AML4DT: A Model-Driven Framework for Developing and Maintaining Digital Twins with AutomationML

Daniel Lehner*, Sabine Sint*, Michael Vierhauser*, Wolfgang Narzt*, Manuel Wimmer*
* Christian Doppler Labratory for Model-Integrated Smart Production (CDL-MINT)
Institute for Business Informatics - Software Engineering
Johannes Kepler University Linz, Science Park 3, 4040 Linz, Austria
{firstname.lastname}@jku.at

***Abstract*—As technologies such as the Internet of Things (IoT) and Cyber-Physical Systems (CPS) are becoming ubiquitous, systems adopting these technologies are getting increasingly complex. Digital Twins (DTs) provide comprehensive views on such systems, the data they generate during runtime, as well as their usage and evolution over time. Setting up the required infrastructure to run a Digital Twin is still an ambitious task that involves significant upfront efforts from domain experts, although existing knowledge about the systems, such as engineering models, may be already available for reuse.**

To address this issue, we present AML4DT, a model-driven framework supporting the development and maintenance of Digital Twin infrastructures by employing AutomationML (AML) models. We automatically establish a connection between systems and their DTs based on dedicated DT models. These DT models are automatically derived from existing AutomationML models, which are produced in the engineering phases of a system. Additionally, to alleviate the maintenance of the DTs, AML4DT facilitates the synchronization of the AutomationML models with the DT infrastructure for several evolution cases. A case study shows the benefits of developing and maintaining DTs based on AutomationML models using the proposed AML4DT framework. For this particular study, the effort of performing the required tasks could be reduced by about 50%.

***Index Terms*—AutomationML, Digital Twin, Cloud, IoT, Model-Driven Engineering**

I. Introduction

With the steady growth of autonomous systems and Internet of Things (IoT) applications, Cyber-Physical Systems (CPS) are becoming ubiquitous in everyone's daily life. In the context of industrial applications, companies are increasingly adopting Cyber-Physical Production Systems (CPPS) and shop floor automation technologies [1], [2]. As a result, new paradigms for managing and supervising these systems have emerged, with Digital Twins (DTs) as a key enabler of such systems [3]. DTs facilitate a wide range of functionalities, such as monitoring physical assets, improving the understanding of the overall system (e.g., through visualization), and triggering system improvement by supporting predictive maintenance [4]. Previous work has described how different functionalities of DTs use engineering models as common source of information [5], and to automatically collect data from the running system [6]. In general, Model-Driven Engineering (MDE) for DTs is gaining interest [7], [8], and also shows to satisfy DT stakeholder requirements [9]. However, besides the benefits of applying MDE to DTs, several challenges remain, making it difficult to apply DTs in practice [10]. One of these challenges is connecting runtime data sent by devices with existing engineering models. Particular services and architectural components are necessary to enable this integration of data into engineering models and to allow the synchronization between physical devices and their DTs.

One currently emerging solution are DT platforms, offered by various service providers, such as *Azure Digital Twins*[1], *AWS Greengrass*[2], or *Eclipse Hono/Vorto/Ditto*[3]. They provide extensive tool support and software services that can be combined, relying on a common definition of physical devices and their data. However, these definitions are currently proprietary and prevent cross-platform reusability, or the combination of services from different vendors. Moreover, these platform-dependent solutions do not provide support for reuse of existing information available in engineering models. This shortcoming leads to duplicate work for creating and updating both the engineering models and the definition of DTs.

In order to tackle these challenges, the AML4DT framework is proposed. AML4DT builds on a platform-independent metamodel for DTs, which is derived from studying the aforementioned DT platforms (bottom-up view) and object-oriented conceptual modeling approaches (top-down view), to keep information between different services consistent. Additionally, AML4DT facilitates the automated generation of DT models from existing engineering models defined in AutomationML. AML4DT aims to reduce the effort of creating and maintaining DTs by (i) automatically keeping information consistent between different services, and (ii) providing reusability by transforming engineering models to DT models. We evaluate the feasibility of using existing engineering models as input for the presented AML4DT framework, and compare the effort of

[1]https://azure.microsoft.com/services/digital-twins

[2]https://aws.amazon.com/greengrass

[3]https://www.eclipse.org/ditto, https://www.eclipse.org/vorto, https://www.eclipse.org/ditto

978-1-7281-2989-1/21/$31.00 ©2021 IEEE

creating and maintaining DTs with the AML4DT framework against using the tool support directly offered on the platform-specific level by an exemplary DT platform.

The remainder of the paper is structured as follows. In Section II, the background for this work and related approaches are discussed. Next, a motivating example and derived challenges are described in Section III. In Section IV, the approach is presented by providing a detailed description of the framework and its constituent parts. It is evaluated by a case study in Section V. Finally, our work with an outlook on future work is concluded in Section VI.

II. Background & Related Work

In this section, a brief overview of relevant background and related work is provided.

A. Background

The concept of **Digital Twins** was first introduced almost two decades ago by Michael Grieves [11]. A DT is a dynamic virtual model of a system, processor, or service, with data (e.g., sensor data) from physical systems or processes integrated into it. By integrating the digital and physical worlds, the DT enables real-time monitoring of systems and processes and helps, for example, to reduce downtimes and detect errors at an early stage [12]. Another related concept are Digital Shadows (DS). DSs represent a subset of DTs, which are only concerned with connecting the physical device to the virtual model, but not the other way round [13]. In this paper, the approach is using the functionality of a DS (cf. Section V). However, it is refered as DT, since the mentioned concepts also apply, and can be easily extended beyond DS capabilities, e.g., by adding communication channels from the DT back to the device.

