Generating Structured AutomationML Models from IEC 62264 Information

Bernhard Wally
Business Informatics – SW Engineering
JKU Linz
Linz, Austria
bernhard.wally@jku.at

Laurens Lang
Information Systems Engineering
TU Wien
Vienna, Austria
e11719751@student.tuwien.ac.at

Rafał Włodarski
Information Systems Engineering
TU Wien
Vienna, Austria
e1327160@student.tuwien.ac.at

Radek Šindelář
Business Informatics – SW Engineering
JKU Linz
Linz, Austria
radek.sindelar@jku.at

Christian Huemer
Information Systems Engineering
TU Wien
Vienna, Austria
huemer@big.tuwien.ac.at

Alexandra Mazak
Business Informatics – SW Engineering
JKU Linz
Linz, Austria
alexandra.mazak@jku.at

Manuel Wimmer
Business Informatics – SW Engineering
JKU Linz
Linz, Austria
manuel.wimmer@jku.at

***Abstract*—AutomationML provides a versatile modeling environment for the description of production systems. However, when starting a new AutomationML project, or when serializing existing data with the AutomationML format, there are no rules on how to structure these models in a meaningful way. In this work, we present an approach for structuring AutomationML models, based on the IEC 62264 standard. In our approach we are implementing the process of serializing IEC 62264 information declaratively, by leveraging the power of model transformations, as they are applied in the context of *model-driven (software) engineering*.**

***Index Terms*—Domain Specific Language, Model Transformation, AutomationML, IEC 62264**

I. Introduction

AutomationML provides a versatile environment for the modeling of role-based technical systems [1]. Its application for the description of equipment hierarchies (representing, e.g., production systems) is well described and supported by a number of role class libraries [2]. However, modeling more complex systems, including information about processes and materials such as described in [3], requires clear modeling concepts in order to keep AutomationML documents maintainable.

Recently, AutomationML e.V. has released an application recommendation for the creation of AutomationML documents that encode information in accordance with IEC 62264 [4]. IEC 62264 is an established standard for the description of manufacturing operations and their resources [5, 6]. It has been adopted from the American standard ISA-95 [7] and is considered a suitable standard for the encoding of manufacturing execution information [8]. This application recommendation is a result of discussions within the AutomationML working group that investigated application scenarios on "higher levels" of the automation hierarchy [9].

In this work, we present a format for the structuring of AutomationML information based on modeling concepts of IEC 62264. Our approach uses a graphical domain specific language for the expression of production system and manufacturing operations information, as well as formalized model transformation rules for encoding this information using language constructs of AutomationML. Technology-wise, we are using the modeling framework of the Eclipse platform as our technology stack [10], with EMF/Ecore[1] as our metamodeling infrastructure.

II. IEC 62264

IEC 62264 provides a set of metamodels for the definition of several parts of a production system. The four basic kinds of resources *personnel*, *equipment*, *physical assets*, *material* are complemented with *process segments* and several kinds of *operations information* that support the definition of a variety of manufacturing approaches, including bills of material and bills of processes [5].

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

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

Personnel: "Information about specific personnel, classes of personnel, and qualifications of personnel" [5].

Equipment: "Information about specific equipment, the classes of equipment, and equipment capability tests" [5].

Physical Assets: "Information about the physical piece of equipment, usually managed as a physical asset within the enterprise often utilizing a specific serial number" [5].

Material: "Defines the actual materials, material definitions, and information about classes of material definitions" [5].

Process Segments: "A hierarchical model, in which multiple levels of abstraction of manufacturing processes may be defined because there can be multiple business processes requiring visibility to manufacturing activities" [5].

