SPES XT

Software Plattform Embedded Systems 2020

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

The Variability Exchange Language

Version 1.0

Project	SPES_XT	
Responsible	Daimler AG	
QA-Responsible		
Authors	Martin Große-Rhode, Fraunhofer FOKUS Michael Himsolt, Daimler AG Michael Schulze, pure·systems	
Created	16 May 2013	
Last Changed	17 December 2015	
Confidential Level		Confidential for Partners: <Partner1>; <Partner2>; …
		Open to Project
	X	Public
Document Status		In Progress
	X	Provided
		Released

List of Changes

Change			Changed Chapter	Description of Change	Author	State
Nr.	Date	Version				
1	16 May 2013	0.5	All	Copied from previous document		
2	January 2015	0.9	All	Prepared for review		
3	5 March 2015	1.0	All	Prepared for inclusion in document		

Table of Contents

List of Figures

List of Listings

1 Abstract

The purpose of the *Variability Exchange Language* is to support the information exchange among variants management tools on the one hand and systems development tools on the other hand. The essential tasks of a variants management tool are to represent and analyze the variability of a system abstractly and to define system configurations by selecting the desired system features. A system development tool captures information of a specific kind, such as requirements, architecture, component design, or tests. In order to support the development of variable systems a development tool either has to offer the capability to express and deal with variability directly, or an adaptor must be provided that adds this capability to the development tool.

To interconnect variants management with systems development the information exchange among the corresponding tools must be established. A variants management tool must be able to read or extract the variability from a development tool and to pass a configuration, i.e. a set of selected system features, to the development tool. Up to now the interfaces that support this information exchange are built for each development tool anew. With a standardized *Variability Exchange Language* a common interface can be defined that is implemented by the development tools and used by the variants management tools. The integration of variants management tools with systems development tools via this interface enables a continuous development process for variable systems and supports a flexible usage of tools for this process.

2 Introduction

2.1 Variants Management, System Variability, and Variation Points

Variants management is an activity that accompanies the whole system development process and, therefore, is orthogonal to the other development tasks. Like safety, security, and other system properties, variability cannot be built into a system at the end of the process. Rather, the desired variability has to be determined, analyzed, designed, implemented and tested continuously, starting at the very beginning of the process through to the final delivery of the system or the system variant respectively. That means that within each development stage – requirements analysis, design, implementation, test, documentation, etc. – variability is an aspect that has to be considered.

We consider as variable system a system that can be tailored by the system producer according to individual clients' needs. All variants of a variable system are developed within one development process. In addition to the standard development tasks the process must also provide the means to tailor the system, i.e. to derive the client specific variant of the system. This may happen at different stages, also known as (variance) binding times.

Variability is embodied in variation points. Consider as example a requirements document. A requirement toward a variable system may be *optional*. In this case two system variants can be formed by either selecting or deselecting the requirement. A set of requirements may be *alternatives*, then each selection of one of these requirements forms one system variant. Finally a requirement may contain a *parameter*, then each value that can be selected for this parameter yields a system variant.

The same definition of variation points holds for all other artifacts that are created in the development process – be it analysis or design models such as the views defined in the SPES-XT meta model, test specifications, code, documentation, or whatever. In each artifact there may be optional elements, alternative elements, and parameterized elements.

We do not specify here how these variation points are represented in the artifacts. Some artifact formats support the definition of variation points, in other cases appropriate means have to be added. This obviously also has an impact on the tools that are used to create and manage the artifacts. In some cases they are capable to express variation points. In other cases adaptors have to be built in order to incorporate variation points.

2.2 Variability View and Variants Management Tools

It is an accepted best practice to define an explicit abstract variability view on a system under development to support variants management continuously throughout the process. This abstraction contains the bare information on the variability of the system. That means that it describes which variants exist, but does not describe how the variability is realized. The variability information is derived from an analysis of the commonalities, differences, and dependencies of the system's variants and is often represented as a feature model.

A variants management tool supports the creation of an artifact – a variability model – that represents the abstract variability information. Moreover, it offers operations to select or deselect system features and via this feature configuration to specify the system's variants.

The information of the variability view has to be connected with the system development artifacts in order to define how the feature selection (system configuration) determines the resolution of the variation points within these artifacts, i.e. the selection of a variation for each variation point. As soon as these connections are established a feature configuration can be carried over to a configuration of the variation points of the concerned artifact. The technical realization of this connection is addressed by the *Variability Exchange Language*.

At present there is no standard that would define how variation points are expressed in different artifacts. That means that a tool supplier who builds a variants management tool has to implement an individual interface to each other tool that is used in a development process to create the corresponding artifacts. The purpose of the *Variability Exchange Language* is to support the standardization of these interfaces by a common exchange format that defines which information is exchanged between a variants management tool and a tool that is used to manage a specific kind of artifacts in a development process. As mentioned above, such a tool may either be a tool that already supports the definition of variation points for the concerned artifact type, or it may be an adaptor that adds this capability to a base tool.

In fact, the *Variability Exchange Language* defines a requirement on tools or tool adaptors that intend to support variants management. Such a tool has to be able to extract the data that is defined in the *Variability Exchange Language* from the artifact that it manages and to incorporate the data that is sent from the variants management tool into this artifact. Beyond the exchange format, i.e. the contents of the information that is exchanged, also some basic operations are defined here. They define in which direction the variability information is intended to flow.

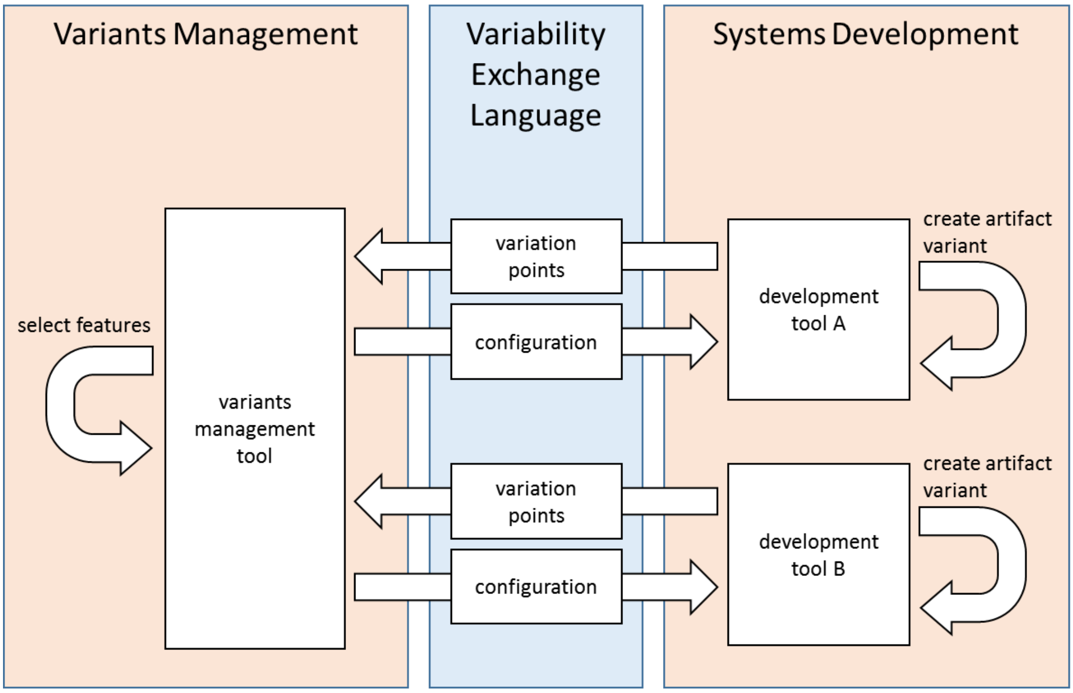


Figure 1 Use case for the *Variability Exchange Language*

A use case for the *Variability Exchange Language* can be defined as follows. Assume an artifact with variation points is given, for instance an artifact created with tool A in Figure 1. First the development tool has to collect the data defined in the *Variability Exchange Language,* essentially given by the variation points contained in the artifact. It passes this data to the variants management tool that builds a variability model based on the data. The variability model can be used to define a system configuration by selecting the desired system features. The corresponding data, i.e. the configuration, formatted according to the *Variability Exchange Language*, is passed back to the development tool or adaptor that uses this data to create or derive an artifact variant that corresponds to the system variant defined in the variants management tool.

Applying this scenario to all development tools and artifacts yields a consistent set of development artifacts for any system variant automatically. The variation points that correspond to customer relevant system features should coincide in all artifacts, i.e. they always induce the same variability model in the variants management tool. In addition to that there may also be internal variation points, for instance implementation variants that do not alter the visible properties of the system but are relevant for the system construction process. These variation points give rise to a staged variability model in which customer features are separated from internal features.

Since the system configuration is built once and for all in the variants management tool an identical configuration is passed to all development tools and thereby ensures consistency of the variants selection. It might only happen that internal features for instance are not interpreted by some development tool because it is not concerned with internal decisions, such as a requirements document or a system test.

3 Overview of the *Variability Exchange Language*

The core of the *Variability Exchange Language* is given by the definition of variation points and their variations – by the classes VariationPoint and Variation (see Figure 2). In the following we immediately use the class names from the meta-model presented in Chapter 4 to discuss the corresponding concepts, such as VariationPoint and Variation. This chapter gives a survey on the main classes, in particular the ones shown in Figure 2.

A detailed specification of all classes is provided in Chapter 4.

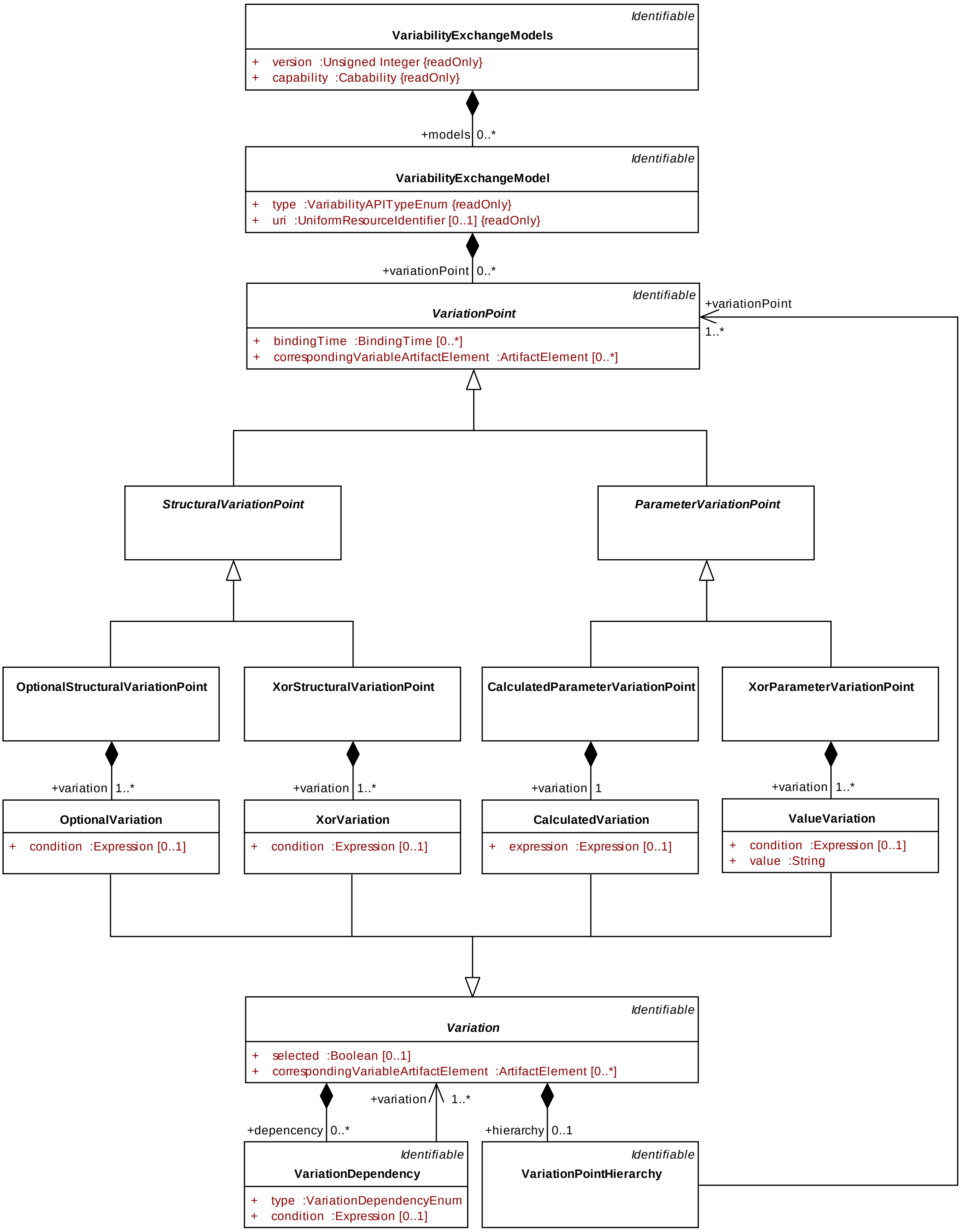

Identifiable
VariabilityExchangeModels
+ version :Unsigned Integer {readOnly}
+ capability :Cabability {readOnly}
+models
0..*
Identifiable
VariabilityExchangeModel
+ type :VariabilityAPITypeEnum {readOnly}
+ uri :UniformResourceIdentifier [0..1] {readOnly}
+variationPoint
0..*
Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
+variationPoint
1..*
StructuralVariationPoint
ParameterVariationPoint
OptionalStructuralVariationPoint
XorStructuralVariationPoint
CalculatedParameterVariationPoint
XorParameterVariationPoint
+variation
1..*
+variation
1..*
+variation
1
+variation
1..*
OptionalVariation
+ condition :Expression [0..1]
XorVariation
+ condition :Expression [0..1]
CalculatedVariation
+ expression :Expression [0..1]
ValueVariation
+ condition :Expression [0..1]
+ value :String
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
+variation
1..*
+depencency
0..*
+hierarchy
0..1
Identifiable
VariationDependency
+ type :VariationDependencyEnum
+ condition :Expression [0..1]
Identifiable
VariationPointHierarchy

Figure 2 An Overview of the Variability Exchange Language

A Variability Exchange Language document starts with a VariabilityExchangeModels element, which contains a number of VariabilityExchangeModel elements. Each VariabilityExchangeModel corresponds to one (or possibly several, but this is implementation dependent) artefacts with variable elements.

