IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 9 August 2023, accepted 31 August 2023, date of publication 5 September 2023, date of current version 15 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3312286

STANDARDS

Overview and Comparison of Asset Information Model Standards

TORBEN MINY[1], (Member, IEEE), MICHAEL THIES[1], (Member, IEEE), LINA LUKIC[2], (Member, IEEE), SEBASTIAN KÄBISCH[3], (Member, IEEE), KAZEEM OLADIPUPO[4], (Member, IEEE), CHRISTIAN DIEDRICH[4], (Member, IEEE), AND TOBIAS KLEINERT[1], (Member, IEEE)

[1]Chair of Information and Automation Systems for the Process and Material Technology, RWTH Aachen University, 52064 Aachen, Germany
[2]SysTec Systemtechnik und Industrieautomation GmbH, 50129 Bergheim, Germany
[3]Siemens AG, 81739 Munich, Germany
[4]Chair of Integrated Automation, Otto von Guericke University Magdeburg, 39106 Magdeburg, Germany

Corresponding author: Torben Miny (t.miny@iat.rwth-aachen.de)

ABSTRACT Different organizations are currently working on concepts and standards pertaining to the integration of industrial automation devices into a communication network. For manufacturers, suppliers, integrators, and operators of automation components, the variety of available protocols for information exchange raises the question of which standard to use. To address this question, this contribution provides an overview of different standards for the virtual description of an automation device in the context of device integration and presents a detailed comparison of the following selected standards: W3C WoT Thing Description, Asset Administration Shell, Digital Factory Framework, Automation Markup Language, Module Type Package, OPC UA Process Automation - Device Information Model, and Field Device Integration. These standards are compared with respect to four categories: 1) Representation of a property; 2) Representation of services; 3) Information modeling for direct automation device access; and 4) Mechanism for discovery. The comparison is summarized in an evaluation of the suitability of each standard for different use cases. Since none of the standards fully covers all use cases generic integration strategies are presented for combining the device information models. Finally, a description of a demonstration showcasing this integration, including an implementation as a proof of concept, concludes this contribution.

INDEX TERMS Asset administration shell (AAS), automation markup language (AML), field device integration (FDI), industrial automation device, integration standards, module type package (MTP), OPC UA process automation–device information model (PA-DIM), W3C web of thing–thing description (W3C TD).

I. INTRODUCTION

The integration of industrial automation devices (e. g., field devices or package units) in a communication network is a recurring task, especially in production systems of higher flexibility [1]. A standardized integration process preferably applicable to a large variety of automation devices could simplify the path towards effective data communication, data usage and automation device control. For example, in the process industry where a single plant can have several thousand of automation devices, standardized and efficient integration processes can add significant value e. g. with respect to accelerated commissioning. In an Industry 4.0 (I4.0) scenario, communication and interpretation of data are automatically accomplished by every connected device performing system functionalities [2]. To achieve this in a simple manner, uniform industry standards for information exchange are especially important [3], instead of technology-specific solutions. For future development, it is therefore necessary that standards provide protocol-agnostic interoperability, utilizing protocol bindings for different communication technologies.

Different organizations are currently working on this topic, e. g., the Industry IoT Consortium (IIC) with its "Industrial Internet of Things" (IIoT) [4], the Institute of Electrical and

The associate editor coordinating the review of this manuscript and approving it for publication was Gaetano Zizzo.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

Electronics Engineers (IEEE) with its "Architectural Framework for the Internet of Things" [5], the European Telecommunications Standards Institute (ETSI) with its "Smart machine-to-machine communication" (SmartM2M) [6], or the World Wide Web Consortium (W3C) with its Web of Things [7]. One goal of these activities is to provide the following information about automation devices:

- What kind of data and functionalities does the automation device provide and is capable of (Capability)?
- How can an application access the data and the invocable functionalities (Service)?
- What kind of protocols and serializations are used?
- What kind of security constraints are applied?
- Are there relationships to other automation devices?
- How is the exchanged information structured?

The following scenario is intended to illustrate a desired way for automated integration of new automation devices: A plant operator modifies one of his plants by integrating a new automation device. Therefore, he purchases the needed automation device and connects it to the network of the automation system. The automation device immediately becomes discoverable in the network, and other applications can access the information about the automation device. This information is used by other applications or automation devices to determine whether they need to interact with this automation device and, if so, to obtain information about the accessibility (where and how) of the automation device. Moreover, the application can find out how to establish a connection to the stored information and, if needed, how to access the automation device itself, e. g., for configuration or actuation.

To achieve this scenario and enable automation devices to be discernible to computer programs and automated decision or optimization algorithms, a **virtual representation** of the automation device is required. This virtual representation supports the automatic exchange of machine-processable information between different automation devices and automation systems throughout the life cycle of an automation device. This exchange includes information about the automation device itself as well as about its communication skills including its semantics and data structures. These virtual representations, also known as ''device descriptions'', ''digital representations'' or ''digital twins'', are used in various environments [8].

To facilitate information exchange between heterogeneous software systems, a common information and data model is necessary to define the virtual description, semantics, and serialization format. In setups where components from different suppliers participate in the information exchange, these models should be provided according to existing industry standards.

Usually originating from the cooperation between multiple parties, standards are typically promoted by different organizations, each providing their own concepts: For manufacturers, suppliers, integrators and operators of automation components, the choice of which standard to use for information exchange raises questions. This is discussed in various industrial user organizations or communities of interest, e. g., NAMUR,[1] ZVEI,[2] or VDI.[3]

Current research investigates existing standards, the problems they can solve, how implementation of existing standards can be interconnected or used together, and the need for further development [9], [10]. Therefore, it is necessary to survey standards for exchanging information about automation devices, compare these standards, and describe and classify the different goals, application areas and information modeling concepts. While a few standards like IEC 63278-1 [11] provide high-level comparisons to other standards, detailed overview is lacking.

Thus, this paper aims to provide an overview of different standards for the virtual description of an automation device, focusing on integration aspects, and conducts a detailed comparison of selected standards. The comparison considers various perspectives, such as the virtual description of automation device properties and services, the represented information about direct automation device access, and the automation device discovery mechanisms. A unified vocabulary for common concepts is presented to facilitate the comparison despite the differing terms used in the considered standards.

Since different automation device description standards may be used in different organizations and companies, tailored to different use cases, combination and integration strategies between implementations of the standards are needed. Therefore, different generic integration strategies to combine the information and data models or integrate them into each other are presented and demonstrated with selected standards, including an implementation as a proof of concept.

The contributions of this paper can be summarized as follows:

- This review provides a detailed summary of the main international standards providing automation device information models and their suitability for the required use cases in the integration task which, is currently not covered in other review papers.
- A detailed overview of each of the suitable standards is provided.
- The selected standards are compared with respect to four different use cases, and the results are presented and discussed.
- Different methods to combine the standards are presented, and one of the methods is demonstrated using two of the described standards.

The paper is structured as follows: Section II defines the concepts of an automation device and its virtual description. A literature review of the main international standards based on the described use cases is presented in Section III. The suitable standards are selected, and a brief presentation of

[1] https://www.namur.net/en/index.html
[2] https://www.zvei.org/
[3] https://www.vdi.de/en/home

each standards is given in Section IV. In Section V, the selected standards are compared in four categories: representation of a property, representation of a service, information modeling for direct automation device access and mechanism for discovery. The results of these comparisons are discussed. Section VI presents three generic methods for integrating multiple standards, along with a demonstration using two standards as an example. Finally, Section VII provides a summary of the paper's content and an outlook.

II. AUTOMATION DEVICE AND ITS VIRTUAL DESCRIPTION

A. DEFINITION OF AUTOMATION DEVICE

In the context of the paper, an automation device refers to a component designed to perform automation-related tasks, consisting of both hardware and software. There are three types of automation devices:

- actuators (e. g., motors or pumps),
- sensors (e. g., temperature or flow sensors), and
- control components (e. g., Programmable Logic Controllers or microcontrollers)

Automation devices always have an industrial communication interface to interact with other devices in the system or with higher-level applications, such as diagnostics or maintenance tools. These devices can also be combined to form aggregated automation devices, which still adhere to the definition of an automation device in this paper. Aggregated automation devices may have a modular design.

Automation devices are characterized by their automation-related services, as their configuration and parameterization can be adjusted through software interfaces. Typically, the automation-related services of a device need be to be customized for specific task it is used for, as devices are often developed for a broad range of applications.

The industrial communication interfaces allow users to access and modify the individual services and properties of the automation device, such as parameters and actual values. To access these services and properties, the user needs the definition and the corresponding description, which can be provided through a machine-processable virtual description of the device's features. In the next section of the paper, various use cases, general considerations and existing approaches for modeling these virtual descriptions are discussed.

B. VIRTUAL DESCRIPTION OF AUTOMATION DEVICES

In the context of automated integration of automation devices, the standardized virtual description of a device plays a crucial role in facilitating information exchange between heterogeneous devices. Different use cases require the exchange of specific information about the automation device between automation systems and other devices. The following list gives some examples of use cases and the associated kind of automation device information:

- assisted device type selection during engineering (digital catalogue data, datasheets)
- assisted mechanical engineering (3D models)
- assisted integration into control system engineering (description of automation device capabilites, associated invocable services, variables, automation device interface)
- integration into monitoring and maintenance systems (configuration parameters, communication interface descriptions, maintenance user interface descriptions)
- asset maintenance management (maintenance life-cycle record, digital manuals, test reports)
- product life-cycle assessment (product data record, maintenance life-cycle record, energy consumption logs)

The virtual description of a device is often called a ''digital twin'' [8]. However, this term is quite ambiguous: Other definitions include specific models — like 3D models or physical simulation models — as well as physical simulation datasets [12], [13], [14]. In contrast to these concepts, in the context of this paper, the concept of a digital representation of the device under consideration will be referred to as the *virtual description*.

Focussing on automated integration of automation devices into an automation system, only some use cases for virtual descriptions are considered in this paper:

- automation device bootstrapping, especially automation device discovery
- representation of static and dynamic automation device properties in the virtual description, with defined semantics and values
- description of automation device capabilities and additional services that are not directly associated with the asset
- description of the automation device's communication interface, including available invocable services and properties

To enable information exchange between heterogeneous automation devices, a common data and information model is required to define the virtual description and its semantics. Industry standards are typically implemented to define these models when multiple vendors are involved. The first purpose of such standards is to define an information model, sometimes also called a ''conceptual data model'' [15]. The information model describes how data and information in a specific domain are structured for the virtual description, allowing automated processing based on the recognized semantics of the structured data.

Additionally, a data model is needed to describe the binary serialization [15]. This data model can be based on generic data format specifications like XML or JSON.

Some automation device information model standards focus specific use cases and provide means to represent automation device information from relevant domains. However, some standards aim to be applicable to a wide range of use cases, related to device information exchange (e. g., OPC UA [16], Asset Administration Shell [17]). Hence, they

only specify the structure of the information model as a set of generic information model elements but leave out the domain-specific details of the semantics of single data elements. Instead, they either allow to specify domain-specific models based on the generic model elements (e. g., OPC UA companion specifications, Asset Administration Shell ''submodel templates'') or they provide means to specify the semantics of individual information elements using semantic vocabulary references.

The represented information about an automation device, according to a specific information model, can be seen as a model of the automation device itself, containing pertinent information about the device required for one or more use cases. Therefore, an information model standard, defining the model elements of each automation device's virtual description, can be seen as a ''metamodel''.[4]

III. LITERATURE REVIEW FOR RELATED INFORMATION MODELS

Standards are published from different national (e. g. DIN, ASME) and international organizations (e. g., IEC, ISO) or industry consortia and associations (e. g., W3C, NAMUR, FieldComm Group). In this paper we focus on standards from I4.0, Smart Manufacturing and IIoT. According to DIN - German Institute for Standardization, there exist more than 650 standards in the considered field, from the following organizations: IEC (256), ISO (208), ISO/IEC (173), IEEE (28), CEN (9) and W3C (7) [19]. Additionally, several literature research activities have investigated the landscape of standards [9], [20], [21], [22], [23]. The result is that the large number of national and international organizations, industry consortia and associations as well as the additional activities in publication of new standards makes it impossible to provide an exhaustive list of the existing works.

For the following comparison of automation device information modeling standards, only standards which address the use cases from Subsection II-B and which are already applied in industry, or are discussed in the actual research activities, respectively, were selected. Furthermore, we consider standards that are protocol-agnostic but allow binding to different corresponding protocols. The detailed analysis regards that a standard should provide at least (1) a representation of static automation device properties, (2) a description of the automation device capabilities and services, (3) an automation device information model and (4) a data model. Furthermore, information about (5) automation device discovery and (9) a description of automation device access interface should be considered by the standard. The results of the analysis are shown below:

[4]This nomenclature does not conform to the Object Management Group's (OMG) model hierarchy but matches the nomenclature used in many other publications. The OMG states, anyway, that ''The 'meta-' prefix should be viewed in a relative rather than absolute sense. Similarly, the numbering of meta-levels is not absolute'' [18].

A. STANDARDS, WHICH PROVIDE ONLY PARTS OF THE REQUIRED ASPECTS ARE

IEC 62424 [24]: "CAEX": A data format for exchange of ''process control engineering requests'', i. e., process plant structure and designated control equipment. It defines an XML-based, abstract, object-oriented data model for modeling attribute types, interface types, role types, object types (''system unit classes''), and a hierarchy of object instances (''internal elements''). This data model is the foundation for the Automation Markup Language (AML/IEC 62714). However, the CAEX standard by itself is strictly oriented towards process plant design and does not provide semantics to represent further information for device integration.

IEC 61987-10 [25] "Lists of properties": A definition of the structure of a property definitions including (informative) examples of using the property definitions for asset data exchange (''transaction data''). The standard describes a metamodel for creating vocabularies of distinguished properties of devices. These properties can afterwards be referenced by device descriptions to uniquely define the semantics of property values, but the standard does not provide a model for device descriptions by itself.

ISO 15926-2 [26] "Conceptual data model for computer representation of technical information about process plants": A conceptual information metamodel, which aims to be used as an ontology for integration of digital information about process plants. However, the primary focus lies on annotating a physical object, e. g., an automation device, with data, and not on the representation of that data itself.

IEC 62264-2 [27] "Definition of object models and attributes of exchanged information between manufacturing control functions and other enterprise functions": Includes an abstract conceptual metamodel for modeling information about physical assets (including capabilities and properties). It is not considered further in this paper, because it does not provide a derived information model for data representation according to the abstract conceptual model.

ISO 22745 [28] "Industrial automation systems and integration – Open technical dictionaries and their application to master data": Similar to IEC 61987, this series of standards describes a metamodel for modeling catalogues (''dictionaries'') of concepts like property definitions as well as processes and rules for creating and maintaining these catalogues and interfaces for accessing them. The application of the defined concepts for digitally representing asset data is not covered by the standard.

IEC 62453 [29] "Field Device Tool (FDT)": The standard allows manufacturers to create a software component called ''Device Type Manager'' (DTM) for their field devices. The application framework of the FDT uses the DTM to configure, parameterize, diagnose and communicate with the field device, comparable to device drivers in office IT. This allows plant operators to use a single software tool for managing field devices from different manufacturers. The focus of FDT is mainly on the application integration of the DTM, provided for field devices, and does not provide an information

model or descriptive model of the devices being integrated. All information about field device properties, capabilities and services is encapsulated in the DTM and provided as a software component.

ISO 15745 [30] "Open Systems Application Integration Framework": This standard focuses on the integration aspects of an industrial automation application system. It has been developed to provide a framework to define elements and rules that describe the integration models and application interoperability profiles based on the process, information exchange and resource views of an application. The device description in the standard focuses on the application integration of automation systems and devices through the interoperability profiles. Based on this ISO standard, the Device Description eXtensible Markup Language (DDXML) is defined, which can be used, e. g., to describe Modbus-based devices. The communication interface helps to expose device services and properties to the network. However, since the description of these services and device capabilities are not within the scope of the standard, it is not further considered in this paper.

ISO 10303 [31] "Product data representation and exchange", also known as the "Standard for the Exchange of Product model data" (STEP): Defines an abstract information model for the exchange of asset data, typically represented in the form of the formal language ''EXPRESS''. The information model and formal language are extended by application-specific schemes (application protocols), defined in separate parts of the standard. However, the standard focuses primarily on the representation of geometric data and information related to the mechanical design and manufacturing of the asset. Thus, there are currently no data models (''application protocols'') provided for the description of automation device capabilities and control interfaces.

B. STANDARDS, WHICH PROVIDE THE REQUIRED MODELING ASPECTS IN SUFFICIENT DETAIL ARE

W3C Thing Description (W3C TD): In 2020, the World Wide Web Consortium (W3C) published two new recommendations for building a so-called "Web of Things" (WoT): one document describing the abstract WoT Architecture [7] and the other one presenting a model for "WoT Thing Descriptions" [32]. Because the TD is a model for the representation of properties and services of a thing, which can be an automation device, and because the architecture also provides other functionalities, such as discovery, this standard is included in the comparison.

IEC 63278-1 [11] "Asset Administration Shell (AAS)": This standard aims to define a concept for the digital representation of asset information. Because it defines an information model including elements to model properties, capabilities and services and an interface description for interaction and discovery functionalities, this standard is considered in the comparison.

IEC 62832 [33] "Digital Factory Framework (DFF)": The standard provides a concept, which is similar to the one of the Asset Administration Shell but with a different information model and focusing on scalar property entries and semantics definition. Because of this, this standard is also included in the comparison.

IEC 62714 [34] "Automation Markup Language (AutomationML, AML)": A data format for exchange of general engineering information, integrating CAEX's generic data model (IEC 62424 [24]), COLLADA (ISO 17506 [35]), PLCopenXML [36] and additional so-called class libraries. With the defined data format, it is possible to model properties of an automation device, including its semantics and roles, and furthermore, how multiple devices are interlinked together. Therefore, this standard is considered in the comparison.