III. IEC 62264 Designer

Domain-specific languages (DSLs) are a tool for expressing expert knowledge using expert terminology. Methods and techniques for the definition and application of such DSLs have been investigated and developed in the context of model-driven software engineering [11]. Given its wide use in software engineering, the Meta Object Facility (MOF) has proven a viable way for constructing and using DSLs [12]. The usual approach for constructing a DSL is (i) defining a metamodel of the domain (the abstract syntax), and (ii) defining the concrete syntax to be used for describing elements of this domain. Two types of concrete syntax are common: (i) textual, and (ii) graphical. Prominent examples for graphical syntaxes are Unified Modeling Language (UML) class diagrams, entity-relationship diagrams or diagrams created according to the business process model and notation (BPMN).

Given a metamodel describing the IEC 62264 domain, formulated using Ecore, it is relatively simple to create interactive graphical toolkits on top. Several development frameworks are available, in this work we have chosen to use Sirius[2] as the visualization engine. In Sirius, the visual syntax it declaratively defined, as depicted in Fig. 1. Fig. 2 shows the resulting visualization of an IEC 62264 equipment hierarchy, Fig. 3 shows the result of such a visualization declaration by the example of an IEC 62264 process segment.

With this tool at hand, it is relatively simple and intuitive to create a fairly complex IEC 62264 model of a production system. In its current state, the *IEC 62264 Designer* supports the creation and visualization of the basic resources (personnel, equipment, physical assets, material; all except for test specifications and test results), as well as process segments and operations definitions.

[2]cf. https://www.eclipse.org/sirius/

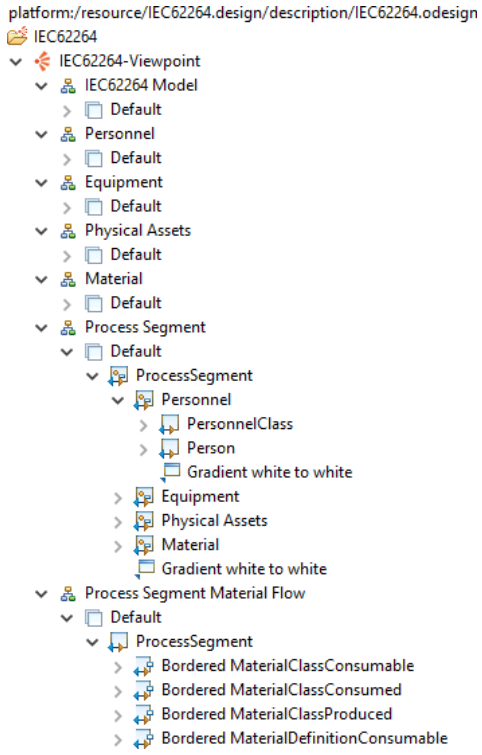

Fig. 1. Defining visualizations and interaction points using the Eclipse Sirius framework.

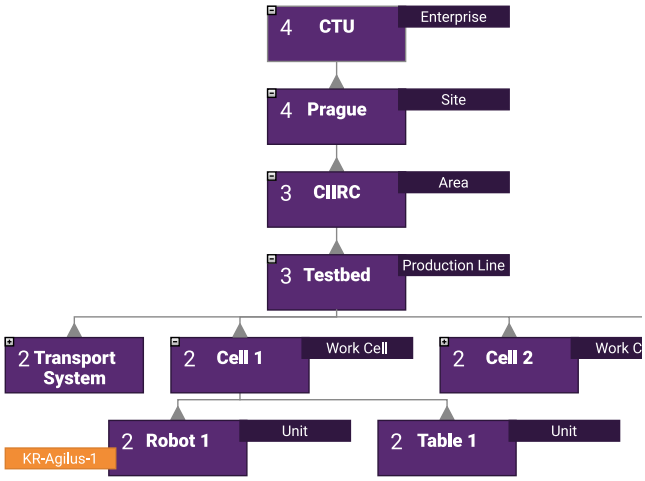

Fig. 2. Excerpt of an equipment hierarchy, as defined in an IEC 62264 model.

IV. IEC 62264 ⇌ AutomationML

In this section we will elaborate on the conversion between IEC 62264 models and AutomationML models.

Model transformations are at the core of model-driven engineering [13], as they provide the means and methods for converting data within a domain or even across domains. Different approaches have been developed for the realization of such model transformations [14], some of them sporting a unidirectional nature, others providing bidirectional semantics [15].

