This paper will be presented at System Analysis and Modeling (SAM) Conference 2016 (http://sdl-forum.org/Events/SAM2016/index.htm)

Evaluating Variability Modeling Techniques for Supporting Cyber-Physical System Product Line Engineering

Safdar Aqeel Safdar [1], Tao Yue[1,2], Shaukat Ali[1], Hong Lu[1]

[1]Simula Research Laboratory, Oslo, Norway
[2] University of Oslo, Oslo, Norway
{safdar, tao, shaukat, honglu}@simula.no

Abstract. Modern society is increasingly dependent on Cyber-Physical Systems (CPSs) in diverse domains such as aerospace, energy and healthcare. Employing Product Line Engineering (PLE) in CPSs is cost-effective in terms of reducing production cost, and achieving high productivity of a CPS development process as well as higher quality of produced CPSs. To apply CPS PLE in practice, one needs to first select an appropriate variability modeling technique (VMT), with which variabilities of a CPS Product Line (PL) can be specified. In this paper, we proposed a set of basic and CPS-specific variation point (VP) types and modeling requirements for proposing CPS-specific VMTs. Based on the proposed set of VP types (basic and CPS-specific) and modeling requirements, we evaluated four VMTs: Feature Modeling, Cardinality Based Feature Modeling, Common Variability Language, and SimPL (a variability modeling technique dedicated to CPS PLE), with a real-world case study. Evaluation results show that none of the selected VMTs can capture all the basic and CPS-specific VP and meet all the modeling requirements. Therefore, there is a need to extend existing techniques or propose new ones to satisfy all the requirements.

Keywords: Product Line Engineering, Variability Modeling, and Cyber-Physical Systems

1 Introduction

Cyber-Physical Systems (CPSs) integrate computation and physical processes and their embedded computers and networks monitor and control physical processes by often relying on closed feedback loops [1, 2]. Nowadays, CPSs can be found in many different domains such as energy, maritime and healthcare. Many CPS producers employ the Product Line Engineering (PLE) practice, aiming to improve the overall quality of produced CPSs and the productivity of their CPS development processes [3].

In [4], a systematic domain analysis of the CPS PLE industrial practice is presented, which focuses on capturing static variabilities and facilitating product configuration at the pre-deployment phase. The systematic domain analysis identifies the following key characteristics of CPS PLE: (1) CPSs are heterogeneous and hierarchical systems; (2) the hardware topology can vary from one product to another; (3) the generic software code base might be instantiated differently for each product, mainly based on the hardware topology configuration; and (4) there are many dependencies among configurable parameters, especially across the software code base and the hardware topology. Various challenges in CPS PLE were also reported in [4] such as lacking of automation and guidance and expensive debugging of configuration data. In general, cost-effectively supporting CPS PLE, especially enabling automation of product configuration, is an industrial challenge.

Cost-effectiveness of PLE is characterized by its support for abstraction and automation. Generally speaking, abstraction is a key mean that enables reuse. Concise and expressive abstractions for CPS PLE are required to specify reusable artifacts at a suitable level of abstraction as commonalities and variabilities. Such abstractions are quite critical and provide the foundation for automation. To capture variabilities at a high level of abstraction, a number of variability modeling techniques (VMTs) are available in the literature, including Feature Modeling (FM) [5], Cardinality Based Feature Modeling (CBFM) [6], a UML-based variability modeling methodology named SimPL [7], and Common Variability Language (CVL) [8]. These VMTs were proposed for a particular context/domain/purpose. For example, SimPL was designed for the architecture level variability modeling. It is however no evidence showing which VMT suits CPS PLE the best.

In this paper, we propose a set of basic variation point (VP) types, CPS-specific VP types, and modeling requirements of CPS PLE. To define basic VP types, we constructed a conceptual model for basic data types in mathematics. Corresponding to each basic data type, we defined one basic VP type (Section 4.1). We also constructed a conceptual model for CPS based on the knowledge gathered from literature about CPSs and our experience of working with industry [4]. The second and third authors of the paper have experience of working with industrial CPS case studies and have derived the conceptual model. From the CPS conceptual model, we systematically derived a set of CPS-specific VP types (Section 4.2). We also derived a set of modeling requirements based on the literature and our experience in working with industry [4] (Section 5). Based on the proposed basic and CPS-specific VP types and the modeling requirements, we evaluated FM [5], CBFM [6] , CVL [8], and SimPL [7]. FM was selected as it is the most widely used VMT in industry [9] and CBFM is an extension of FM. CVL is a language for modeling variability using any domain specific language based on Meta Object Facility (MOF), which was submitted to Object Management Group for standardization but did not go through due to Intellectual Property Rights issues. SimPL is a specific VMT dedicated for CPS PLE and has been applied to address industrial challenges. To evaluate the VMTs, we modeled a case study (Material Handling System-MHS) with all the VMTs and evaluated them using the proposed eight basic and 16 CPS-specific VP types, and nine modeling requirements.

Results of the evaluation show that 1) only SimPL and CVL can capture all the basic VP types, whereas FM and CBFM provide partial support. None of the four VMTs can capture all the CPS-specific VP types; 2) SimPL and CVL provide support for 81% and 75% of the total CPS-specific VP types respectively, whereas CBFM supports 50% and FM supports only 15% of the total CPS-specific VP types; 3)

SimPL satisfies all but one of the modeling requirements, FM and CBFM only covers one modeling requirement, and CVL fully or partially fulfills four requirements out of nine requirements. Based on above results, we can conclude that it is required to either extend an existing technique or propose a new one to facilitate the variability modeling in the context of CPS PLE. The proposed VP types and modeling requirements can be also used as evaluation criteria for selecting existing VMTs or defining new ones for a particular application when necessary.

The rest of the paper is organized as follows: Section 2 presents the related work. Section 3 presents the context of the work. Section 4 presents the proposed VP types. Section 5 presents the modeling requirements. In Section 6, we report evaluation results. Threats to validity are given in Section 7. Section 8 concludes the paper.

2 Related Work

This section discusses the existing literature that compares or classifies VMTs, systematic literature reviews (SLRs) and surveys of VMTs.

Galster et al. [10] conducted a SLR of 196 papers published during 2000-2011, on variability management in different phases of software systems. Results show that most of the papers focus on design time variabilities and a small portion of the papers focus on runtime variabilities. In [11], Chen et al. conducted a SLR of 33 VMTs in software product lines and highlighted the challenges involved in variability modeling such as evolution of variability, and configuration. Arrieta et al. [12] conducted a SLR of variability management techniques, but limited their scope to techniques for Simulink published after 2008. Berger et al. [9] conducted a survey on industry practices of variability modeling using a questionnaire, aiming to discover characteristics of industrial variability models, VMTs, tools and processes. Another industrial survey of feature-based requirement VMTs was conducted to find out the most appropriate technique for a company [13]. They evaluated existing techniques based on requirements collected from the company's engineers, including readability, simplicity and expressive, types of variability and standardization.

