http://dx.doi.org/10.25673/39570

Integration of Asset Administration Shell and Web of Things

H. K. Pakala*, K. O. Oladipupo*, S. Käbisch, Ch. Diedrich*,**

**Otto von Guericke University Magdeburg, Magdeburg, LSA 39106, Germany*
(e-mail: christian.diedirch@ovgu.de, Harish.Pakala@ovgu.de, kazeem.oladipupo@st.ovgu.de)
***Siemens AG, München, B 80333, Germany (e-mail: Sebastian.kaebisch@siemens.com)*

Abstract:

In the course of the digital transformation of industrial production processes, physical assets are encapsulated with software components, where a set of such computing entities working in unison form and build the cyber-physical systems (CPS). The Platform Industry 4.0 (PI4.0) provides a standardized meta-model in terms of submodels and submodel elements to digitally represent information about an asset. The submodels are used to represent a wide range of information like the asset nameplate, administrative aspects, technical specifications and even the accessibility information. Some of these data have to be acquired from the asset. The W3C Web of Things provides the Thing Description meta-model to represent the accessibility runtime information of these assets.

This paper aims to map the W3C Web of Thing (WoT) Thing Description (TD) to the submodel and the submodel elements of the PI4.0 Asset Administration Shell (AAS). A standard submodel template encapsulating each of the Thing Description classes would be the outcome of this paper and also a plugin module embedded into the AAS package explorer that utilizes this template to automatically generate AAS submodel representation from TD for any given asset. This paper also presents a brief description of a use-case consisting of a Delta Robot which can be accessed via an OPC UA server.

Keywords: automation systems, cyber physical systems, Asset Administration Shell, Web of Things

1. Introduction

Cyber Physical Systems (CPS) is the network of a finite set of computing devices that are encapsulated over physical assets, where the network ensures a bi-directional communication between computing and the physical devices (Alur 2015). The information exchanged between different entities over this network can be channelled, extracted and be used for various tasks like machine learning, data analytics, predictive maintenance and, to control and monitor systems through SCADA softwares. These tasks inturn help in providing an efficient production processes and thereby enabling a self-optimizing and self-managing production system. Internet of Things (IoT) refers to the networking of the physical and the virtual world over the internet (ITU-T Y.4000).

The Platform Industry 4.0 (PI4.0), a network of industry stakeholders, research organizations and like-minded partners is a German government initiative, with the primary aim to integrate CPS with the usage of IOT into the industrial production processes (Henning Kagermann 2013). The PI4.0 network is organized into six different working groups, where each group is associated with specific tasks like creation or adaptation of existing standards, security and legal aspects, creation of new use-cases, education and training.

The group that deals with standardization has coined a new term Asset Administration Shell (AAS), a standardized digital representation of an asset. Accordingly, the group has proposed a meta-model to represent the information of an asset digitally (Platform Industry 4.0). This meta-model is primarily composed of submodels and submodel elements like property, range, reference element, multi-language property, operation, event, file and blob. The meta-model is a more generalized one and any information about an asset like nameplate, technical data, access related information can be modelled using the submodels (SM) and the available submodel elements (SME). (Platform Industry 4.0)

The concept of W3C Web of Things (WoT) and the Thing Description is another concerted effort of W3C foundation (McCool and Kaebisch 2021). The group provides a standardized meta-model termed as Thing Description (TD) that aims to increase the interoperability and easier integration of runtime data and functions of IoT platforms (Kaebisch et al. 2021). The meta-data has constructs such as properties, actions, events, forms, links and security definitions. The entire meta-data is well organized and each construct is well attributed in the sense of software development technologies. This enables it to include wide range of access related information of an asset. Both the representations AAS and Thing Description are human readable and address both the physical and logical assets in the realm of the cyber physical systems. The Thing Description restricts itself to the aspects of the accessibility, while the AAS meta-model enables to represent wide range of information like digital nameplate or documentations.

In this paper we aim to integrate Thing Description meta-model with asset administrations shell's SM and SME. We consider this aspect as a problem of mapping one meta-model onto another. A set of mapping rules will be introduced which can be used by a model translator for transforming Thing Description instances into AAS submodel Instances. The AAS package explorer is the official modelling tool from the PI4.0, a plugin module will be integrated into this modelling tool as part of this work.

The rest of the paper is organized into six sections, the next section provides background work related to Asset Administration Shell (AAS) and its meta-model, Thing Description and its meta-model. The third section presents a brief summary of past works from different authors in the field of model transformation or model mapping.

2. BACKGROUND WORK

In this section, a brief overview of the AAS meta-model and the TD meta-model is presented.

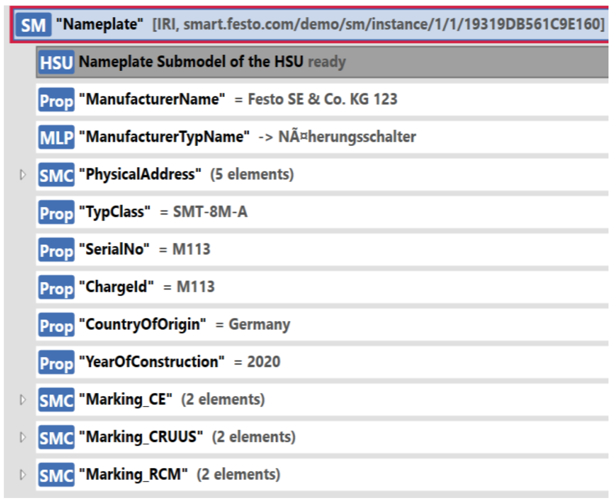
2.1 Asset Administration and its Meta-Model

The PI4.0 defines an Asset Administration Shell *as a standardized digital representation of an asset where the asset could be a logical or a physical entity (Platform Industry 4.0)* and consequently it proposes a meta-model so as to structure the information about the asset in a standardized manner. This structured information is both human and machine readable. It can be represented either in JSON or XML formats and be wrapped in an AASX package container format.

