Usage of Standards in the Smart Factory Web Testbed

An Industrial Internet Consortium White Paper

Version 1.0

2020-06-29

CONTENT

FIGURES

TABLES

Smart Factory Web (SFW) is a platform that connects smart factories through a manufacturing marketplace to enable flexible usage of assets, such as in a supply chain. Factories can register with SFW by describing their manufacturing capabilities, for example in terms of the products they produce or in terms of the processes they offer. Potential business partners such as other factories can search the SFW for factories with the desired capabilities. The business partner can then place an order on the selected factory, say Factory A; the order includes the precise specifications of customized products or services. Factory A may have to adapt or configure its production facilities to meet the order. The actual manufacturing process can be monitored; upon completion the product is delivered to the recipient agreed with the business partner.

Factories need to adapt and respond to orders as efficiently as possible, especially for small lot sizes. This implies minimal manual configuration, which can only be achieved in a sustainable way by using widely accepted standards. SFW requires standards in the following areas:

- communication between factories and the SFW,
- communication between factories and other cloud platforms for monitoring of production steps,
- description of factory capabilities and
- communication between factories to exchange product, order and capability information.

This document describes the usage of the principal pivotal standards in the Industrial Internet Consortium (IIC) Smart Factory Web Testbed. For further information about the testbed itself, see *hub.iiconsortium.org/smart-factory-web* and *https://www.smartfactoryweb.de/*.

We provide guidance in the deployment of the three main SFW standards, OPC Unified Architecture (OPC UA), AutomationML and OGC SensorThings API and aim to convey how key concepts of factory and manufacturing information can be modeled and implemented in the framework of these standards. We describe a functional view but do not specify the concrete implementation.

References are made to the asset administration shell of Platform Industrie 4.0 as these concepts will be applied in future SFW versions.

This document is organized as follows:

- Chapter 2: Standards Overview gives a brief overview of the standards relevant for SFW
- Chapter 3: Smart Factory Web Architecture explains how assets are modeled in SFW and are integrated into the cloud
- Chapter 4: Detailed description of standards usage
- Annex A: Overview of OPC UA
- Annex B: Overview of OGC SensorThings API
- Annex C: Assets in I4.0 as background information concerning on-going SFW work with the Asset Administration Shell of I4.0
- Annex D: Revision History
- Annex E: Acronyms
- Annex F: References

1 STANDARDS OVERVIEW

The standards below are essential for the core functionality of the Smart Factory Web Testbed.

Standard	Reference	SDO
OPC UA	IEC 62541	IEC SC 65E
AutomationML	IEC 62714	IEC SC 65E; AutomationML association
Companion Specification "OPC Unified Architecture for AutomationML"		OPC Foundation and AutomationML association
Combining OPC Unified Architecture and Automation Markup Language	DIN SPEC 16592	DIN
SensorThings API, Part 1: Sensing	OGC 15-078r6	Open Geospatial Consortium (OGC)
RM-SA - Reference Model for Industrie 4.0 Service architectures	DIN SPEC 16593-1	DIN

Table 1: Main standards in the Smart Factory Web Testbed

OPC UA: The multipart standard defines online data exchange between automation systems. The OPC Unified Architecture (OPC UA) information model is a connected graph with Nodes, attributes of Nodes and References (directed edges) between Nodes. The object-oriented modeling paradigm is applied to objects, variables, data types and references. Objects typically represent system components (hardware or software) and are structured hierarchically. The information model of an OPC UA server is divided into groups of Nodes called NodeSets. More information is given in Annex A.

AutomationML: The standard addresses data exchange in the engineering process of production systems, such as between software tools or to configure software systems. AutomationML defines a hierarchy of instances and system unit classes supplemented by role class definitions and interface classes.

OGC SensorThings API: The information model of OGC SensorThings API is based on the ISO standard observation and measurement (O&M) model. The central concepts are Datastream, Observation, ObservedProperty, Sensor and Thing. A Datastream groups a collection of Observations measuring the same phenomenon (called the ObservedProperty) and produced by the same Sensor for the same Thing. The standard defines a RESTful interface for clients to access a SensorThings API server containing the Datastreams and related metadata. See Annex B for additional information.

Reference Model for Industrie 4.0 Service architectures: Reference models shall provide guidance to software and system engineers when defining system (of systems) architectures [Usl 2010]. A reference model encompassing the "big picture" of the system architecture of Industrie 4.0 is defined in DIN SPEC 91345, Reference Architecture Model Industrie 4.0 (RAMI4.0) [DIN 2016a].

Service-orientation and service-oriented architecture (SOA) are mentioned several times in the description of the RAMI4.0 layers as one of the technological cornerstones of the I4.0 vision.

According to RAMI4.0, the "fundamental purpose of Industrie 4.0 is to facilitate cooperation and collaboration between technical objects" (synonymously called assets). RAMI4.0 considers assets of various types, ranging from field devices, control devices, work centers up to enterprises, but supplemented by the asset types product and connected world to enable new value chains. However, to enable the Industrie 4.0 vision of enhanced flexibility and adaptability, the hierarchical order scheme is relaxed on the abstraction level of the I4.0 Components. The ordering scheme has to be generalized and finally turns into a networked scheme of I4.0 Components. This has consequences for the concepts of the resulting IT system architecture.

In classical architectures following the automation pyramid, the information flows and roles of the communication participants are fixed to a large extent. The pre-defined architecture naturally leads to fixed communication schemes, for example, client/server or publisher/subscriber design patterns. Although these patterns can be combined to realize more complex scenarios, role changes and peer-to-peer schemes are difficult and inefficient to implement.

The Reference Model for Industrie 4.0 Service Architectures (RM-SA), motivated in [UslEpp 2015] and published as DIN Specification 16593-1 in April 2018 [DIN 2018], shall allow the system engineers to design flexible system architectures based upon the RAMI4.0 concept of an I4.0 component and the multi-style SOA paradigm by allowing multiple interaction patterns.

The RM-SA relies upon the conceptual service model provided by [OASIS 2016], stating that "a service is a mechanism to enable access to one or more capabilities where the access is provided by a prescribed interface and is exercised consistent with constraints and policies as specified by the service description."

The RM-SA distinguishes between three basic service types (Figure 1):

- infrastructure services (or platform services),
- interaction-based services whose behavior is described by means of procedures and
- Interaction-based services whose behavior is described by means of state machines.

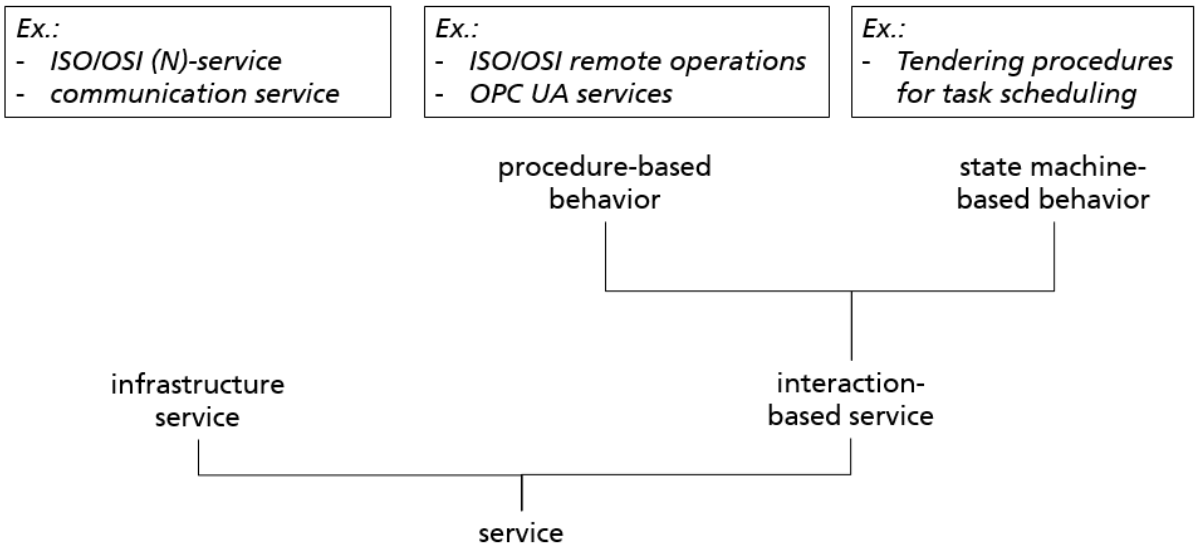

Figure 1: Basic classification of service types

2 SMART FACTORY WEB ARCHITECTURE

The SFW is a platform that connects smart factories over a network to enable flexible sharing and management of resources, assets and inventory to maximize production and efficiency. To become part of the Smart Factory Web, network participants describe their products and factory capabilities so as to improve factory-to-factory collaboration. A Smart Factory Web Portal (SFWP) enables secure data and service integration in cross-site application scenarios as well as 'plug & work' functions for devices, machines and data analytics software by applying industrial standards, in particular Open Platform Communications Unified Architecture (OPC UA) and Automation Markup Language (AutomationML or AML); see Figure 2.

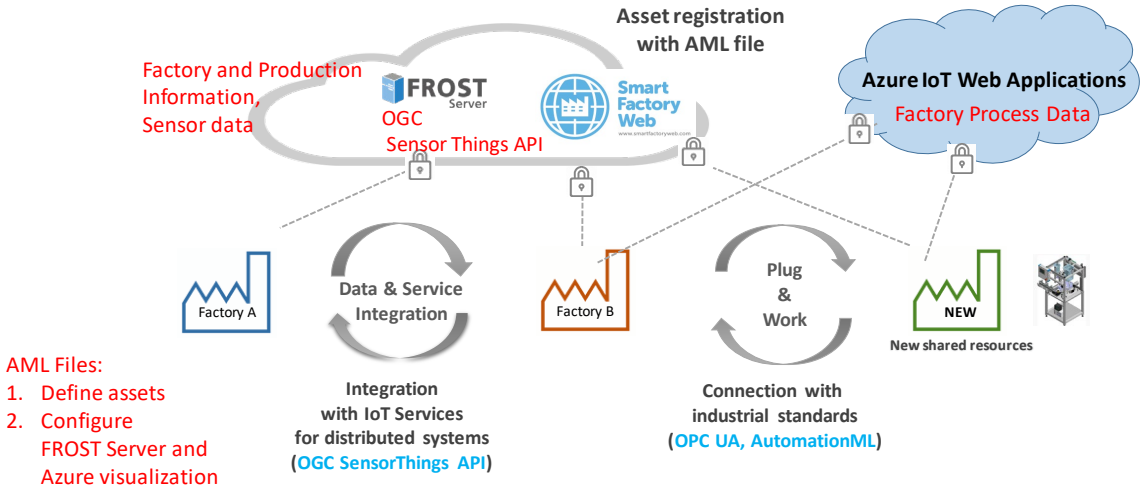

Figure 2: Connections between the factories to the Smart Factory Web Portal and the Azure IoT Cloud

2.1 ASSET DEFINITION

Asset is defined in [IIC 2019b] as a "major application, general support system, high impact program, physical plant, mission critical system, personnel, equipment or a logically related group of systems."

In the context of this report, an Asset is a factory or part of the factory equipment hierarchy as defined in [IEC 2013]. This hierarchy covers from top to bottom: Enterprise, Site, Area, Work Center and Work Unit. Such assets are evidently also Assets according to the wider I4.0 Asset definition (cf. Annex C). Assets are structured in a hierarchy with the relation isSubassetOf. In SFW, factories are top-level assets of type Area representing a physical and geographical grouping of production buildings and their subdivisions, such as a structure or hall.

The ontology used by SFW is shown in Figure 3 with the concepts Asset, Capability, Property and Unit_Of_Measure. The concept Capability is a capability of the Asset, the concept Property represents the properties that can be associated with a capability and the concept Unit of Measure is defined as the unit of measure for values of properties. Properties may have a value or a range of values between MinValue and MaxValue. The concepts Asset and Capability can be composed of sub-Assets and sub-Capabilities respectively.

Capabilities may be structured and further refined with the relation isComposedOf in the ontology in Figure 3. A small example is shown in Figure 4. The Capability sub-classes can be created to match the terminology in common use in a particular sector.

In the current version of SFW, a Capability relates directly to an Asset. However, it is planned to implement the I4.0 approach by defining an Asset Administration Shell (AAS) for each asset and then considering capabilities of the resulting I4.0 Component (the logical combination of the Asset and its Administration Shell). [I4.0 2019] defines a Capability of an I4.0 Component to be "the implementation-independent description of the potential of a resource to achieve a certain effect in the physical or virtual world". There is on-going work to define the derived concept of a skill: "implementation of a capability to realize the effect in the physical or virtual world".

In SFW, a Capability is described in terms of what (product) an asset can produce, or in terms of how the product can be produced. For example, the capability of an asset could be "produce car door for car model 123" (i.e. what) or it could be "cut shape out of flat metal sheet with laser" (i.e. how). In both cases, further property information is needed in general to sufficiently define the capability, for example, the thickness and the type of the metal.

In this report, Properties describe the characteristics of Capabilities.

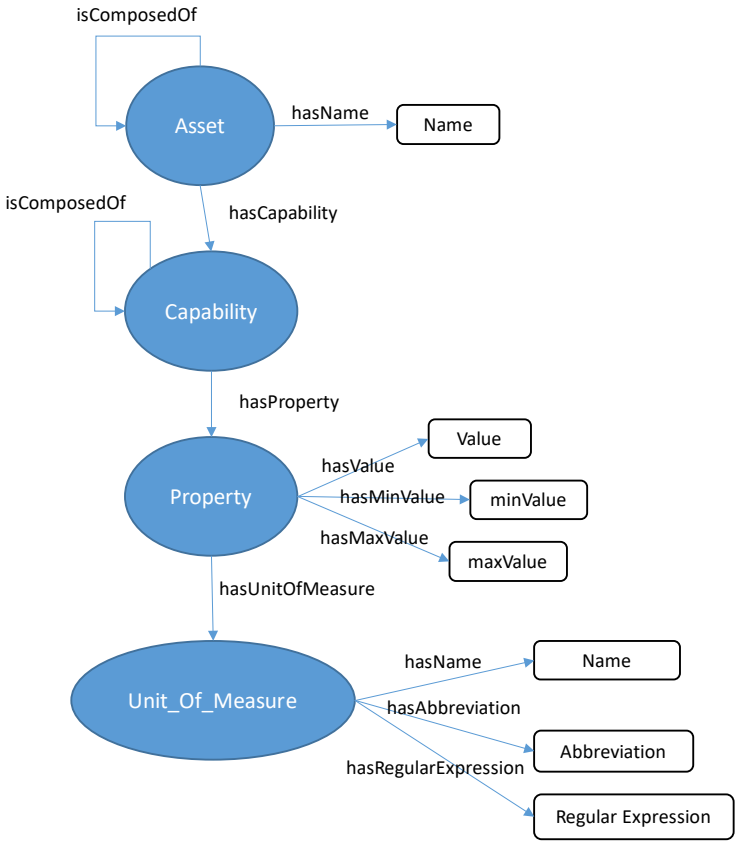

Figure 3: Fundamental ontology of Smart Factory Web

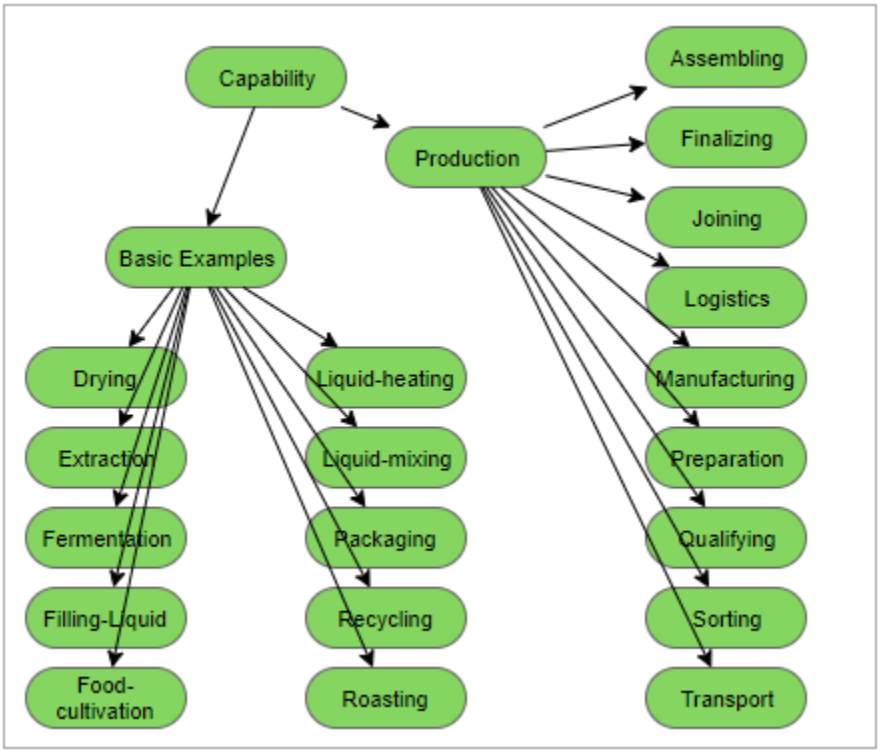

Figure 4: Example sub-classes of capability

2.1.1 EXTENSION TO SUPPLY CHAINS

The SFW ontology of Figure 3 has been extended to include concepts for products and supply chains as presented in Figure 5. This extension is applied in the IIC Negotiation Automation Platform Testbed.

Products may comprise partial products described using the relation isSubProductOf. The relation hasProductSupplier assigns products to their suppliers.

The concepts SupplyChain and SupplyChainElement are required for describing product-specific supply chains and for searching and selecting suitable supply chains. The SupplyChainElement concept allows the specification of product data related to a specific supply chain. Product data can include information on production, the manufacturing company, the sub-products and compliance with possible standards (e.g. sustainability requirements). The relation assignedToProduct enables the assignment of a supply chain element to a product.

