International Journal of Software Engineering and Knowledge Engineering

Handling Quantity in Variability Models for System-of-Systems

Daisuke Shimbara

Research & Development Group, Hitachi, Ltd.
Yoshida-cho 292, Yokohama-shi, Kanagawa, 244–0817, Japan
daisuke.shimbara.gk@hitachi.com

Motoshi Saeki* and Shinpei Hayashi†

School of Computing, Tokyo Institute of Technology
Ookayama 2–12–1, Meguro-ku, Tokyo 152–8552, Japan
**saeki@se.cs.titech.ac.jp*
†*hayashi@c.titech.ac.jp*

Øystein Haugen

Faculty of Computer Science, Østfold University College
NO–1757 Halden, Norway
oystein.haugen@hiof.no

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Problem: Modern systems contain parts that are themselves systems. Such complex systems thus have sets of subsystems that have their own variability. These subsystems contribute to the functionality of a whole system-of-systems (SoS). Such systems have a very high degree of variability. Therefore, a modeling technique for the variability of an entire SoS is required to express two different levels of variability: variability of the SoS as a whole and variability of subsystems. If these levels are described together, the model becomes hard to understand. When the variability model of the SoS is described separately, each variability model is represented by a tree structure and these models are combined in a further tree structure. For each node in a variability model, a quantity is assigned to express the multiplicity of its instances per one instance of its parent node. Quantities of the whole system may refer to the number of subsystem instances in the system. From the viewpoint of the entire system, constraints and requirements written in natural language are often ambiguous regarding the quantities of subsystems. Such ambiguous constraints and requirements may lead to misunderstandings or conflicts in an SoS configuration. Approach: A separate notion is proposed for variability of an SoS; one model considers the SoS as an undivided entity, while the other considers it as a combination of subsystems. Moreover, a domain-specific notation is proposed to express relationships among the variability properties of systems, to solve the ambiguity of quantities and establish the total validity. This notation adapts an approach, named Pincer Movement, which can then be used to automatically deduce the quantities for the constraints and requirements. Validation: The descriptive capability of the proposed notation was validated with four examples of cloud providers. In addition, the proposed method and description tool were validated through a simple experiment on describing

variability models with real practitioners.

Keywords: software product lines, variability modeling, system of systems.

1. Introduction

Software product line (SPL) engineering, which involves organizing both common and variable parts of software under development, allows effective derivative software development for multiple systems in the same domain. Such engineering applies "variability modeling" to highlight differences among systems. A typical method of variability modeling is feature analysis [1], in which features express system functionalities and are described as nodes in a tree structure, called a "variability model" or "feature tree." Child features relate to the parent node, and these relations express variable characteristics such as "mandatory," "optional," and "alternative." A root node expressing the overall system can be analyzed, and the system can be decomposed into child and descendant nodes by applying variability relations. The variability model can then be used in the process of selecting features for a system under development.

Modern systems in many industries consist of multiple subsystems that cannot provide functionality in isolation. Rather, they are connected to each other and provide functionality in a coordinated manner. Each subsystem has variability with respect to functional differences and version updates. Subsystems in software may have been developed as SPLs defined by individual variability models. A modern system composed of such subsystems is called a "system-of-systems" (SoS).

Variability of a simple system without considering subsystems expresses the variability that is confined to the system as a whole. Such variability that is not decomposed further is called simple variability in this paper. In contrast, variability of SoS is an expression of the differences among the systems that are composed of subsystems that have their own variability. Therefore, the variability of SoS is composed of subsystem variability, variability of which and how many subsystems compose an SoS, and variability of the functionality as an entire system which is affected by the combination of subsystem variabilities. The difference between the two variabilities is illustrated in the motivating example in Section 2. A subsystem of the system may be decomposed further, resulting in a multistage variability model (Fig. 1). In such cases, the subsystem of the SoS is also treated as an SoS. The subsystem of SoS may have simple subsubsystems or other SoSs as subsubsystems.

In this paper, two problems are addressed.

- It is difficult to express the variability of an SoS with one simple variability model, because there are two different levels of variability. One is the variability of the SoS as a whole, just as a subsystem has variability. The other is the variability of the combination of various subsystems. If these levels are described in one model, it becomes hard to understand and design correctly. Section 2.2.1 describes this problem in detail.

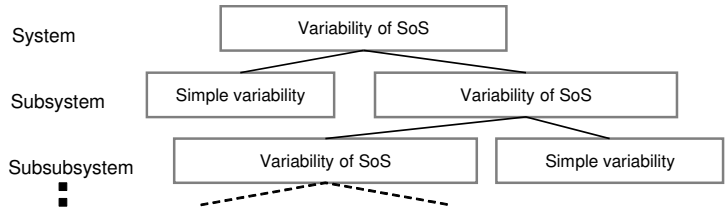

Fig. 1. Multistage variability model.

- For each node in a variability model, a quantity is assigned to express the multiplicity of its instances per one instance of its parent node. Quantities of the whole system may refer to the number of subsystem instances in the system. From the viewpoint of an entire system, constraints and requirements written in natural language are often ambiguous regarding the quantities of subsystems. Such ambiguous constraints and requirements may lead to misunderstandings or conflicts in an SoS configuration. Section 2.2.2 describes this problem in detail.

Therefore, a separate notion for the variability model of an SoS is proposed. The essential points of the proposed approach are the following two ideas:

- A systematic approach to describing the variability of an SoS such that they are simple to describe and possible to analyze. Specifically, the variability model is separated into a system variability model and a structure variability model. Each model is written in Common Variability Language (CVL) [2]. CVL is widely used as a domain-independent language for specifying and resolving variability [3, 4].
- Moreover, a domain-specific notation is proposed to express relationships among the variability properties of systems, to solve the ambiguity of quantities and establish the total validity. This notation adapts an approach, named Pincer Movement, which can then be used to automatically deduce the quantities for the constraints and requirements. The Pincer Movement is a pinching approach by defining a different path than the direct path from source to target in a relationship.

The contributions of this paper are threefold:

- the proposed approach and associated methodology,
- a modeling tool for the variability model of an SoS, and
- validation by means of an experiment.

Section 2 describes the problem via an illustrative example. Sections 3 and 4 describe our approach, and Section 5 introduces the modeling tool. Section 6 describes the experimental validation of our approach. Finally, Section 7 surveys related work, and Section 8 concludes the paper.

2. Illustrative Example and Problem

2.1. *Illustrative Example*

To explain the problems that this study seeks to solve, an illustrative example of an SoS, namely, an air conditioner is introduced. The subsystems of this air conditioner are its inner and outer units. The inner units are embedded in the ceilings of rooms, and the outer units are set on the roof of a building. The number of inner units depends on the number and size of rooms, while the number of outer units depends on the refrigerant capacity required by the inner units.

The SoS thus consists of the inner and outer unit subsystems, and each subsystem is configured for its own variability. The corresponding model is called a "subsystem variability model" in this paper.

Definition 1 (Subsystem variability model). A subsystem variability model is a variability model of a certain subsystem, that contributes to making an SoS. The subsystem variability model is described only for the target subsystem's own functionality, without considering the SoS. A subsystem may be further decomposed.

A subsystem of SoS may be decomposed further, resulting in a multistage variability model as shown in Fig. 1. In the illustrative example, the air conditioner is an SoS and the inner and outer units are subsystems. In the example, the SoS has only simple systems as its subsystems. Details on the case of the multistage variability model are given in Section 3.3.

Figs. 2 and 3 show subsystem variability models of the air conditioner's outer and inner units respectively. In this paper, the variation specification (VSpec) of CVL is used to express the variability model. A VSpec model has a tree structure and is similar to a feature model [1]. The VSpec node types are "Choice," "VClassifier," "Variable," and "CVSpec." A Choice node corresponds to a feature and is shown as a rounded rectangle. A solid line from the parent signifies a mandatory Choice node, while a dashed line signifies an optional Choice node. A VClassifier node includes an instance multiplicity factor that indicates how many instances of it may be created. The *instance multiplicity* describes the allowable number of instances, and it is specified by upper and lower limits. This node is shown as a rectangle with upper and lower limits (but note that Fig. 2 does not include a VClassifier node). A triangle under a Choice or VClassifier node signifies a group multiplicity factor specifying how many choices must be selected from the node's children. A Variable node has an integer value and is shown as an oval. Finally, Fig. 2 does not include a CVSpec node, but that node type is considered later.

In Fig. 2, the outer unit has communication protocol (COM) variability with "$\mathrm{OLD}_{\mathrm{OU}}$" and "$\mathrm{NEW}_{\mathrm{OU}}$," which are mutually exclusive alternatives defined by the group multiplicity "1..1." The new protocol was introduced because the old protocol cannot cope with some of the functionalities added after it was defined. The subscript "OU" denotes the abbreviation of "outer unit" to distinguish subsystems. Later, the subscript "IU" denotes the "inner unit" in the same manner.

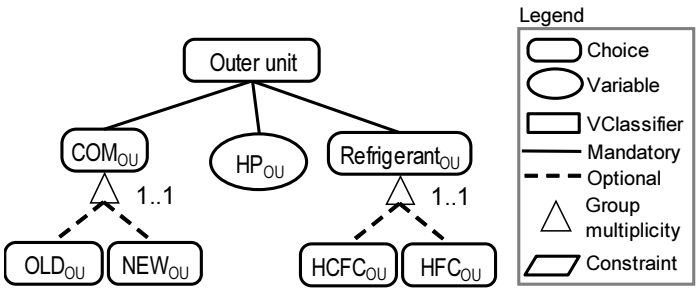

Fig. 2. Subsystem variability model of an outer unit.

