A formal mapping between OPC UA and the Semantic Web

Rainer Schiekofer[1,2], Stephan Grimm[1], Maja Milicic Brandt[1], and Michael Weyrich[2]
[1]Siemens AG [2]University of Stuttgart

***Abstract*—The communication protocol OPC UA is one of the most important IIoT enablers within the automation domain. OPC UA not only aims to provide interoperability on the transport layer, but also interoperability of the semantic layer shall be addressed based on so-called Companion Specifications. However, the lack of OPC UA formal semantics makes automatic validation of OPC UA data models impossible. Another drawback is the shortage of available tools for OPC UA, such as an implementation of the query engine for the specified OPC UA query language. In this paper we provide a formal translation of OPC UA models to the Semantic Web standard OWL, thus making OPC UA implicit semantics, that is described in the documentation, explicit, by means of OWL axioms. Moreover, we outline how this mapping can be used to offer validation and querying of OPC UA data models based on already existing Semantic Web technology.**

***Index Terms*—OPC UA, OWL, Mapping, Query, Validation**

I. Introduction

In the area of factory automation OPC Unified Architecture (OPC UA) [1] is the new standard that is promised to lift field device communication from low-level signal exchange schemes onto a semantic level, contributing to the realization of flexible manufacturing scenarios within the Industry 4.0 vision. OPC UA is a machine to machine communication protocol for industrial automation developed by the OPC Foundation. However, despite all the improvements that OPC UA brings over conventional device communication, it still exhibits certain problems when it comes to capturing the semantics of m2m communication structures: much of the semantics of the OPC UA basic constructs is defined in specification documents in an implicit way, only accessible to the human implementor. Moreover, the relatively new OPC UA specifications also lack implementation in available tools, which now just start to emerge.

On the other hand, Semantic Web [2] technology is state-of-the-art for representing and processing explicit semantics for data models in information systems in general, and specifically for the web. The standardized ontology languages RDF(S) and OWL provide a representation framework for formulating semantically rich knowledge graphs, while auxiliary standards like SPARQL or REST provide means to communicate and query such information. Being supported by an active research community for some years now, it also offers an established set of tools that support these standards. Hence, it appears to be natural to investigate the use of the already matured Semantic Web technology stack for the relatively new OPC UA standard for capturing semantics more formally and for the reuse of Semantic Web tools for tasks like querying or validation.

There are various uses cases in the automation world for which this combination of Semantic Web technology and OPC UA is promising. For accessing data from field devices described by means of OPC UA information models, the OPC foundation introduced OPC UA Query as a mechanism for performing a filtered search for information in the device server's address space. But since OPC UA is still rather new there is not yet an implementation of OPC UA Query available. To fill this gap, the readily available SPARQL language and tool stack could be used to query data from OPC UA information models when bridged to an RDF-based representation. Furthermore, validation of OPC UA information models against the specification is critical for producing high-quality data models that adhere to the OPC UA specification. Current tools already implement some validation checks covering certain aspects of the specification as part of their modeling UIs, but a broader coverage for validation could be reached by formulating OPC UA validity rules by means of ontology languages in a declarative way and use reasoning tools to automate and unify validation. In this way, validation could even be extended to the inclusion of custom rules that come from so-called *Companion Specifications* for specific industries.

Under these circumstances, it is surprising that only a few researchers tried to connect OPC UA to the Semantic Web so far. For example, the authors of [3] describe an approach how OPC UA can be integrated into a Linked Data environment. However, the authors seem to define a new ontology for OPC UA and also do not use most of the built-in OPC UA concepts like the type concept. Another interesting approach with a focus on the reversed mapping direction from OWL to OPC UA is described in [4]. The problem with this approach is that it does not define mappings for all OPC UA concepts like *ReferenceTypes*. Because of that, this mapping cannot be used to lift the rich semantics of OPC UA *Companion Specifications* up into the Semantic Web, mainly due to the fact of the missing concept transformations, which are heavily used in most of the *Companion Specifications*. The authors of [5], [6] developed a concept how OWL-S [7] can be introduced to OPC UA *Methods*. Nevertheless, because this approach only covers the *Method-NodeClass* of OPC UA and does not address the other *NodeClasses* only about 5% of the *Nodes* defined by most *Companion Specifications* can be covered.