A VariabilityExchangeModel in turn contains a number of VariationPoints. Thus, a VariabilityExchangeModel describes the variable aspects of an artifact, *but only those*. All non-variable facets of the artifact are discarded because they are not necessary for our purpose.

3.1 VariationPoints and Variations

As shown in Figure 2, we distinguish between two different kinds of VariationPoints:

1. StructuralVariationPoints are variation points where the structure of a model changes during the binding process. StructuralVariationPoints define which elements are contained in a bound artifact. There are two kinds of structural variation points:
    a. OptionalStructuralVariationPoint – variation points that can be selected or deselected.
    b. XorStructuralVariationPoint – i.e. variation points that represent sets of alternatives from which exactly one can be selected.
2. ParameterVariationPoints are variation points which select a numerical value for a parameter during the binding process. They do not change the structure of an artifact. There are two kinds of parameter variation points:
    a. CalculatedParameterVariationPoint – variation points where the parameter value is calculated by an expression.
    b. XorParameterVariationPoint – variation points where the parameter value is selected from a list of values.

Each VariationPoint is associated with one or more Variations. The Variations enumerate the possible variants for their respective VariationPoints. When an artifact is bound, then one of these variations (OptionalStructuralVariationPoints also allow zero variations here) is selected to be included in the bound artifact, and all others are discarded.

Both Variations and VariationPoints may refer to artifact elements (correspondingVariableArtifactElement), for example the Simulink block or the line of code which correspond to the VariationPoint respectively Variation.

VariationPoints can further define dependencies on other variation points (VariationDependency), for example one variation point may require another variation points. This is useful to express technical dependencies in artifacts.

Furthermore, a VariationPoint may contain other VariationPoints to establish a hierarchy (VariationPointHierarchy), similarly to subsystem blocks in Simulink or hierarchies in software architectures.

3.2 Variation Point Descriptions versus Variation Point Selections

A VariabilityExchangeModel as defined in Figure 2 can actually serve two different purposes:

- A *variation point description* lists all variation points and *all* their variations; that is it describes a complete product line.
- A *variation point description* also lists all variation points, but selects one (or zero for optional variation points) Variation for each variation point. The attribute selected of Variation is used for that purpose. Any such selection must be consistent with the expression or condition attribute of a Variation, as well as with dependencies between variation points.

Both variation point descriptions and variation point selections use the same structure; the attribute type of VariabilityExchangeModel determines how a VariabilityExchangeModel should be interpreted.

3.3 Binding

The *Variability Exchange Language* does not make any assumptions on how the binding process for the associated artifact works. We do however provide a way to attach Conditions or Expressions to Variations:

- In a StructuralVariationPoint, a Variation comes with a Condition that determines whether the associated artifact element is part of a bound artifact.
- In a ParameterVariationPoint, the Variation determines a value for the associated artifact element. This is done either by computing it (CalculatedVariation) or selecting from one of several values (ValueVariation).

In a variation point description (see section 3.2) the result of the evaluation of a condition or expression in a Variation must be compatible with the attribute selected of a Variation. That is, if the attribute selected of a Variation has the value *true*, then its condition must also evaluate to *true*.

3.4 Common Concepts

Most classes in the *Variability Exchange Language* are based on the class Identifiable, which provides them with a name and a unique identifier. Identifable also provide a way to attach application-specific data (SpecialData) to elements in the *Variability Exchange Language*.

3.5 API

In addition to the contents of the exchange format basic operations of a Variability Interface are defined in the class VariabilityAPI. These operations cover the following operations:

- The import and export of VariabilityExchangeModels
- Getting and setting configurations, which are also VariabilityExchangeModels
- Getting information on the read or write access (Capability) to VariationPoints and VariabilityExchangeModels as configurations.

4 *Variability Exchange Language* Class Reference

In this chapter, we use the phrases *must*, *must not*, *shall*, shall not, *should*, *should not* and *may* in conformance with RFC 2119 [6]:

- **MUST** – This word, or the terms "**REQUIRED**" or "**SHALL**", mean that the definition is an absolute requirement of the specification.
- **MUST NOT** – This phrase, or the phrase "**SHALL NOT**", mean that the definition is an absolute prohibition of the specification.
- **SHOULD** – This word, or the adjective "**RECOMMENDED**", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- **SHOULD NOT** – This phrase, or the phrase "**NOT RECOMMENDED**" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- **MAY** – This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

Furthermore, we are using the following typographic conventions:

- An underlined word is the name of an UML class, ULM attribute or other UML element.
- A word set in typewriter font is the name of an XML element or or XML code.
- A paragraph that is marked with a ➢ symbol on the margin is a constraint.

4.1 ArtifactElement `<artifact-element-type>`

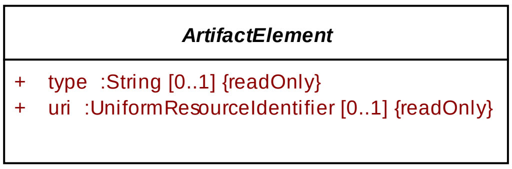

ArtifactElement
+ type :String [0..1] {readOnly}
+ uri :UniformResourceIdentifier [0..1] {readOnly}

Figure 3 UML Diagram for class ArtifactElement

```
<xs:complexType name="artifact-element-type">
   <xs:sequence>
      <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="type" type="xs:string" use="optional"/>
   <xs:attribute name="uri" type="xs:anyURI" use="optional"/>
</xs:complexType>
```

Listing 1 XML Schema for `artifact-element-type`

```
<optional-structural-variationpoint id="vp1">
   <variation id="vp1v1">
      <corresponding-variable-artifact-element uri="file:///C:/SPES/file1.c"/>
      <condition type="single-feature-condition">Feature1</condition>
   </variation>
</optional-structural-variationpoint>
```

Listing 2 XML Example for `artifact-element-type` using URIs

```
<optional-structural-variationpoint id="vp2">
   <variation id="vp2v1">
      <corresponding-variable-artifact-element type="simulink">
         <simulink-id>12</simulink-id>
      </corresponding-variable-artifact-element>
      <condition type="single-feature-condition">Feature1</condition>
   </variation>
</optional-structural-variationpoint>
```

Listing 3 XML Example for `artifact-element-type` using artifact-specific XML elements

4.1.1 Description

An ArtifactElement is a reference to an element in an artifact.

4.1.2 Attribute uri

The optional attribute uri is a reference to the artifact. The content of the attribute uri is a Uniform Resource Identifier.

- The attribute URI of an ArtifactElement should conform to the definition of Uniform Resource Locators as specified in [3].
- Although the attribute URI of ArtifactElement is optional, it is recommended to supply an URI instead of additional attributes (that is, arbitrary XML child elements as described in section 4.1.4) whenever possible.

4.1.3 Attribute type

The optional attribute type specifies the type of artifact that is addressed by this ArtifactElement.

The attribute type is a string, not an enumeration so that new artifact types can be added without changing the XML schema. Nevertheless, the following types are pre-defined:

- `simulink`
- `doors`

➢ Although the attribute type of an ArtifactElement is defined as optional, it is recommended to supply a type.

4.1.4 Adding arbitrary XML Elements

In the XML schema, the type artifact-element-type allows arbitrary XML child elements. This is implemented by using the `<xs:any>` element (see Listing 1), which permits the use of any XML element regardless of whether it is defined in the current schema. The type of the artifact is documented in the `type` attribute.

For example, Listing 3 shows a Variation whose corresponding variable artifact element is a Simulink block with the Identifier 12.

4.2 BindingTime `<bindingtime-type>`

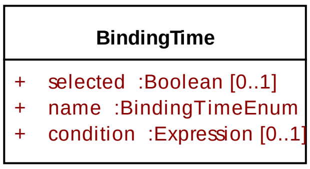


Figure 4 UML Diagram for class BindingTime

```
<xs:complexType name="bindingtime-type">
   <xs:sequence>
      <xs:element name="name"
                  type="bindingtime-enum"/>
      <xs:element name="condition"
                  type="expression-type"
                  minOccurs="0"
                  maxOccurs="1" />
   </xs:sequence>
   <xs:attribute name="selected" type="xs:boolean" use="optional"/>
</xs:complexType>
```

Listing 4 XML Schema for `bindingtime-type`

```
<variability-exchange-model type="variationpoint-description" id="model">
   <optional-structural-variationpoint id="vp1">
      <bindingtime>
         <name>preprocessor-time</name>
      </bindingtime>
      <variation id="vp1v1">
         <condition type="single-feature-condition">Feature1</condition>
      </variation>
   </optional-structural-variationpoint>
   <optional-structural-variationpoint id="vp2">
      <bindingtime>
         <name>preprocessor-time</name>
         <condition type="single-feature-condition">
            SmallSoftwareFootprint
         </condition>
      </bindingtime>
      <bindingtime>
         <name>post-build</name>
         <condition type="single-feature-condition">
            LargeSoftwareFootprint
         </condition>
      </bindingtime>
      <variation id="vp2v1">
         <condition type="single-feature-condition">Feature1</condition>
      </variation>
   </optional-structural-variationpoint>
</variability-exchange-model>
```

Listing 5 XML Example for `binding-time-type` in a `variationpoint-configuration`

```
<variability-exchange-model type="variationpoint-configuration" id="model">
   <optional-structural-variationpoint id="vp1">
      <bindingtime>
         <name>preprocessor-time</name>
      </bindingtime>
      <variation id="vp1v1">
         <condition type="single-feature-condition">Feature1</condition>
      </variation>
   </optional-structural-variationpoint>
   <optional-structural-variationpoint id="vp2">
      <bindingtime selected="false">
         <name>preprocessor-time</name>
         <condition type="single-feature-condition">
            SmallSoftwareFootprint
         </condition>
      </bindingtime>
      <bindingtime selected="true">
         <name>post-build</name>
         <condition type="single-feature-condition">
            LargeSoftwareFootprint
         </condition>
      </bindingtime>
      <variation id="vp2v1">
         <condition type="single-feature-condition">Feature1</condition>
      </variation>
   </optional-structural-variationpoint>
</variability-exchange-model>
```

Listing 6 XML Example for `binding-time-type` in a `variationpoint-configuration`

4.2.1 Description

The *binding time* of a variation point describes how the associated variability is resolved[1]. Common ways to resolve a variation point are

- A variation point is removed from its artefact. For example, the `#ifdef / #endif` idiom commonly found in a C preprocessor code removes part of the source code.
- A variation point is set to “inactive”. For example, an `if` statement may prevent certain code sections from being executed. This is typically used if the binding comes too late in the process and the code cannot be removed.
- A parameter is assigned a fixed value.

What exactly happens when a variation point is bound is implementation specific, and beyond the scope this document.

4.2.2 Attribute selected

A VariationPoint may have more than one BindingTime attributes. This is useful if the decision for the binding time of the variation point is delayed. For example, it may not be clear from the beginning whether a particular subsystem is removed during code

---

[1] Contrary to what the term Binding*Time* suggests, this is not a point in time, but rather a phase in the build process.

generation (binding time CodeGenerationTime, section 4.3.5) or just deactivated during startup (binding time PostBuild, section 4.3.10). This decision is made at some time during the build process.

- The attribute selected of a BindingTime shall be present if the VariabilityExchangeModel which contains the BindingTime is of type VariationPointSelection.
- The attribute selected has no effect if the type of the VariabilityExchangeModel is of type VariationPointDescription and thus shall be omitted.

If a VariationPoint has more than one bindingtime attribute, then the attribute selected is used to designate exactly one of the binding times as the binding time that is actually used for the binding:

- Let v be a VariationPoint which and let $s_1, \dots, s_n$ be the values of the selected attributes of the BindingTimes of v. Then the following conditions shall hold:
  1. $\exists i \in \{1, \dots, n\}: s_i = true$
  2. $\forall j \in \{1, \dots, n\}, i \neq j: s_i = false$
- If a BindingTime has both an attribute selected s and an an attribute condition c, then the following condition shall hold:

$$eval(c) = s$$

4.2.3 Attribute name

The attribute name of a BindingTime is a textual representation of the binding time. It is of type BindingTimeEnum.

4.2.4 Attribute condition

If a VariationPoint v has multiple BindingTimes, then the attribute condition may be used to select one BindingTime as the actual BindingTime for v.

- Let v be a VariationPoint which and let $c_1, \dots, c_n$ be the conditons of the BindingTimes of v. Then the following conditions shall hold:
  3. $\exists i \in \{1, \dots, n\}: eval(c_j) = true$
  4. $\forall j \in \{1, \dots, n\}, i \neq j: eval(c_j) = false$

In other words, if a VariationPoint has more than one BindingTime with a condition, then only one condition shall evaluate to true. Obviously, a condition is only useful of a VariationPoint has more than one BindingTime.

See section 4.2.2 for more information on condition is used to select a binding time.