NAMUR 2658 [37] "Module Type Packages (MTP)": Virtual description format for modules of a modular automation system in process automation. Because the standard allows to model the properties and services of an automation module including interfaces and module integration, where a module is a composite of automation devices and, hence, constitutes an automation device itself, this standard is examined in this paper.

PA-DIM [38] "Process Automation - Device Information Model (PA-DIM)": Information model for OPC Unified Architecture to model different types of assets in a standardized way. The standard defines new object types for modeling, an automation device including its properties and services, and, therefore, it is included in this comparison.

IEC 62769 [39] "Field device integration (FDI)": Concept consisting of a description of an asset (field device) in terms of a declarative language (Electronic Device Description Language (EDDL IEC 61804-3) and an interface with OPC UA information model at the top level within an FDI server. Together with EDDL, the standard allows to model properties and services of an asset and, therefore, it is included in the comparison.

IV. DETAILS OF THE SELECTED STANDARDS

In this section, the standards described in Section III chosen for the evaluation are described in more detail. First, an introduction into OPC UA which is used by all standards except AAS is given. Then, a presentation of each standard is given with the following structure: a short general introduction, a description of the information model as well as the data models, and additional information about the standard, e. g., about device discovery or the ecosystem.

A. OPC UNIFIED ARCHITECTURE

OPC UA is a communication technology developed by the OPC Foundation and standardized in the IEC-62541 standard series [16]. The main goal is the exchange of information in industrial automation [40] and it is considered to be one of the most important basic technologies in

the Industrie 4.0 environment [41]. In its core, the standard defines a service-oriented communication protocol for machine-to-machine communication for client-server architectures with a full communication stack and integrated security [42]. Additionally, it defines an object-oriented modeling language which allows to model general and domain specific information models, called OPC UA Companion Specifications, which are independent to the real structure of the data base [40]. For this purpose, OPC UA defines an information metamodel based on classified typed objects (nodes) and typed relationships (egdes). The metamodel of OPC UA [43] consists of eight node classes. Each node always consists of an identifier (Node ID), which uniquely addresses the node in the OPC UA server address space, a BrowseName for the creation of hierarchical paths, a DisplayName for the use in an user interface, and references to other nodes. Optionally, among other things, access rights or a description can be defined. The node classes ObjectType, VariableType, DataType and ReferenceType are used for the definition of new types. The node classes Variable, Object and Method represent concrete nodes of these types. The class View is a special class and holds only references to a selection of nodes in the address space to allow easier navigation. For this work only the classes Variable and Method are of importance. An object of the class Variable represents a simple or complex value whereas an object of the class Method represents a callable function. Based on the well defined service layer [44], OPC UA clients are able to find servers, build up secure channels as well as application specific sessions, to find, read, write or subscribe nodes, invoke a function and observe the information contained in an OPC UA server.

For finding and automatically retrieving communication parameters from OPC UA servers at known and unknown hosts in the network, the OPC UA specification provides a multistage discovery mechanism, defined in IEC 62541-12 [45]. The discovery architecture is separated into local discovery servers (LDS), providing information about OPC UA servers on the same network host, and (optional) global discovery servers (GDS), which collect information about OPC UA servers from local discovery servers across a full plant network, including multiple segments (subnets). In addition, local discovery servers can be equipped with ''multicast extensions'' (LDS-ME) for automated discovery of other OPC UA enabled network hosts in the same network subnet without prior knowledge. Both, LDS and GDS only provide the location of each server's ''Discovery Endpoint'' to clients. Using this endpoint, a client can query the server itself for its communication parameters, using OPC UA's ''GetEndpoints'' service.

An OPC UA server typically registers at the LDS at its own network host, providing a URL of its discovery endpoint, along with an ''applicationUri'' and a ''productUri'', identifying the server, and the identification of a gateway server, if required. Additionally, the server can provide information about its capabilities (i. e., supported OPC UA information models) as a set of flags from a fixed list, e. g., ''FDI'' = support of the OPC UA FDI information model. If the LDS includes multicast extensions, the registration record is automatically shared with other LDS in the same subnet, using the multicast domain name system (mDNS) protocol. Furthermore, if the LDS has been registered with a GDS, the GDS can collect the record.

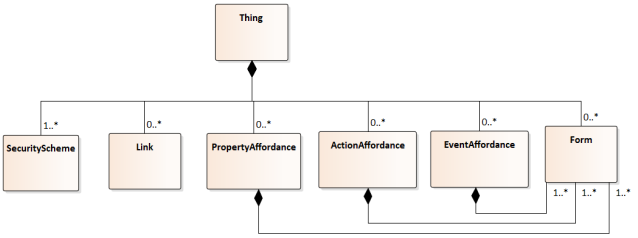

FIGURE 1. Conceptional class model definition of the thing description [46].

Based on this, a client can discover an OPC UA server by either

- directly connecting to its known discovery endpoint URL,
- retrieving the discovery endpoint from the server's LDS by using its known network hostname,
- retrieving the discovery endpoint from its local LDS, if the server is at the same host, or the LDS supports LDS-ME and the server is at the same subnet, or
- retrieving the discovery endpoint from a known GDS.

B. W3C THING DESCRIPTION

The W3C Web of Things (WoT)[5] initiative specifies a set of technology building blocks that contribute to enhance flexibility and interoperability for IoT-based applications. Thereby, the WoT Thing Description [32] (TD) is the central building block that provides a formal semantic-based asset (IoT-based application) interface description that is both machine and human processable. A TD can be used to onboard an asset with its runtime data and services into an IoT system (e. g., edge or cloud services), but it can also be used for rapid IoT application development [7]. Although still a relatively young technology, WoT is being adopted more and more, especially for cross-domain IoT application scenarios. A latest list of WoT implementations and tools can be found at the official WoT web pages.[6]

1) INFORMATION MODEL

The W3C WoT working group provides an information metamodel for the TD consisting of well-defined set of classes [32] and its relations. Fig. 1 shows the top-level metamodel of TD.

The *Thing* class provides the core vocabularies for describing some basic metadata of the asset (e. g., title, ID) and specifies what kind of interaction affordances are exposed.

[5]https://www.w3.org/WoT/
[6]https://www.w3.org/WoT/developers/

WoT introduces three interaction affordances based on properties, events, and actions:

- *Property affordances*: Represent states that an asset exposes. Typically, a property can also be read, written and observed. Typical examples of properties are sensor values, configuration parameters, status, or computation results.
- *Action affordances*: Represent either state manipulations or physical processes that can be invoked. Examples for actions are starting or stopping an engine, batch production execution or starting the execution of a continues process.
- *Event affordances*: Represent notifications and data streams that can be subscribed to. Event samples are alarms, data streams or state changes.

In general, WoT is a protocol-agnostic approach and provides a common mechanism to define how specific protocols such as MQTT, HTTP, Modbus or OPC UA can be mapped to the WoT's interaction properties-action-event abstraction. This information is mainly provided by the forms container within a TD. Based on this information, the client knows how to activate each WoT interaction abstraction through a corresponding network-facing interface for a specific protocol on which the asset relies.

2) DATA MODEL

The TD is serialized in a JSON-based representation format. Thereby, the TD information model is aligned with the syntax of W3C JSON-LD 1.1 [47] in order to enable additional semantic annotations and processing from existing (domain) concepts such as from ECLASS,[7] OPC UA, or schema.org.

3) ADDITIONAL INFORMATION

All interaction affordances can be specified with the established JSON Schema [48] concepts and can be further precised with XML Schema [49] datatypes. These prominent schema languages have rich tool support, such as validators and data model designers which helps to precisely define the data model used by the asset's interface. Additional building blocks in the context of WoT are addressing specific topics such as WoT Discovery [50], WoT API [51] to develop WoT-enabled API interfaces, WoT Thing Models [32], and WoT Privacy and Security [52].

C. ASSET ADMINISTRATION SHELL

In 2015, the concept of the Asset Administration Shell (AAS) was first introduced in [53] as a means of digitally representing and exchanging asset information among manufacturers, suppliers, and customers [54]. An asset is defined as ''physical or logical object owned by or under the custodial duties of an organization, having either a perceived or actual value to the organization'' [17]. The concept and its basic structure were published as a German standard in DIN SPEC 91345 in 2016 [55]. Based on further functional modeling and associated open source reference implementations, Plattform Industrie 4.0 published 2018 a technology-neutral information model of the AAS in UML, including various serialization formats [56]. This is constantly being further developed and is currently available in version 3.0 [17]. An international working group was initiated in November 2019 to develop an international standardization series, the first part IEC 63278-1 was published in November 2020 [57].

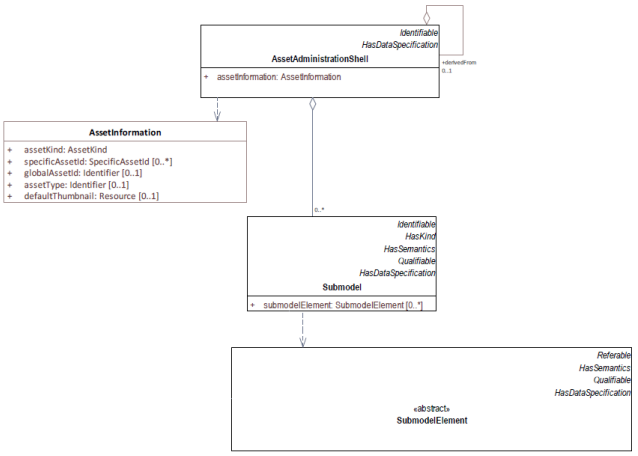

FIGURE 2. Overview of the AAS metamodel (based on [17]).

1) INFORMATION MODEL

The information model of the AAS describes the possible object types and their relationships. Fig. 2 gives an overview of the model elements.

The *AssetAdministrationShell* object serves as the main representation of the AAS. Each instance of the *AssetAdministrationShell* object represents exactly one asset and manages digital models for various aspects of the asset. Additionally, each *AssetAdministrationShell* object has an *AssetInformation* object that is used to represent the identifying information of an asset, such as its asset identification (*globalAssetId*) or whether it is an asset type or instance (*assetKind*). To represent digital models with a specific context, the *AssetAdministrationShell* has references to *Submodel* objects. The Digital Nameplate, Documentation or Product Carbon Footprint are examples of such a submodel.[8] In order to provide a specific data structure for these models, the *Submodel* object can consist of several *SubmodelElement* objects. Different subtypes of these elements can be distinguished and they are uniquely identifiable within the corresponding submodel (see Fig. 3):

- *RelationshipElement*: An object that defines a relationship between two other objects.
- *AnnotatedRelationshipElement*: An object that defines a relationship between two other objects and allows additional information using data elements.

[7] https://www.eclass.eu/

[8] https://industrialdigitaltwin.org/en/content-hub/submodels

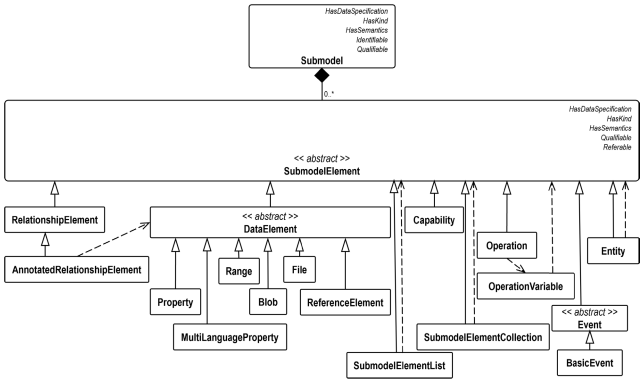

FIGURE 3. Model elements of a submodel object (based on [17]).

- *Property*: A data item that has a single value.
- *MultiLanguageProperty*: A data item which has a set of strings in different languages.
- *Range*: A data item which describes a range of values using a minimum and a maximum value.
- *Blob*: A data item which can store a binary large object.
- *File*: A data item which has an address to a file using the path and the file name including the file extension.
- *ReferenceElement*: A data element which has a logical reference to another AAS object.
- *Capability*: An object which has a reference to a capability description.
- *SubmodelElementCollection*: An object which is a collection with a fixed set of SubmodelElement objects.
- *SubmodelElementList*: An object which is a set, ordered list, bag or ordered set of SubmodelElement objects.
- *Operation*: An object which has input and output variables (arguments) and thus describes a function.
- *BasicEvent*: An object which holds a reference to an observed object.
- *Entity*: An object that describes an entity, which is either another asset managed by an own AAS or a part of this asset managed by the same AAS, e. g. a screw.

To ensure that the concept can also be applied to machine-to-machine interaction, the focus is on semantic annotation to the model elements. *Submodel* and *SubmodelElement* objects are both of type *HasSemantics*, allow for a reference to a corresponding concept description. The use of these semantic descriptions is crucial for ensuring interoperability between machines. This concept is already considered state-of-the-art in the field of product description. IEC 61360 [58] defines an information model for creating such concept descriptions. Several concept libraries have already been established, such as IEC61360-CDD[9] and ECLASS. However, these concept descriptions are not appropriate for submodel objects, as they were designed for simple data elements. Therefore, submodel templates must be created and subsequently instantiated at runtime.

[9]https://cdd.iec.ch/

2) DATA MODEL

Various concrete serialization formats, such as JSON (IETF RFC 8259), XML (W3C XML), AutomationML (IEC 62714 [34]), OPC UA (IEC 62541 [16]), and RDF (W3C RDF) have been published for implementation purposes. Additionally, an AASX package file format based on the Open Packaging Conventions (ISO/IEC 29500-2:2012) has been defined and mapped to a physical package format (i.e., a ZIP archive format).

3) ADDITIONAL INFORMATION

The AAS concept includes more than just an information and data model; it is a complete architecture that comprises an information model, interaction models, and infrastructure components, including interface definitions. In [59], both a generic and an HTTP-specific interface specification are provided. These interfaces define how to interact with a single AAS or Submodel, associated repositories, and an AASX-File Server. Additionally, interfaces and service definitions for registration and lookup are specified, enabling asset discovery based on the concept of a registry.

D. DIGITAL FACTORY FRAMEWORK

The Digital Factory Framework (DFF) is an IEC international standard (IEC 62832) for modeling production systems, published as a technical specification in 2016 and as an official standard in 2019. It comprises three parts: an introduction and overview of models and concepts [33], a detailed presentation of individual model elements [60], and rules for using these elements [61], along with mappings to relevant technologies.

The goal of the standard is to model production systems and its production system assets (PS assets), which "can be a part, a device, a machine, a software, a control system or any collection of PS asset" [33]. For this purpose, an information model is defined to model PS assets in the domain of production systems, the relationships between different PS assets, and the information flow between PS assets. The standard also specifies rules for the use of each model element (e.g. for library creation). The presented model is valid for all types of production (continuous, discrete and batch), for all branches of industry and for all phases in the life cycle of production systems. It should be possible to add, delete, modify or obtain information about a PS asset at any time. Since much preliminary work in the area of modeling has already been developed by standardization bodies (e.g., ISO or IEC), classification consortia (e.g., ECLASS) and data providers, the standard shows a way to integrate and use this work and explicitly names these three groups as stakeholders (see Fig. 4).

1) INFORMATION MODEL

In order to model concrete PS assets, the information model provides five model elements: *DigitalFactory*, *DFasset*, *DataElement*, *CollectionOfDataElements (CDEL)*, and *DFassetLink*. Fig. 5 shows the modeling of a real production

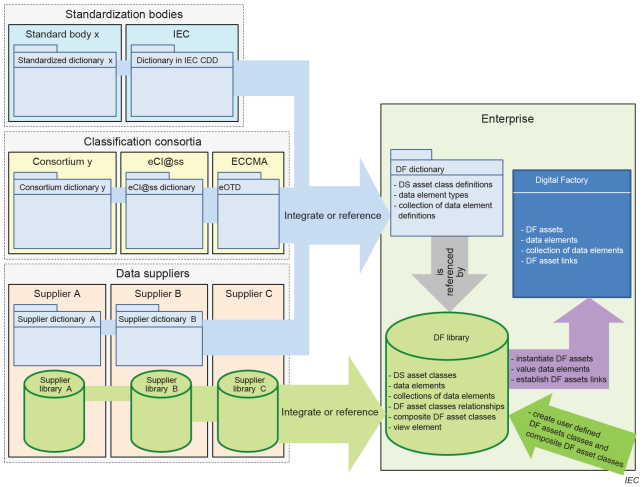

FIGURE 4. Digital factory framework overview [33].

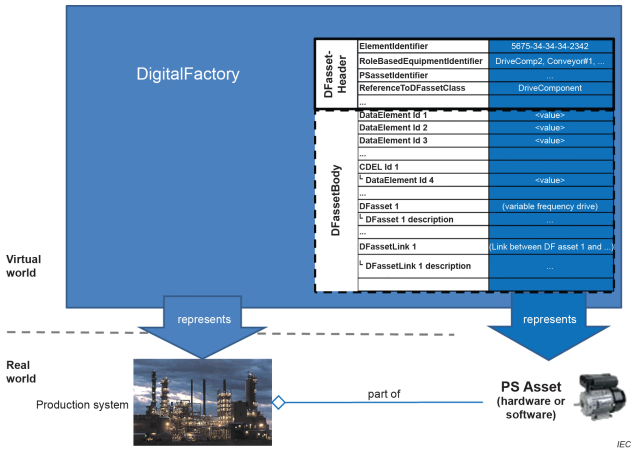

FIGURE 5. Representation of a production system [61].

system in the information world using a *DigitalFactory* object and a PS asset using a *DFasset* object. Both object types consist of a header and a body and can in turn consist of multiple *DFasset* objects. The header describes administrative information, such as the purpose of the production system or identification information. The body models information about the properties, structure and internal relationships of the production system or *DFasset*. This is done using the three other model elements: *DataElement*, *CDEL* and *DFassetLink*. *DataElement* objects can be used to model information about individual properties, including their values, such as description, name or identifier. These can be combined in lists and modeled as CDELs. The third element is the *DFassetLink* used to model relationships between two or more PS assets.

Using object-oriented modeling, *DFassetClass* and *DFassetClassAssociation* serve as types for *DFasset* and DFassetLink. The association between type and instance is represented by a reference. The types can be grouped into libraries using the *Library* objects. In addition, the *ViewElement* has been introduced to realise the possibility of filtering within a Library object or a *DigitalFactory* object.

