Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 81 (2019) 13–18

www.elsevier.com/locate/procedia

52nd CIRP Conference on Manufacturing Systems

Modeling Variability and Persisting Configurations in OPC UA

Bernhard Wally[a,*], Christian Huemer[b], Alexandra Mazak[a], Manuel Wimmer[a], Radek Šindelář[a]

[a]*CDL Model-Integrated Smart Production, Institute of Information Systems Engineering, TU Wien, Favoritenstraße 9-11/E194, 1040 Vienna, Austria*
[b]*Business Informatics Group, Institute of Information Systems Engineering, TU Wien, Favoritenstraße 9-11/E194-03, 1040 Vienna, Austria*

* Corresponding author. Tel.: +43-1-58801-188695; fax: +43-1-58801-9188695. *E-mail address:* wally@big.tuwien.ac.at

Abstract

Variability is crucial in the design of many advanced goods and it is also receiving increasing attention in production systems engineering. Since OPC Unified Architecture plays an important role when it comes to standardized information exchange in modern production systems, it can be a melting pot for information from various engineering domains, such as product design and production engineering—thus, it is an ideal place to hold variability information of products and production systems alike. Based on an initial variability information model we propose additional concepts for the persisting of configurations.

Keywords: feature model; variability; configuration; OPC Unified Architecture; information model

1. Introduction

Managing product and production variability is among the most discussed topics in smart production environments [1], [2]. One reason is that the products that are manufactured become more and more variable in order to satisfy customers' needs (cf. “lot size 1”). Product variability through the definition of families of products is a way of keeping the amount of source and intermediate material low while increasing the amount of different end products. Even the variability of automated production systems (aPS) themselves can be formally modeled in order to track different variants as variations of a common theme [3]. Through that it becomes possible to select a specific aPS setup for a certain production task. A classification of variability issues in production systems is given in [4].

In order to make use of properly crafted variability models during production, it might proof useful to enable variability modeling in the communication system of aPS. For that account, we have chosen to provide a variability modeling information model for OPC Unified Architecture (UA) that allows universal specification and interpretation of products and production systems on various levels: (i) smart production cells can interpret a specific configuration in a way to determine concrete machining steps, (ii) manufacturing execution systems need to know about a specific aPS setup in order to schedule and track manufacturing operations.

Variability information is a common occurrence in many processes and can be viewed from different angles (e.g., for defining constraints in design documents, or for user-based feature selection in configuration wizards)—one popular technique of describing, computing and understanding product variability are *feature models* (FM). They are extensively used in modeling and managing variability in software product lines [5]. They go back to a report by Kang et al. [6] in which systems are analyzed through the identification of prominent or distinctive *features*. Feature models can be visualized by feature diagrams, which show features in a tree layout, where nodes represent features and edges the relation of these features (cf. Fig. 1). Relations that cannot be directly modeled within the tree hierarchy, are explicitly formulated as *constraints* and usually shown below or besides a feature diagram, or they are visualized as cross-tree relations between corresponding feature nodes. Feature models allow the calculation of possible variants; a specific set of selected features that cor-

2212-8271
10.1016/j.procir.2019.03.003

responds to such a variant is called *configuration*, which is reached by *configuring* a feature model.

Fig. 1 shows a cardinality-based feature diagram for a family of lamp products: each lamp must specify a material (either metal or plastic) and one of three sizes (small, medium, large). Optionally one can select bathroom safety (i.e., withstanding humidity), however this requires to use plastic. Furthermore, the large lamp size cannot be produced in metal.

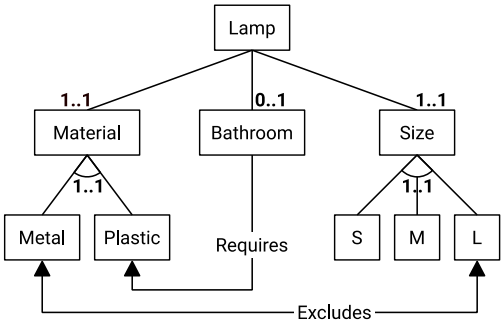

Fig. 1. Feature model for a family of lamp products (for the notation, cf. [7]).