The meta-model aims to structure the information in terms of a collection of finite number of submodels and each submodel containing a set of elements from the available submodel elements. Each submodel is expected to represent a specific category of information like nameplate, asset access related information, technical document, etc. Figure 1 presents the screen shot from AASX package explorer of the nameplate submodel for delta-robot asset considered for the use-case in this paper.

The UML diagram in Figure 2 shows the shortened version of the AAS meta-model, not all the entities are shown in this diagram, the complete description is available at (Platform Industry 4.0). Figure 2 captures the classes AAS, Asset, SM, DataElement, MultiLanguagePropery (MLP), SME and the information about their respective inherited classes. The complete description of the inherited classes is not represented in this diagram.

The SM and SME classes inherit the HasSemantics class. This class enables to associate specific semantic references to the respective instances. Similarly, both SM and SME are Qualifiable, this is to specify a set of qualifiers, where each qualifier is type-value pair that enables to specify additional information about the specific instance. A semantic reference can also be attributed to a qualifier as it inherits the HasSemantics class. Additionally, AAS and SM inherit the identifiable class. This class is defined to attribute unique identifiers and version, revision information to the specific instances. The Referable class inherited by the SME class enables to specify idShort attribute to the specific instance which represent an identifier valid in the local instance of an AAS.



SM "Nameplate" [IRI, smart.festo.com/demo/sm/instance/1/1/19319DB561C9E160]
HSU Nameplate Submodel of the HSU ready
Prop "ManufacturerName" = Festo SE & Co. KG 123
MLP "ManufacturerTypName" -> NÃ¤herungsschalter
SMC "PhysicalAddress" (5 elements)
Prop "TypClass" = SMT-8M-A
Prop "SerialNo" = M113
Prop "ChargeId" = M113
Prop "CountryOfOrigin" = Germany
Prop "YearOfConstruction" = 2020
SMC "Marking_CE" (2 elements)
SMC "Marking_CRUUS" (2 elements)
SMC "Marking_RCM" (2 elements)

Figure 1 Screenshot of Package Explorer depicting nameplate submodel (Ristin et al. 2021)

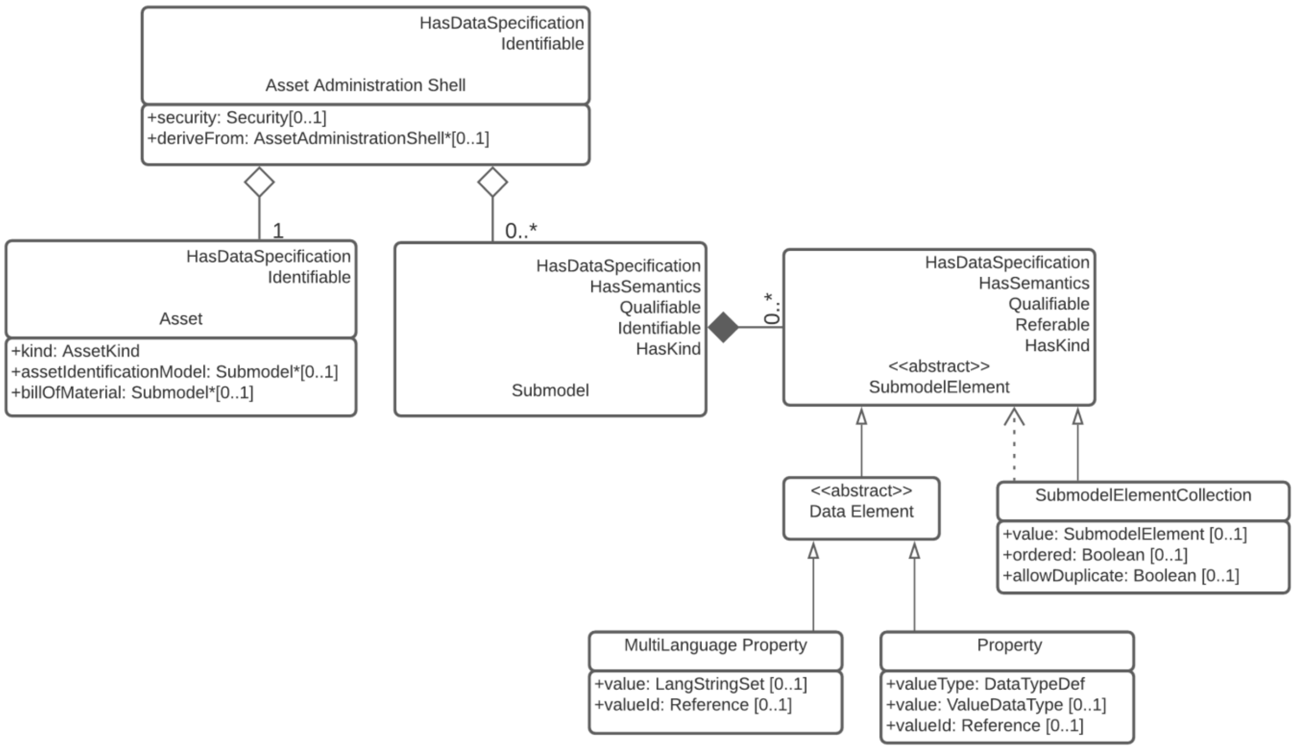

HasDataSpecification
Identifiable
Asset Administration Shell
+security: Security[0..1]
+deriveFrom: AssetAdministrationShell*[0..1]
1
0..*
HasDataSpecification
Identifiable
Asset
+kind: AssetKind
+assetIdentificationModel: Submodel*[0..1]
+billOfMaterial: Submodel*[0..1]
HasDataSpecification
HasSemantics
Qualifiable
Identifiable
HasKind
Submodel
0..*
HasDataSpecification
HasSemantics
Qualifiable
Referable
HasKind
<<abstract>>
SubmodelElement
<<abstract>>
Data Element
SubmodelElementCollection
+value: SubmodelElement [0..1]
+ordered: Boolean [0..1]
+allowDuplicate: Boolean [0..1]
MultiLanguage Property
+value: LangStringSet [0..1]
+valueId: Reference [0..1]
Property
+valueType: DataTypeDef
+value: ValueDataType [0..1]
+valueId: Reference [0..1]

