Towards SysML v2 as a Variability Modeling Language

***Abstract*— Systems engineers are looking for ways to reuse system models to accelerate the initial design development process on new projects. Despite its promise, the deployment of model-based product line engineering (MBPLE) presents challenges to practitioners. Not only is deployment costly, but there is an overwhelming number of variability modeling language options that also lack interoperability. While research efforts work towards creating a universal variability modeling language and developing transformations between existing languages, this paper proposes incorporating variability modeling concepts into a system modeling language for better tool support. In this work, we highlight SysML v2 as a promising language to use for variability modeling, arguing that it can capture concepts from feature-based and variation point-based variability modeling languages, while also offering the capability to orthogonally model systems and their variability. Further, we propose a new metamodel that describes a SysML v2-based approach to variability modeling. Since the SysML v2 specification is still in its early stages of standardization, this paper discusses how efforts in deploying MBPLE with existing approaches will translate to the new metamodel and includes considerations for translating from currently available commercial modeling tools. Ultimately, we hope that the proposed metamodel will motivate SysML v2 tool vendors to incorporate instantiation mechanisms and visualization capabilities in future tool releases.**

Keywords—variability modeling, SysML v2, model-based product line engineering

I. Introduction

The time pressures experienced in early product development are increasingly demanding the reuse of engineering and product assets. In most industries, contract award is typically done through Request for Proposals (RFPs). These RFPs provide bidders with a limited time to propose a preliminary solution to an initial set of needs. To remain competitive in the bidding process, companies often propose an adapted version of an existing design [1]. In doing so, they reduce the proposed cost of their solution by counting on reusing products and engineering assets such as reference architectures, requirements, V&V artefacts, and trade-studies. These assets are often digitally stored and referenced from a model since model-based systems engineering (MBSE) is leading the way in terms of being the de-facto approach to systems engineering in complex development [2]. Without a strategic plan for reuse, many of these assets are stored in project-specific system models. Practitioners are seeking strategies to reuse the discretized knowledge contained within these project-specific models to accelerate the process of generating RFP responses.

Strategic reuse of existing engineering assets is critical for cost-effective product development. Variability modeling is an essential technique that enables engineers to identify the reliability or qualification limitations of existing assets in the pre-bid phase. Although modeling variability results in higher pre-bid phase costs, it reduces bid and post-bid costs by front-loading design activities [3], allowing engineers to adapt to RFP-specific concerns instead of reinventing solutions to previously solved problems. In addition, variability modeling can result in a beneficial requestioning of stakeholder needs by leveraging the ability to generate solutions quickly based on existing knowledge. While most research and applications of variability modeling have focused on the software industry [4], [5], recent efforts in the aerospace industry have led to the development of Model-based Product Line Engineering (MBPLE) which utilizes a structured model-based approach to create product lines of engineering assets [6], [7]. Like MBSE, MBPLE deployment requires a variability modeling language, method, and tool. This work focuses on the language and associated implications on tool implementation.

A. Variability Modeling Language Trade-off Context

There are several variability modeling languages, each with a trade-off between interpretability and expressiveness. The numerous language extensions proposed in the literature have led to a plethora of options, making it difficult for practitioners to choose [8]. To focus research efforts and obtain valuable feedback from practitioners for tool vendors to address, a common approach is necessary. Several community efforts have been undertaken to standardize variability modeling concepts, but they have not yet been widely adopted. The numerous publications proposing translation between languages [8]–[12] highlights this gap in adoption consistency and lack of tool interoperability. Feichtigner et al. [12] argue that the difference between these languages is not their ability to capture variability, but rather their expressiveness. Therefore, we look to leverage the interoperability proposed by advancements in system modeling while maintaining the expressiveness of standalone variability modeling languages.

B. SysML v2 as a Common Variability Modeling Language

A common variability and asset modeling tool can prevent dependence on tool interconnectivity and tool-vendor lock-in [6]. In this paper, we propose SysML v2 as a common variability

modeling language. Modeling assets and their variability using a common language encourages the development of a common tool. SysML v2 is an upcoming standard that improves model robustness and interoperability of the language, and it natively variability modeling concepts [13]. The SysML v2 Submission Team (SST) has developed and proposed a solution in response to the request for proposals (RFP) for SysML v2 put forth by OMG in 2017. The new standard is expected to become widely adopted as it improves upon the current de-facto language, SysML v1. At the time of writing, the OMG approved the SST submission, which is undergoing finalization before being released as a beta specification. The most recent release specification can be found online [14].