In this work, we describe an OPC UA information model that allows the modeling of this kind of cardinality-based feature models using plain OPC UA technology. In Sec. 3 we describe our information model which is then applied in two test cases in Sec. 4. After a critical discussion in Sec. 0, we conclude the paper with an outlook on future efforts in this line of research (Sec. 6).

2. Related Work

An overview of software engineering methods that make sense in industrial automation is given in [2], explicitly stating the application of software product line (SPL) modeling techniques, which is discussed in more detail in [8]: SPL methods are used in the configuration of machine control applications. This might be an application scenario of our proposed solution—reading concrete configurations from an OPC UA server and adapting a programmable logic controller's execution based thereupon.

Feature models in their basic binary form can be translated to propositional logic and evaluated using out-of-the-box boolean satisfyability problem (SAT) solvers [9], [10]. There exist several variations of feature models, one of the most prominent is that of *cardinality-based feature models* [7], [11]: features are no longer only related to one another using a binary notation (allowed vs. not allowed), but relations (both grouped and single edges) can be expressed with cardinalities in mind (well known from, e.g., Unified Modeling Language (UML) class diagrams). Binary feature models are considered special cases of the more generic cardinality-based feature models. Our proposed information model implements such a cardinality-based feature model.

With Clafer [12], [13], a comprehensive toolkit and web frontend for the engineering of product lines has been created. Using text input, the application domain can be modeled by mixing class declarations with feature modeling in a very flexible way. With the Clafer Multi-Objective Optimizer and a corresponding visualizer, it is possible to find and compare discovered feature configurations of optimal variants within a product line [13]. The modeling techniques in this tool are quite complex and not so easy to implement in OPC UA, however it would be possible to use Clafer as a frontend to model variability and then transform the resulting feature models (potentially simplified) into the OPC UA space.

Advanced modeling approaches for feature models have been presented in [14] that allow reuse of features and feature models, through “subscoping”, and the use of reference models (models that can be derived from and deltas be defined). Most notable is the introduction of *configuration links*, which allow the linking of feature models and configurations by defining additional constraints for feature models that are enabled based on selected features in the given input configurations. That way it is possible to influence the variability of a technical feature model by, e.g., different target markets. An improved version of that technique is presented in [15], where also some theoretical background is given for the application of these configuration links. In our approach we do not explicitly support the modeling of configuration links, as this technique has not yet well matured. However, we believe that it could be an interesting feature for future versions of our information model.

Modeling variability of aPS has been investigated in [3], where a delta-based approach [16] is applied. Delta modeling works by identifying a core variant of a system and specifying variations thereof through a description of deltas, i.e., which components have been added to the core variant, which have been removed from it and which were modified. In [3], this approach is being used for describing different configurations of a pick-and-place unit by supporting add and remove operations. Different to our approach, the variability information is not stored online in the production system, but is defined in a separate information system. The investigation of a delta-based approach in the context of OPC UA could be an interesting topic for future research.

Adding variability modeling capabilities to existing technologies has been previously presented in [17], where an AutomationML[1] (AML) role class library was created in order to enable the specification of feature models. Due to the design of AutomationML it is required to alter vendor-specific component libraries in order to introduce variability information. While these manipulations are non-destructive (simply adding role classes with attributes) they introduce some additional maintenance effort in case the vendor-specific component library is updated. We offer an alternative modeling approach by transforming the AutomationML information into OPC UA and defining variability in this context. A parallel definition of variability, both in AML and OPC UA would be possible and could establish a “safety net” in form of redundantly modeled information. Inconsistencies between these models could be a good hint for varying design time decisions and would need to be resolved.

The description of an “Industry 4.0” component using AutomationML and OPC UA is discussed in [18], stating that

1 cf. https://www.automationml.org/

the administration shell of production system components could be realized through a composition of technologies such as OPC UA and AutomationML. The specification of product and production system variability in addition to the already given information would be another application scenario of the variability information model presented in this work.

