A. Garmendia, M. Wimmer, A. Mazak-Huemer, E. Guerra and J. de Lara, Modelling Production System Families with AutomationML, *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Vienna, Austria, 2020, pp. 1057-1060

DOI: https://doi.org/10.1109/ETFA46521.2020.9211894

Modelling Production System Families with AutomationML

Antonio Garmendia, Manuel Wimmer
CDL-MINT, Johannes Kepler University
Linz, Austria
firstname.lastname@jku.at

Alexandra Mazak-Huemer
Montanuniversität Leoben
Leoben, Austria
alexandra.mazak-huemer@unileoben.ac.at

Esther Guerra, Juan de Lara
Universidad Autónoma de Madrid
Madrid, Spain
firstname.lastname@uam.es

***Abstract*—The description of families of production systems usually relies on the use of variability modelling. This aspect of modelling is gaining increasing interest with the emergence of Industry 4.0 to facilitate the product development as new requirements appear. As a consequence, there are several emerging modelling techniques able to apply variability in different domains. In this paper, we introduce an approach to establish product system families in AutomationML. Our approach is based on the definition of feature models describing the variability space, and on the assignment of presence conditions to AutomationML model elements. These conditions (de-)select the model elements depending on the chosen configuration. This way, it is possible to model a large set of model variants in a compact way using one single model. To realize our approach, we started from an existing EMF-based AutomationML workbench providing graphical modelling support. From these artifacts, we synthesized an extended graphical modelling editor with variability support, integrated with FeatureIDE. Furthermore, we validated our approach by creating and managing a production system family encompassing six scenarios of the Pick and Place Unit Industry 4.0 demonstrator.**

***Index Terms*—Model-Driven Engineering, AutomationML, Product Lines, Feature Modelling, Variability**

I. Introduction

Modelling flexible production systems is challenging, but at the same time, it is a must with the emergence of the Industry 4.0 [1]. This breed of systems often require capabilities for expressing variability concerns. In such a context, the notion of product lines to describe and manage families of systems have been explored in different engineering disciplines in general and in software engineering in particular [2].

For instance, the Model-Driven Engineering (MDE) paradigm has been applied in combination with Software Product Lines (SPLs). This combination has proven effective and efficient in several domains [1], [3] and contributed to different techniques that can be applied to provide variability [4]. However, regarding current Industry 4.0 standards in the production systems engineering domain, they often lack explicit support for variability. For instance, in the AutomationML standard [5] there is only limited support to express explicitly different variants of a system. As a consequence, engineers frequently duplicate models as a workaround to represent each variant of a system which is often referred to as *clone-and-own* approaches. However, this alternative solution has potential unwanted side-effects such as duplication of data to mention the most crucial one. Due to this data duplication, the propagation of changes is a very tedious and repetitive task which may easily lead to outdated model variants. Furthermore, the management of a family of production systems and its explicit variation points is missing in such settings.

To tackle this shortcoming, we present a compact way to define families of production systems within AutomationML. Our approach is based on the definition of the variability space as a feature model. Then, we define a so-called production system family model, which is a superposition of all model variants within a single model. The elements of this model are tagged with Presence Conditions (PCs) [6], which express the constraints for a particular element to be present in a certain model variant. When the engineer selects a configuration of the feature model, a concrete production system model is produced from the family model by keeping the elements whose PC evaluates to true, and deleting the elements whose PC evaluates to false. This approach allows for a systematic definition of production system families without having to duplicate a single model element, and permits an automatic and controlled generation of production system variants.

For demonstrating the feasibility, we provide a prototypical implementation of our approach on the existing EMF-based AutomationML workbench integrated with FeatureIDE. We added the variability modelling concept within the current modelling editor, using the Eclipse plug-in VERSO. Hence, the generated AutomationML models are compatible and may be exchanged with any tool that supports this standard.

Paper organization. In Section II, we discuss some background on AutomationML and variability modelling in the production system domain. In Section III, we introduce our variability modelling approach and its integration within the AutomationML environment. Section IV shows the applicability of our approach using the Pick and Place Unit (PPU) demonstrator from TU Munich [7] as a proof of concept. Finally, Section V concludes with an outlook on future work.

II. Background

This section introduces the foundations of this work, i.e., AutomationML, as our base modelling language. Then, we discuss approaches for variability modelling–especially for production systems–and argue why a novel approach for variability would be beneficial in the context of AutomationML.