Figure 2 Class diagram representation of AAS Basic meta-model (Platform Industry 4.0)

The submodel can have a finite set of SME, where this SME could be a submodelElementCollection (SMEC) or a dataElement or any other listed SME from the meta-model. The SMEC can also have a finite set of the SME including another SMEC. The MLP (MultiLanguageProperty) data element is used to specify same text in different languages. The other SME like property, reference, range, file, blob, capability, operation, basic event, entity, relationship element and annotated relationship element are not mentioned in this UML diagram presented in Figure 2, the relevant reasons behind the exclusion is provided in the section 6 of the paper.

2.2 THING DESCRIPTION and TD META-MODEL

The Thing Description (TD) is a recommended JSON-LD based document developed within the W3C Web of Thing (WoT) group (McCool and Kaebisch 2021; Kaebisch et al. 2021). The main idea of a TD is to provide a standardized semantic interface description as an entry point for a physical asset to easily onboard its runtime data and functions as well as enable rapid IoT application development. The WoT working group provides a top meta-model for the TD consisting of well-defined set of classes. Figure 3 presents the top-level meta-model of TD.

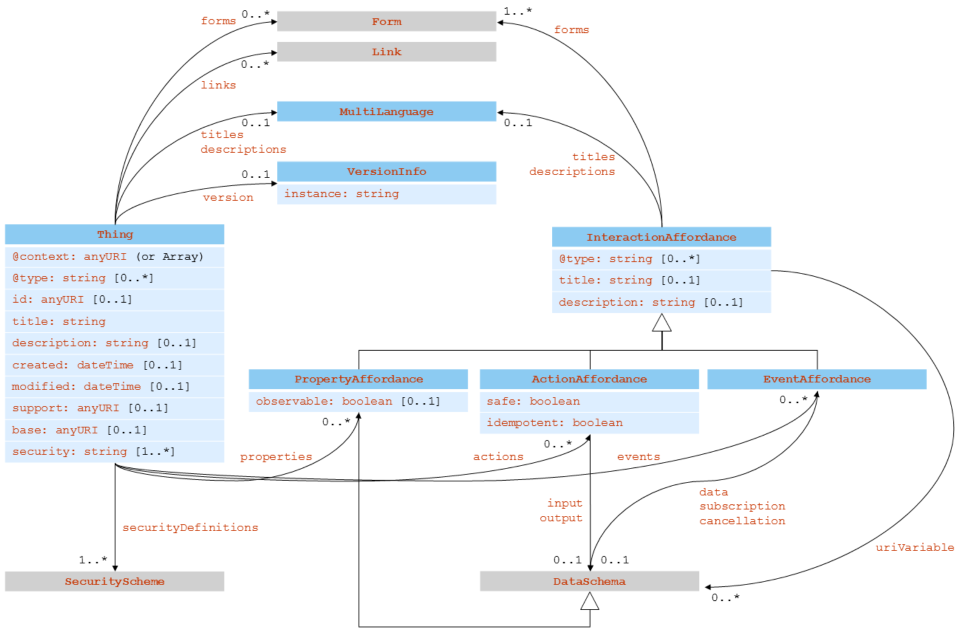


Figure 3 **TD meta model (Kaebisch et al. 2021)**

The *Thing* class provides the core vocabularies for describing the asset's metadata (e.g., title, ID, etc.) and specifies what kind of properties, actions, and events are exposed. The TD uses different standardized interaction affordances to categorize the capabilities offered by the asset.

1. **Property affordances**: Represent states that an asset exposes. Typically, a property can be read, written to and/or observed. Properties can be also used to define dynamic or static parameters.
2. **Action affordances**: Represent either state manipulations or physical processes that can be invoked.
3. **Events affordances:** Represent notifications (e.g., alarms) and data streams that can be subscribed to.

The *SecuritySchema* class is used to provide metadata about what kind of security modes are used by the asset and what authorization service (if any) needs to be involved in the interaction. The *DataSchema* provides a mechanism for specifying the data model reflected in the properties, actions, and events. In doing so, the *DataSchema* vocabularies rely mainly on JSON Schema terms.

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 Thing Description. 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.

3. RELATED WORK

Meta-models define the structure of a model that is a possible representation of an entity in an abstract way (Markus et al. 2013). For example, Programming languages like Java, Python, C are represented by specific meta-models that provide rules and syntax for modelling the programs. There are a vast number of meta-models each related to a specific domain that attempts to solve a modelling problem for that domain. In an interconnected world, interactions between instances or entities modelled using different meta-models is an aspect of concern. The differences in representations made by different meta-models would require to map one meta-model to another and thereby perform model-to-model transformations. In this section, a summary of different research works that address the aspect of model-to-model mapping or transformations is presented.

The Model Transformation Language (MOLA) as described by (Kalnins et al. 2004) provides a structured and descriptive graphical language consisting of flow charts to transform an instance of a source model confirming to a source meta-model into a target model confirming to the target meta-model. The MOLA insists that both source and target meta-models are compliant with MOF 2.0 standard (Overbeek 2006) where this standard provides a framework and set of services for the creation of meta-models and their interoperability.

