Towards Semantic Integration of Plant Behavior Models with AutomationML's Intermediate Modeling Layer

Tanja Mayerhofer[1], Manuel Wimmer[1,2], Luca Berardinelli[2], Emanuel Maetzler[2], and Nicole Schmidt[3]

[1] Business Informatics Group
TU Wien, Vienna, Austria
mayerhofer@big.tuwien.ac.at

[2] Institute of Software Technology & Interactive Systems, CDL Flex
TU Wien, Vienna, Austria
{firstname.lastname}@tuwien.ac.at

[3] Faculty of Mechanical Engineering
Otto-v.-Guericke University Magdeburg, Germany
{firstname.lastname}@ovgu.de

Abstract. AutomationML is an emerging IEC standard for storing and exchanging engineering data among the heterogeneous software tools involved in the engineering of production systems. One important subset of such engineering data is the plant behavior. To make this data exchangeable, AutomationML uses the existing industry data format PLCopen XML. However, at the development stages of production systems, the plant behavior is usually defined using other representation means, such as Gantt charts, impulse diagrams, and sequential function charts. To make such plant behavior models exchangeable, AutomationML introduces the so-called *Intermediate Modeling Layer (IML)* with corresponding transformation rules to decouple the employed modeling languages from the target format PLCopen XML. However, IML itself as well as the transformations from and to IML are only semi-formally described. This not only hinders the adoption of IML as a common language for representing plant behavior, but also renders impossible the composition of heterogeneous plant behavior models for carrying out integrated analyses of the global plant behavior. In this work, we aim at clarifying syntactical and semantical aspects of IML by proposing a metamodel and operational semantics for IML. This constitutes the first step towards formalizing and validating transformations between behavioral modeling languages currently employed in the production domain (e.g., Gantt charts), IML, and PLCopen XML. Having this foundation, we aim at utilizing IML as the semantic domain for the composition of heterogeneous plant behavior models.

Keywords: AutomationML, Production Systems Engineering, Executable Modeling Languages, Operational Semantics, Model Transformations

1 Introduction

AutomationML is a neutral, open, and XML-based data format intended to enable the exchange of data within the heterogeneous software tool landscape of production systems engineering [9]. The engineering of production systems is on the one hand a collaborative work of many different engineering disciplines, which usually use software tools specialized for their specific purpose (e.g., plant layout planning, 3D construction). On the other hand it has different development stages: In the beginning of the engineering process, only a rough overall description of the production system exists, which gets more and more detailed until the production system is fully specified and ready to be built-up and used. It seems reasonable to expect that these software tools are able to exchange data. However, until now much data is exchanged as PDF or printout because of the proprietary and incompatible interfaces of the employed engineering tools. This forces engineers to transfer data between their tools manually, which is time consuming and error-prone. The aim of AutomationML is to avoid or at least reduce this redundant work [2]. To achieve this goal, AutomationML combines three existing industry data formats for the storage and exchange of engineering data: *(i) CAEX* as the main format for representing structural information about production systems, *(ii) COLLADA* for representing geometry and kinematics of production systems, and *(iii) PLCopen XML* for representing behavioral information about production systems.

In this paper we focus on PLCopen XML, which is utilized by AutomationML as an exchange format for plant behavior models [10]. PLCopen XML itself intends to enable an XML-based exchange of PLC code [17] used for programming industrial electromechanical processes. However, at earlier development stages, usually other behavior descriptions with different intentions are used. In particular, AutomationML provides a selection of five behavioral modeling languages typically employed in automation engineering [2]: Gantt charts, PERT charts, impulse diagrams, state charts, and sequential function charts. To make plant behavior models defined with those languages expressible in PLCopen XML, AutomationML introduces the intermediate format *Intermediate Modeling Layer (IML)* [2] as shown at the bottom of Fig. 1. This way, the target format PLCopen XML is decoupled from the various formats used for defining plant behavior models. In particular, complex transformation rules from and to PLCopen XML have to be defined only once for IML. For each employed plant behavior modeling language only transformation rules to the simpler intermediate layer IML need to be defined. This facilitates the extensibility of AutomationML with new plant behavior modeling languages. Furthermore, IML also enables the automated transformation of plant behavior models from one representation into another. This facilitates the refinement of plant behavior models in the consecutive development stages of production systems that usually use different representations.