As the primary contribution of this paper, we present a

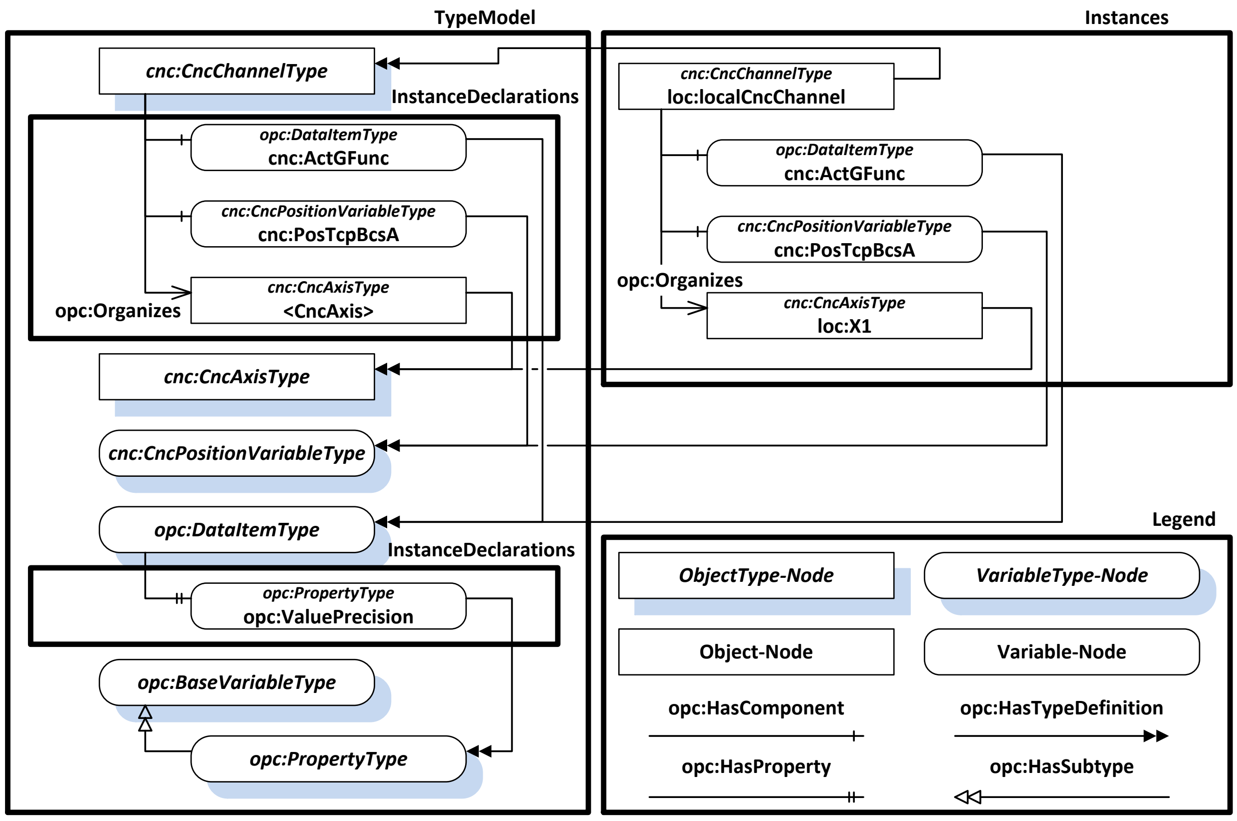


Fig. 1. An example OPC UA information model for CNC machines.

formal mapping between OPC UA information models and the OWL/RDF(S) languages. This shows in particular how any OPC UA-based model can be represented in RDF/OWL as a basis for automated transformation. Furthermore, we exemplify how this formal bridge between OPC UA and RDF/OWL can support the two use cases of querying as well as validating OPC UA information models. This is showing the potential of using Semantic Web technology in the emerging activities of building up semantic information models for device communication in the domain of industrial automation.

II. Background

In the next two sections, we will give a brief overview of OWL and OPC UA.

A. Web Ontology Language

The Web Ontology Language (OWL) [8] is a W3C standard that provides ontological constructs for knowledge representation. OWL is built on top of other W3C standards such as RDF [9] and RDF(S) [10]. OWL semantics is grounded in Description Logics [11], which are decidable fragments of the first-order-logic.

The basic entities in OWL are the following: OWL individuals denote objects (e.g., anna and john); OWL classes denote classes of objects (e.g., Female, Person, Engineer); OWL object properties relate objects to objects (e.g., relating a child to its parent with hasChild); OWL data properties assign data values to objects (e.g. relating a height to a person); and OWL annotation properties to record ontology meta-information, such as the author and creation date. Moreover, OWL provides constructs for building complex class expressions, such as: conjunction (Female and Engineer), disjunction (Engineer or Doctor), existential restrictions (a class of Persons who have at least one child), etc; property expressions (inverse, chains, etc). OWL axioms, such as class subclass axioms (Female subClassOf Person), class disjointness axioms (Female disjointWith Male), class equivalence axioms (Person equivalent (Male union Female)) are used to axiomatize the domain. Moreover, there are property axioms which may specify property domain and range, sub-property relationships, property equivalence and disjointness, and relationships between property chains. Finally, individual axioms make statements about individuals (type assertions, property assertions, same individuals, different individuals). OWL Reasoners [12], [13] can automatically perform reasoning tasks such as checking consistency and inferring implicit relationships.

Moreover, the W3C recommendation SHACL (Shapes Constraint Language) [14] specifies a constraint language over RDF data; there is a number of available SHACL implementations [15]. RDF(S) datasets can be accessed via the W3C standard protocol and query language SPARQL [16].

B. OPC UA

Open Platform Communication Unified Architecture (OPC UA) [1] is one of the most promising industrial communication standards for Industry 4.0 scenarios. OPC UA aims to solve the two most important problems of typical IIoT scenarios, which are interoperability on the transport and semantic layer.

TABLE I
CncChannelType definition (see also [22]).

Attribute	**Value**			
Browse-Name	CncChannelType			
IsAbstract	False			
References	**BrowseName**	**DataType**	**TypeDefinition**	**MRule**
Subtype of the CncComponentType				
HasComp.	ActGFunc	String[]	DataItemType	Mand.
HasComp.	PosTcpBcsA	CncPosDT	CncPosVarT	Mand.
...	...	...	...	...
Organizes	<CncAxis>		CncAxisType	OPlaceh.
NOTE: This row represents no Node in the AddressSpace. It is a placeholder pointing out that instances of the ObjectType will have those Objects.				

Interoperability on the transport layer can be achieved through the standardization of different transport protocols like OPC TCP and HTTP(S) in combination with different encodings like OPC JSON, OPC Binary, or OPC XML. Recently the OPC Foundation also offers a cloud-ready interface based on the well-known publish-subscribe pattern, using transport protocols like MQTT and AMQP. However, only standardizing the messages for various transport protocols is not enough. The second step to reach the goal of interoperability on the transport layer is the standardization of interaction patterns with the service. In OPC UA these interaction patterns are called *Services*, which allow a client to access the graph-based data model of OPC UA. Several services were defined to introspect and manipulate the data model (e.g., the *Read* service, for fetching data and the *Write* service for manipulating data).

Interoperability on the semantic layer can be reached by the graph-based data model of OPC UA in combination with *Companion Specifications*. In previous years most of the *Companion Specifications* were mappings from other existing standards to OPC UA like AutomationML, PLCOpen, ISA-95, etc. [17]–[19]. All of these standards are generic and try to solve the problem of semantic interoperability on an abstract layer. Eventually, these standards can only be considered as the first step towards semantic interoperability. For example, standardizing the notion of a ”Thing” and a concept how ”Skills” of these ”Things” shall be exposed, does not solve the issue of concrete Industry 4.0 applications like automatic skill-matching. For such applications, the semantics of concrete skills, like drilling or clamping, must also be standardized. Because of that, there exists a great demand for standardized detailed semantics for a given domain. This is exactly what is addressed by the VDMA [20]. VDMA represents more than 3200 companies within the manufacturing domain and can, therefore, be considered the largest industry association in Europe. One goal of the VDMA is to standardize domain-specific semantics for a huge part of the automation domain (e.g., machine vision, robotics, powertrain, cnc machines, etc.). Their semantics will be standardized within OPC UA *Companion Specifications* [21], leading to a whole new level of semantic interoperability in the automation domain.