978-1-7281-8956-7/20/$31.00 ©2020 IEEE

A. AutomationML

AutomationML [5] is a neutral XML-based data format for representing engineering knowledge in the area of process automation and control. It is widely accepted in the area of Industry 4.0 by its development within an academic and industrial consortium. For more information on the standardization efforts and current developments, we refer the interested reader to the AutomationML website[1].

AutomationML is an integration format for the following standards: CAEX for system topology, COLLADA for geometry and kinematic, and PLCopen XML for logic. By using CAEX, the entire system model is represented as an *instance hierarchy* in AutomationML. Devices of the system are represented as instances, which are called *internal elements* of the aforementioned *instance hierarchy*. The devices' types are represented as *system unit class*. Interconnections between *interfaces* of devices are represented as *internal links*. For expressing the meaning of captured information, *role class libraries* are used, which should be shared among various projects, and thus, are subject to many standardization efforts in AutomationML. The PLCopen and COLLADA parts of AutomationML are not further used in this paper, thus, we do not detail these parts.

In previous work, we developed dedicated tooling for AutomationML [8] which is also the basis for this work. In particular, we created a full Eclipse Modelling Framework (EMF) based implementation of AutomationML to allow the application of model-driven engineering concepts, techniques, and technologies for AutomationML out of the box. For instance, this allowed us to generate modelling editors, APIs, and validation support for AutomationML. In this paper, we show how AutomationML can be extended with modelling support in order to produce family of systems.

B. Variability Modelling

Some researchers have proposed the use of variability modelling to capture the commonalities and differences of automated production systems [1], [9]. The work presented in [10], [11] is motivated with the PPU case, but proposing different solutions for variability. The former proposes the Model-Driven Evolution Management Framework for Automation Systems (MoDEMAS), which is based on interface behaviour modelling of design artefacts, supporting model evolution and the verification of its correctness. The latter [11] proposes a combination of a set of diagrams from the Unified Modelling Language (UML). Specifically, the activity, component and state charts diagrams are used to capture the variability and evolution of manufacturing systems.

Another approach to represent a family of models is the use of feature models. For instance, Papakonstantinou et al. [12] describe an approach for product configuration and code generation with the use of feature modelling. That paper describes how to validate a configuration to prevent invalid selections of concrete features. To do this, the authors define a mapping between feature models and the UML class diagram, but it requires manual adjustments and no dedicated language for variability is proposed.

Variability modelling can be managed by cardinality techniques [4]. Approaches for modelling cardinalities have been proposed for AutomationML [13]–[15]. The integration of eCl@ss and AutomationML [13] supports cardinalities for blocks, which contain sub-elements. The cardinalities can be defined in any block allowing the dynamic multiplication of its sub-elements within a scope. Another approach can be found by using the new features of CAEX 3.0 attribute type library [14]. Based on this concept, AutomationML now provides a compound attribute type for cardinalities consisting of two nested attributes: the minimum and the maximum occurrence. Before CAEX 3.0, Wimmer et al. [15] introduced a role class library to explicitly represent variability for AutomationML, in particular, for AutomationML System Unit Class (SUC) libraries. While this helps to more systematically apply library elements for a particular model, managing a family of models is not possible with cardinality-based annotations on SUCs as we will discuss later on in this paper. Therefore, dedicated language support is further needed to manage families of models.

Overall, the aforementioned work provides variability for the production system domain from different perspectives. However, to the best of our knowledge, the use of PCs to describe model families for AutomationML in combination with feature models, as we propose in this paper, is a novel contribution that has not been explored before.

III. Presence Conditions in AutomationML Models

We now explain how to apply PCs to AutomationML models. First, we explain the concepts of feature modelling and PC in general (Sec. III-A), second, the customization of variation points for AutomationML (Sec. III-B), and finally, variability modeling support in the graphical editor (Sec. III-C).

A. Feature Modelling and Presence Conditions

The approach of this paper is based on the use of feature diagrams [16]. An example is shown in Fig. 1 (a), where a feature diagram is designed for washing machine controllers inspired in this paper [17]. Each node of this tree structure is a *feature* and the family of models would be created depending on how the features are combined. From the feature model we can generate a feature configuration. For instance, a valid configuration must always have the feature Equipped selected and at most the feature Heat or Delay as well.