In [19] a changing market environment is described, in which customer-requirement-driven individualized products are increasingly demanded. Such an environment fosters the occurrence of configurable standard components and individual parts, which in turn leads to an increase in product variability [19]. This work focuses on the requirements of a specialist for the production of functional furniture fittings, thus a key concept are 3D-CAD models and how they can be varied, combined, assembled, etc. For instance, in this system it is possible to define constraints that define valid ranges of the positioning of handling modules for a physical production process. This kind of information could be of interest in the context of variability modeling in smart production environments running on OPC UA.

In [20] the manufacturing process of CAD (Computer Aided Design) related components such as connecting rods or cylinder heads has been augmented with variability information: standard industry tools have been used and enriched with meta-information in order to follow features throughout the production life-cycle. The given use case and tool integration is focused on geometric data that is relevant for CAD based systems. We have not further investigated the application of our information model to such product design information, however it could be yet another application scenario, by interweaving product information with production process information through feature models.

The transformation of a generic modeling paradigm into OPC UA is discussed in [21], where UML class diagrams are transformed into OPC UA information models. In this work, the authors state that one of the issues with the transformation is the definition of cardinalities on attributes, as the capabilities of OPC UA lag behind those of UML class diagrams. In our proposed solution we model cardinality constraints explicitly as properties of OPC UA objects, and it would be possible to extend the information model in a way that would allow for advanced cardinality constraints such as multiple cardinality ranges, etc. For now, we have not considered such more complex constraints.

Modeling of variability in the context of OPC UA has been presented in [22]; we are building on top of the information model that is presented there in order to allow the persisting of available configurations in an OPC UA server. In [22], an information model for creating feature models in OPC UA is presented—in this work, we present an information model that allows making explicit statements about which nodes form a specific configuration.

This might be important in the context of configurable products that are being produced in flexible manufacturing environments. E.g., such products could be configured following the House of Quality approach [23], using feature models as the technology for providing information about the technical constraints of the product [24].

3. Persisting Configurations in OPC UA

The approach presented in [22], defining the feature model "aside" an existing domain-specific information model, comes with a trade-off: generating a model of the instantiated configurations is a tedious task that involves quite a bit of information model querying; configuration information really needs to be "mined" from the OPC UA server and is not readily available. However, and this is what we propose in this work, one could use an additional information model that enables the tagging of feature instance nodes—so that once the configuration information has been mined it can be persisted in order to allow faster subsequent configuration checking. However, it has to be noted that this configuration information model needs to be continuously synchronized with the corresponding domain model in order to stay up-to-date (it needs to *co-evolve* [23]).

In Fig. 2 we are providing such a *configuration information model* that has been derived from the variability information model presented in [22]. In the text we are using two typographic hints: (i) when we talk about an OPC UA metamodel concept, we are using a mono-spaced font, e.g., for an OPC UA `Object`, (ii) when we talk about an instance of such as concept, as it would be deployed in an OPC UA server, we are using a sans-serif font, e.g., for the OPC UA BaseObjectType.

The information model allows the explicit tagging of configurations by applying the following concepts:

1. FeatureInstances are created as `Objects`; they do not employ dedicated `Properties`.
2. In order to provide meaning to a FeatureInstance, it needs to be related to a corresponding node in the domain model. This is realized by an instance of the IsFeatureInstanceNatureOf `ReferenceType`.
3. Relating a FeatureInstance with its corresponding Feature through is realized by an instance of the IsFeatureInstanceOf `ReferenceType`.
4. Building a FeatureInstance tree is realized by creating `References` of type HasSubFeatureInstance.

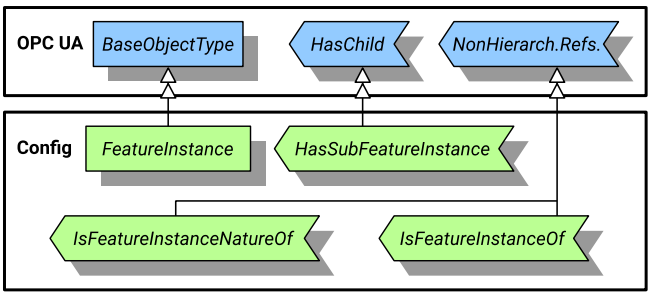

Fig. 2. An OPC UA information model for the tagging of mined configurations.