III. Analysis and Problem Statement

As already mentioned in the previous sections, OPC UA offers the capability to store very rich and standardized semantics in OPC UA information models. However, despite several public requests [23], there was no attempt to fully map the OPC UA specification to a formal ontology language yet. For our use cases of interest, validation and query of OPC UA data models, one would greatly benefit from such a mapping, as it would allow to apply relevant existing tools from the OWL/RDF(S) ecosystem, rather than having to re-invent them in the OPC UA world. The mapping we describe in Section IV aims to be first such comprehensive mapping from OPC UA to OWL. One essential requirement for this approach is that OPC UA restrictions and semantics (implicit and explicit) are transformed correctly to a formal ontology language like OWL. Only if the concepts match on both sides, all benefits of Semantic Web technology can be unleashed and used to solve our use cases. In the following, we will start with a short analysis of OPC UA semantics, revealing some of the issues which are preventing a trivial mapping to OWL. The basic building blocks of OPC UA are eight different types of *Nodes* (categorized in so-called *NodeClasses*). An example of such a *NodeClass* is the *Object-Node*, which can be used to represent a ”Device”, while a *Variable-Node* could be a concrete value (e.g., serial number) of the Device. The connection between different *Nodes* is done by so-called *References*.

Figure 1 shows an example information model of OPC UA based on [22]. The left side of the picture contains a small fracture of a typical OPC UA *TypeModel*, which is provided by the *Companion Specifications*. Several *Type-Nodes* are defined, for example, a ”CncChannelType”, a ”CncAxisType” and also a ”DataItemType”. These *Types* are defined with special *NodeClasses* in OPC UA (e.g., *ObjectType-Node* and *VariableType-Node*). Each *Instance-Node* (Figure 1 right side) references its *TypeDefinition* with a special *ReferenceType* named ”HasTypeDefinition”. While *Type-Nodes* (e.g., *VariableType*, *ObjectType*, etc.) can be subtyped and therefore inherit semantics and restrictions from the supertype, the same is not true for *Instance-Nodes* (e.g., *Variable*, *Object*). However, surprisingly we can also find *Instance-Nodes* in the *TypeModel* of OPC UA. These special *Instance-Nodes* are also called *InstanceDeclarations* within OPC UA. An *InstanceDeclaration* is a *Node* which is defined in the context of a *Type-Node* and is used to model the sub-structure of a *Type*. The sub-structure of the CncChannelType is depicted in Table I. Each *InstanceDeclaration* is defined by several characteristics. (1) The *ReferenceType* interconnects the defining *Type-Node* with the *InstanceDeclaration*. It is also allowed to use a subtype of the concrete *ReferenceType*. (2) The expected *NodeClass* and *BrowseName* (a *BrowseName* is a string with a *NamespaceURI* assigned to it, for example, ”http://opcfoundation.org/UA/CNC/” or in short ”cnc:”, as *NamespaceURI* and ”CncChannelType” as string part), which must be identical on each *Instance-Node*. (3) The *DataType* (if applicable) and *TypeDefinition* (if applicable). Also in this case subtypes are allowed. Finally, the corresponding *ModellingRule* is assigned. OPC UA defines several *ModellingRules* and also allows to define new *ModellingRules*, if necessary. The *Mandatory-ModellingRule*, for example, en-

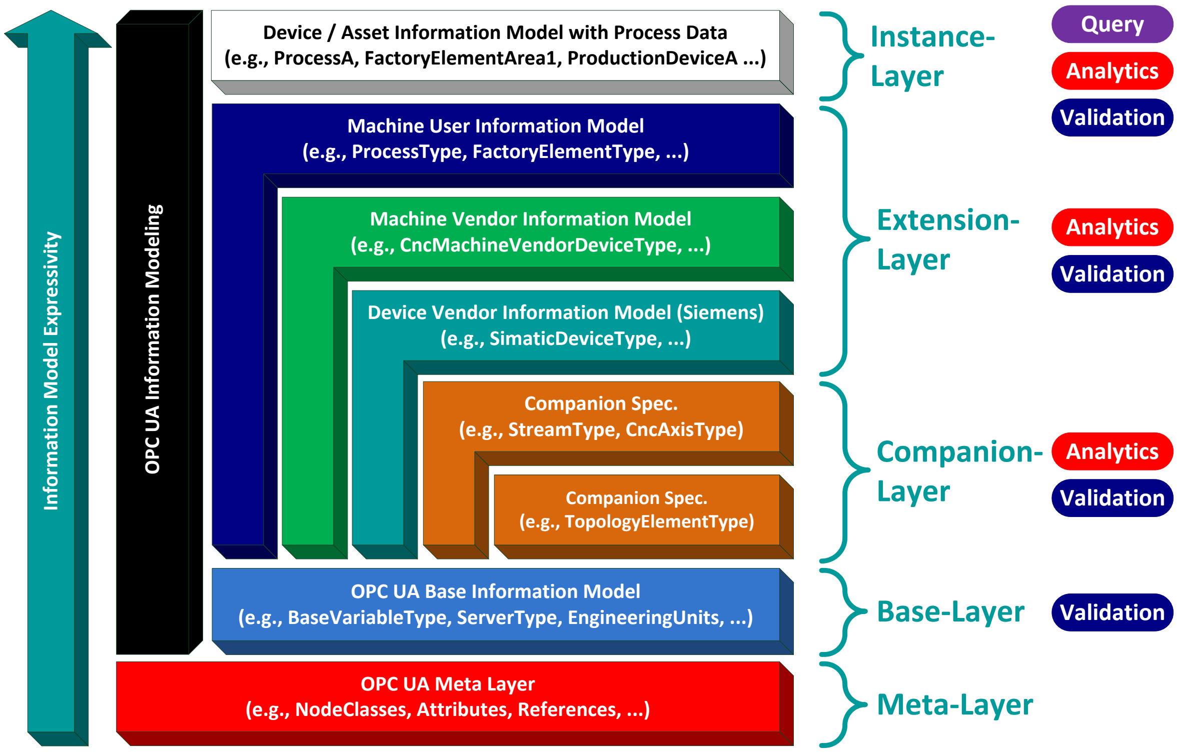

Information Model Expressivity
OPC UA Information Modeling
Device / Asset Information Model with Process Data
(e.g., ProcessA, FactoryElementArea1, ProductionDeviceA ...)
Machine User Information Model
(e.g., ProcessType, FactoryElementType, ...)
Machine Vendor Information Model
(e.g., CncMachineVendorDeviceType, ...)
Device Vendor Information Model (Siemens)
(e.g., SimaticDeviceType, ...)
Companion Spec.
(e.g., StreamType, CncAxisType)
Companion Spec.
(e.g., TopologyElementType)
OPC UA Base Information Model
(e.g., BaseVariableType, ServerType, EngineeringUnits, ...)
OPC UA Meta Layer
(e.g., NodeClasses, Attributes, References, ...)
Instance-Layer
Extension-Layer
Companion-Layer
Base-Layer
Meta-Layer
Query
Analytics
Validation
Analytics
Validation
Analytics
Validation
Validation

Fig. 2. Overview of OPC UA Information Modeling.

forces, that each *Instance-Node* of the CncChannelType must reference a *Node* similar to the corresponding *InstanceDeclaration*. Similar in this case means, the same *NodeClass* and *BrowseName* combined with the same *DataType* and *TypeDefinition* or a subtype (if applicable), referenced by the defined *ReferenceType* or a subtype of it. Similarly, there are *Optional-ModellingRules*, stating that *InstanceDeclarations* are not compulsory on instance level. The *Placeholder-ModellingRule* is used if the *BrowseName* of the *Instance-Node* is not defined within the *Companion Specification* and can be freely chosen for *Instance-Nodes* (only in combination with *Variables* or *Objects*).