In order to assign a semantic meaning to the object instances, elements for the definition of term dictionaries are specified. This is done analogously to the already standardized feature libraries (IEC 61360). A concept dictionary is represented in this standard by a *ConceptDictionary* object. As shown in Fig. 4, there are different concept dictionaries for different stakeholders. This has been taken into account with derived *ConceptDictionary* objects. Within a concept dictionary, the conceptual definition of these concepts can be made using the model elements *DFassetClassDefinition*, *DataElementType* and *CDELdefinition*.

2) DATA MODEL

The aim is to integrate the information model into existing exchange formats or communication standards, such as AutomationML or OPC UA. Corresponding mappings are therefore included in the appendix to [61].

3) ADDITIONAL INFORMATION

Explicitly excluded areas of application are building construction and any type of product that is processed on the production system (e.g,. input material, consumables or end products). The standard also does not include requirements or specification of a software implementation.

E. AUTOMATION MARKUP LANGUAGE

Automation Markup Language [34] (AutomationML) is a modeling language for the standardized description of a production system to enable easy integration of production plant assets into an automation system. It is a descriptive language used for engineering and exchanging data between engineering tools [62]. Its first evaluation and initiation to be made a standard was done in 2006 by engineering organisations (both industry and academics) like Daimler, KUKA, ABB, Siemens, University of Magdeburg and Karlsruhe Institute of Technology [63]. Presently, the AutomationML standard is defined under the IEC 62714 series.

1) INFORMATION MODEL

The AutomationML information model is based on the Computer Aided Engineering Exchange (CAEX) information model [24]. It uses an object-oriented approach to describe the structure of an asset and allows hierarchical composition of assets in an asset. An asset description can cover different engineering aspect like geometry, kinematics, logics and more [64]. For this purpose, AutomationML includes domain-specific information models, leveraging existing standards like COLLAborative Design Activity [35] (geometry and kinematics information) and PLCopen XML [36] (behavioral models and program logic based on IEC 61131-3). Fig. 6 shows the aggregation of these standards for the AutomationML architecture.

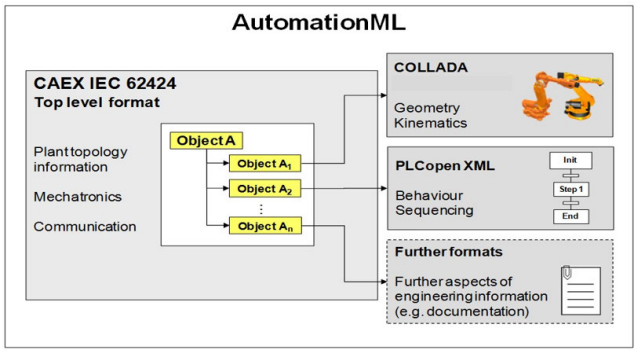

FIGURE 6. AutomationML structure [65].

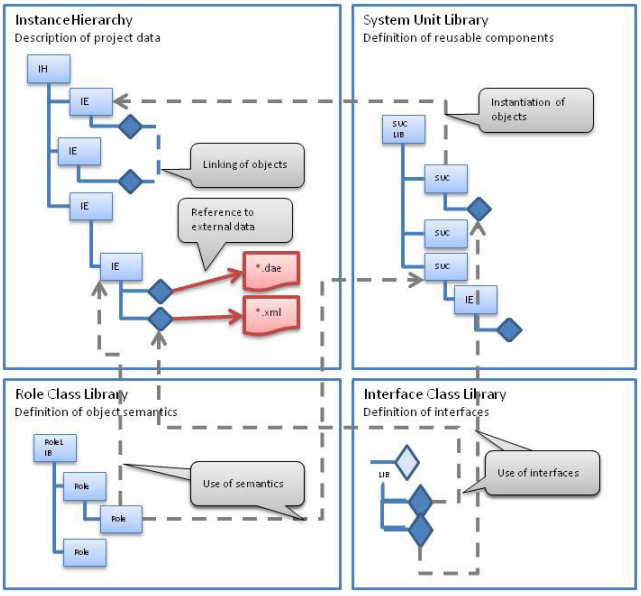

FIGURE 7. AutomationML asset topology description architecture [64].

CAEX is composed of four class libraries (see Fig. 7) called *System Unit Class Library*, *Role Class Library*, *Interface Class Library* and *Instance Hierarchy*. The asset types are defined in the System Unit Class Library, whereas the Role Class Library [66] defines the top-level functionality of an asset by specifying the semantics and roles of asset objects. The Interface Class Library defines how asset objects are interlinked together. In the Instance Hierarchy Class, instantiation of objects is modeled as ''internal elements'', based on System Unit class definitions, with references to its corresponding roles and interfaces.

2) DATA MODEL

AutomationML information is serialized in XML format, in accordance with the standards it is based on.

3) ADDITIONAL INFORMATION

Over the years, there have been different integration activities between AutomationML and other standards in industrial engineering. The integration of AutomationML and ECLASS [65] enables semantic definition of properties and structures of AutomationML objects. The combination of OPC UA and AutomationML enables communication and the exchange of data between industrial automation tools [67].

F. MODULE TYPE PACKAGE

With the VDI/VDE/NAMUR 2658 standard, a general concept regarding "Automation engineering of modular systems in the process industry" [37] is available since 2019 that was internationally standardized in IEC 63280 [68]. This multi-part standard aims at facilitating the integration of intelligent modules (i. e., modules incorporating their own internal logic controller) into existing or new process plants, to gain competitive advantages (time-to-market, time-to-repair, small batches, custom production) [37].

These modules are composed of components and supposed to contain an internal logic or intelligence implementing the module functions and executing them when respective service calls are received. The key concept is the so called *Module Type Package* (MTP) which forms a virtual description of a distinct module type. Once a module type and the associated MTP are designed and integrated, the MTP is no longer of use for the specific setup. During the subsequent operation phase data transfer occurs solely from the module to the Process Orchestration Layer (POL) and back. Here, OPC UA is the preferred communication protocol but others can be realized, too, as long as the concrete protocol mapping is specified in advance in the MTP. Via these interfaces, the internal state information of the module is accessible and direct module control is possible.

1) INFORMATION MODEL

The MTP provides all data required for an informational module integration into the POL. It contains data concerning communication abilities, module control (services) [69], notifications, description of the operator display (HMI) [68] and information for diagnosis.

In addition to the standard contents, some further module characteristics can be included and hence exposed to the POL, e. g., information about measuring points or interlocks [70]. Such an extension can be realized due to the modular structure of the MTP.

2) DATA MODEL

The MTP data is serialized in AutomationML files (see also IV-E) and provided within a PKZIP package. Access to all data is provided by the *Manifest*, an administrative file, and the central MTP element, which references the modular descriptions of different aspects. The architecture of the MTP *Manifest* is visualized in Fig. 8.

3) ADDITIONAL INFORMATION

Since MTP is used as a description delivered with the physical module, no discovery is used. Automatic selection of modules based on a list of required functionality is not part of the

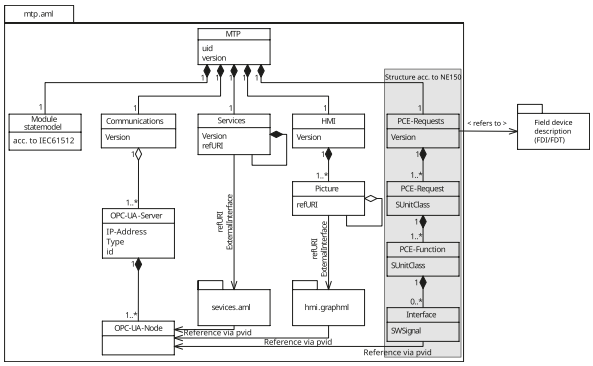

FIGURE 8. Architecture of the MTP *Manifest* (adopted from [71]).

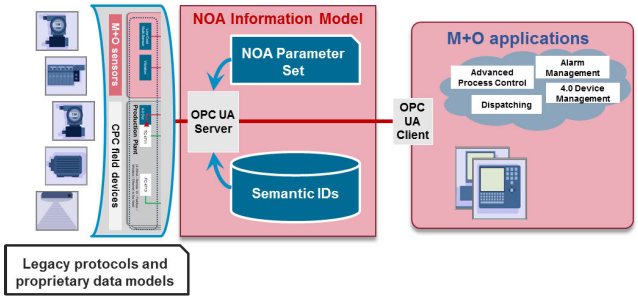

FIGURE 9. The different aspects of the NOA Information Model [72].

standard. However, there are various activities that use and further develop the standard for these purposes.

G. PROCESS AUTOMATION - DEVICE INFORMATION MODEL

The Processs Automation - Device Information Model (PA-DIM) is the OPC UA implementation of the NAMUR Open Architecture (NOA) model [38] as a Companion Specification for the description of device information. NOA is an automation integration concept mainly for the process industry that aims to facilitate implementation and operation of monitoring and optimization applications as well as value-adding services for several use cases with different types of assets (field devices, equipment and even the entire plant) [72]. The activities started in 2017 as a Joint Working Group of the OPC Foundation (OPC) and the FieldComm Group, Inc. (FCG). In 2022 the ownership has been extended to NAMUR, ODVA, PNO, VDMA, WCI-ASCI, and ZVEI.

1) INFORMATION MODEL

Since the considered assets can be very different in type (e. g., a variable speed drive versus a process analyser), the NOA Information Model is made up of different building blocks, depending on which type of asset a specific part of the information model describes. Fig. 9 shows that a device and use case specific parameter set is defined. Each parameter additionally is part of a standardized dictionary (e. g., IEC-CDD entry) so that a ''Key-Value-Pair'' is realized.

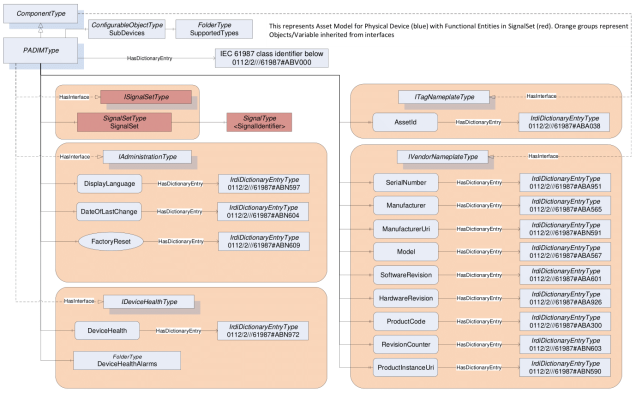

FIGURE 10. PA-DIM OPC UA Information model [38].

The root object of PA-DIM is PADIMType which maps the field device. This type implements a set of interfaces (Fig. 10):

- The *IVendorNameplateType* interface essentially contains properties for identifying the field device, such as manufacturer, model, serial number, hardware and software version, a change counter or a globally unique instance identifier (*ProductInstanceUri*). For these parameters, a *HasDictionaryEntry* reference is foreseen as a pointer to the *IrdiDictionaryEntryType*, which contains the semantic identifier of the parameter in the form of an IEC-CDD IRDI.
- The *ITagNameplateType* interface contains the AssetId, which is an identifier assigned by the user for the device (e. g., as a link to the ERP system).
- The *IAdminitstrationType* contains information about the set language and the last change date.
- The *IDeviceHealthType* interface contains information about the device status. The *DeviceHealth* variable indicates the NE107 [73] status of the device. Detailed information on any pending error states is made available via the *DeviceHealthAlarms* object.

The process values of the device are mapped together with their relevant parameters or variables via the *SignalSet* object, so that multi-variable devices can also be mapped. The information model only describes which parameters are of interest. It does not define the sampling frequency with which a parameter or variable is read out. However, it is obvious that the underlying core process control system determines the sampling.

2) DATA MODEL

The PA-DIM specification is an OPC UA Companion Specification of the OPC Foundation which defines the implementation principles in an OPC UA server based on the OPC UA Information Model.

3) ADDITIONAL INFORMATION

Some of the main scopes of PA-DIM is to ensure compatibility with standards like Field Device Integration (FDI) to

enhance the OPC UA information model based on FDI for applications and concepts steering in the direction of NOA and Industry 4.0.

H. FIELD DEVICE INTEGRATION AND ELECTRONIC DEVICE DESCRIPTION LANGUAGE

Field Device Integration (FDI), standardized in IEC 62796 [39], is a hybrid technology which consists of a description of an asset (field device) in terms of a declarative language (EDDL)) [74] and an interface with the OPC UA information model at the top level within an FDI Server (Fig. 11). The kernel of the FDI Devices Package is the EDDL defined in IEC 61804-3 [74] through IEC 61804-6 [75]. FDI-compliant devices (e. g., PROFIBUS or PROFINET devices) are to be delivered with an FDI device package, i. e., this is mandatory for the scope of delivery of the automation devices, similar to the manual, terminal assignment plans or certificates. Part of the FDI device package is the Electronic Device Description (EDD). The EDD contains information about device variables, functions, necessary communication services, logic and User Interface Descriptions (UID for HMI applications). Business rules such as if-then, switch-case and validity rules as well as the logic and constraints of the field device parameterization can be expressed. Optionally, one may include additional control elements in the form of programmed software components (e. g., Windows DLLs).

1) INFORMATION MODEL

The EDD is based on the EDDL (IEC 61804-3) and defines the elements used to describe the asset. There are different types of elements: identification element, basic construction elements and special elements. In every EDD file the identification information is the entry, where the version of the EDDL, the device type, model codes and revision details are defined. After that the information about the asset can be modeled using the different basic construction elements, like VARIABLE, METHOD or COLLECTION. It is optional to integrate semantic identifiers linking to dictionaries such as IEC-CDD or ECLASS.

2) DATA MODEL

The EDD is a textual file and provided within the FDI Package format following the Open Packaging Convention as specified by ISO/IEC 29500-2.

3) ADDITIONAL INFORMATION

The underlying communication network of field devices like HART, PROFINET, FOUNDATION Fieldbus, PROFIBUS, etc., can be accessed from the FDI Server through the FDI Communication Server [76], see Fig. 11. The communication server provides FDI specific standardized OPC UA services to facilitate the integration of field devices in automation systems. The semantics can be used by the operation-related description parts of the FDI Client host to correctly understand information that is provided by the FDI Server.

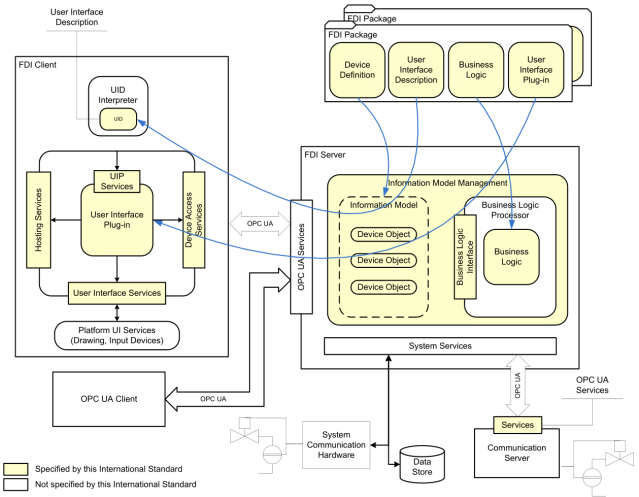

FIGURE 11. Overview of the FDI technology [39].

V. COMPARISON

In this section, the previously described standards are compared with regard to four categories: (1) Representation of a property, (2) Representation of services, (3) Information modeling for direct automation device access, and (4) Mechanism for discovery. To ensure clarity and consistency, a unified terminology with term definitions is introduced in Subsection V-A, and the included terms are mapped to the standard-specific terms. The comparison of the standards is then described, following a structured format for each subsection. First, a brief description is provided on what is being compared in the subsection. Subsequently, in the respective subordinate subsections, which have the same order as in Section IV, each standard is individually evaluated based on the corresponding evaluation criteria. The purpose of the comparison section is to assess the suitability of the information models proposed by the individual standards for representing automation devices and simplifying the integration process in an industrial setting. Finally, a discussion of the different standards, along with their advantages and disadvantages, is presented in Subsection V-F.

A. TERM DEFINITIONS

The standards examined in this section partially use different terms for the same or similar concepts of the asset information models. Furthermore, the standards sometimes assign the same terms to varying meanings, making a consistent and unambiguous comparison more difficult.

Thus, a unified terminology is introduced in this section, providing short explanations of the common purpose or concept for each of the terms. This approach aims to strengthen the common understanding and reduce potential misunderstandings.

For the concepts relevant in the context of the comparison, a direct comparison of the terminology used in the different

standards is presented in Table 1. Note that most of the terms have already been mentioned in the respective introductions of each single standard.

Additionally, different standards employ various serialization formats to exchange the virtual description. Some standards even provide a list of possible exchange data formats, from which a suitable representation of the provided data can be chosen. The available data formats are listed in Table 1 as well.

Asset

The selected standards define a virtual description of an entity or object of interest. All possible objects of interest are typically encompassed with a single term, such as 'Asset', 'Thing', 'Device' or 'Module'. The term 'Asset' has been established as the most general term for a single object of interest described with a virtual representation. Such an asset is always uniquely identifiable and can be a composition of entities or an atomic entity. It may include both software and hardware components.

In the context of this paper, according to the examined use cases, the described objects are typically automation devices (cf. Section II), including all kinds of sensors or actuators. However, an entire plant unit can also be defined as a single asset. Thus, the terms 'Asset' and 'Automation Device' are used interchangeably in this paper.

An asset in this paper is defined as ''an entity that is intended to have (or already has) a virtual representation (i. e., a description).''

Virtual Description

Such representations can be solely descriptive, providing information about the automation device, or they may additionally offer some functionality such as indirect automation device access and executable services.

The standards analyzed in this section differ in how they handle the type-instance relationship of assets. Depending on the standard, virtual representations are created only at the type level, describing all assets of the same type. In this case, they cannot contain any instance-specific information. Other standards incorporate instance-specific virtual representations, including type- and instance-specific information, with an explicit or implicit mechanism of associating the asset type with individual assets.