However, IML itself as well as the mappings between IML and the various behavioral modeling languages supported by AutomationML, and between IML and PLCopen XML are only semi-formally described. As a consequence, the mappings from and to IML are largely unvalidated. In particular, it is unclear whether the mappings preserve the semantics of plant behavior models. This hinders the adoption of IML as a common language for representing plant behavior and impedes the composition of heterogeneous plant behavior models as needed for analyzing the global plant behavior.

In this paper, we aim at clarifying syntactical and semantical aspects of IML by proposing a metamodel and operational semantics for IML. This constitutes the first step towards formalizing and validating transformations from and to IML. Furthermore,

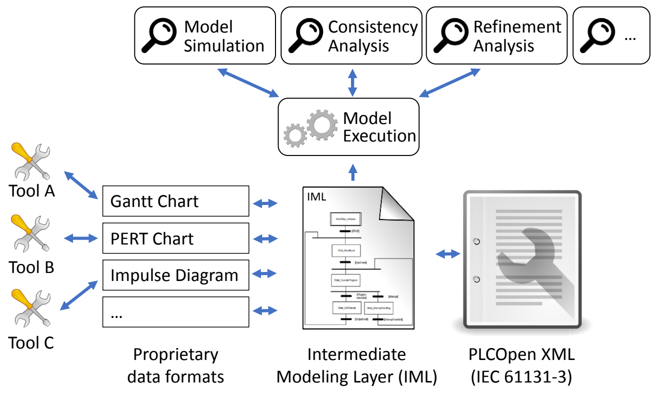

Fig. 1: Plant behavior modeling using IML: An overview.

we advocate the utilization of IML as a semantic integration layer for plant behavior models as depicted at the top of Fig. 1. In particular, we propose to use IML as the semantic domain for the plant behavior modeling languages supported by AutomationML. In a first step, this enables the execution of plant behavior models and the performance of dynamic validation and verification (V&V) activities for such models, such as model simulations and animation, trace analyses, etc. Going one step further, utilizing IML as the semantic domain for plant behavior models enables the composition of heterogeneous plant behavior models in IML and, consequently, the performance of analyses of the global plant behavior defined in the composed models. Such analyses comprise, for instance, the integrated simulation and animation of plant behavior models, the analyses of consistency among a set of plant behavior models, and the validation of refinement relationships between plant behavior models.

The remainder of this paper is structured as follows: In Section 2, we discuss state-of-the-art approaches in production systems engineering. Thereafter, in Section 3, we detail the contribution of this work, namely a formalization of IML. Finally, Section 4 concludes the paper with an outlook on future work.

2 State-of-the-Art

Modeling languages are essential in planning, designing, realizing, and maintaining production systems especially in the light of *Industrie 4.0* [18]. The interest in adopting modeling languages is increasing in industrial automation, and consequently, a huge amount of discipline-specific languages exists [20]. When it comes to cross-disciplinary work, the integration of different modeling tools is becoming an issue and potential benefits of combining tools in tool chains may be achieved [4]. In this context, standardized modeling languages are available for the exchange and integration of behavioral aspects. IML is used for this purpose [1, 13, 14], but there exist of course alternatives. In the following, we discuss prominent alternative approaches with respect to the contribution of this paper.

The System Modeling Language (SysML) [16] provides sublanguages for capturing behavior in terms of activity diagrams, state machines, sequence diagrams, and parametrics diagrams. Transformations may be applied to translate other languages to SysML to facilitate model exchange as well as to form an integrated system model. SysML is a modeling standard heavily influenced by software modeling languages, but it is also used in the automation domain (cf., e.g., [3,11,12,19]). The commonalities and differences of SysML and AutomationML have been discussed in previous work [5]. While there is definitely an overlap between SysML and AutomationML, there are also several features in AutomationML, which are tailored to the industrial automation domain, such as dedicated support for prototype-based system modeling [6].