4.3 BindingTimeEnum `<bindingtime-enum>`

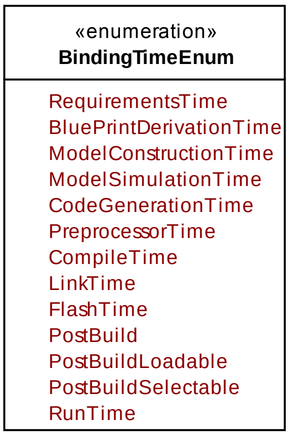


Figure 5 UML Diagram for enumeration BindingTimeEnum

```
<xs:simpleType name="bindingtime-enum">
   <xs:restriction base="xs:string">
      <xs:enumeration value="requirements-time"/>
      <xs:enumeration value="blueprint-derivation-time"/>
      <xs:enumeration value="model-construction-time"/>
      <xs:enumeration value="model-simulation-time"/>
      <xs:enumeration value="code-generation-time"/>
      <xs:enumeration value="preprocessor-time"/>
      <xs:enumeration value="compile-time"/>
      <xs:enumeration value="link-time"/>
      <xs:enumeration value="flash-time"/>
      <xs:enumeration value="post-build"/>
      <xs:enumeration value="post-build-loadable-time"/>
      <xs:enumeration value="post-build-selectable-time"/>
      <xs:enumeration value="run-time"/>
   </xs:restriction>
</xs:simpleType>
```

Listing 7 XML Schema for `bindingtime-enum`

4.3.1 RequirementsTime

At RequirementsTime, variants are bound by selecting a subset of the overall requirements for a product line.

4.3.2 BluePrintDerivationTime

The binding time BlueprintDerivationTime stems from AUTOSAR. In AUTOSAR, Blueprints are predefined templates for partial models. When a blueprint is applied, the variation points in the blueprint indicate locations in the template where a template processor or even human developer needs to fill in more information.

4.3.3 ModelConstructionTime

At ModelConstructionTime, variants are bound by modifying the artifact. This may involve deleting part of the model, but may also be achieved by adding new elements to a model or changing parts of the existing model, or a combination of all three.

4.3.4 ModelSimulationTime

At ModelSimulationTime, variants are bound by excluding parts of the model during simulation. This is typically done by constructing the model in such a way that some parts are not used during the simulation.

4.3.5 CodeGenerationTime

At CodeGenerationTime, variants are bound by generating code that is tailored for one or more variants.

4.3.6 PreprocessorTime

At PreProcessorTime, variants are bound by using a preprocessor that emits code only for specific variants. To do that, the code must contain appropriate preprocessor directives, for example `#ifdef` statements.

4.3.7 CompileTime

At CompileTime, variation points are resolved by the compiler, for example by not generating code for certain variants (dead code elimination) or by using specific compiler switches.

4.3.8 LinkTime

At Linktime, variants are bound by using only those files that are necessary for a particular variant are used to build a library or application.

4.3.9 FlashTime

At FlashTime, variants are bound by (pre)loading variant specific data sets into the flash memory embedded device.

4.3.10 PostBuild

At PostBuild, variants are bound by activating only certain parts of an application.

4.3.11 PostBuildLoadable

At PostBuildLoadable, variants are bound by selecting a parameter set (typically stored in flash memory) at the launch of an application. PostBuildLoadable is often used as a synonym for PostBuild.

4.3.12 PostBuildSelectable

At PostBuildSelectable, variants are bound by selecting one of several parameter sets (typically stored in flash memory) at the launch of an application. PostBuildSelectable is often used as a synonym for PostBuild.

4.3.13 RunTime

At RunTime, variants are bound by switching between different program states or executing different parts of an application. Runtime is usually not regarded as a binding time, but is included for completeness here.

4.4 CalculatedParameterVariationPoint `<calculated-parameter-variationpoint-type>`

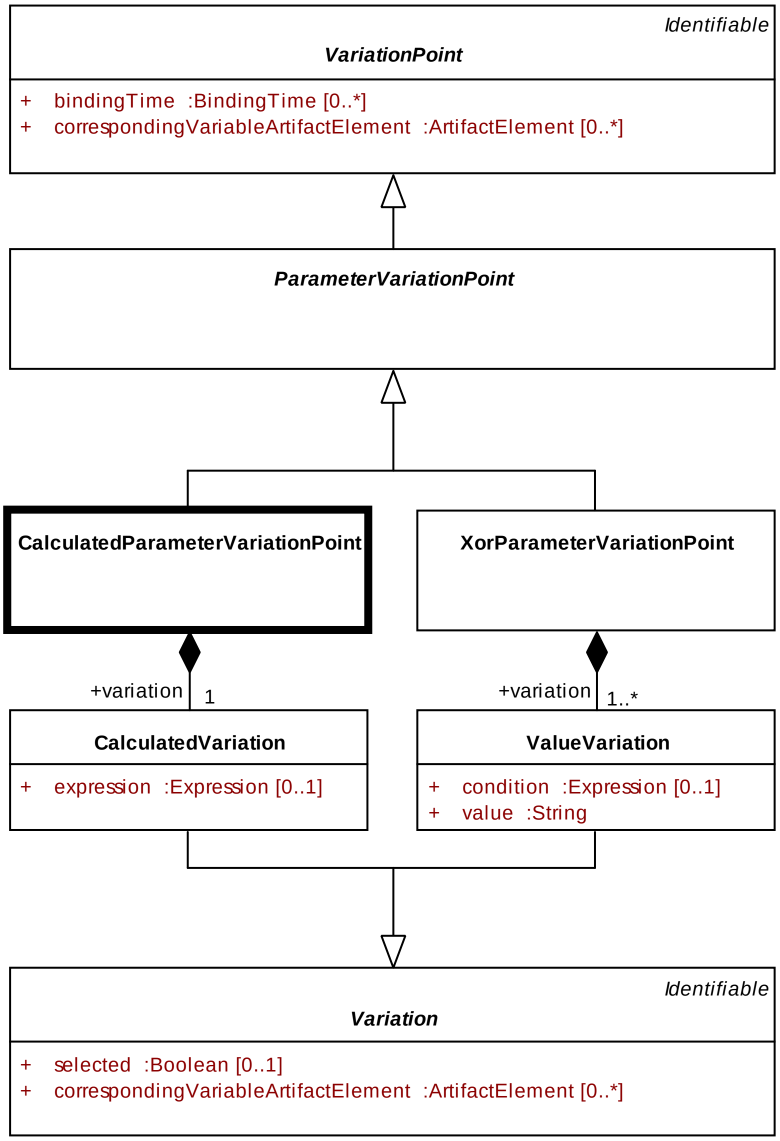

Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
ParameterVariationPoint
CalculatedParameterVariationPoint
XorParameterVariationPoint
+variation
1
+variation
1..*
CalculatedVariation
+ expression :Expression [0..1]
ValueVariation
+ condition :Expression [0..1]
+ value :String
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 6 UML Diagram for class CalculatedParameterVariationPoint

```
<xs:complexType name="calculated-parameter-variationpoint-type">
   <xs:complexContent>
      <xs:extension base="variationpoint-type">
         <xs:sequence>
            <xs:element name="variation" type="calculated-variation-type"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Listing 8 XML Schema for `calculated-parameter-variationpoint-type`

```
<calculated-parameter-variationpoint id="vp1">
   <variation id="vp1v1">
      <expression type="pvscl-expression">6*9</expression>
   </variation>
</calculated-parameter-variationpoint>
```

Listing 9 XML Example for `calculated-parameter-variationpoint-type`

4.4.1 Description

A CalculatedParameterVariatonPoint is a ParameterVariationPoint that defines a value for a variation point in an artifact. Unlike a XorParameterVariationPoint, which picks one value from a number of choices, a CalculatedParameterVariatonPoint uses an expression to define the value.

A CalculatedParameterVariatonPoint contains a single CalculatedVariation whose attribute expression defines the expression that is used to calculate the value for the associated variation point.

4.4.2 Notes

- The class CalculatedParameterVariationPoint inherits from the class ParameterVariationPoint, which inherits from VariationPoint.

4.5 CalculatedVariation `<calculated-variation-type>`

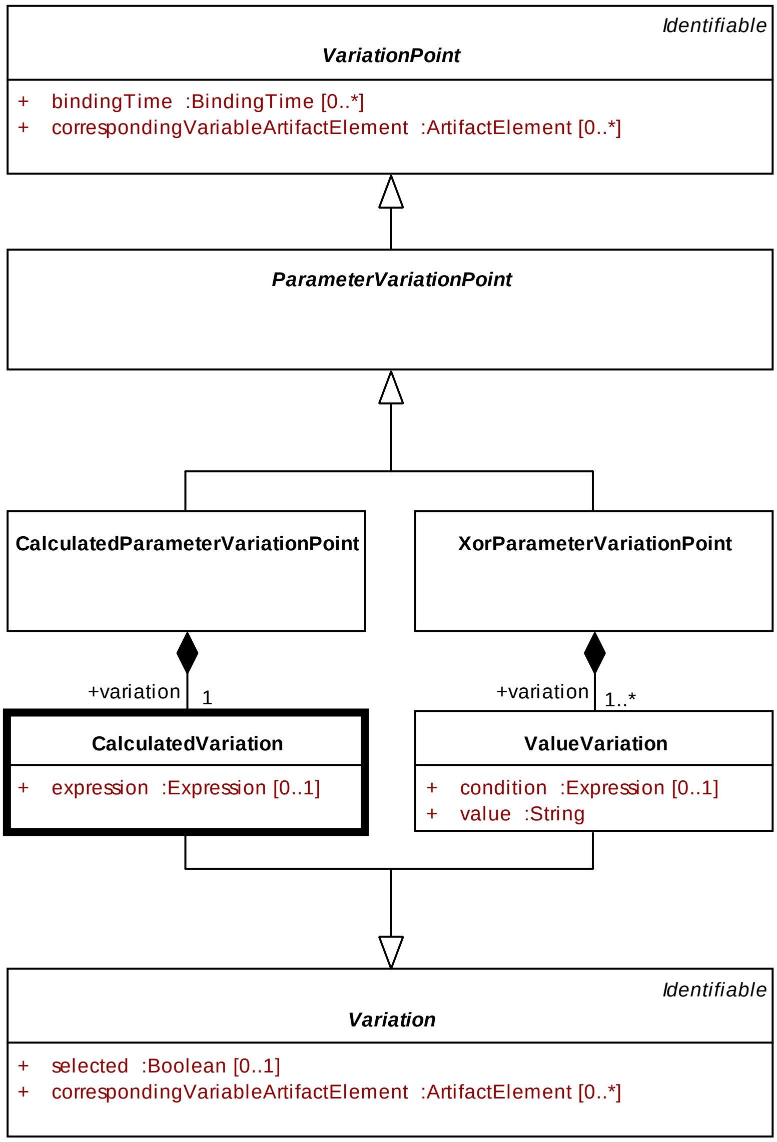

Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
ParameterVariationPoint
CalculatedParameterVariationPoint
XorParameterVariationPoint
+variation
1
+variation
1..*
CalculatedVariation
+ expression :Expression [0..1]
ValueVariation
+ condition :Expression [0..1]
+ value :String
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 7 UML Diagram for class CalculatedVariation

```
<xs:complexType name="calculated-variation-type">
   <xs:complexContent>
      <xs:extension base="variation-type">
         <xs:sequence>
            <xs:element name="expression"
                        type="expression-type"
                        minOccurs="0" maxOccurs="1"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Listing 10 XML Schema for `calculated-variation-type`

```
<calculated-parameter-variationpoint id="vp1">
   <variation id="vp1v1">
      <expression type="pvscl-expression">6*9</expression>
   </variation>
</calculated-parameter-variationpoint>
```

Listing 11 XML Example for `calculated-variation-type`

4.5.1 Description

Each CalculatedParameterVariationPoint aggregates a single CalculatedVariation. A CalculatedVariation is a Variation that determines a value for a Calculated-VariationPoint.

4.5.2 Attribute expression

The optional attribute expression of a CalculatedVariation specifies the expression that is used to compute the value of a CalculatedVariation.

- The attribute expression of a CalculatedVariation may return an arbitrary value. Which values are allowed depends on the artifact elements which are referenced by the attribute correspondingVariableArtifactElement (see 4.21.3).

4.5.3 Binding

When a CalculatedParameterVariationPoint is bound, the expression of its CalculatedVariation is evaluated. The result of the evaluation gets assigned to the artifact element(s) which are referenced by the attribute correspondingVariableArtifact-Element (see section 4.21.3).

- A CalculatedParameterVariationPoint can only be bound when its CalculatedVariation has an attribute expression.

4.5.4 Notes

- The class CalculatedVariation inherits from the class Variation.

4.6 Capability `<capability-type>`

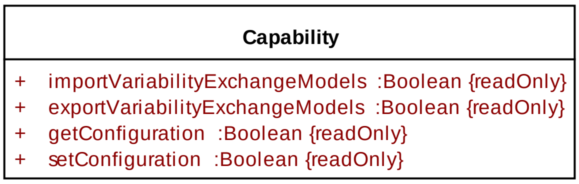

Capability
+ importVariabilityExchangeModels :Boolean {readOnly}
+ exportVariabilityExchangeModels :Boolean {readOnly}
+ getConfiguration :Boolean {readOnly}
+ setConfiguration :Boolean {readOnly}

Figure 8 UML Diagram for class Capability

```
<xs:complexType name="capability-type">
   <xs:sequence>
      <xs:element name="import-variability-exchange-model" type="xs:boolean" />
      <xs:element name="export-variability-exchange-model" type="xs:boolean" />
      <xs:element name="get-configuration" type="xs:boolean" />
      <xs:element name="set-configuration" type="xs:boolean" />
   </xs:sequence>
</xs:complexType>
```

Listing 12 XML Schema for `capability-type`

```
<?xml version="1.0" encoding="UTF-8"?>
<variability-exchange-models id="root"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:noNamespaceSchemaLocation="../VariabilityExchangeLanguage.xsd">
   <version>1</version>
   <capability>
      <import-variability-exchange-model>true</import-variability-exchange-model>
      <export-variability-exchange-model>true</export-variability-exchange-model>
      <get-configuration>true</get-configuration>
      <set-configuration>true</set-configuration>
   </capability>
</variability-exchange-models>
```

Listing 13 XML Example for `capability-type`

4.6.1 Description

A Capability describes which operations are supported by a particular instance of VariabilityAPI. The rationale for introducing Capability is that not all implementations of the VariabilityAPI support all its operations.