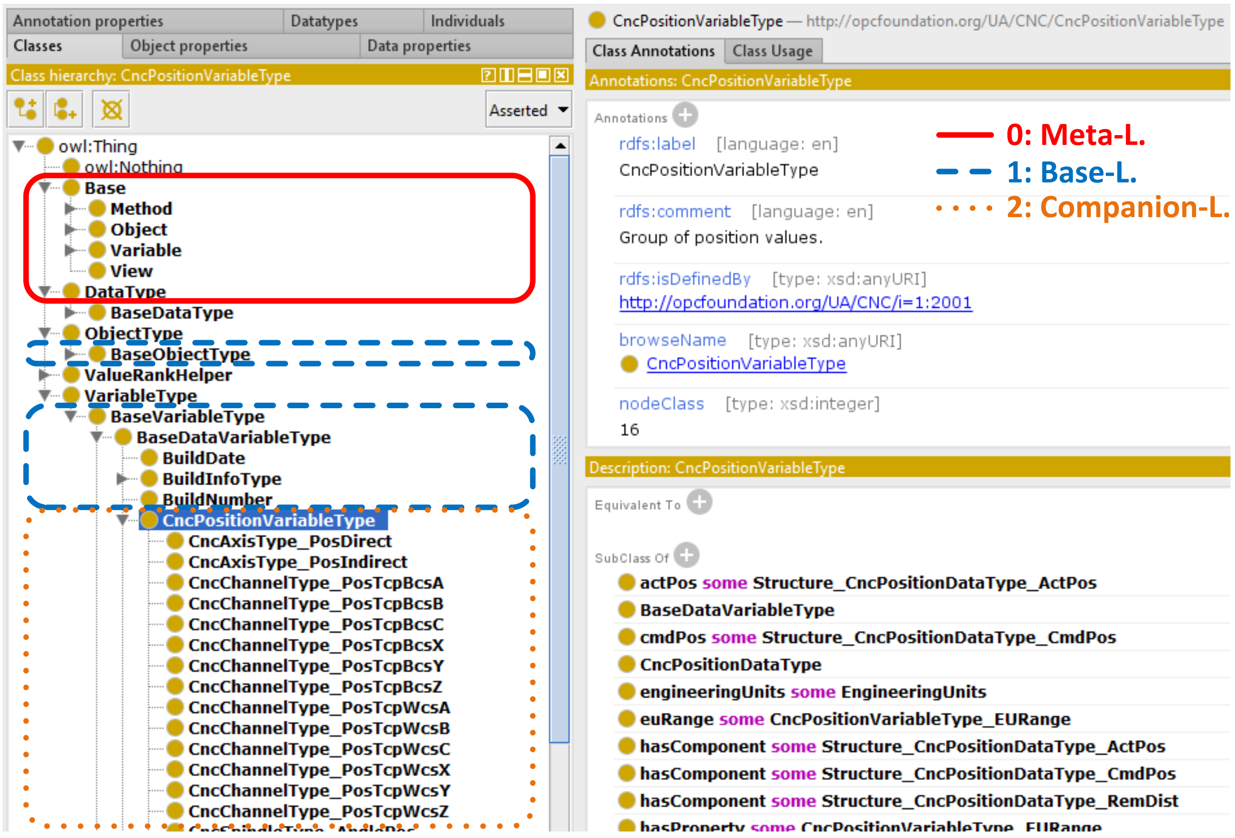
A typical *Companion Specification* also describes the semantics of these *InstanceDeclarations* in textual form and in machine-readable form (OPC UA *NodeSet*). In case of the ActGFunc *InstanceDeclaration* from CncChannelType this is done in the following way [22]: **ActGFunc:** "Array of active G functions; there can be several G functions active at a time (modal and non-modal G functions).";

As shown above, the semantics of such *Companion Specifications* is very rich and can be used for use cases like monitoring applications, which are able to find the necessary data-points automatically based on standardized semantics. However, as also depicted in Figure 1, OPC UA has some special modelling practices. For example, just consider the fact that the semantics of the ActGFunc *Instance-Node* (right side of Figure 1) is defined by the *InstanceDeclaration* (left side of Figure 1), but the *Instance-Node* specifies the DataItemType as its *Type* and not the *InstanceDeclaration*. While the connections between *InstanceDeclarations* and *Instance-Nodes* are pretty obvious for each OPC UA expert, a typical Semantic Web expert would probably not have guessed these implicit connections. Note that, there is no other direct *Reference* between an *InstanceDeclaration* and an *Instance-Node*, the connection is implicitly made through the identical *BrowseName*. Of course, OPC UA defines further rules to ensure that this concept always can be applied. For example, it is forbidden to define two identical *BrowseNames* for *InstanceDeclarations* in the context of the same *Type*. In contrast, a Semantic Web expert would have probably expected a new *VariableType*, which is a subtype of DataItemType and is referenced by the *HasTypeDefinition-ReferenceType* of an *Instance-Node*. Exactly such modelling artefacts have prevented a trivial mapping from OPC UA to a formal language like OWL till now because, for each OPC UA design pattern, the corresponding concept and transformation rule must be identified.

Besides the concepts which already were mentioned above, we identified several additional concepts to express semantics in OPC UA: *ReferenceTypes*, *DataTypes* and some *Attributes*. However, because of space-limitations, we will not go deeper into details.

