Undefined 1 (2009) 1–5
IOS Press

AutomationML Ontology: Modeling Cyber-Physical Systems for Industry 4.0

Olga Kovalenko [a] Irlán Grangel-González [b] Marta Sabou [a] Arndt Lüder [d] Stefan Biffl [a] Sören Auer [c] Maria-Esther Vidal [c]

[a] *Vienna University of Technology e-mail: {firstname.lastname}@tuwien.ac.at*
[b] *University of Bonn and Fraunhofer IAIS e-mail: {grangel@cs.uni-bonn.de}*
[c] *German National Library of Science and Technology e-mail: {soeren.auer|maria.vidal}@tib.eu*
[d] *Otto-von-Guericke-University Magdeburg, Institute of Ergonomics, Manufacturing Systems and Automation IAF e-mail: {arndt.lueder}@ovgu.de*

Abstract. We present an AutomationML ontology (AMLO) that covers the CAEX part of the AutomationML standard. The AutomationML data format facilitates the engineering data exchange during industrial systems design. Having a semantic representation of the AutomationML standard allows industrial practitioners to interlink and integrate heterogeneous data more efficiently and to benefit from the Semantic Web tools and technology stack, while at the same time, using a familiar domain-specific conceptualization. Compared to earlier efforts for semantically representing AutomationML, AMLO (a) covers the entire CAEX standard, and not just portions relevant for a use case; (b) has been developed following best practices for ontology engineering; and (c) is made openly available for the community by following latest guidelines on resource sharing and publishing. We describe AMLO and demonstrate its use in real-life scenarios for improving engineering processes in Cyber-Physical System design.

Keywords: AutomationML, industrial automation, Ontology Engineering, ontology design patterns, Industry 4.0

1. Introduction

The vision of smart manufacturing is currently supported and investigated as part of various initiatives worldwide, including Germany's Industrie 4.0 approach [3], the "Factory of the Future" initiative in France and the UK [30] and the "Industrial Internet Consortium"[1] in the USA. Core to these initiatives is enabling more flexible and efficient industrial production through a higher degree of digitization of the entire manufacturing value-chain. In particular, plants and factories (more generically referred to as *production systems*) will benefit from this digitization to evolve into cyber-physical production systems (CPPS). CPSS combine physical sensing with cyber-elements (i.e., software-based control) to more timely and accurately influence their behavior. They are complex mechatronic systems as, in their realization, they require input from several engineering disciplines, such as mechanical, electric and software engineering. To respond to fast-changing market needs, the design and engineering of CPPS needs to be faster while leading to results that are of high quality and complexity.

During the engineering of complex mechatronic systems such as CPPS, stakeholders typically belonging to different engineering disciplines, have to collaborate efficiently in order to deliver a high-quality end product (e.g., a complete production plant design) and to satisfy strict time frames. The presence of various engineering disciplines leads to a highly complex and software intensive environment, which is characterized by a) a multitude of engineering tools that were not designed to cooperate with each other; b) a variety of engineering domain-specific representations and data exchange formats applied; and c) differences in adopted workflows across the involved disciplines. Therefore, a key challenge for realizing CPPS relies on solving data integration challenges among the various systems, organizations and stakeholders involved in the engineering and operation of CPPS both across engineering do-

[1] http://www.iiconsortium.org

main boundaries (horizontal integration) and between different abstraction levels (business, engineering, operation) of the system (vertical integration) [5].

An approach for addressing data integration during the engineering of complex systems is the automation markup language AutomationML [18], the emerging IEC 62714 standard for facilitating uniform data exchange between engineering tools. AutomationML is an open (specification and schema are available), neutral (manufacturer independent without proprietary interfaces or libraries) and XML-based data exchange format that aims to ensure consistent and lossless data exchange during manufacturing systems design. AutomationML enjoys intense industry interest and adoption, yet, as an XML-based standard lacks a formal semantic basis that is increasingly necessary in industrial projects [23,25].

In fact, the use of knowledge-based approaches in general, and semantic technologies in particular, is a growing trend in the area of CPPS engineering. This development materializes in a number of approaches [19,33] that benefit from the following characteristics of semantic and knowledge representation technologies:

- formal and flexible semantic modeling with ontologies;
- intelligent, web-scale knowledge integration thanks to linked data mechanisms and ontology alignment techniques;
- browsing and exploration of distributed data sets;
- querying and reasoning based data validation and consistency checking; and
- knowledge reuse across diverse projects [32,34].

Our work aims to realize these benefits and support a wide-scale adoption of semantic technologies in AutomationML-enabled CPPS engineering by providing a comprehensive ontology based representation of the standard.

An ontology-based representation of the AutomationML standard enables the following improvements:

- flexible schema refinement and heterogeneous data linking and integration;
- using the semantic technology stack to enhance the engineering processes in CPPS engineering;
- connecting to other industrial standards that already have semantic representation, e.g., eCl@ss catalog [9] or the GoodRelations [16] ontology through ontology reuse or linked data mechanisms; and
- connecting to representations from other domains, e.g., eCore in Model-Driven engineering.

Several approaches already aimed to provide a semantic representation of AutomationML [1,6,12,17, 21,26], as detailed in Section 6. However, these efforts have the following shortcomings:

1. they are not covering the complete standard and are not tailored for specific use cases;
2. they are not developed according to best practices for ontology design; and
3. they are not openly available for consultation, extension or improving current design.

The AutomationML Ontology (AMLO) described in this paper advances the state of the art by addressing the issues above as follows. Firstly, AMLO is an OWL ontology that covers the entire emerging data exchange standard in the CPPS engineering field. Secondly, we followed several ontology engineering best practices during the design of AMLO, such as adopting the Uschold and King ontology engineering methodology [37]. We took as input two ontologies, developed independently of each other as initial efforts to cover the AutomationML standard for specific applications. The first one was created using a *top-down* modeling approach [22], with the focus on capturing the major concepts (i.e. classes and relations between them) and with the goal of enabling consistency checking between different AutomationML files. However, the property coverage was not sufficient to match the AutomationML XSD schema specification, especially for data properties. The second ontology was developed as a part of the Alligator approach [13] following a *bottom-up* modeling approach and had a well elaborated property structure, but was limited w.r.t. a class hierarchy. Combining both ontologies allowed for better class and property coverage w.r.t. the AutomationML XSD schema as well as for the AutomationML standard specification. Thirdly, AMLO is openly available due to following best practices for ontology sharing and publishing, in particular those described in [14]. As a result, AMLO covers 21 classes, 51 object properties and 48 data properties, and is aligned with three well-known vocabularies, i.e., *Provenance ontology*, *ontology of Units of Measure*, and *skos* vocabulary.

The reminder of this article is structured as follows. Section 2 introduces the AutomationML standard, explains its main constructs, and, additionally, presents an example on how a compound device can be mod-

eled in AutomationML. Section 3 describes the used methodology for ontology design, depicts the design process in detail, and also explains major design decisions taken. Section 4 gives an overview of AMLO structure, main classes and properties and important implementation details, i.e., ontology design patterns relevant in the ontology context, alignment to existing vocabularies and following best practices for sharing and reusability. Section 5 showcases two application examples for the AMLO. Section 6 presents the related work and positions the AMLO in the context of other semantic representations in automation engineering. Finally, Section 7 concludes and gives an overview of directions for future work.