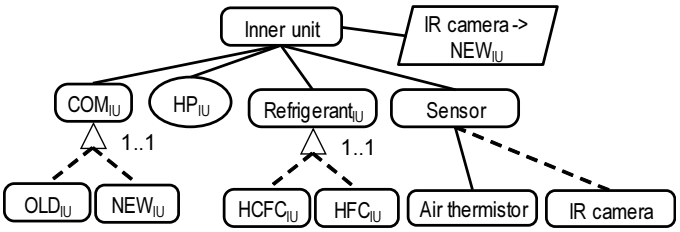

Fig. 3. Subsystem variability model of an inner unit.

Horsepower (HP), which is indicated by the oval Variable node, represents the variability of power capacity. Each outer unit has its own power capacity. Lastly, there are two types of refrigerant: hydro-chloro-fluoro-carbon (HCFC) and hydro-fluoro-carbon (HFC). The HFC type includes later developed refrigerants with better environmental impact.

Fig. 3 shows the variability model of the inner unit. It is similar to the variability model of the outer unit, because it has the COM, HP, and Refrigerant variabilities. While the HP of the outer unit indicates the capacity to provide refrigerant, that of the inner unit indicates the capacity to consume refrigerant. Under the sensor node, the inner unit has an air thermistor as a mandatory node and an infrared (IR) camera as an optional node.

Constraints are imposed between nodes within each subsystem. For example, an inner unit with "IR camera" must have "$\mathrm{NEW_{IU}}$." This means that it requires the new protocol to exchange data recorded by the IR camera. This constraint is local to the subsystem variability model of the inner unit, and most variability languages can support such local constraints. CVL can describe this kind of constraint with Basic Constraint Language (BCL), which is a simple constraint language defined in CVL, or with Object Constraint Language (OCL) [5]. The constraint mentioned above is expressed as "IR camera -> $\mathrm{NEW_{IU}}$" and shown in the parallelogram in Fig. 3.

Table 1. Products of an outer unit.

Product name	COM		HP	Refrigerant	
	OLD	NEW		HCFC	HFC
Outer-LOW	✓		70	✓	
Outer-STD	✓		100	✓	
Outer-HIGH	✓		150	✓	
OuterEX-STD		✓	100	✓	
OuterEX-HIGH		✓	150	✓	
Outer-STD-Eco	✓		100		✓
Outer-HIGH-Eco	✓		150		✓
OuterEX-STD-Eco		✓	100		✓
OuterEX-HIGH-Eco		✓	150		✓

Table 2. Products of an inner unit.

Product name	COM		HP	Refrigerant		IR camera
	OLD	NEW		HCFC	HFC	
Inner-LOW	✓		30	✓		
Inner-HIGH	✓		50	✓		
InnerEX-LOW		✓	40	✓		
InnerEX-HIGH		✓	80	✓		✓
Inner-HIGH-Eco	✓		50		✓	
InnerEX-LOW-Eco		✓	40		✓	✓
InnerEX-HIGH-Eco		✓	80		✓	✓

A subsystem variability model may be specifically bound to obtain a concrete product of the subsystem. For example, Table 1 lists products of the outer unit obtained by binding the subsystem variability model. In the case of CVL, the binding model is called a "resolution model." Nine products are possible, and each product selects its own COM, Refrigerant, and HP value from the choices in the variability model. Table 2 lists products of the inner unit in the same manner.

When an actual air conditioner is configured, products listed in Tables 1 and 2 are selected and connected. The configuration of the outer and inner units needs to satisfy the requirements of the system as a whole. Now we consider one simple system: *an air conditioner for one room, which requires 200 hp for adequate air conditioning*. This simple system can be configured with two "Outer-STD" products and four "Inner-HIGH" products. This configuration both provides and consumes exactly 200 hp to satisfy the system requirement.

As no configuration covers all possible situations, manufacturers of air conditioners sell customized configurations for various situations. They organize the requirements from the market perspective, select and delete those requirements according to the situation, and derive a configuration that satisfies them. The following are two examples of such requirements.

Requirement 1 HCFC refrigerants cannot be used in areas with F-gas regulation.

Requirement 2 The system can detect people in the room so that airflow does not hit them directly.

In general, system requirements are choices, and they are selected or deleted in the requirements definition process. Therefore, these requirements are not always applicable but are selected according to the situation. On the other hand, there are essential constraints that are imposed over the entire system, as in the following three examples.

Constraint 1 The inner and outer units must use the same type of refrigerant.

Constraint 2 The inner and outer units in the entire system must be able to communicate with each other. They have the same COM variability with the choice of OLD or NEW. When these COM variabilities are assigned the same choice, the air conditioner works. Alternatively, outer units with NEW can work even when connected to inner units with OLD because of their backward compatibility. In contrast, inner units with NEW cannot connect to outer units with OLD.

Constraint 3 The consuming HP must not exceed the providing HP.

2.2. *Problem*

In this paper, two problems are addressed.

2.2.1. *Difficulty to Express SoS Variability in One Model*

A variability model of an entire SoS is better described separately from the variability models of subsystems that are developed independently. The paper assumes that each subsystem has a previously defined variability model. Combining them would obscure the distinction between the system variability and the subsystem variabilities, and it may change the subsystem variability models that were already defined. Defining them separately allows for separation of concerns and improves the reusability of the subsystem variabilities. The details are given in Sections 3.

Even if the variability of an entire SoS is expressed separately from the subsystem models, the variability of the SoS is better described as two separated variability models. This is because there are two different levels of variability, namely, system variability and structure variability. By describing the structure independently, it is less error-prone to express and understand how the subsystems are combined. Although these levels could be described in one model by integrating them, such a model consisting of variabilities at the different levels of variability may be confusing.

System variability. A variability model for an SoS should be able to describe requirements and constraints. Requirements, which are satisfied by the entire system, are established by a combination of subsystem functionalities. When configuring an entire system, constraints such as *exclude* or *require* are imposed across

subsystems, yet the subsystem variability models cannot include them, because subsystems do not consider the entire system. In contrast, a variability model for an entire SoS must consider those constraints. The reason for this separation is that, in our experience, each subsystem is developed independently, and its own variability model is analyzed independently from its potential usages. Because of the independence of the system's and subsystems' maintenance responsibility, it is impractical to describe them all together.

In a typical manufacturing organization, the inner and outer units are developed independently in different divisions, and the system division manages configurations of the entire system. Therefore, it is desirable to separate the system variability from the subsystem variabilities. The system division should express this system variability in terms of the requirements and constraints mentioned in Section 2.1. They have relations with the subsystem variability models (Figs. 2 and 3), because they are established by combining subsystem functionalities.

Structure variability. A "system configuration" means which and how many subsystems compose an entire system. It should be derived from the variability model of an SoS, because it is what developers eventually provide as a system. Configuration of an air conditioner system essentially means selecting the types and numbers of the inner and outer units. Instance multiplicities of the inner and outer units are required to be express in the structure variability. The questions of which subsystem and how many instances of them may exist in a system should be expressed as the variability of the entire system.

Both system variability and structure variability describe the entire system, but the perspectives are different. System variability is the variability of the functionality exposed as a system, whereas structure variability represents the variability with respect to combinations of subsystems. Although these perspectives could be described in one model by integrating them, it is less transparent and therefore more error-prone.

2.2.2. *Ambiguity of Which Quantity to Represent*

Next, the following definition is introduced to explain ambiguity of quantity.

Definition 2 (Quantity). The quantity of a child node in a variability model is the number of its instances for a parent node, i.e., the redundancy itself constrained by the instance multiplicity related to it.

An instance multiplicity in a variability model is an expression of a quantity with the allowable number of instances, and it is specified by upper and lower limits. In CVL, an instance multiplicity is expressed as a rectangle node, named VClassifier. Conversely, quantity means the redundancy itself constrained by the instance multiplicities. In the example of the air conditioner system here, the system has at least one inner and outer units, and such constraints can be specified by the instance multiplicities. Here, the number of inner and outer units can be regarded

as the quantities of them.

Constraints and requirements written in natural language are often ambiguous with quantities. Such constraints and requirements may lead to misunderstandings or conflicts in an SoS configuration. To solve such ambiguities, quantities should be identified and handled correctly.

Section 2.1 described Constraint 3 (that the consuming HP must not exceed the providing HP) in terms of a comparison of two HP values. This comparison itself is simple, but the descriptions of "providing HP" and "consuming HP" require attention. For example, consider a configuration with one Outer-LOW unit (HP: 70), one Outer-HIGH unit (HP: 150), and four Inner-HIGH units (HP: 50) (listed in Tables 1 and 2). In this case, there are many possible interpretations regarding the providing HP and consuming HP. Three typical interpretations are the following:

Interpretation 1 The providing HP is the sum of inner unit HP, and the consuming HP is the sum of outer unit HP.

Interpretation 2 The providing HP is the maximum outer unit HP, and the consuming HP is the sum of inner unit HP.

Interpretation 3 The providing HP is the sum of outer unit HP, and the consuming HP is the sum of inner unit HP.