IV. OPC UA to OWL

In this Section, we will give an insight into our mapping from OPC UA to OWL. The full mapping will be contributed later on directly to the OPC Foundation. In Table III we outlined the most essential mapping rules for transforming *Variable-Nodes* to OWL. The left side of Table III lists


Annotation properties
Datatypes
Individuals
Classes
Object properties
Data properties
Class hierarchy: CncPositionVariableType
Asserted
CncPositionVariableType — http://opcfoundation.org/UA/CNC/CncPositionVariableType
Class Annotations
Class Usage
Annotations: CncPositionVariableType
Annotations
Description: CncPositionVariableType
Equivalent To
SubClass Of
0: Meta-L.
1: Base-L.
2: Companion-L.

```
owl:Thing
    owl:Nothing
    Base
        Method
        Object
        Variable
        View
    DataType
        BaseDataType
    ObjectType
        BaseObjectType
    ValueRankHelper
    VariableType
        BaseVariableType
            BaseDataVariableType
                BuildDate
                BuildInfoType
                BuildNumber
                CncPositionVariableType
                    CncAxisType_PosDirect
                    CncAxisType_PosIndirect
                    CncChannelType_PosTcpBcsA
                    CncChannelType_PosTcpBcsB
                    CncChannelType_PosTcpBcsC
                    CncChannelType_PosTcpBcsX
                    CncChannelType_PosTcpBcsY
                    CncChannelType_PosTcpBcsZ
                    CncChannelType_PosTcpWcsA
                    CncChannelType_PosTcpWcsB
                    CncChannelType_PosTcpWcsC
                    CncChannelType_PosTcpWcsX
                    CncChannelType_PosTcpWcsY
                    CncChannelType_PosTcpWcsZ
```

rdfs:label [language: en]
CncPositionVariableType

rdfs:comment [language: en]
Group of position values.

rdfs:isDefinedBy [type: xsd:anyURI]
http://opcfoundation.org/UA/CNC/i=1:2001

browseName [type: xsd:anyURI]
CncPositionVariableType

nodeClass [type: xsd:integer]
16

- actPos some Structure_CncPositionDataType_ActPos
- BaseDataVariableType
- cmdPos some Structure_CncPositionDataType_CmdPos
- CncPositionDataType
- engineeringUnits some EngineeringUnits
- euRange some CncPositionVariableType_EURange
- hasComponent some Structure_CncPositionDataType_ActPos
- hasComponent some Structure_CncPositionDataType_CmdPos
- hasComponent some Structure_CncPositionDataType_RemDist
- hasProperty some CncPositionVariableType_EURange

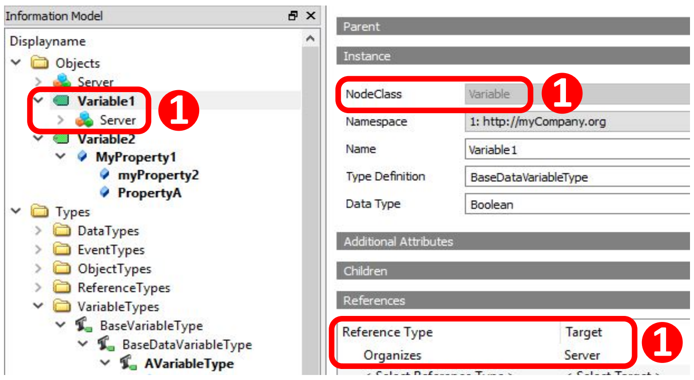
Fig. 3. Protégé-View of the generated OWL ontology.

OPC UA concepts. Note that most OPC UA concepts are defined by more than one line in the Table (marked with indents). The right side contains the OWL part of the mapping, sub-structured in OWL entities and OWL axioms [24]. The URI-schema is based on a RESTful OPC UA interface [25] with additional restrictions. In combination with this RESTful interface our mapping ensures that it will be possible to build up a Linked Data [26] network automatically for OPC UA (including *Companion Specifications* as well as vendor-specific extensions).

OPC UA information models are built in a modular way (see also Figure 2), where the OPC UA meta-model (Meta-Layer) provides the basic building blocks for information models, continuing with the OPC UA core information model (Base-Layer), which is provided by the OPC Foundation itself, followed by OPC UA companions (Companion-Layer), OEM-specific schema extensions (Extension-Layer) of OPC UA companions, and finally OPC UA instance models (Instance-Layer) that describe configuration and data items of individual devices based on schemas of the Base-Layer, Companion-Layer and Extension-Layer. The mapping transforms OPC UA information models by translation of the modules (levels Base-Layer - Instance-Layer described above) into RDF/OWL ontologies that import each other in the same way the respective OPC UA modules do. Once this transformation is performed, the resulting RDF/OWL ontologies can be used for the purpose of validation, querying, and analytics. Figure 2 also depicts the typical OPC UA information model layers for each use case.

A Protégé view of an example result OWL ontology is shown in Figure 3. The class hierarchy depicts the results of transforming the basic concepts of Meta-Layer such as *Variable*, *VariableType*, *ObjectType*, *DataType*; and the Base-Layer and Companion-Layer (sub-concepts of the respective Meta-Layer concepts, denoted with *VarT* and *DataT* in Table III:). The lower right corner of Figure 3 shows the results for transforming parts of the *Mandatory VariableInstanceDeclaration* restrictions. Notice that, in the transformation process also object properties for the *BrowseName* and classes for the *InstanceDeclarations* are generated (e.g., "actPos" and "cmdPos"). *InstanceDeclarations* with the "CncPositionVariableType" as *Type* are depicted in the lower left corner (e.g., "CncChannelType_PostTcpBcsA"). *Type-Attributes* are displayed at the upper right corner of Figure 3. Some of the *Type-Attributes* are directly mapped to some well-known annotation properties, for example, the OPC UA *NodeId* to rdfs:isDefinedBy.

The mapping rules from Table III make use of OWL annotations as appropriate, as many OPC UA *Type-Attributes* are only used for describing the *Type-Node* and should not be inherited by the *Instance-Node* of this *Type* (e.g., *DisplayName-Attribute*). The existence of such attributes cannot be validated by OWL reasoners, but with alternatives such as SPARQL. On the other hand, the instantiation rules of OPC UA and also the subtyping concept of *Types* are captured by means of OWL axioms and therefore can be validated with OWL reasoners. We should mentions that, for using OWL reasoners to validate



NodeClass	Variable
Namespace	1: http://myCompany.org
Name	Variable1
Type Definition	BaseDataVariableType
Data Type	Boolean

Reference Type	Target
Organizes	Server

Fig. 4. Failures in an information model (UaModeler-View).

OPC UA data models, a lot of additional axioms, such as disjointness axioms, are introduced into the OWL ontology. The technical report of this mapping has about 50 pages and also contains, for example, the complete mapping of OPC UA datatypes to OWL datatypes.