Concerning the exchange and integration of continuous behavior models, the Functional Mockup Interface (FMI) [7] enables engineering tools to import/export models through the concept of Functional Mockup Unit (FMU) and to (co-)simulate them. It avoids the existence of transformations between different languages. However, for exchanging and integrating models in the very early engineering phases, FMUs may already be focusing too specifically on (co)-simulation. For instance, for transferring the general formulas needed for describing the system's behavior, additional artifacts are needed [15]. Furthermore, in industrial automation, a mix of continuous and discrete models is needed. This is especially of major importance on the general systems viewpoint. Therefore, the integration of AutomationML and FMUs has been already considered in previous work [15] by providing a dedicated data connector for AutomationML.

For modeling and exchanging discrete-event systems and sequential control processes in the context of industrial automation, GRAFCET [8] was proposed in the past. GRAFCET is a modeling language, which takes inspiration from Petri nets and forms also the basis for sequential function charts (SFCs). As IML is considered a subset of SFCs, GRAFCET also clearly influences IML. This is reflected in the operational semantics of IML proposed in this paper.

3 Formalization of IML

As described above, IML is defined by AutomationML as an intermediate representation format for plant behavior [2]. Thereby, IML does not constitute a new language, but is instead based on the sequential function charts (SFC) language defined by the IEC 61131-3 standard [17]. Therewith, IML essentially allows the definition of plant behaviors as discrete state-based operation sequences. However, IML applies restrictions on SFCs to simplify mappings from and to IML, and also extends SFCs to make plant behavior models defined with different languages expressible with IML.

In this section, we describe our formalization of IML comprising a metamodel defining IML's abstract syntax and an operational semantics defining IML's execution semantics. Thereafter, we discuss an exemplary transformation towards IML.

3.1 IML Metamodel

Figure 2 shows the metamodel that we defined for IML according to the semi-formal definitions given in [2]. The Header class is the container of all elements. The main el-

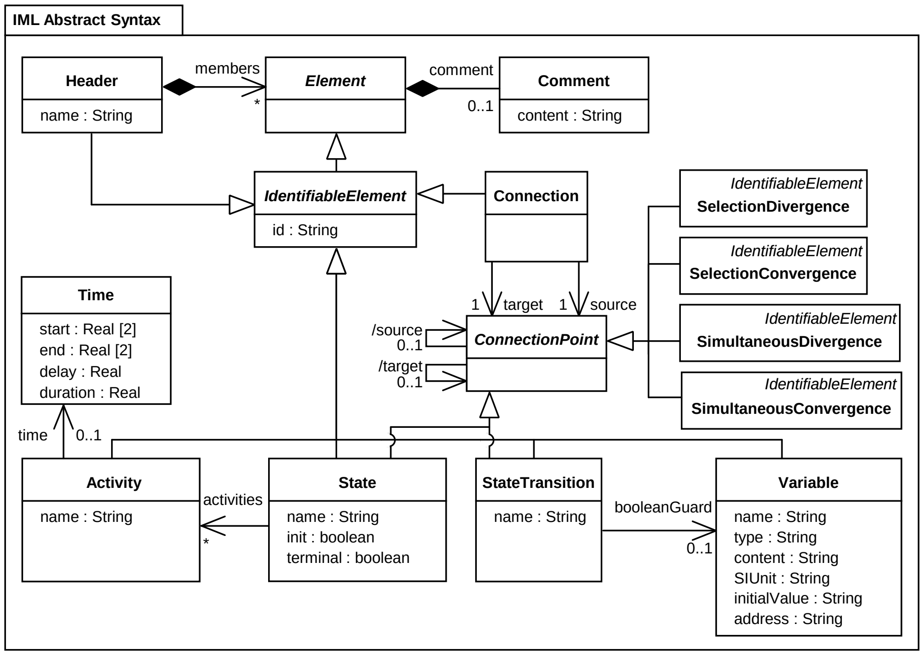

Fig. 2: IML metamodel (excerpt).

ements are State, StateTransition, Activity and Variable. The class State is used to represent the states of a system characterized by, for instance, running operations and values of process variables. Every IML model has to have exactly one initial state and may have a terminal state, which are indicated by the boolean flags init and terminal, respectively. A state may comprise a set of operations represented by the class Activity, which are executed by the system within the state. Thereby, an activity may define its earliest and latest start and end time, time delay, and duration represented by the class Time. Transitions between states are represented by the class StateTransition. They may define variables represented by the class Variable as guards. IML also allows the definition of alternative and parallel state transitions. In particular, the classes SelectionDivergence and SelectionConvergence can be used to represent alternative and exclusive disjoint state transitions, while the classes SimultaneousDivergence and SimultaneousConvergence can be used to represent simultaneous state transitions.