These descriptions involve two ambiguities. The first ambiguity is between the inner and outer units, regarding which one relates to the providing HP and which one relates to the consuming HP. Under Interpretation 1, the providing HP relates to the inner units, and the consuming HP relates to the outer units. Under Interpretations 2 and 3, the providing HP and consuming HP relate to the opposite units from Interpretation 1. Under Interpretation 1, the providing HP is 200, but the consuming HP is 220. Because the consuming HP exceeds the providing HP, Constraint 3 is not satisfied, and this configuration is not suitable as a system.

The second ambiguity is related to how multiple HP instances in subsystems should be aggregated[a] to express the providing HP and consuming HP at the system variability level. One typical approach is to use the sum of all HP instances, while the use of the maximum HP among them might be appropriate. Under Interpretations 1 and 3, all of the aggregation operators are "sum." On the other hand, Interpretation 2 assigns "max" for the providing HP. Under Interpretation 2, the providing HP is 150, but the consuming HP is 200, so Constraint 3 is not satisfied.

The actual intent underlying Constraint 3 is Interpretation 3. With respect to the providing HP, the quantity of the outer units is identified, and the HP values are summed. In the same manner, the consuming HP means the sum of the inner units' HP. This configuration satisfies Constraint 3, because the consuming HP is 200 and the providing HP is 220. Quantities should thus be correctly identified and handled, as in this case of collecting and summing HP instances.

[a]This paper uses the word "aggregation" in the sense of "aggregate functions" in SQL, even though the word is also used in UML [6] for a different concept.

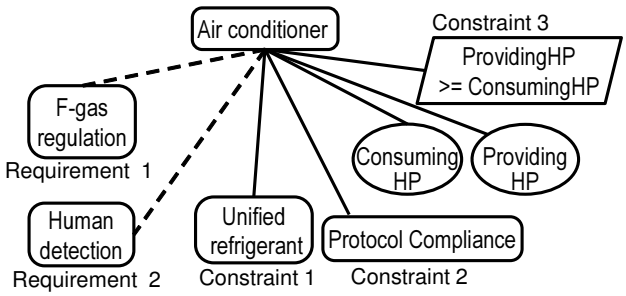

Fig. 4. System variability model.

The variabilities mentioned as those "of a simple system without considering subsystems" in Section 1 correspond to the subsystem variability models of the inner and outer units. Variability of SoS is necessary to express the requirements and constraints, and the variability is affected by the combination of subsystem variability models.

3. Separate Notion of Variability for System-of-Systems

Variability modeling for an SoS is complicated because of several concerns. One concern in SoS variability is the number of subsystems in a system, such as the permissible number of outer units in an air conditioner. Aggregated features, such as total HP, are another concern of variability.

A whole SoS can be seen in two different ways in terms of variability: either as one undivided entity with its own features, or as a composition of the constituent feature models with appropriate quantities. Accordingly, this section introduces a notion of two separate variability models, namely, a system variability model and a structure variability model[b]. In addition, both of these variability models can be consistently combined when resolving the variabilities.

3.1. *System Variability Model*

Definition 3 (System variability model). A system variability model is a model representing the variability in the functionality of an SoS by considering the entire SoS as one entity.

A variability model can be viewed from the standpoint of an entire system, and it relates to the subsystem variability models. It is possible to express system requirements and constraints in the variability model. Decisions in the system

[b]This separate notion is revised from its description in our previous paper [7]. In that paper, these models were called "system feature model" and "structure model."

variability model are determined by the combination of decisions in the subsystem variability models. For example, consider the requirements and constraints over an entire system introduced in Section 2. These requirements and constraints can be regarded as the variability of the entire system.

Potential requirements are elicited from as many stakeholders as possible, and then the actual requirements to satisfy are selected. Hereafter, we state that constraints must be satisfied, while requirements may be optional. The two requirements given in Section 2 may be expressed as shown on the left side of Fig. 4. The connections from the root node are optional because the requirements are selectable.

Requirement 1 is for the possible restriction of HCFCs and is described as an optional choice of "F-gas regulation." In a subsystem variability model, a specific refrigerant type can be selected; from the viewpoint of the system variability model, however, it is important to consider whether regulations require compliance. Requirement 2 for detecting people in the room is described as an optional choice of "Human Detection."

In contrast to the requirements, the constraints across subsystems must be satisfied, because a constraint violation leads to an invalid configuration. In the system variability model, therefore, such constraints become mandatory, as shown on the right side of Fig. 4. For Constraint 1, the mandatory node "Unified Refrigerant" means that all outer and inner units must have the same type of refrigerant. Constraint 2 is a constraint on communication protocols. In the subsystem variability models, the variability of COM with "OLD" and "NEW" exists for both the inner and outer units. In the system variability model, however, it is simply necessary to be able to communicate, and the specific communication protocol is irrelevant. Therefore, a mandatory choice "Protocol Compliance" is introduced. This choice is defined from the viewpoint of the entire system and relates to the COM variabilities of the outer and inner units, as listed in Table 3. If all inner units included in the configuration have the old protocol, the system will work, because outer units with the new protocol have backward compatibility and can simulate the old protocol. On the other hand, inner units with the new protocol cannot communicate to outer units with the old protocol, and this case inactivates the Choice node. Finally, "ProvidingHP" and "ConsumingHP" are Variable nodes that represent the total provided horsepower and the total consumed horsepower, respectively. The HP for the air conditioner is specified via numerical values for these nodes. The inequality in the parallelogram "ProvidingHP >= ConsumingHP" in Fig. 4 refers to Variable nodes and describes Constraint 3.

The nodes representing requirements and constraints in Fig. 4 are described from the viewpoint of the entire system, and their concrete meanings are defined in relation to the subsystem variability models. This is because decisions in the system variability model are determined by the combination of decisions in the subsystem variability models, such as those listed in Table 3. Section 4 discusses these relations in detail.

A system variability model can be used not only for analyzing variability but

Table 3. Decisions for COM variability.

Subsystem		
Outer unit	Inner unit	System Protocol Compliance
OLD_{OU}	OLD_{IU}	Yes (old protocol)
NEW_{OU}	OLD_{IU}	Yes (old protocol)
NEW_{OU}	NEW_{IU}	Yes (new protocol)
OLD_{OU}	NEW_{IU}	No

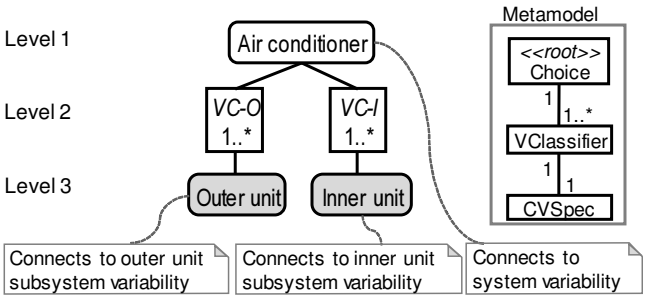

Fig. 5. Structure variability model for an air conditioner.

also for defining the scope of a system under development, considering system architecture with variability, obtaining new features for a derivative product, generating test cases for system testing, and configuring systems.

3.2. *Structure Variability Model*

Definition 4 (Structure variability model). A structure variability model is a variability model to express which subsystems and how many instances of them may exist in an SoS.

The tree of the structure variability model can only have three node levels: a Choice node at the root, VClassifier nodes as children, and CVSpec nodes as grandchildren. The VClassifier nodes indicate the permitted number of instances for each subsystem, while the CVSpec nodes below them signify references to the subsystems.

For example, Fig. 5 shows the structure variability model of an air conditioner. The subsystems of the air conditioner are instances of "outer unit" and "inner unit." Each subsystem has a unique variability, as shown previously in Figs. 2 and 3, respectively. The air conditioner may have one or more outer units, and one or more inner units. The VClassifier nodes express quantities regarding outer and inner units.

Because the syntax of the two types (System and Structure) of variability models uses the same language, it is possible to force them to be combined into one model.

The system variability model is completely independent of the subsystems and describes the variability freely. The structure variability model, on the other hand, describes which subsystems and how many instances may exist in an SoS, within the rigid rule of three layers. Combining the models is more error-prone since the modeler may create associations between these models resulting in contradictions or breaking the three-layer rule.

3.3. *Overview of Variability Models*

The variability models show possible options, alternatives, and variables for defining available variations. As system development progresses, the options or alternatives are bound by applying certain decisions. This binding is a step-by-step operation called "specialization" [8]. For each binding step, a new variability model is obtained by binding from the previous variability model, which requires traceability. In addition, when a variability model has been completely bound to remove all variability, the model has been resolved and is now called a "resolution."

A fully bound product will have bindings of the subsystem variability models, the structure variability model and the system variability model. Fig. 6 overviews these variability models and their bindings. There are three lanes, namely, "individual subsystems," "structure," and "system." The upper parts of the lanes show the variability models, while the lower parts show the bound resolutions.

In the subsystems lane, the upper part corresponds to Figs. 2 and 3, and the lower part corresponds to Tables 1 and 2. A subsystem variability model may be specifically bound to obtain a concrete product for the subsystem.

Moreover, a structure variability model and a system variability model are bound. In the structure lane, the structure variability model shown in Fig. 5 can be bound by selecting subsystem products, resulting in a configuration resolution. For example, two Outer-STD and four Inner-HIGH subsystems may be selected to comprise a system instance.