Following the annotative approach to define a model family [6], we shown an example in Fig. 1 (b). All the model elements that may appear in models of the family, should have a representation in this model. In addition, these model elements may have attached PCs, which are logic formulae over the features defined in the feature diagram. These PCs are depicted in this figure using dashed red arrows. Fig. 1 (c) shows a valid language based on a valid configuration given the model family. In this configuration, the features Dry

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

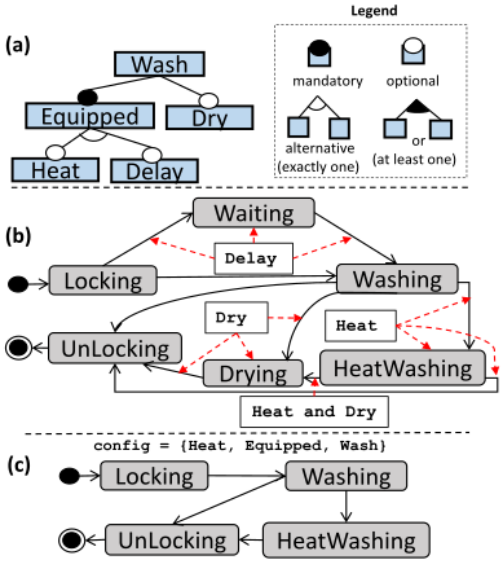

Fig. 1. (a) Feature model with variability for a washing machine controller. (b) Definition of variability annotations for the washing machine controller. (c) Generated derived model from a configuration.

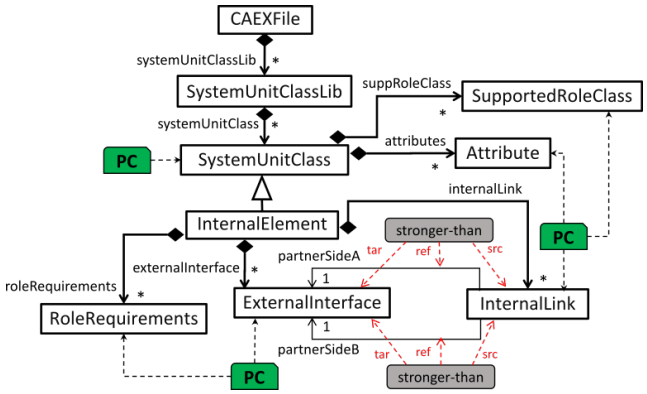

Fig. 2. Specifying the variability for CAEX.

and Delay are not selected, and this is why the target elements they were targeting have been removed.

B. Inserting Variation Points at the Language Level

Fig. 2 shows an excerpt of the CAEX meta-model with the definition of its variability. To specify the variability for the CAEX language, we essentially consider two requirements in order to obtain valid instance models. The first aspect to take into account is which elements may be subject to variability. In this case, we allowed that six classes were PC-enabled.

The other aspect we take into consideration is the well-formedness rules among the PCs. For instance, the PCs of the referenced objects by the partnerSideA reference, should be stronger than the InternalLink PCs. By ensuring this, any generated instance models will not contain InternalLinks objects with partnerSideA references which point to missing ExternalInterface objects. In order to specify this rule, we use the predicate stronger-than. Consequently, we define also this predicate in the partnerSideB reference, as we can see in Fig. 2.

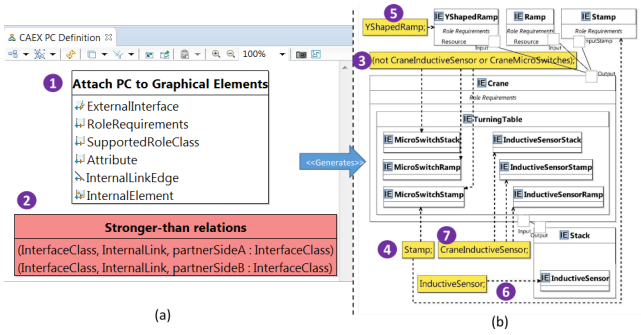

Fig. 3. (a) AutomationML editor enabled for defining presence conditions. (b) Diagram with PC to generate the six scenarios of PPU.

C. Injecting Variability in the Graphical Editor

We now explain how to enable the inclusion of PCs attached to AutomationML models. Our approach is based on annotating the classes in the meta-model that can support variability, and on specifying constraints among the PCs that the elements of a model family may have.