Eichelberger and Schmid [14] classified and compared 10 textual VMTs in terms of scalability. They compared the selected techniques in five different aspects: configurable elements, constraints support, configuration support, scalability, and additional language characteristics. Similarly, Sinnema and Deelstra [15] classified six VMTs and compared them based on key characteristics of VMTs such as constraints, tool support, and configuration guidance. Czarnecki et al. [16] reported an experience report, in which they compared two types of VMTs: decision modeling and feature modeling. They compared them in 10 aspects: application, hierarchy, unit of variability, data types, constraints, modularity, orthogonality, mapping to artifacts, tool support, and binding time and mode. A comparative study [17] was reported to compare two VMTs, i.e., Kconfig and CDL, in the context of operating systems, in terms of constructs, semantics, and tool support.

All the above studies classify and evaluate various types of VMTs either in general or for a particular domain other than CPSs. We however, in this paper, propose a set of basic and CPS-specific VP types as well as a list of modeling requirements for evaluating VMTs in the context of CPS PLE, based on which we evaluated four representative VMTs with a non-trivial case study.

3 Context

Section 3.1 and 3.2 introduce the case study and the four VMTs. In Section 3.3, we present the study procedure.

3.1 Case Study

The case study is a product line of Handling Systems, which consist of various types of sub-systems such as Automatic Storage Retrieval System (ASRS), Automatic Guided Vehicle (AGV), Automatic Identification and Data Collection (AIDC) and Warehouse Management System. We selected three of these systems: AGV, AIDC, and ASRS for the evaluation of the selected VMTs. AGV is a fully automatic transport system that uses unmanned vehicles to transport all types of loads without human intervention. It is typically used within warehouse, production and logistics for safe movement of goods. AIDC is used to identify, verify, record, and track the products. Typically, these systems are used in supply chain, order picking, order fulfillment, and determination of weight, volume, and storage. ASRS is an automated system for inventory management, which is used to place and retrieve the loads from pre-defined locations in the warehouse. The descriptive statistics of the MHS case study's class diagram are given in Table 1. We modeled the case study (MHS) using the four selected VMTs (i.e., FM, CBFM, SimPL, and CVL). The case study models corresponding to selected VMTs are available at [18].

Table 1. Descriptive statistics of the MHS

Element	Count
Class	132
Generalization	56
Composition	62
Association	69
Simple attribute	113
Enumerated attribute	82
Enumeration	23
Enumeration Literal	73

3.2 Variability Modeling Techniques

Feature Modeling (FM) is widely applied in practice [9]. A feature model is organized hierarchically as a tree. The root node of the tree represents the system, whereas the descendent nodes are functionalities of the system (features). A feature can be mandatory, optional or alternative. A feature can either be a compound feature that has one or more descendent features or a leaf feature with no descendent features. Fig. 1 shows an excerpt of the FM model for AGV modeled using Pure::Variants [19]. As shown in Fig. 1, *AGVHardware, Sensor,* and *Connectivity* are mandatory features. The *Connectivity* feature has three alternative features, i.e., *Bluetooth*, *Wifi*, and *NFC.* The *Sensor* feature has two optional features: *MultiRayLEDScanner and LaserScanner*.

Fig. 1. An excerpt of FM for AGV

Cardinality Based Feature Modeling (CBFM) is an extension to FM, which introduces new concepts such as Feature Cardinalities, Groups and Groups

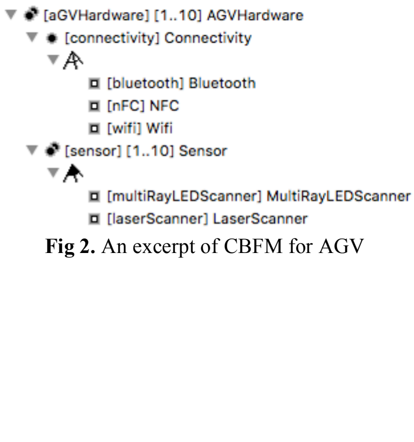

Fig 2. An excerpt of CBFM for AGV

Cardinalities, Attributes, and References. For Feature Cardinalities, features can be annotated with cardinalities such as <1..*> whereas alternative features and optional features are special cases with cardinality <1..1> and <0..1> respectively. A feature group can be or-group with cardinality <1..k> or alternative-group with cardinality <1..1>. For an alternative-group, one can select only one feature, whereas for or-group, one can select 1 to *k* number of features where *k* is the maximum number of features in the group. A feature can have one attribute of either String or Integer type. To achieve better modularization, a special leaf node (i.e., Reference) was introduced to refer to another feature model. This can be used to divide a large feature model into smaller ones to support modularization. As shown in Fig. 1 *AGVHardware, Sensor*, and *Connectivity* are mandatory features. *AGVHardware* and *Sensor* have feature cardinality <1..10>. *Connectivity* has an alternative-group that consists of three features: *Bluetooth*, *Wifi*, and *NFC*. The *Sensor* feature has an or-group consisting of two features with group cardinality <0..2>.

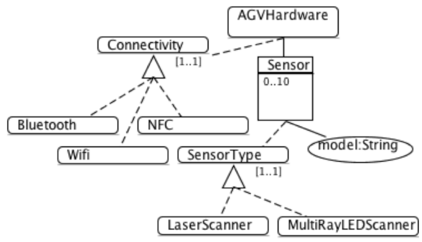

Fig. 3. An excerpt of CVL for AGV

Common Variability Modeling (CVL) is a generic variability modeling language and is composed of three interrelated models: base model, variability model, and resolution model. The base model can be defined in UML or any MOF based Domain Specific Language (DSL). Corresponding to the base model a variability model is defined. The variability model has a tree structure to specify variabilities. The resolution model specifies configurations of variabilities corresponding to a particular product. To support CVL, an Eclipse-based plugin CT-CVL is available [20]. In Fig. 3, rounded rectangles (e.g., *AGVHardware, SensorType, Connectivity)* represent *Choice* elements and a rectangle (e.g., *Sensor*) represents a *VClassifier* element whereas an ellipse represents a variable. Multiplicity inside the *VClassifier Sensor* (0..10) indicates that the number of instances of sensors can be between zero to 10 where for each instance one needs to configure sensor type and model. *Connectivity* and *SensorType* are *ChoiceVP* with group cardinality (1..1), which means only one option can be selected from given alternative options.