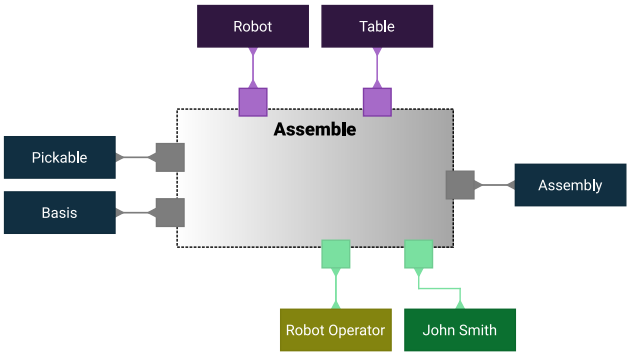

Fig. 3. IEC 62264 process segment "Assemble", as defined with the help of the Sirius framework.

For our implementation we are using the ATL transformation language[3], which belongs to the group of unidirectional transformation languages. ATL is a rule-based language, where elements of different domains can be converted into one another by specifying a set of rules. Concrete examples for such rules are provided below.

In order to have a framework for bidirectional conversion, two separate transformation modules need to be developed. It might sound counter-intuitive to use two unidirectional transformation modules instead of one bidirectional transformation, however, the advantage of this approach is that the two transformations can be implemented with varying strictness. As we have mentioned earlier, we are imposing a certain structure on the AutomationML model that is generated from IEC 62264 information. Yet, this kind of structure is not mandatory, and in order to extract IEC 62264 information from an AutomationML document this structural provisions need to be neglected.

Further, we have abstracted the AutomationML language in two layers: first, we have reverse-engineered the XML schema of CAEX (Computer Aided Engineering Exchange), which provides the syntactic basis for AutomationML, into a metamodel. CAEX itself is standardized as IEC 62424 [16]. The result is a metamodel that we call *CAEX XML-based*. Second, we have constructed a metamodel of CAEX that represents a strictly object-oriented view on CAEX elements. Since the metamodel elements are related via associations, the metamodel represents a graph. Therefore, we call this metamodel *CAEX graph-based*.

For instance, the graph-based metamodel describes explicitly (within the formalisms of the modeling language) that the target of a role requirement must be a role class, as specified in a role class library. This is fundamentally different from the XML-based CAEX metamodel, where the reference to a role class is expressed as a string; and in the accompanying documentation it is stated that this string shall represent a path to a role class following a certain syntax. For a model-driven framework, the graph-based metamodel is of higher value as it enables many "out-of-the-box" features of generic modeling tools, that are not available for the XML-based metamodel due to the lack of formal definitions in several places.

As such, when we work with AutomationML data in model-driven engineering, we usually use the graph-based CAEX model, and only transform to the the XML-based CAEX model, when required. Consequently, the workflow for transforming between IEC 62264 information and the AutomationML exchange format (which corresponds to XML-based CAEX) requires an additional step, which is the transformation between the XML-based and the graph-based CAEX metamodels. The workflow, expressed in BPMN, is depicted in Fig. 4.