A system variability model like that shown in Fig. 4 is bound to a system resolution, thus indicating a selection of features that should be implemented by the configuration resolution. This resolution in Fig. 6 has the value 200 for both ProvidingHP and ConsumingHP. As these values satisfy the HP constraint, the configuration resolution is valid.

As mentioned above, variability modeling for an SoS includes both structure variability and system variability, as well as subsystem variabilities. These three variability models are created independently, but there are dependencies between elements of these models as the paper has shown with the example. The dependencies are formally defined through our relational notation.

So far, the subsystems of the SoS have been explained using the simple variabilities which are not decomposed any further in the illustrative example. A subsystem of an SoS may be decomposed further, resulting in a multistage variability model as discussed in Section 1. An example of a multistage variability model is shown

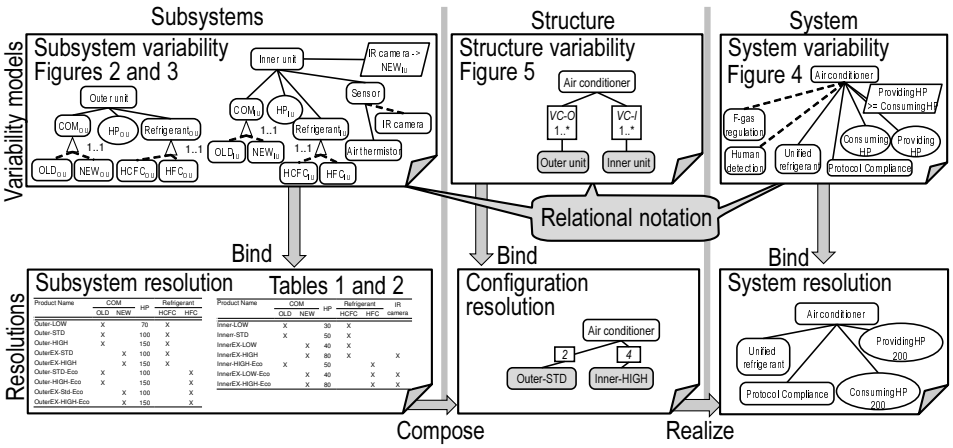

Fig. 6. Overview of variability models and bindings.

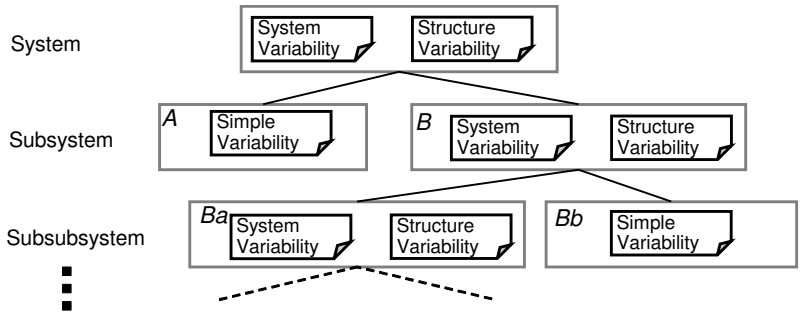

Fig. 7. System variability and structure variability in multistage variability model.

in Fig. 7 in detail. The structural variability in the entire system shows how the subsystems, A and B, are combined. The system variability which shows functional variability of the entire system is influenced by the simple variability model of A and the system variability model of B. The SoS as a subsystem provides its system variability to the higher level system, and the structural variability that indicates how the lower system is combined is hidden to the higher level.

4. Pincer Movement Approach

As mentioned in Section 3.1, decisions in the system variability model are determined by the combination of decisions in the subsystem variability models. Therefore, a description that connects these models is required. In addition, constraints and requirements described in the system variability model are often ambiguous about the quantities. The paper introduces an approach, namely Pincer Movement,

which defined relational notation and steps to identifying and handling the quantities.

Relational notation is a notation that describes the relationships between decisions in the structure variability model with its associated subsystem variability models, and the corresponding system variability model. The formal definition of the relational notation is given in Section 4.4. This relational notation express when requirements and constraints of an SoS are satisfied by a combination of subsystems. Moreover, the notation expresses compatibility across subsystems via the system variability model.

For each relation, the following steps are applied to obtain the relational notation:

(1) specifying the source and target nodes of the relation,
(2) identifying quantities for the relation with the Pincer Movement, and
(3) handling the identified quantities with aggregation operators.

The steps above are now discussed in detail, and a definition language for the relational notation is introduced.

4.1. *Specifying Source and Target Nodes*

A relation connects from a system variability model to a subsystem variability model. For each source node in the system variability model, a target node is manually selected from the subsystem variability model so as to make sense of the source node.

Fig. 8 shows an example for "ProvidingHP." The tree on the left shows an excerpt from the system variability model from Fig. 4. The top-right tree shows the structure variability model from Fig. 5, and the bottom-right tree shows the subsystem variability model from Fig. 2 for the outer unit. "ProvidingHP," the source node in the system variability model, should relate to "$\mathrm{HP}_{\mathrm{OU}}$" in the outer unit's subsystem variability model, because "ProvidingHP" requires summing the values of "$\mathrm{HP}_{\mathrm{OU}}$." Therefore, "$\mathrm{HP}_{\mathrm{OU}}$" is the target node of this relation, as shown by the black arrow in Fig. 8.

4.2. *Identifying Quantities with Pincer Movement*

Here, the concepts of "context" and "path" for identifying quantities are introduced. In addition to pointing the target node directly from the source node, the target node is pointed through the context and path. The application of our domain-specific relational language to define the relations between the system variability model and the structure variability model is what concludes what we call the Pincer Movement of conquering a complex system of systems variability scenario.

Definition 5 (Context). A context is a pointer for the source node of a relation, and it points to the corresponding quantity in the structure and subsystem variability models.

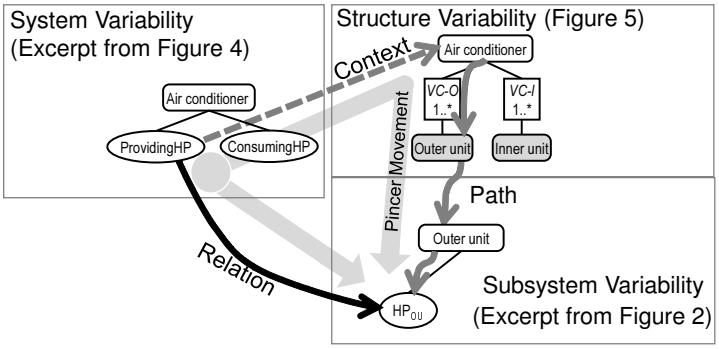

Fig. 8. Pincer movement example for "ProvidingHP."

Note that the root node of the system variability model assigns a context to the root node of the structure variability model, because both nodes are for the entire system. A Choice node and a Variable node in the system variability model do not change their own quantity, and the contexts of these nodes are inherited automatically from the context of the parent node. In contrast to the Choice and Variable nodes, a VClassifier node in the system variability model indicates its own quantity. The context of the VClassifier node is, therefore, manually assigned to a VClassifier node within the structure variability model or a subsystem variability model.

For the example of the relation from "ProvidingHP" to "$\mathrm{HP_{OU}}$" in Fig. 8, the assigned context of "ProvidingHP" is shown by the dashed gray arrow. The context of the root node of the system variability model is assigned to the root node of the structure variability model, and the context of "ProvidingHP" inherits it.

Definition 6 (Path). A path is a route from the node pointed by the context to the target node of a relation. When the path reaches a leaf node of the structure variability model, it then goes to a subsystem variability model, because the leaf node indicates a reference to that subsystem.

A path is uniquely determined if the context and the target node are determined. A VClassifier node that exists on the path represents all multiplexing possibilities in the relation.

For the example of "ProvidingHP" in Fig. 8, the path starts from the root node of the structure variability model and then goes to node "$\mathrm{HP_{OU}}$" in the outer unit variability model, as shown by the solid gray arrow. As a result, it identifies the VClassifier *VC-O*.

4.3. *Handling Identified Quantities*

For the VClassifier nodes on a path, we have to decide how to accumulate the quantities of the VClassifier nodes when writing the corresponding relational notation.

Table 4. Node types and aggregation operators.

System node type	Subsystem node type	Aggregation operator	Comment
Choice	Choice	$\forall$	Indicates that all subsystems are required
		$\exists$	Indicates that at least one subsystem is required
	VClassifier	N/A	
	Variable	N/A	
Variable	Choice	COUNT	Number of choices selected
	VClassifier	COUNT	Number of instances that exist
	Variable	SUM	Sum of subsystem variables
		MAX	Maximum number of subsystem variables
		MIN	Minimum number of subsystem variables
		AVERAGE	Average number of subsystem variables
VClassifier	Choice	N/A	
	VClassifier	CONTEXT	Indicates correspondence of quantity
	Variable	N/A	

In CVL, a node is one of three types: a Choice node for a feature, a VClassifier node for an instance multiplicity, or a Variable node. If the corresponding relation is from a Variable node to a Variable node, how to aggregate (count, sum, max, etc.) for the accumulation is selected. If the relation is from a Choice node to a Choice node, it becomes an accumulator for a Boolean expression (a Choice node can be bound to "selected" or "not selected"). Table 4 summarizes the aggregation operators for relations, covering all possible combinations of node types.

