Modeling AutomationML: Semantic Web Technologies vs. Model-Driven Engineering

Olga Kovalenko[1], Manuel Wimmer[1], Marta Sabou[1], Arndt Lüder[2], Fajar J. Ekaputra[1], Stefan Biffl[1]
[1]Christian Doppler Laboratory Software Engineering Integration for Flexible Automation Systems
Vienna University of Technology, Vienna, Austria
<firstname>.<lastname>@tuwien.ac.at
[2]Otto-v.-Guericke University, Faculty Mechanical Engineering, Magdeburg, Germany
<firstname>.<lastname>@ovgu.de

***Abstract* — Modeling engineering knowledge explicitly and representing it by means of standardized modeling languages and in machine-understandable form enables advanced engineering processes in industrial and factory automation. This affects positively both process and product quality. In this paper we explore how the AutomationML format, an emerging data exchange standard, that supports the Industry 4.0 vision, can be represented by means of two established modeling approaches – Model-Driven Engineering (MDE) and Semantic Web. We report observed differences w.r.t. resulting model features and model creation process and, additionally, present the application possibilities of the developed models for engineering process improvement in a production system engineering context.**

Keywords—modeling engineering knowledge, enterprise models, manufacturing, ontology, ontology engineering, model driven engineering, AutomationML

I. INTRODUCTION

Production system engineering (PSE) projects are highly heterogeneous and collaborative environments. The stakeholders span diverse engineering disciplines (e.g., mechanical, electrical, software engineering) and make use of a diverse set of software tools. Therefore, in a PSE project there may be a multitude of tools, which use different data models for the same engineering concepts. Despite their heterogeneity, the stakeholders need to collaborate for designing and building a complex production system (e.g., a hydro power plant or a steel mill). Such dynamic and complex engineering settings require improved methods and tool support to facilitate effective and efficient communication and data management, with a special focus on supporting engineering activities across discipline and tool boundaries.

The AutomationML (AML) standard [3] for tool data exchange provides a foundation for harmonizing engineering data exchange in heterogeneous PSE projects. Although facilitating data exchange is already an important improvement, there is still a lack of infrastructure for supporting advanced engineering activities across the disciplines and tools in PSE projects, e.g., data linking, change propagation across connected datasets, data analysis and consistency checking.

To this end, the well established approaches that have been used for these tasks in software and knowledge engineering are, of course, also promising solution candidates for the PSE domain. In this paper we explore the capabilities of Model Driven Engineering (MDE) and Semantic Web to contribute to engineering process improvement in PSE. Both areas comprise a stack of technologies and tools ready to be applied and have been used for a wide range of applications, including data linking, constraints checking, data integration, and automatic derivation of new knowledge from existing knowledge bases (e.g., [9]). Both approaches require building an explicit and formal model to capture the knowledge and data in PSE projects. Within this paper we focus on this knowledge capturing process. Taking AML as a starting point, we describe the process of obtaining a representation model with both MDE and Semantic Web approaches. We discuss the resulting models w.r.t. the efforts needed to create them and their potential capabilities and usages for engineering process improvement in PSE.

II. AUTOMATIONML FOR INDUSTRIAL AND FACTORY AUTOMATION

AML is a neutral, open, and XML-based data exchange format that enables the consistent and lossless exchange of engineering information related to manufacturing system topology, geometry, kinematics, and control behavior [3]. AML is not a completely new format, but rather consists of existing formats (CAEX [4], COLLADA [1] and PLCopen [8]), which were extended, adapted, and combined appropriately. AML makes use of the following four main CAEX concepts:

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, and references.

InterfaceClassLibrary defines all interfaces needed to describe a plant model. An *interface class* can be used for two purposes: a) to define relationships between objects in a CAEX file, i.e., between objects of a plant topology (e.g., to connect a sensor with a PLC); and b) to define references to information, which is stored outside the CAEX file.

978-1-4673-7929-8/15/$31.00 ©2015 IEEE

RoleClassLibrary allows defining the requirements by specifying the required properties of the equipment objects. A *role class* describes a physical or logical object as an abstraction of a concrete technical realization, thus giving a meaning to an object enabling automatic semantic interpretation by a tool. A role class may also define attributes, which are common for an object. A role class can be assigned to objects within the *SystemUnitClassLibrary* and *InstanceHierarchy* to formally define the object type.

SystemUnitClassLibrary allows specifying the provided properties of solution equipment objects that can be matched with the requirements of objects in the role class library. A *System Unit* class describes a physical or logical object including the concrete technical realization and internal architecture, thus representing a specific incarnation of a role class. Therefore, *System Unit* classes define vendor-specific classes of objects with multiple hierarchy levels and are instantiated within the *InstanceHierachy*.