2. AutomationML and Engineering Design

The *AutomationML* standard[18] enables modeling systems from single automation components to entire large and complex production systems and supports representation of the various aspects of such systems, i.e., system's topology, geometry, kinematics, and control behavior [8]. AutomationML is currently well recognized by major manufacturing companies such as Daimler, Audi and Siemens and continues gaining acceptance from the manufacturing market players.

AutomationML is not a completely new format, but rather consists of existing formats, which were extended, adapted, and combined appropriately. It allows modeling manufacturing system data sequentially, i.e., starting from the plant structure design, and then adding the geometry and kinematics information up to process sequences and logical dependencies following the sequence of engineering disciplines involved in the engineering chain. The top level of AutomationML is represented in terms of the *Computer Aided Engineering Exchange* (CAEX, IEC 62424) format for plant topology, which is used for storing hierarchical object information, properties, and libraries [10]. The geometry (mechanical drawings) and kinematics (physical properties such as force, speed, or torsion) are implemented with the *COLLAborative Design Activity* (COLLADA) format [2]. Further, the logic, i.e., sequencing, behavior, and control information is implemented with *PLCopen* XML (IEC 61131).

In this article, we focus on modeling topology information by means of the CAEX format. AutomationML is based on the following CAEX concepts:

RoleClassLibrary contains a collection of possible functionalities that can be provided by the plant equipment. A *role class* (RC) defines a physical or a logical object as an abstraction of a concrete technical realization, which is vendor independent, e.g., a robot or a sensor. This way a functional semantics is assigned to an object, enabling an automatic interpretation by a tool. RCs can also define attributes that are generally expected for this object type, e.g., a payload for a robot RC. Additionally, RCs can be assigned to objects within the SystemUnitClassLibrary and InstanceHierarchy to specify an object type.

SystemUnitClassLibrary comprises collections of vendor specific solution equipment objects. Those objects can be matched with the system requirements, defined by the role classes, and used to implement the plant design. A *system unit class* (SUC) defines a concrete technical realization of a physical or logical object, thus representing a specific instantiation for a RC. System unit classes are instantiated within the InstanceHierachy.

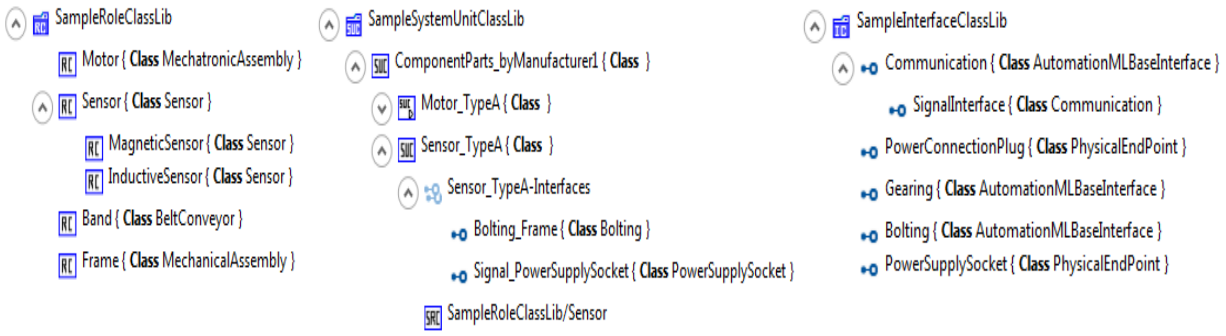
InterfaceClassLibrary defines all interfaces required to describe the plant model. An *interface class* (IC) can be used either a) for specifying relations between objects of a plant topology, e.g., to connect a sensor with a programmable logic controller (PLC); or b) for specifying references to information that is stored outside of the CAEX file, e.g., to assign a 3D description to an object.

InstanceHierarchy describes the plant topology, including the concrete equipment of an actual project - the instance data. The instance hierarchy contains all data including properties, interfaces, role classes, relations, as well as references.

Applying AutomationML in engineering projects is already an important improvement, as it facilitates data exchange between the project stakeholders and defines a project-specific vocabulary to which all engineers can relate. Nevertheless, there is still a lack of infrastructure for supporting advanced engineering activities across the various engineering disciplines and their corresponding tools, e.g., data linking, change propagation across connected datasets, data analysis and consistency checking. Having an ontology for AutomationML will help to address these gaps.

2.1. An AutomationML Modeling Example

To illustrate how a CPSS is represented in AutomationML, we developed an example using the Automa-



(a) RoleClassLibrary (b) SystemUnitClassLibrary (c) InterfaceClassLibrary

Fig. 1. **Sample AutomationML modeling example**. Conveyor consisting of a frame, a band, an inductive sensor and a motor. (a) SampleRoleClassLib contains role classes that define vendor independent functionalities for the conveyor. (b) SampleSystemUnitClassLib lists vendor specific device realizations options for the sample conveyor. (c) SampleInterfaceClassLib comprises Interface classes for the conveyor.

tionML Editor 4.7.0.[2], which allows browsing and developing AutomationML files. The example system models a conveyor that consists of a frame, a band, an inductive sensor and a motor.

The **RoleClassLibrary** contains classes to represent the basic functionalities for the system components, in our case *Motor*, *InductiveSensor*, *Band* and *Frame*. The *Frame* and *Motor* extend the basic class *MechatronicAssembly* that is defined within *AutomationMLBaseRoleClassLib*. Different types of sensors are represented by the RCs *MagneticSensor* and *InductiveSensor*, all of which are derived from the more general *Sensor* RC (cf. Figure 1a). The role classes are vendor independent and do not comprise implementation-specific details, e.g., a role class *Motor* means "something that can fulfill motor function".

The **SystemUnitClassLibrary** contains component realizations that can be used to implement the functionalities, defined by RCs in a real system. SUCs are vendor specific, e.g., *Motor_TypeA* represents a specific device with the motor functionality produced by Manufacturer1 (cf. Figure 1b). RCs are assigned to SUCs to specify the functionalities that a specific device can fulfill. E.g., *Sensor_TypeA* has the RC *Sensor* assigned, meaning that it can be used to implement the functionality defined by *Sensor* RC.

The **InterfaceClassLibrary** contains interface classes representing various relations between the system components. These relations can be of different nature: mechanical ones, e.g., *Bolting* and *Gearing*; electrical, e.g., *PowerConnectionPlug* and *PowerSupplySocket*; of software and control related, e.g., *SignalInterface* (cf. Figure 1c).

Additionally, **attributes** are specified for the SUCs providing further details for the specific devices. For example, *ConveyorBand_TypeB* has the attributes *Weight*, *Material*, *MaxPayload* and *MaxSpeed* defined. For each attribute a description, a measurement unit, a data type and value can be specified. For example, one can see that the *ConveyorBand_TypeB* has the maximum payload of two kilograms (cf. Figure 2).

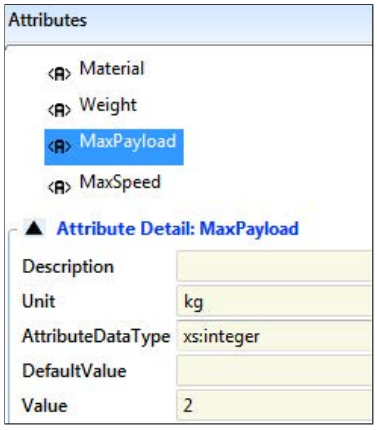


Fig. 2. **MaxPayload Attribute of the ConveyorBand ConveyorBand_TypeB** Values of the MaxPayload attribute of the ConveyorBand_TypeB in the sample system, i.e., Unit, AttributeDataType and actual Value of the Attribute are shown. Other attributes of the system such as Material, Weight and MaxSpeed are also depicted.

