of a relationship specifies whether it can be changed during runtime.

Besides relationships, there are also components to indicate connections between different interfaces. In contrast to relationships, these components define that a certain interface is a fixed part of another interface, meaning that this information cannot be changed during runtime, and a device conforming to the component cannot exist without a device conforming to the outer interface. Besides the target interface (i.e., schema) of this connection, there is no further information specified. However, one limitation of components is that they comprise only Commands, Properties, and Telemetries. Thus, a Component cannot have outgoing component or relationship references. Also, there is no option to specify the multiplicity of a Component, meaning that a Component represents exactly one connection between the source and target Interface (based on the implementation of the component by Azure).

Telemetries represent continuous data streams emitted by devices, whereas properties define variables whose values do not change frequently. Both telemetries and properties contain a schema, unit, and quantity. The quantity references a semantic type, which is a predefined enumeration offered by DTDL, listing several quantities (e.g., temperature, humidity, angle, acceleration). Based on the chosen type, a unit can be selected. Therefore, DTDL offers a list of units, such as degree Celsius or degree Fahrenheit for temperature, and meter per second squared or centimeter per second squared for acceleration. The chosen type restricts the units that can be selected for certain properties of telemetry. The schema of a property or telemetry defines the structure of the data represented by property or telemetry. This can be a primitive schema (e.g., boolean or double), an object (i.e., a list of coupled properties), a list (i.e., a collection of multiple values that all conform to the same type), map (i.e., a key-value combination, where each key and each value must conform to the same type, respectively), enum (i.e., a list of customizable literals, whereas each literal contains a name and an optional description), interface schema (i.e., a reference to a schema described by the encompassing Interface), or a geospatial schema (a set of predefined property structures (i.e., objects) offered by DTDL for describing positions of geometric shapes, e.g., point, line string, or polygon). To improve understandability, the implementation details of schema subclasses are omitted from Figure 1. In addition, properties also define whether they are writable (changeable during system runtime).

Commands allow interactions with the device during runtime. Therefore, each command can specify several `CommandPayloads` that are either sent as a request to the device or emitted by the device as a response to an operation call.

Interfaces, properties, relationships, commands, and telemetries all contain an optional displayName (textual identifier of the element used for display in a user interface), comment, and description, and a mandatory id referencing a `DTMI` object. A

[15] https://github.com/Azure/opendigitaltwins-dtdl/blob/master/DTDL/v2/dtdlv2.md

DTMI object consists of a scheme that is implemented by the element (i.e., the value "dtdl" defines that the element conforms to the DTDL language), a path providing a unique identifier for this specific element, and a version indicating the version number of the current element. All elements except interfaces also allow the specification of a name, which is used to refer to this element within software code.

3.2. Eclipse Vortolang

Vortolang is the language used by Eclipse Vorto to manage and share so-called DT representations of physical assets within the platform provided by Eclipse (comprising Eclipse Vorto, Ditto, and Hono). In Figure 2, we provide a conceptual Vortolang metamodel (from now on referred to as VL-MM) based on the Ecore files published by Eclipse on Github (Eclipse 2020), the respective Xtext grammar[16], and the documentation of this Xtext grammar[17].

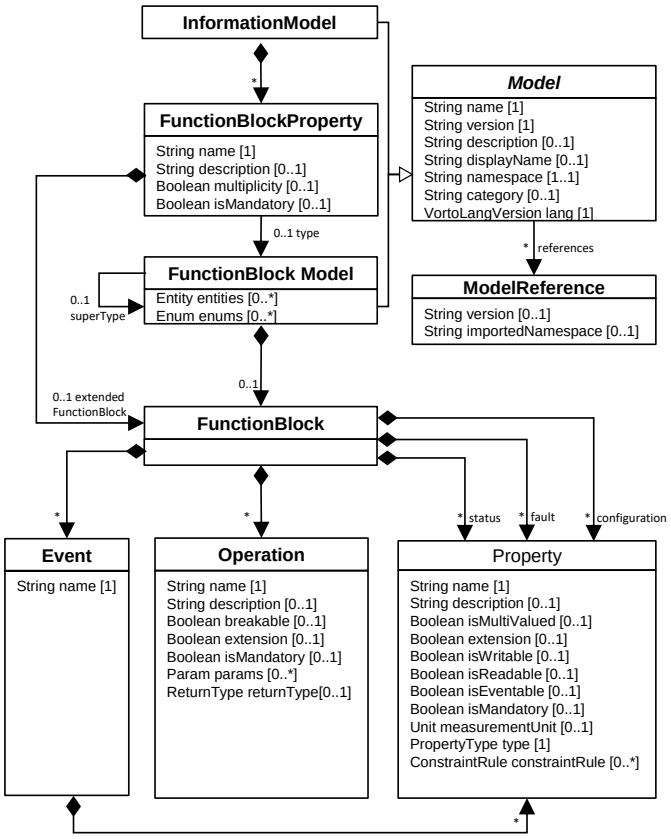

Figure 2 Conceptual metamodel of Vortolang in UML CD notation, based on (Eclipse 2020).

`InformationModels` are the main element. They allow to aggregate several `FunctionBlocks` into bigger components, annotating them with a name, description, and the information whether its presence is mandatory, and whether there can be multiple instances of this FunctionBlock referenced. These FunctionBlocks are either contained by the `FunctionBlockProperties` of the information model, or imported (via the `extendsFunctionBlock` attribute) and extended with additional properties or events.

FunctionBlocks are reusable implementation parts, that summarize `Operation`, `Event`, `configuration`, `fault`, and `status` information required for a certain functionality offered by an InformationModel. Faults, configurations, and status are a list of `Properties`. One difference between the status and configuration properties of a FunctionBlock is that by default, status properties are read-only and configuration properties are read- and writable.

Each property has a type, that can be chosen from a predefined enum of primitive property types (e.g., bool or int), a dictionary, a so-called list, or an entity created as part of the encompassing FunctionBlockModel. This type can be further specified by adding a measurement unit (it can reference a literal of a custom-defined enum) to the property. The constraintRule of a property defines restrictions on the allowed values of that property. Additionally, properties can have a name, description, and can be defined as mandatory, multiple (several instantiations are possible), writable (changeable during runtime), readable (accessible during runtime), and eventable (mapping to an event with the same name).

An event describes data that is omitted by the physical asset described by the FunctionBlock, and contains a name as well as a list of properties. One of these properties must be a timestamp with the predefined primitive property type dateTime as type (this restriction is not modeled in the metamodel provided by Eclipse, but implemented as an informal rule in the Vorto editor, and described in the GitHub documentation).