A virtual description in this paper is defined as ''the virtual representation of an asset or of an asset type, describing it with digital information.''

Property

In this paper, the term ''Property'' is associated with the abstract concept of the (physical or logical) characteristic itself. In contrast, we denote the data representation of a property as ''Property Description'' or ''Property Representation'' (see below).

An important characteristic of properties is the expected frequency of change. We define ''Dynamic Property'' as a property that holds a value expected to change regularly during the operation of the automation device, in contrast to a ''Static Property''. Static properties may remain consistent across all assets of the same type, allowing their representation in a virtual description of the asset type.

A property in this paper is defined as ''a particular characteristic of an asset.''

Static Property

This kind of property stores parameters or constants that remain fixed for a given device or are only changed during initial setup, maintenance, etc. Static properties include, e. g. physical characteristics of the described object, configuration parameters, or compliance information.

A static property can be provided by a property representation in a virtual description because it does not need to be updated during runtime.

A static property in this paper is defined as ''a particular characteristic of the asset that typically does not change during operation.''

Dynamic Property

Dynamic properties include measurement values or process states. The value of a dynamic property is typically dynamically transferred from the asset itself (''online'') to obtain an up-to-date value.

A dynamic property in this paper is defined as ''a particular characteristic of the asset that change regularly during operation.''

Property Description

The virtual description of an asset typically contains data elements for describing the properties of the automation device. Unless they include the actual (numeric) value of the property, we refer to them as ''Property Description''.

A property description is purely descriptive, including metadata about the property, references or addresses for obtaining the actual property value, but not the value itself. The metadata may include the property's semantics, datatype, data representation, physical unit, etc.

A property description in this paper is defined as ''a data structure intended to provide descriptive information about a property.''

Property Representation

A property representation is typically a data element of the virtual description and, unlike a property description, it holds the digital value of the considered property. It can also provide static metadata about the property, similar to a property description.

If a property representation is included in a virtual description, independent of the automation device, it is also available when the automation device itself

TABLE 1. Overview of standard specific terms and serialization formats (× means: not available).

Concept	WoT	AAS	DFF	AML	MTP	PA-DIM	FDI
Asset	Thing	Asset	PSAsset	Production System	Module	Resource	Field Device
Virtual Description	TD	AAS	DFasset	Role Class in AML-File	MTP	PADIMType	EDD
Property Representation	×	DataElement ("Property" and others)	DataElement, CDEL	Attribute	×	OPC UA Variable	EDDL VARIABLE
Property Description	Property-Affordance	×[1]	×[3]	Attribute	DataAssembly	×	EDDL VARIABLE
Asset's Service	Action-Affordance	×[1,2]	×[4]	Role Class	Service[5]	OPC UA Method	FDI Action / EDDL METHOD
Virtual Representation's Service	×	Operation	×[4]	×	×	×	FDI Action / EDDL METHOD [6]
Serialization Format	JSON / JSON-LD	JSON, RDF, XML, OPC UA, AML	AML, OPC UA	XML	AML	OPC UA	×

[1] Submodel Template "Asset Interface Description" is under development https://github.com/admin-shell-io/submodel-templates/tree/main/development/Asset%20Interface%20Description/1/0
[2] Capability descriptions are possible, however, service descriptions are under development (see Footnote 1)
[3] It is mentioned that it should be done with a CDEL, but not explicitly how
[4] A concrete element type is needed, but there is no proposal right now
[5] Function blocks can be combined to Services in the MTP
[6] Platform-dependent User Interface Plugins executables can also be included in the FDI Device Package to be executed by the FDI Client (if capable)

is not connected to a communication network. However, whether a transfer of static values to or from the automation device is intended depends on the applied standard or the specific use case.

It is possible to include property representations of dynamic properties in a virtual description of an asset if the property value is frequently updated by the automation device. Alternatively, an active software component embedded within the virtual description can fetch the value on demand from the automation device. In this case, the property representation serves as a "proxy" object for the dynamic property.

When discussing the accessibility via read or write requests or the observability of a property representation, the focus is always on its value attribute.[10]

A property representation in this paper is defined as "a data structure intended to provide the current value of a property, along with descriptive information about the property."

Asset's Service

Asset's service refers to the services provided by automation devices that interact with the physical world to perform specific production tasks. All the examined standards have a provision to describe "functions", "services", "operations", or "actions" associated with an asset.

It is crucial to differentiate between two types of functions: executable functions of the device itself (referred to as asset's service) and functions that are additional features of the device representation (known as virtual representation's service). However, clients can invoke both types of functionality through the virtual device representation.[11]

An asset's service in this paper is defined as "a procedure within the asset that can be invoked to accomplish a specific task."

Virtual Representation's Service

Virtual Representation's Service refers to an executable function defined within the virtual representation of an asset. This function is executed by an active software component associated with the virtual description. Unlike the asset's service, the virtual representation's service does not involve the execution of an automation device function itself. Instead, it interacts with automation device data such as property values or structural configuration, which can be obtained either from the asset itself

[10] In general, a property may have additional attributes besides the value attribute.

[11] In Subsection V-C, the standards' respective implementations and any uncertainties related to their implementation are explored and discussed.

(online) or from the virtual representation, depending on the standard and the specific implementation of the function.

The purpose of virtual representation's services is often to provide asset management functionality. Examples of such functionality include predictive maintenance computations, product scheduling, or AI-based analytics.

A virtual represenation's service in this paper is defined as ''a procedure within the virtual representation that can be invoked to accomplish a specific (asset management) task.''

B. REPRESENTATION OF A PROPERTY

In this section, the focus is on examining the extent to which the analyzed standards provide suitable data structures for describing properties and representing properties with their values. The objective of each standard is to provide an adequate virtual description, although the definition of ''adequate'' depends on the specific use case.

Table 2 provides an overview of common attributes found in property descriptions across at least two standards, with similar forms. The attributes are described in more detail in Table 3. Any aspects that are exclusively covered by one standard are discussed in the subsections dedicated to that particular standard.

It should be noted that some standards may allow for optional embedding of additional information models within their virtual descriptions. In such cases, the properties provided by different standards may include additional attributes. However, the tables in this section only include attributes defined in the respective standard documents for all property descriptions or property representations.

1) THE *PROPERTY* CLASS IN THE THING DESCRIPTION

In the WoT context, asset properties (''characteristics'') are considered relevant as long as they are directly accessible for other WoT participants, meaning at least one operation can be performed on that characteristic. These properties are described as ''PropertyAffordances'' in the TD.

When describing a property according to the TD metamodel, the available attributes are the same as for any other ''InteractionAffordance,'' such as events and actions, except for the ''observable'' attribute, which is specifically defined for ''PropertyAffordances,'' and for additional attributes inherited from a ''DataScheme'' class.

The data type of a property can be any JSON data type that can be declared using JSON schema. Depending on the data type declared for the property value, additional attributes can be specified. For example, if a property is of type ''number'' or ''integer,'' optional attributes like ''minimum'' and ''maximum'' with numerical values can be added to further specify the property. The TD context extension mechanism (see Chapter 7 in [32]) allows for the use of additional type systems from other standards, such as XML schema, to precisely define properties.

The primary focus of a TD property description is on communication aspects to inform clients about the possibilities of accessing the current property value. The property attributes are defined in a way that enables the description of various protocols, security schemes, and content formats using common scheme. The ''form'' object attribute, which is the only mandatory attribute of a property, encapsulates communication information required for direct access to the property value. It can include attributes for URI template variables, subprotocols, possible operation types executable on the property, or security scheme-specific access control settings. It is even possible to define an array of ''form'' attributes for a single property to allow for different settings or operations when accessing it.

A property can also have an attribute named ''const'' with a value that can be of any data type included in the TD information model. The purpose of this attribute is not explicitly mentioned in the TD specification, but it is not intended to store a constant property value since the property value is always and solely accessed via the mandatory ''href'' attribute of the ''form'' attribute. The ''href'' attribute contains the target IRI of the property value. If a constant property value needs to be described in a TD, an IRI pointing to the value must still be provided because there is no provision to include the actual value in the TD. Therefore, TD supports property descriptions but not property representations.

Furthermore, the TD information model includes attributes that allow for complex data types, which not only allow for subordinate property objects but also for enumerations listing all valid property values. Object definition can be further specified by including their names in a map container. Additionally, the generic WoT approach allows for context extensions to enrich a TD with additional semantics, such as protocol bindings or security schemes. External resources and Thing-related data can be referenced via links when available as web documents.

In the example property serialization provided, the prefixes ''opc'' (indicating the OPC UA specific data type) and ''om'' (adding unit information) qualify the results of the context extensions.

```
"properties": {
  "T1": {
    "title": "T1",
    "description": "Current Temperature",
    "type": "number",
    "unit": "om:degree_Celsius",
    "readOnly": true,
    "observable": true,
    "opc:dataType": "Double",
    "forms": [
      {
        "href":"opc.tcp://acplt.org:9409/
        DvOpcUaServer/ns=2;s=0:T27/PV",
```

TABLE 2. Common attributes of properties - descriptions to the attributes given in Table 3.

Property attribute	WoT	AAS	DFF	AML	MTP	PA-DIM	FDI
Identification name/ title	✓	✓	✓	✓	✓	✓	✓
Data type	✓	✓	✓	✓	✓	✓	✓
Description	✓	✓	✓	✓	✓	✓	✓
Value	×	✓	✓	✓	✓	✓	×
Communication information	✓	×	×	✓[1]	✓[2]	✓	×[3]
Default value	×	×	×	✓	✓	×[4]	✓
Unit	✓	×	✓	✓	✓	×	✓
Semantic reference	✓	✓	✓	✓	×[5]	✓[6]	×[7]
Attributes of complex data type	✓	×	×	×	×	✓	✓[8]
Access level	✓	×[9]	×	×	×	✓	✓
Additional attributes	✓	✓	×	×	×	×	×
Category	×	✓	✓	×	×	×	✓

[1] Reference to CommunicationRoleClassLib
[2] Reference to information element
[3] Not provided unless mapping to OPC UA
[4] Added from the Type Definition if necessary
[5] Semantically defined for parent objects
[6] Via HasTypeDefinition reference
[7] Via seperate SEMANTIC_MAP element
[8] Data Type can be enumerated
[9] Specified elsewhere (in AccessPermissionRule)

```
        "contentType": "application/
        x.opcua-binary",
        "op": [
          "readproperty",
          "observeproperty"
        ]
      }
    ]
  }
}
```

2) THE *DataElement* CLASS IN THE AAS

The AAS information model utilizes the abstract *DataElement* class to define value-containing elements such as variables, parameters, or constants, which serve as representations of properties within an AAS Submodel. The *DataElement* class is always at the bottom of the hierarchical model structure.

Specific subclasses of *DataElement* are chosen based on the type of value they should contain. For example, the *Property* class is used to describe basic property representation, such as process values or a design parameters. Other classes are designed to represent references, property values in multiple languages, media (MIME [77]) type file contents, or define a range with minimum and maximum values (see Subsection IV-C).

The *Property* data element includes at least a data type attribute and optionally a data value or a value identifier. If both a value identifier, which optionally points to a semantic definition of the value, and a data value are present, they must coincide, meaning they should refer to the same value. While the data element classes themselves define only a few attributes, additional options to describe property details are introduced through class inheritance. Properties can be further specified through attributes such as description, category (e. g., parameter, variable, or constant), idshort, kind classification (i. e., template or instance), or constraint. The semantics of the property can be defined by referencing either an internal data specification (template model element or object of the *ConceptDescription* class) or an external one, such as ECLASS or IEC61360-CDD. The semantic definition often results in further characterization of the property with additional attributes, which can be defined once in the template for all property instances or assigned individually to each instance.

In the example property JSON serialization, the *semanticId* refers to a *ConceptDescription* that, in turn, references the data specification template for the property, *DataSpecificationIEC61360*, via a global IRI. This template allows for the addition of auxiliary attributes in the *ConceptDescription* object, such as *unit* (e. g., degree_Celsius), *dataType* (e. g., REAL_MEASURE) or *preferredName* (e. g., [{"language": ‘‘EN’’, ‘‘text": "Current Temperature TU20 (tank B2) T27"}, {"language": "DE’’, "text": "Aktuelle Temperatur TU20 (Tank B2) T27"}]).

By leaving the *value* attribute unset, a *Property* element can also serve as a property description. However, this use case is

TABLE 3. Common attributes of properties: descriptions.

Property attribute	Description
Identification name/ title	Label/title/name that makes a property identifiable in the namespace with multilanguage support for WoT, MTP; sometimes additional (unique) id or display name provided
Data type	Data type of the property value, depending on the serialization format; these always support simple types such as boolean, number, string; some standards allow also complex types such as object or array
Description	Human-readable explanation or comment of the property (with multilanguage support for WoT, AAS, MTP, FDI)
Value	(Current) value of the property (only for property representations, multilanguage support for AAS)
Communication information	Attribute(s) containing sufficient information to gain property access (online and only if access is allowed)
Default value	Default setting for the value attribute
Unit	Unit associated with the property included as an attribute
Semantic reference	Reference to type definitions/ semantic tag
Attributes of complex data type	Enable nested properties
Access level	Determines at least if a property is writable or readable (sometimes also observable; or other permissions can be given, e. g., delete a property)
Additional attributes	Properties can also be extended with additional attributes through semantic context extensions
Category	Indicates whether the property value is static (i. e., parameter, constant) or dynamic (variable)

not specifically addressed in the specification document [17], and there are no standardized attributes for describing access to the actual value when it is available elsewhere. Currently, efforts are underway[12] to develop a submodel template called "Asset Interface Description", which will allow the description of access to asset properties in the future.

```
"submodelElements": [{
  "value": "25",
  "semanticId": {
    "keys": [
      {
        "type": "ConceptDescription",
        "local": true,
        "value": "http://acplt.org/cd/
        temperature",
        "idType": "IRI"
      }
    ]
  },
  "constraints": [],
  "idShort": "T1",
  "category": "VARIABLE",
  "modelType": {
    "name": "Property"
  },
  "valueType": {
    "dataObjectType": {
      "name": "integer"
    }
  },
  "kind": "Instance"
}]
```

[12]https://industrialdigitaltwin.org/en/content-hub/submodels

3) THE *DataElement* CLASS in the DFF

In the DFF information model, the *DataElement* is used to "represent a characteristic of a PS asset or a role" [33]. It consists an identifier, a reference to a data element type, and a value. The identifier is locally unique within its parent object, which can be either the body part of a DFasset object or a CDEL. The referenced *DataElementType* provides meta-information about the data element, including its category, data type (in string representation), definition of permissible values, and allowable physical unit. DFF does not define its own data types (apart from the string representation), so the specific data types are described in semantic descriptions. The *DataElementType* object contains a reference to a *ConceptDictionaryEntry* object, where the semantics of the data element are detailed. The value of the DataElement object represents the actual value and must comply with the data type specified in the referenced DataElementType and the corresponding semantic description (*ConceptDictionaryEntry*). Additionally, a *DataElement* object can include a physical unit, a timestamp indicating when the value was set, and information about the value quality (Good, Bad, Uncertain). If a physical unit is provided, it should be one of the units listed in the referenced

DataElementType object. The *DFassetLink* can be used to model relationships between PSassets.

Similar to the AAS information model, the DFF *DataElement* does not provide attributes to store information about accessing values from external data sources. While the specification suggests representing an interface description with a CDEL, it does not offer specific instructions on how to achieve this. Therefore, a *DataElement* serves solely as a property representation and not as a comprehensive property description.

At the time of writing, there were no examples of serializing a *DataElement* provided in the specification. The specification only presents mappings to OPC UA and AutomationML.

4) THE *InternalElement ATTRIBUTE* OF THE *InstanceHierarchy* CLASS IN AutomationML

AutomationML, based on the CAEX (IEC 62424) standard, utilizes the InstanceHierarchy class as the top level for modeling automation components' properties in a hierarchical structure of objects known as *InternalElements*. These *InternalElements* consist of *Attribute* elements, which are employed to represent object properties.

Attributes in AutomationML can encompass various elements such as *AttributeDataType*, unit, description, and value. The *AttributeDataType* supports XML schema data types, providing flexibility in representing different types of data. This allows AutomationML to support both property descriptions and property representations. Additionally, the *Attribute* element can include a nested attribute called RefSemantic, which stores the value of the property ECLASS ID. This reference to the ECLASS ID helps to define the semantics of the property.

InternalElements in AutomationML can also make use of interfaces from the Interface class library to establish connections between component instances. These interfaces include port connector, communication capabilities, *COLLADAInterface*, and *PLCopenXMLInterface*. The functionality and semantic meaning of an *InternalElement* can be imported from the role class library by utilizing the *RefRoleClassPath* attribute, which references one or more roles from the library. AutomationML provides a comprehensive library of machine roles within the automation environment. However, users also have the flexibility to define their own custom role class that best suits their specific model.

A typical example of the *InternalElement* used to represent an asset property is given below.

```
<InternalElement Name="MyRobot_axis"
  ID="c154d16f-1ffa-4126-b5d3f0e02c5d">
  <Attribute Name="Number_of_Axis"
  AttributeDataType="xs:integer">
    <Description>
      quantitative indication of
      the number robot axis.
    </Description>
    <Value>3</Value>
    <RefSemantic
      CorrespondingAttributePath=
      "ECLASS:0173-1#02-AAK525#003"/>
  </Attribute>
</InternalElement>
```

5) THE *DataAssembly* CLASS IN MTP

In the NAMUR standard, a Manifest is required, which is an AutomationML file that describes the contents of the respective Module Type Package (MTP). Besides referencing MTP elements such as *operator display description* or *service description*, the *Manifest* directly includes descriptions of the module communication and the relevant module components, such as sensors and actors, referred to as *instances*. The *instances* are gathered in a list, where each *instance* with communication capabilities is represented by an object of the abstract *DataAssembly* class or a subclass thereof. An *instance* is provided with a name attribute, a reference to the corresponding class that determines its syntax and semantics, and an identifier that allows it to be referenced by other MTP elements. The *instance* also comprises *Attribute* elements, each serving as a potential interface variable that links to a data item of a data source through its *value* attribute. If the attribute does not link to a data source variable, the *value* attribute contains a static parameter value. Thus, the *Attribute* elements serves as data structures for both property descriptions and property representations.