III. Representing AML with MDE

MDE has been applied in different domains, whereas the most studied application domain seems to be software engineering **Error! Reference source not found.**. The main goals of MDE are to have models explicitly and machine-readable represented by defining modeling languages with so-called metamodels in order to apply model transformations for code generation and model management.

Aim. The initial aim for creating a model-based solution for AML was to support management tasks such as validation, comparison, versioning, and linking possibilities for AML artefacts across discipline/tool boundaries in PSE. This requires a full round-tripping solution for moving from AML XML files to AML models, and vice versa.

Creation Process. For developing the AML metamodel **Error! Reference source not found.**, we resorted on previous experiences in metamodel mining which is based on a semi-automatic transformation of existing (legacy) language specifications into metamodels based on the Eclipse Modeling Framework[1] [10]. The metamodel creation is threefold. **(1)** ***Language artefact selection***: the primary source of the language definition in the case of AML were the XML Schema Definitions (XSDs), namely CAEX XSD in version 2.15; **(2)** ***Metamodel recovery and improvement***: first, an automatic transformation from XSD to Ecore-based metamodels is executed to derive an initial metamodel version, and second, the initial metamodel is manually refined to incorporate more language semantics. In the first step our aim was to provide a complete and correct metamodel definition, which is equivalent to the provided XSDs, to generate efficient tool support. In the second step, we optimized the output, i.e., the metamodel to reflect the AML language definition and to satisfy the graph-based nature of AML artefacts, which is hidden in the corresponding XSD as XML is a hierarchical data model and metamodels describe naturally graph-based structures. In particular, we introduced in addition to the available attributes, which describe path expressions to elements, references which point to those referenced elements; **(3)** ***Model injection/extraction***. To transform data from a source AML file to its model representation and vice versa, we have partially reused and customized a generic injection/extraction transformation between XML data and EMF models. A dedicated processor deals on the model-level with the manual improvements done in the metamodel improvement step.

Resulting artefacts. The obtained metamodel allows to fully represent CAEX files as models. Consequently, all MDE techniques and tools can be applied for CAEX as well.

Sample usage. Consider the consistency check: "*All Internal Elements within an InstanceHierarchy must have unique iDs defined*". To define such constraints, which should be directly enforced during modeling within the modeling editor or checking the constraints in batch mode to reason about the consistency of a given file, the *Object Constraint Language* (OCL) [7] is often used in MDE. Additionally, for sophisticated tool support and enhanced guidance in the modeling process, task specific languages can be used. For instance, the Epsilon Validation Language (EVL) [2], which allows defining in addition to the constraints the error messages for describing the inconsistency for humans and repair rules if a constraint is violated, to better understand and fix the problem. For our example we are able to define (i) the check for finding violations of the uniqueness constraint, (ii) a message to make the consistency violation human-understandable, and (iii) an automatic repair operation, which may be triggered by the user to assign automatically a new unique id. Thus, having a domain-specific language to define such consistency constraints is beneficial to provide not only a logical constraint, but also language support to deal with violations from a user's point of view.

IV. Representing AML with Semantic Web

Semantic Web technologies support the integration of distributed and heterogeneous data, such as those available on the Web and other domains with similar characteristics [9]. Various technology elements can be used for this ranging from knowledge representation in OWL, rule definitions, querying, etc. Here we focus on the ontology creation as this is the pre-requisite for applying further querying and reasoning methods.

Aim. The aim for the ontology creation was to support data analysis activities across the discipline/tool boundaries in PSE.

Creation Process. We followed a top-down modelling approach and the ontology engineering methodology defined by Noy et.al [6]. The ontology building can be divided into following steps:

(1) ***Scope definition***. The AML ontology aims to cover the PSE domain. Its intended usages are facilitating AML data exchange and analysis across different engineering disciplines/tools.

[1] https://www.eclipse.org/modeling/emf/

(2) *Ontology design*. The AML ontology structure was defined based on a) core AML documents, i.e., the CAEX standard [4] and AML specification [5] (identifying the core concepts in the domain and relations between them); and b) discussions with domain experts (refining the structure based on formulated consistency checks typical in AML usage scenarios). As each AML file can have its own *Interfaces*, *Roles* and *System Units* topologies, it is challenging to build one overarching ontology covering all possible AML files. We therefore distinguish between two ontology structures:

A *core AML ontology* that reflects the main design decisions to represent CAEX structures in OWL, i.e., CAEX structures to be found in **any** AML file that are not discipline or tool specific (see Figure 2). Lightweight and concise, the core AML ontology, together with a specific AML file serve as an input for the CAEX to OWL transformation, which produces the extended AML ontology as an output.