Finally, the **InstanceHierarchy** contains the design of the real-life system - the conveyor consisting of a band *myConveyorBand*, a motor *myMotor* to move the band, a frame *myConveyorFrame*, and an induc-

[2]https://www.automationml.org/o.red.c/dateien.html

tive sensor *myInductiveSensor* to identify positioning of the object on the band (cf. Figure 3).

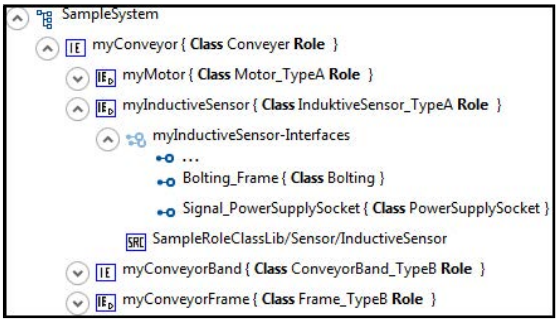


Fig. 3. **Instance hierarchy classes for the sample system**. myConveyor is defined via four IEs: myConveyorBand, myMotor, myConveyorFrame, myInductiveSensor, each representing a specific device for the conveyor design.

3. Ontology Design: Methodology and Modeling Decisions

In this section, we describe the modeling process for the AutomationML Ontology (AMLO). Section 3.1 presents the design methodology and explains the ontology engineering process in detail. Section 3.2 describes major design alternatives and design choices made for the AMLO.

3.1. Methodology

The AMLO was designed following the "Process-based design" methodology of Uschold and King [37]. As the main purpose for ontology creation we formulated the following points:

- AMLO should be maximally compatible with the AutomationML XSD schema and the AutomationML standard;
- The vocabulary used by AMLO should be as close as possible to the terminology in the standard, to allow an intuitive understanding of the ontology by users from industry who are familiar with the standard, but not with Semantic Web technologies.
- To facilitate ontology reuse by other parties, AMLO should be easy to find, access and reuse.

As a starting point for ontology development we relied on two ontologies, initially designed with the intention to provide semantic representation for the AutomationML standard for specific applications.

The first ontology was built at TU Vienna using top-down modeling approach [21]. The result was a light-weight ontology capturing the major design decisions in terms of classes and relations between AutomationML elements. The intended application was consistency checking between different AutomationML files in multidisciplinary engineering projects. However, property coverage was weakly elaborated and did not completely cover the AutomationML XSD schema specification, particularly for data properties.

The second ontology was built following the bottom-up modeling approach as part of the Alligator project at the University of Bonn [13]. The intended application was the automatic semantic integration of AutomationML documents. This ontology had a well elaborated data property structure, but a less developed hierarchy for classes and relations.

The conceptualization phase of AMLO was therefore accomplished by combining both ontologies and refining the conflicting concepts, where necessary. This allowed for better class and property coverage w.r.t. the AutomationML XSD schema and the AutomationML standard specification.

While working on the AMLO we put especial emphasis on following the best practices for ontology design [27,29]. Namely, we make use of Ontology Design Patterns (see Section 4.2), reuse constructs from the already existing semantic sources (see Section 4.3) and ensure that the ontology is fully available and documented (see Section 4.4).

With the goal of validating the ontology resulting from the previous steps we applied the following approach. First, we performed two iterations of structure validation with domain experts. For this we asked the support of colleagues in the Otto-von-Guericke University (Magdeburg), who are actively involved in the development of the AutomationML standard and are, therefore, deeply familiar with the semantics of the AutomationML constructs. Second, we manually implemented several AutomationML data samples, provided on the official AutomationML web-site[3] by means of the AMLO. This guaranteed that various modeling situations supported by the standard can be indeed described by means of AMLO.

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

3.2. Design Decisions

(D1) Flexibility of the standard

In order to support a wide adoption among industrial practitioners, the standard creators intentionally avoided including too many constraints in the AutomationML standard specification level. For instance, the general design approach in AutomationML assumes that one first defines the functional capabilities for a future system using role classes (RCs) and then one selects and assigns suitable system unit classes (SUCs), which will be a concrete realization of those capabilities defined via RCs. However, it is also possible (and will not be an error) if one defines a system directly using the SUC libraries. This way the tool vendors that typically would have elaborated SUC structures, but only limited (if any) hierarchies of general functionalities, could use AutomationML without having to significantly adjust their workflows.

With the aim to support the flexibility of the AutomationML standard in the designed ontology, we decided to keep the amount of predefined restrictions to a minimum. This way, modeling with the AMLO will have the same flexibility as when using the AutomationML standard.

(D2) Modeling hierarchies of ICs, RCs and SUCs

There were three major options to model the hierarchies of interface classes (ICs), RCs and SUCs in AMLO.

First, one can build class hierarchies to capture hierarchical relations between the elements of IC-, RC- and SUC- libraries in AutomationML. For example, *MagneticSensor* and *InductiveSensor* are both subclasses of *Sensor*, which is a subclass of a *RoleClass*. Instances of *InductiveSensor* describe specific roles assigned to a certain device (represented via instance of an *InternalElement* class). For example, *InductiveSensor* role assigned to a specific device *myInductiveSensor* in the instance hierarchy (see modeling example in Section 2.1).

Secondly, all roles can be modeled as instances of the RC concept. In this case, the further hierarchical relations between the individual roles should be modeled via additional constructs in the ontology, which would relate the RC instances to each other.

The third option consists in capturing different roles via relations, e.g., *hasSensor* that would connect instances of *InternalElement* (corresponding to specific devices) to instances of *RoleClass*. The hierarchical relations between the roles are then modeled using the subproperty mechanism, e.g., *hasMagneticSensor* and *hasCurrentSensor* being subproperties of *hasSensor*. These three possibilities for modeling the same semantics are compatible, i.e., rules can be defined to automatically transform one into another [31].

Inspired by the experiences in building the Ontology of Units of Measure [31] we opted for the first approach for modeling the IC, RC, and SUC hierarchical relations in AMLO. We also found this approach the most intuitive in terms of similarity to the AutomationML representation.

(D3) Splitting up multiple-use properties with rdfs:subPropertyOf

There are some relations in AutomationML that capture the same semantics, but for different object types. For example, an AutomationML file can contain all four AutomationML main object collections: InstanceHierarchy, InterfaceClassLib, RoleClassLib and SystemUnitClassLib. In this case, the *contains* property holds the same semantics for all four constructs, meaning 'having this structure within the file'. Therefore, the straight-forward way of modeling this scenario would be simply defining *contains* Object property that would relate instances of a *CAEXFile* to the instances of *RoleClassLib*, *SystemUnitClassLib*, *InterfaceClassLib*, and *InstanceHierarchy*. However, this can potentially cause problems during reasoning and inference tasks, because of the Open Word Assumption logic implied by reasoners. To tackle this problem, we decided to model such relations in the following way: a) a super property is defined capturing the general relation semantics and does not explicitly specify a certain class for its range. E.g., the domain of the *contains* property is the *CAEXFile*, but nothing is specified for the range; b) subproperties are defined to describe each specific case for the property range, e.g., *hasRoleClassLib*, *hasSystemUnitClassLib*, *hasInterfaceClassLib*, and *hasInstanceHierarchy* are subproperties of *contains*.