Besides the terms DS and DT, in the context of Industry 4.0 (I4.0), the term *Asset Administration Shell (AAS)* is also often discussed, providing the virtual representation and business functionality through operations of the I4.0 component [14]. Interactions, which are required during operation for detailed control and observation of the process flow, are exchanged between the different AASs of the components. Wagner *et al.* [15] use both terms (DT and AAS) synonymously, since they converge against each other. Furthermore, approaches have emerged that leverage Model-Driven Engineering in this context [10].

Model-Driven Engineering (MDE) applies the abstraction power of models to tackle the complexity of systems [16], [17], and thus, is by definition related to the concept of DTs [10]. The central artifact of MDE are formal models to address engineering as well as to drive adoption and to ensure the coherence of model-driven techniques. The process automation as well as traceability of engineering artifacts is supported by various techniques such as transformations, validation, verification, and code generation.

AutomationML (AML)[4] is an open, XML-based, data exchange standard widely used in the automation and manufacturing industry to describe complex structures and geometries [18]. In context of MDE, AML is successfully applied as part of a workbench supporting model validation and automated code generation [19]. It provides an interchange format capable of accommodating different types of engineering data, with a clear distinction between system resources, products, and production processes.

[4]https://www.automationml.org

B. Related Work

Based on this background **related work** is discussed. Beisheim *et al.* [20] show how languages, such as AutomationML, can be integrated into the RAMI 4.0 reference framework. Zhang *et al.* [21] present an approach for CPPS information modeling based on DTs and AutomationML. They show that AML can be successfully used with various encapsulated manufacturing service and integrate the corresponding virtual manufacturing resources (DTs). Pauker *et al.* [22] set out the challenge of seamless communication between the different manufacturing services. For this purpose, OPC-UA is often used in the CPPS domain, but due to the inherent implementation complexity, the full capabilities are not exploited. They overcome this complexity issue by enabling an automatic model transformation of UML class diagrams into OPC-UA information models. Bibow *et al.* [7] use a domain-specific language to specify the communication between DTs and CPPS via OPC-UA. In their approach, they focus on events that may occur in the CPPS. In contrast to these approaches with OPC-UA, our approach aims to provide a general architecture, which can be used to realize DTs and support their evolution.

With regards to modeling DTs, Schröder *et al.* [23] propose the use of higher-level models such as AML models for communication and data exchange between systems that make use of DTs. In a follow-up work [24], they suggest a methodology to deploy DTs based on these AML models. The AML model is used as input for automatic generation of web services and visualization of DTs. In contrast to our work, while their focus is on the DT design using model-driven techniques, they, however, do not explicitly take into account the maintenance and evolution aspect. Kirchhof *et al.* [9] propose a model-driven approach for explicitly modeling DTs, CPS and their integration to facilitate systematic engineering.

Although these approaches already show that AML can be used to model and create DTs, our work contributes to this state-of-the-art by (i) proposing a dedicated, platform-independent metamodel for explicitly modeling DTs, and (ii) providing dedicated support for system evolution.

III. Motivating Example & Challenges

In this section, a concrete motivating example is outlined to make the identified challenges explicit.

A. Motivating Example

Consider a system setup to collect CO_2 measurements using sensors located in rooms within several buildings. The system is designed to provide users with recommendations

on when to ventilate to reduce CO_2 levels, which can increase work performance and comfort and significantly decrease virus infection rates. In order to equip these buildings with such sensors, a building operator buys hardware, more specifically, CO_2 sensors and controllers that retrieve and process the CO_2 measurements from the sensors. For performing recommendations and gaining insights into the overall CO_2 concentration of the entire office building, a platform needs to be established that provides information about the rooms and their CO_2 levels. Thus, a DT representation of the building, its rooms and sensors is needed. This platform is used to collect data, show measurements, and perform analyses to recommend actions.

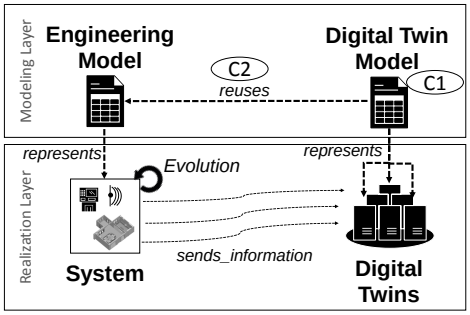

Fig. 1. Overview of the motivating example and the resulting challenges

In order to establish such an infrastructure, different elements are required (cf. Fig. 1). The `System` with its different components has to be specified, e.g., based on an `Engineering Model` such as an AML model. This model defines the structure and relations of the various components and enables transferring them to heterogeneous engineering tool landscapes. Fig. 2 shows an excerpt of the AML model of our CO_2 measurement example.