More detailed analysis and application of this information model is presented in Sec. 4. We believe that this information model enables the persisting of configuration information in a rather intuitive way by following the design approach of the variability information model introduced in [22].

4. Evaluation

We have prepared a test case for the developed information model. It is an abstract (symbolic) instance that involves all the modeling techniques available in [22] and in the approach discussed in this work. The test case is visualized in Fig. 3: the two upper compartments represent the information model defined in this paper ("Config") and the one from [22] ("FM"). The lower compartment comprises three virtual "rows": (i) the top row depicts the symbolic domain model declaration (left) and the feature model that is defined on top of it (right). These two models are interlinked via IsFeatureNatureOf references. (ii) the middle row shows three valid configurations of the feature model as well as (in orange) an explicit configuration instance. (iii) the bottom row shows the remaining four valid configurations. Cardinality constraints of the feature model are annotated next to the feature nodes. In case a cardinality constraint comprises two ranges (e.g., [0..1, 2..4]) the first defines the cardinality of the feature node itself, while the second defines the cardinality of an implicitly defined *feature group* (corresponds to properties groupMin and groupMax).

In our test case, the feature model is defined by a root feature A which has two sub-features: (i) B is an optional feature with cardinality [0..1] that contains two to three sub-features of kind D ([2..3]), and (ii) C is another optional sub-feature with cardinality [0..1]. Different from feature B feature C defines a *feature group* comprising its sub-features E and F. While E might occur zero to three times ([0..3]), F might occur zero to two times ([0..2]). However, since they are bundled in a feature group with cardinality [2..4] the sum of their occurrences must be greater than 2 and less than 4.

Additionally, this feature model defines two cross-tree constraints: (i) "feature E excludes feature B", i.e., these two features must not occur in both in any given configuration, (ii) "feature F requires feature D", i.e., whenever there is a feature F in a configuration, there must also be at least one instance of feature D.

This leads to the following set of seven variants (numbers correspond to the ones given in Fig. 3; note that a cardinality difference of a feature leads to a separate variant):

```
1: 1x "A", 1x "B", 2x "D"
2: 1x "A", 1x "B", 3x "D"
3: 1x "A", 1x "B", 1x "C", 2x "D", 2x "F"
4: 1x "A"
5: 1x "A", 1x "C", 2x "E"
6: 1x "A", 1x "C", 3x "E"
7: 1x "A", 1x "B", 1x "C", 3x "D", 2x "F"
```

The possible variants are also depicted in the lower part of the bottom compartment of Fig. 3 in terms of "instances" of the feature model.

The corresponding domain model of the feature model is defined by a hierarchy of symbolic `ObjectTypes` (OT1 to OT6). These object types span a hierarchy through the application of hierarchical references of type Organizes (we have abstained from tagging these references in order to keep the visualization less cluttered). In the instances, the feature nodes are related to each other using the HasComponent reference type; however, we have chosen the generic "asymmetrical reference" notation (a single closed filled arrow) to make it clear that any kind of reference could be used. Consequently, we are not explicitly showing the SubFeatureReferenceType reference (cf. [22]) from the root node to the HasComponent reference type in order to not irritate and to keep the figure clean.

To avoid cluttering with HasTypeDefinition references, we have opted for an alternative notation by specifying the `ObjectType` of the instance nodes separated from their display names by a double colon (::).

So far, we have discussed concepts that were already defined in [22]. Note, that for the analysis of the validity of a configuration, the configurations need to be mined using the following approach. First, look for `Object` nodes representing a root node of a feature model configuration: check whether the `ObjectType` node (HasTypeDefinition) of the node under observation has an IsFeatureNatureOf `Reference` pointing at it, and whether the source node of this `Reference` has no HasSubFeature `Reference` pointing to it (then it is a feature model root node). If so, the `Object` node under inspection represents a feature model configuration root node and is thus considered part of a configuration. Starting from this node, the following recursive algorithm is used to build the find and tag the complete configuration:

1. Create a new `Object` node of type FeatureInstance and relate it to the current domain node via an IsFeatureInstanceNatureOf `Reference`. In Fig. 3, we are depicting this kind of `Reference` using a dashed arrow with solid head.
2. Relate this FeatureInstance with the corresponding Feature, using an instance of the IsFeatureInstanceOf `ReferenceType` (depicted using a straight orange solid arrow with solid head).
3. Find nodes that are linked to the current node using `References` of the `ReferenceType` identified by the SubFeatureReferenceType `Reference` of the corresponding Feature node. These nodes are instances of subfeatures. For each of them recurse and create a corresponding `Reference` of type HasSubFeatureInstance between the current node and the recursively created node.