This approach allows using a less complex vocabulary for the applications where only querying is important. That is, one can define all the relations using the general *contains* Object property without having to learn the property name for each specific case. At the same time, if the intended application involves reasoning or inference one can use more elaborated and detailed vocabulary (i.e., subproperty labels) to avoid potential conflicts.

(D4) Terminology

We stayed as close as possible to the labels used in the AutomationML standard while choosing terms for naming classes and properties in AMLO. The main in-

tention is to facilitate the understanding of the ontology constructs and their semantics for users that are already familiar with the AutomationML standard, but might not possess deep knowledge of Semantic Web technologies.

(D5) Linking to eCl@ss and domain-specific standards

An important feature of AutomationML is the ability to link to eCl@ss[4] - the standardized catalogue for describing products and services. In AutomationML, semantics of RCs and Attributes can be specified by linking them to corresponding eCl@ss definitions. Namely, Attributes are semantically equivalent if they share the same value for the refSemantic attribute; and RCs are semantically equivalent whenever they share the same eCl@ss references for the AutomationML attributes eClassVersion, eClassClassification, and eClassIRDI [35].

AMLO also supports linking to the eCl@ss standard via the class *ExternalStandard* and object property *hasRefSemantic*, which has *ExternalStandard* as its range. Following the strategy explained in (D3) two subproperties of the *hasRefSemantic* are further defined, with *RoleClass* and *Attribute* as a domain and *ExternalStandard* as a range.

Thanks to the general nature of this linking mechanism, one can also link to other domain specific standards. To this end a corresponding instance of the class *ExternalStandard* should be created. Linking is then done in a similar way as described above for the eCl@ss standard.

4. Ontology Development

In this section we discuss the development aspects of AMLO. Firstly, Section 4.1 gives an overview on ontology structure and major classes and properties. Secondly, Section 4.2 discusses what are the relevant ontology design patterns (ODPs) in the context of AutomationML and how they are reflected in AMLO. Next, Section 4.3 describes the existing vocabularies that have been reused in AMLO. Finally, Section 4.4 documents our efforts to ensure that AMLO follows best practices for ontology sharing and publishing. A summary of the main ontology details is provided in Table 1.

[4]eCl@ss standard: https://www.eclass.eu/en/standard

4.1. Ontology Overview

By considering the aforementioned design decisions, we modeled AMLO with focus on the main AutomationML elements. AMLO covers all the major constructs of the CAEX XSD schema, with one exception for the family types for RCs, SUCs and ICs, i.e., *RoleFamilyType*, *SystemUnitFamilyType* and *InterfaceFamilyType*. Those are XML related structural concepts that are needed in XML to specify the parent-child hierarchical relations. Since OWL provides means to model such relations explicitly there was no need to keep those construct in the ontology. Figure 4 shows the main entities of the ontology.

CAEXFile represents the AutomationML document and is the core element of the ontology. To describe the metadata related to the document, the class *AdditionalInformation* was connected to the *CAEXFile* class. It comprises data about the version, writer identification, name, release, vendor as well as the project to which the document belongs.

Container classes *InterfaceClassLib*, *RoleClassLib*, *SystemUnitClassLib* and *InstanceHierarchy* are container concepts, i.e., classes grouping other classes. These classes describe containers for *InterfaceClass*, *RoleClass*, *SystemUnitClass* and *InternalElement* respectively and are linked to the *CAEXFile* class via *contains* property.

RoleClass represents the AutomationML *role class* concept and defines a vendor independent functionality that can be provided by equipment elements. *RoleClasse*s are used to assign generic functional semantics to instances of *InternalElement* and *SystemUnitClass*, i.e., to describe the functional capabilities of equipment elements. This is done via properties *hasSupportedRoleClass* and *hasRoleRequirements*.

SystemUnitClass represents the AutomationML *system unit class* concept and defines a vendor-specific technical realization of a physical or logical object. Instances of the *SystemUnitClass* can contain (or be a part of) other system unit classes. This is defined via properties *hasPart* and *isPartOf* respectively.

InterfaceClass represents the *interface class* concept in the AutomationML and in general is used to specify either a) relations between the different topology elements; or b) references to various external information sources. Interfaces can be linked to the instances of *RoleClass*, *SystemUnitClass* and *InternalElement* via the *hasInterface* property. *ExternalInterface* is a subclass of *InterfaceClass*.

Table 1
AMLO Ontology details.

General	**Name:**	AutomationML Ontology (AMLO)
	Size	21 classes, 51 object properties, 46 data properties
	DL Expressivity	ALHIF+(D)
Availability	**PersistentURI**	`https://w3id.org/i40/aml`
	GitHub	`https://github.com/i40-Tools/AutomationMLOntology`
	Licence	Creative Commons
Reusability	**VoCol Instance**	`http://vocol.iais.fraunhofer.de/aml/`
Technical	**Ont. Eng. Methodology**	King and Uschold [37]
Quality	**Reused Ontologies**	PROV-O, Units of Measure
	Reused ODPs	transitive PartOf ODP
	Documentation	By means of `skos:definition`, `skos:altLabel`, `skos:prefLabel`

InternalElement represents the *InternalElement* concept in the AutomationML and defines concrete equipment of an actual project, i.e., devices used in a concrete plant. Similar to the *SystemUnitClass*, *InternalElement*s can contain (and be part of) other *InternalElement*s. This is defined via properties *hasPart* and *isPartOf*. One can additionally define what *SystemUnitClass* is instantiated by a specific *InternalElement* via the *hasBaseSystemUnitClass* property.

Attribute expresses the actual properties of CPPS such as length, size, temperature, speed, etc. Attributes can be defined for the instances of *RoleClass*, *InterfaceClass*, *SystemUnitClass* via the *hasAttribute* property. Classes *om:Measure* and *om:Unit_of_measure* are imported from the Ontology of Units of Measurement to formally define the units of measure for attributes.

InternalLink is an interesting element of AutomationML that represents a directed connection between two constructs. Each *InternalLink* references two *ExternalInterface*s and two constructional elements, i.e., either *InternalElement*s or *SystemUnitClass*es, depending on which level of abstraction a given connection is specified. This is done via properties *hasReferencePartnerSide(A/B)_Interface* to link to the *ExternalInterface*s, and via properties *hasReferencePartnerSide(A/B)_Object* to link to *InternalElement*s and *SystemUnitClass*s. The direction is important here, i.e., "from A to B".

4.2. Ontology Design Patterns

Ontology design patterns (ODPs) are reusable modeling building blocks providing solutions to recurrent domain modeling problems [11]. ODPs are an important means to improve the quality of an ontology design as they represent best practices in ontology modeling frequently used by ontology developers.

Sabou et al. distinguished three major groups of patterns that are important in the engineering context: a) part-whole relations; b) connections between components; and c) component roles [34]. We hereby discuss these groups of patterns and how they were applied to support our modeling.

Part-whole relations are important for modeling containment hierarchies. In AMLO we decided to use the transitive version of partOf relation that corresponds to the *PartOf ODP*[5]. We selected this option for part-whole relation, because in the context of engineering system design a potentially important application is recursively getting all parts and their subparts for a specific object. We did not use the *Componency ODP*[6] (nontransitive version of part-whole) or the *TimeIndexedPartOf ODP*[7] (part-whole relation holding only for a specific time interval) in AMLO, since there are no relations with similar semantics in the AutomationML standard.