II. Review of Existing Methodologies

Previous work [9] grouped the multitude of variability modeling languages in five categories: feature-based, variation point-based, UML-based, decision-based, and domain specific languages. Since each category is composed of multiple options, the implementation details such as supported datatypes are omitted from the comparisons presented in this paper. Instead, we focus on the concepts needed to model variability. The following discussion is limited to categories which can be supported by industrial tooling. Therefore, the decision and domain specific categories are not discussed.

Throughout this paper, we consistently use an example from Forlingieri and Weilkiens [6] of a full-height stowage for an aircraft cabin. The example serves as a basis to compare the different variability modeling languages including the proposed SysML v2-based approach.

A. Feature-based Variability Modeling

Feature modeling is the most commonly used approach in many industries [15], including aerospace, automotive, and software, where it has shown numerous benefits [16]. The ISO/IEC 26580 standard presents a high-level process for implementation in a narrower product line engineering context [16], [17]. However, despite the standard providing guidelines, each tool implements feature modeling concepts differently within the allowed freedoms. Existing tools include Pure::Variants, Gears, and FeatureIDE.

Features are abstract characteristics of a system that differentiate it from others [16]. They are organized into hierarchical feature trees, as shown in Figure 1, and a bill of features represents the selection of certain features in these trees. Configurators extract elements from the asset model based on the bill of features, which satisfies the model constraints, resulting in a product instance that represents a product line being instantiated for a given context.

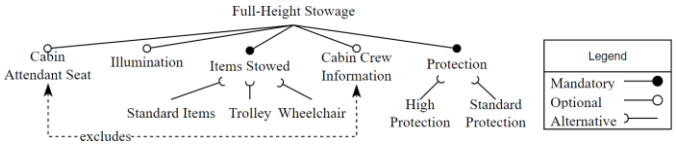

Figure 1 - Feature-based variability model of a full-height stowage (adapted from [6])

While feature-based modeling has been widely deployed and has shown benefits, it is not without its pitfalls, which have been addressed with various other languages. This widespread adoption resulted in *feature* becoming a common term understood by most stakeholders involved in product development, testing, and marketing [18]. As it turns out, the term *feature* has proven to be a source of confusion since it holds different value for each stakeholder [19]. This ambiguity has led to criticisms of feature trees [19].

B. Variation Point-based Variability Modeling

Variation point-based modeling was introduced as an alternative language to enhance or replace feature modeling [19]. While the feature modeling approach represents modeling commonality and variability within a product line, variation point-based modeling only represents the variability around a separate but common core. This scoping change reduces the number of valid interpretations for a variability model. Variation points indicate that an asset varies and can be represented by any of its variants. This variability modeling approach is implemented in PTC's Windchill tool suite. Figure 2 exemplifies the main concepts and the graphical notation of this approach.

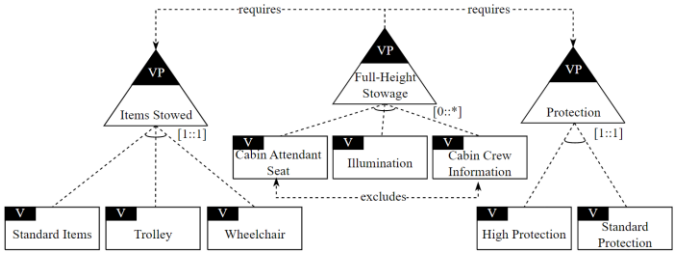

Figure 2 - Variation point-based variability model of a full-height stowage (adapted from [6])

Variation point-based modeling is modular in nature since each variation point cannot have more than 1 level of decomposition into variants. Variation points and variants are related through variability and constraint dependencies. Each variability dependency is denoted by a different line type. Note that the example in Figure 2 only shows alternative variability dependencies. On the other hand, constraint dependencies indicate that the selection at one variation point *requires* or *excludes* another selection. These relationships can also be used to create a feature-like hierarchical decomposition, although it is not semantically required.

C. UML-based Variability Modeling

Due to the popularity of UML and SysML as system modeling languages, they were adapted to incorporate variability modeling concepts. The concepts are either based on the feature modeling [6] or variation point modeling approaches [20], [21]. UML-based approaches stereotype UML elements instead of proposing new ways of modeling variability. The same principle applies to SysML elements, as seen in Figure 3, since it is a UML-based language. Should equivalent concepts exist in SysML v2, work done incorporating the variability modeling concepts in UML-based languages will be directly transferable using the upcoming SysML v1 to SysML v2 transformation. Since this approach is a profile extension of the existing languages, it is supported by any UML and SysML modeling tools.