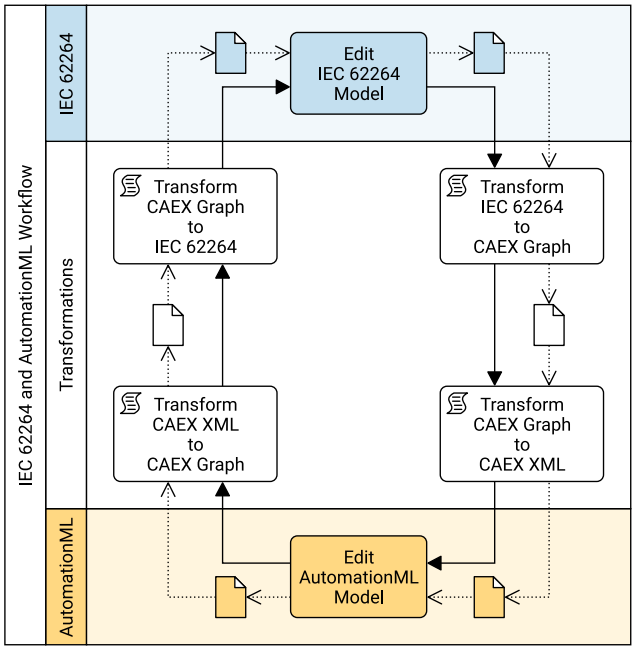

Fig. 4. Workflow of our approach, depicted using BPMN.

An example of an ATL rule for transforming graph-based internal elements into XML-based internal elements is given in Lst.1. Of course, this transformation rule is rather simple, as it mostly copies data between the two elements. The only mapping that is a little bit more complex that of the `refBaseSystemUnitPath`: in the graph-based model it is represented by a formal relation, called `baseSystemUnit`, in the XML-based model it is a string representing a path to the referenced element. This conversion is conducted in a helper method named `getPath` (cf. lines 12–13).

A. Transformation Rules

We have started with a formal implementation of the mapping rules for modeling IEC 62264 information in the context of AutomationML, as it is defined in [4]. Besides the application of mandatory modeling rules, we have chosen to structure the AutomationML instance hierarchy in a semantically meaningful way that allows to quickly access required information.

[3]cf. https://www.eclipse.org/atl/

The "entry" ATL rule of the transformation from IEC 62264 to AutomationML is given in Lst. 2: an IEC 62264 model is converted into an AutomationML document.

Listing 1. ATL rule for converting `InternalElements` from the *graph-based* metamodel (named `CAEX`) into `InternalElementTypes` of the *XML-based* metamodel (named `AML`).

```
rule InternalElement extends SystemUnitClass {
   from
      caex: CAEX!InternalElement
   to
      aml: AML!InternalElementType (
         attribute <- caex.attributes,
         externalInterface <-
            caex.externalInterfaces,
         internalElement <-
            caex.internalElements,
         internalLink <- caex.internalLinks,
         refBaseSystemUnitPath <- thisModule.
            getPath(caex.baseSystemUnit),
         roleRequirements <-
            caex.roleRequirements,
         supportedRoleClass <-
            caex.supportedRoleClasses
      )
}
```

Listing 2. ATL rule for converting an IEC 62264 `Model` into an element of type `CaexFile`.

```
rule Model2File {
   from
      model: IEC62264!Model
   to
      file: CAEX!CaexFile (
         superiorStandardVersions <-
            Sequence{'AutomationML 2.10'}
         fileName <- model.name,
         version <- version,
         sourceDocumentInformations <- sdi,
         instanceHierarchies <- hierarchy,
         roleClassLibs <- roleClassLib,
         systemUnitClassLibs <-
            systemUnitClassLib,
      ),
      version: CAEX!Version (
         version <- model.version
      ),
      sdi: CAEX!SourceDocumentInformation (
         lastWritingDateTime <-
            sdi.currentDateTime()
      ),
      -- ...
}
```

The current state of implementation supports the transformation of basic resources (personnel, equipment, physical assets, material), process segments, and resource relationship networks. Transformations are supported in both directions. Currently unsupported are the different kinds of test specifications and test results. They are to be added in future work.

B. Document Structure

The structure that we are imposing on the AutomationML document follows the categorization of the IEC 62264 standard. With respect to the instance hierarchy, we are first creating a "root" internal element that acts as a common parent for all other internal elements. Then, we are creating separate "directories" for equipment, material, process segments. A similar approach is applied to system unit class libraries and role class libraries (except for the "root" element). Fig. 5 depicts the structure we have chosen for the modeling of production systems, when following the vocabulary and definitions of IEC 62264.

We believe that this structure is as useful for IEC 62264 derived AutomationML models as it is for separately conceived models of production systems or manufacturing processes.