Example. Figure 3 shows an exemplary IML model defining the behavior of a gripping robot. After the initial state *InitialStep*, the robot transitions through the state transition *ST1* and the simultaneous divergence *SD* into the simultaneous states *S_Initialise-Robot1* and *S_LiftSkid*. These states comprise two activities each, namely the activity *DA_InitialiseRobot1* with a start time of *0*, the activity *TA_InitialiseRobot1* with a duration of *6*, the activity *DA_LiftSkid* with a start time of *7*, and the activity *TA_LiftSkid* with a duration of *9*. Through the state transitions *ST2* and *ST3* the robot reaches the synchronization states *Sys_ExecuteManufacturingRobot1_1* and *Sys_ExecuteManufac-*

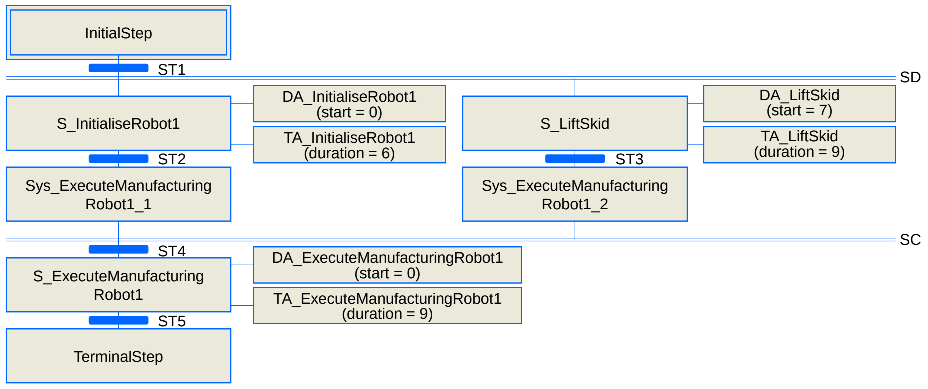
InitialStep
ST1
SD
S_InitialiseRobot1
DA_InitialiseRobot1 (start = 0)
TA_InitialiseRobot1 (duration = 6)
ST2
Sys_ExecuteManufacturing Robot1_1
S_LiftSkid
DA_LiftSkid (start = 7)
TA_LiftSkid (duration = 9)
ST3
Sys_ExecuteManufacturing Robot1_2
SC
ST4
S_ExecuteManufacturing Robot1
DA_ExecuteManufacturingRobot1 (start = 0)
TA_ExecuteManufacturingRobot1 (duration = 9)
ST5
TerminalStep

Fig. 3: Example of an IML model (taken from [2]).

turingRobot1_2, and subsequently transitions into the state *S_ExecuteManufacturing-Robot1* via the simultaneous convergence *SC* and the state transition *ST4*. This state consists of the activity *DA_ExecuteManufacturingRobot1* with a start time of *0* and the activity *TA_ExecuteManufacturingRobot1* with a duration of *9*. Finally, through the state transition *ST5*, the robot reaches the terminal state *TerminalStep*.

3.2 IML Operational Semantics

Figure 4 gives an overview of the operational semantics that we defined for IML. To capture the execution state of an IML model, we introduced the properties current and value into the classes State and Variable, respectively. The property current indicates the current state(s) of an IML model, while the property value captures the current value of a variable. The steps of computation involved in the execution of IML models are defined by means of an endogenous in-place transformation. In Fig. 4, the rules of this transformation are depicted as operations contained by those classes that are matched by the rules. The rule execute of the class Header is the entry point of the transformation and defines the main control loop for executing an IML model. In particular, as shown in Algorithm 1, it first activates the initial state of the IML model by calling the rule activate on the initial state. Thereafter, enabled successor elements of the previously set current state (e.g., a state transition) are fired by calling the rule fire of the class ConnectionPoint. The rule fire of the class StateTransition is shown in Algorithm 2. It first deactivates the current state by calling the rule deactivate. In case the target element is a state, this state is activated. Otherwise, the successor element (e.g., a simultaneous convergence) is fired if it is enabled. The rule activate of the class State (not shown) first sets the state as being the current state and then executes the activities of this state by calling the rule executeActivities. The main control loop of the rule execute continues to fire enabled elements as long as the terminal state is reached.