Another pattern, which is often discussed in the context of part-whole relations, and is also relevant in the AutomationML context is modeling constituency. **Constituency** refers to relations without a clear part-of relationship. Typical example is representing a material from which an object is made, e.g., several types of wood constitute a table. There is a special ODP defined for modeling constituency - *Constituency ODP*[8]. In the context of AutomationML, constituency is typically represented by an attribute *hasMaterial* or similar defined for SUCs or IEs. Since the attribute hierarchy is not intended to be stable, but rather can vary from one AutomationML file to another, we decided to model this aspect as follows: a) an object property *hasConstituent* is included in the ontology; b) after

[5] http://ontologydesignpatterns.org/wiki/Submissions:PartOf
[6] http://ontologydesignpatterns.org/wiki/Submissions:Componency
[7] http://ontologydesignpatterns.org/wiki/Submissions:TimeIndexedPartOf
[8] http://ontologydesignpatterns.org/wiki/Submissions:Constituency

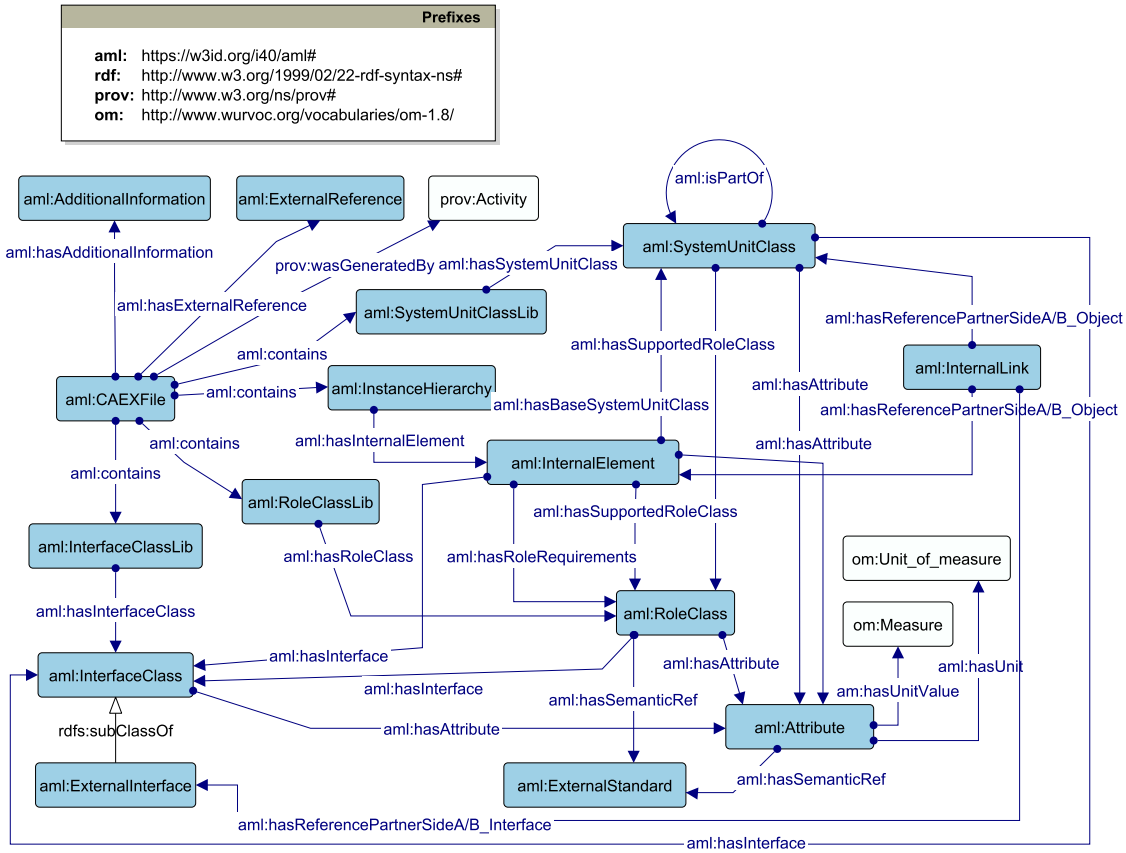


Fig. 4. **Core classes and properties of the AMLO** Core AML concepts are represented in AMLO. Classes and relations of the Ontology are depicted in blue. White classes represent reused classes from important ontologies such as PROV and OM.

a specific AutomationML file is transformed into the AMLO, and, therefore, a certain property hierarchy is formed under the *hasAttribute* property, one can align the specific attribute, e.g., *hasMaterial* with the *hasConstituent*, specifying that these two have similar or the same semantics.

Connections between system components represent interactions between its parts, such as flows of energy, matter or signals [36]. In AutomationML connections are modeled via ICs, which allow specifying connection elements (between which the connection is established), direction, and connection type (e.g., signal transfer for control variables or different kinds of mechanical linking between the hardware elements). ICs as they are modeled in the AutomationML allow for a detailed and flexible definition of various connections in an engineering system. Therefore, we decided not to use additional ODPs in the ontology for modeling connections.

Component roles represent functionality and behavior associated with a component in a system [34]. In AutomationML such information is represented via RCs. A RC is specified for an IE or SUC to define what kind of functionality in a system this component has. Hence, we did not use any additional ODPs to model these kind of relations, since they can be modeled directly by means of AutomationML constructs that AMLO derives from the standard.

4.3. Reusing Existing Ontologies

Reusing existing and recognized ontologies is considered a good practice for ontology building [24,28][9]. There is a rich variety of data sources in semantic formats available for the engineering domain, which can be potentially used to improve the quality of engineer-

[9]https://www.w3.org/TR/ld-bp/#VOCABULARIES

ing data and allows more comprehensive analyses. Examples are ontologies that cover engineering and related topics, or domain-specific sources, e.g., product catalogues like eCl@ss [9]. Once having the semantic representation of the original AutomationML data, existing semantic data sources can be used to enrich original data structures with additional information.

Even though AutomationML comprises very specific concepts and relations, we have reused some well-known ontologies such as the *skos* vocabulary, the *ontology of Unit of Measure* [31], as well as the *Provenance ontology (PROV-O)*[10].

The *Provenance ontology* is used to represent the information about a tool that has generated the given AutomationML documents. The *skos* vocabulary is used to encode additional information about concepts and property related to documenting, e.g., definitions for AMLO constructs or alternative labels. The *Ontology of Units of Measure* (OM) is used to bring formal semantics for encoding units or measurement, which are originally represented as strings in AutomationML. Since the problem of the disambiguation of units of measure is very common and relevant for the engineering domain, we will focus on it in more detail. Below we show how OM can be exploited in AMLO to semantically decsribe units of measurement.

In CPPS environments, units of measurement are of crucial importance for the correct functioning and coordination of the associated processes. They are required for the specification of products as well as for representing the data produced by measuring devices. Commonly, data related to units of measurement are codified as simple strings, thus, losing the semantics associated with them. The Ontology of Units of Measurements (OM) has been proposed to bring semantics to this domain [31]. The OM ontology provides a complete and comprehensive set of units of measurement.