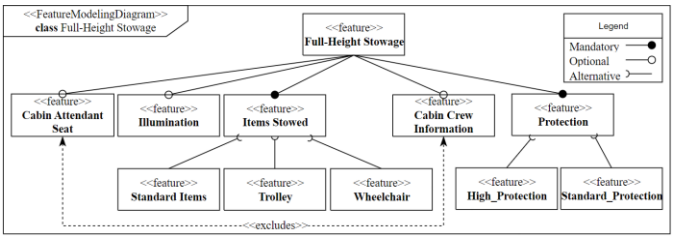

Figure 3 – SysML feature-based variability model of a full-height stowage (adapted from [6])

Using this approach for variability modeling eliminates the need to buy a license and train practitioners on a new tool [21]. The tool-centralized nature of this approach also prevents companies from having to secure a new environment should the model data require that constraint. Despite being in the same tool and using the same elements, practitioners must still learn the UML/SysML profile extension and associated modeling rules.

III. SysML v2-based Variability Modeling and Reuse

A. Native Variability Modeling Concepts

At its core, SysML v2 contains multiple concepts that enable strategic reuse and variability modeling. The ISO/IEC 26580 standard states that a valid variability language should include: a means to capture auxiliary information, a variation point language, a hierarchical decomposition, a constraint language, a feature catalogue, and an instantiation language [16]. By covering all of these concepts, we claim that SysML v2 is a valid variability modeling languageSysML v2 covers all of these concepts, allowing it to be a valid variability modeling language.

One of the key features of SysML v2 is its ability to enable the reuse of model elements within and between models using definition and usage elements. Invariant definition elements can classify multiple context-specific usages, and both usage and definition elements can capture auxiliary information with attributes, comments, and documentation.

SysML v2 also natively includes variation points and variants which can be applied to definition or usage elements. When an element has the variation keyword, it must have variant usages. This parent-child relationship can be nested to any level, providing the structure needed to create a hierarchical decomposition of variability elements [22].

SysML v2 natively includes constraints which define logical relations that are evaluated to be true or false. In this way, constraints take the form of suggestions or model checks. Constraints can also be asserted meaning that they must be evaluated as true for the model to be valid. A feature catalogue can be created by organizing the constraints, variation points, and variants in a namespace (e.g. a package).

SysML v2 describes an instantiation mechanism through the redefinition relationship, which allows a practitioner to select a variant at each variation point. However, current SysML v2 tools have limited instantiation capability since it must be manually defined. Nonetheless, the resulting configurations can be automatically validated using constraints.

Based on these capabilities, we argue that SysML v2 provides a sufficient set of native concepts that enable variability modeling. It is worth noting that this assessment does not include any potential extensions that may be developed in the future to enhance the language's native capabilities.

B. Proposed SysML v2 Variability Modeling Metamodel

SysML v2 can harness the intuitive hierarchical decomposition of feature-based modeling, the modularity of variation point-based modeling, and the interconnectedness of UML-based modeling. The proposed metamodel, as seen in Figure 4, outlines how native SysML v2 concepts can be arranged to perform variability modeling using notation from the latest release specification [22]. The diagram is composed of four major themes: variability, variability multiplicity, constraint dependency, and artefact dependency.

Variability is represented using variation points and variants, similar to the variation point-based approach. Variant memberships represent the parent-child relationship between these variability elements. To reduce model complexity and improve visual clarity, orthogonality is an essential concept in variability modeling [16], [19]. Orthogonality refers to the separation between the variability and asset models, enabling the centralized management of variability elements and dependencies [16]. The metamodel supports orthogonality by not defining the usages to which variation points and variants are associated. For example, a variability model can designate variation points on requirements and an asset model can designate variation points on parts. In doing so, the same variation point and variant concepts can be applied to various levels of abstraction. Despite being modular in nature, the metamodel also supports hierarchical decomposition by allowing the nesting of variation points in variants.

Variability multiplicity represent the types of selections one can make at a variation point. They are represented by defining the multiplicity of elements at each end of a variant membership. A mandatory dependency corresponds to a variant with a default multiplicity of [1..1]. It represents the presence of that variant in every valid configuration regardless of context. In other words, a mandatory variability dependency represents the commonality between all the configurations of a product line. To prevent the variability model from capturing commonality to the point where it becomes too complex, it should be used sparingly to create structure in an abstract variability model. An optional dependency represents a variant with [0..1] multiplicity, implying that they may be omitted from selection. An alternative dependency is used if a variation point has more than one optional variant. The number of alternatives which can be chosen at that variation point is denoted by its multiplicity. For example, if a variation point requires the selection of at least 2 of its 3 variants, then the variation point has a multiplicity of [2..*]. Should the context require it, the asterisk can be replaced with a maximum value to constrain the upper bound of choices. In summary, the mandatory and optional dependencies constrain variant multiplicity, while the alternative dependencies constrain the variation point multiplicities.