Each product has a SupplyChainElement. The SupplyChainElement concept has a reflexive relation (isPredecessorOf) used to link the individual SupplyChainElements with each other. The SupplyChain concept serves to aggregate metadata of supply networks and thus accelerate the search query across multiple supply chains.

The Order concept allows the description of orders (of products) and their division into suborders (relation: isSubOrderOf).

Individual products can be manufactured in one or more production steps as described with the Production-Step concept. Production steps can in turn have upstream and downstream production steps (relations: isPredecessorOf and isSuccessorOf).

Transport steps between assets (e.g. factories) can be described using the Transport-Step concept. Individual transport steps can in turn have upstream and downstream transport steps.

The location concept describes the geographic coordinates of an Asset's location. The relation isLocatedIn allows the assignment to a specific country (concept: Country), which in turn allows a description of the location. Using the country concept, it is possible to specify country-specific customs regulations and trade agreements.

The risk concept describes the risks and serves to create individual risk instances. Risk instances can be divided into risk categories (concept: RiskType). Risk categories include natural disasters, transport risks, political risks and epidemics.

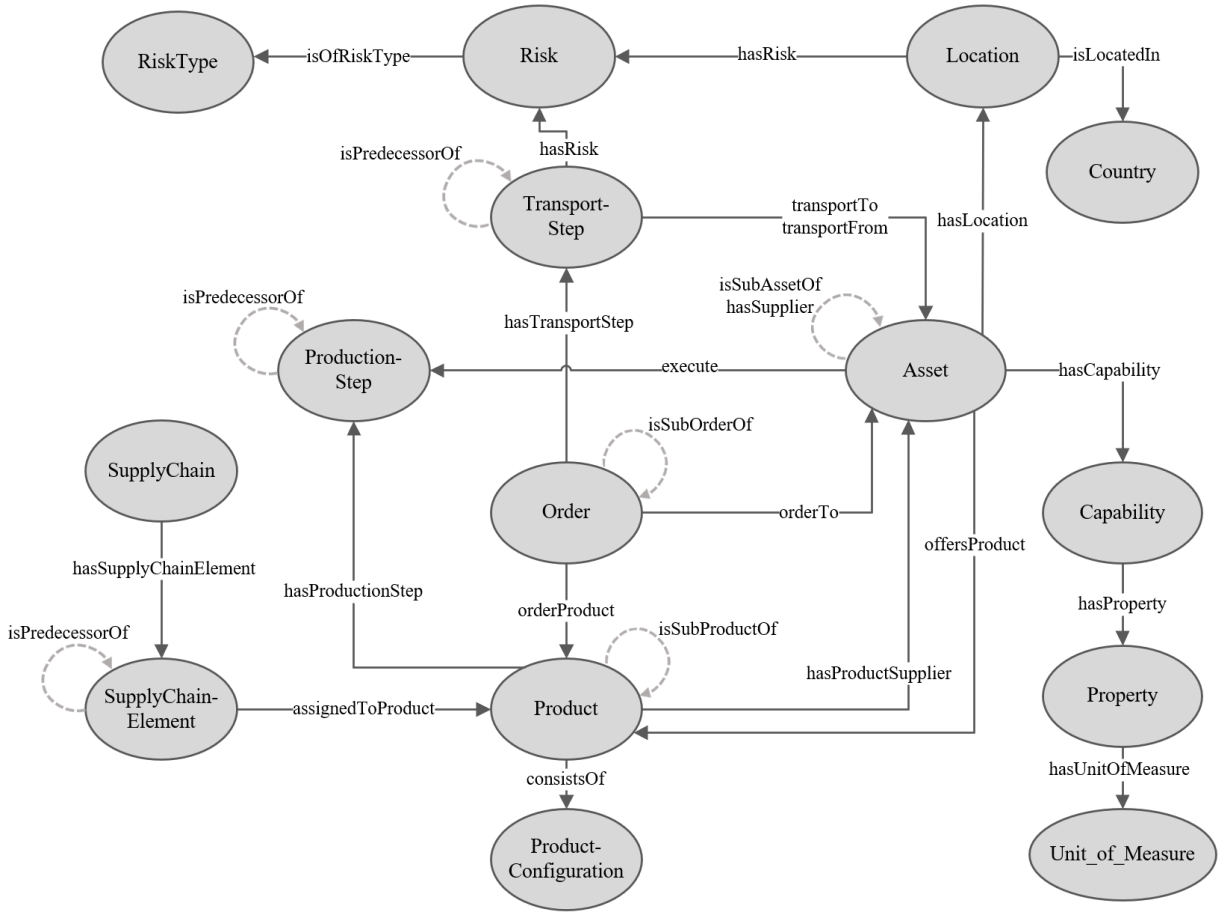

Figure 5: Extended ontology of Smart Factory Web including concepts for products and supply chains

2.2 Describing a Factory in AML

AutomationML (AML) is defined in the multipart standard IEC 62714 [IEC 2018, AML 2018]. It is an XML-based format to exchange engineering data of production systems (e.g. between software tools or to configure software systems). Figure 6 shows the AML structure; for further explanations, refer to [AML 2018, Part 1]. The plant structure ('topology') made up of physical and logical components is described with the standard CAEX IEC 62424 [IEC 2016]; this standard supports links to other standards such as COLLADA[1] (with geometry and kinematics information) and PLCopenXML (with logic information) [IEC 2019]. The plant structure, including communication systems is expressed as a hierarchy of AML objects following CAEX IEC 62424. The AML model also describes the relations between AML objects and references to information stored in documents outside the top-level format, for example to eCl@ss.[2]

[1] *https://www.khronos.org/collada/*
[2] *https://www.eclass.eu/en*

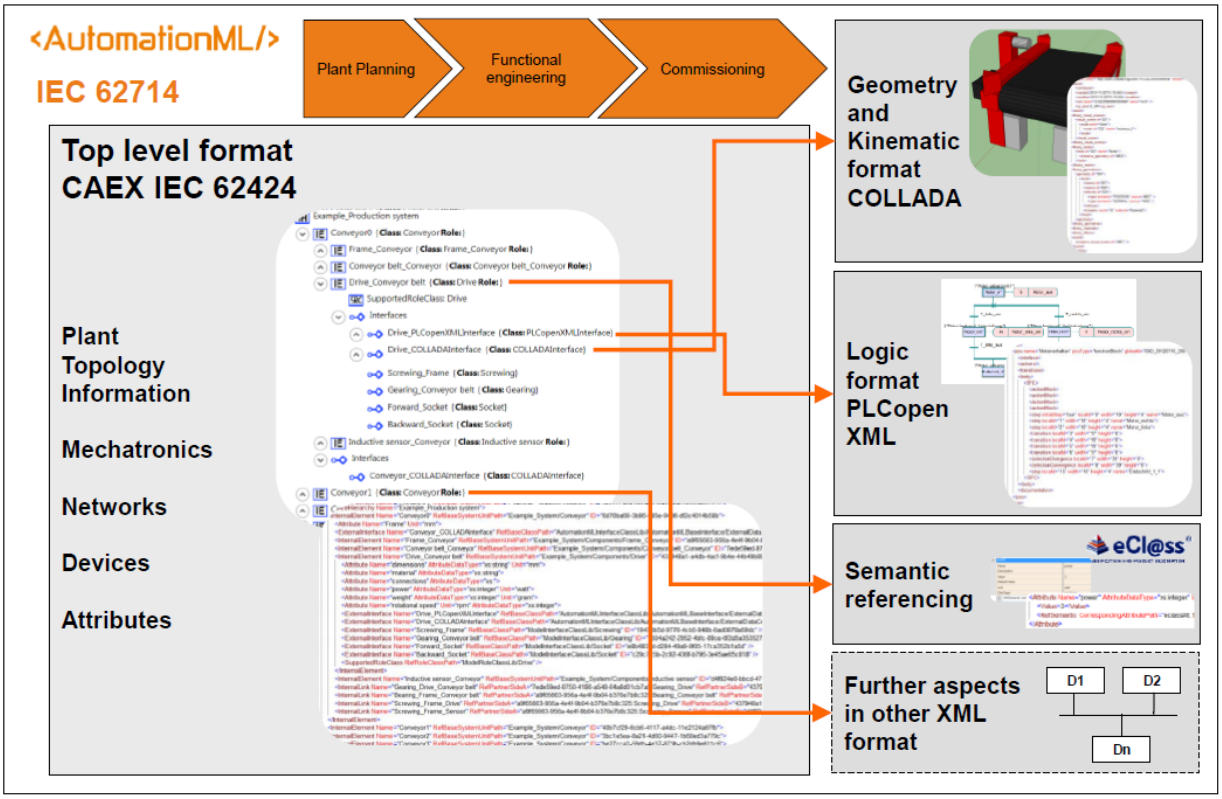

Figure 6: AutomationML structure. Source: [OPC 2016]

An AML model comprises the following hierarchical trees:

- InstanceHierarchy: represents a component in the production system or plant; structured into a hierarchy of InternalElements. An InternalElement represents a real or logical object in a system (plant, factory, production line, equipment, etc.) with its attributes and interfaces.
- RoleClassLib: a hierarchy of RoleClass elements describing abstract functionality and semantics of an InternalElement or a SystemUnitClass. RoleClass libraries are defined for discrete manufacturing industry, continuous manufacturing industry, batch manufacturing industry and control systems.
- SystemUnitClassLib: a hierarchy of SystemUnitClass elements representing classes (reusable templates) of InternalElements.
- InterfaceClassLib: a hierarchy of InterfaceClass elements representing connections between elements or with externally modeled information. The interface may be a data or signal interface, mechanical or electrical interface.

The factory AML file shall contain the relevant asset structure of the factory, its capabilities and optional information about variables accessible from OPC UA servers on the shop floor.

Figure 7 shows the AML instance hierarchy of an example factory. It contains two factories (Big factory and Sweet logistics), several production lines and some stations. The InternalElements

(IE) MakeCandySticks, CoolingCapability, PopcornCapability and ContainerTransport have capabilities modeled as a RoleRequirement (RR) defined in the CapabilityRoleClassLib.

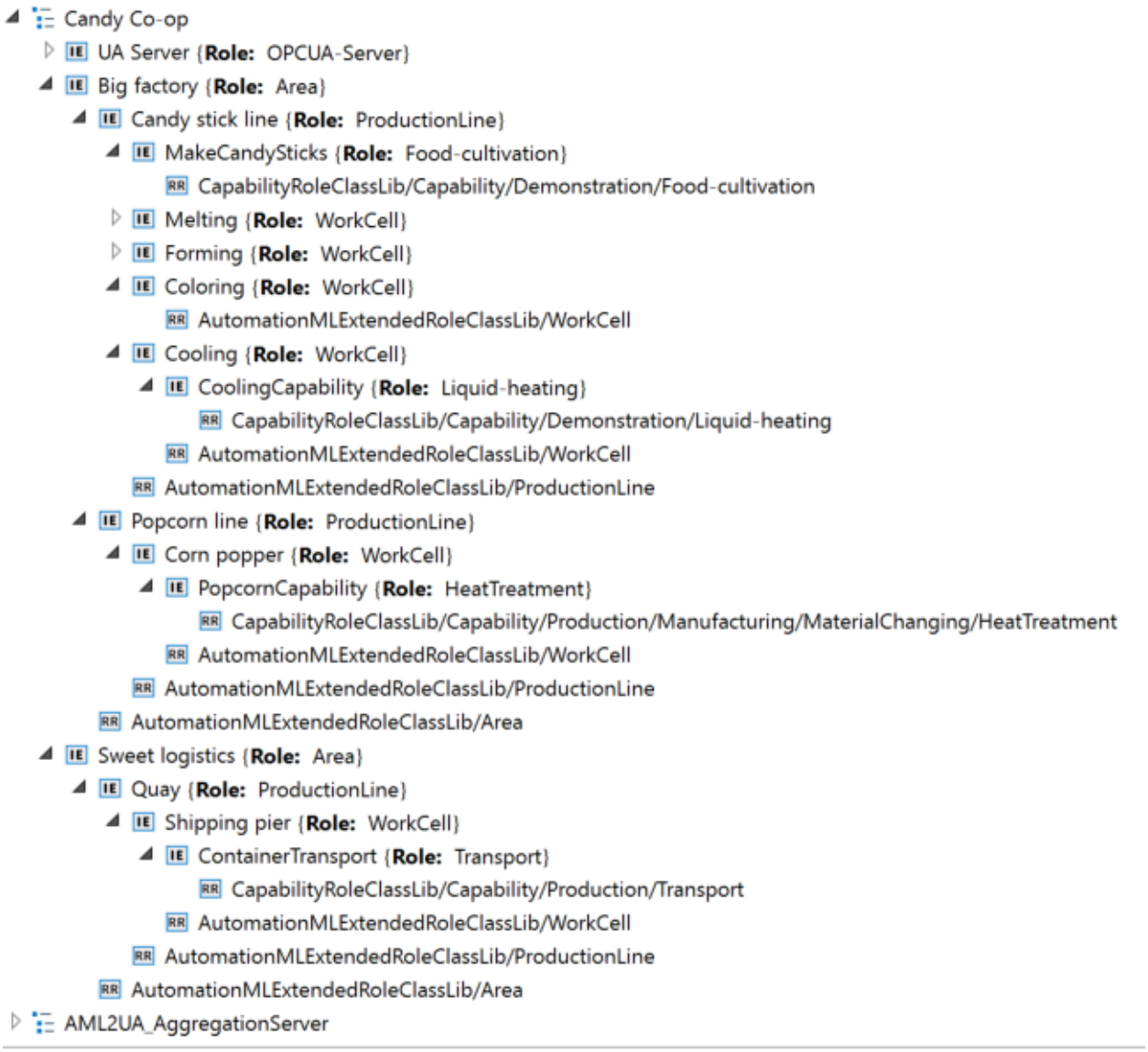

Figure 7: Structure of the AutomationML example "CandyFactory"

Figure 8 shows a part of the CapabilityRoleClassLib in the AML file. [DIN 2003] is the basis for these capabilities.

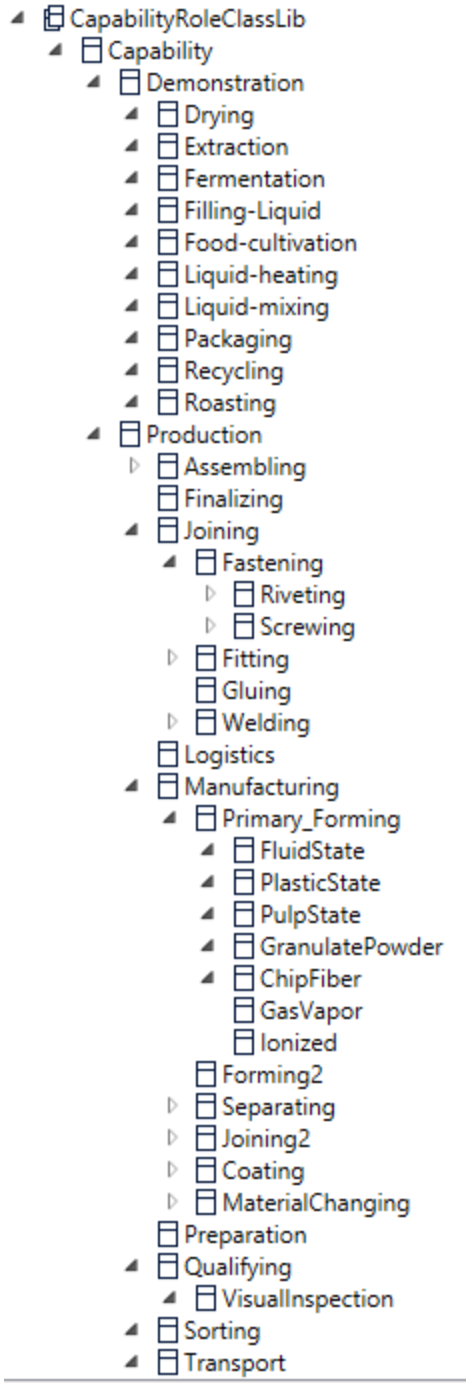

Figure 8: AutomationML role class lib for capabilities

The SFW provides an upload functionality for AML files to import a new factory. In addition, all factory descriptions in the SFW have an update functionality to update the correspondent factory information with another AML file. The request and response details are shown in Figure 9. The SFW sends the AML file and information of the existing assets and capabilities to the AMLExtractor WebService. The response contains the new or changed assets and capabilities.

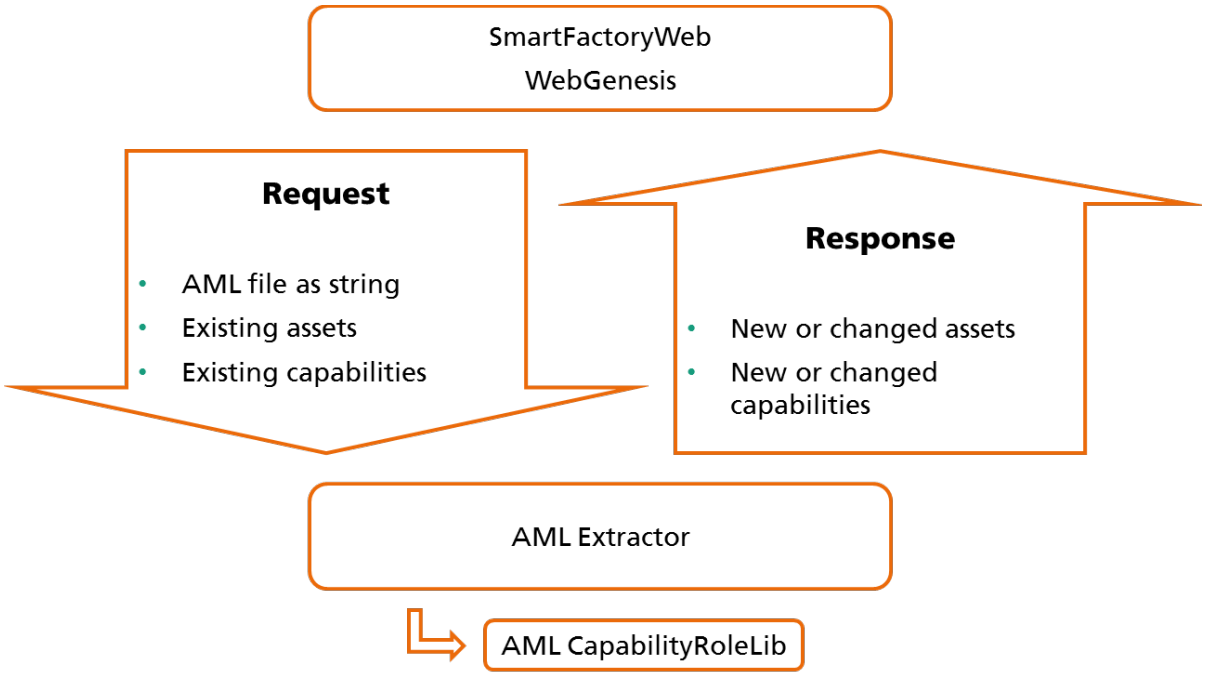

Figure 9: AMLExtractor request and response.