4.6.2 Attribute importVariabilityExchangeModels

The attribute getVariationPoints determines whether the operation importVariabilityExchangeModels of the class VariabilityAPI is supported:

- If getVariationPoints is *true*, then the VariabilityAPI supports the operation importVariabilityExchangeModels.
- If getVariationPoints is *false*, then the VariabilityAPI does not support the operation importVariabilityExchangeModels.

4.6.3 Attribute exportVariabilityExchangeModels

The attribute setVariationPoints determines whether the operation exportVariabilityExchangeModels of the class VariabilityAPI is supported:

- If setVariationPoints is $true$, then the VariabilityAPI supports the operation exportVariabilityExchangeModels.
- If setVariationPoints is $false$, then the VariabilityAPI does not support the operation exportVariabilityExchangeModels.

4.6.4 Attribute getConfiguration

The attribute getConfiguration determines whether the operation getConfiguration of the class VariabilityAPI is supported:

- If getConfiguration is $true$, then the VariabilityAPI supports the operation getConfiguration.
- If getConfiguration is $false$, then the VariabilityAPI does not support the operation getConfiguration.

4.6.5 Attribute setConfiguration

The attribute setConfiguration determines whether the operation setConfiguration of the class VariabilityAPI is supported:

- If setConfiguration is $true$, then the VariabilityAPI supports the operation setConfiguration.
- If setConfiguration is $false$, then the VariabilityAPI does not support the operation setConfiguration.

4.7 Expression `<expression-type>`

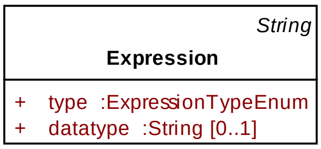


Figure 9 UML Diagram for class Expression

```
<xs:complexType name="expression-type">
   <xs:simpleContent>
      <xs:extension base="xs:string">
         <xs:attribute name="type" type="expression-enum" use="required"/>
         <xs:attribute name="datatype" type="xs:string" use="optional"/>
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
```

Listing 14 XML Schema for `expression-type`

```
<xor-structural-variationpoint id="vp1">
   <variation id="vp1v1">
      <condition type="single-feature-condition" datatype="bool">
         Feature1
      </condition>
   </variation>
   <variation id="vp1v2">
      <condition type="and-feature-condition" datatype="bool">
         Feature2,Feature3
      </condition>
   </variation>
   <variation id="vp1v3">
      <condition type="or-feature-condition" datatype="bool">
         Feature4, Feature5, Feature6
      </condition>
   </variation>
   <variation id="vp1v4">
      <condition type="pvscl-expression" datatype="ps:boolean">
         Feaure7 AND Feature8
      </condition>
   </variation>
</xor-structural-variationpoint>
```

Listing 15 XML Example for `expression-type`

4.7.1 Description

An Expression is similar to an expression in a programming language. In our case, expressions fall into two categories:

- “Genuine” expressions which may return any kind of value. These are represented by the type PVSCLExpression.
- Constraints, which may only return Boolean values. These are represented by the SingleFeatureExpression, AndFeatureExpression and OrFeatureExpression. A constraint may also be of type PVSCLExpression; in this case the return value must be of type Boolean.

Technically, an Expression is a string whose syntax is determined by the attribute type. In the XML representation, the actual expression is contained in the inner text of the expression of condition element[2].

- ➢ An expression shall not be an empty string.

4.7.2 Attribute type

The attribute type defines the kind of expression that in the inner text of the expression of condition element. There are several kinds of expressions:

- SingleFeatureExpression (`single-feature-condition`)
- AndFeatureExpression (`and-feature-condition`)
- OrFeatureExpression (`or-feature-condition`)
- PVSCLExpression (`pvscl-expression`)
- OCLExpression (`ocl-expression`)
- AUTOSARExpression (`autosar-expression`)

The individual expression types are explained in subsections 4.7.4, 4.7.5, 4.7.6, 4.7.7, 4.7.8 and 4.7.9.

4.7.3 Attribute datatype

The attribute datatype constrains the return type of the expression. Since the possible values for datatype depend on the artifact(s) involved, they are not further standardized here.

- ➢ If the attribute datatype of an Expression exists, then the return type of the Expression should be compatible with the data type given by datatype.

4.7.4 SingleFeatureCondition

A SingleFeatureCondition is a type of expression that models a Boolean condition whose literal is a single Feature. SingleFeatureCondition is a special case of OrFeatureCondition or AndFeatureCondition that can be used in cases where a variable element in an artifact depends on a single feature instead of a combination of features.

The example in Listing 15 translates to the Boolean expression

$$Feature_1$$

- ➢ Formally, if a SingleFeatureCondition references the feature f_1, then this translates into the Boolean expression

---

[2] For simplicity and consistency, XML elements of type expression-type are always named `expression` or `condition`.

$$\mathrm{eval}(f_i)$$

where $\mathrm{eval}(f_i)$ is $true$ if feature f_i is selected, and $\mathrm{eval}(f_i)$ is f_i is not selected.

- ➢ The datatype for an Expression of type SingleFeatureExpression should be Boolean.

See also section 4.7.10.1 on how single features are represented in XML.

4.7.5 AndFeatureCondition

An AndFeatureCondition is a special Condition that models a Boolean condition whose literals are features, and which are connected by a Boolean *AND*. The example in Listing 15 translates to the Boolean expression

$$Feature_2 \wedge Feature_3 \wedge Feature_4$$

- ➢ If an AndFeatureCondition references the features $f_1, f_2, \ldots, f_n$, then this translates into the following Boolean expression

$$\bigwedge_{1 \le i \le n} \mathrm{eval}(f_i)$$

where $\mathrm{eval}(f_i)$ is $true$ if feature f_i is selected, and $false$ otherwise.

- ➢ The datatype for an Expression of type AndFeatureCondition should be Boolean.

In the XML representation, an AndFeatureCondition is comma-separated list of features. See also section 4.7.10.2 on how features are represented in XML.

4.7.6 OrFeatureCondition

An OrFeatureCondition is a special Condition that models a Boolean condition whose literals are features, and which are connected by a Boolean *OR*. The example in Listing 15 translates to the Boolean expression

$$Feature_5 \vee Feature_6$$

- ➢ Formally, if a OrFeatureCondition references the features $f_1, f_2, \ldots, f_n$, then this translates into the following Boolean expression

$$\bigvee_{1 \le i \le n} \mathrm{eval}(f_i)$$

where $\mathrm{eval}(f_i)$ is $true$ if feature f_1 is selected, and $false$ otherwise.

- ➢ The datatype for an Expression of type OrFeatureCondition should be Boolean.

In the XML representation, an OrFeatureCondition is comma-separated list of features. See also section 4.7.10.2 on how features are represented in XML.

4.7.7 PVSCLExpression

In terms of syntax and scope, PVSCLExpression is comparable to what most programming languages offer.

- An expression of type PVSCLExpression shall use the syntax defined by [5].
- An expression of type PVSCLExpression shall be evaluated according to the rules defined in [5].

4.7.8 OCLExpression

An OCLExpression uses the expression syntax and semantics defined by OCL, [7].

4.7.9 AUTOSARExpression

An AUTOSARExpression uses the expression syntax and semantics defined by AUTOSAR, [8].

4.7.10 Representation of expressions and features in XML

A Feature is a reference to an element in a model that describes the variability of an artifact, typically a feature model. The exact nature of a feature model is beyond the scope of this document.

- In the XML, a feature is just a name. How exactly a Feature is mapped to its corresponding element in the feature model is implementation dependent and beyond the scope of this document.
- The name of a Feature shall be unique. That is, if features f_1 and f_2 have the same string representation, then they are assumed to refer to the same element of the same feature model.

4.7.10.1 Syntax for `single-feature-condition`

- In the XML representation, a feature is a string that matches the following pattern:

  \s*[a-zA-Z_]([a-zA-Z0-9_]*\s*

That is, a feature is a sequence of characters which starts with a letter or an underscore followed by letters, digits and underscores.

An XML element of type `expression`-type whose attribute `type` has the value `single-feature-condition` must match to the above pattern.

4.7.10.2 Syntax or `and-feature-condition` and `or-feature-condition`

- In the XML representation, a comma-separated list of features is a string that matches the following pattern[3]:

  \s*[a-zA-Z_]([a-zA-Z0-9_]*(\s*,\s*[a-zA-Z_]([a-zA-Z0-9_]*)*\s*

  An XML element of type `expression`-type whose attribute `type` has the value `and-feature-`condition or `or-feature-condition` must match to the above pattern.

---

[3] In this pattern, `\s` donates a white space, typically a space or tab character,or a newline.

4.8 ExpressionTypeEnum `<expression-enum>`

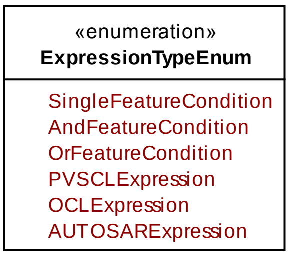


Figure 10 UML Diagram for class ExpressionTypeEnum

```
<xs:simpleType name="expression-enum">
   <xs:restriction base="xs:string">
      <xs:enumeration value="single-feature-condition"/>
      <xs:enumeration value="and-feature-condition"/>
      <xs:enumeration value="or-feature-condition"/>
      <xs:enumeration value="pvscl-expression"/>
      <xs:enumeration value="ocl-expression"/>
      <xs:enumeration value="autosar-expression"/>
   </xs:restriction>
</xs:simpleType>
```

Listing 16 XML Schema for `expression-enum`

4.8.1 Description

The enumeration ExpressionTypeEnum defines the possible values for the attribute type of the class Expression. The semantics of these expression types is explained in Section 4.7.

4.9 Identifiable `<identifiable-type>`

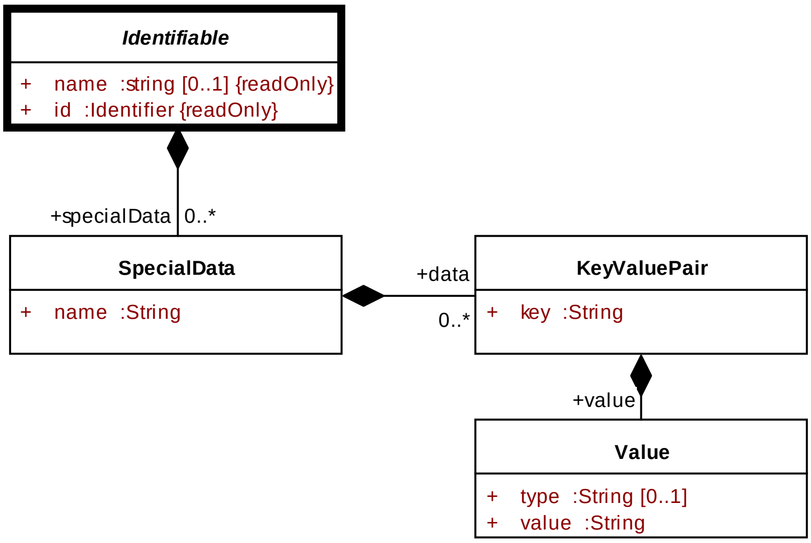


Figure 11 UML Diagram for class Identifiable

```
<xs:complexType name="identifiable-type" abstract="true">
   <xs:sequence>
      <xs:element name="special-data"
                  type="special-data-type"
                  minOccurs="0"
                  maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="name" use="optional">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>
```

Listing 17 XML Schema for `identifable-type`

```
<optional-structural-variationpoint id="vp1" name="optional variationpoint">
   <special-data name="CreatorInfo">
      <data>
         <key>Created</key>
         <value type="xs:date">1998-11-17</value>
      </data>
   </special-data>
   <variation id="vp1v1" name="optional variation">
      <condition type="single-feature-condition">Feature1</condition>
   </variation>
</optional-structural-variationpoint>
```

Listing 18 XML Example for `identifable-type` (`id` attribute)

4.9.1 Description

Identifiable is an *abstract* class that defines means to provide unique identifiers for elements of the variability exchange language. Identifiable is used as the base class of for many classes of the *Variability Exchange Language*.