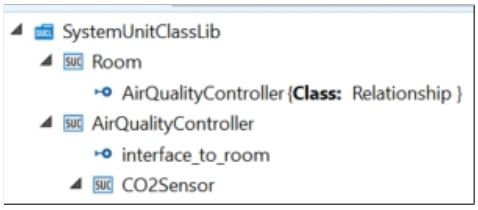

Fig. 2. Excerpt Engineering Model (AML SystemUnitClassLib)

For each component of the implemented system, a `DT` is created to collect the specific information from the system. This DT is implemented using specific platform-dependent tooling. Defining this DT in a platform-independent format via a dedicated `Digital Twin Model` provides abstraction from the used platform and enables platform-independent reuse of the modeled information. By explicitly modeling the DT, even information from engineering models could be automatically reused when creating such DT Models. However, there are some challenges regarding implementation the setting above.

B. Identified Challenges

The setup described involves several manual steps to enter information into a DT platform and keep it consistent between different services. In previous work, different types of services that are required to run a DT are discussed [5]. Even after the initial setup is finished, subsequent updates require additional changes to the different services. For example, if the name of a controller device is changed, this change must be performed (i) on the DT, (ii) on the controller itself to send data to the DT, and (iii) in every service that makes use of data received from this controller, e.g., a time-series database.

To reduce the effort of setting up and maintaining DTs, the following two challenges are stated that are tackled in this work.

Challenge 1 (C1): Explicitly Modeling DTs. To automatically construct and evolve DTs, they should have an open specification. In existing DT platforms, DTs are directly implemented in a platform-specific manner. As an example, Listing 1 shows the JSON code to realize a part of the motivating example in the MS Azure Digital Twin platform[1]. Although this is of course an option, it clearly misses to reuse already defined information in models.

Listing 1. Example JSON Code for a room DT representation

```
[{ "@type": "Interface", "displayName": "Room",
   "@id": "dtmi:Room;1",
   "contents":[{
     "@type": ["Relationship"],
     "displayName": "airQualityControllers",
     "@id": "dtmi:Room:airQualityControllers;1",
     "target": "dtmi:AirQualityController;1",
     "writable": true,
     "name": "airQualityControllers"}],
   "@context": "dtmi:dtdl:context;2" },
 { "dtid": "Lobby100",
   "content": {"$metadata": {"$model": "dtmi:com:
       example:Room;2"}},
   "relationships":[{
     "id": "rel1",
     "content":{ "$targetId": "Raspberry1", "
         $relationshipName": "airQualityControllers"
         } }]
}]
```

Challenge 2 (C2): Reusing Engineering Models for DT Generation. Various service providers already offer DT platforms. However, currently there is a lack of standards and reuse of engineering models. Knowledge provided by such engineering models should be reused for DTs. There are already approaches that realize this, e.g., for simulation models [25]. However, an adequate mapping from industry standards to DT models used by DT platforms is needed to enable automated transformation and generation, thus to reduce redundant manual effort.

IV. AML4DT Framework

Now the AML4DT framework is described to address the aforementioned challenges.

A. Overview

Fig. 3 provides an architectural overview of the AML4DT framework. It consists of two layers and uses a model-driven approach to automate the creation and evolution tasks of DTs.

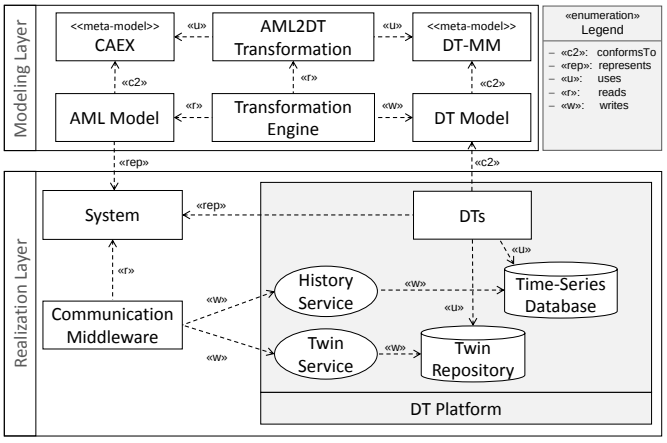

Fig. 3. Architectural overview of AML4DT

In the `Realization Layer`, the physical `System`, which consists of a multitude of interconnected devices, and its corresponding `DTs` are running. The system provides information regarding (i) its own structure (e.g., number, type and relations between devices), and (ii) value streams (e.g., continuously measured sensor data). Over a `Communication Middleware`, through a standardized interface, the collected data is sent to the different services of the `DT Platform` for further processing. More specifically, (i) the `Twin Service` is used to store the current structure of a system at runtime in the `Twin Repository`, and (ii) the `History Service` is used to forward measured sensor data to a `Time Series Database`.

The `Modeling Layer` provides the necessary capabilities for the automation support. It consists of (i) the models that represent the respective entities of the realization layer, (ii) their metamodels, as well as (iii) the transformation between the used metamodels for automation support. To tackle Challenge 1, a `Digital Twin Metamodel (DT-MM)` is proposed that facilitates the creation of models representing types and instances of DTs. It is used as common source of information for the software components of the DT platform. To overcome Challenge 2, AutomationML as representation of existing engineering models is used and a mapping to DT-MM via the `AML2DTTransformation` is provided. As a result, the automated generation of `DT models` describing the DT of the physical system from existing `AML models` representing the physical system is supported.