An operation describes interaction possibilities offered by the described asset. Each operation has a `name`, and a `description`, and defines whether it is `breakable`, an `extension`, or `mandatory`. Additionally, an operation can have a `returnType` and a list of `params`.

A `FunctionBlockModel` adds metadata to a FunctionBlock, and also specifies a list of entities and enums that can be used by the referenced FunctionBlock. Additionally, a FunctionBlockModel can have a `superType` FunctionBlockModel.

Common attributes of the FunctionBlockModels and InformationModels described above are a reference to the `vortolangVersion` that they were created with, a textual `description` of the modeled element, a `displayName` to be used as a representation in user interfaces, and a `category` that groups several FunctionBlockModels and InformationModels. Additionally, they have a `name` and `namespace` that together uniquely identifies them. Based on this unique identifier, different `versionNumbers` of an element can be specified, and finally, an InformationModel can import a FunctionBlockModel based on its namespace, name, and version.

3.3. AWS IoT TwinMaker

The IoT TwinMaker is provided by Amazon's cloud platform Amazon Web Services. It enables users to model devices, equipment, and data sent from the physical to the digital space. We extracted the metamodel from the official user guide provided by

[16] https://github.com/eclipse/vorto/tree/development/core-bundles/language
[17] https://github.com/eclipse/vorto/blob/development/docs/vortolang-1.0.md

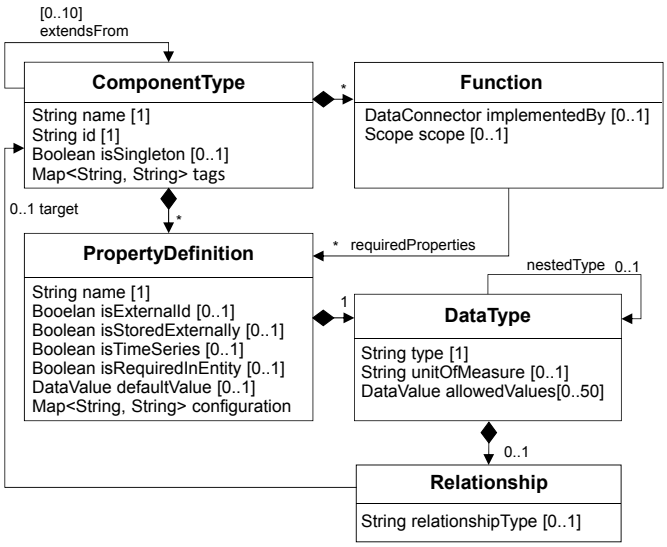

Figure 3 Conceptual metamodel of the AWS IoT TwinMaker in UML CD notation.

Amazon[18]. The user guide does not give an explicit metamodel but rather an API description for REST requests in the JSON format. Thus, the metamodel that we present in the following is derived from and represents this API description of the AWS IoT TwinMaker. Figure 3 shows the extracted metamodel for AWS IoT TwinMaker.

The top most concept of the metamodel of IoT TwinMaker (referred to as TM-MM in this paper) is `ComponentType`. `ComponentTypes` define the context and structure of data. They have a `name` and a `boolean` flag to indicate whether they are singletons. Furthermore, they can be enriched with tags. Tags add meta-information, e.g., a version number, to ComponentTypes.

Furthermore, ComponentTypes can reference many `PropertyDefinitions` that have a name, a boolean field that indicates whether the property's id originates from an external data store, a boolean field `isStoredExternally` that indicates whether the property is stored externally, a boolean field `isTimeSeries` that specifies whether the property contains time-series data, and a boolean field `isRequiredInEntity` that specifies whether the property is required when the corresponding ComponentType is instantiated. Moreover, a `PropertyDefinition` can define a `defaultValue`. The class `DataValue` has one attribute for each boolean, double, expression, integer, list, long, maps, and string values. Other types of values are not supported. A `configuration` is a mapping that specifies configuration information in a map from `String` to `String`. This is used to specify the information that one read from and write to an external source.

A `PropertyDefinition` contains a `DataType` that specifies the data type of the property. It comprises an array of `allowedValues`, the name of the type, nested types, e.g., to specify a list of double values as a type list and nested type double, a `unitOfMeasure`, and a optionally a relationship, that can define relations to other `ComponentTypes` via their id.

Besides, a `ComponentType` can reference multiple functions. They specify a `DataConnector` via an attribute `implementedBy` that provides a reference to the implementation, i.e., the id of a lambda function in AWS, that realizes the function. `RequiredProperties` defines a list of `PropertyDefinitions` that describe required properties for the function to work. To reuse existing definitions, a `ComponentType` can extend up to 10 other `ComponentTypes` to reuse their `PropertyDefinitions` and `Functions`.

3.4. Answering RQ1: What are the conceptual metamodels used by existing DT platforms?

Extracting the metamodels offered by the chosen DT platforms from documentation shows that often, similar modeling concepts are used, although the implementation details might differ, and some concepts may not be available in all metamodels (e.g., TM-MM does not allow the composition of elements). The extraction process was also complicated by the missing description of some language properties (e.g., faults or mandatory operations in VL-MM). Additionally, some interesting modeling restrictions are found (e.g., different upper bounds for maximum number of inherited super-classes, inheritance depth, or composition depth), which are unfortunately not further discussed in the documentation of the individual platforms.

Still, the extracted metamodels show some interesting aspects. All metamodels provide some elements to represent a class of physical assets (i.e., Interface in DTDL-MM, Function-block in VL-MM, and ComponentType in TM-MM).

DTDL-MM and VL-MM even allow the versioning of individual model elements. By this, different model elements with different versions can be combined into one single model. Each model element must therefore be referenced via a unique name and its version number.

Besides such modeling elements, DTDL also offers predefined enumerations for specifying physical quantities such as Temperature or Acceleration, and units for these quantities, such as degrees Celsius or kilometers per hour.

4. Analysing DT Platform Metamodels

After individually presenting the conceptual metamodels of three representative DT platforms, we now aim to analyze and compare them to existing standardized modeling techniques. In a nutshell, all of the presented metamodels enable the design of structural aspects of DTs and their data. To classify the concepts of the metamodels, the following first aligns the platform metamodels' concepts with well-known concepts of object-oriented modeling on a high level of abstraction. Subsequently, we compare them in more detail with the UML CD language which is a standardized structural modeling language.