For instances that are objects of a subclass of *DataAssembly*, all mandatory and optional properties (i. e., *Attribute* elements) are described in [70] and defined as standardized *interfaces* in the AutomationML SystemUnitClassLibrary *MTPDataObjectSUCLib*. These interfaces include variables for minimum and maximum values, control inputs and outputs, as well as variables used for interlocks and other safety mechanisms. The interface variables do not contain direct communication information but are connected to an *ExternalInterface* object through a reference. The *ExternalInterface* object specifies the necessary details related to read/write access and identification, such as the *NodeID* in the case of an OPC UA server, where the server endpoint URL is specified in the parent element.

```
<Attribute Name="T1" Unit=""
  AttributeDataType="xs:IDREF">
    <Description>
        Current Temperature
    </Description>
    <DefaultValue />
    <Value>refIDT1</Value>
    <RefSemantic
      CorrespondingAttributePath=
        "Connectable" />
    <RefSemantic
```

```
      CorrespondingAttributePath=
        "Standard" />
</Attribute>
```

6) THE *VARIABLE* IN THE PA-DIM

According to the OPC UA Address Space Model, the OPC Unified Architecture uses *Variable* nodes to represent object components that can hold any kind of values. The *Variable* nodes can be further categorized into two subtypes: *Properties* and *DataVariables*.

Properties are the bottom elements of any hierarchy and represent simple values. *DataVariables*, on the other hand, can be composed of complex data and may reference other Variables through *HasProperty* or *HasComponent* references. The range of possible attributes may vary between the subtypes. Additionally, there are several inherited attributes from the *Base NodeClass*, many of which address security aspects and can be used for access control purposes. The OPC UA specification allows for detailed access control at the attribute level, including Read, Write, Query, and Subscription Services. However, configurations at the attribute level may have interdependencies and should be carefully managed. In many applications, it is primarily the value attribute that changes frequently during runtime.

The PA-DIM OPC UA Companion Specification defines optional and mandatory property representations to be implemented as *Variable* nodes. These property representations typically represent static properties. However, OPC UA allows *Variable* nodes to describe properties regardless of their nature. When representing a physical process automation device virtually, *Variable* nodes, along with addition *Method* and *Object* nodes, are used. Property instances can represent process variables, parameter settings, device states, or general device information about the manufacturer or series. The semantic definition of these device characteristics are linked via *HasDictionaryEntry* references to a *DictionaryEntryType* object node [38].

Access to a property can be achieved by using the *nodeId* or the browse path, as long as it complies with any access restrictions that may be in place.

7) FDI PROPERTIES OR EDDL *VARIABLEs*

The FDI standard allows for the provision of automation device virtual descriptions written in the platform and technology independent EDDL. These descriptions can be mapped to the FDI OPC UA information model of the FDI server. To ensure a clear mapping from EDDL to OPC UA, mapping rules are defined that specify the types for all EDDL elements [78]. The additional data required for creating instances can be obtained through scanning or may be available offline.

To find the FDI equivalent of property descriptions or representations, the EDDL standard [74] must be consulted because the FDI Device Package contains property information exclusively in EDDL. In EDDL, simple device properties are described using *VARIABLEs*. Complex properties contain *VARIABLEs* logically grouped in *RECORDs* or *VALUE ARRAYs*, depending on whether the members share the same data type or describe a common characteristic. These *VARIABLES*, *RECORDs*, and *VALUE ARRAYs* are also referred to as *Parameters* of the described automation devices, regardless of whether they contain static or dynamic values.

One unique feature of EDDL or FDI standards are the post- and pre-actions VARIABLE attributes. These attributes contain references to actions (EDDL METHODS) that are triggered when a designated interaction with the property occurs. For example, these actions can be triggered before or after performing read requests or editing a value. Other EDDL attributes used for describing properties include the private attribute, which makes a property non-browsable, and the validity attribute, which allows for conditional or temporal deactivation of a property.

The communication endpoint information is not available until the EDDL device data has been mapped to the FDI information model. The communication information included in an EDDL document only applies to the internal communication between the FDI Server and the automation device. The supported attributes for this communication generally depend on the protocol being used, but most EDDL-compatible protocols support the majority of property attributes (e.g., Fieldbus, PROFIBUS/PN, HART).

When writing an FDI property value, it is unlikely that the property value of the automation device and the property value stored in the OPC UA information model change simultaneously. The exact procedure depends on the type of value (determined by the VARIABLE's CLASS attribute), whether it is static (parameter/constant) or dynamic (variable). In the case of static values, the property value is first modified in the OPC UA information model and is only transferred to the automation device when a method defined for that purpose is called. For dynamic values, the property value is directly written to the automation device and is not copied to the OPC UA information model until a read request for that property is received. This behavior can be further controlled by the pre- and post-actions defined in the EDDL device description.

The description below shows a snippet of a PROFIBUS device variable description in EDD and the information about where to access it in the automation device.

```
VARIABLE temperatureValue
  {LABEL "Temperature";
   TYPE FLOAT;}
VARIABLE temperatureValueStatus
  {TYPE ENUMERATED
    {GOOD = 80;
     BAD = 85;}}
ARRAY OF VARIABLE processValue
  {ELEMENTS
    {temperatureValue;
     temperatureValueStatus;}}
```

```
COMMAND read_processValue
  {SLOT 1;
   INDEX 26;
   OPERATION READ;
   RESPONSE_CODES DPV1_rsp_codes;
   TRANSACTION
     {REQUEST{}
      REPLY
        {temperatureValue,
         temperatureValueStatus}}}
```

C. REPRESENTATION OF SERVICES

In this section, it is considered to which extend the functionalities of an asset, i. e., the services it provides, can be described in its virtual representation. Furthermore, the virtual representation itself might be supplemented by additional services, with some standards even including function implementations as a part of their exchangeable virtual representation (esp. FDI and up to a certain point also MTP). This section gives an overview of the possibilities that each standard provides for modeling or embedding services in the virtual description.

1) THE *ACTIONS* CLASS IN THE THING DESCRIPTION

In the WoT framework, services are a subclass of *InteractionAffordances* called *ActionAffordances*. Such *ActionAffordances* have additional attributes to specify those characteristics of the service that are immediately relevant for its invocation. Among these are, e. g., input and output data schemes, an attribute which indicates whether the service involves a change of an internal resource state, or the attribute *idempotent* which provides information on whether the output of the function stays constant throughout multiple calls. It is worth noting that the focus of the WoT TD service descriptions is not only on providing general description elements to fully characterize a device but also on invoking the actual device functionality. A TD service description shall suffice to enable clients to call the listed service. The actual service implementation is done outside of the TD where either the handlers for each action can be defined in the WoT Servient as part of the behavior implementation or, if an existing device comprising a server needs to be integrated in the WoT, the handlers are already implemented in the device hardware or software. Since TD aims to interconnect with the thing the purpose is on Asset's services. Therefore, virtual representation's services are not supported so far.

2) THE *OPERATION* ELEMENT IN THE AAS

The AAS specification includes services as *SubmodelElements*, referring to them as *Operations*. These elements possess description-related attributes that augment those common to all *SubmodelElements* (already discussed in the property section, e.,g., semantics determination and identification) with attributes for input and output parameters. Notably, these operation variables must also be *SubmodelElements*. In addition, the AAS can incorporate *Capabilities*, which are also *SubmodelElements*. Such *Capabilities* aim to describe functionalities an asset can perform, for instance, to assist in automation device selection processes [79].

For both *Operations* and *Capabilities*, the specifications do not provide means for accessing the actual services (of the AAS or the automation device), nor do they stipulate where any specific *Operations* are implemented. The *Submodel Interface* in [59] defines API operations to invoke *Operations* as services of the AAS (*InvokeOperationSync*; *InvokeOperationsAsync*). These *Operations* are presumed to possess a state and a result according to the interface definitions. However, further details about the AAS *Operations* are not provided in the referenced document. As previously mentioned in Section V-B2, there is an ongoing effort to develop a submodel template termed "Asset Interface Description" that will facilitate the description of accessing asset services in the future.

3) FUNCTIONS IN DFF

As previously noted in Subsection V-B3, DFF data types are dictated by semantic descriptions. This necessitates a concrete element type for the representation of a service. However, no proposal has been put forth as to how this might be realized, thus precluding a more detailed analysis at this stage.

4) THE *ROLE* CLASS IN CAEX OF AutomationML

Though the description of the functionality of automation devices in a plant in AutomationML is abstract, the Role class library helps with describing device services. It provides a semantic definition of a component's role and its functionality in an automation system. The CAEX provides various kinds of *role class libraries* which are referenced in the SupportedRoleClass attribute of the InternalElement in Instance Hierarchy class.

For example, the role class library for discrete manufacturing industry (*AutomationMLDMIRoleclassLib*), as defined in [80], includes the role classes 'Robot' and 'Transport' that can be used to model the services of robots and any transport-enabled device. Instances of these roles are referenced with respect to their parent role class, called *AutomationMLDMIRoleClassLib/DiscManufacturingEquipment*, by using *RefBaseClassPath* in the SupportedRoleClass attributes.

Combining the skill(s) (process role class) of a machine (resource role class) to realize a production output (product role class) allows to describe processes in *AutomationMLBaseRoleClassLib*, according to the Product-Process-Resources (*PPR*) concept. These three roles (product role class, process role class and resource role class) are linked by the *PPRConnector* from the *interface class library*.

A process in AutomationML can be any service (e. g., drilling) offered and expected to be performed by a machine. This action is attached to a role class, optionally with some set of attributes and interfaces attached to it.

The modeling of user defined role classes in AutomationML is also possible, but limited by the capability of importer tools to correctly interpret the information.

5) THE *SERVICE* CLASS IN THE MTP

All the services that an MTP module provides and that are modeled in the respective MTP have an equally designed service state machine. The *Services* are often macro functions, consisting of *ServiceProcedures* that can be part of more than one service. These procedures are the only available function blocks used to implement a service. Several feedback variables are provided to keep track of the procedure executions of a service as well as their health status. The current service states are retrieved as return values.

For modeling services, MTP uses basically the same means as for enabling the modeling of properties. Firstly, a library must be provided which contains numerous classes, e. g., for the representation of configuration parameters, report values or incoming and outgoing process values. Each service and its procedures are mainly characterized by its variables and parameters, enabling service parametrization, service control and service-operator interactions. Besides, classes exist for modeling service dependencies (examples are mutually exclusive services, interacting services or service sequences). Secondly, the link mechanism is applied to connect service variables with instance variables and finally with data source items that are able to communicate with the module (e. g., OPC UA nodes) [69].

The MTP itself is merely a descriptive data structure without executable elements and thus not capable to provide services on its own (virtual representation services).

6) THE *METHOD NodeClass* IN PA-DIM

The three different kinds of services that PA-DIM provides (i. e., *FactoryReset*, *ZeroPointAdjustment* and *AutoAdjustPositioner*) are represented by *Method Nodes* in OPC UA. These three executable functions are generic automation device services, which each process automation device of PA-DIM type needs to implement. The *Method NodeClass* is, as any other node class, also derived from the *Base NodeClass*, the attributes of which are consequently inherited. Moreover, an OPC UA function is specified by its input arguments and return values (''output arguments'', included as function properties) and might contain references to events triggered by the function [43]. Additionally, PA-DIM states that each function shall be executable and at least callable by one user [38]. The actual function invocation is done via a standard OPC UA service, the *call* service, whose most relevant arguments comprise the identifier of the function to call and its input parameters [44]. In general, OPC UA methods have no state that is visible to the client, i. e., when called the methods run until completion.

7) FDI ACTIONS OR EDDL *METHODs*

In the FDI Device Package, executable services are represented by *METHOD* implementations in EDDL. These *METHODs* can be used for both, triggering services of the asset (Asset Services) and defining additional functionality, e. g., methods performing calculations of characteristic values (virtual representation services).

METHODs can make use of a large set of functions from the EDDL ''built-in library'', e. g., for communication with the device. The same way that EDD *VARIABLEs* are mapped to OPC UA *Variables*, EDD *METHODs* are mapped to OPC UA *Objects* of *ActionType*, an OPC UA *ObjectType* which has particularly been defined for FDI [81]. In contrast to the OPC UA *Methods* as they are, e. g., used in the PA-DIM, the *Action* objects are not stateless. Instead, their execution state is always defined based on a state machine for each action (with possible states: Start, Created, Running, TimeDelay, WaitingForFeedback, Aborting, TimeDelayA, WaitingForFeedbackA, Aborted and Completed [82]). Such an *Action* object can be called via the OPC UA method *InvokeAction* that is, for instance, triggered when a user presses the respective button provided in an *EditContext* or an invocation might result from a property read action (e. g., as a post- or pre-read action).[13] The FDI Server then returns a transient variable that indicates the current execution state for the *Action*. The transient variable can be subscribed by the FDI Client. By making use of this variable, the FDI Server is able to notify the FDI Client in the case of changes and in particular if user interaction is mandatory before proceeding with the service execution. Possible requests for user interactions could involve, e. g., acknowledgments or value inputs [82].[14] The internal service logic may use EDD *VARIABLES* (with online and/or offline values) and can be implemented by all means that the EDDL provides. A service is described by a list of attributes that are similar to those used for the properties, e. g., determining a label, a description or a class [74], [83].

This abstract about FDI services tries to give a rough idea of some of the most relevant basics, but the FDI standard is rather complex and includes plenty of further specific features. They are described in the IEC 62769 (FDI) and IEC 61804 (EDDL) series and hence, for any details and example descriptions, the reader is referred to these series.

D. INFORMATION MODELING FOR A DIRECT AUTOMATION DEVICE ACCESS

Although all standards unanimously suggest a client-server architecture for accessing a virtual description of an automation device, the standards show considerable differences in the realization of the communication architecture.

On the one hand, there are standardized datasets, serialized in an appropriate data format, in order to provide asset capabilities together with information about the respective asset access points to the user as a network client. In that case, the client is allowed to directly access or control the asset

[13]The *InvokeAction* method is only one of numerous OPC UA *Methods* that the FDI provides [78].

[14]The client-user interaction is an important part of the FDI specification and even before starting a certain service, the user needs to navigate to a automation device, navigate through a menu and select a functional group.

within the limits of the provided and accessible properties and services. Some standards establish the communication without an additional description data element, e. g., using the OPC UA Information Model and providing all device data via OPC UA nodes. The client only needs to access the OPC UA server in order to obtain automation device data, or to call the methods offered by the server.

On the other hand, there are standards that only provide information about an automation device but do not serve as a means to gain direct device access. The virtual description rather serves as a resource itself. Consequently, a client-server communication can be established between such a virtual description and a client, accessing the virtual description via a standardized interface.

An overview of the different client-server connections, required data integrations and architecture elements of the standards is illustrated in Fig. 12.[15]

For a generic approach to directly access the asset, it must be defined **which** services and properties are available and **how** these are accessed. Therefore, the exchangeable data set must either directly contain this information. Table 4 shows where these information can be found in the different standards and how they are presented. This exemplary description is given for properties describing simple measurement values or state variables.

1) JSON PROPERTIES FOR DIRECT ACCESS IN TD

The TD defines a standardized dataset for describing the accessible device properties and services. It is represented in the form of web forms in the context of the respective property with values for operation type (e. g., readproperty or writeproperty), target IRI and content type. Depending on the protocol and server, there might be further specifications necessary, which are defined in protocol bindings. The client integrates the TD, reads out the required information and then establishes communication with the indicated server and gets direct access to the provided information.

2) AAS SERVES AS A RESOURCE ITSELF IN AAS

The question of whether the AAS should provide direct asset access or grant control access to clients is currently under discussion and has not yet been conclusively resolved. All asset information is stored in the AAS, and the defined HTTP API allows retrieval of this data. However, there is currently no standardized method for storing asset access information within the AAS.

A task force is currently working on defining a submodel template, termed ''Asset Interface Description'', which aims to provide the necessary information for gaining access to the asset (see also Section V-B2). Initial considerations suggest that the web forms of the WoT Thing Description should serve as a basis, with this information subsequently being integrated into the submodel. Therefore, in the future, a client could connect to the AAS, extracting information about direct asset access from the corresponding submodel.

3) DELs FOR DIRECT ACCESS IN DFF

Similar to the AAS, the DFF specification does not define how a client should directly access the asset. Within DFF, it is only possible to provide information regarding the interfaces of a PS asset to other PS assets. This information, according to [61], ''should be represented by CDELs in order to support the description and evaluation of connections between PS assets''. The usage of *DFassetLinks* along with their *DFassetLinkEndPoint* could be a viable option. Based on this, various rules can be applied to check whether two PS assets are compatible.

For providing information about direct asset access, the only option is currently to define non-standardized properties (DEL) that contain the necessary information. Since DFF is serialized in OPC UA, this information can be read by an OPC UA client, thereby establishing direct asset access. However, it is important to note that this approach is proprietary and not standardized.

4) OPC UA INFORMATION MODEL FOR DIRECT ACCESS IN AutomationML

AutomationML, being a data exchange format, only answers the question of ''what'' can be done by means of description and not ''how'', which is achieved by implementation. Presently, AML and OPC UA are working together to provide a companion specification [84][16] based on the AML information model that would reflect production plant elements and their properties on an OPC UA server at the operational level.

The implementation of the AML information model mapping to OPC UA model will allow to instantiate an OPC UA server that reflects the production plant. This instantiation can provide direct access to the devices of the plant by browsing through the OPC UA nodes.