B. AML Metamodel

In order to establish one common source of information and facilitate the transformation between AML and DT models, metamodels are required. For creating AML models representing a physical system, the **CAEX metamodel** from previous work is reused (cf. Fig. 4). The reader interested in the full CAEX metamodel is kindly referred to [19]. Here a short summary is provided. In the metamodel, both type and instance level are considered. The components of the system are represented as `InternalElements`, which are instances of `SystemUnitClasses`, organized in

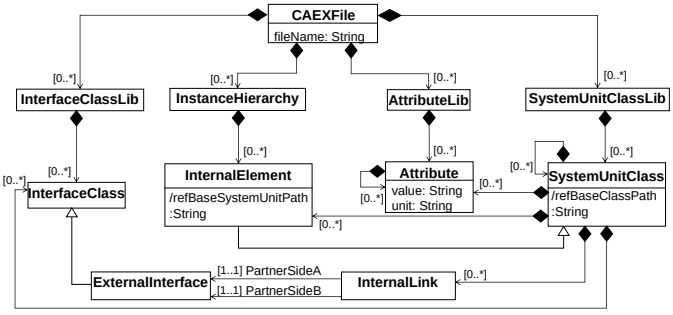

Fig. 4. Excerpt CAEX Metamodel based on [19]

an `InstanceHierarchy`. Compositions of components are represented by building a hierarchy among InternalElements. An InternalElement might have `Attributes` as well as links based on interfaces (`ExternalInterface`,`InternalLink`). The `SystemUnitClassLib` consists of several SystemUnitClasses, each of them may contain `InterfaceClasses`, InternalElements, Attributes, or nested SystemUnitClasses.

C. DT Metamodel

Based on our investigations of DT platforms, such as Azure Digital Twins[1], Eclipse Hono/Vorto/Ditto[3], and AWS Greengrass[2], as well as conceptual modeling languages such as UML, a generic **DT Metamodel (DT-MM)** is proposed for representing DTs, their data, as well as their connections (cf. Fig. 5). The `DigitalTwinPlatform` is the main element and describes relevant meta-information to find required parts of the DT platform. This environment comprises a list of `DT_Types` and `DT_Instance`. DT_Types describe the schema of DTs and consist of `Properties`, `Relationships` to other DT_Types, and nested DT_Types. Properties represent data points of the DT and can be `writable` (i.e., `Value` changeable during system execution), `loggable` (i.e., historical values are persisted in a Time Series Database), and have a certain `DataType` (e.g., String, Integer). Relationships specify their cardinality based on the `minimal and maximal occurrence`, as well as whether it is `writable` during runtime. DT_type, property and relationship inherit from the abstract `VersionableNamedElement`. Thus, they

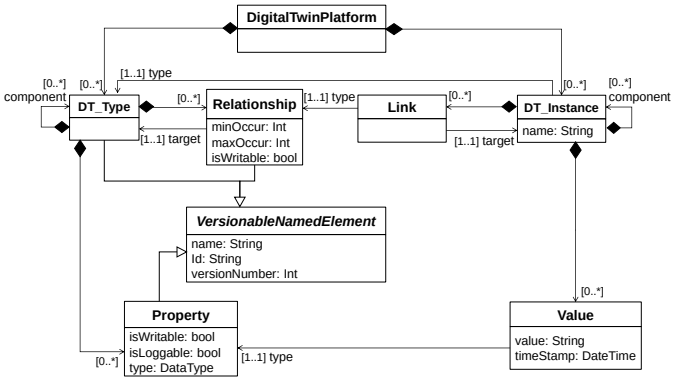

Fig. 5. Digital Twin Metamodel (DT-MM)

can be uniquely identified by a persisted `id`. Additionally, they have a `name` and a `versionNumber` to track changes of the corresponding element.

DT_Instances are concrete instantiations of DT_Types and each have a unique name that identifies the DT_Instance within its DigitalTwinPlatform, as well as a reference to the corresponding DT_Type. Based on the schema imposed by the referenced DT_Type, a DT_Instance can contain values, `Links` and further DT_Instances. A value represents an instantiation of a property of the corresponding DT_Type. It contains a specific value that has to conform to the DataType specified by the corresponding property (for simplification reasons, a serialized version of the value represented in a String format is used) and a `timeStamp` (i.e., `DateTime` at which the respective value is measured or has changed). Links are instantiations of relationships and must have as `target` a DT_Instance which conforms to the DT_Type that is target of the corresponding relationship.

D. Mapping of AML to DT-MM

To realize the mapping from AML to DT-MM, extensions to the CAEX language by additional libraries are required. First, two attribute types *Loggable* and *Constant* are added to indicate whether a given attribute is history-aware and writeable. By default, attributes without a specific type are mapped to non-loggable writable properties. Second, the InterfaceClass *Relationship* is added to annotate (i) the cardinality of a relationship (minOccur, maxOccur), and (ii) whether a relationship is writable (isWriteable). If an ExternalInterface in an AML model is not annotated as relationship, it is ignored.

For ExternalInterfaces that are contained by InternalElements, this annotation is not applicable, as the corresponding link in DT-MM does not require any further specification. The reference between a link and its corresponding relationship is established via name equivalence of the respective InternalLinks.

Based on these extensions, it is possible to perform model-to-model transformations from AML to DT-MM. Therefore, an outplace transformation is used, where the elements of AML are mapped to the components of DT-MM. Table I summarizes the mappings between the elements of the two metamodels.