V. Proof-of-Concept

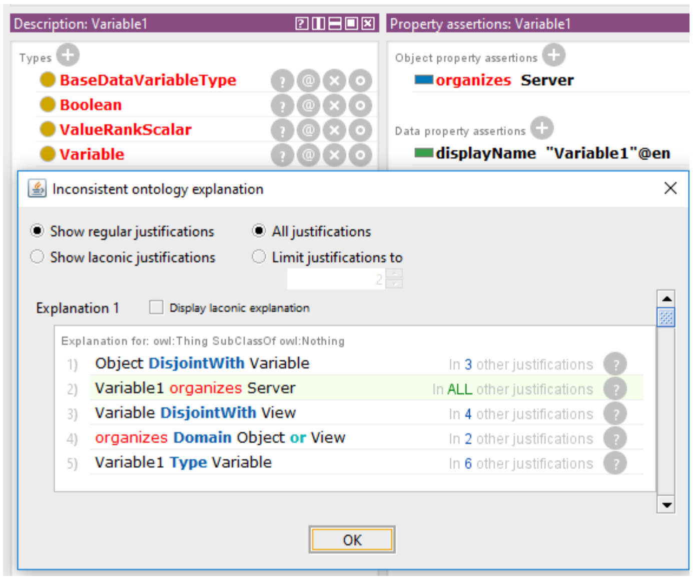
In the following Section, we will discuss two practical use cases to address some major challenges of OPC UA, which could not be solved till now. The underlying OWL ontology of our prototype is generated based on our transformation rules and has the DL expressivity $\mathcal{SROIQ}(\mathcal{D})$.

A. Validation

OPC UA defines a lot of rules regarding the structure of information models. However, currently there is no tool available which is able to validate information models for OPC UA. Most tools like SiOME [27] and UaModeler [28], only support basic checks, such as ”All types are inherited by the BaseTypes of OPC UA”, but at the end cannot be used to ensure that the designed information model is fully compliant to the OPC UA specification.

One example of an invalid OPC UA information model is displayed in Figure 4. For our use case, we modelled the following invalid statement: A *Variable-Node* which uses the *Organizes-ReferenceType* to reference another *Node* (OPC UA [1]: Allows the *Organizes-ReferenceType* to be used only with the *View-NodeClass* or *Object-NodeClass* as *Source-Node*.). Up to now, such failures cannot be detected by any consistency-check of UaModeler or SiOME.

We will now show how OPC UA information models can be validated based on our OPC UA to OWL mapping in combination with existing reasoner technology. Figure 5 depicts how the problem of Figure 4 can be addressed. This restriction can be modelled easily through the usage of OWL domain and range restrictions in combination with the disjoint concept and some further concepts, which are a part of our mapping. Nevertheless, not each rule in OPC UA can be checked easily with only OWL reasoners. This is mainly due to the fact that OWL reasoning uses the so-called open-world assumption (OWA). For example, a mandatory OPC



organizes Server

displayName "Variable1"@en

[● Show regular justifications] ○ Show laconic justifications

[● All justifications] ○ Limit justifications to 2

☐ Display laconic explanation

Explanation for: owl:Thing SubClassOf owl:Nothing

1)	Object DisjointWith Variable	In 3 other justifications
2)	Variable1 organizes Server	In ALL other justifications
3)	Variable DisjointWith View	In 4 other justifications
4)	organizes Domain Object or View	In 2 other justifications
5)	Variable1 Type Variable	In 6 other justifications

Fig. 5. Validation of OPC UA restrictions with HermiT [12] (Protégé-View).

UA property which is not present in the current graph is automatically considered as absent in OPC UA. In contrast, an OWL reasoner would not raise an exception if a mandatory property is missing, because there is no violation under OWA. Naturally, it is possible to introduce so-called closure axioms into the ontology, which can be used to add the fact that the mandatory property is not modelled anywhere else. Based on this axiom, the reasoner would now raise an exception, which is exactly the behaviour which is expected for our validation use case. Nevertheless, such rules can be checked much more efficiently with SPARQL or SHACL.

In conclusion, we showed that validation of OPC UA information models using OWL reasoners is feasible, but some further research is necessary regarding some special rules in OPC UA. We also notified the OPC Foundation about an inconsistency within the core OPC UA specification (Base-Layer) we detected. In addition, we also identified rules which cannot be checked because of missing support in the underlying reasoners. For example, OPC UA defines a *ReferenceType* restriction of ”non-looping” hierarchies. This restriction could be easily modelled through irreflexive and transitive object properties. Nevertheless, most reasoners do not support the combination of these two object property characteristics.

B. Query

The OPC Foundation specified a query language for accessing the information model of OPC UA. Nevertheless, due to the high complexity, nobody has implemented OPC UA Query until now for a publicly available product. We will now explain in greater detail how the presented mapping can be used to formulate OPC UA queries in SPARQL. As an example, which is well-known in the OPC UA universe, we chose Example B.2.4 from OPC UA Part 4. An OPC UA Query can be partitioned in two steps: First, specifying a so-called *Content-Filter*, which is used to apply certain filters;

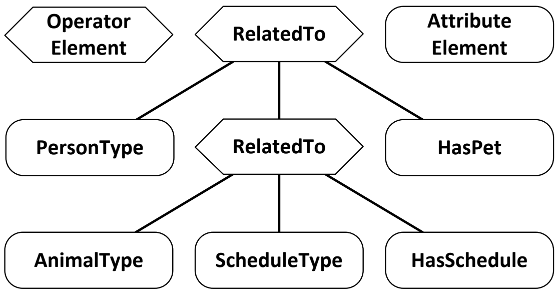


Fig. 6. OPC UA Part 4 Annex B Figure B.5 - Example B.2.4 Filter [1].

second, defining a so-called *NodeTypeDescription* for selecting the data which shall be returned.

The *Content-Filter* of Example B.2.4 can be formulated in the following way: Find all *Instances* of PersonType, where the *Instances* are connected to an *Instance* of AnimalType with a HasPet *ReferenceType*. In addition, the AnimalType *Instance* must be connected to a ScheduleType *Instance* with a HasSchedule *ReferenceType*. These conditions can also be represented in an OPC UA specific graphical notation, which is depicted in Figure 6.

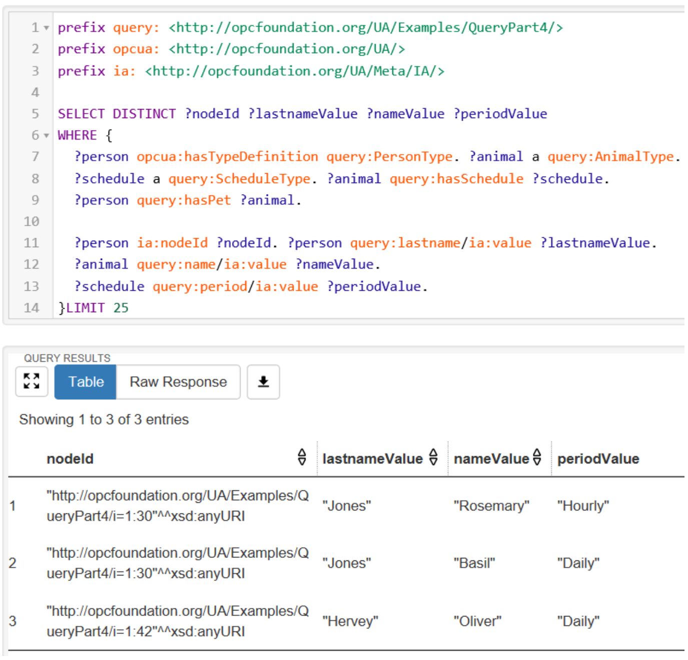
The *NodeTypeDescription* of Example B.2.4 can be formulated in the following way: Return the LastName *Property* of the PersonType *Instance* and the Name *Property* of the corresponding AnimalType *Instance* and the Period *Property* of the ScheduleType *Instance* (see also Table II).