In the example of Fig. 8, "ProvidingHP" should be defined as the sum of the "$\mathrm{HP_{OU}}$" values of the individual outer units, and the path should pass through *VC-O*. In other words, "ProvidingHP" should be determined by accumulating the values of "$\mathrm{HP_{OU}}$" of the outer unit multiplexed by *VC-O* as a sum. Therefore, the SUM operator applies to *VC-O*.

A relation from a Choice or Variable node aggregates the quantities of subsystems appearing on the path of the relation. In contrast, a relation from a VClassifier node of a system variability model does not signify aggregation, as indicated by the three bottom rows in Table 4. When a VClassifier node appears in a system variability model, it shows that a quantity exists from the viewpoint of the entire system, and that the target of the relation has other quantities. Therefore, a relation cannot connect from a VClassifier node to a Choice or Variable node. On the other hand, a relation from a VClassifier node of the system variability model to another VClassifier node in a subsystem variability model signifies that the quantities of the system and subsystem correspond, and that the correspondence has already been

$$
\begin{array}{lll}
\langle relationalNotation\rangle & ::= & \langle systemVSpec\rangle = \langle clause\rangle^* \\
\\
\langle clause\rangle & ::= & \langle choiceClause\rangle \mid \langle variableClause\rangle \mid \langle vclassifierClause\rangle \\
\\
\langle choiceClause\rangle & ::= & \langle subsystemChoice\rangle \\
 & \mid & \langle quantifier\rangle\ {}_{\langle vclassifier\rangle}\ (\langle choiceClause\rangle) \\
 & \mid & \langle choiceClause\rangle \wedge \langle choiceClause\rangle \\
 & \mid & \langle choiceClause\rangle \vee \langle choiceClause\rangle \\
\\
\langle variableClause\rangle & ::= & \langle subsystemVSpec\rangle \\
 & \mid & \langle aggregation\rangle\ {}_{\langle vclassifier\rangle}\ (\langle variableClause\rangle) \\
 & \mid & \langle aggregation\rangle\ (\langle variableClause\rangle +) \\
\\
\langle vclassifierClause\rangle & ::= & \mathsf{CONTEXT}\ (\langle subsystemVclassifier\rangle) \\
\\
\langle quantifier\rangle & ::= & \forall \mid \exists \\
\\
\langle aggregation\rangle & ::= & \mathsf{COUNT} \mid \mathsf{SUM} \mid \mathsf{MAX} \mid \mathsf{MIN} \mid \mathsf{AVERAGE}
\end{array}
$$

Fig. 9. Syntax of the relational notation.

introduced as a context.

4.4. *Definition Language for Relational Notation*

The language for a relational notation is defined by using Backus-Nour form (BNF) as shown in Fig. 9. The terminals of this grammar are given as follows. $\langle systemVSpec\rangle$ means the source node in the system variability model. $\langle choiceClause\rangle$, $\langle variableClause\rangle$, and $\langle vclassifierClause\rangle$ are bodies of the relational notation, and they correspond to the respective source node types. A $\langle choiceClause\rangle$ is a calculation from elements of the subsystem variability models to one Choice node of the system model, and can be combined via $\wedge$ or $\vee$. Likewise, a $\langle variableClause\rangle$ is a calculation from elements of the subsystem variability models to the value of a variable in the system model, and can be combined via aggregation operators. $\langle subsystemChoice\rangle$ and $\langle subsystemVSpec\rangle$ point to a node in the subsystem variability model that is the target of a relationship. In particular, $\langle subsystemChoice\rangle$ can point only to a Choice node, while $\langle subsystemVSpec\rangle$ can point to any type of node. Finally, the subscript $\langle vclassifier\rangle$ points to a VClassifier that is assigned with the aggregation operator.

The grammar of the relational notation is defined, but it is used as a metamodel for a modeling tool that we proposed. Therefore, a user does not have to describe any textual relational notation according to the grammar. After the user specifies the source and target nodes of a relation, the tool can automatically show the quantities to be handled. Then, the user simply selects an aggregation operator for

Table 5. Pincer Movement for an air conditioner.

	Source	Target	VClassifier	Relational definition
Constraint 1	Unified Refrigerant	$\mathrm{HCFC_{OU}}$ $\mathrm{HFC_{OU}}$ $\mathrm{HCFC_{IU}}$ $\mathrm{HFC_{IU}}$	*VC-O* *VC-O* *VC-I* *VC-I*	$(\forall_{VC\text{-}O}(\mathrm{HCFC_{OU}}) \wedge$ $\forall_{VC\text{-}I}(\mathrm{HCFC_{IU}})) \vee$ $(\forall_{VC\text{-}O}(\mathrm{HFC_{OU}}) \wedge$ $\forall_{VC\text{-}I}(\mathrm{HFC_{IU}}))$
Constraint 2	Protocol Compliance	$\mathrm{NEW_{OU}}$ $\mathrm{OLD_{OU}}$ $\mathrm{OLD_{IU}}$	*VC-O* *VC-O* *VC-I*	$\forall_{VC\text{-}O}(\mathrm{NEW_{OU}}) \vee$ $(\forall_{VC\text{-}O}(\mathrm{OLD_{OU}}) \wedge$ $\forall_{VC\text{-}I}(\mathrm{OLD_{IU}}))$
Constraint 3	ProvidingHP	$\mathrm{HP_{OU}}$	*VC-O*	$\mathsf{SUM}_{VC\text{-}O}(\mathrm{HP_{OU}})$
	ConsumingHP	$\mathrm{HP_{IU}}$	*VC-I*	$\mathsf{SUM}_{VC\text{-}I}(\mathrm{HP_{IU}})$
Requirement 1	F-gas Regulation	$\mathrm{HFC_{OU}}$ $\mathrm{HFC_{IU}}$	*VC-O* *VC-I*	$\forall_{VC\text{-}O}(\mathrm{HFC_{OU}}) \wedge$ $\forall_{VC\text{-}I}(\mathrm{HFC_{IU}})$
Requirement 2	Human Detection	$\mathrm{NEW_{OU}}$ IR camera	*VC-O* *VC-I*	$\forall_{VC\text{-}O}(\mathrm{NEW_{OU}}) \wedge$ $\forall_{VC\text{-}I}(\text{IR camera})$

each quantity. Section 5 discusses the tool in detail. In this paper, textual BNF helps to write down relationships of the illustrative example in Section 4.5 and the examples used in the experiments in Section 6.

For "ProvidingHP," the relation can be written as follows by using our definition language:

$$\text{ProvidingHP} = \mathsf{SUM}_{VC\text{-}O}(\mathrm{HP_{OU}})$$

Our relations can also be expressed by standardized notations such as OCL [5]. The developed relational notation is specific to describing the effects of quantities, while OCL is general. Our notation is domain-specific and takes advantage of different kinds of variability specification elements, giving the user valuable guidance and syntactic precision. For "ProvidingHP," an OCL description can be written as the following:

context ac:AirConditioner **inv**:
ProvidingHP = ac.*VC-O*
−>collect(ou:OuterUnit | ou.$\mathrm{HP_{OU}}$)−>sum()

The context in this description specifies the root node of the structure variability model, and it fulfills the same role as a context in our relational notation. *VC-O* can be regarded as a *Bag* in OCL, and "ac" and "ou" indicate their respective instances of the air conditioner and an outer unit. OCL thus has sufficient descriptive capability for this purpose, but it produces a longer and probably less obvious expression than the relational notation does.

4.5. *Illustrative Example Revisited*

In this section, the relational notation is applied to all the requirements and constraints of the air conditioner introduced in Section 2, giving the results listed in Table 5. The first column lists the requirements and constraints. Each requirement or constraint is expressed as a node in the system variability model, as shown in Fig. 4, and the second column of Table 5 lists these nodes as the sources of the relations. The third column lists the target nodes, which are searched according to their meanings. As noted previously, a path is defined as a trace from a context to a target node, and the VClassifier nodes on a path can be identified as quantities. These are listed in the VClassifier column and correspond to the target nodes. The aggregation operators are applied to the identified VClassifier nodes, and the operators are combined as listed in the relational definition column.

Because a system variability model is analyzed from the viewpoint of the entire system, it uses a different expression from that of a subsystem variability model. Constraint 1 and Requirement 1 relate to the refrigerant used in the entire system. Constraint 1 represents a fundamental constraint that the air conditioner cannot operate unless the refrigerant is unified throughout the entire system, and express that either HCFC or HFC in the subsystem is unified and used in the system. Next, "F-gas Regulation" in Requirement 1 indicates that HFC must be used to reduce environmental impact. Thus, Constraint 1 is mandatory for the system to operate, while Requirement 1 is selectable according to whether the corresponding regulation must be met. In the subsystem variability models, the refrigerant name is concretely described as variability, whereas in the system variability model, abstract descriptions are used from the system perspective.

Our relational notation supports compatibility across subsystems. The second row of Table 5 expresses the relation for Constraint 2, which relates to the communication protocol. The new protocol for outer units has backward compatibility to connect to inner units with the old protocol. The Choice node "Protocol Compliance" was introduced in the system variability model, as shown in Fig. 4. This mandatory node indicates two cases. First, when the outer units have the new communication protocol, the system can operate through backward compatibility regardless of which communication protocol the inner units have. Second, when the outer units have the old communication protocol, all inner units must use the old protocol to properly communicate. These cases are combined via the disjunctive normal form. Therefore, complex conditions such as backward compatibility can be expressed with the relational notation.