In summary, the following items were added to the existing information model in order to order to persist the configuration corresponding to "Valid Instance 1":

1. Four `Objects` (I_A, I_B, I_{D1}, I_{D2}) of type FeatureInstance, that represent selected Features.
2. Three `References` of type HasSubFeatureInstance that create a configuration tree, corresponding to the feature tree of the feature model.
3. Four `References` from the FeatureInstances to the corresponding Features.
4. Four `References` from the FeatureInstances to the nodes of the domain model.

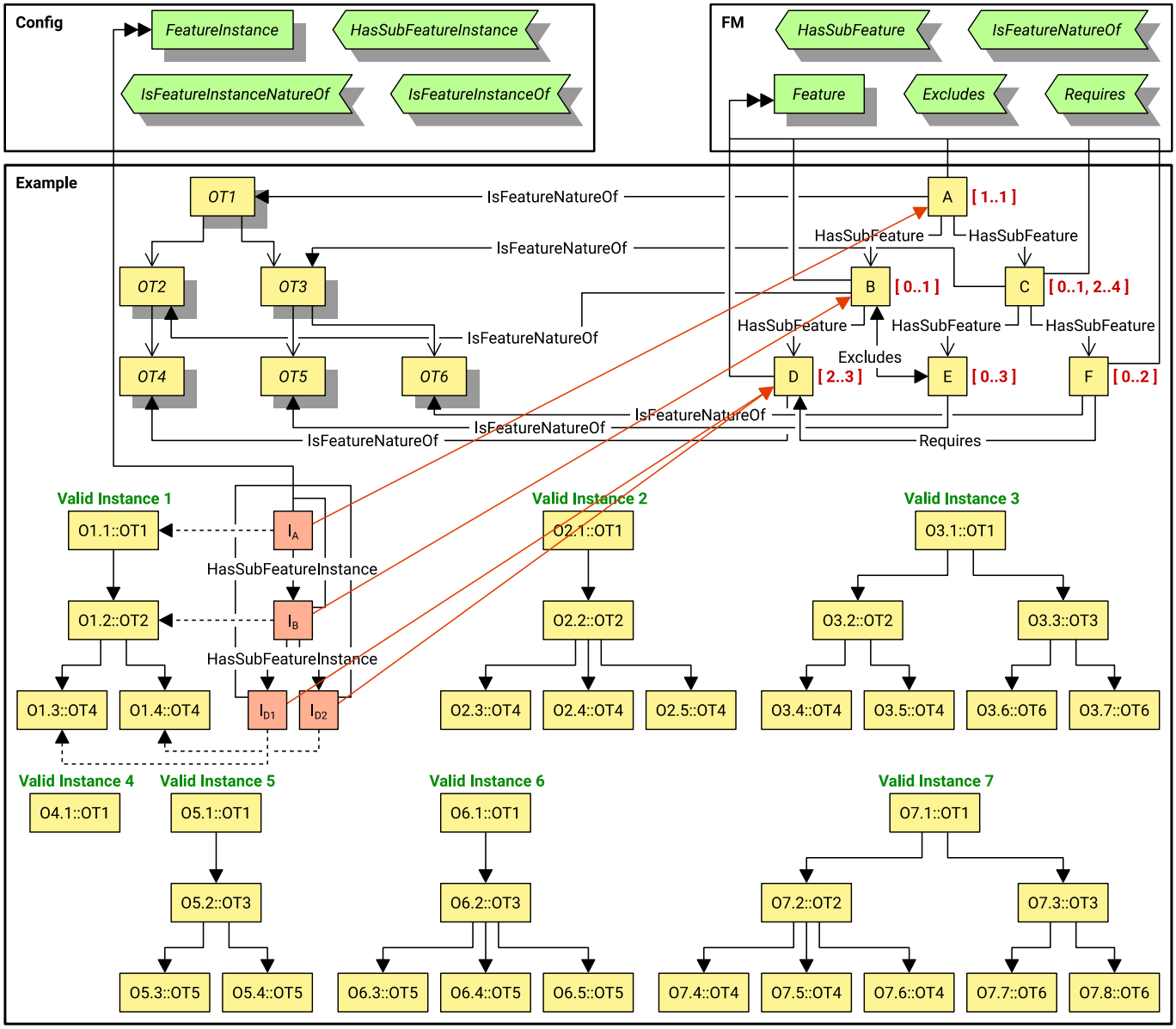

Fig. 3. (Upper compartments) the configuration information model (left) and the variability information model (right, from [22], slightly simplified); (lower compartment) OPC UA information model, valid instances and an explicit configuration model (orange nodes) for our symbolic test case.

5. Discussion

Our slim, yet expressive set of modeling concepts enables the explicit modeling of feature model configurations. It serves three purposes:

1. *Being explicit:* providing explicit models for the configurations provides a clearer understanding of the configuration intent. While such configurations are redundant and could be mined (as explained above) from the information model, changes to the domain model that lead to inconsistencies with the configuration model could provide a hint for incompatible changes in the domain model that might not have been intended.
2. *Adding properties:* in case that the feature model supports properties on feature nodes, these configuration-related properties can be added to the nodes of configuration model, preventing the domain model from being cluttered with external information.
3. *Providing a cache:* explicit configurations can also be seen as a cache for mined feature model configurations and enable quick access to feature model related information without the need for costly re-mining of that data.

6. Conclusion

The OPC UA information model we have presented allows the non-intrusive persisting of configuration information based on feature models in OPC UA. Our approach does not interfere with existing models but has been designed to be instantiated “aside” these models. Our approach provides an implementation of future work as presented in [22]: “Future developments could […] include […] the persisting of configurations”.