To bridge the gap between different applications that utilize different meta-models to represent the available information the authors (Agostinho et al. 2010) have proposed an intermediary conceptual meta-model described using UML representation. This meta-model is expected to have features and concepts such that a mapping between this model and any other meta-model is possible. Firstly, the one-to-one mappings are made between conceptual meta-model and all the other available meta-models and later these are used to translate instantiations of any two different meta-models via the intermediary representation.

(Henßen and Schleipen 2014) try to provide a mapping between the information models OPC UA (Stefan-Helmut Leitner 2006) an IEC 62541 standard and the AutomationML (AML) an IEC 62714 standard (Drath et al.). The underlying concept of the OPCUA meta-model is a node class, specializing this class there are a set of classes likes Objects, Variables, references where references provide the relation between the other nodes. The information about any real-world object can be modelled using these node classes and can be hierarchically organized using Folders. The AML on the other hand is a combination of other standards that is specifically designed to model plant engineering information. An AML representation contains a set of InterfaceClassLibs composed of Interface definitions, RoleClassLibs composed of semantic role definitions, SystemUnitClassLibs composed of reusable AML objects and lastly the Instance Hierarchy representing the actual plant description. The authors (Henßen and Schleipen 2014) have mapped these Libraries to OPC UA folder types, and different classes types as OPC UA object types that are organized into the folder types, new reference types HasAMLChild, HasAMLRoleReference, HasAMLInternalLink are created to model the relationships between the AML Objects and ObjectTypes.

(Lelde et al. 2015) propose a unique approach for situations where there is one common target met-model and different source meta-models are expected to be mapped to it in this approach, a new intermediate virtual meta-model that represents the target meta-model from a higher abstraction level and with a better structural foundation is introduced. Fixed set of mappings between the intermediate and the target meta-model (in both the directions) are provided. New mappings with respect to the intermediate meta-model will have to be created for every source meta-model. With the underlying intermediate-to-target meta-model mappings, it is implicit that a source meta-model instance is transformed to the intermediate meta-model instance.

(Cavalieri et al. 2019) have made an attempt to map AAS (Platform Industry 4.0) meta-model onto the OPC UA meta-model, for complex types such AAS, Submodel, SubmodelElement corresponding OPC UA object types are created. In case of simple types such as version, revision, idShort, value related variables are crated and the HasSemantic entity is mapped to an opcua hierarchical reference type. The collection elements such as submodels, submodelCollections are organized as opcua foldertypes.

(Miny et al. 2020) have proposed custom build model transformation language by extending the features of OCL to translate a source submodel instance to a target submodel instance. A summary of the works presented in the sections indicates that whenever the source and target meta-models follow a standard, a generalized translation framework could be used to translate their corresponding instances. Also, a common higher abstracted meta-model could be used as an intermediate and mapping or translation can be done between this and the actual meta-models. Lastly the custom defined solutions are posed by different authors where the source or the target meta-models do not confirm to exiting standards.

In this paper we propose custom mapping rules between AAS and Thing Description meta-models, the mapping rules are inspired by theory proposed in the (Marco and Yannis 2003). Although the authors attempt to address the aspect of semantic interoperability, the underlying logic seems to be a good fit to the problem addressed in this paper.

4. PROPOSED METHODOLOGY

This section firstly presents a high-level comparison between the two meta-models TD and AAS. Secondly, it proposes new mapping rules for transforming the individual instances from one model representation to another.

The TD meta-model is composed of finite set of classes each represented by a set of attributes, where an attribute could be a variable of simple type like string, float and integer or an object type of another class or a map, list of simple types and of objects. The entire structure is quite complex with circular references where these references could be well suited to accommodate different attributes related to web of things. On the other-hand, even though class diagram representation of the AAS meta-model is much complex, it offers very simple entities like property (of simple type like integer, string, float, anyURI etc), operation that can expect a set of input and output variables, multi-language property, Entity, BasicEvent, Qualifier, Range, File, Blob, Submodel Element Collection, Submodel etc.

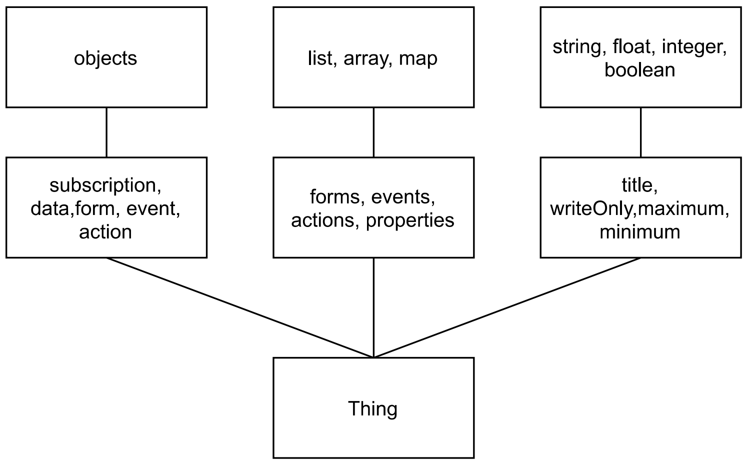

objects
list, array, map
string, float, integer, boolean
subscription, data,form, event, action
forms, events, actions, properties
title, writeOnly,maximum, minimum
Thing

Figure 4 Hierarchical representation of Thing Description Entities

(Marco and Yannis 2003) utilize the concepts of Information-Flow theory to address semantic interoperability between two different systems representing the same domain. In this work firstly, responsibilities that are primitive and common to both the systems are identified. Secondly, these identified responsibilities are mapped to the divisions within the respective systems that address them. Thirdly, these divisions are associated with groups that are composed of a set of divisions. Such a kind hierarchical approach is continued until all the entities among the respective systems are considered. Lastly, the IF logic calculations are performed to map the entities from both the systems. This paper aims to construct such hierarchical structures and then construct mapping logics between TD meta-model and AAS SM and SME.