The AMLExtractor transforms the AML elements to the following hierarchical structure:

- list of CapabilitiesConcepts
  - list of Properties
  - list of CapabilitiesConcepts
- list of Assets
  - list of Capabilities
    - list of Properties
    - list of Capabilities
  - list of UANodes (Nodes in OPC UA; cf. 3.2.1 and Annex A)
  - list of Assets

Here you see the detailed mapping of AML elements to SFW objects:

- CapabilityConcept
  - RoleClass inherited from role "CapabilityRoleClassLib/Capability" and used by any InternalElement
  - mapped information:
    - Name, reference string
- Property
  - attribute according to a Capability RoleClass
  - mapped information:
    - Name, Value, Unit
    - MinValue, MaxValue from the first OrdinalScaledConstraint

- Asset
  - InternalElement referring to "AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Resource" or "AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Structure/ResourceStructure" or is parent of a Capability.
  - mapped information:
    - Name, ID, Description
- Capability
  - InternalElement referring to a role inherited from "CapabilityRoleClassLib/Capability"
  - mapped information:
    - Name and ID
- UANode
  - DataVariable of type OPCUA
  - mapped information:
    - Names of parents, DataType, NodeId, DiscoveryUrl of the server

The first lines of the assets' descriptions of the CandyFactory example are shown in Figure 10. The asset "Big factory" has a sub-asset "Candy Stick line" with capability "MakeCandySticks" and property "Mass".


```
{
    "Assets": [
        {
            "Name": "Big factory",
            "Description": "The fantastic candy factory near Zoological Garden in Karlsruhe.",
            "WgId": "",
            "AMLId": "27eb6276-03dd-4b9b-a0be-dccf2d2ffc05",
            "Changed": false,
            "UANodes": [],
            "Capabilities": [],
            "Assets": [
                {
                    "Name": "Candy stick line",
                    "Description": "",
                    "WgId": "",
                    "AMLId": "7a80d115-cf6e-44d9-bd93-cd9385fbad5c",
                    "Changed": false,
                    "UANodes": [],
                    "Capabilities": [
                        {
                         "Name": "MakeCandySticks",
                         "AMLId": "CapabilityRoleClassLib\/Capability\/Demonstration\/Food-cultivation",
                         "WgId": "",
                         "WgConceptId": "68297",
                         "Changed": false,
                         "Properties": [
                                {
                                    "Changed": false,
                                    "ConceptName": "Mass",
…}
```


Figure 10: Part of the Asset description for the Candy Factory

2.3 THE ASSET GRAPH VIEW

The structure of the assets connected can be visualized as a graph. This network of assets contains different data objects with metadata displayed in the visualization. The display of data objects can be filtered by concepts and relations.

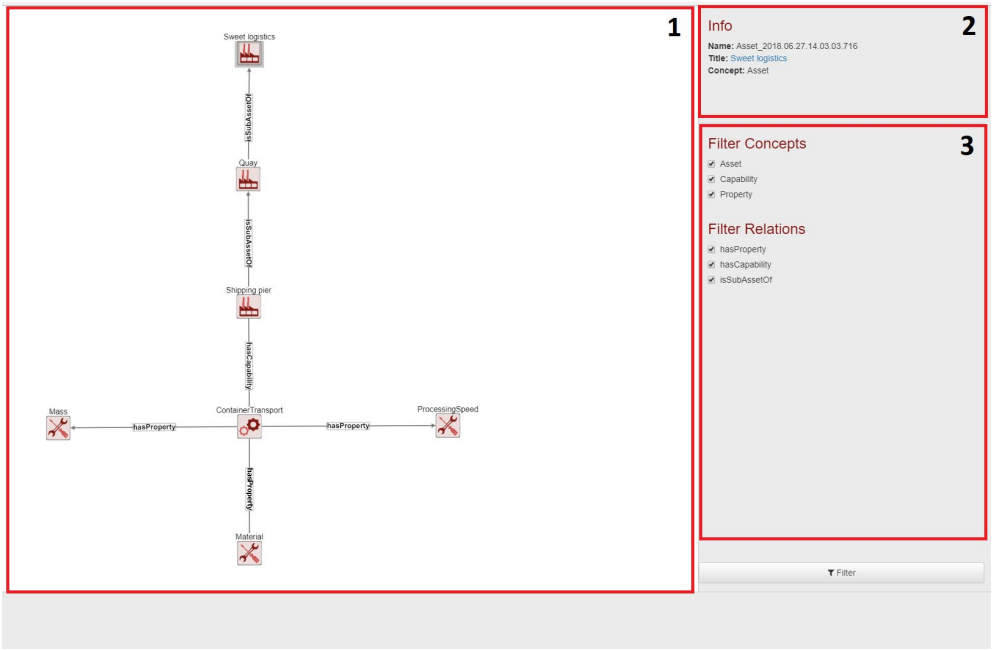

Figure 11 Asset Graph View

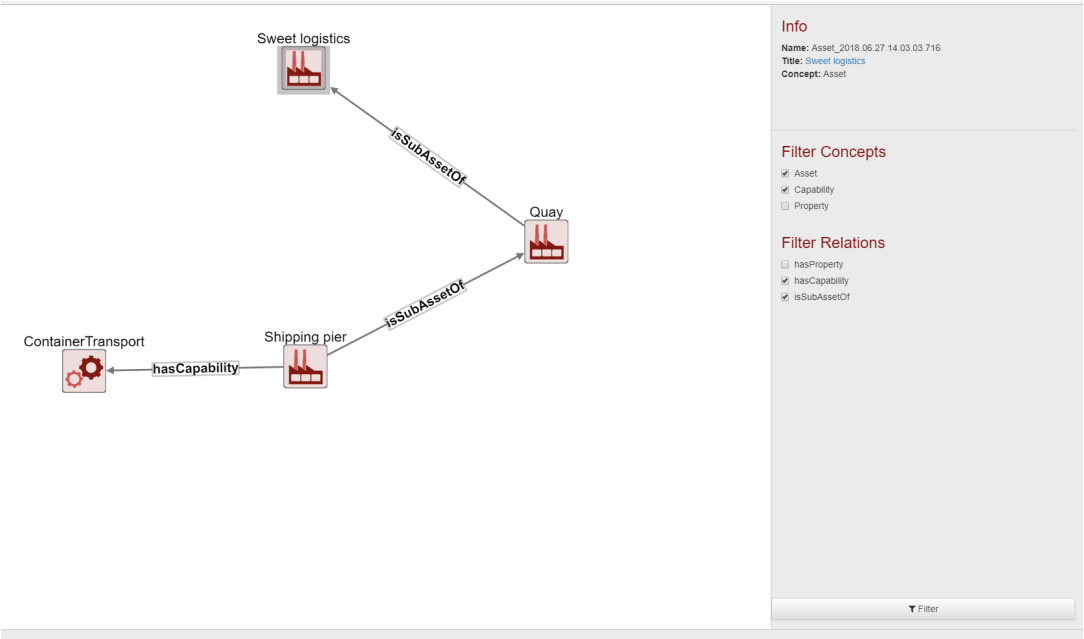

Figure 12 Filter objects to omit properties

2.4 FACTORY REGISTRATION

A new factory is registered on SFW by a form with the following meta-data:

- factory name
- short and full descriptions of the factory (general information)
- geo-location on a map as a point or polygon. The UI supports selection and display of the location on a map (the standard setting is OpenStreetMap).
- capabilities and their properties as selected with a user interface to the SFW ontology or alternatively as extracted from an imported AML file (see 2.2).
- link to parent asset
- access rights to the asset

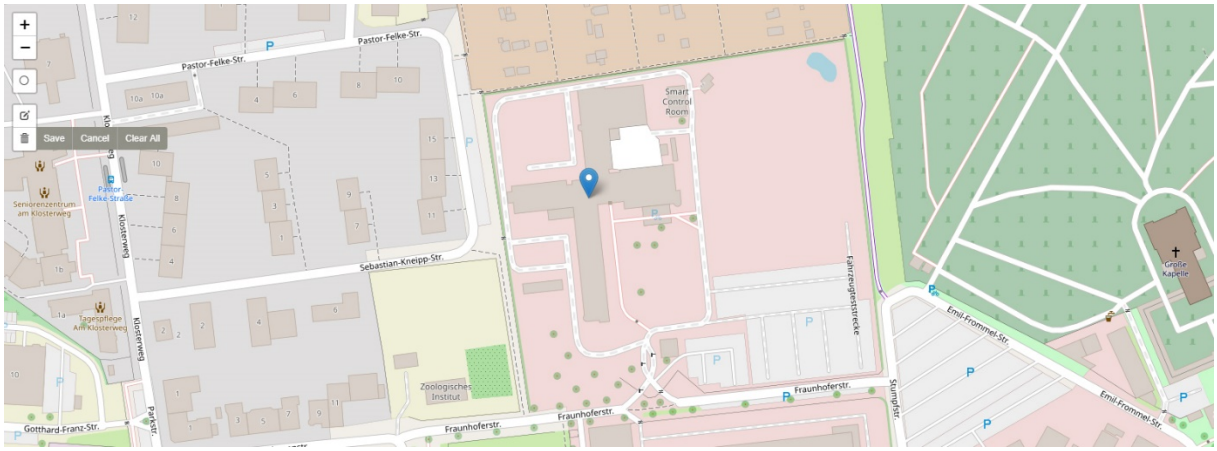

Figure 13: Set factory coordinates

2.5 CLOUD INTEGRATION

Figure 14 shows the entire architecture from the machine level of a single factory to the cloud including the relationship to the Industrial Internet Reference Architecture (IIRA) three tier architecture. Figure 15 shows a schematic for the integration of two factories A and B into the SFW.

1. OPC UA Servers for shop floor devices registered in network (Plug and Work).
2. The AutomationML (AML) model of each device is loaded dynamically from the OPC UA Servers in the shop floor at the bottom right. (Plug and Work).
3. The Fraunhofer IOSB web service AML2UA generates the OPC UA aggregation server based on an AML model that defines a mapping from the underlying AML models at the shop floor level. The OPC UA aggregation server adds additional semantics to the information model as required. See also section 2.5.2 for details.
4. Data is transferred from the OPC UA servers in the shop floor to the OPC UA Aggregation server in the Platform Tier with OPC UA.
5. CEP (Complex Event Processing) can reduce the amount of data to be sent to the cloud by extracting higher level, context-sensitive information in real-time and sending a

notification (or a complex event) when a situation is detected. CEP can also generate new information such as KPIs derived from the factory data.

6. The publisher establishes a connection to the SFW Server and the IoT Hub in the Microsoft® Azure™ Cloud and tunnels the OPC UA data with a secure encryption.

The Smart Factory Web Portal and the IoT Web Applications visualize the OPC UA data using JSON files derived from the same AML file for configuration.

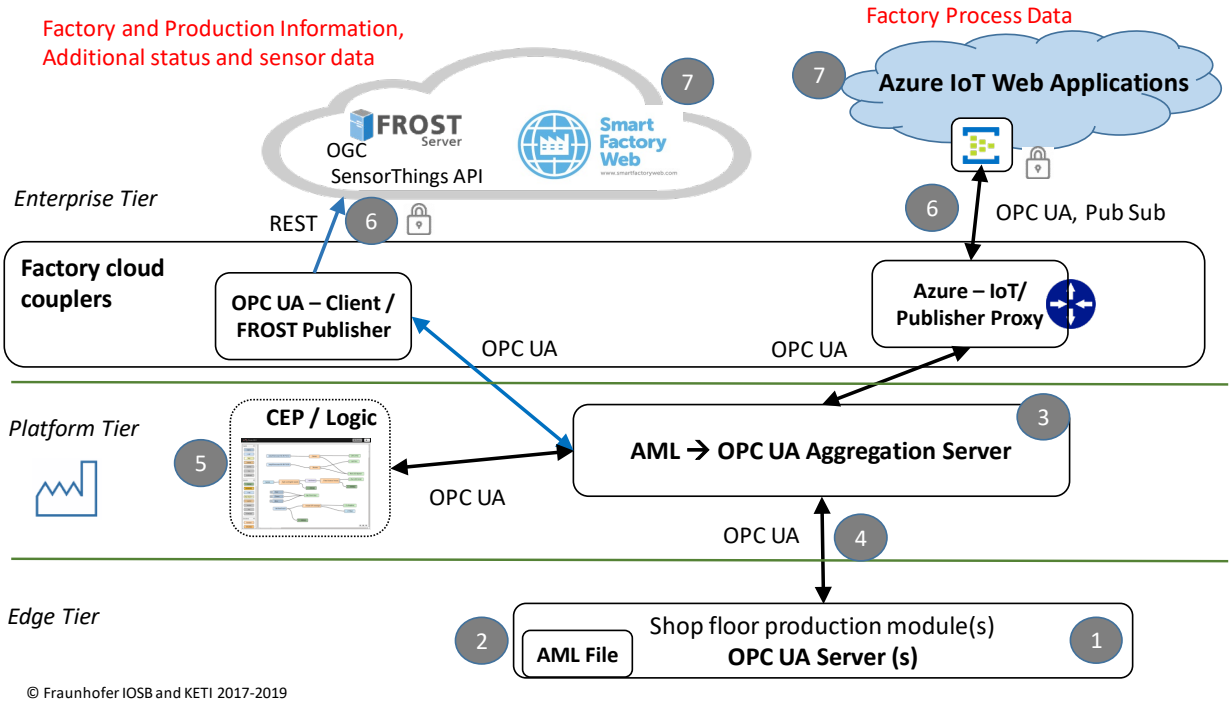

Figure 14: Integration of a factory into the clouds Smart Factory Web and Azure

OPC UA serves as a comprehensive and secure communication protocol from the machine level into the cloud. A cloud coupler on the shop floor publishes the availability of a factory and selected process data in the Smart Factory Web portal. Customers or even smart machines can use this information to make decisions for placing an order and track their orders in real time. To simplify the setup of the connection to the cloud and minimize the commissioning time the cloud coupler also aggregates all OPC UA servers on devices in the factory into a single OPC UA aggregated server address space.