In AutomationML, attributes are used to express properties of different objects. For instance, let's consider the motor from the sample conveyor system presented in the Section 2.1. An important attribute of a motor is a rotational speed, which can be measured both in radians per second and in revolutions per second. If represented informally as a string, both of those units can be expressed as "r/s". Listing 1 depicts a fragment of an AutomationML document showing an attribute setting the rotational speed of the motor. The intended meaning of "r/s" in this case is radians per second, but could be also interpreted as revolutions per second. This example demonstrates the importance of semantically representing units of measurement to avoid ambiguity as well as to express the correct semantics of the attribute.

```
<Attribute Name="RotationalSpeed" Unit="r/s"
      AttributeDataType="xs:float">
  <Value>16.6</Value>
</Attribute>
```

Listing 1: **Fragment of the RotationalSpeed attribute of the motor myMotor**. A fragment of the attribute RotationalSpeed encoded in the AML standard representing the Unit, the AttributeDataType as well as the value.

As a solution, Listing 2 depicts an example where AML and OM were used together to describe units of measure.

```
prefix om:<http://www.wurvoc.org/vocabularies/om-1.8/> .
prefix aml:<https://w3id.org/i40/aml/> .

aml:SpeedAttribute a aml:Attribute,
aml:hasNameAttribute "RotationalSpeed"@en,
om:value aml:spMeasurement;
aml:RotationalSpeed a om:Rotational_speed,
om:phenomenon aml:SpeedAttribute;
aml:spMeasurement a om:Measure,
om:unit om:radian_per_second-time,
om:numerical_value 16.6.
```

Listing 2: **Representing units using the OM ontology.** Representing the unit of measure of the RotationalSpeed attribute, i.e., radians per second by combining AML and OM ontologies.

4.4. Following Best Practices for Ontology Sharing and Reusability

While designing AMLO, we followed the guidelines for ontology sharing and reuse created for the ISWC2016 Resources Track [14]. The major focus here was on performing all necessary steps to ensure high-quality documentation and availability of the ontology for other interested parties, thus, facilitating ontology reuse. There are three big parts in the guidelines: 1) steps related to reusability (denoted with "R.x"); 2) steps related to availability (denoted with "A.x") of the ontology; and 3) design and technical quality (denoted with "DTQ.x"). Below we describe how we followed those steps for AMLO.

(R.1) How easy it is to (re)use the ontology (e.g., is there documentation or tutorials available?). There

[10]https://www.w3.org/TR/prov-o/

is a detailed description of AMLO structure available at `https://w3id.org/i40/aml`. In addition, the documentation regarding the ontology is publicly available via a *VoCol* instance[11]. VoCol is a platform to support the collaborative development process of ontologies based on version control systems, git and github in this case [15]. After every push to the github repository, VoCol automatically provides features such as documentation generation, evolution of changes, visualization, ontology validation, etc.

(R.2) Possibility to apply in a wide set of scenarios. We did not make application specific design decisions during the AMLO development. The major goal was to fully cover the standard while preserving its flexibility in the ontology. Therefore, AMLO can be used in a wide range of scenarios.

(R.3) Potential for extensibility. AMLO can be aligned to other standards using the `ExternalStandard` class and the object property `hasRefSemantic`.

(A.1) Publishing at a persistent URI. AMLO is published at the w3id[12].

(A.2) License specification. AMLO has the Creative Commons license.

(A.3) Is the resource publicly available / findable? The source code for the AMLO as well as its evolution track is publicly available on GitHub[13].

(DTQ.1) Following best practices for ontology design. We followed the design methodology of Urshold and King[37] (see Section 3.1 for details).

(DTQ.2) Reuse or extension of high-quality resources (e.g., ODPs or well-known ontologies). The AMLO design process covered both the reuse of ODPs (see Section 4.1) and the reuse of well-known ontologies (see Section 4.2).

(DTQ.3) Being self-explanatory (e.g., are there appropriate human and machine readable descriptions?). With the objective to ensure that the AMLO is self-explanatory we included the following skos constructs: a) `skos:definition` to explain the meaning of main entities; b) `skos:altLabel` to include the alternative names for ontology entities, e.g., "SUC" for "System Unit Class"; and c) `skos:prefLabel` to include the most commonly used name for ontology entities.

[11] http://vocol.iais.fraunhofer.de/aml/
[12] https://w3id.org/i40/aml
[13] https://github.com/i40-Tools/AutomationMLOntology

5. Applying AMLO for CPPS System Engineering

In this section, we present three representative use cases supported by AMLO. The first use case aims for the integration of AutomationML documents across different engineering disciplines (Section 5.1). The second and third use cases show how querying (Section 5.2) and reasoning (Section 5.3) can be applied on top of integrated AMLO data.

The context for the all three use cases is developing an automation system, e.g., a production plant and the corresponding control system. Such engineering setting is characterized by a rich variety of engineering tools, terminologies, heterogeneous data models, and data formats applied by project stakeholders. Stakeholders work on the same engineered system, but consider it from different points of view, e.g., plant planning, mechanical engineering, electrical wiring or control system implementation. Therefore, this is a complex and data-rich setting, with high demand for effective and efficient data integration and advanced analytics within and across the invilved disciplines. Project participants apply AutomationML as a common vocabulary for data exchange, which is a first step to facilitate data exchange and communication. However, there is still need for tool-support and technologies, which would allow a) data integration across the different engineering disciplines and b) comprehensive and flexible analytics over the system's engineering data. Semantic technologies possess rich capabilities for both tasks, e.g., by offering comprehensive querying with SPARQL and inference and reasoning facilities.

5.1. UC1 - Integration of Multi-disciplinary AutomationML Documents

AutomationML is used to describe CPSS components and to facilitate their integration in the context of multi-disciplinary engineering of CPPS. In this context, AutomationML documents are developed from different expertise, i.e., mechanic, electric and software. Different expertise generate different views over the same elements described in AutomationML and, thus, semantic heterogeneity when integrating CPPS components [4,20]. Semantic heterogeneity needs to be solved while keeping the meaning of the CPSS components. In the *Alligator* approach [13], this problem is tackled by defining an RDF-based representation of AutomationML documents which are based on AMLO. AMLO is used here as a canonical represen-

tation defining the semantics of the elements with respect to the AutomationML standard. Based on this, a Datalog-based representation of the AutomationML input documents, and a set of rules for identifying semantic heterogeneity conflicts are developed. A deductive engine is used to resolve the conflicts, to merge the input documents and produce an integrated AutomationML document.

5.2. UC2 - Inconsistency Checking in Engineering Design

After the AutomationML data from various disciplines have been integrated (e.g., in Alligator approach described above), SPARQL can be used to perform analyses on the top of engineering data.

Hereinafter we present the example consistency check that was implemented in the use case of our industry partner, a power plant system integrator, based on the AMLO. In the example we leverage the formal foundations of AMLO to access the device characteristics transitively to gather the overall system statistics.

Assume there are maximum weight and electrical consumption thresholds specified by a customer for the system under design based on the location settings where the system will be deployed. Project engineers can run the following SPARQL query over the engineering data to obtain this information:

```
SELECT (SUM(xsd:integer(?deviceWeight)) AS ?systemWeight)
       (SUM(xsd:integer(?devicePowerConsumption)) AS ?
      systemPowerConsumption)
WHERE {
aml:myConveyor aml:hasPart* ?device
?device a aml:InternalElement .
?device aml:hasAttribute ?attribute .
?attribute aml:hasAttributeName "Weight" .
?attribute aml:hasValue ?deviceWeight .
?device aml:hasAttribute ?attribute .
?attribute aml:hasName "PowerConsumption".
?attribute aml:hasValue ?devicePowerConsumption . }
```

Listing 3: **Returning the weight and power consumption of the production model.** SPARQL query returning the overall weight and power consumption of the production model by using data annotated according to AMLO.