4.1. Alignment of Platform Metamodels with Concepts of Object-Orientated Modeling

For orientation purposes and a more detailed comparison in the next subsection, we give a course-grained alignment for the concepts found in the metamodels provided by the DT platforms

[18] https://docs.aws.amazon.com/iot-twinmaker/latest/guide/what-is-twinmaker.html

OOM Concepts	DTDL Metamodel	Vortolang Metamodel	TwinMaker Metamodel
Class	Interface	FunctionBlock	Component Type
Property	Property	Property	Property Definition *isTime-Series=false*
Temporal Property	Telemetry	Event	Property Definition *isTime-Series=true*
Association	Relationship	-	Property Definition. DataType. relationship
Method	Command	Operation	Function
Composition	Component	FunctionBlock Property	-

Table 1 Comparison of concepts of object-oriented modelling with metamodels provided by DT platform.

Shared Concepts			
DTDL	**Vortolang**	**TwinMaker**	**UML CD**
name	name	name	name
description	description	description	description
comment	-	-	comment
DTMI.content	vortolang	-	container. umlVersion
DTMI. namespace	namespace	-	namespace
displayName	displayName	-	-
DTMI.version	version	-	-
DTMI. identifier	-	-	-

Table 2 Comparison of shared concepts provided by DT platforms with the metamodel of UML CD.

with concepts of Object-Oriented Modeling (OOM) (Booch et al. 2005)–see Table 1 for an overview. Most of the concepts found in the DT platform metamodels can be related to common concepts of OOM. The basic concepts of OOM are classes and objects, i.e., instances of these classes. A comparable concept to class is available in all of the three platform metamodels. In the DTDL-MM, it is called interface, in the VL-MM it is called FunctionBlock and in the derived TM-MM it is called ComponentType.

Classes can have properties to represent the underlying object structures. This is also true for the metamodels of the DT platforms. The DT platform metamodels differentiate between time-series properties and non-time-series properties. With TemporalProperty, such a concept is also available in extended versions of the OOM concepts (Gómez et al. 2018). Although the term event used by Vortolang can be misleading here, it means the same as the concept in the other platforms describing data emitted by a device over time.

Associations enable connections between different classes in OOM. This concept is also supported by the metamodels of DTDL and Twinmaker, but not by VL-MM. If the target of an association is contained by its source, a Composition is used instead of an association in OOM. This composition is not yet supported by the IoT TwinMaker metamodel. In DTDL-MM, it is supported if the target class does not have any outgoing associations or compositions. In VL-MM, it is supported by referencing a FunctionBlock (Class in UML CD) from an InformationModel using a FunctionBlockProperty. However, this is also only possible on one level, as the target FunctionBlock cannot contain any further FunctionBlocks.

Methods, which provide an outside interface to the functionality of a class in OOM, are supported by all metamodels provided by the examined DT platforms.

4.2. Comparison of DT Platform Metamodels with UML Class Diagrams

In the following, we investigate in detail how the concepts comprised in the platform metamodels compare to a standardized modeling language for OOM concepts. More precisely, we use UML CD (Cook et al. 2017) as representative for standardized modeling languages for describing structural aspects of software systems. We chose UML CD as it is a unification of OOM languages that are widely used in industry and allows for extensions (Hutchinson et al. 2011). The following comparison is structured using the concepts of OOM (cf. Table 1).

4.2.1. Shared Concepts Some modeling concepts are used by most of the individual modeling elements, which are collected in Table 2. First, these elements have a name and a description in all three examined metamodels as well as in UML CD. In DTDL-MM and VL-MM, elements also reference the used language version, which can be mapped to the umlVersion attribute of the model tag containing elements in UML, a namespace, which is mapped to the UML CD namespace of an element, and a version number of the element, which cannot be mapped to UML CD. One limitation here is that the version number is limited to classes in VL-MM. One attribute that is supported by DTDL-MM and UML CD is the comment. One attribute supported by both DTDL-MM and VL-MM, but not by UML CD is the displayName, which offers an additional, human-readable representation of the name, e.g., for presentation in a dashboard.

4.2.2. Class As we already recognized, the concept of class is available in all of the platforms' metamodels. Table 3 shows a

Class			
DTDL	**Vortolang**	**TwinMaker**	**UML CD**
extends	superType	extendsFrom	Generalization
-	enum	-	enum
-	category	tag	tagged value
-	-	isSingleton	-
-	entity	-	nestedClassifier

Table 3 Comparison of class concepts provided by DT platforms with the metamodel of UML CD.

detailed comparison of how the concept of classes is realized in the metamodels provided by the different DT platforms and how these concepts align with the ones provided by the metamodel of UML CD. Because class diagrams realize object-oriented concepts, classes are represented there, too. Also representable in class diagrams and provided by all considered metamodels of DT platforms are extensions from other existing classes. However, this extension is restricted in different ways across the metamodels, e.g., TM-MM allows extending from up to ten other ComponentTypes, whereas VL-MM is limited to single inheritance. Representing enums and entities as part of classes is supported by VL-MM and also by UML CD. Supported by metamodels of DT platforms and provided by UML CD, too, are concepts to categorize models via tags, for instance, sensors, actuators, etc.. However, there is a difference between its realization in Vortolang and TwinMaker. In Vortolang the category is one String describing the category, whereas in the metamodel provided by TwinMaker it is a map, that maps keys of tags to possible values, e.g., a key would be "sensor" and the value would be the type of sensor. In UML there are tagged values to do so. Tagged Values are additional properties (and their values) that can be set for any UML element. One concept not supported by UML CD is marking a class as a singleton, i.e., to be instantiated only once in a system.