The third row of Table 5 expresses the relations for Constraint 3. Two Variable nodes "ProvidingHP" and "ConsumingHP" were explained as examples in Sections 4.1 to 4.3. Constraint 3 uses these nodes in an inequality expressing that the consumed HP must not exceed the provided HP. The inequality is described in the parallelogram in Fig. 4. The relations define the quantities of the Variable nodes, and the comparison of the quantities is described as a Constraint node (parallelo-

gram) of CVL as usual.

A relationship that is not visible with only a subsystem variability model can occur in a system variability model. Specifically, the last row of Table 5 gives the relation for Requirement 2, indicating that the system may implement the human detection function for the air conditioner. In the subsystem variability model of an inner unit, there is an optional choice "IR camera," and it is essential for the human detection function. There is a constraint that "IR camera" requires the inner unit to have the new communication protocol. On the other hand, in the subsystem variability model of an outer unit, there is no direct expression related to human detection. A relation to the new communication protocol of the outer unit is required, however, for human detection, because data from the IR camera is transferred via the new protocol.

As described above, by applying our relational notation, it is possible to express all requirements and constraints with a system variability model and relations to subsystem variability models. The aggregation operators characterize the requirements and constraints appropriately.

5. Tool: CT-CVL

A tool, called the "Configuration Tool with CVL" (CT-CVL), was developed for use with our relational notation. CT-CVL is deployed as an Eclipse plug-in. The tool consists of a VSpec editor for subsystem variability models, an editor for subsystem resolution models, a VSpec editor for structure variability models, a VSpec editor for system variability models, an editor for system resolution models, a configuration-resolution generator, and a result viewer for the configuration models. Fig. 10 shows a screenshot of structure variability model editor in the tool. The top left and right area shows the structure variability model and the tools for drawing it, respectively. In the bottom area, the properties of the selected node can be modified. The model shown is the same as the one in Fig. 5. Only the configuration-resolution generator is supported as an automatic function by CT-CVL, while the other editors require manual operations. The editors do, however, have validation functions for models that were input in the other editors.

Using the editor for the system variability model, a user can open a dialog for each node of that model. The specified node indicates the source node of the relation. The dialog shows candidate target nodes from the subsystem variability models, and the user selects the target node. Then, the tool implicitly checks with Pincer Movement and shows the quantities to be handled for the relation, and the user simply selects an aggregate operator for the relation.

CT-CVL uses the Eclipse Modeling Framework [9]. In addition, the tool uses the Graphical Modeling Framework [10] for the VSpec editors and CVC4 [11] as the satisfiability module theories (SMT) solver for checking suitability and constraints.

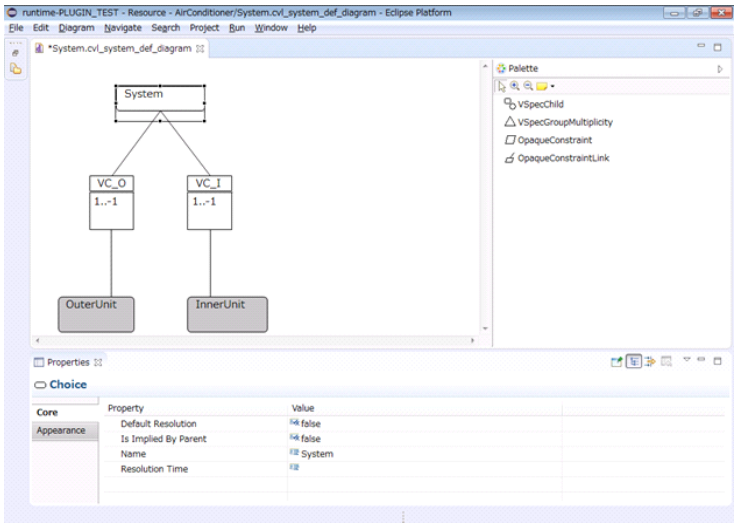

Fig. 10. Screenshot of CT-CVL.

6. Evaluation

The proposed approach may seem complicated, because it describes multiple variability models and the relationships among them. Therefore, the paper evaluates whether this notation can sufficiently describe the variability of a whole system and whether the proposed method can actually be described. The following research questions are thus defined:

RQ1 How much is the descriptive capability of the proposed notation compared with an existing variability modeling method?

RQ2 Is it feasible to describe the requirements and constraints for an entire system by using the proposed notation?

To answer the first question, the paper examined whether an existing variability modeling method could be described by our notation. To answer the second question, we conducted a simple experiment on describing variability models with experts in the field of building personal computers.

6.1. *RQ1: How much is the descriptive capability of the proposed notation compared with an existing variability modeling method?*

As another approach, relative cardinalities have been proposed [12], mainly to represent quantity. The notation of relative cardinalities assumes that only one variability model exists and does not directly support an SoS. Relative cardinalities define not only the quantity for the direct parent node but also that for an ascendant node.

Sousa et al. evaluated their approach by describing variability models for four cloud providers [12]. For example, the following is a simplified version of the cloud provider OpenShift in the notation of relative cardinalities:

```
OpenShift {
    Cartridge [1..100] {
        Gear ⟨OpenShift⟩ [1..100]
    }
}
```

This description indicates that OpenShift can have from 1 to 100 cartridge instances and that one cartridge can have multiple gears. In addition, these gears have a limit of 1 to 100 instances for one OpenShift, and this limitation describes a relation to the grandparent node. This distant relation of quantity is called "relative cardinality."

In the evaluation by Sousa et al., the total number of features appearing in the four cloud providers was 181, 11 of which had relative cardinalities. In addition, a total of 40 constraints were imposed, and 27 of them had relative cardinalities. Therefore, we explored whether these features and constraints could be described by our proposed relational notation.

One of the authors described the variability of the four cloud providers by using the proposed notation and found that all features and constraints were correctly described. While the notation of relative cardinalities had one variability model for each cloud provider, our notation had 21 variability models for the four cloud providers. The total number of nodes for all the variability models was 241, and the number of constraints was 34. Our proposed notation enabled separation of concerns, and therefore, the number of nodes increased.

Relative cardinality can be described with the COUNT and SUM operators in our notation. Fig. 11 shows variability models for the simplified OpenShift as expressed by our notation. A cartridge is defined as a subsystem, and the node "Gear" dangles from the VClassifier node "*VC-G*" in the subsystem variability model. The structure variability model indicates that OpenShift can have from 1 to 100 cartridge instances, with the VClassifier node "*VC-C*." Finally, in the system variability model, the Variable node "NumOfGear" means the number of the gear in an entire OpenShift system. The variable node "NumOfGear" directly relates to the node "Gear" in the subsystem variability model as its name suggests. On the other hand, the context of it points to the root of the structure variability model, because it exists one instance in the entire system. Consequently, the path goes through the "*VC-C*" and "*VC-G*." These VClassifier nodes are handled with appropriate aggregation operators. Therefore, The variable node "NumOfGear" has the following relation:

$$\text{NumOfGear} = \mathsf{SUM}_{VC\text{-}C}(\mathsf{COUNT}_{VC\text{-}G}(\text{Gear}))$$

This relation indicates that the node "NumOfGear" represents the total number of "Gear" instances in OpenShift. The relation and numeric constraint in the paral-

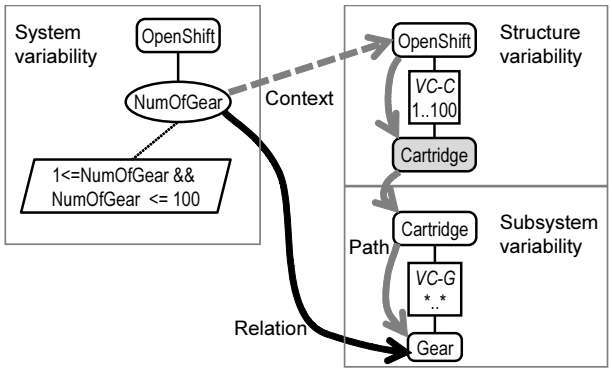

Fig. 11. OpenShift example using the proposed notation.

lelogram can thus describe the relative cardinality.

Therefore, with respect to RQ1, these results confirmed that the proposed relational notation has at least the same descriptive capability as an existing variability modeling method.

6.2. *RQ2: Is it feasible to describe the requirements and constraints for an entire system by using the proposed notation?*

To evaluate the feasibility of the proposed notation, we gave experimental participants the requirements and constraints of an entire system and asked them to describe the corresponding models (i.e., a structural variability model, system variability model, and relations). If all the requirements and constraints could be modeled correctly, then this research question could be answered affirmatively.

In the experiment, six participants had no experience with variability modeling but were practitioners of software engineering in industry, with experience ranging from 0 to 12 years. They were provided with a tutorial of CVL and allowed to ask questions about the proposed notation before attempting to describe variability models. This tutorial included the air conditioner example used in this paper. After that, the participants used CT-CVL to describe a structure variability model and system variability model with the relational notation. While the participants worked on this task, they were only given help with the usage of the tool. Each participant worked separately.

For the task, an example of an imaginary home-built personal computer (PC) system composed of many parts is prepared. The subsystems were the PC case, CPU, memory, power supply, and video card. Subsystem variability models, like those shown in Figs. 2 and 3, were given to the participants. In addition, two constraints and four requirements written in natural language were imposed on the entire PC system. The constraints (C1 and C2) and requirements (R1 to R4) were

the following:

C1 Comparison of aggregated values of subsystem variables, like HP in the air conditioner example.