5) XML ELEMENTS FOR DIRECT ACCESS IN MTP

Similar to the TD, MTP defines a standardized dataset (called ''MTP''), describing the connection to an OPC UA server. It uses XML elements identified by Reference IDs, linked (via ID-Link) to communication endpoints. Each XML element describes the information needed for direct asset access. Therefore, the communication endpoint with all necessary values for ID, Name, Access (read/write/both), Namespace and server endpoint (URI schema + IP address + port) is contained. During the integration of the MTP file into the POL, the OPC UA server endpoint must be manually added. After that, the POL (as an OPC UA client) can connect to the corresponding server and get the provided information using the OPC UA information model.

[15]The depicted architectures are not claimed to be complete, as other architectures may be realized for the standards as well. The depicted ones, however, are considered to reflect the central ideas for an architecture proposed by the particular standards.

[16]https://github.com/AutomationML/AML-UA-XSLT

FIGURE 12. Overview of system architectures supported by the compared standards.

6) OPC UA INFORMATION MODEL FOR DEVICE ACCESS IN PA-DIM

Being an OPC UA companion specification, PA-DIM provides the device information already in the namespace of OPC UA by using its information model and direct asset access. Once the client obtains the endpoint of the OPC UA server, it builds up a connection and can get all the information directly using the standard methods of OPC UA. The information is displayed as OPC UA variables identified by a name, such that the requested NodeID can be obtained through browsing.

7) EDDL VARIABLES FOR DIRECT ACCESS IN FDI

FDI uses EDDL variables identified by name and the OPC UA mechanism. Therefore, all EDDL variables are mapped to OPC UA variables. The requested NodeIDs can be obtained through browsing (for public variables) the OPC UA server or obtaining the User Interface Plugin information from the FDI server via UID type node reading (for private variables). After that, the client can get the asset information using the standard methods of OPC UA.

E. MECHANISMS FOR DISCOVERY

Due to the increasing amount of information about assets, it is important to manage this data and enable search mechanisms to find specific data content that may be related to one or more specific assets. In the following subsections the different approaches of the standards are described. Only the Web of Things and the Asset Administration Shell standards define a concrete discovery mechanism which are briefly introduced. All other standards do not explicitly mention how discovery should work. However, since they all are either based on or use the OPC UA protocol, they can use the discovery mechanisms of OPC UA for discovery of known and unknown servers on the network and retrieving their communication parameters via Local Discovery Servers and Global Discovery Servers (cf. Subsection IV-A)

1) W3C WoT DISCOVERY

The W3C Web of Things has a technology building block for discovery that is specified in the discovery document [50]. There are different discovery approaches provided how to discover specific Things or TD, respectively:

TABLE 4. Overview of data provided by the standards for realizing a direct control access to a device property.

Standard	Where	How
TD	JSON properties (properties) identified by name	Web forms found in the context of the respective property with values for operation type (e. g., readproperty/writeproperty), target IRI (URI schema + IP address + port + NodeID), content Type, ...[1]
AAS	JSON properties (submodelElements) identified by referable name (i. e., idShort)	Original AAS metamodel has no variable for specifying a server endpoint of a device;[2] For a single AAS property, idShort-Path can be retrieved from standard API operation GetAllSubmodelElements
DFF	No standardized information provided[3]	No standardized information provided[3]
AML	Not subject of the AML specification[4]	Not subject of the AML specification[4]
MTP	XML elements (Attributes) identified by Reference IDs, linked (via ID-Link) to communication endpoints	XML elements describing the communication endpoints with all necessary values for ID, Name, Access (read/write/both), Namespace, server endpoint (URI schema + IP address + port)[5, 6]
PA-DIM	OPC UA variables (UA Variable) identified by name	NodeID obtained through browsing
FDI	EDDL variables (VARIABLEs) identified by name, mapped to OPC UA variables	NodeID obtained through browsing (for public variables) or variable access through User Interface Plugins information obtained from FDI Server via UID type node reading (for private variables)

[1] Depending on protocol and server there might be further specifications necessary.
[2] There exist example solutions to realize some kind of communication (e. g., write/read direct access, publish/subscribe mechanism, intermediary implementations) between device (data source) and AAS but nothing standardized so far.
[3] Workarounds using non standardized properties and OPC UA to access them are conceivable.
[4] Mapping of (InternalElements) to OPC UA variables is work in progress and these could be accessed via OPC UA standard browsing service.
[5] The server endpoint must be manually added during the MTP integration process into the POL.
[6] This applies to OPC UA servers.

- **Direct**: Describes a mechanism to directly obtain the location of the TD definition as a URL. This can be provided by Bluetooth beacons, QR codes or as a simple link in a document (e. g., manual).
- **Well-Known**: The well-known approach expects the usage of a predefined URL pattern such as */.well-known/wot-thing-description* that can be used to request and expect the TD.
- **DNS-based**: Things can be discovered via a DNS-based Service Discovery (DNS-SD) approach which can be combined with mDNS. The mDNS response message will provide a link to the location of thing's TD.
- **Directory**: A directory service that manages TDs based on a prescribed API. Search engines like SPARQL or JSON Path can be used to discover specific Things or TDs.
- **Decentralized Identifier**: W3C Decentralized Identifier (DID) [85] approach is applied to resolve a DID document, containing a link to the location of the associated TD.

2) ASSET ADMINISTRATION SHELL DISCOVERY

For discovery of AAS', the concept of a registry is used. Therefore, an application offers interfaces to register and unregister an AAS and also interfaces to publish and discover AAS based on given information. It is possible to use the same process with submodels, as well. The interfaces are defined in an abstract manner and mapped to the HTTP protocol [59].

For registration, a descriptor of the AAS or submodel is submitted to the registry for storage. This descriptor encapsulates the necessary information to access the interface of the AAS or submodel, including details about the endpoint, the global identifier, and optionally, additional information such as description or version number. The range of possible information is defined in the service description. Using the identifier, the stored descriptor can be retrieved. Unregistration implies the deletion of the stored descriptor entry.

Additionally, repositories can be discovered using specific queries to locate the appropriate AAS or information. A formal discovery language, similar to SPARQL, has not been defined yet. In the latest version, the publishing of an asset link has been made possible. This feature enables an application to locate an AAS based on a provided asset identifier.

3) DIGITAL FACTORY FRAMEWORK DISCOVERY

The DFF specification does not provide any information regarding discovery. However, since OPC UA is used as the

data model, its discovery mechanism could potentially be utilized.

4) AutomationML DISCOVERY

AutomationML as a data exchange format for the engineering and commission phase of a production plant does not have any defined discovery mechanism. The ongoing work on the integration with OPC UA [84] will prospectively make OPC UA's discovery mechanism available for AutomationML-based applications.

5) DISCOVERY OF MTP MODULES AND PA-DIM-ENABLED DEVICES

MTP and PA-DIM standards do not explicitly specify a discovery mechanism to discover available modules or automation devices on the network. Still, both standards depend on the OPC UA protocol for communication and thus allow leveraging OPC UA discovery mechanism. In addition, the MTP standard delegates network communication bootstrapping to well-established internet standards such as DHCP (Dynamic Host Configuration Protocol) and DNS (Domain Name System), allowing to address known MTP modules in the plant network [86].

6) FDI SERVER DISCOVERY

FDI (IEC 62796) and its underlying standards do not specify a mechanism for discovery of a specific FDI server. However, the FDI server is based on the OPC UA protocol, allowing to use it's discovery mechanisms. Using the "FDI" capability flag, FDI servers can explicitly inform about their support for the FDI information model to be distinguished from other OPC UA servers during discovery.

F. DISCUSSION

After comparing the individual standards in detail and describing the results in the previous sections, a discussion of the standards with respect to the four use cases defined in Subsection II-B is given in the following. The four use cases are:

- automation device bootstrapping, esp. for automation device discovery
- representation of static device properties (with defined semantics) (property representation and description)
- description of device capabilities and services (asset's service and virtual description's service)
- description of automation device communication interfaces, including services and properties

A brief description is given of how well each standard is suited to serve the considered use cases in our opinion. In addition to a textual evaluation, the following metric is used:

--	No suitability
-	Poor suitability, lacks in many points
o	Suitable, but there are still open points
+	Good suitability, but there could be minor improvements in some points
++	Full suitability

The results are summarized in Table 5.

1) WoT TD

The W3C WoT standardization activity formalizes several discovery options (e. g., central- and decentral-based) and therefore, is suitable for the first use case.

Device properties are realized by the property affordance approach in TD definitions. Such property definitions can be optional enriched with additional semantics to precise the context of the property. However, property affordances are not intended to represent actual property value and are used to describe the way how to get to the actual value.

Automation device capabilities are reflected by the WoT TD document, while virtual representation services are not supported up to now. Based on the context extension approach in TDs, the behavior of the automation device can be described. However, services are not callable, because the TD is just a file format.

The last use case is fully supported through the property, action and events affordances, which include information, such as address information (e. g., nodeID or Modbus address), content type (e. g., binary or JSON) and other communication metadata of an underlying protocol.

2) AAS

The standard only provides a limited interface specification with few discovery options for automating device bootstrapping and discovery. One explanation could be the absence of an information model (or submodel) for the automation device address. As a result, there is no automated self-registration mechanism available for an automation device with an AAS, and there is also no defined workflow for how such registration should be performed. Furthermore, the registry concept is incomplete, as it does not include a mechanism for discovering the registry server itself, nor does it provide for hierarchical discovery.

Since the primary focus of the AAS is to represent automation device related property information, the standard is well-suited for the second use case. It provides support for a variety of data structures, such as Property, Blob, or Range, and includes attributes that represent semantics.

The standard defines a model element called *Capability* for describing the capabilities of automation devices. This model element has no attributes and is solely used to reference a semantic definition of the capability. While the usage of this element is not described in the standard, a reference to another publication [87] is provided. However, as this is just a white paper, it is unclear whether it is intended as a supplement to the standard or just for discussion purposes. If it is intended as a supplement, it should be integrated into

TABLE 5. Summary of the suitability of the compared standards considering the use cases from Subsection II-B.

	WoT	AAS	DFF	AML	MTP	PA-DIM[2]	FDI
automation device bootstrapping	++	o	--	—	--	-	-
automation device static and dynamic properties	o	++	o	+	-	+	+
automation device capabilities and virtual representation services	o	+	--	+	--	+	++
communication interface for properties and asset services	++	--[1]	--	o	++	++	++

[1] Is under development in a working group of the IDTA
[2] Only usable for OPC UA

the standard. To define the services of automation devices and virtual description services, the standard includes a model element called *operation*, which enables the description of a particular functionality.

The fourth use case is not yet supported due to the absence of an information model (submodel template) for this type of information. A working group from the IDTA is currently addressing this issue (also see Section V-B2). According to the API definition in [59], it should be possible to invoke an operation even when there is no option to specify the endpoint location.

3) DFF

The standard does not provide any information about bootstrapping or discovery. However, since OPC UA is used as a data format, it is possible to utilize its discovery mechanisms. This approach, however, only supports OPC UA use cases for finding OPC UA servers based on profiles, rather than representations of specific automation devices.

Analogous to AAS, the primary focus of this specification is the representation of automation device properties. The standard provides a data model for modeling assets (automation devices), which consists of model elements for asset properties (data elements) and relationship elements (references). These properties can be structured using so-called CDELs, which are collections of such data elements. Unlike in AAS, there are no different kinds of data elements; therefore, the entire model semantics are defined in the referenced data element type, which is specified in a concept dictionary. On one hand, the complete interoperability of this standard relies on well-maintained concept dictionaries. On the other hand, it remains open for future data element types without requiring changes to the standard. However, it is unclear how interoperability can be achieved with data elements from different concept dictionaries, as there is no authoritative organization as in the case of AAS.

The same problem that occurs for automation device properties also applies to the description of automation device capabilities and services. As there are no special data elements for this information, interoperability is dependent on the concept dictionaries and the establishment of tools.

CDELs could be used for describing automation device communication interfaces. Once again, CDELs are defined generically, with their semantics specified in the referenced CDEL definition element within a concept dictionary. As of now, the authors are not aware of any concept dictionary that includes such a CDEL definition.

4) AutomationML

As of today, AutomationML as a standard does not support any bootstrapping and discovery mechanism. The future combination with OPC UA, might bring a change, but that would only allow the discovery of the OPC UA server and not the automation device itself. The combination focuses on bringing AutomationML data from the engineering phase to the operational phase of a production system.

For the second use case, AutomationML provides a well structured data model within the CAEX standard to describe the production system properties, including references to COLLADA or PLCopen XML data. For now, it uses the ECLASS dictionary to provide semantics for these properties which allows properties interpretation to be uniform within different tools using a particular AML file.

Description of automation device capability and services within a production system can be realized in PLCopen XML file which is part of the AML file. In PLCopen XML, the behaviour of the automation device in a production system is described as an SFC and linked to other automation devices with virtual IOs.

On the fourth use case, the description of automation device communication is done at the level of CommunicationInterfaceClass Library which is part of interface classes that describes the relations between elements within CAEX and COLLADA. Just like EDD and TD, AutomationML is basically a descriptive file and does not have any mechanism to deploy it as a server instance. The mapping of its elements to the OPC UA information model and the instantiation of AML2Nodeset as OPC UA server instance is still work in progress.

5) MTP

MTP does not provide a mechanism for automated discovery and bootstrapping of process modules (i. e., automation devices). Instead, the MTP standard delegates to standard internet protocols for bootstrapping (DHCP) and address resolving (DNS) of known modules. In addition, one may use the discovery mechanisms provided by OPC UA, which is used as MTP's primary communication protocol, which do not allow the discovery of the automation device itself.

Since MTP primarily focusses on modeling the runtime interface, it does not provide extensive modeling solutions for

properties with their value. MTP DataAssembly objects allow to represent properties and can provide basic meta information like data type and unit of measurement. In addition, MTP can only provide information about each *module type*, not about a single module.

MTP allows to provide information about the services provided by the described module type, specifically, it specifies how the described services and procedures can be activated, parameterized and controlled at operation time. However, services and procedures cannot be tagged with additional metadata, making MTP unsuitable for describing asset's capabilities or virtual representation services.

Describing the communication interface of an automation device is the MTP standard's primary focus and strength. The interface behaviour of MTP modules is mainly predefined by the MTP standard, whereas all device-specific aspects such as available dynamic properties, services or service parameters, can be defined in the MTP package file. Although the design of MTP is tailored to the process industry, it should be flexible enough to use it's communication interface model for a wider range of automation domains.

6) PA-DIM

OPC UA provides a comprehensive set of discovery services to detect server endpoints. Once detected and successfully connected to the server endpoint, the OPC UA browse service can be used to even detect non-standardized elements. However, it is not possible to find a specific automation device based on its automation device id, its properties or any other criteria. The PA-DIM Companion Specification defines standardized types and elements, which can be considered similar for all automation devices modeled according to this specification. Hence, bootstrapping and especially replacement of automation devices that implement interfaces according to the same PA-DIM specification can be considered less complex.

Properties in the PA-DIM are modeled as variables, always comprising a distinct type and a dictionary reference determining their semantics. Because PA-DIM defines only a few mandatory properties but rather various optional properties, automation devices that vary greatly in complexity can be described reasonably well.

Automation device capabilities and services that are represented by methods are inherently executable.[17] Purely descriptive capabilities which are not intended for dynamic changes can only be modeled as static properties. As with properties, the semantics of a method is defined by a reference to a dictionary entry.

The OPC UA standard makes all specified automation device information accessible via its interface. To sum up, OPC UA and PA-DIM combine automation device description and access for process automation devices in a well elaborated way exept for the drawbacks in the bootstrapping functionality.

[17]They are callable in principle if permissions are granted.

7) FDI

Automation device information can be packaged as an FDI device package file and delivered to the FDI engine by manually importing it into a configurable tool. FDI can only leverage OPC UA discovery mechanisms for setting up communication with the FDI server. A discovery of other automation device information is not possible.

The description of automation device properties is contained in an EDD file that is part of the FDI device package. The EDD provides different elements like Variable, RECORD, ARRAY e.t.c to describe automation device properties but does not provide any semantics on these properties. A mapper in the FDI engine maps the EDD properties to the OPC UA data model. At the OPC UA level, an integration of ECLASS dictionary allows the EDD properties being mapped to OPC UA to be semantically defined and identified.

The services of an automation device are described with EDDL METHODs and mapped to the FDI OPC UA object called ActionType. The definition of how the service can be invoked are provided as OPC UA Method nodes using state machines for each action. These nodes are components of the ActionServiceType object.

The description of automation device communication interfaces and properties is primarily the main focus of EDD. The language makes it possible to define different kinds of interfaces based on IEC 61158 fieldbus protocols and some other Ethernet protocols. The mapping to the OPC UA information model hides the communication information of the automation device at the top level, however, the FDI engine provides functionalities to route the information between the FDI server and the automation device.

8) SEMANTIC ANNOTATIONS

Most of the standards allow for semantic annotation virtual descriptions' data elements by means of a reference to a semantic description. For the use of this semantic tagging there are still some practical issues, arising from the coexistence of different dictionaries with semantic definitions using different identifier naming schemes. Therefore, solutions are needed for combining semantic descriptions from different dictionaries. One possibility is tagging the identifier of each semantic definition. Yet, it is required to determine whether two identifiers described in different dictionaries refer to the same semantic concept. So far, none of the presented standards addresses this issue.

VI. COMBINATION

A. INTRODUCTION

In today's industrial automation, the goal of interoperability has led to ever-increasing collaboration between standardization bodies. Such collaborations allow for the use of multiple existing technologies and standards to create a solution to automation problems by ensuring interoperability and overcoming the limitations of one model or standard through integration with another. In this section, a classification of

generic integration strategies is presented (Subsection VI-B), subsequently exemplified by a concrete implementation concept in Subsections VI-C.

B. GENERIC INTEGRATION STRATEGIES

The description of automation device properties and services is achieved by a structured and well-defined information model as shown previously in this paper. These information models are designed to enable the integration of static and dynamic information of automation devices into automation systems. A combination of such information models could help to make some automation solutions more broadly applicable and suitable for all target use cases. The general strategies of model integration or combination are:

1) Model composition
2) Model transformation
3) Combination of model composition and transformation

To evaluate each of these strategies, the following metrics will be used throughout this section: (a) software required to retrieve information of the model being integrated, (b) potential information loss, (c) need for an additional standard and (d) possibility to append or attach information to specific elements of the integrated model.