In the XML Schema, `identifiable-type` does not define an XML element of its own, but adds two new attributes `id` and `name` to any type that is an extension of `identifiable-type`.

```
<xs:complexType name="variationpoint-type" abstract="true">
   <xs:complexContent>
      <xs:extension base="identifiable-type">
         <xs:sequence>
            <xs:element name="bindingtime"
                        type="bindingtime-type"
                        minOccurs="0"
                        maxOccurs="unbounded"/>
            <xs:element name="corresponding-variable-artifact-element"
                        type="artifact-element-type"
                        minOccurs="0"
                        maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Figure 12 Use of identifiable-type in the XML Schema

4.9.2 Attribute id

The attribute id of an Identifiable provides a unique identifier for an element.

In XML, `id` is an attribute of type `xs:ID`, which means that `id` is guaranteed to be unique within a *Variability Exchange Language* document. Other XML elements may use an attribute of type `xs:IDREF` to refer to an XML clement that is Identifiable.

- The value of the attribute id of an Identifiable shall be unique within a single *Variability Exchange Language* document. That is, the following condition holds:

  Let i_1 and i_2 be the values of the id XML attributes of the XML elements e_1 and e_2, with $i_1 \text{ equals } i_2$. Then e_1 and e_2 are the same elements.

  This is consistent with the definition of the types `xs:IDREF` and `xs:IDREFS` in XML.

- The value of the attribute id of an Identifiable shall not change over the lifetime of the element which the Identifiable represents.

  The reason for introducing the latter constraint is as follows. Imagine the following situation: the operations importVariabilityExchangeModels and getConfiguration return variability language exchange documents that contain information about the same variation point (in this context, “same” usually means that they refer to the same artifact elements).

Then, the attribute id should have an identical value in both the documents returned from importVariabilityExchangeModels and getConfiguration; otherwise there would be no way to match the variation points.

4.9.3 Attribute name

The attribute name of an Identifiable provides a human readable name for an element. It is recommended (but not enforced by the XML Schema) that all the name attributes of the Identifiable elements in a *Variability Exchange Language* document have unique values.

- ➢ The value of the attribute name of an Identifiable is not guaranteed to be unique within a single variability exchange language document. It is however strongly recommended to use unique values for name attributes as well.
- ➢ The value of attribute name shall not be an empty string.

4.9.4 Attribute specialData

Each Identifiable may aggregate one or more SpecialData objects. This makes sure that most elements in the *Variability Exchange Language* can be augmented with application specific data.

4.9.5 Notes

- Identifiable is an abstract class. Most of the classes described in this document inherit from Identifiable.

4.10 KeyValuePair `<key-value-pair-type>`

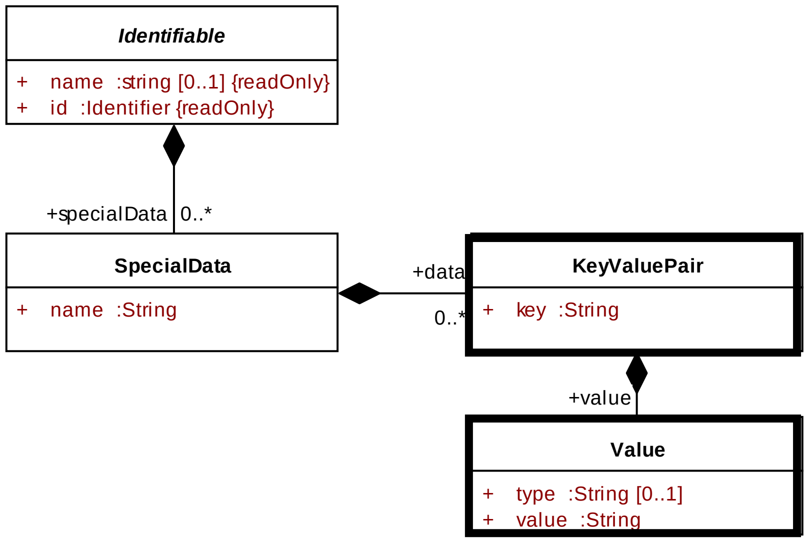


Figure 13 UML Diagram for class KeyValuePair

```
<xs:complexType name="key-value-pair-type">
   <xs:sequence>
      <xs:element name="key">
         <xs:simpleType>
            <xs:restriction base="xs:string">
               <xs:minLength value="1"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:element>
      <xs:element name="value">
         <xs:complexType>
            <xs:simpleContent>
               <xs:extension base="xs:string">
                  <xs:attribute name="type" type="xs:string" use="optional"/>
               </xs:extension>
            </xs:simpleContent>
         </xs:complexType>
      </xs:element>
   </xs:sequence>
</xs:complexType>
```

Figure 14 XML Schema for `key-value-pair-type`

```
<optional-structural-variationpoint id="vp1" name="optional variationpoint">
   <special-data name="CreatorInfo">
      <data>
         <key>Created</key>
         <value type="xs:date">1998-11-17</value>
      </data>
   </special-data>
   <variation id="vp1v1" name="optional variation">
      <condition type="single-feature-condition">Feature1</condition>
   </variation>
</optional-structural-variationpoint>
```

Figure 15 XML Example for `key-value-pair-type`

4.10.1 Description of Class KeyValuePair

Application specific data for VariationPoint and Variation objects is implemented by the class SpecialData, which aggregates a number of KeyValuePair elements. As the name already suggests, a KeyValuePair consists of a key and a value.

KeyValuePair is restricted to data that can be represented as strings. How key and value are interpreted is up to the application. It is strongly recommended to use the attribute key as some kind of (unique) identifier, and store the data associated with key in the attribute value.

4.10.2 Attribute key of Class KeyValuePair

The attribute key of class KeyValuePair provides a way to identify a KeyValuePair.

- A SpecialData object shall not contain two or more KeyValueData objects whose attribute key have the same value.

4.10.3 Description of Class Value

An object of class Value is a container for the value of a KeyValuePair.

4.10.4 Attribute value of Class Value

The attribute value of an object of class Value contains the application specific data that is associated with the key of the KeyValuePair object which aggregates this object.

4.10.5 Attribute type of Class Value

The attribute type of class Value can be used to indicate the data type of the value of a Value object. The contents of type are not standardized, but using XML data types such as `xs:string` or `xs:date` is recommended.

4.10.6 XML Representation

- As shown in Figure 15, a key-value pair is implemented by the XML elements `key` and `value`, which are enclosed by a `data` element[4]. The elements `key` and `value` are XML strings.
- The XML representation of a Value object is an XML element named `value` which contains an arbitrary string. Its definition is based on the XML type `xs:string` and defines an additional attribute `type` which indicates the data type of the content.

---

[4] The XML element data is not strictly necessary, but makes it easier to extend the key-value pair implementation in the future, if neccessary.

4.11 OptionalStructuralVariationPoint `<optional-structural-variaton-point-type>`

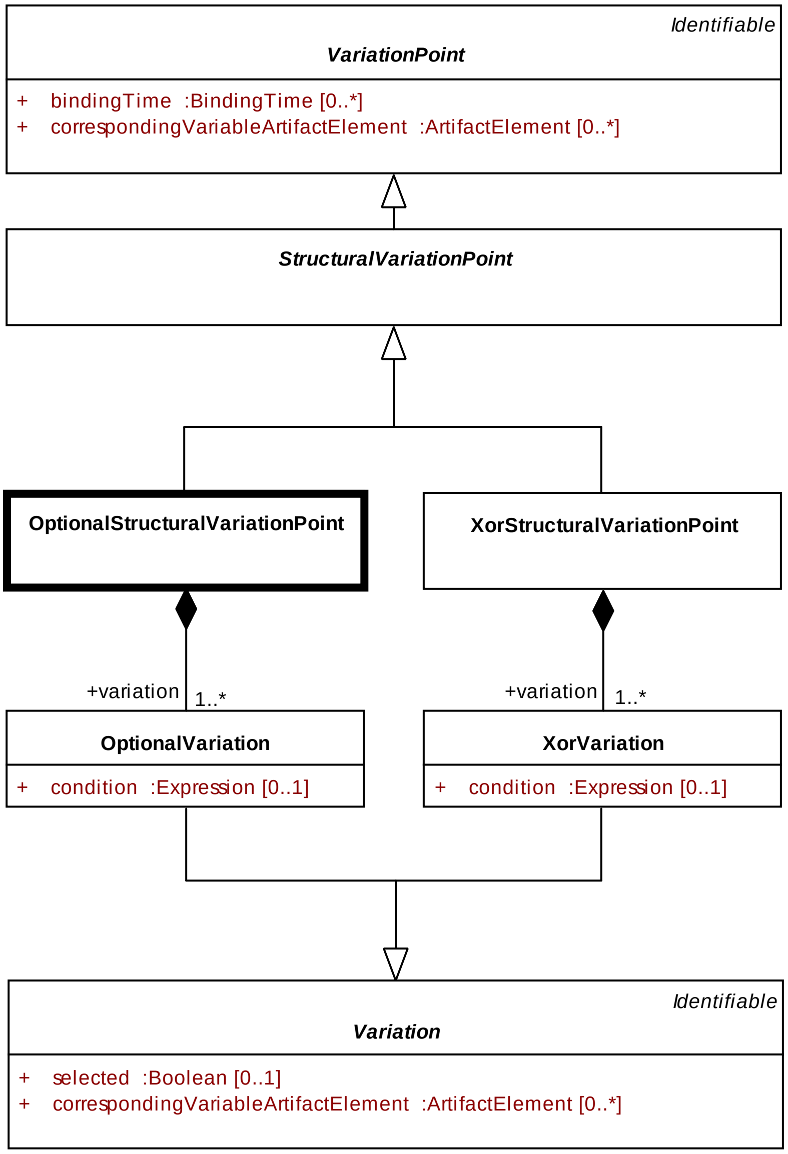

Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
StructuralVariationPoint
OptionalStructuralVariationPoint
XorStructuralVariationPoint
+variation
1..*
+variation
1..*
OptionalVariation
+ condition :Expression [0..1]
XorVariation
+ condition :Expression [0..1]
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 16 UML Diagram for class OptionalStructuralVariatonPoint

```
<xs:complexType name="optional-structural-variationpoint-type">
   <xs:complexContent>
      <xs:extension base="variationpoint-type">
         <xs:sequence>
            <xs:element name="variation"
                        type="optional-variation-type"
                        minOccurs="1"
                        maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Listing 19 XML Schema for `optional-structural-variaton-point-type`

```
<optional-structural-variationpoint id="vp1">
   <variation id="vp1v1">
      <condition type="single-feature-condition">feature1</condition>
   </variation>
</optional-structural-variationpoint>
```

Listing 20 XML Example for `optional-structural-variaton-point-type`

```
<optional-structural-variationpoint id="vp2">
   <variation id="vp2v1">
      <condition type="single-feature-condition">Feature1</condition>
   </variation>
   <variation id="vp2v2">
      <condition type="single-feature-condition">Feature2</condition>
   </variation>
   <variation id="vp2v3">
      <condition type="single-feature-condition">Feature3</condition>
   </variation>
</optional-structural-variationpoint>
```

Listing 21 XML Example for `optional-structural-variaton-point-type` with multiple variations

4.11.1 Description

An OptionalStructuralVariationPoint is a VariationPoint that contains one or more OptionalVariation objects.

4.11.2 Notes

- The class OptionalStructuralVariationPoint inherits from the class StructuralVariationPoint.

4.12 OptionalVariation `<optional-variation-type>`

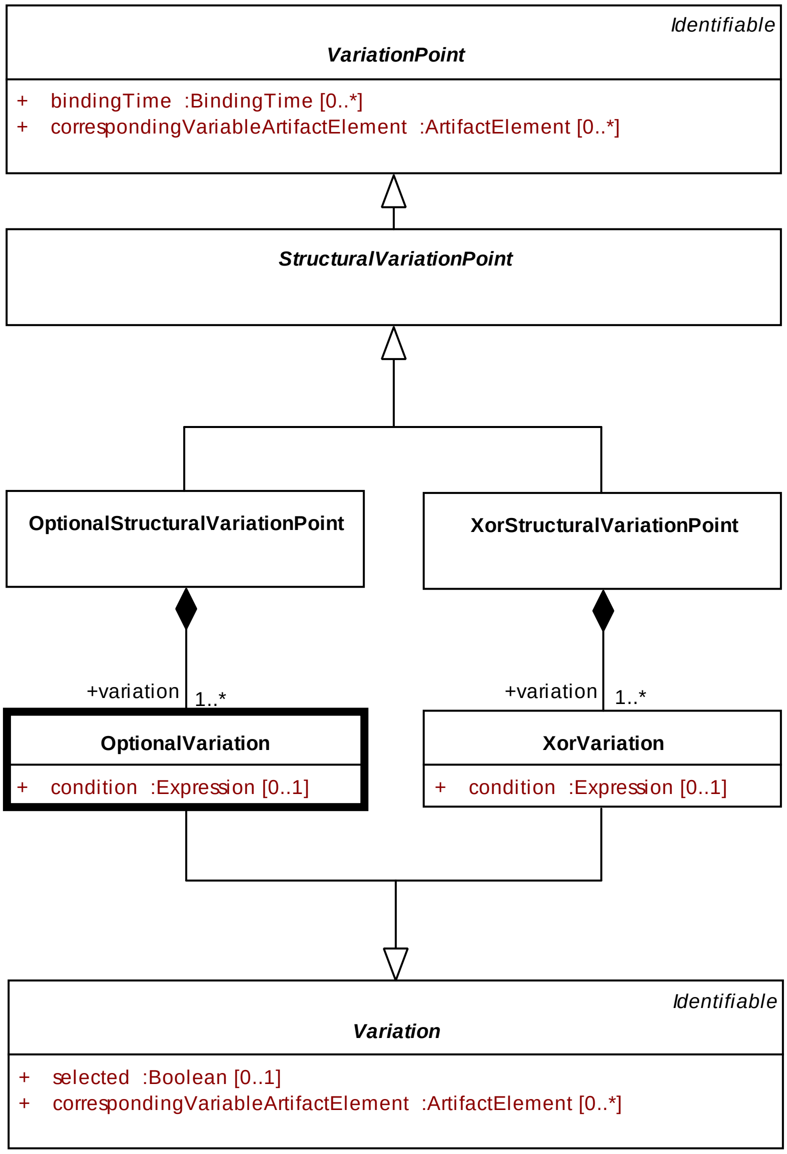

Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
StructuralVariationPoint
OptionalStructuralVariationPoint
XorStructuralVariationPoint
+variation
1..*
+variation
1..*
OptionalVariation
+ condition :Expression [0..1]
XorVariation
+ condition :Expression [0..1]
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 17 UML Diagram for class OptionalVariation

```
<xs:complexType name="optional-variation-type">
   <xs:complexContent>
      <xs:extension base="variation-type">
         <xs:sequence>
            <xs:element name="condition"
                        type="expression-type"
                        minOccurs="0" maxOccurs="1"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Listing 22 XML Schema for `optional-variation-type`