Figure 4 presents a much simpler and abstract representation of the TD meta-model. This representation is hierarchically structured, at the top- level are the types that any of the TD entity could be, where these are simple types (like string, float, integer and boolean), collection types (like array and map) and complex types like objects. The second level of the representation indicate all the entities from the TD meta-model that are of particular type from the first level. Such a kind of hierarchical representation is similar to the one presented in (Marco and Yannis 2003). In the case of AAS meta-model a similar structure is not contemplated, rather the elements submodel, submodelElementCollection and the qualifiers are only considered for mapping with the elements from TD meta-model.

Figure 5 present mapping rules between TD and AAS. All the entities of the simple type are mapped as qualifiers from AAS meta-model, where the entity name will be the qualifier type and the entity value will be qualifier value. All the complex types including lists, classes and objects are mapped to submodelElementCollection and lastly the ThingDescription docunment or the Thing class is mapped to submodel. For example, the idempotent attribute of the ActionAffordance will be mapped as a qualifier and the action class is mapped to the submodelElementCollection. This ~~is~~ mapping rule is base-line rule for the TD-AAS model transformation proposed as part of this work.

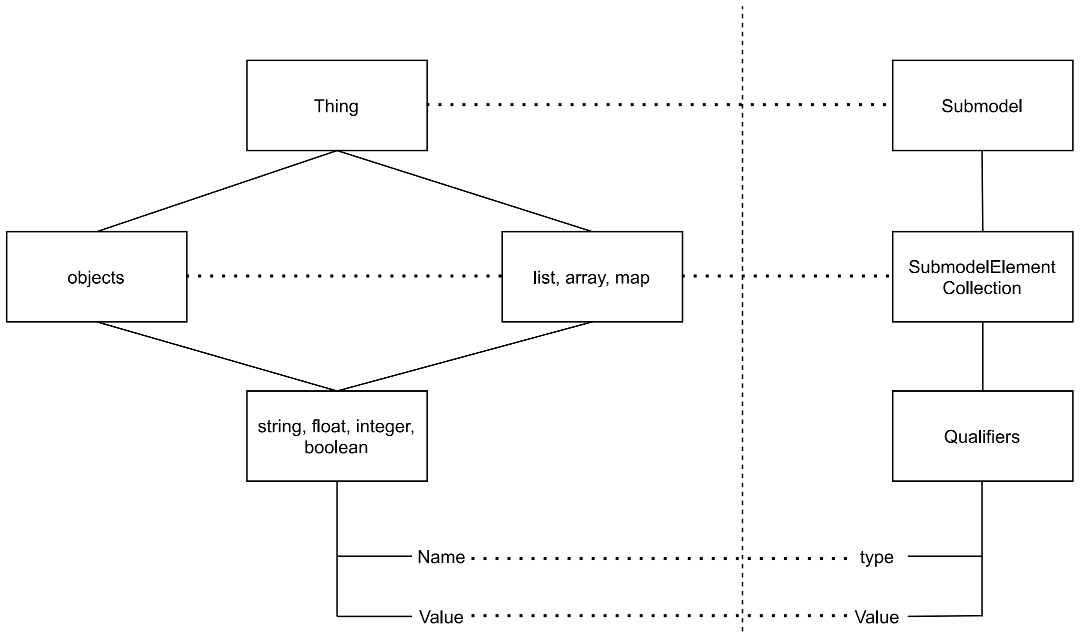
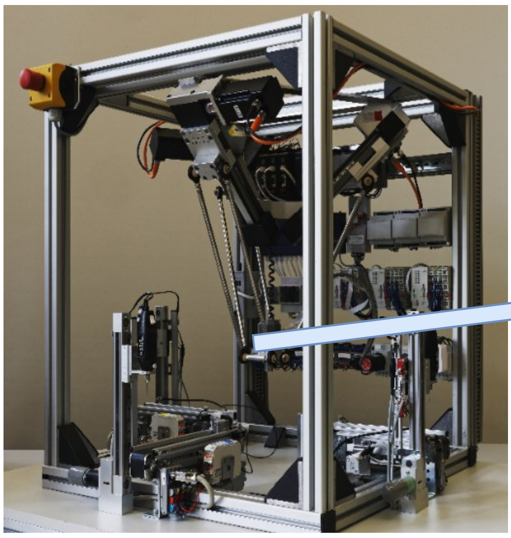


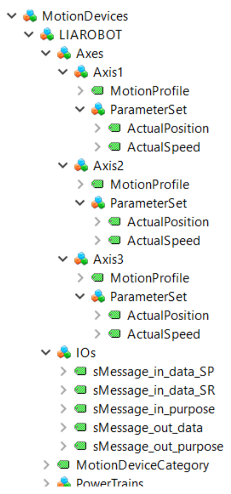
Figure 5 Mapping between Thing Description and AAS meta-models

Few variations compared to this base line mapping rule is associated with to titles, description, versionInfo and id entities. The titles entity is a list of strings, it is mapped to multilanguage property.The descriptions entity is mapped to description attribute of the corresponding submodelElementcollection. The versionInfo attribute of Thing class is mapped to the administration attribute of the submodel, where version is attached to version and model is attached to revision. Lastly the id attribute of the thing class is mapped to the id attribute of identification class related to corresponding submodel.