SimPL is a UML based VMT, which provides notations and guidelines for modeling variabilities and commonalities of CPS product lines at the architecture and design level. To support SimPL, several modeling tools [21] (RSA, MagicDraw, and Papyrus) are available. It captures four types of VPs: Attribute-VP, Type-VP, Topology-VP, and Cardinality-VP. A SimPL product line model can be specified with a subset of UML structural elements and stereotypes defined in the SimPL profile. Constraints are specified in the Object Constraint Language (OCL). SimPL has two major views: SystemDesignView and VariabilityView. SystemDesignView is composed of HardwareView, SoftwareView, and AllocationView to represent hardware components, software components and their relationship. VariabilityView is for capturing and structuring variabilities using UML

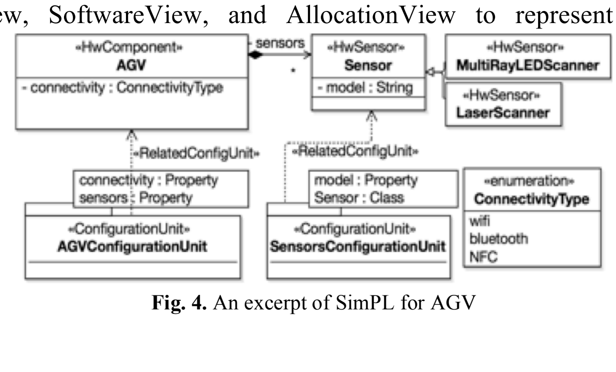

Fig. 4. An excerpt of SimPL for AGV

packages and template parameters. Stereotype «ConfigurationUnit» is applied on UML packages to group relevant variabilities. Variabilities are defined as template parameters of a package template and can trace back to hardware or software elements in the SystemDesignView. Fig. 4 presents an excerpt of the *HardwareView* of MHS, in which *AGV* is a hardware component composed of zero to many *Sensors*. *Sensor* can be of two types: *LaserScanner and MultiRayLEDScanner*. *AGV* has one Attribute-VP (*connectivity*) and one Cardinality-VP (*sensors*) denoting the number of instances of *Sensor*. For *Sensor*, two variabilities are specified: model (Attribute-VP) and type of sensor (Type-VP). *AGVConfigurationUnit* and *SensorsConfigurationUnit* are the template packages that are used to organize the variabilities corresponding to hardware component *AGV* and hardware *Sensor* respectively.

3.3 Procedure of the Study

Fig. 5 describes the procedure that we followed to conduct the study. First, we constructed a conceptual model for defining data types in mathematics and then we validated the data types with MARTE [22] and SysML [23], as these two standards are often used for modeling embedded systems and therefore can be used for modeling CPSs. In the third step, we defined a set of basic VP types (Section 4.1), based on the mathematical basic data types. We used basic data types for defining the basic VP types, as configuring a VP always requires assigning/selecting a value to/for a basic type variable. In the fourth step, we derived a set of modeling requirements (Section 5) based on knowledge collected from the literature and our experience of conducting industry-oriented research in the field of CPS PLE [4]. In the fifth step, we constructed a conceptual model for CPS, which is used to systematically derive the CPS-specific VP types (Step 6, more details in Section 4.2). In Step 7, we modeled the MHS case study with the selected VMTs, followed by the evaluation of the selected VMTs (Step 8, details in Section 6), based on the basic VP types, CPS-specific VP types, and the set of modeling requirements.

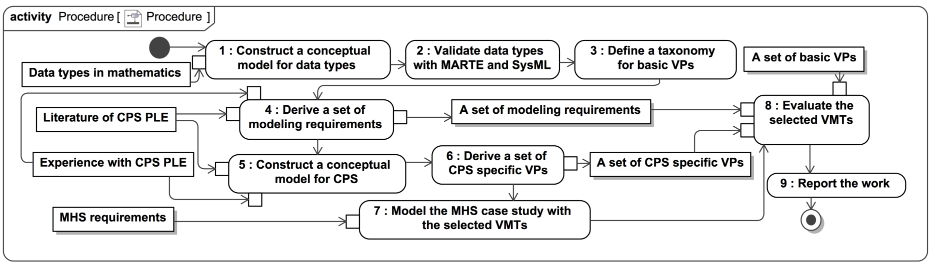

Fig. 5. Procedure of the study

4 Basic and CPS-specific Variation Point Types

4.1 Basic Variation Point Types

Based on the basic data types in mathematics, we constructed a conceptual model to classify them, as shown in Fig. 6. A *Variable* can be a *VariationPoint* or a *Non-configurableVariable*, which represents the configurable and non-configurable variable in CPS PLE. Each *Variable* has a *Type*, which is classified into two categories: *Atomic* (taking a single value at a given point of time) and *Composite* (composed of more than one atomic type, where each atomic type variable takes exactly one value at a given point in time). Atomic types are further classified into *Quantitative* types (taking numeric values) and *Qualitative* types (taking non-numeric values). *Quantitative* types can be *Discrete* (taking countable values) or *Continuous* (taking uncountable values). *Integer* is the concrete *Discrete* type, whereas *Real* is the concrete *Continuous* type. *Qualitative* types are categorized into *String*, *Binary* and *Categorical* that is further classified into *Ordinal* and *Nominal*.

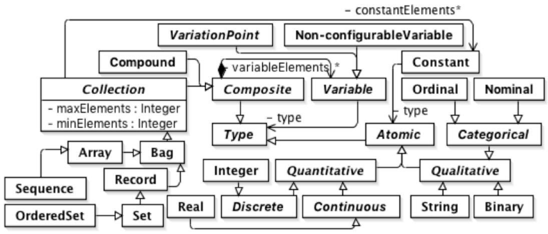

Fig. 6. Basic data types

A *Composite* data type combines several variables and/or constants, which is classified as: *Compound* and *Collection*. *Compound* takes only variables (e.g., complex numbers in SysML containing two variables realPart and imaginaryPart [23]) whereas *Collection* takes *Variables* and/or *Constants* (e.g., collection of colors). Attributes *minElements* and *maxElements* of *Collection* specify the minimum and maximum numbers of elements in a collection. As shown in Fig. 6, we have classified *Collection* into six types (i.e., *Bag*, *Array*, *Record*, *Set*, *OrderedSet* and *Sequence*) based on three properties: homogeneity, uniqueness and order. The homogeneity, uniqueness, and order properties of each collection type are specified as OCL constraints (Appendix A). Table 2 summarizes the six types of *Collection* along with their properties.