4.2.3. Property All metamodels offered by DT platforms provide the concept of properties. Table 4 shows a detailed comparison between the metamodels provided by the DT platforms and by UML CD. Supported by all platform metamodels, and also by UML CD, properties have a type and constrain the allowed values assignable to a property (by explicitly defining a constraintRule in VL-MM or by enumerating allowed values in DTDL-MM, TM-MM, and UML CD metamodel). However, the different platforms provide different capabilities of modeling the type of a property (cf. Sec. 3). Both VL-MM and TM-MM allow enabling and disabling the multiplicity of properties, and VL-MM additionally allows to mark properties as mandatory. Both of these aspects are supported by UML CD, and in addition, it offers a wider range of expressibility by specifying lower and upper bounds for multiplicities. Both VL-MM and DTDL-MM allow marking properties as read-only, which

Property			
DTDL	**Vortolang**	**TwinMaker**	**UML CD**
Schema	type	DataType.type	type
-	multiplicity= true	-	upper>1
-	multiplicity= false	-	upper=1
-	isMandatory	isRequired InEntity	lower > 0
isWritable	isReadable/ isWritable	-	readOnly
-	extension	-	redefined Property
-	-	defaultValue	defaultValue
enum	constraintRule	DataType. allowedValues	enum
unit	measurement Unit	unitOfMeasure	-
-	-	isExternalId	-
-	-	isStored Externally	-
-	-	configuration	taggedValue
quantity	-	-	-

Table 4 Comparison of property concepts provided by DT platforms with the metamodel of UML CD.

is also supported by UML CD. Concepts provided by only one platform and supported by UML CD, are assigning a default value or further configurations to properties (both supported by TM-MM), and the redefinition or extension of properties (supported by VL-MM).

Available in VL-MM and TM-MM, but not supported by UML CD, is the option to define the measurement unit of properties. In addition, TM-MM can mark properties as external or externally stored, which is unsupported by UML CD. Exclusive to DTDL-MM, and not represented in UML CD, are physical quantities that further describe the values of properties.

4.2.4. TemporalProperty Not represented in UML CD, but available in all metamodels provided by the DT platforms are temporal properties that define time-series data omitted by a class. DTDL and Vortolang provide own concepts for this, whereas in TwinMaker it is a flag in a PropertyDefinition. In VL-MM, this concept can reference multiple properties, whereas each specifies further information (cf. Table 5). In DTDL-MM, however, this concept exists separately from properties, and thus, defines units, quantities, and schemas.

TemporalProperty		
DTDL	**Vortolang**	**TwinMaker**
-	properties	-
unit	-	measurementUnit
quantity	-	-
Schema	-	type

Table 5 Comparison of temporal property concepts provided by DT Platforms. Temporal properties are not supported by UML class diagrams.

4.2.5. Method Methods are supported by all platform metamodels and by UML CD. Table 6 shows a detailed comparison.

Similar to Parameters in UML CD, TM-MM offers requiredProperties for a method. In DTDL-MM and TM-MM, this can be further distinguished between input and output parameters. This differentiation is also supported for UML CD Parameters.

The extension of methods is supported by VL-MM and also by UML CD. A platform exclusive feature of IoT TwinMaker is the reference to the implementation of the method by the attribute implementedBy. Exclusive to VL-MM and not supported by UML CD is the definition of mandatory, although it is unclear what this means semantically, and marking methods as breakable (potentially throwing errors).

Method			
DTDL	**Vortolang**	**TwinMaker**	**UML CD**
response	returnType	required properties	Parameter. direction=return
request	params		Parameter .direction=in
-	extension	-	redefined Operation
-	-	implementedBy	method
-	isMandatory	-	-
-	breakable	-	-

Table 6 Comparison of method concepts provided by DT platforms with the metamodel of UML class diagrams.

4.2.6. Association The concept of associations is available in all considered metamodels, except VL-MM. Table 7 shows a detailed comparison of this concept. In all metamodels supporting Associations, they have a target element. The definition of an upper bound of the multiplicity that is supported by DTDL-MM is also available in UML CD. Setting associations as writable are supported by DTDL-MM and also provided by the property on the association end in UML CD. Not supported by UML CD are properties on associations. In TM-MM, a relationship has a relationshipType, which is a String attribute that can be used to classify the relationship. This can be, e.g., used to describe that a certain relationship represents a composition. However, relationshipTypes cannot be directly mapped to compositions in UML CD, as (i) a relationshipType can also have any arbitrary other value, which could not be mapped to compositions, and (ii) a semantical interpretation of such a composition is difficult, as a relationshipType can also have any other value. Thus, additional support has to be developed in the surrounding framework to provide the intended semantics.

Association		
DTDL	**TwinMaker**	**UML CD**
maxMultiplicity	-	memberEnd.upper
minMultiplicity	-	memberEnd.lower
target	target	OwnedEnd
isWritable	-	OwnedEnd. isReadOnly
properties	-	-
-	relationshipType*	-

* for instance, can be used to represent composite associations

Table 7 Comparison of association concepts provided by DT platforms with UML class diagrams. Vortolang is not displayed as it does not support associations.

4.2.7. Composition Composition is supported by DTDL-MM and VL-MM, and also represented in UML CD (cf. Table 8). VL-MM can characterize composed elements as mandatory and define whether there can be multiple instances linked or not. Both are supported by UML CD, too.

Composition		
DTDL	**Vortolang**	**UML CD**
schema	type	OwnedEnd
-	isMandatory	lower>0
-	multiple	upper>1

Table 8 Comparison of composition concepts provided by DT platforms with UML class diagrams. IoT TwinMaker is left out as it does not support composition.

4.3. Answering RQ2: How do metamodels offered by DT platforms relate to existing modeling concepts and standards?

To answer this question, we compared the metamodels provided by three DT platforms with existing modeling concepts. A high-level comparison of modeling elements shows that most modeling elements found in the metamodels provided by DT platforms can be related to existing OOM capabilities. The only addition is the modeling of time-series properties, which in OOM is only available via extensions. A detailed comparison of DT platform metamodels with the related concepts of the UML CD shows that many of the platforms' concepts are covered by UML CD. However, some of the concepts also available in UML CD underlie restrictions in the metamodels provided by the platforms. For instance, the number of how many classes can be extended is different in each platform. Furthermore, types of properties are limited to common types found in programming, i.e., String, Integer, List, Map, etc.. A feature that can be explained with the IoT background of the platforms is the capability of the definition of measurement units for properties. A platform-exclusive concept to DTDL is the type definition that can define a semantic type. With this, quantities (Burgueño et al. 2019), i.e., observable and measureable properties, e.g., distances, and dimensions, e.g., length, and units, e.g., metre, mile, etc., can be formulated and applied to properties. The fact that two of the platforms are being integrated into a cloud environment explains that they provide concepts for cross-referencing artifacts available in other services of the platform. There are also similar features in terms of their semantics in the platforms but realized differently in the respective metamodels. For instance, constraining the values assignable to properties, is expressable via an expression over the property value, whereas in IoT TwinMaker it is an array of allowed values, that can contain up to 50 values. In UML CD and DTDL, this is supported by defining allowed values as an enum that is used as property type, or by using the Object Constraint Language (OCL) in UML CD.