```
<optional-structural-variationpoint id="vp1">
   <variation id="v1">
      <condition type="or-feature-expression">Feature1,Feature2</condition>
   </variation>
</optional-structural-variationpoint>
```

Listing 23 XML Example for `optional-variation-type`

4.12.1 Description

Each OptionalStructuralVariationPoint aggregates one or more OptionalVariation objects. An OptionalVariation is a Variation that determines whether an OptionalStructuralVariationPoint gets deleted or set inactive during the binding process.

4.12.2 Attribute condition

The optional attribute condition of an OptionalVariation defines the expression that is used to compute the condition of an OptionalVariation.

- ➢ The attribute condition of an OptionalVariation shall return a Boolean value. That is, its datatype attribute (if present) should be a Boolean or a data type which can be converted into a Boolean.
- ➢ If an OptionalVariation has an attribute condition c and an attribute selected s (inherited from Variation), then the following condition shall hold:

$$eval(c) = s$$

4.12.3 Binding

When an OptionalStructuralVariationPoint is bound, the condition of each of its OptionalVariations is evaluated. If the result of the evaluation is $false$, then the artifact elements which are referenced by the attribute correspondingVariableArtifactElement (see section 4.21.3) of the OptionalVariation get deleted or set inactive.

- ➢ An OptionalStructuralVariationPoint can only be bound when all its OptionalVariations have a condition.

4.12.4 Notes

- The class OptionalVariation inherits from the class Variation.

4.13 ParameterVariationPoint

`<parameter-variationpoint-group>`

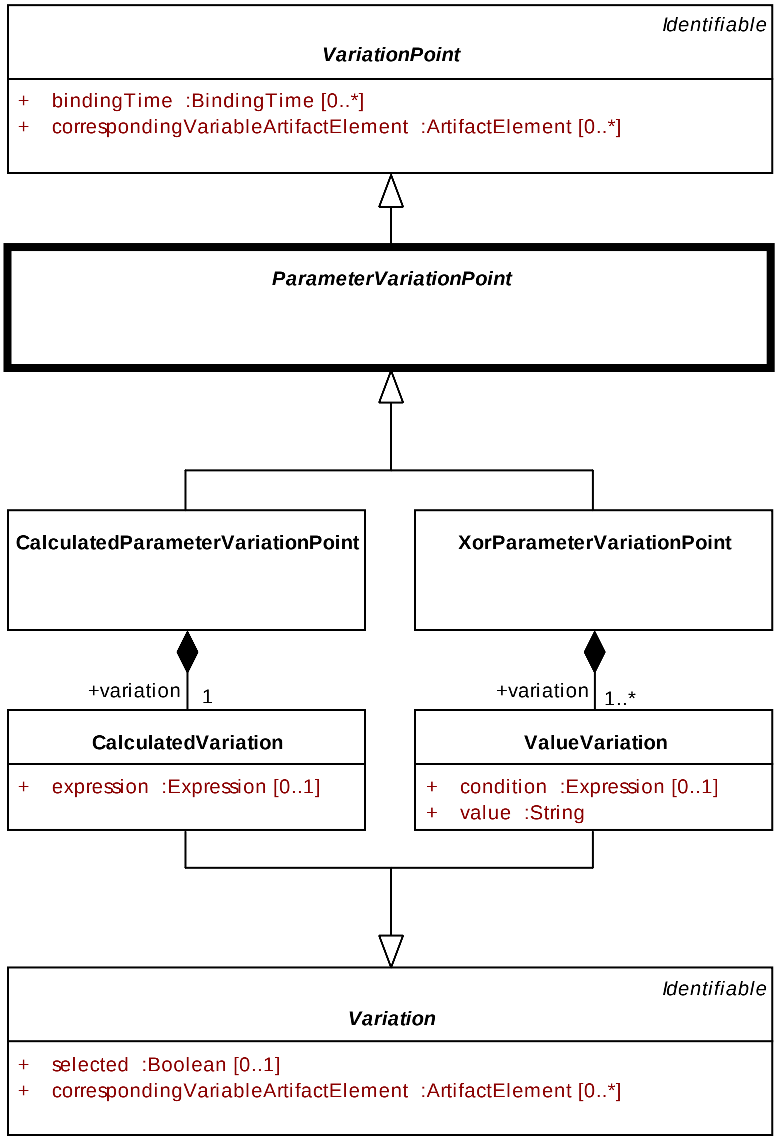

Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
ParameterVariationPoint
CalculatedParameterVariationPoint
XorParameterVariationPoint
+variation
1
+variation
1..*
CalculatedVariation
+ expression :Expression [0..1]
ValueVariation
+ condition :Expression [0..1]
+ value :String
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 18 UML Diagram for class ParamaterVariationPoint

```
<xs:group name="parameter-variationpoint-group">
   <xs:choice>
      <xs:element name="calculated-parameter-variationpoint"
                  type="calculated-parameter-variationpoint-type"/>
      <xs:element name="xor-parameter-variationpoint"
                  type="xor-parameter-variationpoint-type"/>
   </xs:choice>
</xs:group>
```

Listing 24 XML Schema for `parameter-variationpoint-group`

4.13.1 Description

A ParameterVariationPoint defines a value for a variable element in an artifact, for example

- A value or a C-preprocessor symbol (`#define`)
- A initialization value for a variable or a constant in a programing language
- A value for a variable in a Matlab workspace

The artifact elements are referenced by the attribute correspondingVariableArtifactElement of the ParameterVariationPoint and the attribute correspondingVariableArtifactElement of its Variation(s) (see the classes VariationPoint and Variation in Figure 18)

4.13.2 Notes

- The class ParameterVariationPoint is an abstract class which inherits from the class VariationPoint.
- There are two subclasses of ParameterVariationPoint: CalculatedParameterVariationPoint und XorParameterVariationPoint.
- Like StructuralVariationPoint, ParameterVariationPoint is implemented in the XL schema as group, not a type. We chose to use a group here because a type would have established an extra XML element for ParameterVariationPoint, which would only have complicated the document structure.

4.14 SpecialData <special-data-type>

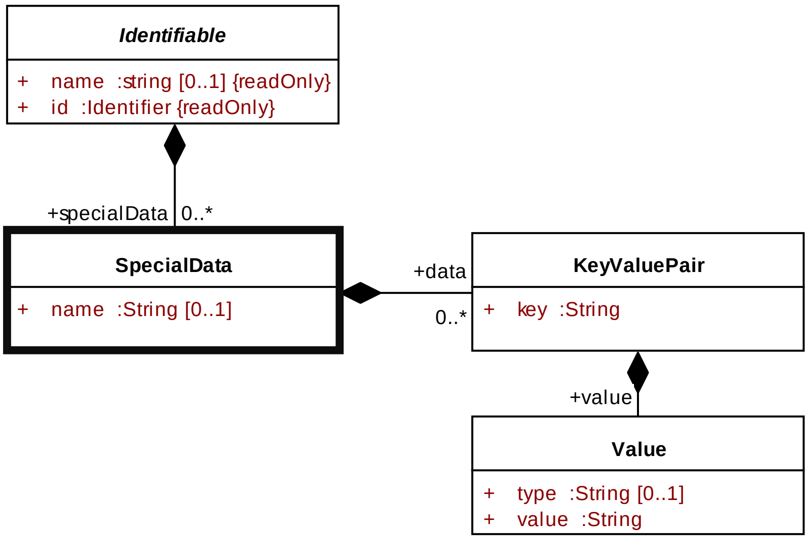


Figure 19 UML Diagram for class SpecialData

```
<xs:complexType name="special-data-type">
   <xs:sequence>
      <xs:element name="data"
                  type="key-value-pair-type"
                  minOccurs="0"
                  maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:string" use="optional"/>
</xs:complexType>
```

Figure 20 XML Schema for `special-data-type`

```
<optional-structural-variationpoint id="vp1" name="optional variationpoint">
   <special-data name="CreatorInfo">
      <data>
         <key>Created</key>
         <value type="xs:date">1998-11-17</value>
      </data>
   </special-data>
   <variation id="vp1v1" name="optional variation">
      <condition type="single-feature-condition">Feature1</condition>
   </variation>
</optional-structural-variationpoint>
```

Figure 21 XML Example for `special-data-type`

4.14.1 Description

The class SpecialData allows adding application specific information to VariationPoint and Variation objects. SpecialData aggregates a number of KeyValuePair elements which contain the actual information.

4.14.2 Attribute name

The attribute name of a SpecialData indicates which kind of data is contained in the SpecialData structure. The values of name are not standardized; it is highly recommended to use a descriptive name that has a high probability of being unique.

- The attribute name of a SpecialData is optional.
- Any application that deals with variability information read from an artifact via methods exportVariabilityExchangeModels or getConfiguration (see Section 4.17) shall not read or write the information contained in SpecialData if its name is unknown to the application.
- If an application reads variability information from an artifact via methods exportVariabilityExchangeModels or getConfiguration (see Section 4.17), then changes this information, and later uses the methods importVariabilityExchangeModels or setConfiguration (see Section 4.17) to write the information to an artifact, then any SpecialData whose type is not known to the application may be in an undefined state. This is because the information contained in SpecialData may depend on the overall structure.

4.15 StructuralVariationPoint `<structural-variationpoint-group>`

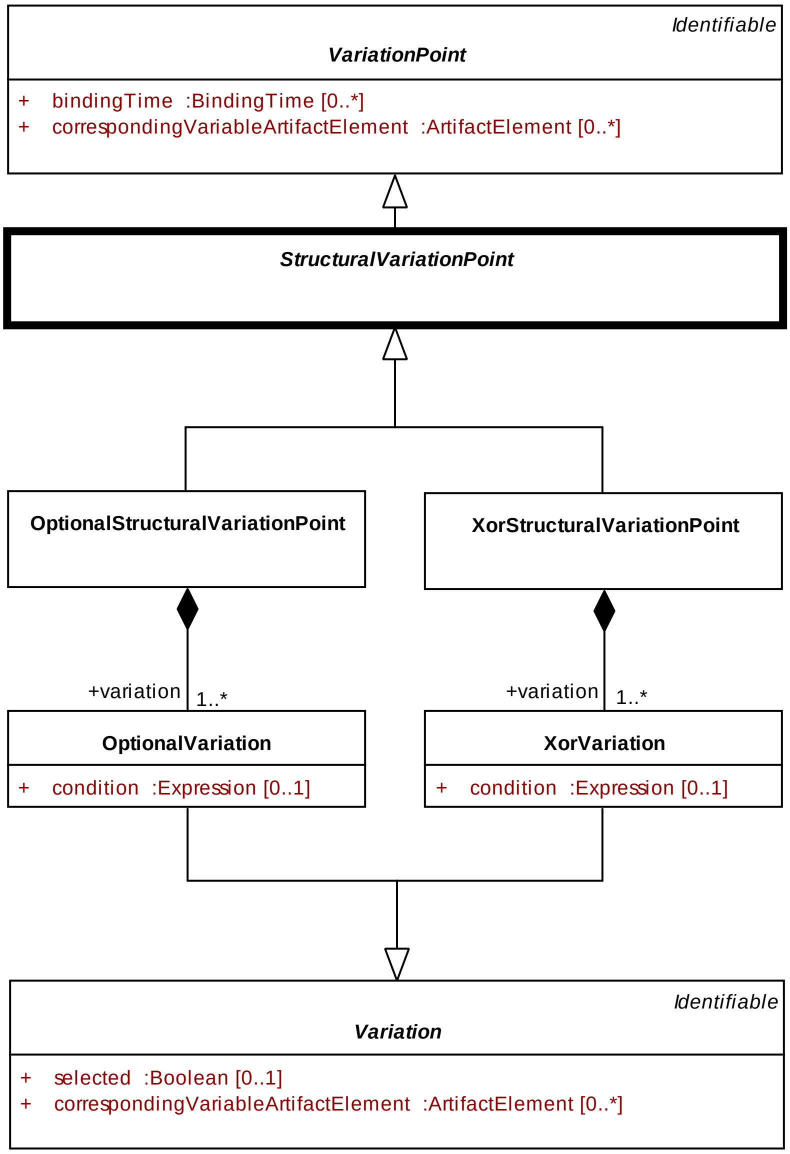

Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
StructuralVariationPoint
OptionalStructuralVariationPoint
XorStructuralVariationPoint
+variation
1..*
+variation
1..*
OptionalVariation
+ condition :Expression [0..1]
XorVariation
+ condition :Expression [0..1]
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 22 UML Diagram for class StructuralVariationPoint

```
<xs:group name="structural-variationpoint-group">
   <xs:choice>
      <xs:element name="optional-structural-variationpoint"
                  type="optional-structural-variationpoint-type"/>
      <xs:element name="xor-structural-variationpoint"
                  type="xor-structural-variationpoint-type"/>
   </xs:choice>
</xs:group>
```

Listing 25 XML Schema for `structural-variationpoint-group`

4.15.1 Description

A StructuralVariationPoint determines whether one or more elements in an artifact gets deleted or set inactive during the binding process.

The artifact elements are referenced by the attribute correspondingVariableArtifactElement of the StructuralVariationPoint and the attribute correspondingVariableArtifactElement of its Variations (see the classes VariationPoint and Variation in Figure 22)

4.15.2 Notes

- The class StructuralVariationPoint is an abstract class which inherits from the class VariationPoint.
- The class StructuralVariationPoint has two subclasses: OptionalStructuralVariationPoint and XorStructuralVariationPoint.
- Like ParameterVariationPoint, StructuralVariationPoint is implemented in the XML Schema as a group, not as a type. We choose to use a group here because a type would have established an extra XML element for StructuralVariationPoint, which would only have complicated the document structure.

4.16 ValueVariation `<value-variation-type>`

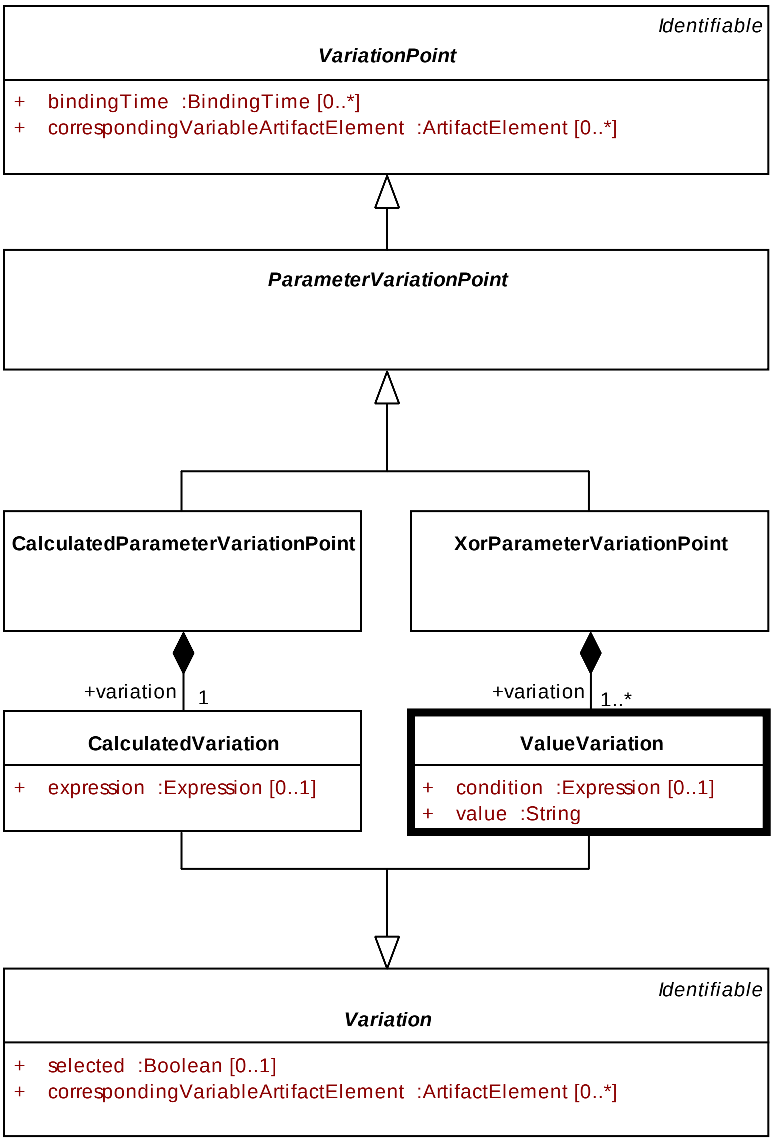

Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
ParameterVariationPoint
CalculatedParameterVariationPoint
XorParameterVariationPoint
+variation
1
+variation
1..*
CalculatedVariation
+ expression :Expression [0..1]
ValueVariation
+ condition :Expression [0..1]
+ value :String
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 23 UML Diagram for class ValueVariation