Table 2. Collection types

Collection	Hom.	Uni.	Ord.
Bag	No	No	No
Record	No	Yes	No
Set	Yes	Yes	No
OrderedSet	Yes	Yes	Yes
Array	Yes	No	No
Sequence	Yes	No	Yes

To validate the conceptual model of the basic data types, we mapped the data types defined in the MARTE Value Specification Language-VSL [22] and SysML [23] to the basic data types presented in Fig. 6. We used MARTE and SysML for validation because these two modeling languages can be used for modeling CPSs [24, 25]. During the validation, we do not include the extended data types provided in MARTE, as they are defined by extending the data types used in our mapping. In case of SysML we include all the data types. Results of the mapping are presented in Table 3, from which one can see that each data type in MARTE and SysML has a correspondence in our basic data type classification, which suggests that our classification of the basic data types is complete.

Table 3. Mapping MARTE and SysML data types to the basic data types

MARTE	SysML	Basic data types
Integer	Integer	Integer
UnlimitedNatural	UnlimitedNatural	Integer
Boolean	Boolean	Binary
String	String	String
Real	Real	Real
DateTime	Complex	Compound
EnumerationType	Enumeration	Ordinal/Nominal
	ControlValue	Nominal/Ordinal
IntervalType	UnitAndQuantityKind	Compound
TupleType		Compound
ChoiceType		Compound
CollectionType		Collection

In Fig. 7, we present a classification of basic VP types where one basic VP type is defined corresponding to each basic data type presented in Fig. 6. A *VariationPoint* can be a *CompositeVP* or an *AtomicVP*. An *AtomicVP* can come with any of the six concrete types: *StringVP*, *BinaryVP*, *NominalVP*, *OrdinalVP, IntegerVP, and RealVP* corresponding to *String*, *Binary*, *Nominal*, *Ordinal*, *Integer*, and *Real* respectively. A *CompositeVP* can be *CompoundVP* or *CollectionVP*, which are defined corresponding to *Compound* and *Collection* data types respectively. As shown in Fig. 7, a *CompositeVP* may have several *AtomicVP*s and/or *CompositeVP*s depending on the number of *variableElements* (Fig. 6) involved in the *Composite* data type. *CollectionVP* may have two additional *IntegerVP*(s), i.e., *lowerLimitVP* and *upperLimitVP* corresponding to the minimum and maximum numbers of the elements in the collection.

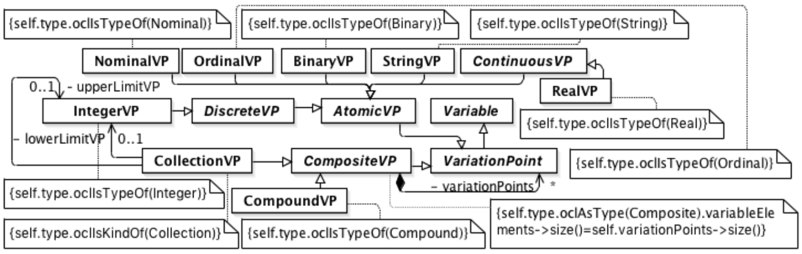

Fig. 7. Classification of the basic VP types

4.2 CPS-specific Variation Point Types

In this section, first we present a conceptual model for CPS (Fig. 8), based on which we then derive a set of CPS-specific VP types (Table 4). As shown in Fig. 8, a CPS can be defined as a set of physical components (e.g., human heart, engine), interfacing components (e.g., sensor, actuator, network), and cyber components (with deployed software), which are integrated together to accomplish a common goal.

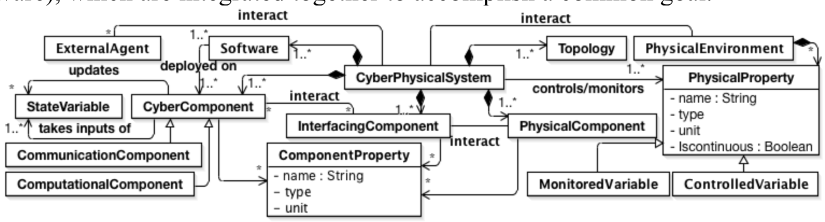

Fig. 8. A CPS conceptual model

A CPS can have one or more topologies, which define how various components are integrated. A CPS controls and monitors a set of physical properties. A *CyberComponent* can either be a *CommunicationComponent* or *ComputationalComponent*, which takes values of *StateVariables* as input and updates their values when needed. Each component in CPS has several component properties. CPS may interact with *PhysicalEnvironment* and *ExternalAgent*s (e.g., external systems). Both *PhysicalProperty* and *ComponentProperty* have attributes *name*, *type*, and *unit* to specify the name, type (e.g., descriptive, numeric, Boolean), and unit of a specific property. *PhysicalProperty* has an extra Boolean attribute *isContinuous* to specify either it is a continuous or a discrete type of property.

In Table 4, the first column represents the CPS concepts used to derive CPS-specific VP types and the second column shows the derived CPS-specific VP types. The last column presents the basic VP type corresponding to a particular CPS-specific VP type.

Table 4. CPS-specific VP types

CPS Concept	CPS-Specific VP Type	Basic VP Type
CP	Descriptive-VP	StringVP
CP, PP	DiscreteMeasurement-VP	IntegerVP
CP, PP	ContinuousMeasurement-VP	RealVP
CP, PP	BinaryChoice-VP	BinaryVP
CP, PP	PropertyChoice-VP	NominalVP/OrdinalVP
CP, PP	MeasurementUnitChoice-VP	OrdinalVP
CP, PP	MeasurementPrecision-VP	RealVP
CP, PP, COM	Multipart/Compound-VP	CompoundVP
COM	ComponentCardinality-VP	IntegerVP
COM	ComponentCollectionBoundary-VP	IntegerVP
COM	ComponentChoice-VP	NominalVP/OrdinalVP
COM	ComponentSelection-VP	CollectionVP
Topology	TopologyChoice-VP	NominalVP
Deployment	AllocationChoice-VP	NominalVP
Interact	InteractionChoice-VP	NominalVP
Constraint	ConstraintSelection-VP	CollectionVP

*CP=ComponentProperty, PP =PhysicalProperty, COM=Physical, Interfacing, or Physical Component