Technically, our proposal relies on the AutomationML EMF-based workbench and its graphical Sirius-based editor shared on Github[2]. To allow defining PCs in AutomationML models, we have extended the graphical editor by using the Eclipse plug-in VERSO[3]. This plug-in automatically adapts existing Sirius graphical editors to enable the attachment of PCs to the desired model elements. In addition, VERSO is integrated with FeatureIDE [18], which is a tool that covers the entire product-line development process. In particular, it supports the creation of feature models and the generation of its corresponding configurations.

Fig. 3 (a) shows the customization to inject the variability in the AutomationML graphical editor using VERSO. This plug-in took as an input the initial Sirius model defined for CAEX (label 1) and then, using the VERSO editor, configure which elements can be attached with PCs and also the definition of stronger-than predicates (label 2). From this customization, VERSO is able to extend the existing CAEX graphical editor with variability functionalities.

In our case, the obtained extended editor allows defining AutomationML model families which unify several variants in a single specification. This way, we can define feature models declaring the variability points of a model family, and by selecting a subset of the features in the feature model, we obtain a particular model variant automatically.

IV. THE PPU DEMONSTRATION CASE

This section demonstrates how to apply our approach to a concrete case. Vogel-Heuser et al. [7] present 15 scenarios of a lab demonstrator system, namely the Pick and Place Unit (PPU). These scenarios are different variants of the same core system. Thus, we can move from one to another with a set of changes. To demonstrate our approach, we took the first six

[2]https://github.com/amlModeling/caex-workbench
[3]https://github.com/antoniogarmendia/ecore-product-line

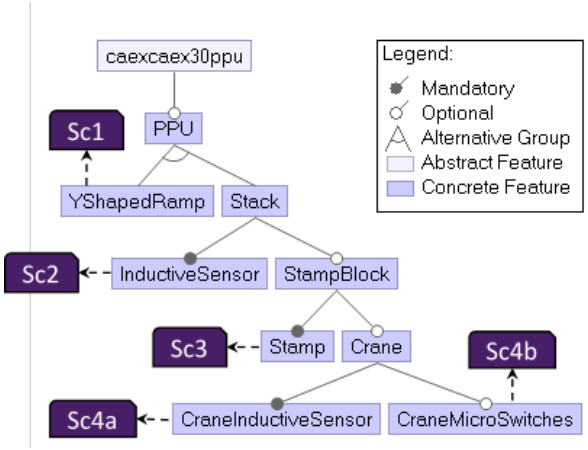

Fig. 4. Feature model for six scenarios of PPU

documented scenarios (Sc0, Sc1, Sc2, Sc3, Sc4a and Sc4b) of the PPU and remodelled them with our approach as one single model representing the whole family of variants. We share our implementation in this Github repo [4].

Fig. 4 shows the feature model to represent and generate the six configurations. If no features are selected, the configuration represents the Scenario 0 (Sc0). The root feature PPU has a set of alternative features, and so exactly one subfeature must be selected. To convert Sc0 into Sc1, the feature YShapedRamp must be selected. The generation of the Sc2 is with the selection of the Stack feature, which leads to the selection of the mandatory feature InductiveSensor. The Scenario 3 (Sc3) is generated when the StampBlock is selected, which therefore select the feature Stamp. The Sc4a is generated when selecting CraneInductiveSensor, and the last scenario (Sc4b) depends on the selection of the feature CraneMicroSwitches.

After the creation of the feature model described above, we design the AutomationML model for the family of systems with the attached PCs. Fig. 3 (b) shows the family which includes the 6 aforementioned configurations of the PPU case study. If all the values of the feature model are false, then all the model elements that have attached a PC will be removed except the MicroSwitches (label 3). These three microswitches are presented in all scenarios, except when the feature CraneInductiveSensor or feature CraneMicroSwitches are set to true. However, MicroSwitcheStamp (label 4) also depends on if the Stamp (label 4) feature flip to true. The model element YShapedRamp will be available only in Sc1 (label 5). The InductiveSensor (label 6) will be presented in the Sc2, Sc3, Sc4a and Sc4b. Lastly, the CraneInductiveSensors (label 7) will be available in the scenarios 4a and 4b.