E. Prototypical Implementation

For our approach, a prototypical implementation is developed. Therefore, the Eclipse Modeling Framework (EMF)[5] is used, in particular the Atlas Transformation Language (ATL)[6] due to its wide range of functionality and acceptance in the scientific community for model transformation and Xtend[7] for automatic generation of JSON files needed for the DT platform.

Currently, as a demonstrator, a mapping to Microsoft's Azure platform is provided, in particular the Azure Digital Twins service (ADT)[1]. The setup of the DTs is based on our DT model and performed automatically via JSON files. For storing and visualizing historical values, the Azure Time Series Insights (TSI)[8] service is used. In the future, the approach will be extended to other DT platforms.

For system evolution, (i) adding new DT_Instances, components of DT_Instances and Links, (ii) updating Values and names of DT_Instances, and (iii) deleting DT_Instances, components, and Links are supported.

The full implementation and further details are provided on GitHub[9].

TABLE I
MAPPING OF THE CAEX METAMODEL TO THE DT-MM

CAEX	**DT-MM**
SystemUnitClass	DT_Type
SystemUnitClass/Attribute	Property *isWritable=true, isLoggable=false*
SystemUnitClass/Attribute *Type: Constant*	Property *isWritable=false, isLoggable=false*
SystemUnitClass/Attribute *Type: Loggable*	Property *isWritable=true, isLoggable=true*
SystemUnitClass/InternalLink *PartnerSide[A|B]: Relationship*	Relationship
InternalElement	DT_Instance
InternalElement/Attribute	Value
InternalElement/InternalLink	Link

V. EVALUATION

With the evaluation the automation potential should be shown when using AML4DT for developing and maintaining DTs. The aim is to demonstrate that (i) the creation of models, serving as input for AML4DT, is possible with reasonable effort, (ii) the AML4DT framework can provide required DT functionality using these models, and (iii) the AML4DT has automation potential for setting up and maintaining a DT.

For this purpose, the effort required for developing and maintaining DTs using AML4DT against a traditional approach not leveraging model-driven techniques is compared. Thus, a case study is conducted following the guidelines by Runeson & Höst [26].

A. Research Questions

Based on the overall goals, the following research questions (RQs) are defined.

RQ1 (Feasibility): Is it possible to describe a DT use case and derive the virtual elements using AML4DT, involving reasonable adaptation effort to the initial AML model?

RQ2 (Effort of Initial Setup): What is the effort of setting up the DT infrastructure using AML4DT, compared to a traditional setup without AML4DT?

RQ3 (Effort of System Evolution): What is the effort of maintaining the system during evolution using AML4DT, compared to a traditional maintenance without AML4DT?

[5]https://www.eclipse.org/modeling/emf
[6]https://www.eclipse.org/atl
[7]https://www.eclipse.org/xtend
[8]https://azure.microsoft.com/de-de/services/time-series-insights
[9]https://github.com/derlehner/etfa2021

B. Case Study Design

Requirements. As a suitable input for our case study, a system is needed, in which different devices are in relation with each other and we can observe measurements by different sensors. It has to be possible to exchange data by a communication interface. It is necessary to provide a storage for historical data and to have an application to visualize and analyze this historical data and metadata.

Use Cases. In our case study, the following two use cases are used. For each use case, existing AML models are applied, which are adapted and serve as input for the AML4DT framework.

Use Case 1 (UC1) extends our motivating example described in Section III. In this scenario, sensors are connected to controllers that report CO_2 values for a particular room in a building. The initial setting comprises three rooms, each equipped with a Raspberry Pi that acts as controller that is connected to a CCS811 CO_2 sensor. The sensor periodically captures the CO_2 measures of the respective room and sends them to a cloud server that implements a DT using MS Azure (ADT-service as information basis, TSI-service for querying).

Use Case 2 (UC2) involves computer-controlled transportation vehicles that, for example, move items during a production process. The created AML model for these cars is based on previous work [27]. In our use case, two cars (*Car1* and *Car2*) are used, where Car1 is connected to Car2 and can thereby adjust its speed and direction. Each car consists of distance sensors, a motor control, and a servo control.

Evaluation Settings and Evolution Cases. To evaluate the effort of our approach (RQ2/3), the following two settings for UC1 were deployed and compared.

In the *AML4DT setting*, our prototypical implementation (cf. Section IV-E) is used, alongside with the AML model. The Python script that sends the CO_2 values to the cloud uses information from the AML model (a JSON file deployed on the Raspberry Pi) to retrieve the correct name of the sensor.

In the *traditional setting*, only tools and services provided by Microsoft Azure are used. In addition, two internal variables in the Python script contain the name of the AirQualityController that is used for mapping the data correctly in the DT platform. This means that consistency between the DT platform services and the controller code has to be ensured manually.

To investigate the required effort for maintaining a system over time, four evolution cases for both settings are evaluated.

Evolution Case 1 (EC1): Add a new raspberry. A new Raspberry Pi is added to the system, together with a corresponding CO_2 sensor. In a first step, the Raspberry Pi is used for testing new functions and is not assigned to a specific room. But data is sent to the DT.

Evolution Case 2 (EC2): Add raspberry to room. The device (setup in EC1) is assigned to a particular room (Room201) to report values for this specific room.