Constraint dependencies represent the restrictions imposed on selection options at variation points based on selections at other variation points. This dependency can be between any combination of variation point and variant with the exception being from a variation point to a variant which is already

represented using variability dependencies. Defining a *requires* constraint dependency from a variation point to a variant is equivalent to nesting a variation point in a variant. Since SysML v2 does not include a native graphical representation for constraints dependencies, nesting enables the more intuitive hierarchical representation of variability decomposition.

Artefact dependencies represent an asset being tagged as a variant. For example, when representing variability in an asset model, each variant is a product asset. When a variant is selected, then its dependent asset becomes a product asset instance of the resulting configuration. Best practice states that an artefact should have a directed dependency towards a variant to prevent having to modify the variability model for every change in the asset model [23].

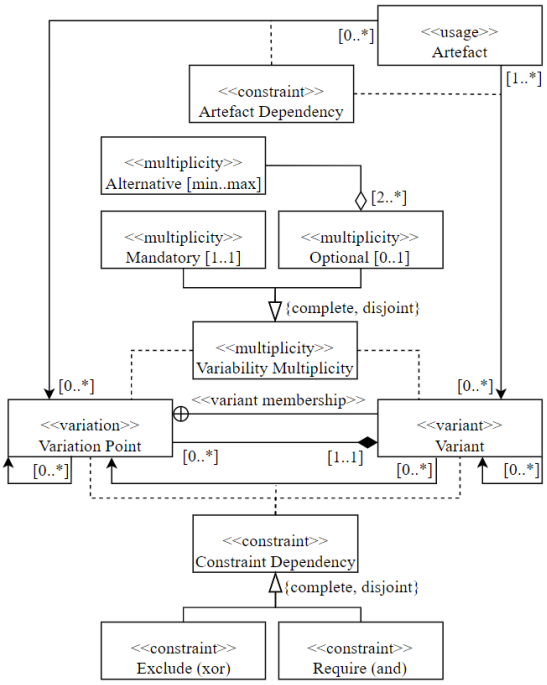

Figure 4 - SysML v2 variability modeling metamodel proposal (partially adapted from [19])

C. Comparison to Existing Work

Although strongly inspired by existing work [19], the proposed metamodel is novel due to its considerations of the limitations imposed by SysML v2. The first difference is how variability dependencies are represented. The OVM notation presented in Pohl et al. [19] describes unique graphical links between variation points and variants defining the mandatory, optional, and alternative selections. In contrast, SysML v2 has only a single relation, termed variant membership, which links variation points to their associated variants. To overcome this challenge, we propose variability multiplicity to represent selection groups. The second difference is the inclusion of a composition or nesting relation between variants and variation points. Nesting improves interpretability by enabling intuitive hierarchical decomposition commonly associated with feature modeling approaches. The third difference is related to artefact and constraint dependencies. While Pohl et al. [19] proposes a graphical artefact dependency notation from variability elements towards their associated artefacts, we propose a textual notation and change the dependency direction. The change in notation comes from the lack of graphical notation in SysML v2 and the change in direction of the dependency is to facilitate variability model maintenance and comply with best practice described in the feature modeling space [23]. Like artefact dependencies, defining constraint dependencies using SysML v2 constraints means that they are not graphically visualized. In summary, the proposed metamodel is novel in that it addresses concerns specific to SysML v2.

IV. SysML v2 Implementation of a Full-Height Stowage

In this section, we use the same full height stowage example [6] to compare with existing variability modeling approaches and demonstrate how to deploy the proposed metamodel using a current SysML v2 implementation. Specifically, we used the textual notation in JupyterLab to create a SysML v2 model and PlantUML to generate the graphical visualizations.