Example. Figure 5 shows an excerpt of the execution of the example IML model shown in Fig. 3. For starting the execution, the rule execute is called for the single Header element. This rule first activates the initial state *InitialStep*. Then, it fires the state transition

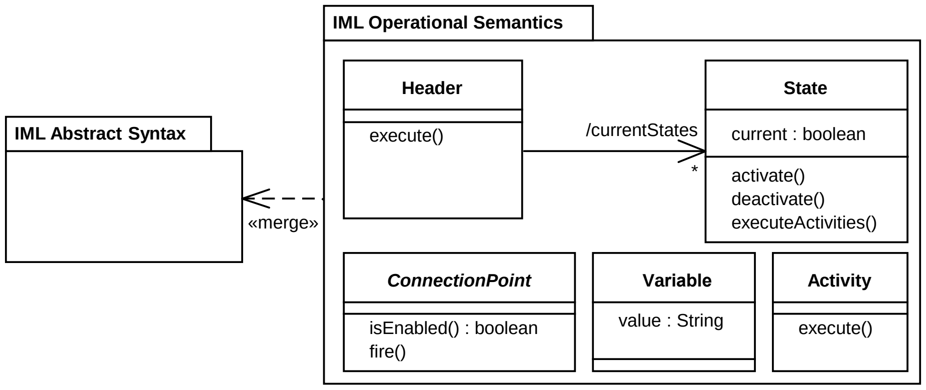

Fig. 4: Overview of IML operational semantics.

Algorithm 1: execute(Header)

```
begin
    $initial \leftarrow \{m \in members \mid m.isTypeOf(State) \wedge m.init\}$
    $initial.activate()$
    $terminate \leftarrow false$
    while $\neg terminate$ do
        foreach $s \in currentStates$ do
            foreach $c \in \{m \in members \mid m.isTypeOf(Connection) \wedge m.source = s\}$ do
                if $c.target.isEnabled()$ then
                    $c.target.fire()$
        if $ts \in \{cs \in currentStates \mid cs.terminal\}$ then
            $terminate \leftarrow true$
```

Algorithm 2: fire(StateTransition)

```
begin
    if $source.isTypeOf(State) \wedge source.current$ then
        $source.deactivate()$
    if target.isTypeOf(State) then
        $target.activate()$
    else if $target.isEnabled()$ then
        $target.fire()$
```

ST1, which in turn fires the simultaneous divergence *SD*. This results in the deactivation of the initial state *InitialStep*, and the activation of the states *S_InitialiseRobot1* and *S_LiftSkid*, which execute their contained activities by calling the rule executeActivities.