Evolution Case 3 (EC3): Change name of raspberry. DT evolution may also require modifications of existing elements. As an example case, the name of one Raspberry Pi is changed.

Evolution Case 4 (EC4): Delete Room. Besides creating and updating elements, they can also be deleted. As an example, we delete a room from our system as it is no longer monitored.

Evaluation Metrics. For RQ1, two metrics are used and evaluated by UC1 and UC2. The *required changes* represent the effort that are needed to adapt an existing AML model so that it can serve as input for AML4DT. The number of modifications (i.e., tagging existing elements) are counted that needed to be performed so that the existing model could be used with AML4DT. The *supported DT capabilities* represent the functionality expected from a DT that can be achieved using the AML4DT framework. Thereby, the following functionality is analyzed: (i) visualization of the system in the DT, (ii) sending data from physical device to the DT, and (iii) querying historical data. For each of these functionalities that is actually available after executing the AML4DT framework, the value of the supported DT capabilities metrics is incremented. To answer RQ2 and RQ3, the number of required changes to certain artifacts is used to measure the effort for performing particular actions. Each investigated action is performed for both evaluation settings and the changes are counted in the process. In the AML4DT setting, changes have to be performed only on the AML model. In the traditional setting, changes are performed on the ADT-service, TSI-service, and in the controller and service code. For JSON files, each property that has to be created or updated is counted as one change.

C. Results

In the following, the results of our case study are presented.

Results RQ1. Fig. 6 shows an excerpt InstanceHierarchy of the AML model belonging to the SystemUnitClassLib used for the different rooms and their CO_2 sensors (cf. Fig. 2). This AML model was used as input to the AML4DT framework to automatically generate the DT model. Three changes were required. The *Relationship* Class was added to the ExternalInterface of room, the *Constant* type was added to attribute type and *Loggable* type was added to attribute co2Value of CO2Sensor. Using the resulting model, (i) DTs of the system could be visualized, (ii) CO_2 data could be sent from the Raspberry Pis to the respective DTs, and (iii) the historical CO_2 values could be queried using the TSI-service. For UC2, the realization from AML to the DT platform required 9 *changes*. *Loggable* types had to be set to 6 different attributes,

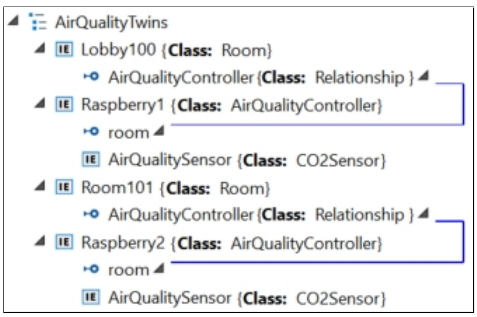

Fig. 6. Excerpt AML InstanceHierarchy of UC1.

Constant types had to be set to 2 different attributes, and the *Relationship* class had to be added to one ExternalInterface. Using the resulting model, as for UC1, all DT capabilities could be realized. Thus, in both cases, the *supported DT capabilities* are 3 out of 3.

Results RQ2. For calculating the effort of the setup procedure, the overall process was divided into two main parts. First, the types were added, second, specific instances were created.

In the AML4DT setting, for the setup of types, two SystemUnitClasses (Room, AirQualityController) as well as one inner SystemUnitClass (Sensor) were added. This aggregated to 6 changes in the model for adding and renaming them. To establish the needed relationships, interfaces and their multiplicity and writable properties were set. To create the type and the co2value properties for the SystemUnitClass Sensor, 7 additional model changes were required. Based on those types, the respective InternalElements were created and InternalLinks between the ExternalInterfaces of rooms and controllers were added and the values of the type property were set for sensors. Table II shows that the setup for types and instances requires 18 operations each for the AML4DT framework.

In the traditional setting, the ADT-service JSON-files were uploaded for each type (Room, AirQualityController, Sensor). To create these JSON-files, various properties had to be defined for each type. In the TSI-service, a type was added for sensor, since it contained a history property (co2value) and a variable was added. To setup instances based on these types, first in the ADT-service, the rooms and controller were created and renamed. Links had to be established and values for the sensors were set. Second, in the TSI-service, an instance for each sensor was created. In total, the setup of types and instances required 80 operations (cf. Table II).

Results RQ3. For analyzing the maintenance support, the four different ECs were evaluated for the two evaluation settings. Table II shows the required operations for each EC based on the different evaluation settings. In EC1, the corresponding instance for the controller had to be created in the DT platform. In the AML4DT setting, this was achieved by adding the new InternalElement and adapting its name accordingly. In the traditional setting, add and rename changes were performed in the ADT-service, the TSI-service and in the code. For EC2, the new instance had to be created and renamed, and a link between room and controller had to be established. These changes had to be made in both the AML model for the AML4DT setting and the ADT-service for the traditional setting. In EC3, only the name of the corresponding InternalElement was updated in the AML model for the AML4DT setting. In the traditional setting, the renaming had to be done in the ADT-service, TSI-service and deployed code. For deleting purpose in EC4, in the AML4DT setting the room and corresponding controller were deleted from the AML model. In the traditional setting, elements had to be deleted in the ADT-service and TSI-service.