5. USECASE

In the context of this work, we have modelled Thing Description of the delta robot (DR) which is part of the OVGU pick and place demonstrator (Diedrich et al. 2021). The robot is accessible through an OPC UA server, which is hosted on a PLC and the actual interaction with the robot is done using this PLC. The OPC-UA server instance is modelled as per the robotics OPCUA companion specification. (OPC Foundation 2019). Figure 6 presents the picture of the OVGU demonstrator and a screenshot of the associated OPCUA server instances.





```
⌄ MotionDevices
  ⌄ LIAROBOT
    ⌄ Axes
      ⌄ Axis1
        › MotionProfile
        ⌄ ParameterSet
          › ActualPosition
          › ActualSpeed
      ⌄ Axis2
        › MotionProfile
        ⌄ ParameterSet
          › ActualPosition
          › ActualSpeed
      ⌄ Axis3
        › MotionProfile
        ⌄ ParameterSet
          › ActualPosition
          › ActualSpeed
    ⌄ IOs
      › sMessage_in_data_SP
      › sMessage_in_data_SR
      › sMessage_in_purpose
      › sMessage_out_data
      › sMessage_out_purpose
    › MotionDeviceCategory
    › PowerTrains
```

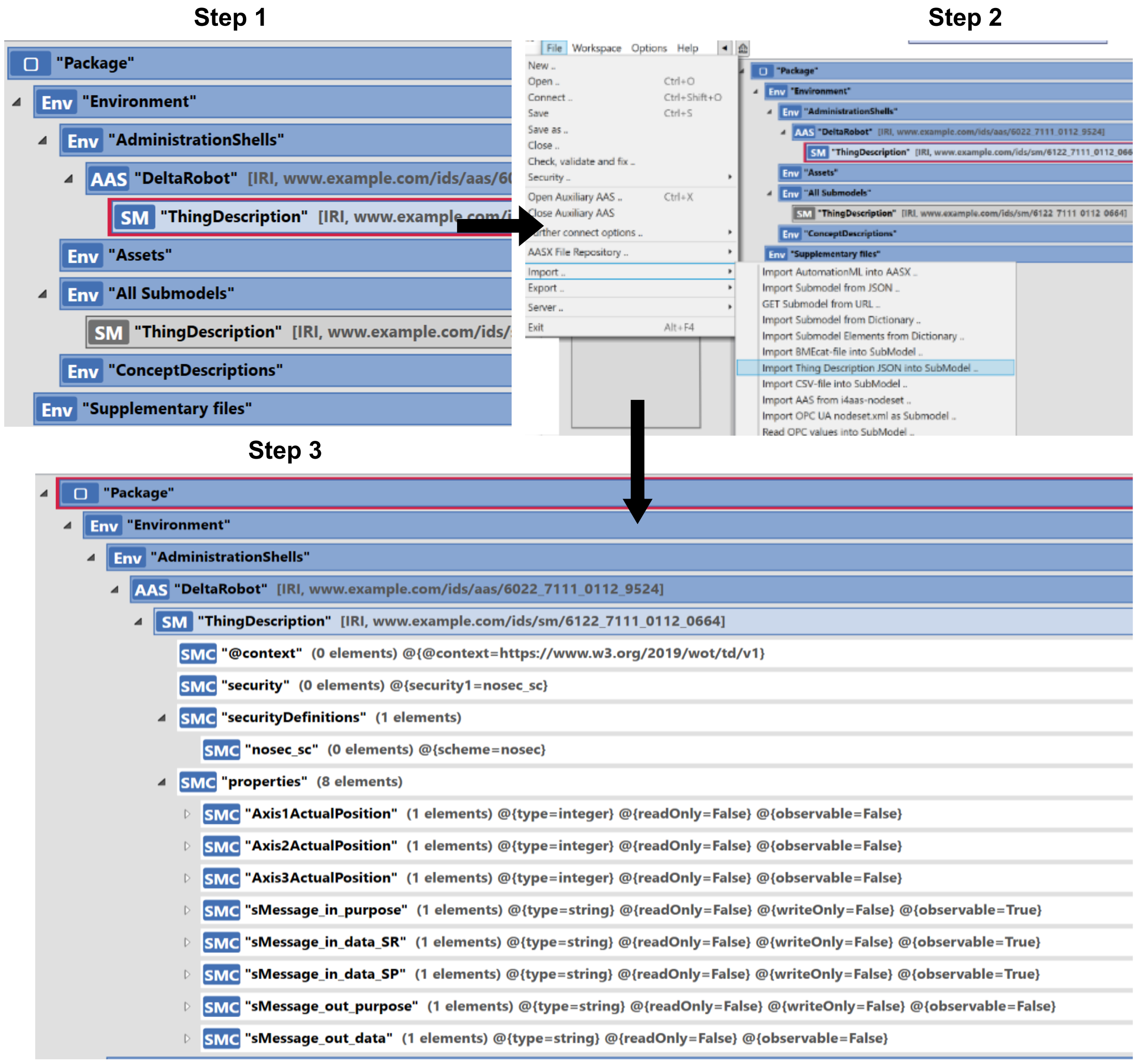
Figure 6 Snaphost of OVGU Demonstrator and Screen shot of OPCUA interface