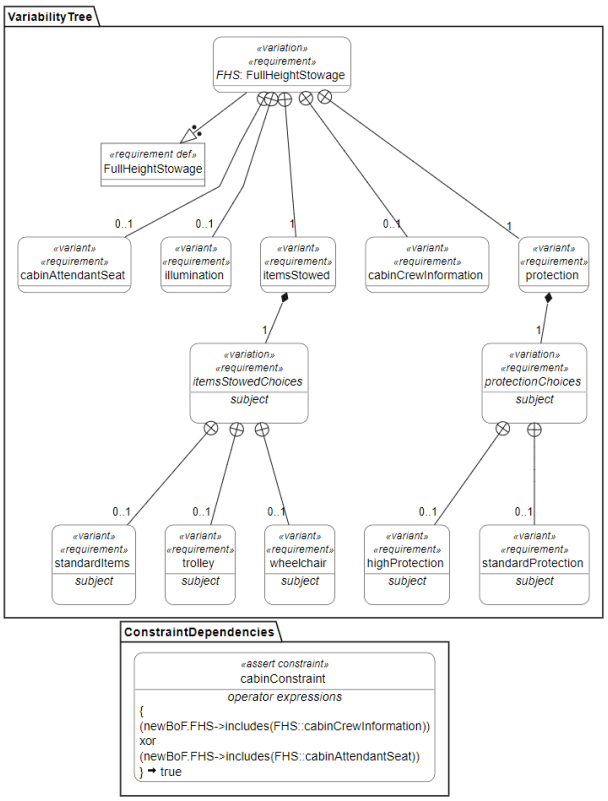

Figure 5 – SysML v2-based variability model of a full-height stowage composed of a variability tree and associated constraint dependencies (adapted from [6])

As seen in Figure 5, we use requirements to denote that the described variability is abstract and specifies artefacts in the asset model. The elements form a hierarchical structure thanks to variability nesting. Since a variant cannot also be a variation point, nesting results in the creation of the *itemsStowedChoices* and *protectionChoices* elements through which the structure can be visualized. It is also worth noting that the constraint dependencies are textually shown outside the diagram since they do not have equivalent graphical notations. The constraint states that a valid configuration is obtained if the new bill of features (BoF) for the full-height stowage (FHS) includes either *cabinCrewInformation* or *cabinAttendantSeat*.

The difficulty with the current implementation of SysML v2 lies in its ability to instantiate an asset model based on selections made in a variation model. As seen in Figure 6 (a), instantiation is possible by manually redefining (:>>) a variant to its owning variation point. Despite instantiation being possible, SysMLv2 needs an automatic instantiation mechanism to translate the choices made in the variability model to their dependent asset variants Figure 6 (b). Without the mechanism, practitioners have to make choices in both variability and asset models even though the choices in the asset model are already constrained.

a

```
package BillOfFeatures{
    import VariabilityTree::*;

    requirement newBoF :> FHS{
        requirement :>> FHS = FHS::cabinAttendantSeat;
        requirement :>> FHS.protection.protectionChoices
            = FHS::protection::protectionChoices::standardProtection;
        requirement :>> FHS.itemsStowed.itemsStowedChoices
            = FHS::itemsStowed::itemsStowedChoices::standardItems;
    }
}
```

b

```
variation part :>> stowageCompartment {
    variant part trolleyCompartment [1]{
        assert constraint {
            newBoF.FHS-> includes(FHS::itemsStowed::itemsStowedChoices::trolley)
            }
    }
    variant part wheelchairCompartment [1]{
        assert constraint {
            newBoF.FHS-> includes(FHS::itemsStowed::itemsStowedChoices::wheelchair)
            }
    }
    variant part standardCompartment [1]{
        assert constraint {
            newBoF.FHS-> includes(FHS::itemsStowed::itemsStowedChoices::standardItems)
            }

        variation attribute bumperCount{
            variant attribute standardNumberOfBumpers{
                assert constraint {
                    newBoF.FHS-> includes(FHS::protection::protectionChoices::standardProtection)
                    }
                attribute :>> standardCompartment.numberOfBumpers [2];
            }
            variant attribute highNumberOfBumpers{
                assert constraint {
                    newBoF.FHS->
                        includes(FHS::protection::protectionChoices::highProtection)
                    }
                attribute :>> standardCompartment.numberOfBumpers [3];
            }
        }
    }
```

Figure 6 – Lack of automatic instantiation mechanism in current SysML v2 implementation (a) Manual variability model instantiation (b) Asset model artefact dependencies (asserted constraints) which cannot be automaticlly instantiated despite being constrained

V. Discussion

This section outlines considerations when transforming each of the previously described approaches to SysML v2. We consider a one-way transformation since the objective is to move to SysML v2 as the interoperability standard.