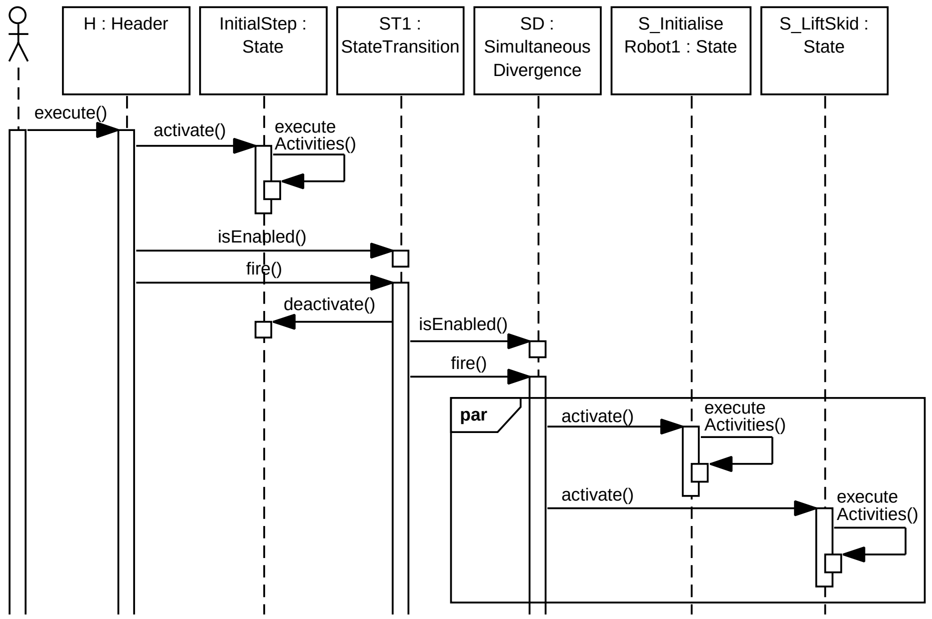

Fig. 5: Example of an IML model execution.

Implementation. We have implemented the operational semantics for IML using the language and modeling workbench GEMOC studio[4]. The complete implementation may be found in our project repository[5].

3.3 Exemplary Transformation Towards IML

The presented formalization enables the execution of plant behavior models defined with IML, as well as the performance of dynamic V&V activities for such models. GEMOC studio provides, for instance, model debugging, model animation, and trace exploration tools that can be utilized for IML models directly based on the defined operational semantics. However, to utilize IML as a semantic domain for other commonly used plant behavior modeling languages, also the transformations from and to IML have to be formalized. We are currently in the process of developing such formalizations. In the following, we discuss as an example the transformation of Gantt charts.

Gantt charts are commonly used in the early phases of plant engineering processes to represent timing aspects of manufacturing processes. They offer two main types of modeling concepts: activities represented as bars, and predecessor-successor relationships between activities represented as connections between bars. For activities, a start time, end time, and duration relative to a global clock may be defined in addition.

Figure 6 shows the Gantt chart corresponding to the IML model depicted in Fig. 3. For each of the bars defined in a Gantt chart, one state is created in the IML model

[4] `http://gemoc.org/studio`
[5] `https://github.com/moliz/moliz.gemoc`

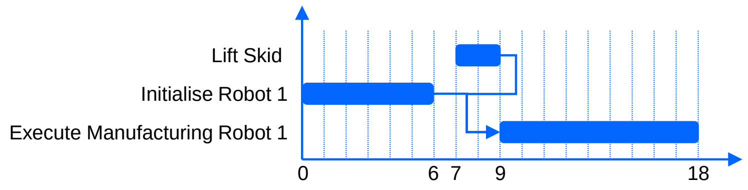

Fig. 6: Example Gantt chart (taken from [2]).

(*S_InitialiseRobot1*, *S_LiftSkid*, *S_ExecuteManufacturingRobot1*). Two activities are added to each created state: The first activity represents the start time of the bar (e.g., *DA_InitialiseRobot1*) and the second activity represents the duration of the bar ensuring the correct deactivation of the state (e.g., *TA_InitialiseRobot1*). The start time and duration of bars have to be converted from a global clock used in Gantt charts to a local clock used in IML. If a Gantt bar has multiple successors, a simultaneous divergence is created in the IML model (*SD*). Similarly, a simultaneous convergence (*SC*) is created for bars that have multiple predecessors. In addition to the simultaneous convergence, synchronization states are created for each predecessor state (*Sys_ExecuteManufacturingRobot1_1* and *Sys_ExecuteManufacturingRobot1_2*). Furthermore, one initial state *InitialStep* and one terminal state *TerminalStep* are created. The created elements are connected with state transitions (*ST1*-*ST5*).

4 Conclusions and Future Work

In this paper, we have introduced a formalization of the Intermediate Modeling Layer (IML) of AutomationML comprising a metamodel and an operational semantics. Furthermore, we have proposed the utilization of IML as a semantic domain for heterogeneous plant behavior models.

Currently, we are working on the refinement of the operational semantics of IML, such as the introduction of timing aspects. Furthermore, we are in the process of formalizing and validating transformations between IML and the various plant behavior modeling languages supported by AutomationML based on the semi-formal descriptions given in [2]. Based on these foundations, our next step will be the investigation of model composition possibilities provided by AutomationML. In particular, we will investigate how heterogeneous plant behavior models can be syntactically integrated by means of the linking mechanisms provided by the CAEX format of AutomationML (cf. Chap. 6 of [2]) and semantically integrated by means of the developed transformations to IML. The ultimate goal of these efforts is to enable the performance of analyses of the global plant behavior defined by a set of heterogeneous plant behavior models.