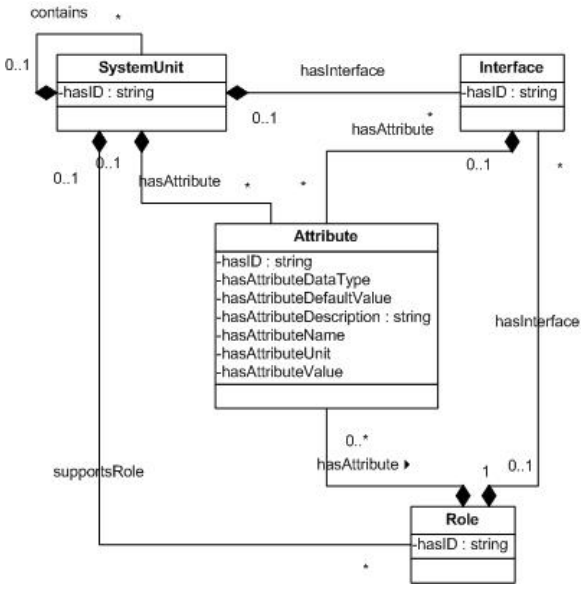

Figure 2: Core AML Ontology Structure (UML diagram).

An *extended AML ontology* represents the complete semantics and data from an AML file in OWL. It comprises all the *Roles*, *System Units* and *Interfaces* defined in an AML file as well as all the *Instance Hierarchy* data of this AML file.

(3) *Semantic lifting of AML data*. To transform data from the source AML files into an ontology, we defined the transformation from AML into OWL[2] and implemented it with Apache Jena[3]. During the transformation the complete class hierarchies for *Interfaces*, *Roles* and *System Units* are extracted from the AML file and explicitly captured. These hierarchies constitute the vocabulary necessary for describing a specific production system. The *Instance Hierarchy* elements are then converted into instances of *System Units* classes, finishing the transformation.

Resulting artefacts. The obtained ontology captures all necessary semantics and data from a given AML file to apply reasoning and querying, e.g., for consistency checking.

Sample usage. Let us consider the consistency check: "*All controller devices in a production system must have exactly 1 connection to automation object defined*". The "controller" *Role* is not defined explicitly in the AML hierarchy and must be defined separately for each AML topology of roles.

The ontology-based representation of AML data allows applying querying (via SPARQL[4]) and reasoning (e.g., using SWRL[5]) for flexible definition and executing of such consistency checks. In this case, the explicitly represented *Roles* hierarchy allows applying reasoning to enrich the existing classification, e.g., the following SWRL rule can be defined to automatically classify the controllers:

```
SystemUnit(?device) ∧ Role(?role) ∧
supportsRole(?device, ?role) ∧
Ro_MechatronicAssembly →  Controller(?device)
```

This rule allows enriching 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.

V. DISCUSSION

We analyse the differences of MDE and Semantic Web approaches for modelling AML and discuss the benefits and limitations of the model creation process and resulting models.

Modeling Process. In terms of the effort needed to build a model, creating a model with MDE approach can be automated, if the semi-structured representation (e.g., XSD in case of AML) is available. The conversion of an XSD into an Ecore metamodel is mostly straight-forward but language engineers may consider performing semantic enrichments. However, if changes are made in the metamodel, also the data transformations have to be adapted.

In contrast, when modeling with Semantic Web technologies and according to a top-down approach, designing an ontology was an iterative process with domain experts in the loop. This leads to a longer time for model creation, as the knowledge engineer has to get a detailed understanding of the modeled domain together with expected application to design a suitable model. Also, several verification iterations with domain experts are required during model creation. Defining a complete data transformation from AML format into ontology is more complex, as the resulting model structure differs from the original AML XSD (both on syntactic and semantic level). However, the advantage is obtaining a light-weight, concise and application-tailored model as a result.

[2] http://www.w3.org/TR/owl-ref/
[3] https://jena.apache.org
[4] http://www.w3.org/TR/rdf-sparql-query/
[5] http://www.w3.org/Submission/SWRL/

Resulting model. The obtained resulting models (metamodel and ontology) differ in many aspects for two major reasons: a) the data representation language (Ecore and OWL); and b) the provided reasoning capabilities.

Ecore enables specifying dynamic behaviour, for instance, defining a model simulator for AML. This is not directly possible in OWL. The Object Constraint Language (OCL) [7] complements Ecore and enables specifying query operations, derived values, pre- and post-conditions. Another advantage is the straight-forward mapping from Ecore to Java, which makes the programmatic access to models supported out-of-the-box. A limitation, however, is that the Ecore model must stay stable, i.e., once defined and populated with data, it is challenging to modify the metamodel, e.g., changing a class or property, and would lead to co-evolution problems for the existing models. Because of this limitation, it was decided to store all AML data (*Role* and *SystemUnit* classes and *Internal Elements* from the *Instance Hierarchy*) on the instance level. On a downside, this potentially makes model management such as model queries and validations more challenging. Also, by representing everything on the instance level, there is no out-of-the-box semantics available for roles and classes, e.g., the inheritance between roles and classes is just a simple data link, which is not automatically considering inherited elements in query tasks.