TABLE II
CHANGE OPERATIONS FOR AML4DT (CHANGES IN THE AML MODEL) AND THE TRADITIONAL SETTING (ADT, TSI, AND LOC CHANGES)

Setting	**AML4DT**	**Traditional**		
	AML	ADT	TSI	Code
Setup Types	18	3	4	43
Setup Instances	18	18	9	3
Sum for Setup (RQ2)	**36**		**80**	
EC1: Add Raspberry	2	2	2	1
EC2: Add Raspberry to Room	3	3	0	0
EC3: Change name of Raspberry	1	1	1	1
EC4: Delete Room	2	3	1	0
Sum for ECs (RQ3)	**9**		**12**	
Overall Sum	**45**		**92**	

D. Discussion

Answering RQ1. For both use cases, it was possible to achieve all examined *supported DT capabilities* using our AML4DT framework. For UC1, this required 3 annotations and for UC2 9 annotations. This shows that DTs can be described by AML and that the required infrastructure can be automatically created by AML4DT.

Answering RQ2. In terms of the required effort, it can be concluded that our approach did reduce the effort required to create and set up DTs. The number of changes is reduced by 55%. A key advantage is the single source where changes are made in our approach compared to the traditional one where manual changes have to be performed in three different places.

Answering RQ3. Regarding the evolution cases, using AML4DT yields a 25% decrease in changes compared to the traditional setting. While the reduction of required changes was not possible in EC2, for EC3, in which just the name of the Raspberry was changed, a 67% reduction could be achieved. In none of the cases, the traditional approach required fewer changes than AML4DT.

E. Threats to Validity

AML4DT is applicable to real systems, however, our case study uses only a limited number of devices and development scenarios. Therefore, generalizability beyond the case study is not possible. Additional evaluations are required to ensure that AML4DT is applicable to a wider range of systems. Our approach is demonstrated using an established DT platform. To demonstrate applicability to a broader range of technologies, the approach needs to be extended to support different platforms as subject of future work. Our case study only deals with structural changes for evolution. It remains open as future work, what happens with data, e.g., from deleted devices. In addition, our evaluation only shows the communication from DT to the physical hardware. The communication the other way around needs further investigations.

VI. CONCLUSION AND FUTURE WORK

In this paper, AML4DT is presented, a model-driven framework for development and maintenance of DT infrastructures.

Using a dedicated DT metamodel for abstracting from concrete platforms and AutomationML as engineering model, AML4DT automatically generates the necessary infrastructure for establishing a connection between hardware devices and their DT representation linking even back to the engineering models. The evaluation shows the feasibility of the framework and its automation potential by applying it to a case study with multiple sensor devices integrated into the MS Azure DT platform. Using our framework reduces the effort for setting up and maintaining DTs by 51% in the used setting. More specifically, the effort for system setup was reduced by 55% and the effort for evolving the system was reduced by 25%.

As part of our future work, we aim to extend our framework for supporting emerging initiatives to standardize DT platforms (e.g., the *Industrial Digital Twin Initiative*[10]) and find common representations for DTs (e.g., the AML component description [28]). Additionally, the metamodel will be extended, e.g., to cover digital process twins [29] or variability modeling [30]. Another interesting path is to allow reverse-engineering of engineering models from DT models. This would allow the representation of existing DTs by well-known standards, and further enable the propagation of changes on the DTs back to engineering models. Besides extending the proposed framework, we also envision positioning our approach in the context of a general manufacturing process, e.g., as shown by Yli-Ojanperä *et al.* [31].

Acknowledgment

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).

References

[1] L. Monostori, "Cyber-Physical Production Systems: Roots, expectations and R&D Challenges," *Procedia CIRP*, vol. 17, pp. 9–13, 2014.

[2] B. Vogel-Heuser, C. Diedrich, D. Pantförder, and P. Göhner, "Coupling heterogeneous production systems by a multi-agent based cyber-physical production system," in *Proc. of INDIN*, pp. 713–719, IEEE, 2014.

[3] T. H.-J. Uhlemann, C. Lehmann, and R. Steinhilper, "The digital twin: Realizing the cyber-physical production system for Industry 4.0," *Procedia CIRP*, vol. 61, pp. 335–340, 2017.

[4] M. Sjarov, T. Lechler, J. Fuchs, M. Brossog, A. Selmaier, F. Faltus, T. Donhauser, and J. Franke, "The Digital Twin Concept in Industry – A Review and Systematization," in *Proc. of ETFA*, pp. 1789–1796, IEEE, 2020.

[5] D. Lehner, S. Wolny, A. Mazak-Huemer, and M. Wimmer, "Towards a Reference Architecture for Leveraging Model Repositories for Digital Twins," in *Proc. of ETFA*, pp. 1077–1080, IEEE, 2020.

[6] A. Mazak, A. Lüder, S. Wolny, M. Wimmer, D. Winkler, K. Kirchheim, R. Rosendahl, H. Bayanifar, and S. Biffl, "Model-based generation of run-time data collection systems exploiting AutomationML," *Autom.*, vol. 66, no. 10, pp. 819–833, 2018.

[7] P. Bibow, M. Dalibor, C. Hopmann, B. Mainz, B. Rumpe, D. Schmalzing, M. Schmitz, and A. Wortmann, "Model-Driven Development of a Digital Twin for Injection Molding," in *Proc. of CAiSE*, pp. 85–100, Springer, 2020.