A. *Transition from Feature-based Modeling*

Feature modeling concepts, such as those in FeatureIDE and Pure::Variants, have equivalent SysML v2 concepts as described in Table 1. The feature modeling concepts were extracted from previous work [9]. In SysML v2, parent and child features correspond to variation points and variants, respectively, while their mandatory or optional status defines their multiplicity. All variation points and variants in SysML v2 are considered abstract, which maintains their feature abstraction. To reduce user-facing complexity and to represent selection outcomes, elements can be hidden by filtering relevant views. Variability dependencies are associated with SysML v2 multiplicity, where the difference between an *or* or *alternative* dependency is the number of options. Specifically, a feature with more than two child features corresponds to a variation point with alternative multiplicity with context-dependent bounds. In such a scenario, an alternative variability dependency corresponds to an upper bound of 1. Additionally, constraints between features correspond to constraint expressions in SysML v2. The supported Boolean comparison relationships such as *not*, *and*, and *or* enable a direct semantic transfer. Furthermore, the requires relation can also be expressed using an *and* constraint between valid combinations of variation points and variants Finally, two features can be considered equivalent by binding them to each other.

Table 1 - Feature-based concepts mapped to SysML v2 concepts

Category	**Variability Modeling Concepts**	
	Feature-based	***SysML v2-based***
Feature	Mandatory	Variation point / variant with default multiplicity [1..1]
	Optional	Variation point / variant with optional multiplicity [0..1]
Property	Abstract	Abstract
	Hidden	View
Variability Dependency	Or	Optional multiplicity [0..1]
	Alternative	Alternative multiplicity [0..1]
	Group Cardinality (n:m)	Alternative multiplicity [n..m]
Constraint Dependency	Not	Not
	And	And
	Or	Or
	Requires	And
	Equivalence	Binding

In addition to those already discussed, tool vendors have implemented their own concepts which must be considered when transforming a variability model to SysML v2. For example, the Pure::Variants implementation of feature modeling contains hard and soft relations to express mandatory and suggested constraints, respectively. In SysML v2, the constraint concept is like the soft relation in that it acts as an expression that can be evaluated true or false without invalidating the model. Soft relations can be thought of as suggestions or best practices. SysML v2 constraints can also be asserted to represent hard relations, which must be true for the model to be valid. Hard relations can be thought of as requirements. As for FeatureIDE, its implementation does not store additional feature-attributed information [9]. Therefore, a FeatureIDE model will not transfer any additional feature-related information to the SysML v2 model.

B. *Transition from Variation Point-based Modeling*

Table 2 illustrates the conceptual overlap between variation point-based modeling and SysML v2. Overall, the transfer of concepts is nearly seamless, with a few exceptions to note. Firstly, the variation point status *mandatory* and *optional* are represented in SysML v2 by equivalent multiplicities, rather than specific keywords. Secondly, variability dependencies are also represented by multiplicities in SysML v2, similar to their representation in feature-based languages, instead of by unique graphical connections and keywords. While SysML v2 does not have direct equivalents for the *requires* and *excludes* concepts, they can still be represented using *and* and *xor* constraints. In addition to these differences, SysML v2 introduces the ability to organize variation point and variant groups within namespaces

(e.g. a package), allowing for greater modularity and organization within the model.

Table 2 - Variation point-based concepts mapped to SysML v2 concepts

Category	Variability Modeling Concepts	
	Variation Point-based	***SysML v2-based***
Unit	Mandatory variation point	Variation point / variant with default multiplicity [1..1]
	Optional variation point	Variation point / variant with optional multiplicity [0..1]
	Variant	Variant
Variability Dependency	Or	Optional multiplicity [0..1]
	Alternative	Alternative multiplicity [0..1]
	Group Cardinality (n:m)	Alternative multiplicity [n..m]
Constraint Dependency	Requires	And
	Excludes	Xor

C. Transition from UML-based Variability Modeling

The main advantage of using a UML-based approach is that it allows for the use of a single tool for both variability and asset models. This is possible because a UML-based approach supports variability modeling concepts, enabling the variability model to directly access assets without the need for dependencies between tools. Alongside cost savings, this results in increased interoperability and a reduced risk from interactions between tools.

Additionally, since UML-based approaches do not introduce new ways of modeling variability, there is no need to discuss a transformation matrix to SysML v2. However, it is worth noting that the implementation of a UML-based approach can vary significantly depending on the modeling tool and domain-specific metamodel being used.

VI. LIMITATIONS

A. Metamodel-Specific

This work assumes that no further changes are made to the variability modeling elements natively present in SysML v2 at the time of writing. The authors believe that the current proposal [22] is unlikely to change substantially enough to invalidate the claims of this paper. However, since the language is not yet published as a complete standard by the OMG, readers are encouraged to revisit the validity of the proposed metamodel before implementing.