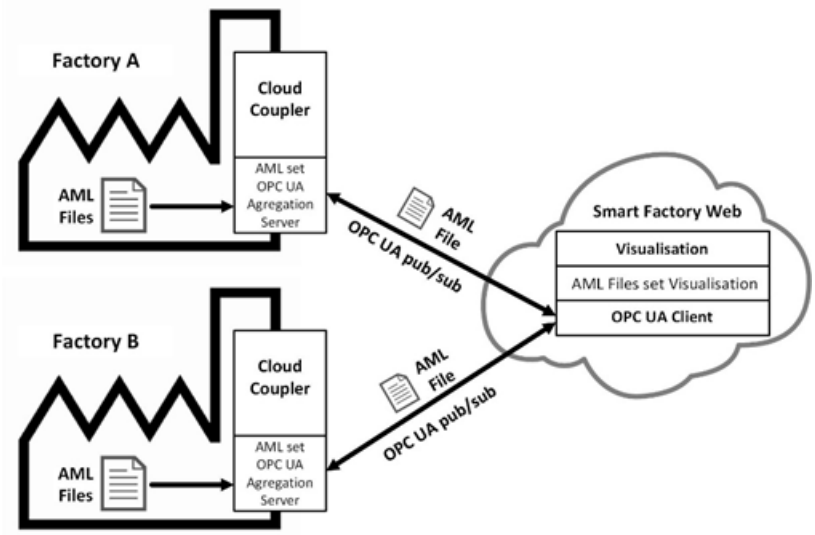

Figure 15: Integration of factory A and B into Smart Factory Web with OPC UA configured with AML

2.5.1 SOLUTION THROUGH STANDARDIZED SYSTEM BOUNDARIES

Through standardized system interfaces, manual engineering effort can be reduced and plug-and-work-capable devices and algorithms can be used. Figure 16 shows how the architecture of the SFW leads to improved interoperability at the interfaces through the use of established standards such as OPC UA and AutomationML. In point 1, the modeled machine data, up to point 7, is made available via OPC UA to a cloud service such as the SFW without further manual interaction. The quality of adaptability achieved in this way makes it possible to react dynamically to changes in the production environment and make these available sufficiently quickly in cloud services. The client/client architecture of the edge gateway makes it possible to access and use cloud services from the inside without having to adapt the configuration of the local production network. The production network and the publicly used network are always separated from each other via the edge gateway. The cloud OPC UA server at point 5 of Figure 16 forms the digital twin of one or more smart Factories. The information models of the individual machines of a Smart Factory are mirrored by a method called in the cloud OPC UA server. Thus, a cloud service such as the Smart Factory Web has access to all released data of the machines without a direct connection to the shop floor. The indirect connection (firewall property), the user role management that restricts the view of the data and the native use of OPC UA security mechanisms such as "sign & encrypt" lead to a high degree of trustworthiness. [Hey 2019]

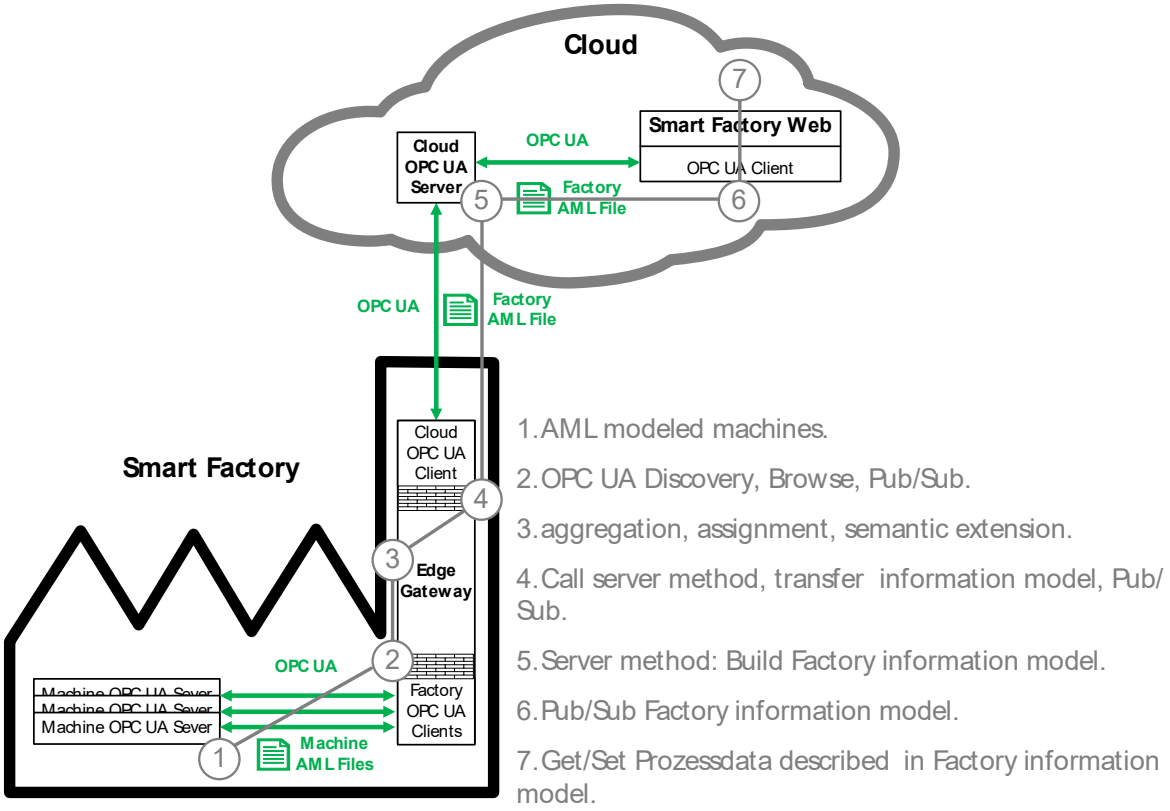

Figure 16: Factory integration into Smart Factory Web

2.5.2 AML TO OPC UA AGGREGATION SERVER

Creating an information model of an OPC UA server is still a mostly manual process, very time-consuming, expensive and error prone. There is a growing need for automated methods that are able to use information that already exists in different sources to build useful OPC UA information models. The AML2UAConverter[3] is a software service that automates creation of an OPC UA information model by reusing information already included in an AML model. The service has been developed based on the companion specification [OPC 2016] and [DIN 2016b], which define how to combine OPC UA and AutomationML standards to simplify the creation of OPC UA information models. Note that an OPC UA server can be created based on multiple AML models.

In [OPC 2016], AML InternalElements are represented by OPC UA Objects. AML SystemUnitClasses and RoleClasses are represented by OPC UA ObjectTypes. The mapping of relations is defined too.

Such a server is called an OPC UA Aggregation Server. In this case, the AML2UA converter resolves references between individual models, ensures consistency of the whole information model and eliminates redundancy. The AML2UA service is used in the SFW to automatically create the OPC UA information models of our demo factories. An example is shown in Figure 18. Figure 17 shows

[3] *https://aml2ua.iosb.fraunhofer.de*

a part of the AML model of the model factory of Fraunhofer IOSB in Karlsruhe. The corresponding part of the OPC UA model is shown in Figure 18.

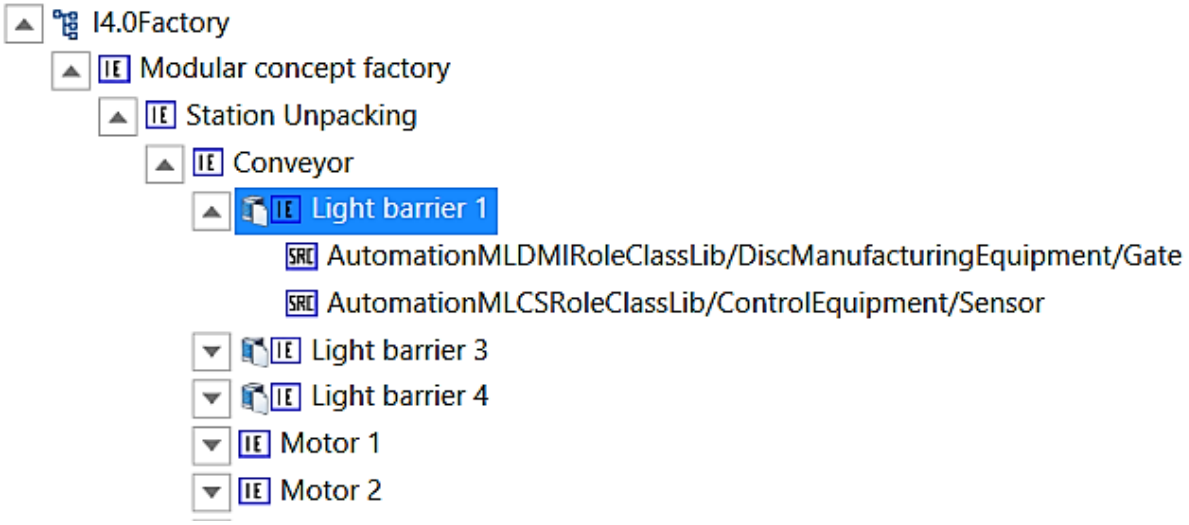

Figure 17: AutomationML example for the demo factory in Karlsruhe

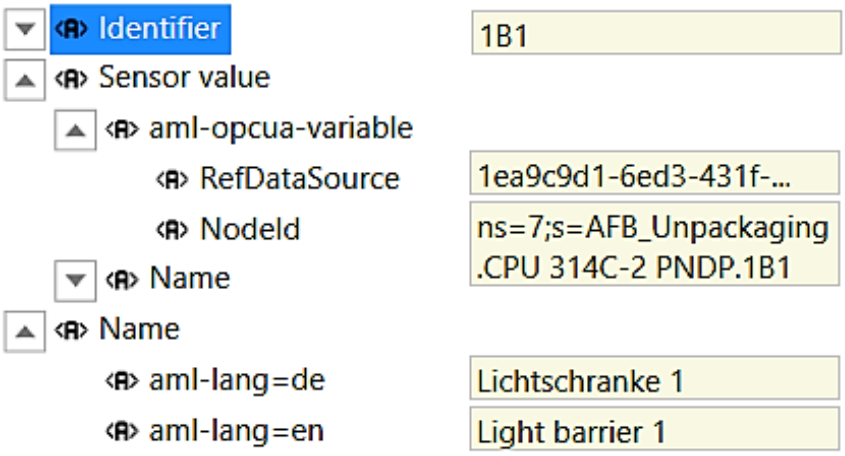

Figure 18: A part of the OPC UA information model for the AutomationML example

The advantage of the proposed solution is in speeding up the development of an OPC UA information model, and enriching it with explicit semantics. Instead of a flat structure of variables and nodes with non-intuitive naming (c.f. “Identifier = 1B1” in Figure 18), which is difficult to understand even by humans, the OPC UA model is hierarchically structured and also multi-lingual (cf. “Lichtschranke 1” in German and “Light barrier 1” in English).

The AML2UA converts AML elements to OPC UA nodes and the server information model mirrors the structure of the AML file. This automatically created OPC UA server can act as aggregation server, if the AML file contains variables accessible from an OPC UA server on the shop floor. These shop floor variables are defined according to the AML DataVariable concept that serves to describe information and configuration data needed for communication with a protocol such as OPC UA [AML 2017a], cf. also 3.2.1 for details.

The easiest way to ensure the accessibility to the shop floor is to launch the aggregation server on a device in the factory network.

Figure 19 shows the process of the AML2UA converter and all the intermediate products.

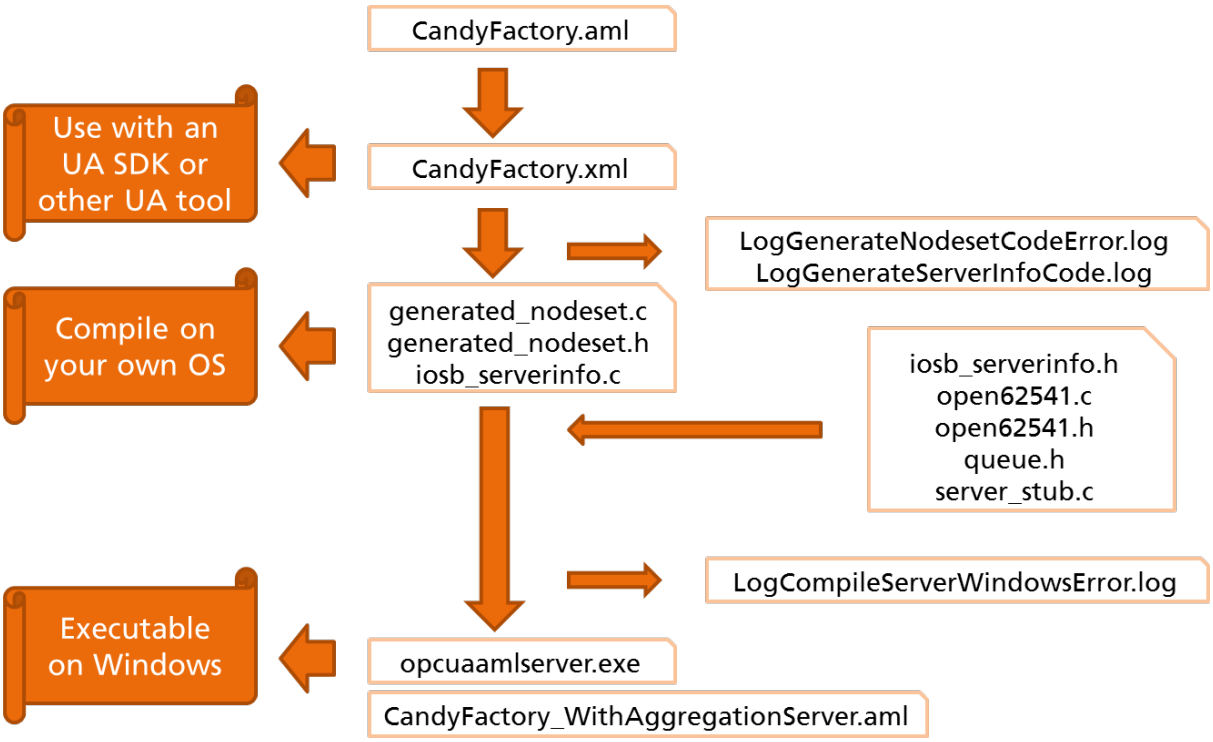

Figure 19: Functioning of the AML2UA converter

Fraunhofer IOSB has developed an AML2UAConverter that transforms an AML model into a UA-XML-Nodeset. This has been done based on the rules of the [DIN 2016b], which is built on [OPC 2016] and the Automation Markup Language [IEC 2018].

The converted UA-XML-Nodeset is used for automatic creation of C-code for an aggregation OPC UA server based on the open source open62541 toolkit (available at *https://open62541.org/*). For all DataVariables of type OPC UA available in the initial AML File, an integrated OPC UA Client, which subscribes to all variables in the underlying OPC UA-Servers, is integrated into the aggregation server automatically.

The AML2UA converter supports dynamic integration of components and systems with total change management.

2.5.2.1 DEALING WITH MULTIPLE AML FILES

The concept of generating an OPC UA server from an AML file has been extended to include generating hierarchical OPC UA servers from several AML files. In addition to the component AML2UAConverter, which enables a one-step generation, Fraunhofer IOSB has developed two components, the AML2UAAggregator and the AMLConfigurator, to enable multi-level OPC UA server generation. The component AMLConfigurator combines all underlying AML files in an AML container (.amlx), which is imported by the AML2UAAggregator and processed to a new master

AML file. The "intermediate products" generated by the AML2UA converter (NodeSets and C-code) can be used to generate executable OPC UA servers. The procedure can be applied iteratively as an iterative procedure is available in the AML2UA Center. Using the DataVariable concept, the newly created master component (as a hierarchical OPC UA server) is directly linked to the subordinate servers.

The extended version of the AML2UA converter is shown in Figure 20. See also [Okon 2019].

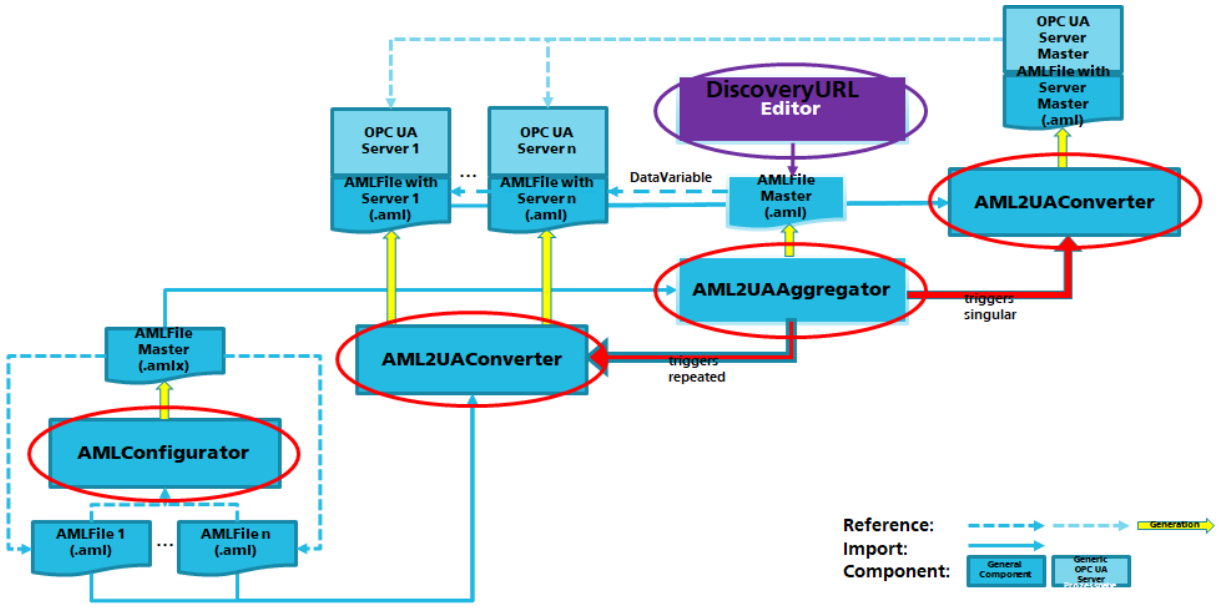

Figure 20: Extended AML2UA converter - Multi-level conversion

The multi-level AML2UA converter consists of the following components:

- AMLConfigurator creates an AML container (.AMLX) from multiple AML files (.aml)
- AML2UAConverter converts an AML file into:
  - an extended AML file (for the aggregation server)
  - an OPC UA NodeSet according to [OPC 2016] and [DIN 2016b].
- AML2UAAggregator performs the following actions:
  - triggering of the AML2UAConverter for each file in an AML container
  - aggregation of the extended AML files (via AML2UAConverter) into an integrated description into one master AML file with DataVariables referencing the generated NodeIds described in the extended AML files.

The extended AML2UA converter implements the following algorithm:

1. Create AML files.
2. Check created files for correctness in the AMLTest-Center and correct them if necessary.
3. Integrate files into an AML container.
4. Make AML containers available in the AML2UACenter for generation and generate NodeSets, code and executable servers, as well as the master AML file.

5. Carry out any subsequent changes to the master AML file at your own responsibility, in particular to the DiscoveryURL, test it for correctness in the AMLTest Center and correct it if necessary.
6. Provide the (modified) AML master file to the AML2UACenter for generation and generate NodeSet, code and executable server.

The algorithm is shown in Figure 20.