By using the feature model, we are able to generate all six variants as expected but not a single variant going beyond the presented ones. This allows a controlled management of the variability which would not be possible with before presented variability mechanisms for AutomationML based on cardinalities. For instance, if the feature CraneInductiveSensor is not mandatory, then it is generated an invalid scenario with no CraneInductiveSensor and no CraneMicroSwitches.

[4]https://github.com/amlModeling/caex-workbench/tree/variability

V. Conclusion

We have presented a proposal for creating families of AutomationML models in a concise way by using PCs. A specific model variant can be extracted by selecting a valid configuration of the feature model and evaluating the PCs in the model family.

In the future we plan to further explore the usage of the PCs for a set of models coming from different engineering domains such as mechanical, electrical, and software engineering. We also plan to devise methods to automatically create a model families out of an existing set of models.

References

[1] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy, "Evolution of software in automated production systems: Challenges and research directions," *Journal of Systems and Software*, vol. 110, pp. 54–84, 2015.

[2] S. Trujillo, D. S. Batory, and O. Díaz, "Feature oriented model driven development: A case study for portlets," in *ICSE*. IEEE, 2007.

[3] S. Trujillo, J. M. Garate, R. E. Lopez-Herrejon, X. Mendialdua, A. Rosado, A. Egyed, C. W. Krueger, and J. D. Sosa, "Coping with variability in model-based systems engineering: An experience in green energy," in *ECMFA*. Springer, 2010.

[4] M. Sinnema and S. Deelstra, "Classifying variability modeling techniques," *Inf. Softw. Technol.*, vol. 49, no. 7, pp. 717–739, 2007.

[5] R. Drath, A. Lüder, J. Peschke, and L. Hundt, "AutomationML - The glue for seamless automation engineering," in *ETFA*. IEEE, 2008.

[6] C. Kästner, S. Apel, and M. Kuhlemann, "Granularity in software product lines," in *ICSE*. ACM, 2008.

[7] B. Vogel-Heuser, C. Legat, J. Folmer, and S. Feldmann, "Researching evolution in industrial plant automation: Scenarios and documentation of the pick and place unit," Institute of Automation and Information Systems, Technische Universität München, Tech. Rep., 2014.

[8] T. Mayerhofer, M. Wimmer, L. Berardinelli, and R. Drath, "A model-driven engineering workbench for CAEX supporting language customization and evolution," *IEEE Trans. Ind. Informatics*, vol. 14, no. 6, pp. 2770–2779, 2018.

[9] J. Coplien, D. Hoffman, and D. Weiss, "Commonality and variability in software engineering," *IEEE software*, vol. 15, no. 6, pp. 37–45, 1998.

[10] C. Legat, J. Mund, A. Campetelli, G. Hackenberg, J. Folmer, D. Schütz, M. Broy, and B. Vogel-Heuser, "Interface behavior modeling for automatic verification of industrial automation systems' functional conformance," *Automatisierungstechnik*, vol. 62, no. 11, pp. 815–825, 2014.

[11] M. Kowal, C. Legat, D. Lorefice, C. Prehofer, I. Schaefer, and B. Vogel-Heuser, "Delta modeling for variant-rich and evolving manufacturing systems," in *MoSEMInA*. ACM, 2014.

[12] N. Papakonstantinou, S. Sierla, and K. Koskinen, "Generating and validating product instances in iec 61131–3 from feature models," in *ETFA*, 2011.

[13] "AutomationML and eCl@ss Integration," 2015. [Online]. Available: https://www.automationml.org/o.red/uploads/dateien/1448438009-20151030_WP_AutomationML_and_eClass_integration_v1.0_neu.pdf

[14] "AutomationML," 2018. [Online]. Available: https://www.automationml.org/o.red/uploads/dateien/1544706233-automationml.pdf

[15] M. Wimmer, P. Novák, R. Sindelár, L. Berardinelli, T. Mayerhofer, and A. Mazak, "Cardinality-based variability modeling with automationml," in *ETFA*. IEEE, 2017.

[16] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Software Engineering Institute, Carnegie Mellon University, Tech. Rep., 1990.

[17] R. Salay, M. Famelis, J. Rubin, A. Di Sandro, and M. Chechik, "Lifting model transformations to product lines," in *ICSE*. ACM, 2014.

[18] J. Meinicke, T. Thüm, R. Schröter, F. Benduhn, T. Leich, and G. Saake, *Mastering software variability with FeatureIDE*. Springer, 2017.