PhysicalProperty and **ComponentProperty:** Descriptive-VP, DiscreteMeasurement-VP, ContinuousMeasurement-VP, BinaryChoice-VP, PropertyChoice-VP, MeasurementUnitChoice-VP, and MeasurementPrecision-VP are defined for physical properties and/or component properties of CPS. Descriptive-VP is a *StringVP*, which requires setting a value in order to configure it. It can be defined for a textual *ComponentProperty* such as ID of a sensor. DiscreteMeasurement-VP and ContinuousMeasurement-VP are *IntegerVP* and *RealVP* respectively. Both these two types of VPs can be defined for numeric component properties (e.g., data transmission interval of a sensor) or physical properties (e.g., length and weight of a physical component) of CPS. BinaryChoice-VP is a *BinaryVP*, which can be defined for Boolean physical properties (e.g., the presence of a magnetic field) and component properties (e.g., whether a sensor keeps the events' log). PropertyChoice-VP is a *NominalVP* or an *OrdinalVP*, which requires selecting one value from a list of pre-defined values. For example, a *ComponentProperty* can be connectionType, which can be configured as wired, 3G, or Wi-Fi, which can be captured as a PropertyChoice-VP. MeasurementUnitChoice-VP is an *OrdinalVP*, which is derived from the *unit* of *PhysicalProperty* and *ComponentProperty*. For example, one can select meter, centimeter or millimeter as a unit for length (a *PhysicalProperty*). MeasurementPrecision-VP is a *RealVP*, which is related to the degree of measurement precision for a *PhysicalProperty* or *ComponentProperty*.

Component: ComponentCardinality-VP, ComponentCollectionBoundary-VP, ComponentChoice-VP, and ComponentSelection-VP are derived from the different types of CPS components: *CyberComponent*, *InterfacingComponent*, *PhysicalComponent*. ComponentCardinality-VP is an *IntegerVP*, which is related to varying number of instances of a CPS component (e.g., number of temperature sensors). ComponentCollectionBoundary-VP is an *IntegerVP*, which is related to the upper limit and/or the lower limit of a collection of CPS components. For example, the maximum and minimum numbers of sensors supported by a controller. ComponentChoice-VP is a *NominalVP*/*OrdinalVP*, which is about selecting a particular type of CPS component such as selecting a speedometer sensor from several speedometers with various specifications. ComponentSelection-VP is a *CollectionVP*, which is about selecting a subset of CPS components from a collection of CPS components such as selecting sensors for a product from available sensors.

Multipart/Compound-VP is a *CompoundVP*, which can be specified for a *PhysicalProperty*, *ComponentProperty*, or a component (Physical, Cyber, or Interfacing) that requires configuring several constituent VPs involved in it. As in the domain of CPS, it is common that different properties do not give complete meaning unless they are combined together. For example, length is a *PhysicalProperty*, which is meaningless without a unit. Hence, we need a Compound-VP type, which involves two VPs including length and its unit. A Compound-VP can also be defined for a component (e.g., sensor), which contains several other VPs defined for its properties.

Topology: TopologyChoice-VP is a *NominalVP*, which is related to selecting a topology from several alternatives. For example, how *CyberComponent* (e.g., controller) is connected with *InterfacingComponent*s (e.g., sensors and actuators).

Deployment: AllocationChoice-VP is a *NominalVP*, which is about the deployment of software on a *CyberComponent* (e.g., controller). For example, the same version of software can be deployed on different controllers or different versions of software can be deployed on the same controller.

Interaction: InteractionChoice-VP is a *NominalVP*, which is about the interaction (presented as association named interact in Fig. 8), of two CPS components (e.g., *CyberComponent* and *InterfacingComponent*) or interaction of CPS with an external agent, which can be for example an external system.

Constraint: ConstraintSelection-VP is a *CollectionVP*, which is about selecting a subset of constraints in order to support the configuration of a specific product, from a set of constraints defined for the corresponding CPS product line.

5 Modeling Requirements

In addition to capturing different types of VPs, a VMT should also accommodate some modeling requirements to enable automation of configuring CPS products. These requirements (Table 5) are derived from the literature and our experience of working with industry [4].

In Table 5, R_1 is related to support different binding times of a VP, as a VP can be configured at three different phases [26]: the pre-deployment phase, the deployment phase and the post-deployment phase. Requirements R_2 focuses on a traceability mechanism to link the variability model and its base whereas R_3 is related to realizing the separation of concerns principle in the product line model. R_4-R_8 are relevant to different types constraints that a VMT should be able to capture for enabling automation of the configuration process in CPS PLE [3]. In [3], a constraint classification was presented and we extended it by adding two more categories: inference and conformance. These constraints are needed to facilitate different functionalities of an interactive, multi-step and multi-staged configuration solution, such as consistency checking, decision inferences. R_9 is related to modeling different types of configurable elements of CPSs.

Table 5. Modeling requirements

ID	Name	Description
R_1	VP binding time	Support different binding times for a VP (e.g., pre-deployment, deployment, and post-deployment phases).
R_2	Linkage between VP and the base	Provide a mechanism to relate a VP to the corresponding base model element.
R_3	Separation of Concerns	Provide a mechanism to realize the principle of separation of concerns to enable multi-staged and cross-disciplinary configuration of CPS.
R_4	Variability dependency	Capture dependencies between a VP and a variant, two VPs, and two variants.
R_5	Ordering	Specify constraints on the order of configuration steps.
R_6	Inference	Specify constraints that can be used to configure VPs automatically.
R_7	Conformance	Specify conformance rules for ensuring the correctness of configuration data.
R_8	Consistency	Specify consistency rules for checking the consistency of the configuration data and variability models.
R_9	Multidisciplinary	Model *Software*, *PhysicalComponent*, *InterfacingComponent*, *CyberComponent*, and *PhysicalEnvironment* elements of CPS.

6 Evaluation

The purpose of the evaluation is to compare the selected four VMTs with the aim to help modelers to select an appropriate VMT or propose a new one if necessary for CPS PLE, which can capture different types of VPs (Section 4) and meet the modeling requirements (Section 5). Corresponding to this goal, we pose the following research questions: **RQ1**: To what extent can each selected VMT capture the basic VPs? **RQ2**: To what extent can each selected VMT capture the CPS-specific VPs? **RQ3**: To what extent does a selected VMT comply with the modeling requirements? We answer RQ1, RQ2 and RQ3 in Section 6.1, Section 6.2, and Section 6.3, respectively.

6.1 Evaluation Based on Basic VP Types (RQ1)

To answer RQ1, we evaluate the selected VMTs based on the basic VP types. In Table 6, the first column represents the basic VP type and the second column indicates if a basic VP type is required by the MHS case study, whereas columns 3-6 show how each selected VMT supports each basic VP type.