```
<xs:complexType name="value-variation-type">
   <xs:complexContent>
      <xs:extension base="variation-type">
         <xs:sequence>
            <xs:element name="condition"
                        type="expression-type"
                        minOccurs="0"
                        maxOccurs="1"/>
            <xs:element name="value" type="xs:string"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Listing 26 XML Schema for `value-variation-type`

```
<xor-parameter-variationpoint id="vp1">
   <variation id="vp1v1">
      <condition type="single-feature-condition">Feature1</condition>
      <value>1</value>
   </variation>
   <variation id="vp1v2">
      <condition type="single-feature-condition">Feature2</condition>
      <value>2</value>
   </variation>
   <variation id="vp1v3">
      <condition type="single-feature-condition">Feature3</condition>
      <value>3</value>
   </variation>
</xor-parameter-variationpoint>
```

Listing 27 XML Example for `value-variation-type`

4.16.1 Description

A ValueVariation selects a value for the corresponding artifact element of a XorParameterVariationPoint. The artifact element in question is referenced by its attribute correspondingVariableArtifactElement (see section 4.21.3).

Each XorParameterVariationPoint contains one or more ValueVariation objects. When a XorParameterVariationPoint gets bound, the attribute condition of each ValueVariation is evaluated. The condition may evaluate to $true$ for only one ValueVariation, and the attribute value of this ValueVariation is then used to provide a value for its correspondingVariableArtifactElement.

- Let v be a XorParameterVariationPoint which and let $s_1, \dots, s_n$ be the values of the attribute selected of the ValueVariations of v. Then the following conditions shall hold:
  1. $\exists i \in \{1, \dots, n\}: s_i = true$
  2. $\forall j \in \{1, \dots, n\}, i \neq j: s_i = false$

4.16.2 Attribute condition

- When evaluated, the attribute condition of a ValueVariation shall return a Boolean value. That is, its datatype attribute (if present) should be a Boolean or a data type which can be converted into a Boolean.
- Let $c_1, c_2, \dots, c_n$ be the conditions of all the ValueVariations that are contained in a given XorParameterVariationPoint. Then the following conditions shall hold
  1. $\exists j \in \{1, \dots, n\}: eval(c_j) = true$
  2. $\forall i \in \{1, \dots, n\}, i \neq j: eval(c_i) = false$
- If a ValueVariation has an attribute condition c and an attribute selected s (inherited from Variation), then the following condition shall hold:

$$eval(c) = s$$

4.16.3 Attribute value

- ➢ The attribute value of a ValueVariation is a constant, not an expression.
- ➢ The data type (e.g. Boolean, Integer, Floating Point, or an enumeration) and range (e.g. 1…10) that is allowed for the attribute value of a ValueVariaton is defined by the artifact element that is associated with ValueVariation (see correspondingVariableArtifactElement, section 4.21.3).

4.16.4 Binding

Each XorParameterVariationPoint contains one or more ValueVariation objects. When a XorParameterVariationPoint gets bound, the attribute condition of each ValueVariation is evaluated. The condition may evaluate to $true$ for only one ValueVariation, and the attribute value of this ValueVariation is then used to provide a value for its correspondingVariableArtifactElement.

- ➢ A XorParameterVariationPoint can only be bound when all its ValueVariations have a condition.

4.16.5 Notes

- The class ValueVariation inherits from the abstract class Variation.

4.17 VariabilityAPI

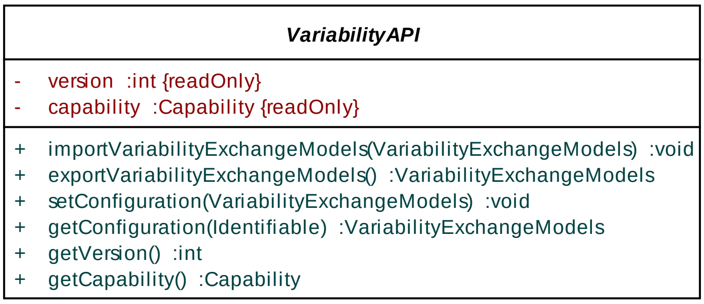

VariabilityAPI
- version :int {readOnly}
- capability :Capability {readOnly}
+ importVariabilityExchangeModels(VariabilityExchangeModels) :void
+ exportVariabilityExchangeModels() :VariabilityExchangeModels
+ setConfiguration(VariabilityExchangeModels) :void
+ getConfiguration(Identifiable) :VariabilityExchangeModels
+ getVersion() :int
+ getCapability() :Capability

Figure 24 UML Diagram for class VariabilityAPI

4.17.1 Description

The class VariabilityAPI defines the methods that are available for exchanging variability information through the *Variability Exchange Language*.

4.17.2 Attribute version

The attribute version documents the version of the variability language which is supported by this implementation of the *Variability Exchange Language*. It is obviously a read-only attribute.

- The attribute version shall be a positive integer.
- See the attribute version of the class VariabilityExchangeModels (section 4.19.2) for further constraints on this attribute.

4.17.3 Attribute capability

Not all implementations of the VariabilityAPI support all the methods that are shown in Figure 24. The attribute capability documents which of these methods – most importantly, importVariabilityExchangeModels, exportVariabilityExchangeModels, getConfiguration, and setConfiguration – are supported by this implementation of the VariabilityAPI.

4.17.4 Method importVariabilityExchangeModels

The method importVariabilityExchangeModels synchronizes all changes in the artifacts with the VariabilityExchangeModels structure. This means that new variation points may be introduced, and existing variation points in the artifact may me changed or deleted.

- Let $m_1, m_2, \dots, m_k$ be the VariabilityExchangeModel objects which are contained by the parameter of type VariabilityExchangeModels object which is the input to the method importVariabilityExchangeModels. Then for all m_i, the attribute type shall have the value VariationPointDescription.

- The method importVariabilityExchangeModels is only available if the attribute capability.setVariationPoints has the value $true$.

4.17.5 Method exportVariabilityExchangeModels

The method exportVariabilityExchangeModels reads information on the variation points in all available artifacts and returns a VariabilityExchangeModels structure.

- Let $m_1, m_2, \ldots, m_k$ be the VariabilityExchangeModel objects which are contained by the VariabilityExchangeModels object which is returned from the method exportVariabilityExchangeModels. Then for all m_i, the attribute type shall have the value VariationPointDescription .
- The method exportVariabilityExchangeModels is only available if the attribute capability.getVariationPoints is set to $true$.

4.17.6 Method getConfiguration

The method getConfiguration reads one or more variant configurations from an artifact.

- Let $m_1, m_2, \ldots, m_k$ be the VariabilityExchangeModel objects which are contained by the VariabilityExchangeModels object which is returned from the method getConfiguration. Then for all m_i, the attribute type shall have the value VariationPointConfiguration.
- The method getConfiguration is only available if the attribute capability.getConfiguration has the value $true$.

4.17.7 Method setConfiguration

The method setConfiguration writes one or more variant configurations to an artifact.

- Let $m_1, m_2, \ldots, m_k$ be the VariabilityExchangeModel objects which are contained by the parameter of type VariabilityExchangeModels object which is the input to the method setConfiguration. Then for all m_i, the attribute type shall have the value VariationPointConfiguration.
- The method setConfiguration is only available if the attribute capability.setConfiguration has the value $true$.

4.18 VariabilityExchangeModel `<variability-exchange-model-type>`

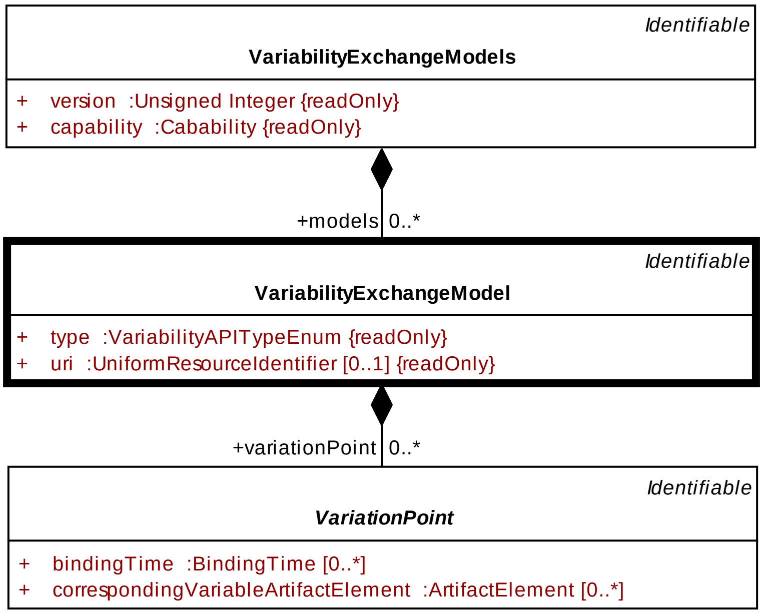

Identifiable
VariabilityExchangeModels
+ version :Unsigned Integer {readOnly}
+ capability :Cabability {readOnly}
+models
0..*
Identifiable
VariabilityExchangeModel
+ type :VariabilityAPITypeEnum {readOnly}
+ uri :UniformResourceIdentifier [0..1] {readOnly}
+variationPoint
0..*
Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 25 UML Diagram for class VariabilityExchangeModel

```
<xs:complexType name="variability-exchange-model-type">
    <xs:complexContent>
        <xs:extension base="identifiable-type">
            <xs:sequence>
                <xs:group ref="variationpoint-group"
                          minOccurs="0"
                          maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="type" type="variability-api-enum" use="required"/>
            <xs:attribute name="uri" type="xs:anyURI" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Listing 28 XML schema for `variability-exchange-model-type`

```
<?xml version="1.0" encoding="UTF-8"?>
<variability-exchange-models id="root"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:noNamespaceSchemaLocation="../VariabilityExchangeLanguage.xsd">
   <version>1</version>
   <capability>
      <import-variability-exchange-model>true</import-variability-exchange-model>
      <export-variability-exchange-model>true</export-variability-exchange-model>
      <get-configuration>true</get-configuration>
      <set-configuration>true</set-configuration>
   </capability>
   <variability-exchange-model type="variationpoint-description" id="model1"
                               uri="file:///C:/SPES/file1.c">
      …
   </variability-exchange-model>
   <variability-exchange-model type="variationpoint-description" id="model2"
                               uri="file:///C:/SPES/file2.c">
      …
   </variability-exchange-model>
   <variability-exchange-model type="variationpoint-description" id="model3"
                               uri="file:///C:/SPES/file3.c">
      …
   </variability-exchange-model>
   <variability-exchange-model type="variationpoint-description" id="model4"
                               uri="file:///C:/SPES/file4.c">
      …
   </variability-exchange-model>
</variability-exchange-models>
```

Listing 29 XML example for `variability-exchange-model-type`

4.18.1 Description

A VariabilityExchangeModel is an artifact which may contain variation points. Examples for artifacts are

- C/C++ files
- Matlab/Simulink Models
- DOORS databases

4.18.2 Attribute type

The attribute type of a VariabilitExchangeModel determines whether this model is a description of the variation points in the artifacts or defines a variant configuration:

- If the value of type is VariationPointDescription, then the attribute selected of all Variations (see section 4.21.2) and BindingTimes (see section 4.2.2) has no effect and shall be omitted.
- If the value of type is VariationPointConfiguration, then the attribute selected of all Variations (see section 4.21.2) and BindingTimes (see section 4.2.2) is not optional, and the attribute expression of CalculatedVariation must contain a constant.

See also section 4.21.2.

4.18.3 Attribute uri

The attribute uri of a VariabilityExchangeModel defines the Uniform Resource Locator (URI, see [3]) of the artifact that is associated with the VariabilityExchangeModel.

4.19 VariabilityExchangeModels `<variability-exchange-models-type>`

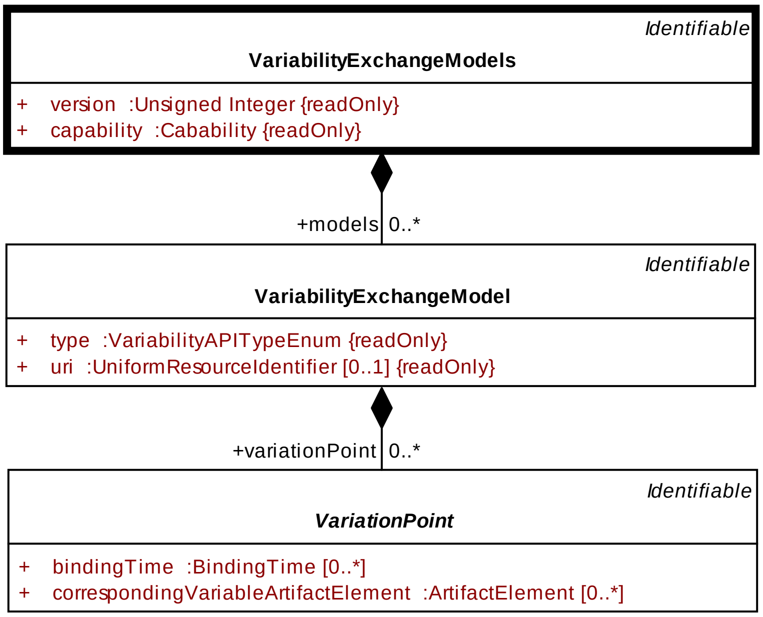

Identifiable
VariabilityExchangeModels
+ version :Unsigned Integer {readOnly}
+ capability :Cabability {readOnly}
+models 0..*
Identifiable
VariabilityExchangeModel
+ type :VariabilityAPITypeEnum {readOnly}
+ uri :UniformResourceIdentifier [0..1] {readOnly}
+variationPoint 0..*
Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 26 UML Diagram for class VariabilityExchangeModels