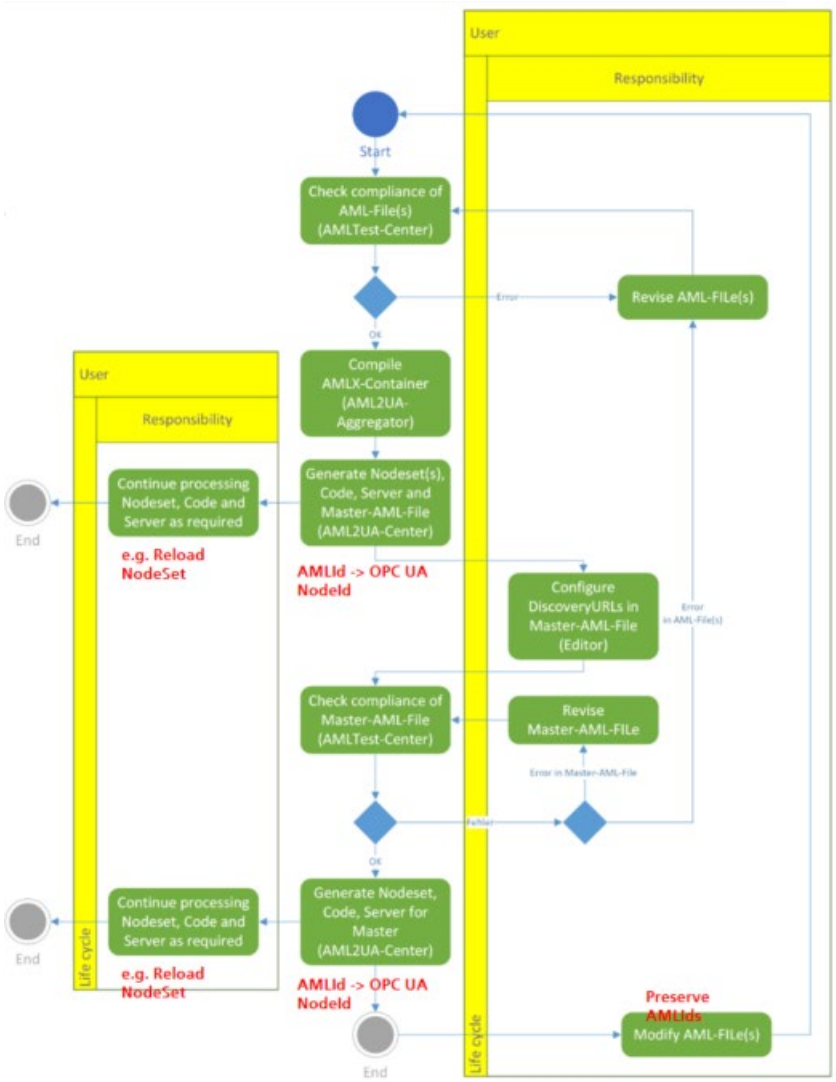

Figure 21: AML2UA generation algorithm with change management for multi-level AML files

2.5.2.2 MANAGEMENT OF AML FILES

With the rising importance of digitalization, many industrial companies have adopted AML for modeling of data exchange, in particular between engineering tools. Business dynamics and changes in the operating environment often give rise to continuous changes in the underlying AML models. This is especially true for the SFW, as it is based on heterogeneous and highly distributed information resources and therefore needs efficient mechanisms to cope with changes in the environment.

Change management is defined as the methods and tools in which a company describes and implements change within both its internal and external processes. Here we restrict our attention to the underlying OPC UA servers and AML models in the IT systems and do not address other software systems or organizational and human resource aspects. The goal is to ensure the consistency of the underlying information model and all dependent artefacts and to support the user to manage changes more easily.

In the context of Smart Factory Web, we go beyond a standard change management process; to a continual improvement process. We propose methods for the discovery of the changes by analyzing the IDs of the AML entities and for change propagation to all OPC UA servers created based on this model.

Change discovery is supported by implementing a mapping based on the AML IDs. The precondition is that the AML models were changed by ensuring that elements describing the same entity retain their AML ID.

Change propagation has been supported by ensuring that an OPC UA server continuously processes the data without the need to restart it. The precondition is that an OPC UA server can dynamically load an OPC UA NodeSet.

The approach is summarized in Figure 21.

There are many benefits of the proposed approach, such as:

- automatic generation of OPC UA-NodeSets from AML-Files considering hierarchical aspects,
- dynamic modifications of OPC UA-NodeSets after AML model changes,
- consistency of data models because of automatic generation and modification and
- OPC UA-Wrapping of (not only OPC UA) variables via the DataVariable concept.

In the future, we will extend the AML2UA converter to account for domain-specific knowledge, such as OPC UA Companion Specifications and to allow users to select a subset of an AML model that should be converted into an OPC UA information model.

2.5.3 AML FOR CONFIGURATION OF MICROSOFT AZURE

Microsoft provides a pre-configured “Connected Factory” solution filled with simulated demo factories. Two configuration files must be edited to connect to a real factory. The first file specifies what is to be visualized and the second how. The first file PublishedNodes.json is a JSON file generated from the AML model of the factory by a web service on the SFWP; this file is configured by a publisher (a software component) on a shop floor “edge” device and contains a list of the OPC UA variables from the OPC UA Aggregation server to be published to Microsoft Azure. The publisher connects to the factory OPC UA Aggregation Server set in the JSON file and transfers process data to the Microsoft Azure cloud. The second configuration file ContosoTopology.json, describing how visualization is to be done, is also a JSON file generated from the same AML model by a web service on the SFWP. This second JSON file defines the mapping of the process data from a factory (which the publisher puts in the cloud), to the

associated factory visualization in the Microsoft Azure “Connected Factory” solution. The file ContosoTopology.json is located in the Microsoft Azure Cloud and is updated as soon as a new factory is added or an existing one is changed. In addition, this file defines the configuration of units, hierarchies, relationship to and visualization of Key Performance Indicators (KPIs) and Overall Equipment Efficiency (OEEs).

With these two JSON configuration files a new factory can provide process data as OPC UA data for visualization in the Azure “Connected Factory” solution.

The PublishedNodes.json file has the following structure:

- list of OpcNodes
  - OpcNodeId

Here you see the detailed mapping from the AML elements to the PublishedNodes elements:

- OPC UA servers
  - all InternalElements referring to role “DataVariableRoleClassLib/DataSource/ OPCUA-Server/AggregationUA-Server”
  - if there is no such server, take all InternalElements referring to role “DataVariableRoleClassLib/DataSource/OPCUA-Server”
- OpcNode
  - all DataVariables referring to the UA Servers
  - mapped information:
    - DiscoveryUrl of the server
    - UseSecurity attribute (bool)
- OpcNodeId
    - NodeId

The top of the JSON file for the CandyFactory example is displayed in Figure 22 for the first two OPC UA variables. The original file is much longer.

```
[
    {
        "EndpointUrl": "opc.tcp://localhost:16664",
        "UseSecurity": false,
        "NodeId": {
            "Identifier": "ns=1;s=64290cf8-aa91-4141-be53-d973294a62a9.Temperature.Temperature-UA"
        }
    },
    {
        "EndpointUrl": "opc.tcp://localhost:16664",
        "UseSecurity": false,
        "NodeId": {
            "Identifier": "ns=1;s=64290cf8-aa91-4141-be53-d973294a62a9.Random name.Random name-UA"
        }
    }
]
```

Figure 22: Part of the JSON configuration file for two OPC UA Nodes

The second file, ContosoTopology.json, is to configure the Azure IoT Web Application. It contains the hierarchical structure of the factory assets. The converted AML file contains only the description of one factory. The SFW merges all factory JSON files into one combined ContosoTopology file used by the Azure IoT Web Application.

The structure of the Azure IoT Web Application is strict and limited to the three hierarchy levels described below. The modeling possibilities of the SFW are more generic and not restricted.

The ContosoTopology file follows this simplified structure:

- GlobalConfigurationData
- list of Factories
    - list of ProductionLines
        - list of Stations
            - list of OpcNodes

Here you see the detailed mapping from the AML elements to the ContosoTopology elements:

- GlobalConfigurationData (used by the root element, factory, productionline and station)
    - InternalElement with Guid, Name, Description and optional attributes OeeOverall, OeePerformance, OeeAvailability, OeeQuality, Kpi1, Kpi2, ImagePushpin and Simulation
    - an optional image reference (ExternalDataReference with MIMEType starting with “image/” and a “RefURI” attribute.
- Factory
    - InternalElement referring to “AutomationMLExtendedRoleClassLib/Area”
    - <GlobalConfigurationData>
    - Location attribute (City, Country, Latitude, Longitude)
- ProductionLine
    - InternalElement referring to “AutomationMLExtendedRoleClassLib/ProductionLine”
    - [Adding Dummy-ProductionLine if there is none]
    - <GlobalConfigurationData>
- Station
    - InternalElement referring to “AutomationMLExtendedRoleClassLib/WorkCell”
    - <GlobalConfigurationData>
    - DiscoveryUrl of the UA Server with the most DataVariables
- OpcNode
    - NodeId
    - Names of the 2 parent objects
    - Unit
    - optional attributes (Relevance, OpCode, Visible, ConstValue, Minimum, Maximum)
    - several attributes with RefSemantic MinimumAlertActions or MaximumAlertActions (Type, Parameter and description)

The top lines of the JSON file of the CandyFactory example are shown below in Figure 23. The first two OPC UA nodes of the station "Chocolate Buttons Production Module" are displayed.

```
{
   "Factories": [
      {
         "Guid": "282c58aa-1c61-40c0-adbe-c6006c8aee6b",
         "Name": "Big factory",
         "Description": "First, cocoa beans are processed into cocoa mass. In order to make chocolate from the cocoa mass,
            it is mixed with sugar, possibly also cocoa butter and milk products (today almost exclusively in dry form,
            for example as milk powder) and spices.",
         "Image": "bigfactory.png",
         "ProductionLines": [
            {
               "Guid": "8f11d523-8848-4964-8c18-b26b5ab2fcc0",
               "Name": "Chocolate Buttons Production Line",
               "Image": "factory_floor.gif",
               "Location": {
                     "City": "Karlsruhe",
                     "Country": "Germany",
                     "Latitude": 49.001893,
                     "Longitude": 8.3996389
                },
               "Stations": [
                  {
                     "Name": "Chocolate Buttons Production Module",
                     "Description": "The chocolate buttons production module is responsible for pressing of the chocolate.",
                     "Image": "assembly_station.png",
                     "OpcUri": "http://www.iosb.fraunhofer.de/candyfactory.aml",
                     "OpcNodes": [
                        {
                           "NodeId": "ns=1;s=64290cf8-aa91-4141-be53-d973294a62a9.iProductionCounter",
                           "SymbolicName": "Chocolate Buttons Production Module.Production Counter",
                           "OpCode": "Avg",
                           "Units": "pcs"
                        },
                        {
                           "NodeId": "ns=1;s=64290cf8-aa91-4141-be53-d973294a62a9.iStatusKPI",
                           "SymbolicName": "Chocolate Buttons Production Module.Status KPI",
                           "OpCode": "Avg",
                           "Units": "INT"
                        },
```


Figure 23: Part of the JSON configuration file for the candy factory example

2.5.3.1 SECURE DATA FLOW

Figure 24 shows the secure data flow in the Microsoft Azure Cloud. The following steps describe the communication and security mechanisms used to exchange process data between the OPC UA Aggregation Server and the Azure Cloud.

1. OPC Publisher reads the required OPC UA X509 certificates and IoT Hub security credentials from the local certificate store.
   - if necessary, the OPC Publisher creates and stores any missing certificates or credentials in the certificate store.
2. OPC Publisher registers itself with IoT Hub.
   - uses the configured protocol. Can use any IoT Hub client protocol supported by the SDK (Software Developer Kit). The default is MQTT.
   - protocol communication is secured by TLS.
3. OPC Publisher reads configuration file.
4. OPC Publisher creates an OPC Session with each configured OPC UA Server.
   - uses TCP connection.
   - OPC Publisher and OPC UA Server authenticate each other using X509 certificates.
   - all further OPC UA traffic is encrypted by the configured OPC UA encryption mechanism.
   - OPC Publisher creates, in the OPC Session for each configured publishing interval, an OPC Subscription.

- creates OPC Monitored items for the OPC Nodes to publish in the OPC Subscription.

5. If a monitored OPC Node value changes, OPC UA Server sends updates to OPC Publisher.
6. OPC Publisher transcodes the new value.
   - batches multiple changes if batching is enabled.
   - creates an IoT Hub message.
7. OPC Publisher sends a message to IoT Hub.
   - use the configured protocol.
   - communication is secured by TLS.
8. Time Series Insights (TSI) reads the messages from IoT Hub.
   - uses AMQP over TCP/TLS.
   - this step is internal to the datacenter.
9. Data at rest in TSI.
10. Connected Factory WebApp in Azure AppService queries required data from TSI.
    - uses TCP/TLS secured communication.
    - this step is internal to the datacenter.
11. Web browser connects to the Connected Factory WebApp.
    - renders the Connected Factory dashboard.
    - connects over HTTPS.
    - access to the Connected Factory App requires authentication of the user via Azure Active Directory.
    - any WebAPI calls into Connected Factory app are secured by Anti-Forgery-Tokens.
12. On data updates, the Connected Factory WebApp sends updated data to the web browser using the SignalR protocol[4] and secured by TCP/TLS.

[4] *https://dotnet.microsoft.com/apps/aspnet/signalr*

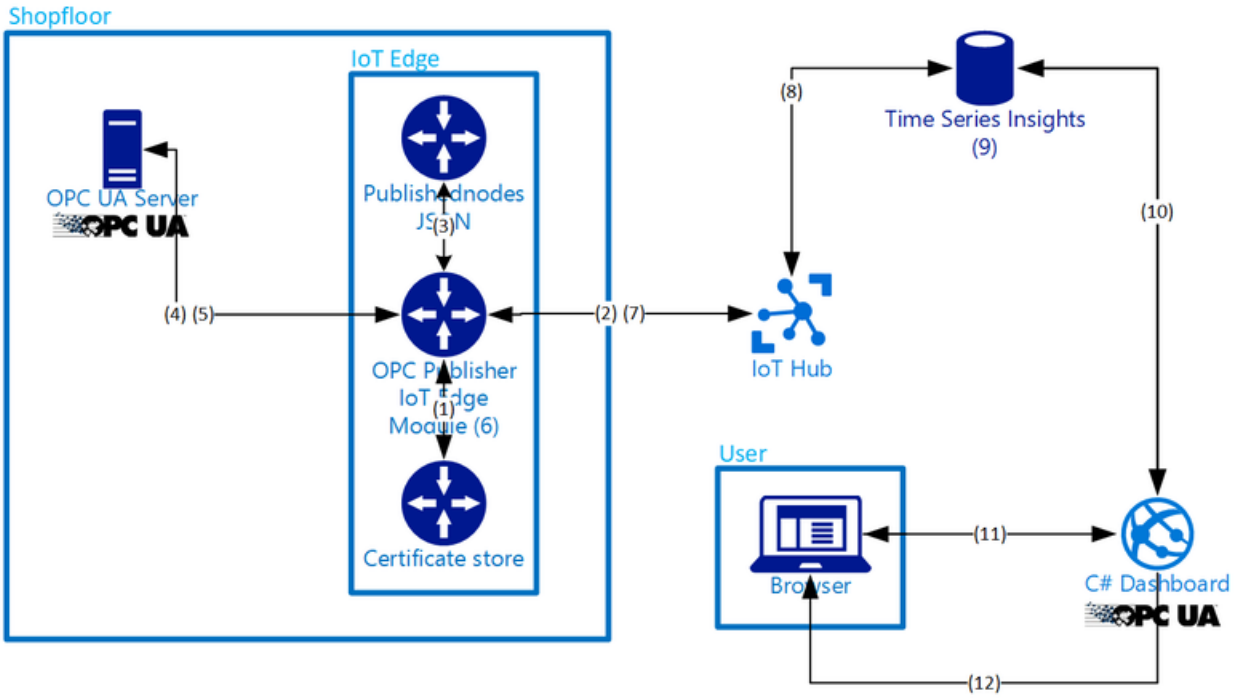

Figure 24: Secure Data Flow in MS Azure

2.5.4 AML FOR CONFIGURATION OF SENSORTHINGSAPI

The Fraunhofer Open Source SensorThings API Server - FROST® implements the SensorThings API standard[5]. SFW configures the FROST Server after every AML file upload. Therefore, all assets with UANodes are sent to the FROST Server. The DiscoverUrl of the OPC UA Aggregation Server and the NodeIds of the variables are used for assignment.

The FROST Publisher is an OPC UA client launched in the factory network or at least with access to the OPC UA Aggregation Server. This client connects to the aggregation server, browses the information model and subscribes to changes of all DataVariables. All value changes are published to the appropriate FROST Server.

2.6 IMPLEMENTATION STATUS

Table 2 describes the functions that have already been implemented in the SFW testbed project and those that are to be implemented next. It also describes which standard has been used or is planned as a solution for the respective function. Most of the implemented functions follow the standards OPC UA and AutomationML. Due to the semantic strengths of these two standards, a high degree of interoperability was achieved, especially in the data representations (ISO/OSI model layers 6 and 7).

Nevertheless, there were functions that the standards could not solve on their own. On the one hand, there is the need to aggregate an increasing number of OPC UA servers (e.g. for controllers, sensors, user interfaces) in the shop floor (cf. Figure 14). This drastically reduces the number of

[5] *https://www.iosb.fraunhofer.de/servlet/is/82077/*

connections to a cloud service. The resulting OPC UA Aggregation Server generated with the AML2UA converter (cf. section 3.3) uses the standards mentioned above, but is itself implemented in a proprietary way. Using the AML2UA aggregation server requires the configuration of the OPC UA server addresses, EndpointURLs, renaming of variables, semantics, and so on using an AML file. Increasing numbers of OPC UA servers and variables leads to considerable manual engineering effort. On the other hand, the challenge of establishing communication with a factory from the internet to retrieve data is not desired or is considered to be critical (cybersecurity). To make full use of cloud services such as data analysis or predictive maintenance, bidirectional connections with semantic information to the shop floor are required. This also requires manual engineering effort to set up port releases and change firewall settings. The two manual engineering efforts mentioned above are to be automated by the planned solutions. The aggregation of OPC UA servers is solved by an iteration of Discovery, Browse, Pub/Sub in the edge gateway as shown in Figure 16. This permits dynamic reactions without manual interaction to changes of the OPC UA servers on the shop floor (cf. change management in section 3). The connection to the cloud is realized from inside the edge gateway. The service of a static cloud OPC UA server is used as a relay. For different factories, the same server is used. The visibility of factory data is controlled through user roles and certificates. Due to the connection setup from the smart factory to the cloud, no further network configurations such as port releases or firewall settings have to be made (assuming that standard ports are used).