As it currently stands, SysML v2 does not support variability bindings which would allow elements to refer to artefacts hosted in other tools such as Simulink, CAD, DOORS, or DOORS Next [22]. Therefore, the variability metamodel can only boast a centralized management of all artefacts similar to UML-based modeling for artefacts supported by the language. For example, should requirements be stored in DOORS, the SysML v2 API could be used to reference artefacts stored in other tools, but this was not explicitly explored. It is possible that future tools may explicitly support the management of other assets using the SysML v2 API [24].

B. Metamodel-Specific

SysML v2 tools are being developed by vendors [25], but their implementations are not widely available at the time of writing. To promote the adoption of the proposed metamodel, a few features could be implemented in SysML v2 modeling tools to support practitioners in adopting a single tool for all variability and asset modeling activities.

SysML v2 modeling tools can best support this metamodel by implementing user-friendly instantiation and constraint definition mechanisms. Instantiation implies the selection of a bill-of-features and the automatic redefinition of the asset model to reflect this selection. Text-based interfaces like the ones currently present in available tools require tedious manual redefinition to represent the selection of a variant at each variation point. This process could benefit from a graphical or tabular interface for practitioners to make their selections. The automatic redefinition of the asset model is not currently supported, rather, constraints can be placed to validate a manual redefinition. An ideal SysML v2 tool would employ the constraints that tie asset variants to feature catalogue elements to automatically redefine the variation points in the asset model.

SysML v2 modeling tools should support constraint and instantiation visualization to facilitate the instantiation process. Concretely, this includes a visual cue to the user that the selection at one variation point impacts the choices they can make at others in the variability model. This also includes a visual cue highlighting the impact a choice has on the assets. SysML v2 supports the use of views as a way of hiding elements which should no longer be part of the model post instantiation. If this concept is used, it will require the automatic definition and application of a view based on the selections made in the variability model.

VII. CONCLUSION

This paper proposes a preliminary metamodel to deploy variability modeling using SysML v2 once it is approved and supported by industrial tools. It builds upon existing feature-based, variation point-based, and UML-based variability modeling approaches by harnessing their respective hierarchical, modular, and tool integration benefits. Tool vendors need to incorporate instantiation and constraint definition mechanisms alongside their visual representations to facilitate the deployment of this metamodel. Ultimately, this paper provides a metamodel which combines the PLE and system modeling language and tools to meet what is referred to as the big picture solution [26].

This paper utilizes an aerospace industry example to illustrate the need in complex development. However, the proposed metamodel can be applied to problems in other industries with some considerations for Return-on-Investment. The authors refer to established work in determining the value added when integrating PLE principles in the systems engineering space [3], [27].

ACKNOWLEDGMENT

References

[1] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An Exploratory Study of Cloning in Industrial Software Product Lines," in 2013 17th European Conference on Software Maintenance and Reengineering, Mar. 2013, pp. 25–34. doi: 10.1109/CSMR.2013.13.

[2] R. J. Torok, "Using Model-Based Systems Engineering Methods to Capture a Department of Defense Acquisition Life Cycle," Naval Postgraduate School, 2021.

[3] J. Fortune and R. Valerdi, "A framework for reusing systems engineering products," Syst. Engin., vol. 16, no. 3, pp. 304–312, Sep. 2013, doi: 10.1002/sys.21232.

[4] A. Allian, E. OliveiraJr, R. Capilla, and E. Nakagawa, "Have Variability Tools Fulfilled the Needs of the Software Industry?," JUCS, vol. 26, no. 10, pp. 1282–1311, Oct. 2020, doi: 10.3897/jucs.2020.067.

[5] M. Cohen, T. Thüm, and J. Mauro, "Proceedings," presented at the 17th International Working Conference on Variability Modelling of Software-Intensive Systems, Odense, Denmark, Odense, Denmark: Association for Computing Machinery, Jan. 2023.

[6] M. Forlingieri and T. Weilkiens, "Two Variant Modeling Methods for MBPLE at Airbus," INCOSE International Symposium, vol. 32, no. 1, pp. 1097–1113, 2022.

[7] R. Malone, B. Friedland, J. Herrold, and R. Houde, "Incorporating Variability and Reuse into System Architecture Models - Principles and Challenges," presented at the INCOSE International Symposium, Detroit, USA, Detroit, USA, Jul. 2018, pp. 1–15. doi: 10.1002/j.2334-5837.2018.00463.x.