As one can see from Table 6, modeling the MHS case study requires all the basic VP types. However, FM supports only three out of eight basic VP types: *BinaryVP*, *NominalVP* and *OrdinalVP*. Optional feature and alternative-group with two features of FM map to *BinaryVP*s. In FM, alternative-group corresponds to *NominalVP*s and *OrdinalVP*s, but FM does not differentiate *NominalVP* from *OrdinalVP*. CBFM provides support for all the basic VP types except for *CompoundVP*. Corresponding to *RealVP*s and *StringVP*s, CBFM provides attributes (one attribute per feature) of Real and String respectively. However, for *IntegerVP*s, it offers feature and group cardinalities together with Integer attributes. For *BinaryVP*, CBFM has optional features, alternative-groups, feature cardinalities (0..1), and Boolean attributes. Similar to FM, CBFM also provides alternative-groups, which map to *NominalVP*s and *OrdinalVP*s and CBFM does not differentiate these two types. For *CollectionVP*, CBFM provides alternative-groups and or-groups.

Both SimPL and CVL support all the basic VP types. In SimPL, Attribute-VP defined with Real and String attributes map to *RealVP*s and *StringVP*s. *IntegerVP*s can map to Attribute-VPs defined on Integer attributes or Cardinality-VP. To support *BinaryVP*, SimPL provides Attribute-VP defined on attributes of the binary type, Cardinality-VP with two options, Type-VP with two types, and Topology-VP with two topologies. Cardinality-VP, Type-VP, and Topology-VP offered by SimPL can be mapped to *NominalVP*s and *OrdinalVP*s. SimPL does not differentiate *NominalVP* and *OrdinalVP*. To support *CompoundVP*, SimPL defines «ConfigurationUnit», which can be applied on packages, to organize a set of relevant VPs. In SimPL, *CollectionVP* corresponds to Cardinality-VP.

Table 6. Evaluation based on the basic VP types (RQ1)

Basic VP Type	MHS	VMT			
		FM	CBFM	SimPL	CVL
IntegerVP	Yes	No	One At/F, G & F Cardinality	Attribute-VP, Cardinality-VP	Multiplicity, ParametricVP
RealVP	Yes	No	One At/F	Attribute-VP	ParametricVP
StringVP	Yes	No	One At/F	Attribute-VP	ParametricVP
BinaryVP	Yes	OF, Alt. F	One At/F, OF, Alt. G, F-Cardinality	Attribute-VP, Cardinality-VP, Type-VP, Topology-VP	ChoiceVP (ObjectSubstitution, SlotAssignment, ObjectExistence, SlotValueExistence, LinkExistence), Multiplicity, ParametricSlotAssignment
NominalVP	Yes	Alt. G	Alt. G	Attribute-VP, Type-VP, Topology-VP	Group of SlotAssignment (i.e., ChoiceVP) with group Multiplicity (1,1), ParametricObjectSubstitution (i.e., ParametricVP).
OrdinalVP	Yes	Alt. G	Alt. G		
CompoundVP	Yes	No	No	Configuration Unit	CompositeVP, VClassifier with several Repeatable-VP(s).
CollectionVP	Yes	No	Alt. G, OR G	Cardinality-VP	VClassifier with configurable Multiplicity, group of SlotAssignment (i.e., ChoiceVP).

*F=feature, OF=optional feature, G=group, At=attribute, Alt=Alternative, /= per, &= and

To support *RealVP* and *StringVP*, CVL provides ParametricVP. For *IntegerVP* it provides ParametricVP and cardinalities. For *BinaryVP*, CVL has different types of ChoiceVPs (i.e., ObjectSubstitution, SlotAssignment, ObjectExistence, SlotValueExistence, and LinkExistence) along with multiplicity and ParametricSlotAssignment (i.e., ParametricVP). In CVL, both *NominalVP*s and *OrdinalVP*s can be mapped to SlotAssignments (i.e., ChoiceVP) with group multiplicity (1..1) or ParametricObjectSubstitution (i.e., ParametricVP). Similar to all the other VMTs, CVL does not differentiate *NominalVP* and *OrdinalVP*. In CVL, *CompoundVP* maps to CompositeVP and a VClassifier with several RepeatableVP(s) can also be used to model *CompoundVP*s. For *CollectionVP*, CVL has VClassifier with the multiplicity other than (1..1) and a group of SlotAssignment (i.e., ChoiceVP).

To summarize, both SimPL and CVL support all the basic VP types whereas FM and CBFM provide partial support. None of the selected four VMTs differentiate NominalVP and OrdinalVP.

6.2 Evaluation Based on the CPS-Specific VP Types (RQ2)

To answer RQ2, we evaluate the selected four VMTs based on the CPS-specific VP types (Section 4.2) and VPs modeled for the MHS case study. In Table 7, the first column represents the CPS-specific VP types and the second column indicates if a particular CPS-specific VP type is required by the MHS case study. Columns 3-6 are related to the four VMTs to signify if they support a particular CPS-specific basic VP type. The seventh column shows the number of VPs in the MHS case study corresponding to a particular CPS-specific VP type, whereas columns 8-11 show the number of VPs modeled using the four VMTs.

As one can see from Table 7, our case study (MHS) contains VPs corresponding to all the CPS-specific VP types. FM does not cater majority of the CPS-specific VP types and only supports fully or partially three out of 16 CPS-specific VP types: BinaryChoice-VP, PropertyChooice-VP, and ComponentChoice-VP.

CBFM supports six of 16 CPS-specific VP types: ComponentCardinality-VP, ComponentCollectionBoundary-VP, MeasurementPrecision-VP, PropertyChoice-VP, ComponentChoice-VP, and ComponentSelection-VP. It provides partial support for three CPS-specific VP types (i.e., Descriptive-VP, DiscreteMeasurement-VP, and ContinuousMeasurement-VP) because CBFM allows adding only one attribute for each feature. BinaryChoice-VP is also partially supported, as it can be captured using optional feature or cardinality but CBFM does not allows adding Boolean attribute. The remaining six CPS-specific VP types are not supported by CBFM.

Both SimPL and CVL support Descriptive-VP, DiscreteMeasurement-VP, ContinuousMeasurement-VP, ComponentSelection-VP, ComponentCardinality-VP, ComponentCollectionBoundary-VP, BinaryChoice-VP, MeasurementPrecision-VP, MeasurementUnitChoice-VP, PropertyChoice-VP, ComponentChoice-VP, and Compound-VP. SimPL also supports TopologyChoice-VPs, which cannot be captured using CVL. The remaining three CPS-specific VP types (i.e., AllocationChoice-VP, InteractionChoice-VP, and ConstraintSelection-VP) are not catered by either SimPL or CVL.