5.3. UC3 - Flexible Hierarchy Adaptation using Reasoning

Assume there is the following requirement defined by a project engineer: "All controller devices in a production system must have exactly one connection to automation object defined". However, the "controller" role was not defined explicitly in the project and must be defined separately for each topology of roles.

The ontology-based representation of AutomationML data allows flexible reconfiguration of the defined structures. Reasoning can be applied to the roles hierarchy in order to enrich the existing classification, e.g., the following SWRL rule can be defined to automatically classify the controllers (assuming that the marker for being a controller is supporting the "Ro_MechatronicAssembly" role):

```
SystemUnitClass(?device_type) Λ RoleClass(?role) Λ
      hasSupportedRole(?device_type, ?role) Λ
      Ro_MechatronicAssembly → Controller(?device)
```

Listing 4: **SWRL rule for reclassification of the RoleClass hierarchy** The rule relies of the semantics encoded in AMLO for reclassifying the RoleClass hierarchy.

This rule enriches the existing Roles hierarchy. The newly derived triples (i.e., knowledge) can be then either saved into the ontology (then the new classification will be available for all later check executions) or stay in memory (therefore being only available during the current checking session). All the devices in the production system can be now automatically checked for being controllers or not. After re-classification has been performed, one can run a SPARQL query to check whether all controller devices in the system have the required property.

6. Related Work

In recent years, much attention has been paid to represent the knowledge regarding the automation domain by using ontologies [7]. Concretely, the AutomationML standard has been at the core of many efforts in this regard. The main focus of these works has been in formalizing the CAEX format into an ontology as follows. Abele et al. [1] present an ontology for the validation of plant models, e.g., attribute consistency checking and correctness of internal links. The ontology covers base concepts of AutomationML, i.e., CAEX, and how they are mapped to OWL; Björkelund et al. [6] model an ontology exploiting core concepts of AutomationML and utilize the resulting ontology as a common vocabulary to transform AutomationML models into RDF. [26] describes a knowledge integration framework for robotics. In this context, the knowledge is represented in AutomationML and transformed

Table 2

Comparison of existing semantic representations of the AML standard. Comparison of existing AutomationML ontologies with AMLO; Comparison is performed regarding the use of ontology engineering methodology, the inclusion of ODPs, the reuse of terms from other ontologies, the availability of ontology sources, the source used to develop the ontology, the ontology formalization as well as the generality of the use case in which the ontology is utilized.

Approach	*Methodology*	*ODPs*	*Reuse*	*Availability*	*Source*	*Formalization*	*Use Case*
Abele et al. [1]	No	No	No	-	CAEX	OWL	Plant Validation
Björkelund et al. [6]	No	-	-	-	-	-	Knowledge and Skill Representation
Glawe et al. [12]	No	No	No	-	-	-	Security in Automation
Persson et al. [26]	No	No	No	-	-	-	Knowledge Integration
Hua et al. [17]	No	No	Yes	-	-	-	A Model-driven Approach for Robotics
AMLO	Yes	Yes	Yes	Yes	CAEX	OWL	General applicability

to RDF to publish the RDF data according to Linked-Data principles. To this end, they propose an ontology covering the main CAEX concepts. Another definition of a CAEX ontology is developed in the context of using knowledge to support the engineering process of automation systems [12]. While the focus of this work is on security, core concepts of the CAEX scheme are designed as an OWL ontology. Further, SWRL rules are introduced to logically connect CAEX elements. Hua et al. [17] developed a semantic-based approach to software engineering of industrial robotics. Authors propose an approach to deal with robotic components and how they can be classified and modeled with AutomationML. Further, how AutomationML models can be processed by means of an ontology is demonstrated in this work. To this end, an AutomationML ontology covering main aspects of AutomationML is created.

To analyze existing works for an AutomationML ontology, aspects considered of crucial importance for ontology development are investigated (cf. Table 2), namely: a) The utilization of an Ontology Engineering methodology; b) the inclusion of Ontology Design Patterns (ODPs); c) the reuse of existing well-known ontologies; d) the availability of ontology resources, i.e., on Github, or Linked Open Vocabularies (LOV)[14]; e) the language used as an input for the ontology, e.g., CAEX, f) the language utilized to describe the ontology, e.g. OWL, and, finally g) the main use case in which the ontology is used.

Overall, existing works lack common desirable features for an AutomationML ontology. First, previous ontologies are tailored for specific use cases and do not consider all the details of the AutomationML standard. Second, most of the existing ontologies are designed without considering any methodology for ontology design or best practices such as the inclusion of ontology design patterns or reusing well-known vocabularies. Lastly, besides being described in papers existing ontologies are not available for consulting or reusing. We addressed all these gaps with AMLO.

[14]Linked Open Vocabularies: http://lov.okfn.org/dataset/lov

7. Conclusions

We presented the AutomationML ontology (AMLO) that covers the AutomationML data exchange standard in the industrial engineering domain. AMLO provides concepts to support the design of an engineering system - components and subcomponents, required and supported functionality of the components, various attributes (e.g., mechanical, electrical of logical ones), logical and physical connections between the system elements, to name the main ones. The ontology design process was based on domain-specific and ontological requirements that were identified for the AutomationML context. Particular attention during the AMLO design was given to following the best practices for ontology design (acc. to [14]). The resulting ontology covers completely the XSD schema for AutomationML and provides means for enhancing the engineering data with additional resources (e.g., by including the Ontology of Units of Measure). We also showed how the AMLO can be used in real life scenarios to improve the engineering processes during system design.

Future work includes extending the ontology with the COLLADA and PLCOpen fragments of the AutomationML standard. Additionally, we will explore what other resources in the industrial engineering area have semantic representations available and would be useful to link with the AMLO to provide more possibilities for enhancing the original AutomationML data content and data structure.

References

[1] L. Abele, C. Legat, S. Grimm, and A. W. Müller. Ontology-based validation of plant models. In *INDIN*, pages 236–241. IEEE, 2013.

[2] M. Barnes and E. L. Finch. COLLADA-Digital Asset Schema Release 1.5. 0 specification. Technical report, Khronos Group, Sony Computer Entertainment Inc, 2008.

[3] T. Bauernhansl, M. ten Hompel, and B. Vogel-Heuser. Industrie 4.0 in Produktion, Automatisierung und Logistik: Anwendung, Technologien. *Migration. Wiesbaden: Springer Vieweg*, 2014.

[4] S. Biffl, O. Kovalenko, A. Lüder, N. Schmidt, and R. Rosendahl. Semantic mapping support for Mechatronic Objects in AutomationML. In *AutomationML User Conference, Blomberg, Germany*. IEEE, 2014.

[5] S. Biffl, A. Lüder, and D. Winkler. Multi-disciplinary engineering for industrie 4.0: Semantic challenges and needs. In S. Bill and M. Sabou, editors, *Semantic Web for Intelligent Engineering Applications*, pages 17–51. Springer, Springer International Publishing, 2016.

[6] A. Björkelund, J. Malec, K. Nilsson, and P. Nugues. Knowledge and skill representations for robotized production. In *Proceedings of the 18th IFAC Congress, Milan*, 2011.