5. Towards Using UML CD as Pivot Model for Ensuring Portability

Based on the analysis of the DT platform metamodels, we propose an extension to the UML CD, named DTUML, that enables us to use UML CD as the pivot language for platform portability. The intention of this pivot language is to inject and extract proprietary models used by DT platforms to reduce the transformation effort for transforming between the individual platforms as it was done for other domains (e.g., see (Kappel et al. 2006)) to reduce the required transformations to $2 \cdot n$, where n is the number of proprietary modeling languages. We demonstrate this platform portability based on the DTUML pivot model by transforming a DT example from the literature from DTDL-MM to TM-MM. With this experiment, we aim to gain first insights into using UML CD as pivot for porting such models between DT platforms. We chose UML CD as our pivot language for this experiment, because besides the benefits mentioned above, this approach also enables us to integrate existing DT platforms with the technological space that was already built for UML by the scientific community, e.g., for model verification (Gogolla et al. 2007) to name just one prominent example.

5.1. Extending UML CD to DTUML

The gap between UML CD and the metamodels of the examined DT platforms can be closed by extending UML using a profile. By applying this profile to an existing UML CD, the model can be extended to what we call in this paper a DTUML model, which contains all information that is available in the metamodels provided by DT platforms. This created profile consists of different kinds of stereotypes for the different UML CD elements. There are generic stereotypes comprising element attributes that are available in all three examined platforms. Then, we have more specific stereotypes containing the extensions required by two of the three platforms, which extend the generic stereotypes. These stereotypes are again extended by platform-specific stereotypes, which are only available in one specific platform. As an example, the stereotypes that extend the Property element of UML are visualized in Figure 4.

Overall, the following stereotypes are required for the different UML CD elements:

- **All Elements** are extended by a description on VL-MM and DTDL-MM, and an additional displayName, identifier, and version in DTDL-MM.
- Besides these extensions that apply to all elements, **Classes** are extended by a displayName, description, namespace, and version in both TM-MM and DTDL-MM. Additionally, DTDL-MM allows an identifier, VL-MM allows a category, and TM-MM allows the isSingleton and tag information to be modeled. Although tag and category are semantically equivalent, their different syntax requires two dedicated stereotypes here.
- For **Properties**, isTimeSeries and unit can be added to UML CD for all platforms. IsFault and constraintRule are only valid for the VL-MM, quantity is only valid for DTDL-MM, and isExternalId, isStoredLocally, and allowedValues is only valid for TM-MM. Additionally, DTDL requires the unit to be restricted to a specific value based on the semantic type chosen via the quantity attribute. Also, the quantity attribute must be chosen from the list of semantic types available in DTDL (cf. Section Sec. 3.1).
- An **Association** requires only one extension (relationshipType), which is only valid for TM-MM.
- **Operations** require two extensions for VL-MM, namely isBreakable and isMandatory.

5.2. Demonstrating DT Platform Portability using DTUML

To analyze the portability of models used by the individual DT platforms using DTUML, we perform an experiment transforming an existing DT case we created in previous work from DTDL-MM to TM-MM, using DTUML as pivot.

5.2.1. Demonstration Case We use the DT exemplar of a smart room (Govindasamy et al. 2021)to demonstrate portability between DTDL-MM and TM-MM. In this example, the air

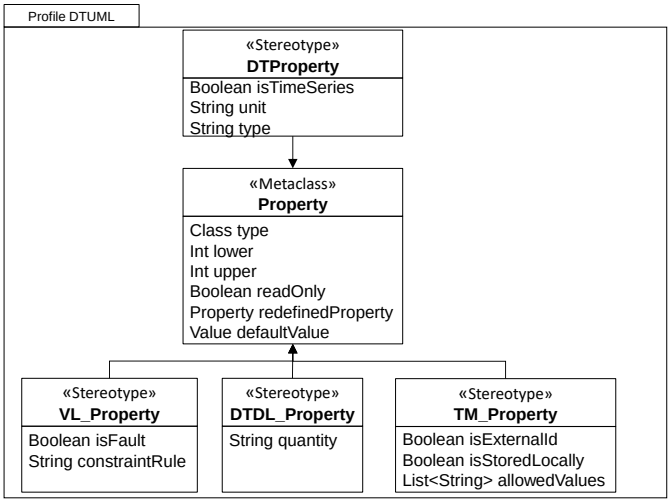

Figure 4 UML profile excerpt for DT platforms. Full profile available on Github[21].

quality of rooms is measured and optimized using different evolution scenarios of the overall system. We use the evolution case 2 described from the encompassing GitHub repository[19], which describes rooms within a building. To measure air quality in these rooms, the building also contains mobile robots that can be moved between rooms. Each robot contains a controller with a temperature and CO_2 sensor and an alarm that is activated once the measured temperature or CO_2 values are too high. If such violations are detected, the ventilation system of the respective room can also be activated to improve air quality.

In the DTDL model of this exemplar, Buildings, Robots, Rooms, VentilationSystems, Controllers, Alarms, TemperatureSensors, and CO2Sensors are modeled as interfaces. The VentilationSystem, TemperatureSensor, and CO2Sensor also have a displayName, which provides a more human-understandable notation (i.e. Ventilation System, Temperature Sensor, CO_2 Sensor) instead of the name attribute that corresponds to the code representation of the class (i.e., VentilationSystem, TemperatureSensor, CO2Sensor). The aqLevel of Room, the value of temperature and CO_2 sensors, and the information whether an alarm is active, are modeled as telemetries, as they represent time-series data omitted by a device. The quantity of the value property of the temperature sensor is set to temperature, and the unit to degree fahrenheit. For the CO_2 sensor, no quantity and unit can be set, as the CO_2 quantity is not included in the enumeration offered by DTDL. The type of a sensor is modeled as non-writable property. Additionally, for each model element, the namespace is set to *at.jku.se.airquality*, the identifier is set equally to the element name, and the version number is set to 1. We also investigated an extended version of this case, including methods and inheritance, and a second case that involves a Lego Mindstorms Car (Munoz et al. 2021). The data for these cases is provided on Github[21].

[19] https://github.com/derlehner/IndoorAirQuality_DigitalTwin_Exemplar/tree/main/digital_twin/models

5.2.2. Required Transformations

To enable the transformation of a model from DTDL-MM to TM-MM, the following transformations are implemented using the Atlas Transformation Language (ATL)[20], and provided on GitHub[21].