Table 7. Evaluation of VMTs based on the CPS-specific VP types and VPs (RQ2)

CPS-Specific VP Type	**VP Types Coverage**					**VP Coverage**				
	MHS	**FM**	**CBFM**	**SimPL**	**CVL**	**MHS**	**FM**	**CBFM**	**SimPL**	**CVL**
Descriptive-VP	Yes	No	Partial	Yes	Yes	34	0	4	34	34
DiscreteMeasurement-VP	Yes	No	Partial	Yes	Yes	23	0	5	23	23
ContinuousMeasurement-VP	Yes	No	Partial	Yes	Yes	51	0	18	51	51
ComponentCardinality-VP	Yes	No	Yes	Yes	Yes	42	0	42	42	42
ComponentCollectionBoundary-VP	Yes	No	Yes	Yes	Yes	42	0	42	42	42
MeasurementPrecision-VP	Yes	No	Yes	Yes	Yes	2	0	2	2	2
BinaryChoice-VP	Yes	Partial	Partial	Yes	Yes	3	0	0	3	3
PropertyChoice-VP	Yes	Yes	Yes	Yes	Yes	82	82	82	82	82
ComponentChoice-VP	Yes	Yes	Yes	Yes	Yes	12	12	12	12	12
TopologyChoice-VP	Yes	No	No	Yes	No	9	0	0	9	0
AllocationChoice-VP	Yes	No	No	No	No	3	0	0	0	0
InteractionChoice-VP	Yes	No	No	No	No	15	0	0	0	0
MeasurementUnitChoice-VP	Yes	No	No	Yes	Yes	59	0	18	59	59
ConstraintSelection-VP	Yes	No	No	No	No	1	0	0	0	0
ComponentSelection-VP	Yes	No	Yes	Yes	Yes	42	0	42	42	42
Multipart/Compound-VP	Yes	No	No	Yes	Yes	64	0	0	64	26
Total (count)	**16**	**2.5**	**8**	**13**	**12**	**484**	**94**	**267**	**465**	**418**
Coverage (%)	**100%**	**15%**	**50%**	**81%**	**75%**	**-**	**19%**	**55%**	**96%**	**86%**

As shown in Table 7, none of the selected VMTs supports all the CPS-specific VP types. SimPL supports 81%, FM supports only 15%, CVL caters 75%, and CBFM covers 50% of the total CPS-specific VP types. Using SimPL and CVL we were able to model 96% and 86%, whereas with FM and CBFM, we could model only 19% and 55% of total VPs in our case study.

6.3 Evaluation Based on the Modeling Requirements (RQ3)

Table 8 summarizes the results of our evaluation of the four VMTs in terms of modeling requirements (Section 5) with MHS. In Table 8, the first two columns are used to identify the requirements and the third column indicates if a requirement is required by MHS. Columns 4-7 signify if the VMTs support a particular requirement.

Table 8. Results for the evaluation of the VMTs based on the modeling requirements (RQ3)

ID	Name	MHS	FM	CBFM	CVL	SimPL
R_1	VP binding times	Yes	No	No	Yes	No
R_2	Linkage between VP and the base	Yes	No	No	Yes	Yes
R_3	Separation of Concerns	Yes	No	No	Partial	Yes
R_4	Variability dependencies	Yes	Partial	Partial	Partial	Yes
R_5	Ordering	Yes	No	No	Depends on base modeling language	Yes
R_6	Inference	Yes	No	No		Yes
R_7	Conformance	Yes	No	No		Yes
R_8	Consistency	Yes	No	No		Yes
R_9	Multidisciplinary	Yes	No	No		Partial

None of the selected VMTs except for CVL allows specifying the binding time (R_1) of a VP to enable its configuration in different phases. CVL and SimPL support linking a VP to the corresponding base model element explicitly (R_2), which is however not supported by FM and CBFM, as they do not have separate base models. FM and CBFM do not support the separation of concerns (R_3) and CVL supports partially as it models variabilities separately from the base model. SimPL supports R_3 as it provides hardware, software and allocation views in addition to the variability view. For MHS, we captured all the four views defined in SimPL. But, it still requires a view for specifying environment elements and corresponding VPs.

R_4-R_8 are related to capturing different types of constraints to enable automation in CPS PLE. FM and CBFM provide partial support for capturing variability dependencies such as requires and excludes, but they are unable to capture other complex constraints such as consistency rules. In the case of CVL, it uses the Basic Constraint Language [8] for capturing simple propositional and arithmetic constraints but it is unable to capture all the types of constraints discussed in Section 5. If the base model is modeled in UML, then OCL can be integrated with CVL, thereby allowing the specification of all the types of constraints. SimPL is based on UML and OCL, which makes it possible to capture all the types of constraints.

MHS is a multidisciplinary system, which contains *Software*, *CyberComponent*, and different types of *PhysicalComponent* and *InterfacingComponent* interacting with *PhysicalEnvironment* but none of the selected VMTs explicitly model these multidisciplinary elements of CPS (R_9). SimPL supports all, except for *PhysicalEnvironment* elements. In case of CVL, it depends on the DSL used for modeling the base model, which may or may not have the capability of modeling different elements of CPS.

7 Threats to validity

One threat to validity of our study is the selection of the VMTs. Since it is not practically feasible to evaluate all existing VMTs, we therefore selected four representative VMTs. Another threat to validity is the completeness of the basic and CPS-specific VP types and modeling requirements. Note that our approach for deriving the basic VP types is systematic, which to certain extent ensures their completeness. In addition, we validated them using SysML and MARTE, which are two existing standards often used for embedded system modeling. We derived CPS-specific VP types based on thorough domain analyses and our experience in working with industry. We also verified that the MHS case study covers all the CPS-specific VP types.

8 Conclusion

In this paper, we present a set of basic and CPS-specific VP types that need to be supported by a VMT in the context of CPS PLE. Moreover, we present a set of modeling requirements, which need to be catered to enable the automation of configuration in CPS PLE. Based on the proposed basic and CPS-specific VP types and modeling requirements, we evaluated four VMTs: feature model, cardinality based feature model, CVL, and SimPL, with a real-world case study. Results of our evaluation show that the selected four VMTs cannot capture all the VP types and none of the four VMTs meets all the requirements. This necessitates the extension of an existing technique or proposal of a new one to facilitate CPS PLE. The proposed VP types and modeling requirements can be used as evaluation criteria to select a suitable VMT or develop a new one if necessary.