1) MODEL COMPOSITION

Composition of information models involves the integration of one model into an instance of another model without altering the structure of both models. This method can be achieved in two ways as described in Fig. 13. The first way is to define the resource location of the integrated model (model A in Fig. 13) in the main model (model B in Fig. 13). The second way is to integrate the data of one model in its entirety into the main model as BLOB or a supported file. In most cases, this is made feasible via a model structure that supports hierarchical or modular models by design.

This method of integration is sometimes favorable in an application that uses model B as its base model but that also needs additional information from another model to maximize its functionality.

a: REQUIRED SOFTWARE

The integration procedure only requires software for parsing, interpreting and manipulating model B, since the data and structure of model A are left unchanged.

Accessing and using information from the integrated model (model A) requires a parser and a dedicated interpreter for model B as well as a parser and interpreter for model A.

First, the parser lexically and syntactically analyses the elements of model B and decomposes them into smaller elements. After parsing the model, the output is fed to the interpreter for model B. This resolves the semantics of elements defined in the metamodel of model B, including the data or reference for model A. To access the contained information, a suitable parser and interpreter for model A are required.

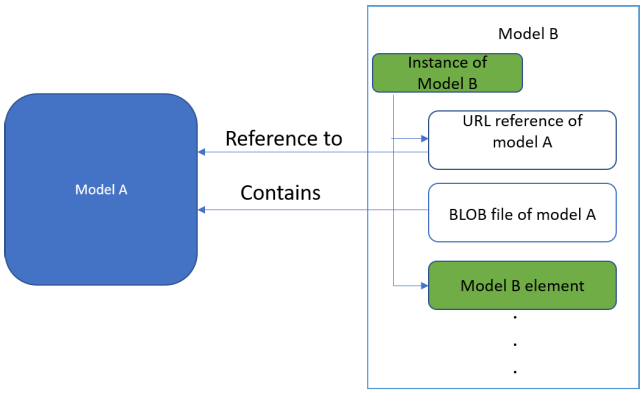

FIGURE 13. Model composition.

b: INFORMATION LOSS

Model integration by composition does not result in any information loss because each model has its own dedicated interpreter at implementation level.

c: REQUIRED STANDARDIZATION

Because the models are processed independently, there is no connection between the models other than creating a placeholder for the content of integrated models in the main model. Thus, this approach does not require any specification of an additional standard, but it might come with a need for some implementation documentation for developers to understand how the models can be used together.

d: POSSIBILITY OF APPENDING OR ATTACHING SPECIFIC INFORMATION FROM MODEL A THROUGH MODEL B

Both general information about model A and additional information about specific elements from model A can be added to model B. While for the first case the standard elements of model B can be used, for the second case a concept for addressing and referencing elements in the structure of model A is needed. This means that for model B, the element containing the information to be appended or attached must contain an additional element containing a reference to the corresponding element of model A.

2) TRANSFORMATION

Model transformation uses a mapping approach to integrate elements of the source model A into the target model B (see Fig. 14). This is done by analyzing the structure of the considered model and the semantics of its elements.

In industrial automation, most information model designs are based on an object-oriented concept, which helps to map similar classes of objects with elements that are alike. The transformation method uses a set of rules called mapping or transformation rules to map elements from source to target models.

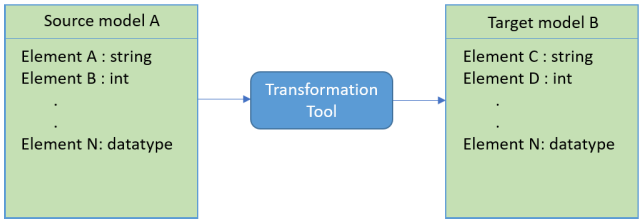

FIGURE 14. Model transformation.

Opposed to the case of composition, the data originating from the source model is represented in the target model structure after integration.

a: REQUIRED SOFTWARE

Integrating the data of a source model A into target model B requires a parser and an interpreter for model A as well as a software tool built specifically for mapping the model elements. Fig. 15 displays an overview of the transformation process.

The software tool (transformation tool) iterates through all elements of the source model A and assigns the contained data to corresponding elements of the target model B. This process is based on a set of so-called modeling rules. These rules specify the elements to be mapped to each other and sometimes define a set of conditions for a feasible mapping. Once the mapping is complete, the result is serialized to the data representation of target model B.

The transformation can be executed in offline or online mode. The major difference between these two transformation modes lies in their execution frequency: the model transformation is only executed once or when explicitly triggered in the offline mode, while for the online mode, the transformation is carried out on-the-fly at every read or write request.

While the transformation tool is on the data provider level accessing and using information from the integrated model (model A) for an user just requires a parser and a dedicated interpreter for model B.

b: INFORMATION LOSS

The meta-models of most standards are structured differently. These differences in model structure can significantly reduce mapping accuracy, resulting in information loss during model transformation. The larger the structure of the source model compared to the target model, the more loss of information can apply.

In the best case, a transformation method operates bidirectionally, which allows for backward and forward transformations between source and target model without any loss of information. This case is restricted to situations in which the structure and semantic definition of elements in both models match to a sufficient degree.

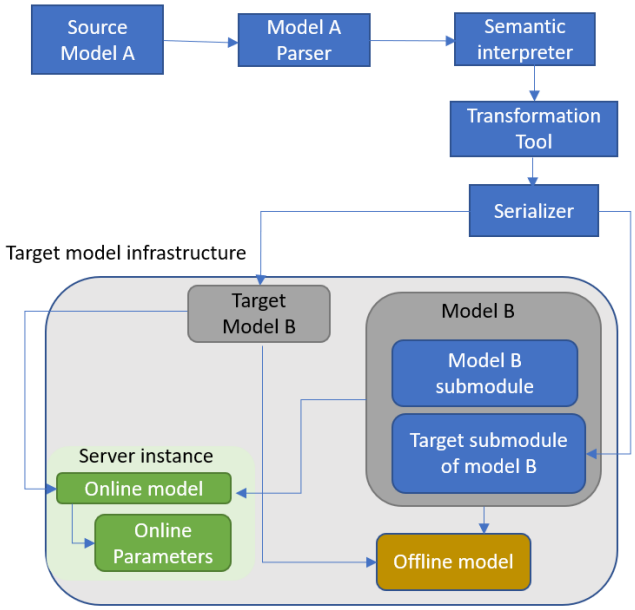

FIGURE 15. Overview of model transformation integration.

c: REQUIRED STANDARDIZATION

Unlike the composition method, solutions based on the transformation method always require standardization of the model integration, in addition to the standardization of the source and target model. This additional standard must describe the exact representation of the integrated model elements in the main model, i.e., the transformation rules, allowing the correct interpretation of the integrated model data. Typically, this standard is featured in one or both of the technology standard series.

d: POSSIBILITY OF APPENDING OR ATTACHING SPECIFIC INFORMATION FROM MODEL A THROUGH MODEL B

With the transformation method, the elements and objects of the source model are completely mapped onto the target model, whereby the elements of the source model adopt the structure of the target model. Therefore, additional information can easily be appended or attached to data from a source model A by using native modeling elements of model B for its representation.

3) COMBINED COMPOSITION AND TRANSFORMATION METHOD

As a full one-to-one transformation of models is often not feasible, a combination of the composition and the transformation method can be favorable for some applications. This combination can either involve a transformation of some part of the source model to the target model and a composition of the remainder into the target model; or the entire source model is transformed to the target model with an additional placeholder for or reference to the source model being created in the target model (see Fig. 16).

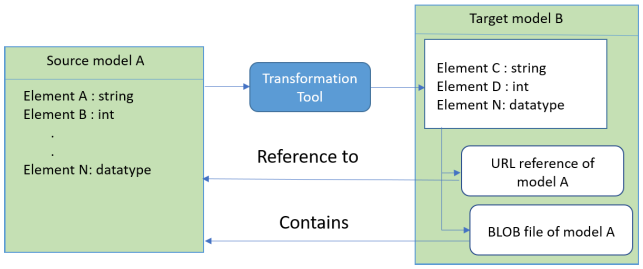

FIGURE 16. Combination of model Composition and model Transformation.

A prototypical use case of the latter combination type would be a solution that uses a complex model to deploy distinct services within one environment. One service could use the transformed model while the other service might use the composed model. This, however, may increase the implementation complexity.

a: REQUIRED SOFTWARE

The software required for this method is identical to the total software used for the two methods already discussed.

b: INFORMATION LOSS

The information loss with this method of integration mainly affects the model transformation part of the source model.

If the source model structure is more robust than the target model, the model transformation is used for parts of the source model that could be reverse engineered or bidirectionally transformed to minimize information loss. The model composition approach can be applied to parts of the source model that are more complex in their representation than the target model.

c: REQUIRED STANDARDIZATION

Due to its implementation complexity, a combined solution requires a standard that explicitly explains which parts of the source model need to be transformed, which parts need to be composed and how the integration could be achieved.

d: POSSIBILITY OF APPENDING OR ATTACHING SPECIFIC INFORMATION TO ELEMENTS OF MODEL A THROUGH MODEL B

For the different model parts, whether composed or transformed, the corresponding techniques apply (refer to VI-B1 and VI-B2).

C. EXAMPLE COMBINATION OF TD AND AAS

In this section, the concept of integrating a WoT TD (see Subsection IV-B) into an AAS submodel (see Subsection IV-C) is briefly sketched; the full example can be found in [88]. The proposed submodel template aims at describing the asset interface of the AAS and serves as an example for an information model transformation.

The TD metamodel (Fig. 1) is composed of finite sets of classes, each represented by a set of attributes. Every attribute could be a variable of a simple type, i.e., string, float or integer, an object or a data container such as a map, or a list. For the AAS, the complexity of objects or data containers can result in a large file. Thus, simpler mapping rules between the source and target metamodels are proposed in [88] and displayed in Fig. 17.

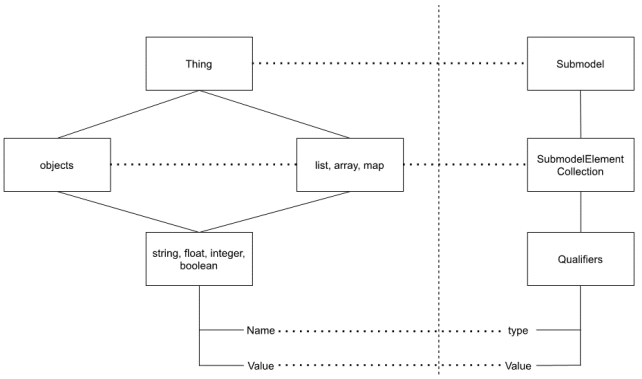

FIGURE 17. Mapping between TD and AAS metamodel.

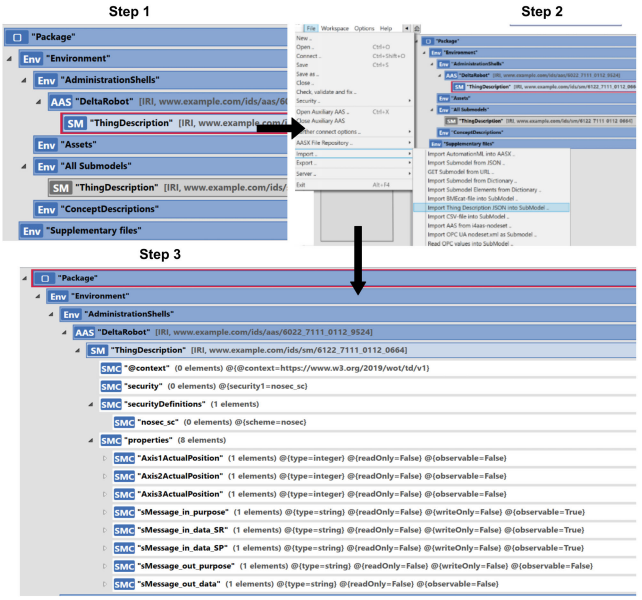

FIGURE 18. Screenshot of the AASX package explorer depicting on how to import a thing description document.

All the attributes of simple type are mapped to the corresponding submodel element qualifiers of the AAS metamodel (Fig. 2), where the attribute names become the qualifier types and the values become the qualifier values. All the complex types including lists, classes and objects are mapped to submodel element collections. The thing class of the TD is mapped to the AAS submodel.

Every element (vocabulary) of the TD metamodel is defined with an RDF schema that is mapped to the semantic identifier of the corresponding AAS element. Furthermore, there are a few exceptions from these basic mapping rules, namely those for title, description, versionInfo and id [88].

The presented approach can be realized as an implementation for the AASX package explorer.[18] Fig. 18 shows an

[18]https://github.com/admin-shell-io/aasx-package-explorer

example of how the content of a TD can be mapped to an AAS submodel representation. This content enables AAS applications to retrieve metadata for accessing asset properties or services at runtime on the basis of a standardized approach.

VII. CONCLUSION AND OUTLOOK

In this paper, different standards for the automated integration of automation devices into an automation system are compared. After giving a literature review six standards where selected and analyzed: W3C Thing Description (TD), Asset Administration Shell (AAS), Digital Factory Framework (DFF), AutomationML (AML), Module Type Package (MTP), Process Automation - Device Information Model (PA-DIM) and Field Device Integration (FDI). These standards are compared with respect to four categories: (1) Representation of a property, (2) Representation of services, (3) Information modeling for a direct automation device access and (4) Mechanism for discovery. Based on this comparison, the advantages and disadvantages of each standard are summarized. The result is that there is no universal standard that addresses the four categories equally well, however, a combination of some standards would make sense. Therefore, different generic integration strategies are presented as well as an example combination of TD and AAS.

The paper shows the importance of such comparison to help manufacturers, suppliers, integrators and operators of automation components as well as current researchers and standardization bodies to evaluate the suitability of single standards or combination of standards. In future work, further aspects of asset information model standards are to be investigated, including (1) means of representating statefulness in the automation device interface (i. e. required order of service invocation), (2) definition of additional aspects of the ecosystem and architecture and (3) ''explorability'' of the used information and data model including versioning, protocol profiles, etc. to reduce required knowledge of the client.

REFERENCES

[1] M. Mohamed, ''Challenges and benefits of industry 4.0: An overview,'' *Int. J. Supply Oper. Manage.*, vol. 5, no. 3, pp. 256–265, 2018.

[2] L. D. Xu, E. L. Xu, and L. Li, ''Industry 4.0: State of the art and future trends,'' *Int. J. Prod. Res.*, vol. 56, no. 8, pp. 2941–2962, Apr. 2018.

[3] W. MacDougall, *Industrie 4.0: Smart Manufacturing for the Future*. Berlin, Germany: Germany Trade & Invest, 2014.

[4] D. Lou, J. Höller, D. Patel, U. Graf, and M. Gillmore, *The Industrial Internet of Things Networking Framework*, document 7, Industry IoT Consortium (IIC), 2021.

[5] *IEEE Standard for an Architectural Framework for the Internet of Things (IoT)*, IEEE Standard 2413-2019, 2019.

[6] *European Telecommunications Standards Institute (ETSI)*, Standard ETSI TS 103 267, 2020.

[7] K. Kajimoto, M. Lagally, T. Kawaguchi, R. Matsukura, and K. Toumura. (2022). *Web of Things (WoT) Architecture 1.1*. W3C working draft. [Online]. Available: https://www.w3.org/TR/2020/WD-wot-architecture11

[8] 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. 22nd IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2017, pp. 1–8.

[9] E. Oztemel and S. Gursev, ''Literature review of industry 4.0 and related technologies,'' *J. Intell. Manuf.*, vol. 31, no. 1, pp. 127–182, Jan. 2020.

[10] E. Y. Nakagawa, P. O. Antonino, F. Schnicke, R. Capilla, T. Kuhn, and P. Liggesmeyer, ''Industry 4.0 reference architectures: State of the art and future trends,'' *Comput. Ind. Eng.*, vol. 156, Jun. 2021, Art. no. 107241.

[11] *Asset Administration Shell for Industrial Applications—Part 1: Asset Administration Shell Structure (CDV)*, Standard IEC 63278-1 ED 1, Geneva, Switzerland, 2022.

[12] M. Singh, E. Fuenmayor, E. Hinchy, Y. Qiao, N. Murray, and D. Devine, ''Digital twin: Origin to future,'' *Appl. Syst. Innov.*, vol. 4, no. 2, p. 36, May 2021.

[13] S. Boschert and R. Rosen, ''Digital twin-the simulation aspect,'' in *Mechatronic Futures*. Cham, Switzerland: Springer, 2016, pp. 59–74.

[14] R. Sacks, I. Brilakis, E. Pikas, H. S. Xie, and M. Girolami, ''Construction with digital twin information systems,'' *Data-Centric Eng.*, vol. 1, p. e14, Jan. 2020.

[15] T. B. Steel, Jr., ''ANSI/X3/SPARC study group on data base management systems interim report,'' *ACM SIGMOD FDT*, vol. 7, no. 2, 1975.

[16] *OPC Unified Architecture—Part 1: Overview and Concepts*, Standard IEC 62541-1, Geneva, Switzerland, 2016.

[17] *Specification of the Asset Administration Shell—Part 1—Metamodel*, Industrial Digital Twin Association, Frankfurt, Germany, 2023.

[18] *Common Warehouse Metamodel (CWM) Specification*, Object Management Group, Inc., Needham, MA, USA, 2003.

[19] DIN Deutsches Institut für Normung e.v. *Collection: Standards Concerning Industry 4.0*. Accessed: May 3, 2022. [Online]. Available: https://www.din.de/en/innovation-and-research/industry-4-0/standards

[20] I. Grangel-González, P. Baptista, L. Halilaj, S. Lohmann, M.-E. Vidal, C. Mader, and S. Auer, ''The industry 4.0 standards landscape from a semantic integration perspective,'' in *Proc. 22nd IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2017, pp. 1–8.