The **DTDL2UML** transformation translates an existing DTDL model into UML CD, leveraging the DTDL-specific stereotypes described in Sec. 5.1. A direct mapping without any constraint is possible, as all elements in DTDL can be represented in the DTUML model.

The **UML2TM** transformation translates a DTUML model into a TM model by leveraging appropriate stereotypes of the DTUML profile. If a stereotype is not set on a specific element, it is transformed with default values for elements that are not available natively in UML CD. More specifically, this means that by default, (i) for classes, the isSingleton attribute is false, and the tag attribute is null, and (ii) for properties, the isTimeSeries, isExternalId and isStoredLocally attributes are false, and the allowedValues attribute is null. For the relationshipType attribute added via the TM_Association stereotype, by default, the value is set to *Association* if the isComposition attribute of the source association is false, and to *Composition* if this attribute value is true. However, if the relationshipType attribute is set in the DTUML model, this differentiation between composite and non-composite associations is ignored. Additionally, similarly to the UML2DTDL transformation, (i) abstract classes, non-abstract classes, and interfaces all have to be mapped to Interfaces in TM-MM, and (ii) all additional information from the class diagram that cannot be mapped to TM-MM is omitted during the transformation.

5.3. Results

In our experiment, the initial DTDL model is automatically transformed to DTUML using the DTDL2UML transformation. In the DTUML model, all information from the DTDL model is preserved. However, when creating the TM-MM model, only the unit and isTimeSeries information for properties can be preserved from the extensions that are made to the UML CD metamodel. Other information, such as display names, version numbers, or quantities, cannot be represented in TM-MM, thus must be omitted when applying the UML2TM transformation. Also, namespaces, which are available natively in UML CD, but not in the TM-MM, must be omitted. On the other side, the following information is added to the DTUML model created from the initial DTDL model. Additionally, the Building is annotated as singleton, meaning that a model can only be created for one specific building. For the CO_2 sensor, the unit is set to ppm (as the TM-MM only requires a String here, setting a value here is possible, in contrast to the DTDL-MM). The isExternalId, isStoredLocally attributes are not set for any property, meaning that they will be set to false during the transformation. The allowedValues, isImplementedBy, and tag attributes are also not set, meaning that they are empty in the created TM-MM model. The annotated UML CD is also represented in Figure 5. This model can be transformed using the UML2TM transformation to create the air quality representation in TM-MM.

[20] https://www.eclipse.org/atl/
[21] https://github.com/derlehner/dt_language_comparison

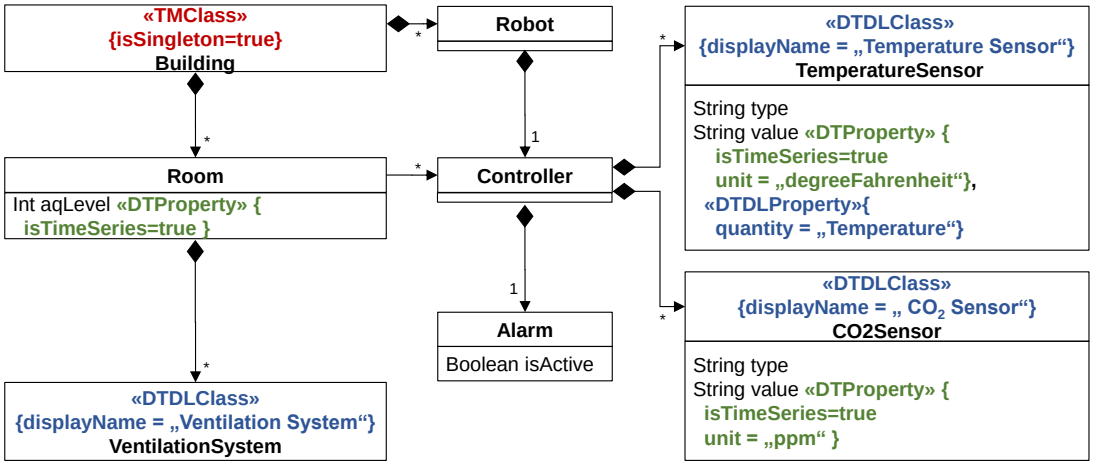

Figure 5 DTUML model for the air quality case (stereotypes added for DTDL and TM).

In summary, these results indicate that for the air quality case, (i) as expceted, there is no adaptation effort to create the DTUML model from the initial DTDL model, (ii) 20 elements of the initial DTDL model cannot be represented in the TM model without additional workarounds, and (iii) 2 elements of the TM model have to be specified in the DTUML model, as they are not available in the DTDL model at all. The annotated DTUML model is shown in Figure 5.

5.4. Answering RQ3: How can existing modeling concepts be leveraged to enable portability of models between these platforms?

In the above experiment, we show that UML CD models seem promising to be used as pivot when transforming models from DTDL-MM to TM-MM, if the DTUML profile is used to enable platform modeling capabilities that are not natively supported in UML CD. However, (i) information that is available in DTDL-MM, but not in TM-MM (unique identifiers, display names, version numbers, namespaces, quantities) is lost in the transformation process, (ii) information that is available in TM-MM, but not in DTDL-MM (measurement unit for CO_2, and values for the isSingleton, isExternalId, isStoredLocally, allowedValues, and tag attributes), can be added in the pivot DTUML model. The information that is lost in (ii) could be added to the TM-MM via the tag attribute of ComponentTypes, and the configuration attribute of PropertyDefinitions. However, this mapping was not scope of the performed experiment.

6. Conclusion

The answers to our research questions provided in the previous sections indicate that modeling capabilities of the examined DT platforms mostly rely on well-known OOM techniques, but they also extend these general-purpose concepts with domain-specific features such as quantities, units, flexible instantiation, etc. In addition, the languages also contain additional constraints which may result from performance considerations in the runtime environments such as constraints for the inheritance relationships and properties. However, when looking into existing contributions on required modeling capabilities to represent DTs or digital shadows, such as the Asset Administration Shell or the digital shadow conceptual model by (Becker et al. 2021), there are also additional aspects which are currently not covered by the languages offered by the platforms, and thus, require dedicated development based on the provided frameworks. Furthermore, there are a multitude of extensions for UML CD for CPS such as for uncertainty, quantities, MARTE, etc. It seems these aspects are often missing in the DT platforms. At the same time, UML CD themselves offer several modeling features which are not provided by the platforms such as abstract classes, uniqueness and ordered constraints for properties, or dedicated inheritance features such as re-definitions. Finally, the evaluation of the DT languages also reveal that based on the life-cycle phase, different concrete syntaxes may be employed. We have seen a wide spectrum of JSON, Xtext/EMF, and API related modeling in the different platforms. For communication and analysis purposes, UML CD may be an option here due to the well-known concrete syntax, but the general application of UML CD for this particular field requires further investigations in addition to the first experiment performed in this work.