Annex B of OPC UA Part 4 also specifies an example information model and the results which should be returned for that query executed against this information model. Based on our mapping, it is possible to transform OPC UA information models into RDF-graphs, which can be loaded into already existing SPARQL-endpoints. We will now take a closer look at Example B.2.4 from the OPC UA specification and its transformation into SPARQL using our OWL ontology (see also Figure 7). For our prototype we used Apache Fuseki [29] from the Jena Semantic Web framework.

Line 1-2 of Figure 7 shows how OPC UA *NamespaceURIs* of the OPC UA *NamespaceArray* can be used as SPARQL-prefix. Line 3 defines the *Namespace* for our OPC UA meta-model in OWL. The OPC UA *Content-Filter* is modelled in SPARQL based on simple triple statements (Line 7-9). Note that more complex filters are also no problem for SPARQL. For example, a "greater than" comparison can be easily modelled in SPARQL with the following statement: $FILTER(?value > 10)$. Based on Line 5 in combination with the Lines 11-13, the *NodeTypeDescription* is expressed

TABLE II
OPC UA PART 4 TABLE B.3 - NODETYPEDESCRIPTION [1].

Type-DefinitionNode	Include Subtypes	QueryDataDescription	
		Relative Path	**Att.**
PersonType	FALSE	".12:LastName"	value
		"<12:HasPet>12:AnimalType .12:Name"	value
		"<12:HasPet>12:AnimalType <12:HasSchedule>12:Schedule-Type.12:Period"	value


```
 1 prefix query: <http://opcfoundation.org/UA/Examples/QueryPart4/>
 2 prefix opcua: <http://opcfoundation.org/UA/>
 3 prefix ia: <http://opcfoundation.org/UA/Meta/IA/>
 4
 5 SELECT DISTINCT ?nodeId ?lastnameValue ?nameValue ?periodValue
 6 WHERE {
 7   ?person opcua:hasTypeDefinition query:PersonType. ?animal a query:AnimalType.
 8   ?schedule a query:ScheduleType. ?animal query:hasSchedule ?schedule.
 9   ?person query:hasPet ?animal.
10
11   ?person ia:nodeId ?nodeId. ?person query:lastname/ia:value ?lastnameValue.
12   ?animal query:name/ia:value ?nameValue.
13   ?schedule query:period/ia:value ?periodValue.
14 }LIMIT 25
```

Showing 1 to 3 of 3 entries

	nodeId	lastnameValue	nameValue	periodValue
1	"http://opcfoundation.org/UA/Examples/QueryPart4/i=1:30"^^xsd:anyURI	"Jones"	"Rosemary"	"Hourly"
2	"http://opcfoundation.org/UA/Examples/QueryPart4/i=1:30"^^xsd:anyURI	"Jones"	"Basil"	"Daily"
3	"http://opcfoundation.org/UA/Examples/QueryPart4/i=1:42"^^xsd:anyURI	"Hervey"	"Oliver"	"Daily"

Fig. 7. Example B.2.4 of OPC UA Part 4 in SPARQL (Apache Fuseki).

in SPARQL. The lower part of Figure 7 displays the results which were returned from Apache Fuseki after executing the query against the information model of OPC UA Part 4 Annex B. Not surprisingly the results exactly match the results which shall be returned for the given query.

In conclusion, we showed how OPC UA information models can be filtered and navigated using SPARQL in combination with a suitable OPC UA to OWL mapping. Furthermore, the same query formulated in OPC UA Query based on the OPC UA C++ SDK of Unified Automation needs about 100 lines of code (see also [30]), which is much more effort for users than our SPARQL-based query of Figure 7.

VI. Summary and Outlook

In this paper, we showed how Semantic Web technology can be used to solve some big challenges of OPC UA data models. First, we analysed how semantics is expressed within OPC UA data models and identified common pitfalls, which prevent a trivial extraction of the semantics. After that, we gave some insights into our OPC UA to OWL mapping and explained parts of the mapping by means of an example (This work will be contributed to a new OPC UA working group which will focus on this research). Finally, we outlined how this mapping can be used to validate and query OPC UA based data models.

Nevertheless, there are still some open challenges which have to be addressed. For example, OPC UA Query was designed to be used in production environments with thousands of updates per millisecond. This places a huge load on the triple store of the SPARQL query engine and also on the underlying controllers. At the moment we are investigating concepts to improve the performance based on classifying OPC UA data models in static and dynamic data.

TABLE III
OPC UA TO OWL TRANFORMATION RULES (NOT COMPLETE). NOTATION M:$E \rightarrow$ M(E).