Acknowledgment

This work has been supported by the Christian Doppler Forschungsgesellschaft, the BMWFW (Austria) and the COST Action MPM4CPS (IC1404).

References

1. Alvarez, M.L., Sarachaga, I., Burgos, A., Estévez, E., Marcos, M.: A Methodological Approach to Model-Driven Design and Development of Automation Systems. IEEE Transactions on Automation Science and Engineering PP(99), 1–13 (2016)
2. AutomationML Consortium: Whitepaper AutomationML Part 4 (2010)
3. Barbieri, G., Kernschmidt, K., Fantuzzi, C., Vogel-Heuser, B.: A SysML based design pattern for the high-level development of mechatronic systems to enhance re-usability. In: IFAC World Congress (2014)
4. Barth, M., Drath, R., Fay, A., Zimmer, F., Eckert, K.: Evaluation of the openness of automation tools for interoperability in engineering tool chains. In: ETFA (2012)
5. Berardinelli, L., Biffl, S., Lüder, A., Mätzler, E., Mayerhofer, T., Wimmer, M., Wolny, S.: Cross-disciplinary engineering with AutomationML and SysML. Automatisierungstechnik 64(4), 253–269 (2016)
6. Berardinelli, L., Biffl, S., Mätzler, E., Mayerhofer, T., Wimmer, M.: Model-based co-evolution of production systems and their libraries with AutomationML. In: ETFA (2015)
7. Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Elmqvist, H., Junghanns, A., Mauß, J., Monteiro, M., Neidhold, T., Neumerkel, D., et al.: The functional mockup interface for tool independent exchange of simulation models. In: Modelica Conference (2011)
8. David, R.: Grafcet: A powerful tool for specification of logic controllers. IEEE Transactions on Control Systems Technology 3(3), 253–268 (1995)
9. Drath, R. (ed.): Datenaustausch in der Anlagenplanung mit AutomationML. Springer (2010)
10. International Electrotechnical Commission: IEC 62714-1 - Engineering data exchange format for use in industrial automation systems engineering - Automation markup language - Part 1: Architecture and general requirements (2014)
11. Kernschmidt, K., Barbieri, G., Fantuzzi, C., Vogel-Heuser, B.: Possibilities and Challenges of an Integrated Development Using a Combined SysML-Model and Corresponding Domain Specific Models. In: MIM (2013)
12. Kerzhner, A.A., Paredis, C.J.J.: Combining SysML and Model Transformations to Support Systems Engineering Analysis. ECEASST 42 (2011)
13. Lüder, A., Estévez, E., Hundt, L., Marcos, M.: Automatic transformation of logic models within engineering of embedded mechatronical units. The International Journal of Advanced Manufacturing Technology 54(9-12), 1077–1089 (2011)
14. Lüder, A., Schmidt, N., Rosendahl, R.: Data exchange toward PLC programming and virtual commissioning: Is AutomationML an appropriate data exchange format? In: INDIN (2015)
15. Lüder, A., Schmidt, N., Yemenicioglu, E.: Herstellerunabhängiger Austausch von Verhaltensmodellen mittels AutomationML. In: Automation (2016)
16. OMG: Systems Modeling Language (SysML) (2015), `www.omg.org/spec/SysML`
17. PLCopen Association: PLCopen XML (2012), `http://www.plcopen.org`
18. VDI: Statusreport Referenzarchitekturmodell Industrie 4.0 (RAMI4.0). `www.vdi.de/industrie40` (April 2014)
19. Vogel-Heuser, B., Schütz, D., Frank, T., Legat, C.: Model-driven engineering of Manufacturing Automation Software Projects - A SysML-based approach. Mechatronics 24(7), 883–897 (2014)
20. Vyatkin, V.: Software engineering in industrial automation: State-of-the-art review. IEEE Transactions on Industrial Informatics 9(3), 1234–1249 (2013)