C2 Compatibility between subsystems, like the backward compatibility of the communication protocol.

R1 and R2 Selection of appropriate aggregation operators. The PC could take multiple video cards, and the video card had multiple digital interfaces. R1 and R2 had similar descriptions; however, while R1 had a relation like "$\exists_{Videocard}(\exists_{DigitalInterface}(\text{Target}))$," R2 had one like "$\forall_{Videocard}(\forall_{DigitalInterface}(\text{Target}))$."

R3 Counting of all instances in subsystems, like the gears in the cloud provider example.

R4 Specification of subsystem instances in the system variability model, as with Requirement 1 in the air conditioner example.

The purpose of the experiment was thus to verify that these constraints and requirements could be properly modeled by the participants using the tool.

Table 6 lists the experimental results[c]. Each row represents a participant. The first column lists the participants' years of experience in industry. The second and third columns show the times in minutes for the tutorial and the actual work. The remaining columns indicate whether the constraints and requirements could be appropriately modeled.

All participants could correctly describe the structure variability model. The modeling time was at most 2 hours, which is within a practical range. Regarding the description of constraints and requirements in the system variability model, however, the results varied among the participants.

Three participants (those with 1, 8, and 12 years of experience) could correctly describe all constraints and requirements in the system variability model. In contrast, the participant with 3 years of experience misunderstood the description of R2 ($*1$). While the requirement statement was "All digital interfaces have the target feature," he understood it as "All video cards have at least one digital interface that has the target feature." We interviewed that participant after the experiment and to confirmed that he could correctly describe the models without misunderstanding. On the other hand, the participant with 10 years of experience could create correct models, but he added an extra description for R2 ($*2$). He defined a Choice node for R2, which had a correct relation of "$\forall_{Videocard}(\forall_{DigitalInterface}(\text{Target}))$." That node, however, was dangled under an extra VClassifier node that was unnecessary. Nevertheless, despite these minor issues, the participants with 3 and 10 years of experience could describe the models correctly most of the time. On the other hand, the participant with no experience made mistakes in modeling both C2 and R2.

[c]The test subject and results of the experiment, including the tutorial and the CT-CVL tool, are available at https://doi.org/10.6084/m9.figshare.13370714.

Table 6. Experimental results for describing models.

Years of software experience	Minutes for tutorial	Minutes for task	C1	C2	R1	R2	R3	R4
0	45	110	✓	×	✓	×	✓	✓
1	45	40	✓	✓	✓	✓	✓	✓
3	45	120	✓	✓	✓	∗1	✓	✓
8	45	65	✓	✓	✓	✓	✓	✓
10	45	65	✓	✓	✓	∗2	✓	✓
12	40	95	✓	✓	✓	✓	✓	✓

Note: ✓: Correct, ×: Wrong, ∗: Almost correct

When that participant is interviewed, it turned out that the cause of his mistakes was the difficulty of modeling (because he had no modeling experience).

Generally, it is natural that people who are unskilled in modeling cannot properly describe models. With respect to the answer to RQ2, our notation seems complicated like any other modeling method, however, it can be applied by someone with experience.

6.3. *Threats to Validity*

The internal validity of the experiment involved two threats. First, the subsystem variability models were provided to the participants, meaning that they only developed the structure variability and system variability models. If the participant had defined the subsystem variability models, it could have influenced the results. The aim of the experiment, however, was to determine the feasibility of describing a system variability model in which multiple subsystem variability models are integrated, instead of judging the possibility of variability modeling. As for the second threat, the participants used the proposed tool to describe the system variability model. The tool may have affected the description results, because the user interface is a little complicated. Specifically, the participants had to navigate between multiple views to describe the relations between models. Accordingly, we helped the participants in using the tools.

For RQ1, the variability models of four cloud providers were confirmed. For RQ2, the illustrative example of an air conditioner was introduced as a tutorial, and a home-built PC system was experimentally described with the proposed tool. One threat to external validity is that the paper did not verify the effectiveness of the proposed method in regard to other, actual complex systems. As with any case-study research, it might not be possible to generalize the experiment beyond the considered cases. Another threat to external validity is that the participants were software engineers from the same company, and such an experiment with employees of only one company may have caused bias. In the future, it will be necessary to investigate trials with various complex systems.

7. Related Work

Other approaches to variability modeling have been discussed from the viewpoints of structure variability, system variability, and relations between variability models. Structure variability modeling requires a description of the variability of the number of subsystem instances via quantity. System variability modeling requires multiple models for an entire system and subsystems, and relations for decision propagation. The relations between variability models are required to describe aggregations of quantities.

Textual languages were surveyed by Eichelberger et al. [13]. Rosenmüller et al. proposed Velvet [14], which focuses on multi-dimensional variability modeling. In the case of Velvet, a system can have instances of subsystems with identifiers (names). Each instance requires a different identifier, however, and the number of identifiers cannot vary. This means that Velvet cannot directly support quantity to describe the number of instances. It can, however, describe the binding of variability. It does so by defining an inheritance relation between a base feature model and an inherited feature model bound from the base feature model.

Classen et al. proposed TVL [15], which can describe quantity via numerical variability. Moreover, it has aggregation operators such as SUM, MAX, and MIN for child instances. These aggregation operators can only be used by a parent node for its children in a tree structure.

Abele et al. proposed VSL [16], which describes quantity. It can describe the number variability of subsystem instances and specify identifiers for instances that should be distinguished from other instances. In addition, VSL provides decision propagation between feature models. This propagation relation is defined as a ConfigLink, which has two types of decision propagation: selection and inclusion. Selection means that all features from the target feature to the root feature are selected when the source of the ConfigLink is selected. Inclusion means that propagation applies only to the target feature. Two types of negative propagation are possible: de-selection and exclusion. Aggregation operators are not shown in the ConfigLink.

Graphical variability languages may support descriptions of system variability. Although FODA [1], in its original form of feature tree modeling, did not support quantity, Czarnecki et al. extended it with an instance multiplicity called "cardinality" [17]. Czarnecki et al. proposed a multi-level staged configuration [8] that enables feature abstraction. Feature models with different abstraction levels are connected, and decisions, such as selecting or removing features in a feature model, are propagated to other feature models. In the multi-level configuration, three propagation types are defined: positive (only propagate a selection decision), negative (only propagate a removal decision), and complete (propagate both decisions). Moreover, Czarnecki et al. proposed constraints between features by using OCL [18]. Although not mentioned directly in the paper, these constraints could possibly support the aggregation of quantities.

Reiser et al. [19] proposed a feature analysis method with a component diagram.

In the component diagram, each component has a variability model, and the entire diagram shows the system structure. Feature relations between components are defined, and five patterns of feature relations are introduced. These patterns focus on the propagation of feature decisions between the system and subsystems.

Chavarriaga et al. proposed decision propagation in a multi-step configuration process [20]. Two types of propagation relations are possible: "force" and "prohibit," which are simple positive and negative feature decision propagations, respectively. This propagation adds a selected or de-selected mark to each target feature. After these marks are added, the existence of a feature marked as both selected and de-selected indicates a conflicting configuration. While decisions are propagated by the relations between feature models of different abstraction levels, an abstraction level may be implemented by changing the viewpoint of the feature model.

Hubaux et al. proposed a multi-view feature based configuration [21]. In this multi-view approach, stakeholders can see each feature model that has been grayed out, pruned, or collapsed from a base feature model from their own viewpoints.

Janeta et al. proposed a formal approach to integrating feature and architecture models [22]. This approach defines decision propagation links between features and components, and it can be regarded as defining the relationship between a system variability model and a structural variability model. In comparison with that approach, the proposed relational notation interposes subsystem variability models between them.

CVL [23] is a domain-independent GUI language for specifying and resolving variability. For variability representation, it has a VSpec model that is similar to feature tree modeling. BVR [24]evolved from CVL, and the VType in BVR may implement a structure variability model.

Krueger et al. [25] proposed an ontology for variability of SoS integrating subsystem feature models and a system feature model. Tekinedagon et al. [26] proposed a feature driven development process extending the ontology for SoS. These are similar to our approach in that the subsystem variability is analyzed independently of the system and the system side analysis combines them. On the other hand, the proposals do not support a representation corresponding to the structure variability model and quantity.

While both the staged configuration and Velvet use the same metamodel for the base model and bound model, dedicated models for resolution, such as VSL [16] and CVL [23], have been proposed. Such a resolution model signifies a bound and resolved model with no variability. VSL has a configuration description that is bound from the feature model. In the case of CVL, the feature model is expressed as a VSpec model, and the resolution model shows selections from the VSpec model. When binding is implemented using the same metamodel, it is easy to describe the binding operation step-by-step. On the other hand, variability models are finally bound to a concrete instance of a system, which has no variability. This concrete instance is distinguished in CVL and VSL. Reiser and Weber proposed a multi-level feature tree [27], which has reference relations between feature models. The

referencing feature model is bound from the referenced model. In addition, the referencing feature model may be extended by means such as adding features, but such extension deviates from the binding.

On the whole, the multi-level configuration by Czarnecki et al. can support structure variability, system variability, and relations between models, but it does not completely support identifying quantities in the relations between variability models.