V. Conclusions

We have briefly described our approach for modeling IEC 62264 information in the context of AutomationML: we are deriving AutomationML documents from genuine IEC 62264 data using formal transformation rules. The resulting AutomationML document provides an internal structure that fosters clear modeling by providing separate sub-trees for different concerns, such as equipment, material, or processes.

Further, we are arguing that a dedicated view on AutomationML information that abstracts from syntactic compromises defined in the underlying CAEX XML-file format helps (i) keeping AutomationML data consistent and (ii) applying model-driven tools on AutomationML models.

References

[1] International Electrotechnical Commission. *Engineering data exchange format for use in industrial automation systems engineering – Automation Markup Language – Part 1: Architecture and general requirements*. International Standard. IEC 62714-1:2018. 2018.

[2] International Electrotechnical Commission. *Engineering data exchange format for use in industrial automation systems engineering – Automation markup language – Part 2: Role class libraries*. International Standard. IEC 62714-2:2015. 2015.

[3] Miriam Schleipen and Rainer Drath. "Three-view-concept for modeling process or manufacturing plants with AutomationML". In: *Proceedings of the 14th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2009. ISBN: 978-1-4244-2728-4. DOI: 10.1109/etfa.2009.5347260.

[4] Bernhard Wally. *Provisioning for MES and ERP. Support for IEC 62264 and B2MML*. Application Recommendation. TU Wien and AutomationML e.V., 2018.

[5] International Electrotechnical Commission. *Enterprise–control system integration – Part 2: Objects and attributes for enterprise–control system integration*. International Standard. IEC 62264-2:2013. 2013.