[7] V. Damjanovic, W. Behrendt, M. Plößnig, and M. Holzapfel. Developing ontologies for collaborative engineering in mechatronics. In *The Semantic Web: Research and Applications, 4th European Semantic Web Conference, ESWC, Innsbruck, Austria, June 3-7, Proceedings*, pages 190–204, 2007.

[8] R. Drath. *Datenaustausch in der Anlagenplanung mit AutomationML: Integration von CAEX, PLCopen XML und COLLADA*. Springer-Verlag, 2009.

[9] eClass e.V. eCl@ss - Standardized Material and Service Classification, 2016. `http://www.eclass.eu/`.

[10] M. Fedai, U. Epple, R. Drath, and A. Fay. A metamodel for generic data exchange between various cae systems. In *Proceedings of 4th mathmod conference*, volume 24, pages 1247–1256, 2003.

[11] A. Gangemi and V. Presutti. Ontology Design Patterns. In *Handbook on Ontologies*, pages 221–243. 2009.

[12] M. Glawe, C. Tebbe, A. Fay, and K. Niemann. Knowledge-based engineering of automation systems using ontologies and engineering data. In *KEOD Proceedings of the International Conference on Knowledge Engineering and Ontology Development, Volume 2, Lisbon, Portugal, November 12-14*, pages 291–300, 2015.

[13] I. Grangel-González, D. Collarana, L. Halilaj, S. Lohmann, C. Lange, M.-E. Vidal, and S. Auer. Alligator: A deductive approach for the integration of industry 4.0 standards. In F. Vitali and E. Blomqvist, editors, *Knowledge Engineering and Knowledge Management - 20th International Conference, EKAW, Bologna, Italy, November 19-23. Proceedings*. Springer, 2016.

[14] A. Gray and M. Sabou. ISWC resources track review instructions. *ISWC*, 2016. `https://dx.doi.org/10.6084/m9.figshare.2016852`.

[15] L. Halilaj, N. Petersen, I. Grangel-González, C. Lange, S. Auer, G. Coskun, and S. Lohmann. Vocol: An integrated environment to support version-controlled vocabulary development. In *Knowledge Engineering and Knowledge Management - 20th International Conference, EKAW, Bologna, Italy, November 19-23, 2016, Proceedings*, pages 303–319, 2016.

[16] M. Hepp. Goodrelations: An Ontology for describing Products and Services Offers on the web. In *Knowledge Engineering: Practice and Patterns, 16th International Conference, EKAW 2008, Acitrezza, Italy, September 29 - October 2, 2008. Proceedings*, pages 329–346, 2008.

[17] Y. Hua, S. Zander, M. Bordignon, and B. Hein. From automationml to ROS: A model-driven approach for software engineering of industrial robotics using ontological reasoning. In *21st IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2016, Berlin, Germany, September 6-9, 2016*, pages 1–8, 2016.

[18] IEC. Iec 62714-1:2014 engineering data exchange format for use in industrial automation systems engineering - automation markup language, 2014.

[19] E. Kharlamov, B. C. Grau, E. Jiménez-Ruiz, S. Lamparter, G. Mehdi, M. Ringsquandl, Y. Nenov, S. Grimm, M. Roshchin, and I. Horrocks. Capturing industrial information models with ontologies and constraints. In P. Groth, E. Simperl, A. Gray, M. Sabou, M. Krötzsch, F. Lecue, F. Flöck, and Y. Gil, editors, *The Semantic Web – ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II*, pages 325–343. Springer International Publishing, 2016.

[20] O. Kovalenko and J. Euzenat. Semantic matching of engineering data structures. In S. Bill and M. Sabou, editors, *Semantic Web for Intelligent Engineering Applications*. Springer, 2016.

[21] O. Kovalenko, M. Wimmer, M. Sabou, A. Lüder, F. J. Ekaputra, and S. Biffl. Modeling automationml: Semantic web technologies vs. model-driven engineering. In *ETFA*, pages 1–4. IEEE, 2015.

[22] O. Kovalenko, M. Wimmer, M. Sabou, A. Lüder, F. J. Ekaputra, and S. Biffl. Modeling automationml: Semantic web technologies vs. model-driven engineering. In *20th IEEE Conference on Emerging Technologies & Factory Automation, ETFA 2015, Luxembourg, September 8-11, 2015*, pages 1–4, 2015.

[23] S. Liu, J. Mei, A. Yue, and Z. Lin. *XSDL: Making XML Semantics Explicit*, pages 64–83. Springer Berlin Heidelberg, 2005.

[24] C. Pedrinaci, J. Cardoso, and T. Leidig. Linked usdl: a vocabulary for web-scale service trading. In *The Semantic Web: Trends and Challenges*, pages 68–82. Springer, 2014.

[25] S. Peroni, A. Gangemi, and F. Vitali. Dealing with markup semantics. In *Proceedings of the 7th International Conference on Semantic Systems*, I-Semantics '11, pages 111–118, New York, NY, USA, 2011. ACM.

[26] J. Persson, A. Gallois, A. Björkelund, L. Hafdell, M. Haage, J. Malec, K. Nilsson, and P. Nugues. A knowledge integration framework for robotics. In *ISR/ROBOTIK 2010, Proceedings for the joint conference of ISR 2010 (41st Internationel Symposium on Robotics) und ROBOTIK 2010 (6th German Conference on Robotics), 7-9 June 2010, Munich, Germany - Parallel to AUTOMATICA*, pages 1–8, 2010.

[27] H. S. Pinto and J. P. Martins. Ontologies: How can they be built? *Knowledge and Information Systems*, 6(4):441–464, 2004.

[28] M. Poveda-Villalón. A reuse-based lightweight method for developing linked data ontologies and vocabularies. In *The Semantic Web: Research and Applications*, pages 833–837. Springer, 2012.

[29] V. Presutti, E. Blomqvist, E. Daga, and A. Gangemi. Pattern-based ontology design. In M. C. Suárez-Figueroa, A. Gómez-Pérez, E. Motta, and A. Gangemi, editors, *Ontology Engineer-*

ing in a Networked World, pages 35–64. Springer Berlin Heidelberg, 2012.

[30] K. Ridgway, C. Clegg, and D. Williams. The Factory of the Future, Future Manufacturing Project: Evidence Paper 29. *London: Foresight, Government Office for Science*, 2013.

[31] H. Rijgersberg, M. van Assem, and J. L. Top. Ontology of units of measure and related concepts. *Semantic Web*, 4(1):3–13, 2013.

[32] M. Sabou. An introduction to semantic web technologies. In S. Biffl and M. Sabou, editors, *Semantic Web Technologies for Intelligent Engineering Applications*, pages 53–81. Springer International Publishing, 2016.

[33] M. Sabou, O. Kovalenko, F. J. Ekaputra, and S. Biffl. Semantic web solutions in engineering. In S. Biffl and M. Sabou, editors, *Semantic Web Technologies for Intelligent Engineering Applications*, pages 281–296. Springer International Publishing, 2016.

[34] M. Sabou, O. Kovalenko, and P. Novák. Semantic modelling and acquisition of engineering knowledge. In *Semantic Web Technologies for Intelligent Engineering Applications*, pages 105–136. Springer, 2016.

[35] N. Schmidt and A. Lüder. AutomationML and eClass integration, 2015.

[36] T. Tudorache. Employing ontologies for an improved development process in collaborative engineering. 2006.

[37] M. Uschold and M. King. *Towards a methodology for building ontologies*. Citeseer, 1995.