A Thing Description JSON-LD document is created for this OPCU-UA server instance, here only the variables that are required to access the robot are modelled in the document, the figure 7 presents the screenshot of TD document. Eight variables from the OPCUA server instance are modelled as properties in the TD document and the required security definitions for access to the server are also captured.

```
{    "@context" : ["https://www.w3.org/2019/wot/td/v1"],
     "title": "LIADeltaRobot",
     "base": "opc.tcp://192.168.1.2:4840",
     "description": "The thing description of LIA delta robot.",
     "created" : "2021-09-30T13:22:05+01:00",
     "support": "mailto:kazeem.oladipupo@st.ovgu.de",
     "security": ["nosec_sc"],
     "securityDefinitions": {"nosec_sc": {"scheme": "nosec"}},
     "properties":{
                 "Axis1ActualPosition" :{
                 "Axis2ActualPosition" :{
                 "Axis3ActualPosition" :{
                 "sMessage_in_purpose" : {
                 "sMessage_in_data_SR": {
                 "sMessage_in_data_SP": {
                 "sMessage_out_purpose": {
                 "sMessage_out_data": {
                         "observable" : false,
                         "type" : "string",
                         "readOnly" : false,
                         "writeonly": false,
                         "description" : "This affordance is used to output the state of the task",
                         "forms" : [{
                             "op" : ["readproperty", "writeproperty"],
                             "href" :
                             "opc.tcp://192.168.1.2:4840/ns=4;s=|var|HX-CP1H16.Application.OPC_UA_Symbols.Motio
                             iceSystem.MotionDevices.LIAROBOT.IOs.sMessage_out_data"
                         }]
```

Figure 7 Snapshot of Thing Description for OPCUA Server Instance

As part of this work a plugin module is added to the AAS package explorer (Ristin et al. 2021), the relevant code is available under the “ovgu/ThingDescription” branch (Pakala 2021). This plugin module implements the mapping rules presented in the section 4, it consumes a Thing Description document (with an extension .json) and creates the relevant submodel. The figure 8 presents the screenshot of the package explorer and the relevant process for creation of the submodel.


Step 1
Step 2
Step 3
File
Workspace
Options
Help
New ..
Open .. Ctrl+O
Connect .. Ctrl+Shift+O
Save Ctrl+S
Save as ..
Close ..
Check, validate and fix ..
Security ..
Open Auxiliary AAS .. Ctrl+X
Close Auxiliary AAS
Further connect options ..
AASX File Repository ..
Import ..
Export ..
Server ..
Exit Alt+F4
Import AutomationML into AASX ..
Import Submodel from JSON ..
GET Submodel from URL ..
Import Submodel from Dictionary ..
Import Submodel Elements from Dictionary ..
Import BMEcat-file into SubModel ..
Import Thing Description JSON into SubModel ..
Import CSV-file into SubModel ..
Import AAS from i4aas-nodeset ..
Import OPC UA nodeset.xml as Submodel ..
Read OPC values into SubModel ..

```
"Package"
  Env "Environment"
    Env "AdministrationShells"
      AAS "DeltaRobot" [IRI, www.example.com/ids/aas/6022_7111_0112_9524]
        SM "ThingDescription" [IRI, www.example.com/ids/sm/6122_7111_0112_0664]
    Env "Assets"
    Env "All Submodels"
      SM "ThingDescription" [IRI, www.example.com/ids/sm/6122_7111_0112_0664]
    Env "ConceptDescriptions"
  Env "Supplementary files"
```

```
"Package"
  Env "Environment"
    Env "AdministrationShells"
      AAS "DeltaRobot" [IRI, www.example.com/ids/aas/6022_7111_0112_9524]
        SM "ThingDescription" [IRI, www.example.com/ids/sm/6122_7111_0112_0664]
          SMC "@context" (0 elements) @{@context=https://www.w3.org/2019/wot/td/v1}
          SMC "security" (0 elements) @{security1=nosec_sc}
          SMC "securityDefinitions" (1 elements)
            SMC "nosec_sc" (0 elements) @{scheme=nosec}
          SMC "properties" (8 elements)
            SMC "Axis1ActualPosition" (1 elements) @{type=integer} @{readOnly=False} @{observable=False}
            SMC "Axis2ActualPosition" (1 elements) @{type=integer} @{readOnly=False} @{observable=False}
            SMC "Axis3ActualPosition" (1 elements) @{type=integer} @{readOnly=False} @{observable=False}
            SMC "sMessage_in_purpose" (1 elements) @{type=string} @{readOnly=False} @{writeOnly=False} @{observable=True}
            SMC "sMessage_in_data_SR" (1 elements) @{type=string} @{readOnly=False} @{writeOnly=False} @{observable=True}
            SMC "sMessage_in_data_SP" (1 elements) @{type=string} @{readOnly=False} @{writeOnly=False} @{observable=True}
            SMC "sMessage_out_purpose" (1 elements) @{type=string} @{readOnly=False} @{writeOnly=False} @{observable=False}
            SMC "sMessage_out_data" (1 elements) @{type=string} @{readOnly=False} @{observable=False}
```

Figure 8 Screenshot of the AASX Package explorer depicting on how to import Thing Description Document

6. DISCUSSION and PROPOSALS

PI4.0 intends to provide a standardised meta-model to digitally model information about an asset. The primary or basic building block of this representation is the submodel and PI4.0 intends to structure all information about an asset like the nameplate, technical data using this submodel. The meta-model additionally provides a set of elements to capture details within the specific category of information. The PI4.0 expects that research and industry bodies come up with standardized submodel templates using the available meta-model elements. It would be easier for manufacturers, suppliers, vendors and other interested partners to utilize these templates and properly organize the information about their assets. This work is such an attempt to create a submodel template for representing access related information of an asset.

The WoT Thing Description (TD) is an ~~better~~ organized and well-represented meta-model used to structure the asset access related information. In this work, instead of working from scratch, we utilize the constructs from the TD meta-model and have created a new submodel template. This process is done by creating a mapping between the two meta-models, thereby custom mapping rules are introduced. Even though the AAS meta-model offers elements like property, reference element, range, MultiLanguage property, operation, basic event and collection only the collection element is used for representing complex entities. The qualifier attribute associated with each of these elements is used to represent the simple types from the TD.