OWL, in contrast to Ecore, is flexible w.r.t. class descriptions and supports various ways to define classes (e.g., by class identifier, property restrictions etc.). Classes can also constitute nested hierarchies, which makes explicit type definition (e.g., for consistency checking) not mandatory. Type inference can be performed based on class descriptions, thus enabling dynamic classification of objects into classes. Also, unlike Ecore, OWL is flexible w.r.t. modifications on schema level, i.e., modifying classes or properties, if there is a need to update an existing ontology. Another important feature is that OWL allows definition of transitive properties, which simplifies queries formulation for some cases. This is not explicitly possible with Ecore/OCL and may require some workarounds, e.g., using derived features.

Usage. The AML ontology is well suited for applications such as data analysis and consistency checking, both because it was designed with those in mind and because of the OWL features that make ontologies useful for performing those tasks. For example, reasoners are available to process ontologies for consistency checking, concept satisfiability, instance and concept classification. Some of the reasoners also provide axiom explanation, if violations have been found, facilitating understanding, localizing and fixing activities for engineers.

The AML metamodel allows applying model transformation languages to perform in-place transformations on AML data, such as providing improvements to a model, as well as out-place transformations such as transforming AML to other systems modeling languages or for mapping AML to formal languages, which facilitate model analysis. Furthermore, highly task specific languages (e.g., model validation, model comparison, model merging languages) are available and proved useful in several engineering scenarios.

An important difference for applying AML Ecore metamodels and AML ontology is that semantics in MDE approach adopts Closed World Assumption, while OWL adopts an Open World Assumption by default. This might lead to different results during querying and reasoning. However, it is also possible to force OWL reasoning and querying under Closed World Assumption.

A limitation is that applying both models in PSE requires sufficient understanding from developers and engineers. Bridges between the modelware and ontoware will allow combining the benefits of both worlds.

VI. Summary and Future Work

In this paper we presented our experiences for representing AutomationML with two different modeling approaches: MDE and Semantic Web. For each of these approaches, we described the model creation process, the resulting model and presented an example of model usage for engineering process improvement in the production system engineering. Furthermore, we analysed the obtained models (metamodel and ontology) w.r.t. to their quality and potential usages. Future work includes providing data exchange between both representations (Ecore and OWL), so that their benefits could be exploited (if necessary) within the same system.

VII. References

[1] COLLADA 1.4.1:March08 COLLADA – Digital Asset Schema Release 1.4.1, http://www.khronos.org/files/collada_spec_1_4.pdf . 2008

[2] Dimitrios K.S., Paige R.F., Polack F.A.C.. On the evolution of OCL for capturing structural constraints in modelling languages. In *Rigorous Methods for Software Construction and Analysis*, J. R. Abrial and U. Glässer (Eds.). Springer-Verlag, Berlin, Heidelberg, pp: 204-218, 2009.

[3] Drath R., ed., Datenaustausch in der Anlagenplanung mit AutomationML: Integration von CAEX, PLCopen XML und COLLADA. Springer, 2010.

[4] IEC 62424:2008, Representation of process control engineering – Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools, 2008.

[5] IEC 62714:2014 (all parts), Engineering data exchange format for use in industrial systems engineering – Automation Markup Language.

[6] Noy N., and McGuinness D.L.. Ontology development 101. *Knowledge Systems Laboratory, Stanford University*, 2001.

[7] Object Modeling Group: Object Constraint Language Specification, version 2.3.1. http://www.omg.org/spec/OCL, 2012.

[8] PLCopen XML2.0:December3rd 2008 and PLCopen XML2.0.1:May 8th 2009, XML, formats for *IEC 61131*-3, http://www.plcopen.org.

[9] Runde S., Fay A.. Software Support for Building Automation Requirements Engineering-An Application of Semantic Web Technologies in Automation. IEEE Trans. Ind. Inf. 7(4):723-730. 2011.

[10] Schauerhuber A., Wimmer M., Kapsammer E., Schwinger W., Retschitzegger W.: Bridging WebML to model-driven engineering: from document type definitions to meta object facility. IET Software 1(3):81-97. 2007.

[11] S. Biffl, A. Lüder, E. Mätzler, N. Schmidt and M. Wimmer, "Linking and Versioning Support for AutomationML: A Model-Driven Engineering Perspective", Accepted for INDIN, 2015.