A threat to validity is that we only considered the DT platforms of Amazon, Microsoft, and Eclipse. We selected these platforms based on our existing research and experience. However, we plan to look into other DT platforms in the future. Furthermore, we decided to choose UML CD, a general purpose modeling language, as pivot although the languages provided by the platforms are domain-specific. Together with the UML profiles this could be leveraged to develop a platform-independent domain-specific DT language in the future. Besides, we consider the following additional research lines. First, we plan to provide a model-based description of platforms, as proposed by (Thomas et al. 2008; Chehade et al. 2010) for embedded systems platforms. Second, we aim towards a language family for describing DTs including platform specifics (Butting et al. 2020) as well as code generation options. Finally, the evaluation of languages by other platforms is considered interesting for further uncovering the nature of DT modeling. This knowledge can be used to align DT modeling with modeling capabilities in similar domains, e.g., IoT (Kirchhof et al. 2022).

Acknowledgments

This work has been supported by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development (CDG) as well as the German Federal Ministry of Economic Affairs and Climate Action (BMWK, Bundesministerium für Wirtschaft und Klimaschutz) under grant no. 19S21002L.

References

Azangoo, M., Taherkordi, A., & Blech, J. O. (2020). Digital twins for manufacturing using UML and behavioral specifications. In *25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)* (pp. 1035–1038).

Becker, F., Bibow, P., Dalibor, M., Gannouni, A., Hahn, V., Hopmann, C., ... Wortmann, A. (2021). A conceptual model for digital shadows in industry and its application. In *International Conference on Conceptual Modeling (ER)* (pp. 271–281). Springer.

Bibow, P., Dalibor, M., Hopmann, C., Mainz, B., Rumpe, B., Schmalzing, D., ... Wortmann, A. (2020). Model-driven development of a digital twin for injection molding. In *International Conference on Advanced Information Systems Engineering* (pp. 85–100).

Biesinger, F., & Weyrich, M. (2019). The facets of digital twins in production and the automotive industry. In *23rd International Conference on Mechatronics Technology (ICMT)* (pp. 1–6).

Booch, G., Rumbaugh, J. E., & Jacobson, I. (2005). *The Unified Modeling Language user guide* (Vol. 2). Addison-Wesley.

Brambilla, M., Cabot, J., & Wimmer, M. (2017). *Model-driven software engineering in practice*. Morgan & Claypool.

Brauner, P., Dalibor, M., Jarke, M., Kunze, I., Koren, I., Lakemeyer, G., ... Ziefle, M. (2022). A Computer Science Perspective on Digital Transformation in Production. *ACM Trans. Internet Things*, *3*(2), 15:1–15:32.

Broman, D., Lee, E. A., Tripakis, S., & Törngren, M. (2012). Viewpoints, formalisms, languages, and tools for cyber-physical systems. In *6th International Workshop on Multi-Paradigm Modeling* (pp. 49–54).

Bruynseels, K., Santoni de Sio, F., & Van den Hoven, J. (2018). Digital twins in health care: ethical implications of an emerging engineering paradigm. *Frontiers in genetics*, 31.

Burgueño, L., Mayerhofer, T., Wimmer, M., & Vallecillo, A. (2019). Specifying quantities in software models. *Information and Software Technology*, *113*, 82-97.

Butting, A., Pfeiffer, J., Rumpe, B., & Wortmann, A. (2020). A compositional framework for systematic modeling language reuse. In *23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)* (pp. 35–46).

Chehade, W. E. H., Radermacher, A., Gérard, S., & Terrier, F. (2010). Detailed Real-Time Software Platform Modeling. In *Asia Pacific Software Engineering Conference (APSEC)* (pp. 108–117).

Cook, S., Bock, C., Rivett, P., Rutt, T., Seidewitz, E., Selic, B., & Tolbert, D. (2017). *Unified Modeling Language (UML) Version 2.5.1* (Standard). Object Management Group (OMG). Retrieved from https://www.omg.org/spec/UML/2.5.1

Derler, P., Lee, E. A., & Sangiovanni Vincentelli, A. (2012). Modeling Cyber–Physical Systems. *Proceedings of the IEEE*, *100*(1), 13-28.

Eclipse. (2020). *Ecore files representing the Vortolang*. https://github.com/eclipse/vorto/tree/development/core-bundles/meta-model. (Accessed: 2022-02-28)

Eramo, R., Bordeleau, F., Combemale, B., Brand, M. v. d., Wimmer, M., & Wortmann, A. (2022). Conceptualizing digital twins. *IEEE Software*, *39*(2), 39-46.

France, R., & Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. In *Future of Software Engineering (FOSE'07)* (pp. 37–54).

Francisco, A., Mohammadi, N., & Taylor, J. E. (2020). Smart city digital twin–enabled energy management: Toward real-time urban building energy benchmarking. *Journal of Management in Engineering*, *36*(2), 04019045.

Gogolla, M., Büttner, F., & Richters, M. (2007). USE: A UML-based specification environment for validating UML and OCL. *Science of Computer Programming*, *69*(1), 27-34.

Gómez, A., Cabot, J., & Wimmer, M. (2018). TemporalEMF: A temporal metamodeling framework. In *International Conference on Conceptual Modeling (ER)* (pp. 365–381).

Govindasamy, H. S., Jayaraman, R., Taspinar, B., Lehner, D., & Wimmer, M. (2021). Air quality management: an exemplar for model-driven digital twin engineering. In *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)* (pp. 229–232).

Hutchinson, J., Rouncefield, M., & Whittle, J. (2011). Model-driven engineering practices in industry. In *33rd International Conference on Software Engineering (ICSE)* (pp. 633–642).

Jain, P., Poon, J., Singh, J. P., Spanos, C., Sanders, S. R., & Panda, S. K. (2019). A digital twin approach for fault diagnosis in distributed photovoltaic systems. *IEEE Transactions on Power Electronics*, *35*(1), 940–956.