The aim is to make cloud services available with the functions provided in Table 2 without manual engineering of machine data and information by means of plug and work. This is achieved using standards and applying interoperability. [Hey 2019]

Function	Status in SFW	Used solution, software, hardware, implementation	Standard
Asset Description	implemented	Automation ML	yes
Find machines	scheduled	OPC UA Discovery	yes
Find machine data	scheduled	OPC UA Browse	yes
Aggregate machine data	implemented	AML2UA Aggregation Server	in part
Generate machine data AML	scheduled	Automation ML / OPC UA	no
Subassets (Machines) to Factory (Asset)	scheduled	Automation ML	yes
Assignment of Local to Cloud Data	scheduled	Webinterface, Automation ML	no
Local Client Configuration	implemented	OPC UA, AML2UA Aggregation Server	in part
Cloud Client Configuration	implemented	FROST Publisher, Azure IoT Publisher	no
Hardware Edge Gateway	scheduled	Hilscher IoT Gateway	no
OPC UA as Cloud Service	scheduled	NodeRed with OPC UA	in part
SFWP: Load information model	implemented	Automation ML	yes
SFWP Import: Load AML Model from OPC UA Data	scheduled	Automation ML, OPC UA	yes
SFWP: transmit data	implemented	FROST Publisher, Azure IoT Publisher	no
SFWP: transmit data	scheduled	OPC UA	yes
SFWP: Integrate Assets via I4.0 Management Shell	scheduled	OPC UA, Automation ML, I4.0 AAS	in part
IT-Security	scheduled	OPC UA user roles, sign and encrypt	yes
Display data dynamically	implemented	SFW Portal, Automation ML	in part
Find factory	implemented	SFW Portal Search, Web Map Service ISO 19128	in part

Table 2: Smart Factory Web functions

3 Detailed Description of Standards Usage

This section gives a detailed specification of the usage of the standards OPC UA, AutomationML and SensorThings API. Familiarity with these standards is assumed for a full understanding, but we provide some background information to attain a basic understanding.

Figure 25 gives an overview of the principal concepts of these standards and how they are used in Smart Factory Web.

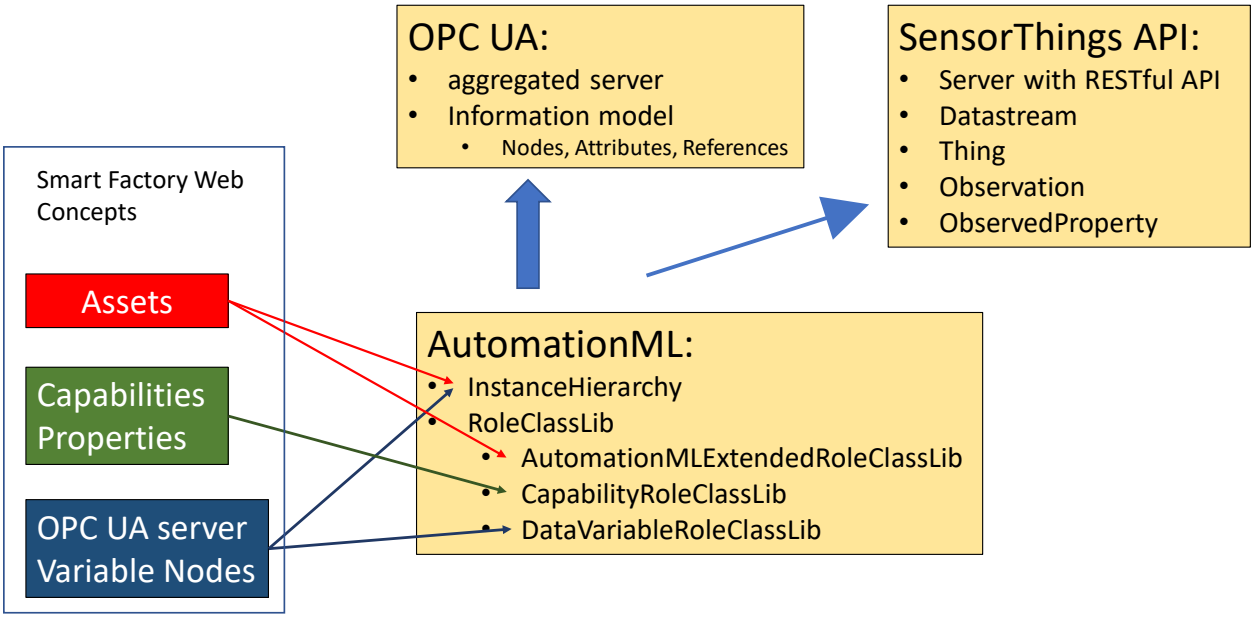

Figure 25: Conceptual Mapping (simplified)

3.1 AUTOMATIONML

A factory is described in AML to register the factory in the SFWP by importing an AML file. The structure of the file is explained in the following sections.

3.1.1 THE ASSET STRUCTURE OF THE FACTORY IN A HIERARCHY OF INSTANCES

The factory is represented as a hierarchy of AML Internal Elements, each of which stands for an Asset or sub-Asset in the context of Smart Factory Web and can be one of the following elements:

- Tag Description (optional)—description of the Asset
- Attribute Shortdescription (optional)—short description of the Asset
- Attribute Localisation (optional)—geo coordinates of the Asset (needed only for the Top-Asset). This may be a point or polygon.
- Attribute OPC UA Node (optional)—definition of the OPC UA Node of the Asset. Details in section 3.2.1
- InternalElement Sub-Asset (optional)—further sub-Assets
- InternalElement OPCUA Server (optional, but at least once in the instance hierarchy)—definition of the internal OPC UA Server (needed only for the Top-Asset). Details cf. section 3.2.2
- InternalElement Capability (optional)—definition of a Capability of an Asset. Details cf. 3.1.2
- RoleRequirement AMLExtendedRoleClassLib (mandatory, cf. section 3.1.8)—an asset shall have one of the following roles (RoleRequirements, direct or indirect) to be recognized as an Asset:
    - AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Structure/

ResourceStructure

- AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Resource

In addition, the Asset shall not have any of the following roles:

- DataVariableRoleClassLib/DataSource
- AutomationMLCSRoleClassLib/ControlEquipment/Controller
- AutomationMLExtendedRoleClassLib/Panel
- CommunicationRoleClassLib/PhysicalNetwork
- CommunicationRoleClassLib/PhysicalConnection
- CommunicationRoleClassLib/PhysicalDevice

3.1.2 Structure of a Capability in AML

Similar to Assets, Asset Capabilities are described in AutomationML with Internal Elements. However, an Asset Capability may contain only the following elements:

- Role Requirement CapabilityRoleClassLib (mandatory)—every Capability shall be linked via this Role Requirement to a Capability concept (cf section 3.1.10).
- Attribute with Values—each property of a Capability shall have an Attribute with a value. Such an Attribute has a name and possibly a unit.

3.1.3 Interface Class Libs

Contains the description of the interfaces.

3.1.4 Role Class Libs

Role class libraries (Libs) are an important part of the AML model as they are used to structure and define the semantics of classes of system objects. The following sections 3.1.5 to 3.1.10 summarize the usage of these libraries in the context of SFW.

3.1.5 CommunicationRoleClassLibs

These roles are used as a basis for the roles in DataVariableRoleClassLib (cf section 3.1.9)

3.1.6 AMLBaseRoleClassLibs

Contains basic AML role definitions

3.1.7 AMLCSRoleClassLibs and DMIBaseRoleClassLibs

These roles are a basis for the roles in AMLExtendedRoleClassLib (cf section 3.1.8)

3.1.8 AMLExtendedRoleClassLibs

Contains the Asset type hierarchy. Every Asset shall reference one of these type definitions. AMLExtendedRoleClassLibs is derived from AMLBaseRoleClassLib (cf section 3.1.6)

Example Asset types defined in the Companion Standard and IEC 62264-1:

- Area: an area is a physical, geographical or logical grouping determined by the site. It may contain work centers such as process cells, production units, production lines and storage zones.
- Work centers are elements of the equipment hierarchy under an area that performs production, storage, material movement or any other Level 3 or Level 4 scheduled activity. Types of work centers specifically defined in IEC 62264 are
    - process cells,
    - production units,
    - production lines or
    - storage zones (typically has the capability needed for the receipt, storage, retrieval, movement and shipment of materials)

3.1.9 DATAVARIABLEROLECLASSLIBS

Description of further protocols used at interfaces, e.g. OPC UA, ProfiNET, CanBus, etc.

3.1.10 CAPABILITYROLECLASSLIBS

Contains all capabilities in a tree structure. The capabilities of assets shall reference one of these capability classes.

3.2 AML AND OPC UA

How to use AML and OPC UA in combination is specified into two parts [DIN 2016b]:

1. Providing an AML model via OPC UA; this involves the AML integration into OPC UA. The goal is to communicate, exchange and operationalize AML using OPC UA. The purpose is to simplify the creation of OPC UA information models based on existing AML data. This can be used for re-engineering and maintenance use cases where the AML model evolves over time.

2. Accessing an OPC UA server via AML; this involves the integration of OPC UA information into AML. The goal is the lossless exchange of OPC UA system configuration within AML models to simplify the setup of OPC UA client connections to an OPC UA server. This reduces the need for manual configuration for discovery and browsing mechanisms. This can be used for the configuration of communication networks based on the description of network configuration and structure (including communication components of sensors and actuators with respect to communication system parameters, quality of service, network structure and wiring).

This specification is based on the companion specification “OPC Unified Architecture for AutomationML” [OPC 2016], which was created jointly by OPC Foundation and AutomationML e.V. It defines an OPC UA information model to represent the AML models.

The supplementary Best Practices Recommendation DataVariable [AML 2017a] specifies the modeling of the OPC UA server configuration in AML using the DataVariable concept.

3.2.1 Structure of an OPC UA Node in AML

OPC UA Nodes are described in AML with Attributes. OPCUA-Node Attributes contain the following elements:

- Tag Description (optional)—description of the UA Node
- Attribute internal OPC UA Node—description of the internal UA-Node with following elements:
  - RefSemantic—type description, in this case OPCUA
  - Atrribute RefDataSource—reference to the corresponding internal OPC UA Server
  - Attribute NodeId—contains the corresponding Node ID as value
- Attribute Aggregation Server UA Node—description of the aggregation UA-Node with following elements:
  - RefSemantic—type description, in this case OPCUA
  - Attribute RefDataSource—reference to the aggregation OPC UA Server
  - Attribute NodeId—contains the corresponding aggregation Node ID as value

3.2.2 Definition of the OPC UA Servers

There may be one or more OPC UA servers and OPC UA aggregation servers linked to underlying OPC UA servers.

An OPC UA server is modeled in AML as an InternalElement of RoleClass OPCUA-Server, a subclass of DataSource in the DataVariableRoleClassLib. The RoleClass OPCUA-Server has several defined attributes, including the DiscoveryURL which contains the complete information needed to connect to a specific endpoint of an OPC UA server.

The definition of the OPC UA Server comprises the following elements:

- ID (mandatory)—This ID is referenced e.g. by the Attribute RefDataSource of a RefSemantic aml-dataSourceType:OPCUA.
- Attribute Discovery URL (mandatory)—contains as value the URL (internal or public IP/URL if available) of the OPC UA Server including Port
- Attribute ApplicationUri (mandatory)—unique identifier, needed for the connection to Microsoft Azure.
- Attribute Name Space Table (needed only for the Aggregation Server)—contains the unique identifier of the OPC UA Server
- Role Requirement DataVariableRoleClassLib (mandatory)—defines the Internal Element as an OPC UA Server: DataVariableRoleClassLib/DataSource/OPCUA-Server defines the attributes of the internal Server; DataVariableRoleClassLib/DataSource/OPCUA-Server/AggregationUA-Server defines the attributes of the aggregation Server

The variables (Nodes) of the OPC UA server are modeled as attributes of the system elements (AML InternalElements) they belong to. The Element ‚RefSemantic‘ with content ‘CorrespondingAttributePath="aml-dataSourceType:OPCUA”’ has two sub-attributes. The sub-attribute RefDataSource stores the AML GUID (Globally Unique Identifier) of the DataSource

InternalElement corresponding to the OPC UA server. The sub-attribute ‚NodeId' stores the OPC UA NodeId of the referenced OPC UA Node.

3.2.3 SEMANTIC LINK TO EXTERNAL DICTIONARIES

AML can reference the semantic definitions of external dictionaries such as eCl@ss [AML 2017b]. eCl@ss is a hierarchical semantic system for grouping materials, products and services. eCl@ss classifies materials, products and services enabling a unique identification of production system component classes. Standardized properties are defined for each class. A central element of the eCl@ss specification is the International Registration Data Identifier (IDRI), which is based on the international standards ISO/IEC 11179-6, ISO 29002, and ISO 6532. The IRDI provides a unique identification code for each attribute and each class of objects.

The current version of SFW does not use eCl@ss. However, since eCl@ss is used in the AAS of I4.0, its use in SFW is under consideration for a future version.

Other product metadata dictionaries such as IEC 61360—Common Data Dictionary (CDD; *https://cdd.iec.ch/*) or the Harmonized System of the World Customs Organization[6] may also be referenced.

3.3 SENSORTHINGS API

The main concepts of the SensorThings API information model are summarized in Annex B. The FROST publisher creates one SensorThings API Datastream for each variable Node on the OPC UA Aggregation Server. As defined in [OGC 2016] a Datastream has links to the entities Thing, Sensor and ObservedProperty in Table 3 and has the properties shown in Table 4.

Entity	Definition [OGC-2016]	Use in Smart Factory Web; source in AML file
Thing	a thing is an object of the physical world (physical things) or the information world (virtual things) that is capable of being identified and integrated into communication networks (following the ITU definition)	Asset linked to OPC UA Aggregation Server with name, description, properties of the UA Node and Locations.
Sensor	an instrument that observes a property or phenomenon with the goal of producing an estimate of the value of the property	Name of OPC UA Node or empty?
ObservedProperty	specifies the phenomenon of an Observation	Name of OPC UA Node or empty?

Table 3: Mandatory Entities linked to a Datastream in SensorThings API

[6] *http://www.wcoomd.org/en/topics/nomenclature/overview/what-is-the-harmonized-system.aspx*

The Thing properties are DataType, Name, NodeID and ServerURL of the respective UA Node. A Thing has a Location, which is the location of the corresponding asset or the next higher Asset in the hierarchy with a defined location. Only the top Asset (factory) must have a defined location.

Name	Definition [OGC 2016]	Data Type	Use in Smart Factory Web; source in AML file
name	A property provides a label for Datastream entity, commonly a descriptive name.	CharacterString	Name of source UA Node on aggregation OPC UA server
description	The description of the Datastream entity.	CharacterString	Attribute description of the OPC UA Node
unitOfMeasurement	A JSON Object containing three key-value pairs. The name property presents the full name of the unitOfMeasurement; the symbol property shows the textual form of the unit symbol; and the definition contains the URI defining the unitOfMeasurement.	JSON Object	Unit of the OPC UA Node
observationType	The type of Observation (with unique result type), which is used by the service to encode observations.	Code value from list	OM_Observation; the result type is Any
properties	A JSON Object containing user-annotated properties as key-value pairs.	JSON Object	contains the metadata of the source OPC UA node on the aggregation server: DataType, Name, NodeID and ServerUrl.

Table 4: Properties of a Datastream in SensorThings API

The properties object for a Datastream in Table 4 is permissible in SensorThings API V1.1, but not in V1.0. V1.1 is expected to be approved as an OGC standard in 2020.