[8] M. Azangoo, A. Taherkordi, and J. Olaf Blech, "Digital Twins for Manufacturing Using UML and Behavioral Specifications," in *Proc. of ETFA*, pp. 1035–1038, IEEE, 2020.

[10]https://idtwin.org/

[9] J. C. Kirchhof, J. Michael, B. Rumpe, S. Varga, and A. Wortmann, "Model-driven digital twin construction: synthesizing the integration of cyber-physical systems with their information systems," in *Proc. of MoDELS*, pp. 90–101, ACM, 2020.

[10] F. Bordeleau, B. Combemale, R. Eramo, M. van den Brand, and M. Wimmer, "Towards Model-Driven Digital Twin Engineering: Current Opportunities and Future Challenges," in *Proc. of ICSMM*, pp. 43–54, Springer, 2020.

[11] M. Grieves, "Digital Twin: Manufacturing Excellence through Virtual Factory Replication," tech. rep., LLC: Melbourne, 03 2014.

[12] F. Tao, H. Zhang, A. Liu, and A. Nee, "Digital twin in industry: State-of-the-art," *IEEE TII*, vol. 15, pp. 2405–2415, 2019.

[13] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital Twin in manufacturing: A categorical literature review and classification," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1016–1022, 2018.

[14] J. Fuchs, J. Schmidt, J. Franke, K. Rehman, M. Sauer, and S. Karnouskos, "I4.0-compliant integration of assets utilizing the Asset Administration Shell," in *Proc. of ETFA*, pp. 1243–1247, IEEE, 2019.

[15] C. Wagner, J. Grothoff, U. Epple, R. Drath, S. Malakuti, S. Grüner, M. Hoffmeister, and P. Zimermann, "The role of the Industry 4.0 asset administration shell and the digital twin during the life cycle of a plant," in *Proc. of ETFA*, pp. 1–8, IEEE, 2017.

[16] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*. Morgan & Claypool, 2017.

[17] D. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[18] R. Drath, A. Lüder, J. Peschke, and L. Hundt, "AutomationML-the glue for seamless automation engineering," in *Proc. of ETFA*, pp. 616–623, IEEE, 2008.

[19] T. Mayerhofer, M. Wimmer, L. Berardinelli, and R. Drath, "A Model-Driven Engineering Workbench for CAEX Supporting Language Customization and Evolution," *IEEE TII*, vol. 14, no. 6, pp. 2770–2779, 2018.

[20] N. Beisheim, M. Kiesel, M. Linde, and T. Ott, "Using AutomationML and Graph-Based Design Languages for Automatic Generation of Digital Twins of Cyber-Physical Systems," in *Transdisciplinary Engineering for Complex Socio-technical Systems – Real-life Applications*, vol. 12, pp. 135 – 142, 2020.

[21] H. Zhang, Q. Yan, and Z. Wen, "Information modeling for cyber-physical production system based on digital twin and AutomationML," *Journal of Advanced Manufacturing Technology*, pp. 1–19, 2020.

[22] F. Pauker, S. Wolny, S. M. Fallah, and M. Wimmer, "UML2OPC-UATransforming UML Class Diagrams to OPC UA Information Models," *Procedia CIRP*, vol. 67, pp. 128–133, 2018.

[23] G. N. Schroeder, C. Steinmetz, C. E. Pereira, and D. B. Espindola, "Digital twin data modeling with AutomationML and a communication methodology for data exchange," *IFAC-PapersOnLine*, vol. 49, no. 30, pp. 12–17, 2016.

[24] G. N. Schroeder, C. Steinmetz, R. N. Rodrigues, R. V. B. Henriques, A. Rettberg, and C. E. Pereira, "A Methodology for Digital Twin Modeling and Deployment for Industry 4.0," *Proc. of IEEE*, vol. 109, no. 4, pp. 556–567, 2021.

[25] C. Härle, M. Barth, and A. Fay, "Assistance system for the automated composition and configuration of a co-simulation," in *Proc. of ESM*, 2020.

[26] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empir. Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2009.

[27] S. Wolny, A. Mazak, and M. Wimmer, "Automatic Reverse Engineering of Interaction Models from System Logs," in *Proc. of ETFA*, pp. 57–64, IEEE, 2019.

[28] R. Drath, M. Rentschler, and M. Hoffmeister, "The AutomationML Component Description in the context of the Asset Administration Shell," in *Proc. of ETFA*, pp. 1278–1281, IEEE, 2019.

[29] B. Caesar, A. Hänel, E. Wenkler, C. Corinth, S. Ihlenfeldt, and A. Fay, "Information Model of a Digital Process Twin for Machining Processes," in *Proc. of ETFA*, pp. 1765–1772, IEEE, 2020.

[30] A. Garmendia, M. Wimmer, A. Mazak-Huemer, E. Guerra, and J. de Lara, "Modelling Production System Families with AutomationML," in *Proc. of ETFA*, pp. 1057–1060, IEEE, 2020.

[31] M. Yli-Ojanperä, S. Sierla, N. Papakonstantinou, and V. Vyatkin, "Adapting an agile manufacturing concept to the reference architecture model industry 4.0: A survey and case study," *Journal of Industrial Information Integration*, vol. 15, pp. 147–160, 2019.