We have evaluated our approach in a synthetic test case, discussing the available modeling concepts in breadth. In this test case we could show that our proposed information model is sufficiently expressive to persist configuration information of cardinality-based feature models in the context of OPC UA.

With the availability of configuration information in OPC UA models, it is possible to bring some design time information into the runtime: normally, variability is modeled at design time (and potentially improved at runtime), but this information is often lost in runtime systems. With our approach it is possible to hold a specific view of design time information in the runtime system and allow for the detection of potential conflicts with the intended use of certain components. Also, with the help of the auxiliary information model presented in Sec. 3 it would be possible to track changes in the runtime model over time (model evolution). With this work, we have provided a basis for further analysis of design-time vs. runtime information; this is, however, a topic for upcoming research.

Future developments could further include tool support for online-evaluation of configurations that are undergoing manipulation in an OPC UA server. Frequent checking of configuration validity could inform users about conflicts in OPC UA enabled equipment with their intended use cases. This enables a kind of introspection mechanism that could notify users in case essential parts of a configuration have been altered.

Acknowledgements

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.

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] V. Vyatkin, "Software Engineering in Industrial Automation: State-of-the-Art Review," *IEEE Transactions on Industrial Informatics,* vol. 9, no. 3, pp. 1234-1249, 2013.

[3] B. Vogel-Heuser, J. Mund, M. Kowal, C. Legat, J. Folmer, S. Teufl and I. Schaefer, "Towards interdisciplinary variability modeling for automated production systems: Opportunities and challenges when applying delta modeling: A case study," in *Proceedings of the 13th IEEE International Conference on Industrial Informatics (INDIN 2015)*, 2015.

[4] J. Deuse, C. Heuser, B. Konrad, D. Lenze, T. Maschek, M. Wiegand and P. Willats, "Pushing the Limits of Lean Thinking - Design and Management of Complex Production Systems," in *Closing the Gap Between Practice and Research in Industrial Engineering*, E. Viles, M. Ormazábal and A. Lleó, Eds., Springer International Publishing, 2018, pp. 335-342.