Elements like property, range and operation can be well utilized to represent certain entities from the TD. The property element can be used to represent the simple types and probably operations can be mapped to forms. The main reason, to avoid such mapping rules is to avoid a huge hierarchical structure within the AASX representation and also visual complexity in the package explorer. That is the design and capabilities of the explorer has greatly influenced this work. During the time of this work, the only available modelling tool for the AAS is the package explorer (Marko et al.). Another reason for such a consideration is to avoid complex mapping rules and have an underlying base rule for all the components. A note on the package explorer, operation element can accept any other submodel element as input and output parameters, but the explorer restricts it to property element.

The property element from the AAS meta-model has the feasibility to specify only a single value of a specific type like integer, boolean, string etc. In case this element is re-modelled to represent multiple values, it would greatly influence the mapping rules presented in this paper. An additional suggestion towards the modification of the AAS meta-model would be to add a DateTime class that has the attributes create datetime, modify datetime, last modified and all the elements of the meta-model inherit this class.

7. CONCLUSION

In this paper, we have introduced a submodel template that aims to represent access related information of an asset. For this the WoT TD meta-model is used as a reference. During the course of the research work we have understood that the AAS some online related assets meta-model elements needs to be evolved, existing submodel elements needs to be upgraded and further elements need to be introduced.

8. ACKNOWLEDGEMENTS
The research was carried out in the project “VWS Vernetzt / AAS Networked” under support of the Federal Ministry for Economic Affairs and Energy, FKZ: 01QE1544B.

Publication bibliography

Agostinho, Carlos; Correia, Filipe; Jardim-Goncalves, Ricardo (2010): Interoperability of Complex Business Networks by Language Independent Information Models. In Jerzy Pokojski, Shuichi Fukuda, Józef Salwiński (Eds.): New World Situation: New Directions in Concurrent Engineering. London: Springer London (Advanced Concurrent Engineering), pp. 111–124.

Alur, Rajeev (2015): Principles of cyber-physical systems. Cambridge, Massachusetts, London, England: The MIT Press.

Cavalieri, Salvatore; Mule, Salvatore; Salafia, Marco Giuseppe (2019): OPC UA-based Asset Administration Shell. In : IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society: IEEE.

Diedrich, C.; Belyaev, A.; Schröder, T.; Urban, C.; Werner, T.; Pakala, H. (2021): Modell einer Pick & Place-Anlage basierend auf Verwaltungsschalen. In : Automation 2021: VDI Verlag, pp. 65–74.

Drath, Rainer; Luder, Arndt; Peschke, Jorn; Hundt, Lorenz: AutomationML - the glue for seamless automation engineering. In : 2008 IEEE International Conference on Emerging Technologies and Factory Automation. Factory Automation (ETFA 2008). Hamburg, Germany: IEEE, pp. 616–623.

Henning Kagermann (2013): Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Forschungsunion, acatech.

Henßen, Robert; Schleipen, Miriam (2014): Interoperability between OPC UA and AutomationML. In *Procedia CIRP* 25, pp. 297–304. DOI: 10.1016/j.procir.2014.10.042.

Kaebisch, Sebastian; Kamiya, Takuki; McCool, Michael; Charpenay, Victor (Eds.) (2021): Web of Things (WoT) Thing Description. Available online at https://w3c.github.io/wot-thing-description/, checked on 10/10/2021.

Kalnins, Audris; Barzdins, Janis; Celms, Edgars (2004): Model Transformation Language MOLA. In : Model Driven Architecture: Springer, Berlin, Heidelberg, pp. 62–76. Available online at https://link.springer.com/chapter/10.1007/11538097_5.

Lelde, LACE; Audris, KALNINS; Agris, SOSTAKS (2015): Process DSL Transformation by Mappings Using Virtual Functional Views. Available online at https://www.bjmc.lu.lv/fileadmin/user_upload/lu_portal/projekti/bjmc/contents/3_2_4_lace.pdf.

Marco, Schorlemmer; Yannis, Kalfoglou (2003): Using information-flow theory to enable semantic interoperability. Available online at https://www.icsa.inf.ed.ac.uk/publications/online/0161.pdf.

McCool, Michael; Kaebisch, Sebastian (Eds.) (2021): W3C Web of Things. Available online at https://www.w3.org/WoT/, updated on 10/10/2021.

Miny, Torben; Thies, Michael; Epple, Ulrich; Diedrich, Christian (2020): Model Transformation for Asset Administration Shells. In : IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society: IEEE.

OPC Foundation (Ed.) (2019): Robotics OPCUA Companion Specification. Available online at https://reference.opcfoundation.org/Robotics/, checked on 10/11/2021.

Overbeek, J. F. (2006): Meta Object Facility (MOF): investigation of the state of the art. Available online at http://essay.utwente.nl/57286/.

ITU-T Y.4000, 6/15/2012: Overview of the Internet of things.

Pakala, Harish Kumar (Ed.) (2021): AASX Package Explorer Thing Description Plugin. Available online at https://github.com/admin-shell-io/aasx-package-explorer/tree/ovgu/ThingDescription, checked on 10/11/2021.

Platform Industry 4.0: Details of the Asset Administration Shell. Part 1 - The exchange of information between partners in the value chain of Industrie 4.0 (Version 3.0RC01). In.

Ristin, Marko; Orzelski, Andreas; Hoffmeister, Michael (Eds.) (2021): AASX Package Explorer. Available online at https://github.com/admin-shell-io/aasx-package-explorer, checked on 10/11/2021.

Stefan-Helmut Leitner, Wolfgang Mahnke (2006): OPC UA–service-oriented architecture for industrial applications. Available online at http://cimug.ucaiug.org/kb/knowledge%20base/soa%20for%20industrial%20applications.pdf.