Jimenez, J. I., Jahankhani, H., & Kendzierskyj, S. (2020). Health care in the cyberspace: Medical cyber-physical system and digital twin challenges. In *Digital twin technologies and smart cities* (pp. 79–92). Springer.

Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., ... Wimmer, M. (2006). On models and ontologies - a layered approach for model-based tool integration. In *Modellierung* (pp. 11–27).

Kirchhof, J. C., Michael, J., Rumpe, B., Varga, S., & Wortmann, A. (2020). Model-driven digital twin construction: synthesizing the integration of cyber-physical systems with their information systems. In *23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* (pp. 90–101).

Kirchhof, J. C., Rumpe, B., Schmalzing, D., & Wortmann, A. (2022). MontiThings: Model-Driven Development and Deployment of Reliable IoT Applications. *Journal of Systems and Software*, *183*, 111087.

Kraft, E. M. (2016). The air force digital thread/digital twin-life cycle integration and use of computational and experimental knowledge. In *54th AIAA Aerospace Sciences Meeting*.

Kritzinger, W., Karner, M., Traar, G., Henjes, J., & Sihn, W. (2018). Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, *51*(11), 1016–1022.

Laaki, H., Miche, Y., & Tammi, K. (2019). Prototyping a digital twin for real time remote control over mobile networks: Application of remote surgery. *IEEE Access*, *7*, 20325–20336.

Lehner, D., Pfeiffer, J., Tinsel, E.-F., Strljic, M. M., Sint, S., Vierhauser, M., ... Wimmer, M. (2022). Digital Twin Platforms: Requirements, Capabilities, and Future Prospects. *IEEE Software*, *39*(2), 53-61.

Lehner, D., Sint, S., Vierhauser, M., Narzt, W., & Wimmer, M. (2021). AML4DT: a model-driven framework for developing and maintaining Digital Twins with AutomationML. In *26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)* (pp. 1–8).

Lu, Q., Xie, X., Heaton, J., Parlikad, A. K., & Schooling, J. (2019). From BIM towards digital twin: strategy and future development for smart asset management. In *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing* (pp. 392–404).

Mandolla, C., Petruzzelli, A. M., Percoco, G., & Urbinati, A. (2019). Building a digital twin for additive manufacturing through the exploitation of blockchain: A case analysis of the aircraft industry. *Computers in Industry*, *109*, 134–152.

Munoz, P., Troya, J., & Vallecillo, A. (2021). Using UML and OCL Models to Realize High-Level Digital Twins. In *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)* (pp. 212–220).

Petrasch, R., & Hentschke, R. (2016). Process modeling for industry 4.0 applications: Towards an industry 4.0 process modeling language and method. In *13th International Joint Conference on Computer Science and Software Engineering (JCSSE)* (p. 1-5).

Qi, Q., Tao, F., Hu, T., Anwer, N., Liu, A., Wei, Y., ... Nee, A. (2021). Enabling technologies and tools for digital twin. *Journal of Manufacturing Systems*, *58*, 3–21.

Rasheed, A., San, O., & Kvamsdal, T. (2020). Digital twin: Values, challenges and enablers from a modeling perspective. *IEEE Access*, *8*, 21980–22012.

Schroeder, G. N., Steinmetz, C., Pereira, C. E., & Espindola, D. B. (2016). Digital twin data modeling with AutomationML and a communication methodology for data exchange. *IFAC-PapersOnLine*, *49*(30), 12–17.

Tang, S., Shelden, D. R., Eastman, C. M., Pishdad-Bozorgi, P., & Gao, X. (2019). A review of building information modeling (BIM) and the internet of things (IoT) devices integration: Present status and future trends. *Automation in Construction*, *101*, 127-139.

Tao, F., Cheng, J., Qi, Q., Zhang, M., Zhang, H., & Sui, F. (2018). Digital twin-driven product design, manufacturing and service with big data. *The International Journal of Advanced Manufacturing Technology*, *94*(9), 3563–3576.

Tao, F., Qi, Q., Wang, L., & Nee, A. (2019). Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0: Correlation and comparison. *Engineering*, *5*(4), 653–661.

Tao, F., Zhang, H., Liu, A., & Nee, A. Y. (2018). Digital twin in industry: State-of-the-art. *IEEE Transactions on Industrial Informatics*, *15*(4), 2405–2415.

Thomas, F., Delatour, J., Terrier, F., & Gérard, S. (2008). Towards a Framework for Explicit Platform-Based Transformations. In *11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)* (p. 211-218).

Wortmann, A., Barais, O., Combemale, B., & Wimmer, M. (2020). Modeling languages in Industry 4.0: an extended systematic mapping study. *Software and Systems Modeling*, *19*(1), 67–94.

Xu, Q., Ali, S., & Yue, T. (2021). Digital Twin-based Anomaly Detection in Cyber-physical Systems. In *14th IEEE Conference on Software Testing, Verification and Validation (ICST)* (pp. 205–216).

Yue, T., Arcaini, P., & Ali, S. (2021). Understanding digital twins for cyber-physical systems: A conceptual model. In *Leveraging applications of formal methods, verification and validation: Tools and trends* (pp. 54–71). Springer.

About the authors

Jérôme Pfeiffer is a research assistant at the Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW) of the University of Stuttgart. His research interests include Software Language Engineering techniques and applied Model-Driven Engineering with a focus on digital twins and Industry 4.0. You can contact the author at jerome.pfeiffer@isw.uni-stuttgart.de or visit https://www.isw.uni-stuttgart.de/en/institute/team/Pfeiffer-00005/.

Daniel Lehner is a PhD candidate at the Department of Business Informatics - Software Engineering, and also associated with the Christian Doppler Laboratory for Model-Integrated Smart Production (CDL-MINT), both at Johannes Kepler University Linz. His research interests include applying Model-Driven Engineering techniques and practices to Digital Twins. You can contact the author at daniel.lehner@jku.at or visit https://se.jku.at/daniel-lehner/.

Manuel Wimmer is Full Professor and Head of the Department of Business Informatics – Software Engineering at Johannes Kepler University Linz. His research interests include Software Engineering, Model-Driven Engineering, and Cyber-Physical Systems. You can contact the author at manuel.wimmer@jku.at or visit https://www.se.jku.at/manuel-wimmer/.

Andreas Wortmann is a professor at the Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW) of the University of Stuttgart where he conducts research on model-driven engineering, software language engineering, and systems engineering with a focus on Industry 4.0 and digital twins. You can contact the author at andreas.wortmann@isw.uni-stuttgart.de or visit www.wortmann.ac.