[5] D. Benavides, S. Segura and A. Ruiz-Cortés, "Automated Analysis of Feature Models 20 Years Later: A Literature Review," *Information Systems,* vol. 35, no. 6, pp. 615-636, 2010.

[6] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," 1990.

[7] K. Czarnecki, S. Helsen and U. Eisenecker, "Formalizing cardinality-based feature models and their specialization," *Software Process: Improvement and Practice,* vol. 10, no. 1, pp. 7-29, 2005.

[8] N. Papakonstantinou, S. Sierla and K. Koskinen, "Object oriented extensions of IEC 61131-3 as an enabling technology of software product lines," in *Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2011)*, 2011.

[9] D. Batory, "Feature Models, Grammars, and Propositional Formulas," in *Proceedings of the 9th International Conference on Software Product Lines (SPLC 2005)*, 2005.

[10] K. Czarnecki and A. Wasowski, "Feature diagrams and logics: There and back again," in *Proceedings of the 11th International Conference on Software Product Lines (SPLC 2007)*, 2007.

[11] K. Czarnecki and C. H. P. Kim, "Cardinality-Based Feature Modeling and Constraints: A Progress Report," in *Proceedings of the International Workshop on Software Factories, Colocated with OOPSLA'05*, 2005.

[12] K. Bąk, K. Czarnecki and A. Wąsowski, "Feature and Meta-Models in Clafer: Mixed, Specialized, and Coupled," in *Proceedings of the 3rd International Conference on Software Language Engineering (SLE 2010)*, 2010.

[13] M. Antkiewicz, K. Bąk, A. Murashkin, R. Olaechea, J. H. J. Liang and K. Czarnecki, "Clafer tools for product line engineering," in *Proceedings of the 17th International Software Product Line Conference co-located workshops*, 2013.

[14] M.-O. Reiser, "Managing Complex Variability in Automotive Software Product Lines with Subscoping and Configuration Links," 2008.

[15] A. Rein-Jury, "Feature Constraint Propagation along Configuration Links for Advanced Feature Models," 2013.

[16] I. Schaefer, "Variability modelling for model-driven development of software product lines," in *Proceedings of the 4th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS 2010)*, 2010.

[17] M. Wimmer, P. Novák, R. Šindelár, L. Berardinelli, T. Mayerhofer and A. Mazak, "Cardinality-based Variability Modeling with AutomationML," in *Proceedings of the 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2017)*, 2017.

[18] A. Lüder, M. Schleipen, N. Schmidt, J. Pfrommer and R. Henßen, "One step towards an Industry 4.0 component," in *Proceedings of the 13th IEEE Conference on Automation Science and Engineering (CASE 2017)*, 2017.

[19] C. Lutz, "Rechnergestütztes Konfigurieren und Auslegen individualisierter Produkte," 2011.

[20] L. J. H. Weilguny, "Fertigungsgerechte Produktgestaltung mittels integrierter Produkt- und Prozess-Features: Knowledge-based Engineering unter Einsatz von Feature-Technologie in der Automobilindustrie," 2009.

[21] F. Pauker, S. Wolny, S. M. Fallah and M. Wimmer, "UML2OPC-UA - Transforming UML Class Diagrams to OPC UA Information Models," in *Proceedings of the 11th CIRP Conference on Intelligent Computation in Manufacturing Engineering (CIRP ICME 2017)*, 2017.

[22] B. Wally, C. Huemer, A. Mazak and M. Wimmer, "A Variability Information Model for OPC UA," in *Proceedings of the 23rd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2018)*, 2018.

[23] J. R. Hauser and D. Clausing, "The House of Quality," *Harvard Business Review,* 1988.

[24] E. Mätzler, B. Wally and A. Mazak, "A Common Home for Features and Requirements: Retrofitting the House of Quality with Feature Models," in *Proceedings of the 9th International Workshop on Variability Modeling in Software-intensive Systems (VaMoS 2015)*, 2015.

[25] J. F. Terwilliger, P. A. Bernstein and A. Unnithan, "Automated Co-evolution of Conceptual Models, Physical Databases, and Mappings," in *Proceedings of the 29th International Conference on Conceptual Modeling (ER 2010)*, 2010.