The Datastream is the time sequence of values of the OPC UA Node, each of which is an Observation in SensorThings API. Figure 28 shows one Observation of the Datastream in Figure 27. The combination of OPC UA Node ID and OPC UA Server URL is used to identify the data stream. The Datastream relates to the Thing in Figure 26. The navigationLinks allow one to easily access the related data on the FROST server through its RESTful interface. The phenomenonTime and resultTime of a DataStream are time intervals in ISO 8601 syntax and cover all Observations of the Datastream.

```
{
  "name" : "PLCatRaspberryPi",
  "description" : "raspberry pi running a PLC",
  "properties" : {
    "FactoryId" : 73629,
    "WgId" : 73632,
    "AMLID" : "8f11d523-8848-4964-8c18-b26b5ab2fcc1"
  },
  "Datastreams@iot.navigationLink" :
       "https://frost-smartfactoryweb.iosb.fraunhofer.de/v1.0/Things(1261)/Datastreams",
  "MultiDatastreams@iot.navigationLink" :
       "https://frost-smartfactoryweb.iosb.fraunhofer.de/v1.0/Things(1261)/MultiDatastreams",
  "Locations@iot.navigationLink" :
       "https://frost-smartfactoryweb.iosb.fraunhofer.de/v1.0/Things(1261)/Locations",
  "HistoricalLocations@iot.navigationLink" :
       "https://frost-smartfactoryweb.iosb.fraunhofer.de/v1.0/Things(1261)/HistoricalLocations",
  "@iot.id" : 1261,
  "@iot.selfLink" : "https://frost-smartfactoryweb.iosb.fraunhofer.de/v1.0/Things(1261)"
}
```


Figure 26: Thing definition


```
{
    "name" : "Chocolate Lenses Production Module.Temperature Difference Max",
    "description" : "maximum temperature difference between the top and bottom of the Stirling motor",
    "properties" : {
      "UANode" : {
        "DataType" : "xs:float",
        "Name" : "Chocolate Lenses Production Module.Temperature Difference Max",
        "NodeId" : "ns=1;s=8f11d523-8848-4964-8c18-b26b5ab2fcc1.rTempDifferenzMax.rTempDifferenzMax",
        "ServerUrl" : "opc.tcp://edgegateway:16664"
      }
    },
    "observationType" : " OM_Observation",
    "phenomenonTime" : "2018-10-02T16:30:34.663Z/2020-02-03T12:57:37.505Z",
    "resultTime" : "2018-10-02T16:30:55.817Z/2020-02-03T12:57:38.674Z",
    "Sensor@iot.navigationLink" :
        "https://frost-smartfactoryweb.iosb.fraunhofer.de/v1.0/Datastreams(4684)/Sensor",
    "Thing@iot.navigationLink" :
        "https://frost-smartfactoryweb.iosb.fraunhofer.de/v1.0/Datastreams(4684)/Thing",
    "Observations@iot.navigationLink" :
        "https://frost-smartfactoryweb.iosb.fraunhofer.de/v1.0/Datastreams(4684)/Observations",
    "unitOfMeasurement" : {
      "name" : "degree Celsius",
      "symbol" : "°C",
      "definition" : "http://unitsofmeasure.org/ucum.html#para-30 "
    },
    "ObservedProperty@iot.navigationLink" :
        "https://frost-smartfactoryweb.iosb.fraunhofer.de/v1.0/Datastreams(4684)/ObservedProperty",
    "@iot.id" : 4684,
    "@iot.selfLink" : "https://frost-smartfactoryweb.iosb.fraunhofer.de/v1.0/Datastreams(4684)"
}
```


Figure 27: Datastream definition


```
{
    "phenomenonTime" : "2020-02-03T12:56:57.525Z",
    "resultTime" : "2020-02-03T12:56:57.863Z",
    "result" : 29.063,
    "Datastream@iot.navigationLink" :
        "https://frost-smartfactoryweb.iosb.fraunhofer.de/v1.0/Observations(5374447)/Datastream",
    "FeatureOfInterest@iot.navigationLink" :
        "https://frost-smartfactoryweb.iosb.fraunhofer.de/v1.0/Observations(5374447)/FeatureOfInterest",
    "@iot.id" : 5374447,
    "@iot.selfLink" : "https://frost-smartfactoryweb.iosb.fraunhofer.de/v1.0/Observations(5374447)"
}
```


Figure 28: An Observation of a Datastream

3.4 Reference Model for Industrie 4.0 Service Architectures

A service is defined in [DIN 2018] following the OASIS definition to be “a mechanism to enable access to one or more capabilities where the access is provided by a prescribed interface and is exercised consistent with constraints and policies as specified by the service description”. This aligns with the IIC service definition [IIC 2019b]: “distinct part of the functionality that is provided by an entity through interfaces”; this definition follows ISO/IEC TR 14252:1996. [DIN 2018] defines a conceptual model for an Interaction-based Architecture, whereby an interaction (between components) is a behavior that is specified by a sequence of messages between two or more system components, each comprising primitives related to call events or notifications related to signal events. An interaction pattern comprises a typical sequence of messages (primitives or notifications) between interacting components as a solution to often-posed interaction needs. Typical primitive types at the service interface are request, indication, response and confirm. [DIN 2018] addresses both procedure-based interactions and state-machine-based interactions. A service with a procedure-based interaction (like a remote procedure call) comprise the typical primitives: invoke, result and error (the latter two often combined into a response primitive). The component that issues an invoke primitive acts in a client role, whereby the receiver of an invoke primitive acts in a server role. State machine-based interactions are message based. A firing transition in a state machine of an application process can trigger a message to another component where it is input to the recipient’s state machine.

Interaction patterns in Smart Factory Web:

- Request-response, stateless:
  - FROST server with clients accessing according to the SensorThings API apply a RESTful interaction
  - OPC UA read services
  - The OPC UA PublishRequest belongs to the Subscription Service Set and allows an OPC UA client to subscribe to changes of attribute values in nodes of an OPC UA server.
  - AML2UA converter

- Publish-find-bind-execute:
  - used to include new OPC UA Servers and link them to an Aggregation OPC UA server

Annexes

Annex A OVERVIEW OF OPC UA

OPC Unified Architecture (OPC UA) is defined in the multi-part standard IEC 62541. The specifications are maintained and also provided by the OPC Foundation. In this section we describe the characteristics of the base information model of OPC UA. For full details, refer to OPC UA Part 3 (Address Space Model) and OPC UA Part 5 (Information Model).

In the following text a word in italics with an initial uppercase letter (such as *Node*) denotes a term defined in the OPC UA standard.

A growing family of so-called Companion Specifications extends the base information model with elements specific to certain classes of machines and applications. An OPC UA Companion Specification defines *ObjectTypes*, *VariableTypes*, *DataTypes* and *ReferenceTypes* that represent specific semantics. OPC has a template for creating a Companion Specification. A wide range of industries define their standards for information models building on the OPC UA base information model. The verticals covered by the Companion Specifications are in diverse process control domains, including manufacturing, oil & gas, building automation and utilities. For AutomationML (IEC 61714) there is a Companion Specification “OPC Unified Architecture for AutomationML” (DIN SPEC 16592) that defines how to generate an OPC UA Server from an AML model; AML defines an XML-based data format for the storage and exchange of plant engineering information in a heterogeneous, multi-disciplinary tool landscape.

OPC UA defines mappings for *DataEncodings*, *SecurityProtocols* and *TransportProtocols*. The *DataEncodings* specify the serialization of the information model in UA Binary, UA XML or UA JSON. The security protocol UA Secure Conversation defines how serialized data is encoded and transferred over a Secure Channel. The underlying transport layer applies the *TransportProtocols* UA TCP, HTTPS and AMQP to transfer the secured message. OPC UA defines an abstract Connection Protocol as a duplex channel between clients and servers. The OPC UA Connection Protocol defines the byte structure of a message on the transmission medium and interacts with the secure channel.

The OPC UA information model is a connected graph with *Nodes*, Attributes of *Nodes* and *References* (directed edges) between *Nodes*. The *Nodes* are coupled to OPC UA functionalities: Data Access, Historical Data Access, Alarms & Events and Commands.

The object-oriented modeling paradigm is applied to objects, variables, data types and references. Objects typically represent system components and are structured hierarchically. The information model of an OPC UA server is divided into groups of *Nodes* called *NodeSets*. A *Node* belongs to exactly one *NodeSet*. The base *NodeSet* defined in OPC UA Part 3 and Part 5 is the basis for all other *NodeSets*. Every *Node* has a URL to identify its unique name space. The name

space *http://opcfoundation.org/UA/* is contained in all *Nodes* defined in the OPC UA standard.[1] OPC UA Part 6 describes the XML schema of the XML files of all *NodeSets*. An OPC UA server stores a list of all used *NodeSet* name spaces in a *NamespaceArray*. The *Nodes* and their references can then just index the respective element of the *NamespaceArray*. A *Node* in a *NodeSet* may be available in several OPC UA servers, but with different indices depending on the server. The name space index 0 is, however, reserved for the base *NodeSet* of the OPC Foundation.

Each *Node* contains attributes for identification, description or definition of access rights. The value of the attribute *NodeID* is unique over the complete server information model, i.e. over all *NodeSets* of the server. The *NodeID* is comprised of the name space URL (or index) and an identifier that is unique within the *NodeSet*. There are four types of identifiers:

- Numeric (a positive integer),
- String (max. 4096 Byte, case sensitive),
- Guid of format 00000000-0000-0000-0000-000000 and
- Opac: a free format of data type ByteString with max. 4096 Byte

The *BrowseName* of a *Node* is a language independent and human readable name. The *BrowseName* of a *Node* is not necessarily unique in a *NodeSet*, but unique amongst *Nodes* of the same hierarchy level. The sequence of *BrowseNames* (*BrowseNamePath*) to a *Node* is unique if the information model is hierarchical. The *DisplayName* of a *Node* is used to visualize a *Node*. A *Node* may have several *DisplayNames*; they are language dependent and may include a language reference. Similarly, a *Node* may have language dependent descriptions.

The optional property *WriteMask* can be used to define if the *DisplayName* or *NodeID* may be changed at run time. The property *UserWriteMask* defines the rights of a connected client or user. The *RolePermissions* is an array of access rights of type *RolePermissionType*. These rights refer to the services of the server, e.g. browsing, deletion or creation of *Nodes*, read and write of historic data.

[1] *https://opcfoundation.org/UA/schemas/1.04/Opc.Ua.NodeSet2.xml*

NodeClass	Usage
Object	Represent an object made up of variables, methods and further objects; the object may be a system, system component, real-world object or software object.
ObjectType	Define requirements for Object Nodes (contained variables, methods, etc.)
Variable	Represent a value (scalar, array, multi-dimensional array)
VariableType	Define requirements for Variable Nodes (value type, array size, etc.)
DataType	Define simple and structured data types
ReferenceType	Define a type for references between nodes (hierarchical or non-hierarchical)
Method	Define callable remote procedures
View	Provide access to a subset of nodes

Table 5: NodeClass types in OPC UA

The OPC UA information model has 8 *Node* types (*NodeClass*) as defined in OPC UA Part 3.

The informative Annex B of OPC UA Part 3 describes the OPC UA Meta Model as UML classes. This covers the meta-models of the Base model of *Nodes*, *ReferenceTypes*, *Attributes*, *Object* and *ObjectType*, *EventNotifier*, *Variable* and *VariableType*, *Method*, and *DataType*. The UML for the Base model of *Nodes* is shown below as an important part of the Metamodel.

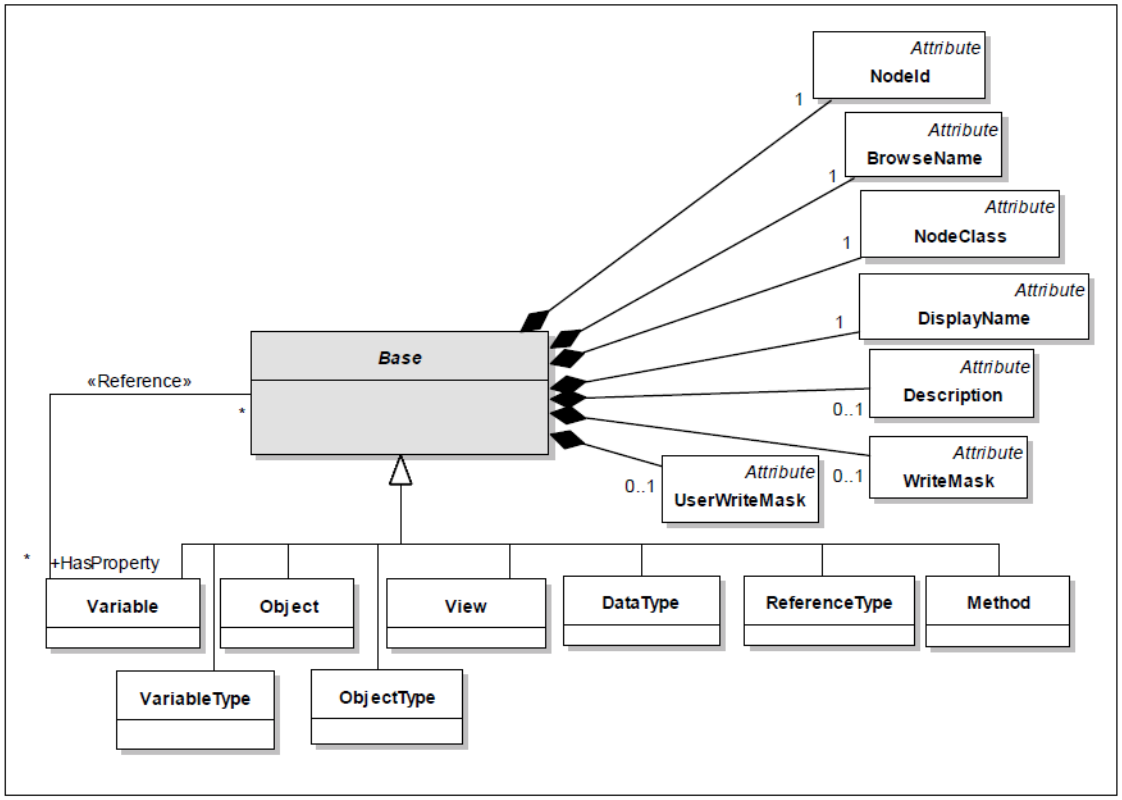

Figure 29: Base metamodel of Nodes (OPC UA Part 3, Fig. B.4)

Annex B OVERVIEW OF OGC SENSORTHINGS API

SensorThings API is a standard of the Open Geospatial Consortium OGC comprising Part 1: Sensing (OGC 15-078r6) and Part 2: Tasking Core (OGC 17-079r1):
https://www.opengeospatial.org/standards/sensorthings

The Smart Factory Web Testbed uses Part 1 Sensing. The entities in the SensorThings API Part 1 ontology are shown in Figure 30.

A '*Thing*' is the central entity in the data model. It can be physical or virtual, and is equipped with one or more 'Sensor' to collect Observations. Depending on the use case this can be the object being observed, or the sensor platform, such as a satellite.

A *Thing* is defined to be an object of the physical world (physical things) or the information world (virtual things) that is capable of being identified and integrated into communication networks.

- a Thing may have a Location and several Location representations, e.g. latitude/longitude, street address or reference to a building section,
- a Thing may have HistoricalLocations to trace previous locations (useful in case of a moving Thing),
- a Thing has zero-to-many DataStreams and
- a Thing has a freely definable set of properties which can be used to add semantic information about the Thing. The properties are encoded as a JSON object containing user-annotated properties as key-value pairs.

Each Observation has a FeatureOfInterest that it observes, as well as a Datastream.

Datastream:

- A *Datastream* groups a collection of Observations measuring the same phenomenon (called the *ObservedProperty*) and produced by the same Sensor for the same Thing.
- observationType (e.g. category, count, measurement, truth,..)
- observedArea (bounding polygon of coordinates)
- resultTime (The temporal interval of the result times of all observations belonging to this Datastream in ISO 8601 encoding)
- unitOfMeasurement (expressed as a Unified Code for Unit of Measure (UCUM), e.g. degree Celsius)
- additional properties with metadata (in v1.1 of the standard under approval)

Observation:

- an *Observation* is the act of measuring or otherwise determining the value of an *ObservedProperty*. An *Observation* results in a value being assigned to a phenomenon. The phenomenon is a property of a feature, the latter being the *FeatureOfInterest* of the *Observation*. The concept *Observation* is according to the Observations and

Measurements (O&M) model [OGC 10-004r3[1] and ISO 19156:2011 Geographic information—Observations and measurements].

- result (estimate of property value, e.g. temperature, wind speed and direction)
- phenomenonTime (time instance or interval when the observation was made)
- resultTime (when the result was generated)
- resultQuality may be defined to specify accuracy or validation status
- validTime (The time period during which the result may be used)
- parameters (used to make observation)
- Refers to exactly one *FeatureOfInterest*. In IoT, *FeatureOfInterest* is typically the Location of the *Thing*, and *Thing* is carrier of sensor. For example, the *FeatureOfInterest* of a wifi-connected thermostat can be the Location of the thermostat (e.g. the living room where the thermostat is located). However, it is also permissible to consider the *FeatureOfInterest* to be an object to be observed (e.g. machine, robot, lake, etc.) and the *Thing* to be the (mobile) carrier of a sensor such as a camera.

A SensorThings API server provides RESTful services for Create, Read, Update and Delete operations. Clients use the protocols HTTP POST, GET, PATCH, DELETE and MQTT to access the database on the server. There is also a notification of updates with a MQTT extension. The server supports enhanced access through advanced queries, e.g. spatial-temporal and filter queries, pagination (to return a subset of a large data set).

[1] *https://www.opengeospatial.org/standards/om*

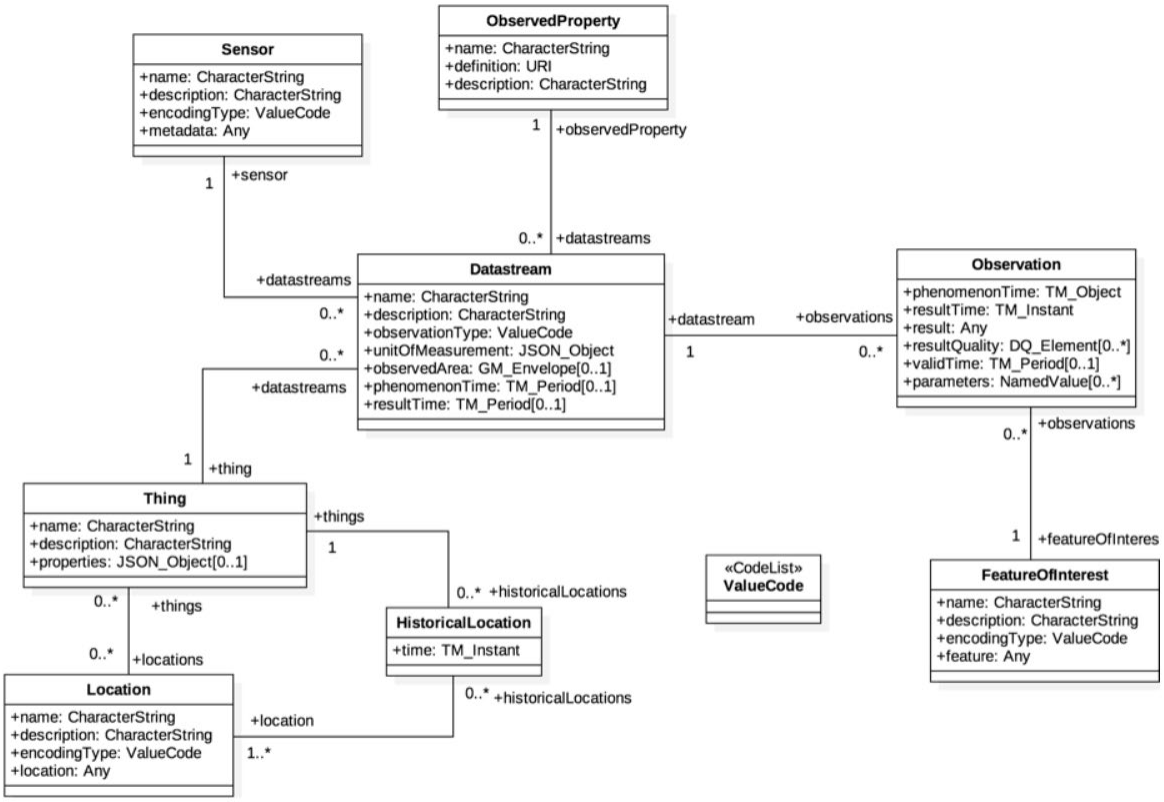

Figure 30: Entities in SensorThings API Part 1 [OGC 2016]

Annex C ASSETS IN I4.0

Plattform Industrie 4.0 published its "Details of the Asset Administration Shell –Part 1: The exchange of information between partners in the value chain of Industrie 4.0 (Version 2.0)" [I4.0 2019] in November 2019. This document includes a mapping of the AAS (Asset Administration Shell) to AML; cf. also [AML 2019]. There are also mappings to OPC UA and JSON. Future releases of Smart Factory Web will take these specifications into account.