Acknowledgement. This work was supported by the Zen-Configurator project funded by the Research Council of Norway (grant no. 240024/F20) under the category of Young Research Talents of the FRIPO funding scheme. Tao Yue and Shaukat Ali are also supported by the EU Horizon 2020 project U-Test (http://www. u-test.eu/) (grant no. 645463), the RFF Hovedstaden funded MBE-CR (grant no. 239063) project, the Research Council of Norway funded MBT4CPS (grant no. 240013/O70) project, and the Research Council of Norway funded Certus SFI (grant no. 203461/O30).

References

1. http://cyberphysicalsystems.org/
2. Rawat, D.B., Rodrigues, J.J., Stojmenovic, I.: Cyber-Physical Systems: From Theory to Practice. CRC Press (2015)
3. Nie, K., Yue, T., Ali, S., Zhang, L., Fan, Z.: Constraints: the core of supporting automated product configuration of cyber-physical systems. Model-Driven Engineering Languages and Systems, pp. 370-387. Springer (2013)
4. Yue, T., Ali, S., Selic, B.: Cyber-physical system product line engineering: comprehensive domain analysis and experience report. In: Proceedings of the 19th International Conference on Software Product Line, pp. 338-347. ACM, (2015)
5. Kang, K., Cohen, Sholom., Hess, James., Novak, William., & Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study (CMU/SEI-90-TR-021). Carnegie Mellon University (1990)
6. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged configuration using feature models. Software Product Lines, pp. 266-283. Springer (2004)

7. Behjati, R., Yue, T., Briand, L., Selic, B.: SimPL: a product-line modeling methodology for families of integrated control systems. Information and Software Technology (2013)
8. Haugen, O.: Common Variability Language (CVL). OMG Revised Submission (2012)
9. Berger, T., Rublack, R., Nair, D., Atlee, J.M., Becker, M., Czarnecki, K., Wąsowski, A.: A survey of variability modeling in industrial practice. In: Proceedings of 7th International Workshop on Variability Modelling of Software intensive Systems, pp. 7. ACM, (2013)
10. Galster, M., Weyns, D., Tofan, D., Michalik, B., Avgeriou, P.: Variability in software systems-A systematic literature review. IEEE Transactions on Software Engineering, 40, 282-306 (2014)
11. Chen, L., Ali Babar, M., Ali, N.: Variability management in software product lines: A systematic review. 13th International Software Product Line Conference, pp. 81-90 (2009)
12. Arrieta, A., Sagardui, G., Etxeberria, L.: A comparative on variability modelling and management approach in simulink for embedded systems. V Jornadas de Computación Empotrada, ser. JCE (2014)
13. Djebbi, O., Salinesi, C.: Criteria for comparing requirements variability modeling notations for product lines. In: 4th International Workshop on Comparative Evaluation in Requirements Engineering, pp. 20-35. IEEE, (2006)
14. Eichelberger, H., Schmid, K.: A systematic analysis of textual variability modeling languages. Software Product Line Conference, pp. 12-21. ACM (2013)
15. Sinnema, M., Deelstra, S.: Classifying variability modeling techniques. Information and Software Technology 49, 717-739 (2007)
16. Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., Wąsowski, A.: Cool features and tough decisions: a comparison of variability modeling approaches. In: 6th international workshop on variability modeling of software intensive systems, pp. 173-182. ACM,(2012)
17. Berger, T., She, S., Lotufo, R., Wąsowski, A., Czarnecki, K.: Variability modeling in the real: a perspective from the operating systems domain. International conference on Automated software engineering, pp. 73-82. ACM (2010)
18. http://www.zen-tools.com/SAM2016.html
19. http://www.pure-systems.com
20. http://modelbased.net/tools/ct-cvl/
21. Safdar, S.A., Iqbal, M.Z., Khan, M.U.: Empirical Evaluation of UML Modeling Tools–A Controlled Experiment. European Conference on Modeling Foundations and Applications, vol. 11, pp. 33-44. Springer, Italy (2015)
22. The UML MARTE profile, http://www.omgmarte.org/.
23. OMG: Systems Modeling Language (SysML) v1.4, http://sysml.org/. (2015)
24. Selic, B., Gérard, S.: Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems. Elsevier (2013)
25. Derler, P., Lee, E.A., Vincentelli, A.S.: Modeling cyber–physical systems. Proceedings of the IEEE 100, 13-28 (2012)
26. Murguzur, A., Capilla, R., Trujillo, S., Ortiz, Ó., Lopez-Herrejon, R.E.: Context variability modeling for runtime configuration of service-based dynamic software product lines. In: Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools, pp. 2-9. ACM, (2014)

Appendix A: OCL Constraints

Homogeneity: context Array, Set (Sequence, OrderedSet)(**self**.constantElements->size()=0 **and** **self**.variableElements->select(a|a.**oclIsKindOf**(Collection))->size()=0 **and self**.variableElements->forAll(a,b| a.type=b.type))**or** (**self**.variableElements->size()=0 **and self**.constantElements->forAll(a,b| a.type=b.type)) **or** (**self**.constantElements->size()=0 **and** **self**.variableElements->size()=**self**.variableElements->select(a:Variable|a.type.**oclIs KindOf**(Collection))->size() **and** **self**.variableElements->forAll(v1, v2|(v1.type.**oclAsType**(Collection).constant Elements->size()=0 **and** v1.type.**oclAsType**(Collection).variableElements->forAll(v3:Variable | v3.type = v2.type.**oclAsType**(Collection).variableElements->asSequence()->first().type)) **or** (v1.type.**oclAsType**(Collection).variableElements->size()=0 **and** v1.type.**oclAs Type**(Collection).constantElements->forAll(v3:Constant| v3.type=v2.type.**oclAsType** (Collection).constantElements->asSequence()->first().type))))	**Uniqueness: context** Record (Set, OrderedSet) **self**.variableElements->select (**self**.variableElements ->forAll(a,b| a=b))->isEmpty() **and** **self**.constant Elements->select (**self**.constantElements->forAll(a,b| a=b))->isEmpty()
	Order: context Sequence **self**.variableElements->asSet()->size() >1 **implies self**.variableElements->asSequence()->reverse() <> **self**.variableElements->asSequence() **and** **self**.constantElements->asSet()->size() >1 **implies** **self**.constantElements->asSequence()->reverse() <> **self**.constantElements->asSequence() **context** OrderedSet **self**.variableElements->asOrderedSet()->reverse() <> **self**.variableElements->asOrderedSet() **and** **self**.constantElements->asOrderedSet()->reverse() <> **self**.constantElements->asOrderedSet()