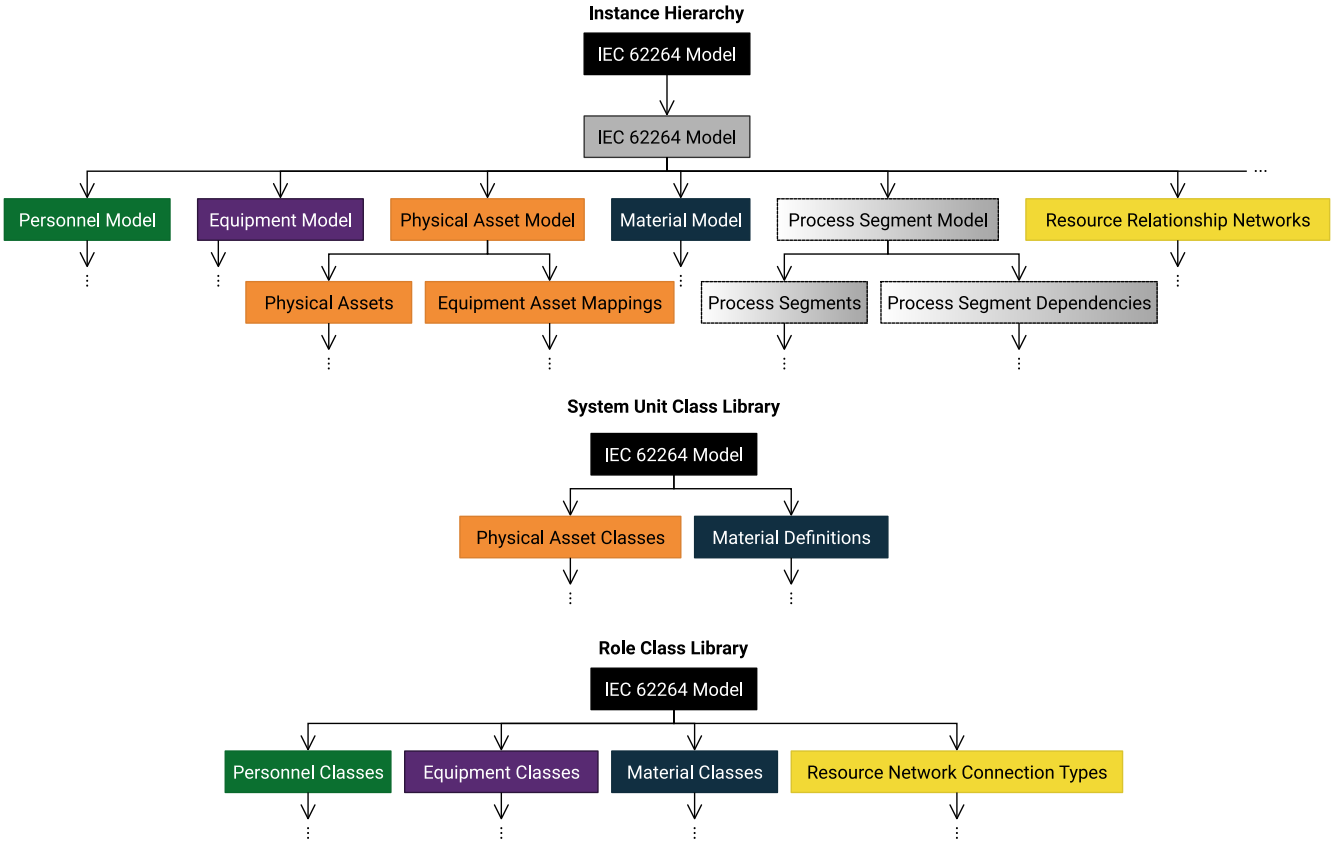

Fig. 5. AutomationML document structure, when applying IEC 62264 vocabulary and relations.

[6] International Electrotechnical Commission. *Enterprise–control system integration – Part 4: Object model attributes for manufacturing operations management integration*. International Standard. IEC 62264-4:2015. 2015.

[7] American National Standards Institute. *Enterprise–Control System Integration Part 1: Models and Terminology*. ANSI Standard. 2000.

[8] Yan Lu, Katherine C. Morris, and Simon P. Frechette. *Current Standards Landscape for Smart Manufacturing Systems*. NIST Interagency/Internal Report NISTIR 8107. 2016. DOI: 10.6028/NIST.IR.8107.

[9] Bernhard Wally, Miriam Schleipen, Nicole Schmidt, Nikolai D'Agostino, Robert Henßen, and Yingbing Hua. "AutomationML auf höheren Automatisierungsebenen. Eine Auswahl relevanter Anwendungsfälle". In: *Proceedings of AUTOMATION 2017. Technology networks Processes*. 18. Leitkongress der Mess- und Automatisierungstechnik. VDI-Berichte 2293. VDI-Verlag, 2017. ISBN: 978-3-18-092293-5.

[10] Richard C. Gronback. *Eclipse Modeling Project: A Domain-specific Language Toolkit*. 1st ed. Addison-Wesley, 2009. ISBN: 0321534077.

[11] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. 2nd ed. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2017. ISBN: 978-1627057080. DOI: 10.2200/S00751ED2V01Y201701SWE004.

[12] Object Management Group. *OMG Meta Object Facility (MOF) Core Specification*. 2016.

[13] Shane Sendall and Wojtek Kozaczynski. "Model Transformation: The Heart and Soul of Model-Driven Software Development". In: *IEEE Software* 20.5 (2003), pp. 42–45. ISSN: 0740-7459. DOI: 10.1109/MS.2003.1231150.

[14] Tom Mens and Pieter Van Gorp. "A Taxonomy of Model Transformation". In: *Electronic Notes in Theoretical Computer Science* 152 (2006). Proceedings of the International Workshop on Graph and Model Transformation (GraMoT), pp. 125–142. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2005.10.021.

[15] Zhenjiang Hu, Andy Schurr, Perdita Stevens, and James F. Terwilliger. "Dagstuhl Seminar on Bidirectional Transformations (BX)". In: *SIGMOD Rec.* 40.1 (2011), pp. 35–39. DOI: 10.1145/2007206.2007217.

[16] International Electrotechnical Commission. *Representation of process control engineering – Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools*. International Standard. IEC 62424:2016. 2016.