We previously proposed a method [7] for variability modeling with CVL [23]. CVL originally supported the binding concept, and the proposed method gave a relational notation for system composition without supporting an abstraction level. The application domain of that method is software testing, and it generates a small set of configurations covering all test cases. Although the concept of dividing the variability representation into system and structure variabilities was already mentioned in [7], the available node type was Choice only, and the aggregation operators were "$\forall$" and "$\exists$" only (lines 1 and 2 in Table 4) in the previous proposal. In this paper, the remaining node types (VClassifier and Variable) are now supported. With the addition of the node types, various aggregation operators are introduced. Moreover, it is proposed in this paper that quantities are identified and handled in the relationship between variability models.

The proposed method is compared with related works. For the comparison, we have selected two existing approaches. The first one is the relative cardinality [12], which has the descriptive ability of quantity as discussed in Section 6.1. To the best of our knowledge, this is the only existing work that can directly handle quantities. To make our comparison fairer, we additionally selected the multi-level configuration [8, 17], which can express the relationship between structure variability, system variability, and subsystem variability. Although there is no direct way to express quantity constraints in this approach, the approach is applicable to handle quantities in an SoS if it is used together with existing constraint languages such as OCL. Table 7 shows the comparison summary. The comparison items are the following three items:

- Is it possible to separate variability expression to multiple models and refer to other models? It improves the reusability of the variability model.
- Is it possible to identify quantities between variability models? It makes it easier to be consistent when each variability is described and modified.
- Is it possible to aggregate quantities with various operators between variability models? It makes it possible to describe complicated relations.

In the relative cardinality, variability is represented by a single model. When the variability model is described, there is no way to find the quantities between nodes, which would require manual verification. Concerning the description of relationships, while it is possible to express the upper and lower limits of the number of instances between nodes in the variability model, complex expressions such as the maximum and minimum number of instances are difficult to describe.

Table 7. Comparison with related work.

	Style	Separate model	Identify quantities	Aggregation in relations
Relative cardinality [12]	Text	–	–	–
Multi-level configuration [8, 17]	Graphical	✓	–	–
Proposed method	Graphical	✓	✓	✓

Note: ✓: Available, –: Not available

The multi-level configuration provides a node type that references other variability models. In this approach, there is no way to find the quantities between nodes. Since OCL is introduced with respect to the representation of relations, it is possible to perform all aggregate operations, however, it is not possible to represent the relations between variability models written separately in systems and subsystems. Although aggregate operations can be described by merging the separated variability models into a single model, this would make the model more complicated.

Since the proposed method defines a dedicated notation, structural, system, and subsystem variability are described separately. The proposed method uses a pincer movement approach to easily identify the quantities to be handled. The relational notation in the proposed method supports various aggregate operators.

Although none of the existing methods were equally satisfactory, the proposed methods met all qualities.

8. Conclusion and Future Work

The paper proposed a method for variability modeling of an SoS composed of many instances of subsystems. Each subsystem has its own variability model. It is difficult to express the variability of an SoS with one simple variability model, because there are two different levels of variability. In addition, there are ambiguities in the quantities representing the constraints and requirements of an SoS, because it includes multiple instances of multiple types of subsystems.

Therefore, a separate notion is proposed for variability modeling of an SoS, with a system variability model and a structure variability model. Moreover, the Pincer Movement approach was proposed for identifying and handling quantities with a relational notation. The variability models of an SoS can describe requirements and constraints across subsystems. In addition, a modeling tool that implements the relational notation was developed.

An air conditioner composed of many subsystems is used as an illustrative example. In addition, by describing four cases of cloud providers, we confirmed that the proposed notation has the same descriptive capability as an existing variability modeling method. Furthermore, the proposed notation was experimentally evaluated by describing variability models for another example of a home-built PC

system. In this experiment, participants with experience could correctly describe a system variability model by using the separate notion and the Pincer Movement approach. The supporting tool mentioned in Section 5 helped the participants to identify quantities. These examples are small, however, and case studies with industrial projects will be needed to fully validate the proposed notation.

The paper assumes that the target SoS is not sensitive to differences in subsystem connections, unlike in the case of a network. Our relational notation focuses on quantity. When constraints and requirements for an SoS include how to connect subsystems, our relational notation will need to include the concept of network topology.

References

[1] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak and A. S. Peterson, Feature-oriented domain analysis (FODA) feasibility study, Tech. Rep. CMU/SEI-90-TR-21, Carnegie-Mellon University, Software Engineering Institute (1990).

[2] Ø. Haugen, A. Wasowski and K. Czarnecki, CVL: Common variability language, in *Proc.16th International Software Product Line Conference - Volume 2*, 2012, pp. 266–267.

[3] J.-M. Horcas, A. Cortiñas, L. Fuentes and M. R. Luaces, Integrating the common variability language with multilanguage annotations for web engineering, in *Proc. 22nd International Systems and Software Product Line Conference - Volume 1*, 2018, pp. 196–207.

[4] D. Calegari, A. Delgado and L. Pena, Automated generation of variants in business process families based on the common variability language (CVL), in *2019 XLV Latin American Computing Conference*, 2019, pp. 1–10.

[5] Object Management Group, Object Constraint Language (OCL). Version 2.3.1 (2012), http://www.omg.org/spec/OCL/2.3.1/.

[6] J. Rumbaugh, I. Jacobson and G. Booch, *Unified Modeling Language Reference Manual*, 2nd edn. (Pearson Higher Education, 2004).

[7] D. Shimbara and Ø. Haugen, Generating configurations for system testing with common variability language, in *Proc. 17th International SDL Forum*, 2015, pp. 221–237.

[8] K. Czarnecki, S. Helsen and U. W. Eisenecker, Staged configuration through specialization and multilevel configuration of feature models, *Software Process: Improvement and Practice* **10**(2) (2005) 143–169.

[9] The Eclipse Foundation, Eclipse modeling framework (2007), http://www.eclipse.org/emf/.

[10] The Eclipse Foundation, Graphical modeling framework (2007), http://www.eclipse.org/gmf-tooling/.

[11] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds and C. Tinelli, CVC4, in *Procc of the 23rd International Conference on Computer Aided Verification*, 2011, pp. 171–177.

[12] G. Sousa, W. Rudametkin and L. Duchien, Extending feature models with relative cardinalities, in *Proc. 20th International Systems and Software Product Line Conference*, 2016, pp. 79–88.

[13] H. Eichelberger and K. Schmid, A systematic analysis of textual variability modeling languages, in *Proc. 17th International Software Product Line Conference*, 2013, pp. 12–21.

[14] M. Rosenmüller, N. Siegmund, T. Thüm and G. Saake, Multi-dimensional variability

modeling, in *Proc. 5th International Workshop on Variability Modeling of Software-Intensive Systems*, 2011, pp. 11–20.

[15] A. Classen, Q. Boucher and P. Heymans, A text-based approach to feature modelling: Syntax and semantics of TVL, *Science of Computer Programming* **76**(12) (2011) 1130–1143.

[16] A. Abele, Y. Papadopoulos, D. Servat, M. Torngren and M. Weber, The CVM framework - a prototype tool for compositional variability management, in *Proc. 4th International Workshop on Variability Modelling of Software-Intensive Systems*, *ICB-Research Report* **37**, (Universitat Duisburg-Essen, 2010), pp. 101–105.

[17] K. Czarnecki, S. Helsen and U. Eisenecker, Formalizing cardinality-based feature models and their specialization, *Software Process Improvement and Practice* **10**(1) (2005) 7–29.

[18] K. Czarnecki and C. H. P. Kim, Cardinality-based feature modeling and constraints: A progress report, in *Proc. International Workshop on Software Factories*, 2005, pp. 16–20.

[19] M. O. Reiser, R. T. Kolagari and M. Weber, Compositional variability - concepts and patterns, in *Proc. 42nd Hawaii International International Conference on Systems Science*, 2009, pp. 1–10.

[20] J. Chavarriaga and C. Noguera, Propagating decisions to detect and explain conflicts in a multi-step configuration process, in *Proc. ACM/IEEE 17th International. Conference on Model Driven Engineering Languages and Systems*, 2014, pp. 337–352.

[21] A. Hubaux, P. Heymans, P.-Y. Schobbens and D. Deridder, Towards multi-view feature-based configuration, in *Requirements Engineering: Foundation for Software Quality*, 2010, pp. 106–112.

[22] M. Janota and G. Botterweck, Formal approach to integrating feature and architecture models, in *Proc. 11th International Conference on Fundamental Approaches to Software Engineering*, 2008, pp. 31–45.

[23] Object Management Group, Common Variability Language (CVL) (2012).

[24] Ø. Haugen and O. Øgård, BVR – better variability results, in *Proc. 8th System Analysis and Modeling Conference*, 2014, pp. 1–15.

[25] C. Krueger and P. Clements, Enterprise feature ontology for feature-based product line engineering and operations, in *Proc. 21st International Systems and Software Product Line Conference - Volume A*, 2017, pp. 227–236.

[26] B. Tekinerdogan, S. Duman, H. Caner and B. Durak, Customizing a feature ontology for product line engineering within a system-of-systems context, in *Proc. 2019 International Symposium on Systems Engineering*, 2019, pp. 1–6.

[27] M. O. Reiser and M. Weber, Managing highly complex product families with multi-level feature trees, in *Proc. 14th IEEE International Requirements Engineering Conference*, 2006, pp. 146–155.