OPC UA	**OWL**	
Element E	**Mapping m(E): OWL Entity**	**Mapping m(E): OWL Axiom**
Meta-Model Basic Concepts		
Variable-Meta-Concept	owl:Class *Variable*	SubClassOf(*Variable* m(*Base*))
NodeClass-Attribute	has-value class expression	SubClassOf(*Variable* DataHasValue(ia:nodeClass 2))
VarT	owl:Class *VarT*	SubClassOf(*VarT* m(*VariableType*))
DataType-Attribute	subclass axiom	SubClassOf(*VarT* m(*DataT*))
ValueRank-Attribute	subclass axiom	SubClassOf(*VarT* m(*VR*))
DataT	owl:Class *DataT*	SubClassOf(*DataT* m(*DataType*))
VR	owl:Class *VR*	SubClassOf(*VR* m(*ValueRankHelper*))
RefT	owl:ObjectProperty *RefT*	SubObjectPropertyOf(*RefT* m(*opcua:topObjectProperty*))
SourceNode-Restriction (textual)	domain object property axiom	ObjectPropertyDomain(*RefT* "<SourceNode>")
TargetNode-Restriction (textual)	range object property axiom	ObjectPropertyRange(*RefT* "<TargetNode>")
Symmetric-Attribute	symmetric object property axiom	SymmetricObjectProperty(*RefT*)
VID	owl:Class *VID*	SubClassOf(*VID* m(*Variable*))
HasTypeDefinition-Reference	subclass axiom	SubClassOf(*VID* m(*VarT*))
DataType-Attribute	subclass axiom	SubClassOf(*VID* m(*DataT*))
ValueRank-Attribute	subclass axiom	SubClassOf(*VID* m(*VR*))
BN	owl:ObjectProperty *BN*	SubObjectPropertyOf(*BN* m(*opcua:topObjectProperty*))
IA	owl:DataProperty *IA*	SubDataPropertyOf(*IA* *owl:topDataProperty*)
TA	owl:AnnotationProperty *TA*	AnnotationProperty(*TA*)
Instance-Model Basic Concepts		
VarI	owl:NamedIndividual *VarI*	ClassAssertion(m(*Variable*) *VarI*)
VID (if applicable)	class assertion	ClassAssertion(m(*VID*) *VarI*)
Meta-Model Relations		
$VarT_1$ *HasSubtype* $VarT_2$	subclass axiom	SubClassOf(m($VarT_2$) m($VarT_1$))
$DataT_1$ *HasSubtype* $DataT_2$	subclass axiom	SubClassOf(m($DataT_2$) m($DataT_1$))
$RefT_1$ *HasSubtype* $RefT_2$	object subproperty axiom	SubObjectPropertyOf(m($RefT_2$) m($RefT_1$))
Mandatory-VID $VarT_1$ $RefT_1$ VID_1	existential class expression	SubClassOf(m($VarT_1$) ObjectSomeValuesFrom(m($RefT_1$) m(VID_1)))
BN-Concept of VID_1	owl:ObjectProperty $IDOP_1$	SubObjectPropertyOf($IDOP_1$ m(*BN*))
Implicit-Concept	has-value class expression	SubClassOf(m(VID_1) DataHasValue(ia:browseName "<BrowseNameValue>")))
Implicit-Concept	functional object property axiom	FunctionalObjectProperty($IDOP_1$)
Implicit-Concept	irreflexive object property axiom	IrreflexiveObjectProperty($IDOP_1$)
Implicit-Concept	existential class expression	SubClassOf(m($VarT_1$) ObjectSomeValuesFrom($IDOP_1$ m(VID_1)))
Instance-Model Relations		
$VarI_1$ $RefT_1$ $VarI_2$	positive object property assertion	ObjectPropertyAssertion(m($RefT_1$) m($VarI_1$) m($VarI_2$))
IA of $VarI_1$	positive data property assertion	DataPropertyAssertion(m(*IA*) m($VarI_1$) "<AttributeValue>")
NOTE: VarI = Instance-Variable, VarT = VariableType, RefT = ReferenceType, DataT = DataType, VID = VariableInstanceDeclaration IDOP = InstanceDeclarationObjectProperty, IA = Instance-Attributes, TA = Type-Attributes, BN = BrowseName-Concept, VR = ValueRank-Attribute		

REFERENCES

[1] "Iec 62541: Opc unified architecture," Standard, 2015.

[2] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, 2001.

[3] M. Graube, L. Urbas, and J. Hladik, "Integrating industrial middleware in linked data collaboration networks," in *IEEE Emerging Technologies and Factory Automation*, 2016.

[4] A. Bunte, O. Niggemann, and B. Stein, "Integrating owl ontologies for smart services into automationml and opc ua," in *IEEE Emerging Technologies and Factory Automation*, 2018.

[5] B. Katti, C. Plociennik, M. Schweitzer, and M. Ruskowski, "Sa-opc-ua: Introducing semantics to opc-ua application methods," in *IEEE International Conference on Automation Science and Engineering*, 2018.

[6] B. Katti, C. Plociennik, and M. Schweitzer, "Semopc-ua: Introducing semantics to opc-ua application specific methods," *IFAC-PapersOnLine*, vol. 51, no. 11, 2018, iFAC Symposium on Information Control Problems in Manufacturing.

[7] "Owl-s: Semantic markup for web services," https://www.w3.org/Submission/OWL-S/, 2018.

[8] "Owl 2," https://www.w3.org/TR/owl2-primer/, 2018.

[9] "Rdf," https://www.w3.org/TR/rdf11-new/, 2018.

[10] "Rdf schema," https://www.w3.org/TR/rdf-schema/, 2018.

[11] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, *The Description Logic Handbook: Theory, Implementation and Applications*, 2003.

[12] "Hermit owl reasoner," http://www.hermit-reasoner.com/, 2018.

[13] "Fact++ reasoner," http://owl.cs.manchester.ac.uk/tools/fact/, 2018.

[14] "Shacl - shapes constraint language," https://www.w3.org/TR/shacl/, 2018.

[15] "Shacl implementations," https://w3c.github.io/data-shapes/data-shapes-test-suite/, 2018.

[16] "Sparql," https://www.w3.org/TR/sparql11-overview/, 2018.

[17] "Automationml opc ua information model - companion specification release 1.00," Standard, 2016.

[18] "Plcopen opc ua information model - iec 61131-3 - companion specification release 1.00," Standard, 2010.

[19] "Isa-95 common object model - companion specification release 1.00," Standard, 2013.

[20] "Vdma - members," http://www.vdma.org/en/mitglieder, 2018.

[21] "Opc ua companion specifications," https://opcfoundation.org/markets-collaboration/, 2018.

[22] "Vdw opc ua information model for cnc systems - companion specification release 1.00," Standard, 2017.

[23] "Existings verions of opc ua ontology or vocabulary?" https://www.researchgate.net/post/Existing_versions_of_OPC_UA_Ontology_or_Vocabulary, 2018.

[24] "Owl 2 structural specification and functional-style syntax," https://www.w3.org/TR/owl2-syntax/, 2018.

[25] R. Schiekofer, A. Scholz, and M. Weyrich, "Rest based opc ua for the iiot," in *IEEE Emerging Technologies and Factory Automation*, 2018.

[26] "Linked data," https://www.w3.org/standards/techs/linkeddata, 2018.

[27] "Siome - siemens opc ua modeling editor," https://support.industry.siemens.com/cs/ww/en/view/109755133, 2018.

[28] "Uamodeler," https://www.unified-automation.com/products/development-tools/uamodeler.html, 2018.

[29] "Apache jena fuseki sparql server," https://jena.apache.org/documentation/fuseki2/, 2018.

[30] T. Goldschmidt and W. Mahnke, "An internal domain-specific language for constructing opc ua queries and event filters," in *European Conference on Modelling Foundations and Applications*, 2012.