[21] A. J. Trappey, C. V. Trappey, U. H. Govindarajan, A. C. Chuang, and J. J. Sun, ''A review of essential standards and patent landscapes for the Internet of Things: A key enabler for industry 4.0,'' *Adv. Eng. Informat.*, vol. 33, pp. 208–229, Jan. 2017.

[22] Y. Lu, H. Huang, C. Liu, and X. Xu, ''Standards for smart manufacturing: A review,'' in *Proc. IEEE 15th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2019, pp. 73–78.

[23] Y. Lu, K. C. Morris, and S. Frechette, ''Standards landscape and directions for smart manufacturing systems,'' in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2015, pp. 998–1005.

[24] *Representation of Process Control Engineering—Requests in P&I Diagrams and Data Exchange Between P&ID Tools and PCE-CAE Tools*, Standard IEC 62424, Geneva, Switzerland, 2016.

[25] *Industrial-Process Measurement and Control—Data Structures and Elements in Process Equipment Catalogues—Part 10: List of Properties (LOPs) for Industrial-Process Measurement and Control for Electronic Data Exchange—Fundamentals*, Standard IEC 61987-10, Geneva, Switzerland, 2009.

[26] *Industrial Automation Systems and Integration—Integration of Life-Cycle Data for Process Plants Including Oil and Gas Production Facilities—Part 2: Data Model*, ISO Standard 15926-2:2003, Geneva, Switzerland, 2003.

[27] *Enterprise-Control System Integration—Part 2: Objects and Attributes for Enterprise-Control System Integration*, Standard IEC 62264-2, Geneva, Switzerland, 2015.

[28] *Industrial Automation Systems and Integration—Open Technical Dictionaries and Their Application to Master Data—Part 1: Overview and Fundamental Principles*, Standard ISO 22745-1, Geneva, Switzerland, 2010.

[29] *Field Device Tool (FDT) Interface Specification—Part 1: Overview and Guidance*, Standard IEC 62453-1, Geneva, Switzerland, 2016.

[30] *Industrial Automation Systems and Integration—Open Systems Application Integration Framework—Part 1: Generic Reference Description*, Standard ISO 15745-1, Geneva, Switzerland, 2003.

[31] *Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 1: Overview and Fundamental Principles*, Standard ISO 10303-1, Geneva, Switzerland, 2021.

[32] S. Käbisch, M. McCool, T. Kamiya, and V. Charpenay. (2022). *Web of Things (wot) Thing Description 1.1*. W3C Working Draft, W3C. [Online]. Available: https://www.w3.org/TR/wot-thing-description11/

[33] *Industrial-Process Measurement, Control and Automation—Digital Factory Framework—Part 1: General Principles*, Standard IEC 62832-1, Geneva, Switzerland, 2020.

[34] *Engineering Data Exchange Format for Use in Industrial Automation Systems Engineering—Automation Markup Language—Part 1: Architecture and General Requirements*, Standard IEC 62714-1, Geneva, Switzerland, 2018.
[35] *Industrial Automation System and Integration—COLLADA™ Digital Asset Schema Specification for 3D Visualization of Industrial Data*, Standard ISO 17506, Geneva, Switzerland, 2022.
[36] *Programmable Controllers—Part 10: PLC Open XML Exchange Format*, Standard IEC 61131-10, Geneva, Switzerland, 2019.
[37] *Automation Engineering of Modular Systems in the Process Industry: General Concept and Interfaces*, Standard VDI/VDE/NAMUR 2658, 2019.
[38] *PA-DIM—Process Automation Device Information Model: OPC UA for Process Automation Devices*, FieldComm Group, Austin, TX, USA, 2020.
[39] *Field Device Integration (FDI)—Part 1: Overview*, Standard IEC 62769-1, Geneva, Switzerland, 2021.
[40] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Cham, Switzerland: Springer, 2009.
[41] M. Schleipen, S.-S. Gilani, T. Bischoff, and J. Pfrommer, "OPC UA & industrie 4.0—Enabling technology with high diversity and variability," *Proc. CIRP*, vol. 57, pp. 315–320, Jan. 2016.
[42] *OPC Unified Architecture—Part 2: Security Model*, Standard IEC 62541-2, Geneva, Switzerland, 2018.
[43] *OPC Unified Architecture—Address Space Model*, Standard IEC 62541-3, Geneva, Switzerland, 2020.
[44] *OPC Unified Architecture—Part 4: Services*, Standard IEC 62541-4, Geneva, Switzerland, 2015.
[45] *OPC Unified Architecture—Part 12: Discovery and Global Services*, Standard IEC 62541-12, Geneva, Switzerland, 2018.
[46] T. Miny, "Concept for semantic interoperability between information models," Ph.d. thesis, Chair Inf. Automat. Syst. Process Mater. Technol., RWTH Aachen Univ., Aachen, Germany, 2022.
[47] P.-A. Champin, G. Kellogg, and D. Longley. (Jul. 2020). *JSON-LD 1.1*. W3C Recommendation, W3C. [Online]. Available: https://www.w3.org/TR/2020/REC-json-ld11-20200716/
[48] A. Wright, H. Andrews, and B. Hutton, "A JSON media type for describing the structure and meaning of JSON documents," document Internet Draft Draft-ZYP-JSON-Schema-01, 2020.
[49] P. V. Biron and A. Malhotra, *XML Schema Part 2: Datatypes Second Edition*, World Wide Web Consortium, Standard Recommendation REC-xmlschema-2-20041028, Oct. 2004.
[50] A. Cimmino, M. McCool, F. Tavakolizadeh, and K. Toumura. (2023). *Web of Things (WoT) Discovery*. W3C Candidate Recommendation, W3C. [Online]. Available: https://www.w3.org/TR/wot-discovery/
[51] C. Aguzzi, D. Peintner, and Z. Kis. (2022). *Web of Things (WoT) Scripting API*. W3C Note, W3C. [Online]. Available: https://www.w3.org/TR/2020/NOTE-wot-scripting-api
[52] M. McCool and E. Reshetova. (2022). *Web of Things (WoT) Security and Privacy Guidelines*. [Online]. Available: https://www.w3.org/TR/2019/NOTE-wot-security
[53] *Implementation Strategy Industrie 4.0 Report on the Results of the Industrie 4.0 Platform*, BITKOM e.V., VDMA e.V. and ZVEI e.V., Berlin, Germany, 2016.
[54] B. Boss, S. Bader, A. Orzelski, M. Hoffmeister, M. T. Hompel, B. Vogel-Heuser, and T. Bauernhansl, "Verwaltungsschale," in *Handbuch Industrie 4.0: Produktion, Automatisierung und Logistik*. Berlin, Germany: Springer, 2019.
[55] *Reference Architecture Model Industrie 4.0 (RAMI4.0)*, Standard DIN SPEC 91345, Berlin, Germany, 2016.
[56] *Details Asset Admin. Shell—Part 1—The Exchange Informationen Between Partners Value Chain Industrie 4.0 (Version 1.0)*, Bundesministerium für Wirtschaft und Energie, Berlin, Germany, 2018.
[57] *Asset Administration Shell for Industrial Applications—Part 1: Administration Shell Structure*, Standard IEC 63278-1, Geneva, Switzerland, 2020.
[58] *Standard Data Elements Types With Associated Classification Scheme for Electric Items—Part 1: Definitions—Principles and Methods*, Standard IEC 61360-1, Geneva, Switzerland, 2017.
[59] *Details of the Asset Administration Shell—Part 2—Interoperability at Runtime—Exchanging Information via Application Programming Interfaces*, Bundesministerium für Wirtschaft und Energie, Berlin, Germany, 2021.
[60] *Industrial-Process Measurement, Control and Automation—Digital Factory Framework—Part 2: Model Elements*, Standard IEC 62832-2, Geneva, Switzerland, 2020.
[61] *Industrial-Process Measurement, Control and Automation—Digital Factory Framework—Part 3: Application of Digital Factory for Life Cycle Management of Production Systems*, Standard IEC 62832-3, Geneva, Switzerland, 2020.
[62] M. Babcinschi, B. Freire, P. Neto, L. A. Ferreira, B. L. Señaris, and F. Vidal, "AutomationML for data exchange in the robotic process of metal additive manufacturing," in *Proc. 24th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2019, pp. 65–70.
[63] L. Hundt, R. Drath, A. Lüder, and J. Peschke, "Seamless automation engineering with AutomationML®," in *Proc. IEEE Int. Technol. Manage. Conf. (ICE)*, Jun. 2008, pp. 1–8.
[64] A. Lüder and N. Schmidt, "Automationml in a nutshell," in *Handbuch Industrie*. Cham, Switzerland: Springer, 2017, pp. 213–258.
[65] O. Graeser, L. Hundt, M. John, G. Lobermeier, and A. Lüder, "AutomationML and ECL@SS integration," AutomationML e.V., Magdeburg, Germany, Whitepaper V2.0.0, Nov. 2021.
[66] *Engineering Data Exchange Format for Use in Industrial Automation Systems Engineering—Automation Markup Language—Part 2: Role Class Libraries*, Standard IEC 62714-2, Geneva, Switzerland, 2015.
[67] R. Henßen and M. Schleipen, "Interoperability between OPC UA and AutomationML," *Proc. CIRP*, vol. 25, pp. 297–304, Mar. 2014.
[68] *New Work Item Proposal (NP). Automation Engineering of Modular Systems in the Process Industry—General Concept and Interfaces*, International Electrotechnical Commission, Geneva, Switzerland, 2019.
[69] *Automation Engineering of Modular Systems in the Process Industry: Modelling of Module Services: VDI/VDE/NAMUR 2658 Part 4*, Verein Deutscher Ingenieure, Alexisbad, Germany, 2020.
[70] *Automation Engineering of Modular Systems in the Process Industry: Library for Data Objects: VDI/VDE/NAMUR 2658 Part 3*, Verein Deutscher Ingenieure, Alexisbad, Germany, 2020.
[71] J. Bernshausen, A. Haller, T. Holm, M. Hoernicke, M. Obst, and J. Ladiges, "Namur modul type package–definition," *atp Magazin*, vol. 58, nos. 1–2, pp. 72–81, 2016.
[72] *NAMUR Open Architecture—NOA Concept*, Standard NAMUR NE 175, 2020.
[73] *Self-Monitoring and Diagnosis of Field Devices*, Standard NAMUR NE 107, 2017.
[74] *Function Blocks (FB) for Process Control and Electronic Device Description Language (EDDL)—Part 2: Specification of FB Concept*, Standard IEC 61804-2, Geneva, Switzerland, 2018.
[75] *Devices and Integration in Enterprise Systems—Function Blocks (FB) for Process Control and Electronic Device Description Language (EDDL)—Part 6: Meeting the Requirements for Integrating Fieldbus Devices in Engineering Tools for Field Devices*, Standard IEC 61804-3, Geneva, Switzerland, 2012.
[76] *Field Device Integration (FDI)—Part 7: Communication Devices*, Standard IEC 62769-7, Geneva, Switzerland, 2020.
[77] N. Freed and N. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, document RFC2046, 1996.
[78] *Field Device Integration (FDI)—Part 5: Information Model*, Standard IEC 62769-5, Geneva, Switzerland, 2021.
[79] *Describing Capabilities of Industrie 4.0 Components*, Plattform Industrie, Berlin, Germany, 2020.
[80] *Whitepaper AutomationML Part 2—Role Class Libraries*, AutomationML Consotium, Magdeburg, Germany, 2016.
[81] *UA for Field Device Integration (FDI)—Part 5: Host System Information Model*, Standard OPC 30080-5, OPC Foundation, 2020.
[82] *Field Device Integration (FDI)—Part 3: Server*, Standard IEC 62769-3, Geneva, Switzerland, 2021.
[83] *Field Device Integration (FDI)—Part 1: FDI Client*, Standard IEC 62769-2, Geneva, Switzerland, CH, 2021.
[84] *OPC Unified Architecture Information Model for AutomationML*, AutomationML e.V Office, Magdeburg, Germany, 2016.
[85] D. Longley, C. Allen, M. Sporny, M. Sabadello, and D. Reed. (May 2021). *Decentralized Identifiers (DIDs) V1.0*. Candidate Recommendation, W3C. [Online]. Available: https://www.w3.org/TR/2021/CRD-did-core-20210529/
[86] *Automation Engineering of Modular Systems in the Process Industry: Runtime and Communication Aspects: VDI/VDE/NAMUR 2658 Part 5 (Draft)*, Verein Deutscher Ingenieure e.V., Alexisbad, Germany, 2022.
[87] *Describing Capabilities of Industrie 4.0 Components*, Plattform Industrie 4.0, Berlin, Germany, 2020.

[88] C. Diedrich, H. K. Pakala, K. Oladipupo, and S. Käbisch, "Integration of asset administration shell and Web of Things," in *Kommunikation und Bildverarbeitung in der Automation*. Berlin, Germany: Springer, Nov. 2021.

TORBEN MINY (Member, IEEE) received the B.Eng. degree in mechanical engineering from the RFH–University of Applied Science, Cologne, Germany, in 2014, and the M.S. and Ph.D. degrees in automation from RWTH Aachen University, Aachen, Germany, in 2016 and 2022, respectively.

Since 2022, he has been a Senior Researcher with the Chair of Information and Automation Systems for the Process and Material Technology, RWTH Aachen University. His area of expertise lies in the automated exchange of information through model transformations in the domain of Industrie 4.0. In this context, he deals with the topic of asset administration shell and discovery.

Dr. Miny is an active member of various working groups in national standardization associations in Germany and in international, such as ZVEI, VDI/VDE, NAMUR, Plattform Industrie 4.0, DKE, DIN, OPC Foundation, and IDTA.

MICHAEL THIES (Member, IEEE) received the B.Sc. degree from the Bachelor's Program "Technical Informatics," Gottfried Wilhelm Leibniz University of Hannover, in 2016, and the M.Sc. degree from the Master's Program "Automation Engineering," RWTH Aachen University, in 2019.

He was a Research Assistant with the Chair of Information and Automation Systems for the Process and Material Technology, RWTH Aachen University, from 2019 to 2022. Since June 2022, he has been a software engineer on software solutions for public transport services. His research activities focused on the acquisition, integration, and representation of industrial data.

LINA LUKIC (Member, IEEE) received the B.Sc. degree in mechanical engineering (focus on process engineering) and the M.Sc. degree in automation engineering from RWTH Aachen University, in 2018 and 2021, respectively.

Since 2022, she has been a System Software Developer of weighing terminals with SysTec Systemtechnik und Industrieautomation. Her work focuses on embedded devices communication based on MQTT and OPC UA.

Ms. Lukic is a member of the joint VDMA and OPC Foundation Working Group DMA OPC UA Weighing Technology Initiative.

SEBASTIAN KÄBISCH (Member, IEEE) received the Diploma (Dipl.-Inf. Univ.) degree in computer science from the University of Passau, Germany, in 2008, and the Ph.D. (Dr. rer. nat) degree from the University of Passau.

Since 2013, he has been with Siemens Technology, Munich, Germany, where he is currently a Senior Key Expert. As a Guest Lecturer, he regularly gives lectures with the University of Passau, on the topic of web of things. His work focuses on the efficient realization and usage of standardized internet and web technologies for the Industrial Internet of Things domain.

Dr. Käbisch is an Active Member of various standardization and developer associations, e.g., the Co-Chair of the W3C Web of Things (WoT) Standardization Group, the Co-Editor of the W3C WoT Thing Description Specification, and a Leader of the IDTA Asset Interface Description Sub-Model Working Group for the Asset Administration Shell. He is also one of the initiators and authors of the open source projects, such as OpenV2G, Eclipse node-wot, and Eclipse ediTDor.

KAZEEM OLADIPUPO (Member, IEEE) received the bachelor's degree in electronic and computer engineering from Lagos State University, Lagos, Nigeria, in 2014, and the M.Sc. degree in electrical engineering and information technology from the Otto Von Guericke University Magdeburg, Magdeburg, Germany, in 2021, where he is currently pursuing the Ph.D. degree.

He was a Research Assistant with the Chair of Institute of Integrated Automation, Otto Von Guericke University Magdeburg, from 2021 to 2022. The Focus of his research surrounds the convergence of operational technology and information technology.

Ms. Oladipupo is an active member of working groups, that develops sub-model templates for asset administration shell in Industrial Digital Twin Association (IDTA).

CHRISTIAN DIEDRICH (Member, IEEE) received the Diploma degree in electrical engineering with the option of automation and the Ph.D. degree in semi-formal specification of fieldbus interfaces and fieldbus profiles from the Otto Von Guericke University Magdeburg, Germany, in 1985 and 1994, respectively, under Prof. Dr.-Ing. habil. Peter Neumann.

He was with many German and European research and development projects (main topics are industrial communication, engineering of automation systems, formal description methods, and information and semantic modeling). Since 2005, he has been the Deputy Head of ifak e.V. Magdeburg. He assumed full professorship of integrated automation, in 2006, and holds the Chair of Integrated Automation, Otto von Guericke University Magdeburg, where he is also the Head of the Institute of Control Technology. He is currently collaborating with research institutions and companies on industry 4.0 projects.

Dr. Diedrich is an Active Member of various standardization bodies, such as IEC, DKE, VDI/VDE, ZVEI, and NAMUR.

TOBIAS KLEINERT (Member, IEEE) received the degree in mechanical engineering from RWTH Aachen University, in 1999, and the Ph.D. degree from the Chair of Automation and Computer Control, Ruhr-Universitt Bochum, in 2005, under Prof. Jan Lunze.

Subsequently, he was with BASF SE, worked in advanced process control, process control systems engineering and lifecycle support, regulated automation solutions, manufacturing execution solutions, and smart manufacturing. In 2020, he assumed full professorship of information and automation systems in industrial materials production and processes with RWTH Aachen. He collaborates with research institutions and industry. His research and teaching include concept and method development, system solution design, and general concept formation for standardization in smart manufacturing. His current research activities include automated process data processing, component-based process control, and virtualization for safe and secure automation systems.

Dr. Kleinert is engaged in standardization bodies, such as Plattform Industrie 4.0, DKE, VDI/VDE, ZVEI, and NAMUR.