[8] K. Feichtinger and R. Rabiser, "Towards Transforming Variability Models: Usage Scenarios, Required Capabilities and Challenges," in Proceedings of the 24th ACM International Systems and Software Product Line Conference - Volume B, Montreal QC Canada: ACM, Oct. 2020, pp. 44–51. doi: 10.1145/3382026.3425768.

[9] K. Feichtinger, C. Sundermann, T. Thüm, and R. Rabiser, "It's your loss: classifying information loss during variability model roundtrip transformations," in Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume A, in SPLC '22, vol. A. New York, NY, USA: Association for Computing Machinery, Sep. 2022, pp. 67–78. doi: 10.1145/3546932.3546990.

[10] D. Romano, K. Feichtinger, D. Beuche, U. Ryssel, and R. Rabiser, "Bridging the gap between academia and industry: transforming the universal variability language to pure::variants and back," in Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume B, in SPLC '22. New York, NY, USA: Association for Computing Machinery, Sep. 2022, pp. 123–131. doi: 10.1145/3503229.3547056.

[11] F. Roos-Frantz, D. Benavides, and A. Ruiz-Cortés, "Feature Model to Orthogonal Variability Model Transformation towards Interoperability between Tools," in Knowledge Industry Survival Strategy Initiative, Auckland, NewZealand, 2009.

[12] K. Feichtinger, J. Stöbich, D. Romano, and R. Rabiser, "TRAVART: An Approach for Transforming Variability Models," in Proceedings of the 15th International Working Conference on Variability Modelling of Software-Intensive Systems, Krems Austria: ACM, Feb. 2021, pp. 1–10. doi: 10.1145/3442391.3442400.

[13] M. Bajaj, S. Friedenthal, and E. Seidewitz, "Systems Modeling Language (SysML v2) Support for Digital Engineering," INSIGHT, vol. 25, no. 1, pp. 19–24, 2022, doi: 10.1002/inst.12367.

[14] E. Seidewitz and M. Bajaj, "OMG Systems Modeling Language (SysML) v2 Release," OMG Systems Modeling Language (SysML) v2 Release, Apr. 11, 2023. https://github.com/Systems-Modeling/SysML-v2-Release (accessed Apr. 14, 2023).

[15] T. Berger et al., "A survey of variability modeling in industrial practice," in Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, in VaMoS '13. New York, NY, USA: Association for Computing Machinery, Jan. 2013, pp. 1–8. doi: 10.1145/2430502.2430513.

[16] ISO/IEC, "Software and systems engineering. Methods and tools for the feature-based approach to software and systems product line engineering," ISO/IEC 26580:2021, Apr. 2021

[17] ISO and IEC, "Software and systems engineering - Reference model for product line engineering and management," ISO/IEC 26550, Dec. 01, 2016

[18] P. C. Clements, "Product Line Engineering Comes to the Industrial Mainstream," INSIGHT, vol. 22, no. 2, pp. 7–14, 2019, doi: 10.1002/inst.12241.

[19] K. Pohl, G. Bockle, and F. van der Linden, Software Product Line Engineering, vol. 10. Heidelberg: Springer, 2005. Accessed: Feb. 04, 2023. [Online]. Available: https://link.springer.com/book/10.1007/3-540-28901-1

[20] O. Ferchichi, R. Beltaifa, R. M. Peña, and L. L. Jilani, "A SysML-based Holistic Variability Modelling of Software and Systems Product Lines," presented at the 34th International Conference on Computer Applications in Industry and Engineering, in EPiC Series in Computing, vol. 79. 2021, pp. 99–112.

[21] T. Weilkiens, Variant Modeling with SysML. in MBSE4U Booklet Series. 2016.

[22] "OMG Systems Modeling Language (SysML) Version 2.0." Feb. 2023.

[23] "pure::variants User's Guide." pure-systems GmbH, 2022.

[24] "Systems Modeling Application Programming Interface (API) and Services." Feb. 2023.

[25] T. Weilkiens, "SysML v2 Modeling Tools," Model Based Systems Engineering, Mar. 09, 2022. https://mbse4u.com/2022/03/09/sysml-v2-modeling-tools/ (accessed Apr. 08, 2023).

[26] M. Chami, M. Forlingieri, and P. Oggier, "Model-Based Variability Management Solution with SysML," 2017.

[27] M. Hause and J. Hummell, "Model-based Product Line Engineering–Enabling Product Families with Variants," INSIGHT, vol. 22, no. 2, pp. 43–48, 2019, doi: 10.1109/AERO.2015.7119108.