The document [I4.0 2019] defines an asset to be "physical or logical object owned by or under the custodial duties of an organization, having either a perceived or actual value to the organization."

An asset is represented in the virtual world by its administration shell and may have multiple virtual representations, i.e. multiple administration shells. An asset can be an idea, software program, an archive, service or physical item. An asset has a lifetime and a clearly defined identity.

In I4.0 the hierarchical levels in the framework RAMI4.0 are (from top to bottom): Connected World, Enterprise, Work Centre, Station, Control Device supplemented by Field Device and Product, cf. [DIN 2016A] and Figure 31. The levels Connected World and Product go beyond the classic factory levels in the automation pyramid and represent a key aspect of I4.0.

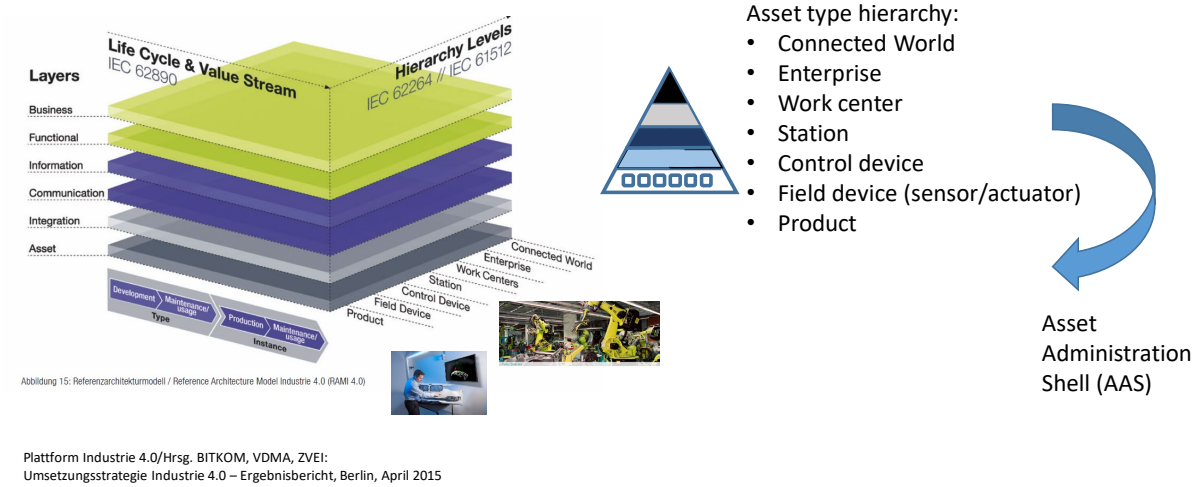

Figure 31: Reference Architecture Model Industrie 4.0 (RAMI4.0) [DIN 2016a]

An Industrie 4.0 Component is the combination of an Asset and its Administration Shell (a logical representation of the Asset) [DIN 2016A].

Capability is defined in [I4.0 2019] as the "implementation-independent potential of an Industrie 4.0 component to achieve an effect within a domain."

For an overview of the standards used in I4.0, see *http://i40.semantic-interoperability.org/*. The standards include OPC UA and AML, both of which play a central role in the SFW.

C.1 SUB-MODELS AND PROPERTIES

In I4.0, the AAS may comprise several sub-models, each with a structured set of properties (in a hierarchy): “Each sub-model contains a structured quantity of properties that can refer to data and functions. A standardized format based on IEC 61360-1/ ISO 13584-42 is envisaged for the properties. Thus, property value definition shall follow the same principles as also ISO 29002-10 and IEC 62832-2. Data and functions may be available in various, complementary formats.”

[I4.0 2019] on the concept of properties: "The IEC 61360 series provides a framework and an information model for product dictionaries. The concept of product type is represented by 'classes' and the product characteristics are represented by 'properties.' Such properties are standardized data elements. The definitions of such properties can be found in a range of repositories, such as IEC CDD (common data dictionary) or eCl@ss. The definition of a property (aka standardized data element type, property type) associates a worldwide unique identifier with a definition, which is a set of well-defined attributes. Relevant attributes for the Administration Shell are, amongst other, the preferred name, the symbol, the unit of measure and a human-readable textual definition of the property.”

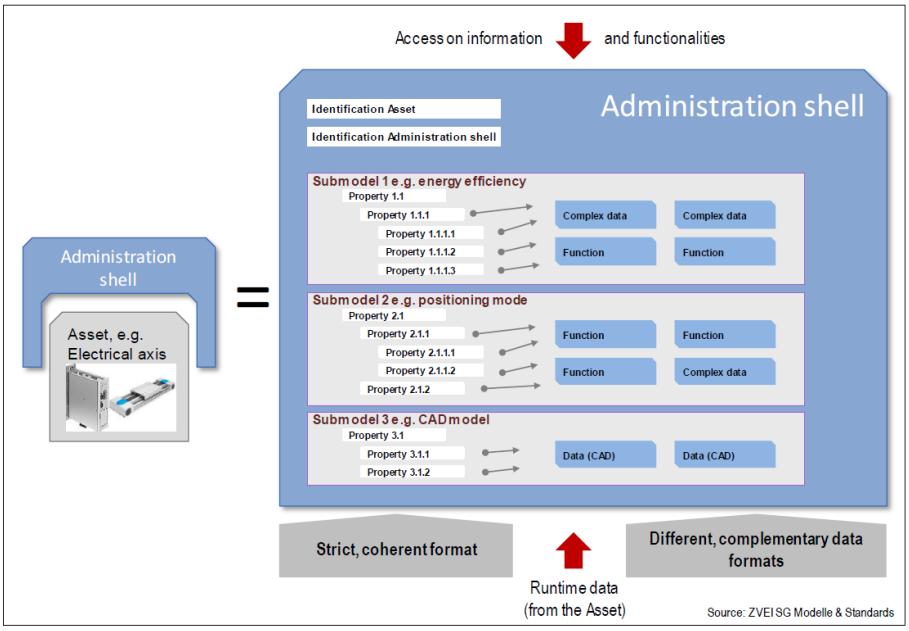

Figure 32: Basic structure of the AAS [I4.0 2019, Fig. 132]

Currently, only the meta-model for sub-models has been specified. In the future it is expected that the sub-models will be standardized to make them reusable and provide semantic interoperability.

The AASX Package Explorer is a tool designed to create and maintain an AAS of type 1 (so called passive AAS in file format). It is an open source software, which is licensed under Eclipse Public License 2.0 and can be downloaded at *https://github.com/admin-shell/aasx-package-explorer*. The AASX Package Explorer can export and import AAS models in several serialization formats: XML, JSON, RDF, OPC UA and AML.

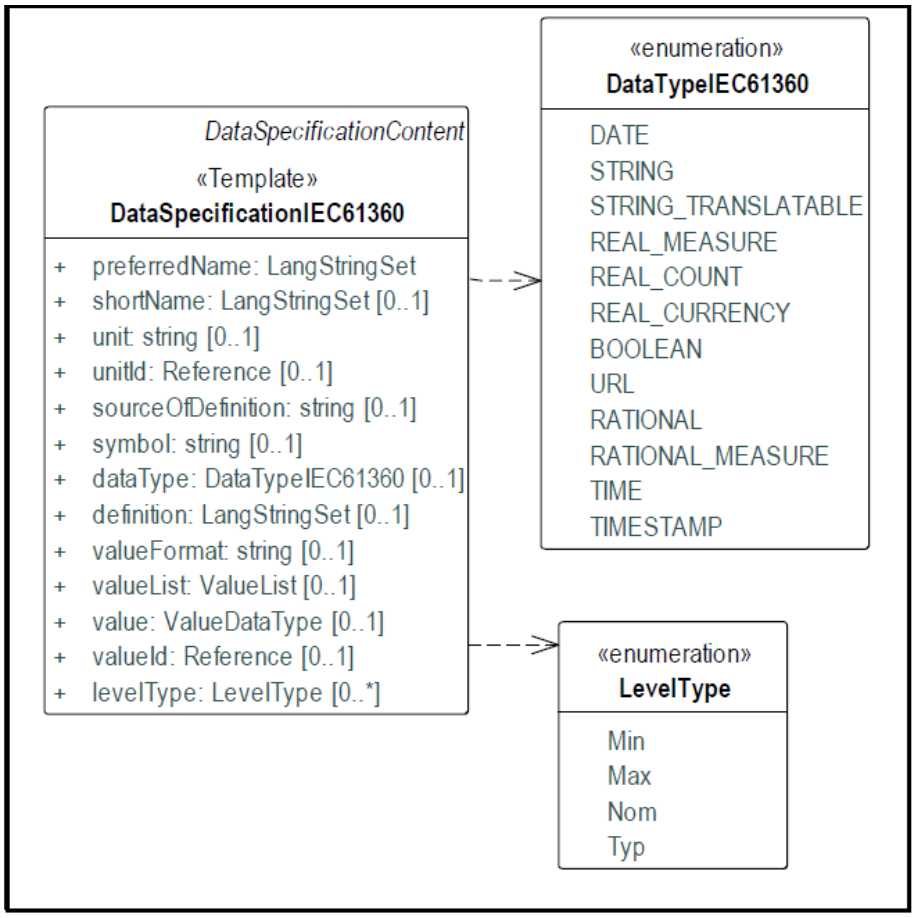

Figure 33: Data Specification Template in I4.0 AAS [I4.0 2019, Fig. 44]

C.2 IDENTIFIERS

Identifiers are one of the focal subjects in [I4.0 2019]. They are defined for AAS, assets, sub-models (instances and templates), property definitions and concept descriptions in external repositories such as eCl@ss and IEC CDD. Their dual fold purpose is to uniquely distinguish elements of an AAS and to bind semantics to these elements. This approach supports the semantic interoperability of applications.

For example, the Property “Number per minute” has the eCl@ss identifier 0173-1#02-AAT096#001 as an IRDI (International Registration Data Identifier).

As a second example, the Property ‘Max. rotation speed’ (e.g. of a motor) has the eCl@ss IRDI identifier 0173-1#02-BAA120#008. It is defined to be the greatest permissible rotation speed with which the motor or feeding unit may be operated and has the unit of measure 1/min.

Annex D ACRONYMS

AAS	Asset Administration Shell (of Platform Industrie 4.0)
AML	AutomationML (Automation Markup Language)
AMQP	Advanced Message Queuing Protocol
CAEX	Computer Aided Engineering Exchange format of IEC 62424
CDD	Common Data Dictionary (of IEC)
CEP	Complex Event Processing
COLLADA	Collaborative design activity
DIN	Deutsches Institut für Normung e.V. (*https://www.din.de/en*)
FROST	Fraunhofer Open Source SensorThings API Server
I4.0	Industrie 4.0
IEC	International Electrotechnical Commission (*https://www.iec.ch/*)
IETF	Internet Engineering Task Force
IIoT	Industrial Internet of Things
IIRA	Industrial Internet Reference Architecture
IRDI	International Registration Data Identifier
JSON	JavaScript Object Notation
MQTT	Message Queuing Telemetry Transport
OGC	Open Geospatial Consortium
OPC	Open Platform Communications
OPC UA	OPC Unified Architecture
O&M	Observation and Measurement Model
SFW	Smart Factory Web
SFWP	Smart Factory Web Portal
TCP	Transmission Communication Protocol
TLS	Transport Layer Security
XML	Extensible Markup Language

Annex E REFERENCES

[AML 2017a] AutomationML: Best Practices Recommendation: DataVariable, BPR DatVar, V1.0.0, 2017-May, *https://www.automationml.org/o.red/uploads/dateien/1494589220-BPR_007E_BPR_DataVariable_V1.0.0.zip*

[AML 2017b] AutomationML: Whitepaper AutomationML and eCl@ss integration, V 1.0.1, 2017-December, *https://www.automationml.org/o.red/uploads/dateien/1513936451-WP_AutomationML_and_eClass_integration_V1.0.1.pdf*

[AML 2018] AutomationML Association: Whitepapers on IEC 62714 Parts 1-4, Engineering data exchange format for use in industrial automation systems engineering - Automation markup language *https://www.automationml.org/o.red.c/publications.html*

[AML 2019] Application recommendation: AAS Representation, AR AAS, AutomationML consortium, V1.0.0, 2019-November https://www.automationml.org/o.red/uploads/dateien/1574258013-AR_004E%20-%20Assset%20Administration%20Shell%20Representation%20V1_0_0.zip

[DIN 2003] DIN 8580:2003-09: Combining OPC Unified Architecture and Automation Markup Language, 2016-December *https://www.beuth.de/de/norm/din-8580/65031153*

[DIN 2016a] DIN SPEC 91345: Reference Architecture Model Industrie 4.0 (RAMI4.0), 2016-April *https://www.beuth.de/de/technische-regel/din-spec-91345/250940128*

[DIN 2016b] DIN SPEC 16592-2016: Manufacturing processes - Terms and definitions, division, 2003-March *https://www.beuth.de/de/technische-regel/din-spec-16592/265597431*

[DIN 2018] DIN SPEC 16593-1:2018: RM-SA –Reference Model for Industrie 4.0 Service Architectures –Part 1: Basic Concepts of an Interaction-based Architecture, 2018-April *https://www.beuth.de/de/technische-regel/din-spec-16593-1/287632675*

[Hey 2018] Heymann, Sascha; Stojanovic, Ljiljana; Watson, Kym; Nam, S.; Song, B.; Gschossmann, H.; Schriegel, Sebastian; Jasperneite, Jürgen: Cloud-based Plug and Work architecture of the IIC Testbed Smart Factory Web. International Conference on Emerging Technologies and Factory Automation (ETFA), Torino, Italy. ISBN: 978-1-5386-7108-5, p187-194, 2018-September.

[I4.0 2019] Details of the Asset Administration Shell –Part 1: The exchange of information between partners in the value chain of Industrie 4.0 (Version 2.0), 2019-November
https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Details-of-the-Asset-Administration-Shell-Part1.html

[IEC 2013] IEC: IEC 62264-1 Enterprise-control system integration – Part 1: Models and terminology, Edition 2, 2013-May.

[IEC 2016] IEC: IEC 62724:2016, Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools

[IEC 2018] IEC: IEC 62714 Parts 1-4, Engineering data exchange format for use in industrial automation systems engineering - Automation markup language

[IEC 2019] IEC: IEC 61131-10:2019, Programmable controllers - Part 10: PLC open XML exchange format

[IIC 2019a] IIC: The Industrial Internet, Volume G1: Reference Architecture, version 1.9, 2019-June-19
http://www.iiconsortium.org/IIRA.htm

[IIC 2019b] IIC: The Industrial Internet, Vocabulary Technical Report, version 2.2, 2019-11-06
https://www.iiconsortium.org/vocab/index.htm

[OASIS 2016] OASIS Reference Model for Service Oriented Architecture 1.0, Committee Specification 1, 2006.

[OGC 2016] Open Geospatial Consortium: OGC SensorThings API Part 1: Sensing, OGC 15-078r6, 2016-July.
http://docs.opengeospatial.org/is/15-078r6/15-078r6.html

[Okon 2019] Michael Okon, Ljiljana Stojanovic: Generation of hierarchical OPC UA-Servers from AutomationML-Models, AutomationML PlugFest, Hamburg, Germany, September 25-26, 2019

[OPC 2016] OPC 30040: Companion Standard “OPC UA for AutomationML”, 2016-February
https://opcfoundation.org/developer-tools/specifications-unified-architecture/opc-unified-architecture-for-automationml/

[Usl 2010] Usländer, T., Service-oriented Design of Environmental Information Systems, PhD thesis of the Karlsruhe Institute of Technology (KIT), Faculty of Computer Science, KIT Scientific Publishing ISBN 978-3-86644-499-7
https://publikationen.bibliothek.kit.edu/1000016721/1317767

[UslEpp 2015] Usländer, T; Epple, U: Reference model of Industrie 4.0 service architectures — Basic concepts and approach, in: at – Automatisierungstechnik 63(10): 858–866, 2015.

AUTHORS AND LEGAL NOTICE

This document is a work product of the Industrial Internet Consortium Testbed Working Group, chaired by Howard Kradjel (IIC staff).

AUTHORS

The following persons contributed substantial written content to this document:

Kevin Becker, Sascha Heymann, Michael Okon, Boris Schnebel, Felix Schöppenthau, Ljiljana Stojanovic and Kym Watson (all with Fraunhofer IOSB).

CONTRIBUTORS

The following persons contributed valuable ideas and feedback that significantly improved the content and quality of this document:

Thomas Usländer (Fraunhofer IOSB)

TECHNICAL EDITOR

Stephen Mellor (IIC staff) oversaw the process of organizing the contributions of the above Authors and Contributors into an integrated document.

IIC ISSUE REPORTING

All IIC documents are subject to continuous review and improvement. As part of this process, we encourage readers to report any ambiguities, inconsistencies or inaccuracies they may find in this Document or other IIC materials by sending an email to *admin@iiconsortium.org*.