```
<xs:complexType name="variability-exchange-models-type">
   <xs:sequence>
      <xs:element name="version" type="xs:unsignedInt" fixed="1" />
      <xs:element name="capability" type="capability-type" />
      <xs:element name="variability-exchange-model"
                  type="variability-exchange-model-type"
                  minOccurs="0"
                  maxOccurs="unbounded" />
   </xs:sequence>
</xs:complexType>
```

Listing 30 XML Schema for `variability-exchange-models-type`

```
<?xml version="1.0" encoding="UTF-8"?>
<variability-exchange-models id="root"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:noNamespaceSchemaLocation="../VariabilityExchangeLanguage.xsd">
   <version>1</version>
   <capability>
      <import-variability-exchange-model>true</import-variability-exchange-model>
      <export-variability-exchange-model>true</export-variability-exchange-model>
      <get-configuration>true</get-configuration>
      <set-configuration>true</set-configuration>
   </capability>
   <variability-exchange-model type="variationpoint-description" id="model1">
      …
   </variability-exchange-model>
   <variability-exchange-model type="variationpoint-configuration" id="model2">
      …
   </variability-exchange-model>
</variability-exchange-models>
```

Listing 31 XML example for `variability-exchange-models-type`

4.19.1 Description

VariabilityExchangeModels is the top level object of a *Variability Exchange Language* document. In the XML representation, `variability-exchange-models` is the root element of the XML document object.

4.19.2 Attribute version

The attribute version of VariabilityExchangeModels defines the version of the *Variability Exchange Language* to which the *Variability Exchange Language* document conforms.

- The attribute version of VariabilityExchangeModels should be a positive non-zero Integer.
- If a specific implementation of the *Variability Exchange Language* supports version i and a *Variability Exchange Language* document is in version j, then the following conditions should hold:
  1. The implementation shall reject the document if $i < j$.
  2. The implementation shall accept the document if $i = j$.
  3. The implementation may accept the document if $i > j$.

In other words, an implementation of the *Variability Exchange Language* should never accept a document where the attribute version of the element VariabilityExchangeModels is a greater than the one that is supported by the implementation. It may, however accept a document with a smaller version number (backwards compatibility). Obviously, if both version numbers are equal, the document should be accepted[5].

- The attribute version of VariabilityExchangeModels is read-only.

4.19.3 Attribute cabability

The attribute capability of VariabilityExchangeModels defines which API operations (see section 4.16.5) are supported by the implementation of the *Variability Exchange Language* that created the *Variability Exchange Language* document.

For more information see the class Capability.

---

[5] Of course, the document might still be rejected later for another reason, for example a data type mismatch.

4.20 VariabilityAPITypeEnum `<variability-api-enum>`

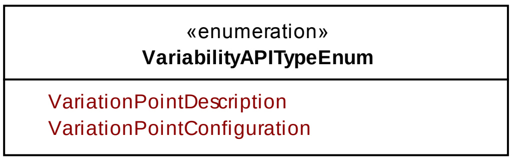


Figure 27 UML Diagram for class VariabilityAPITypeEnum

```
<xs:simpleType name="variability-api-enum">
   <xs:restriction base="xs:string">
      <xs:enumeration value="variationpoint-description"/>
      <xs:enumeration value="variationpoint-configuration"/>
   </xs:restriction>
</xs:simpleType>
```

Listing 32 XML Schema for `variability-api-type-enum`

```
<?xml version="1.0" encoding="UTF-8"?>
<variability-exchange-models id="root"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:noNamespaceSchemaLocation="../VariabilityExchangeLanguage.xsd">
   <version>1</version>
   <capability>
      <import-variability-exchange-model>true</import-variability-exchange-model>
      <export-variability-exchange-model>true</export-variability-exchange-model>
      <get-configuration>true</get-configuration>
      <set-configuration>true</set-configuration>
   </capability>
   <variability-exchange-model type="variationpoint-description" id="model1">
      …
   </variability-exchange-model>
   <variability-exchange-model type="variationpoint-configuration" id="model2">
      …
   </variability-exchange-model>
</variability-exchange-models>
```

Listing 33 XML example for `variability-api-type-enum`

4.20.1 Description

The enumeration VariabilityAPITypeEnum differentiates between the two flavors of VariabilityExchangeModel objects:

1. VariationPointDescription
2. VariationPointConfiguration

See the class VariabilityExchangeModel and section **Fehler! Verweisquelle konnte nicht gefunden werden.** for more details.

4.21 Variation `<variation-type>`

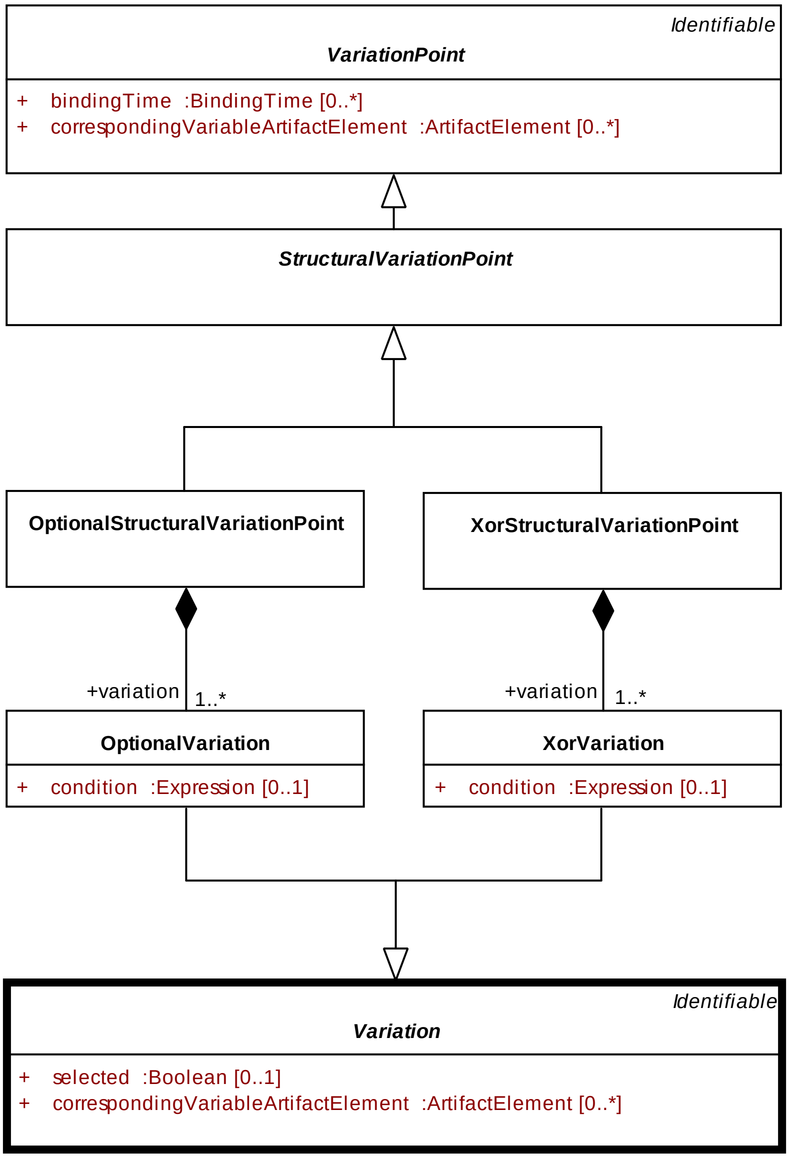

Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
StructuralVariationPoint
OptionalStructuralVariationPoint
XorStructuralVariationPoint
+variation
1..*
+variation
1..*
OptionalVariation
+ condition :Expression [0..1]
XorVariation
+ condition :Expression [0..1]
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 28 UML Diagram for class Variation

```
<xs:complexType name="variation-type" abstract="true">
   <xs:complexContent>
      <xs:extension base="identifiable-type">
         <xs:sequence>
            <xs:element name="hierarchy"
                        type="variationpoint-hierarchy-type"
                        minOccurs="0" maxOccurs="1"/>
            <xs:element name="depencency"
                        type="variation-dependency-type"
                        minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="corresponding-variable-artifact-element"
                        type="artifact-element-type"
                        minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="selected" type="xs:boolean" use="optional"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Figure 29 XML Schema for `variation-type`

4.21.1 Description

The abstract class Variation implements a variation of a variation point. Each instance of the class VariationPoint contains one or more instances of the class Variation.

There are four classes that derive from Variation, namely OptionalVariaton, XorVariation, CalculatedVariation and ValueVariation.

4.21.2 Attribute selected

If the attribute type of a VariabilityExchangeModel has the value VariationPointConfiguration, then the attribute selected of a Variation v decides wether v is contained in the variation point configuration which is defined by the VariabilityExchangeModel which contains v.

- If the attribute type of a VariabilityExchangeModel M has the value VariationPointDescription, then no Variation v in M shall have an attribute selected.
- If the attribute type of a VariabilityExchangeModel M has the value VariationPointConfiguration, then every Variation v in M shall have an attribute selected.
- If the attribute type of a VariabilityExchangeModel M has the value VariationPointConfiguration, and the attribute selected of a Variation v contained by M has the value $true$, then v is a member of the variation point configuration defined by M.
- If the attribute type of a VariabilityExchangeModel M has the value VariationPointConfiguration, and the attribute selected of a Variation v contained by M has the value $false$, then v is not a member of the variation point configuration defined by M.

4.21.3 Attribute correspondingVariableArtifactElement

The attribute correspondingVariableArtifactElement of a Variation v implements a reference to the artifact elements which correspond to v.

- The attribute correspondingVariableArtifactElement is optional.
- If a Variation v has more than one correspondingVariableArtifactElements $c_1, \dots, c_n$ then the URIs of $c_1, \dots, c_n$ do not need to point to the same artifacts. That is, the URI attributes of $c_1, \dots, c_n$ may have different values for each c_i.

4.21.4 Notes

- The class Variation inherits from the class Identifiable.
- The classes OptionalVariation, XorVariation, CalculatedVariation and ValueVariation inherit from Variation.

4.22 VariationPoint <variationpoint-type>

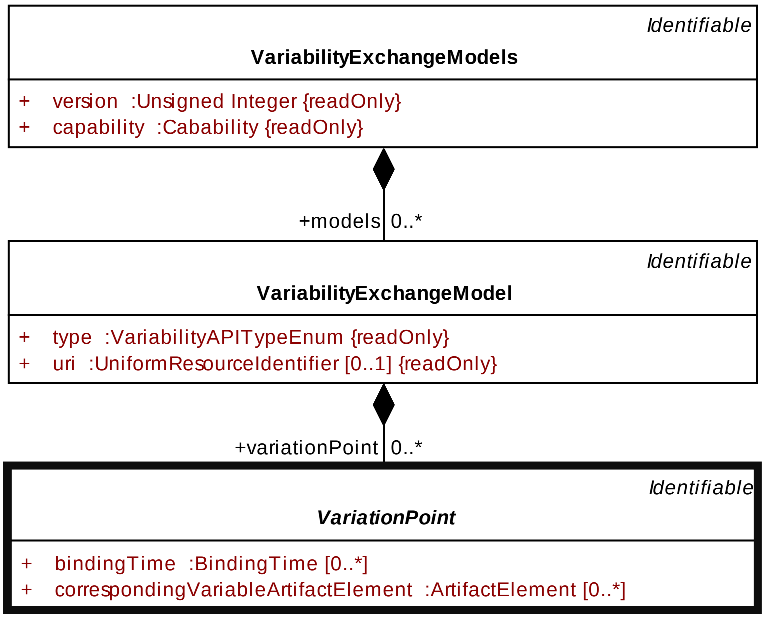

Identifiable
VariabilityExchangeModels
+ version :Unsigned Integer {readOnly}
+ capability :Cabability {readOnly}
+models
0..*
Identifiable
VariabilityExchangeModel
+ type :VariabilityAPITypeEnum {readOnly}
+ uri :UniformResourceIdentifier [0..1] {readOnly}
+variationPoint
0..*
Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 30 UML Diagram for class VariationPoint

```
<xs:complexType name="variationpoint-type" abstract="true">
   <xs:complexContent>
      <xs:extension base="identifiable-type">
         <xs:sequence>
            <xs:element name="bindingtime"
                        type="bindingtime-type"
                        minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="corresponding-variable-artifact-element"
                        type="artifact-element-type"
                        minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

<xs:group name="variationpoint-group">
   <xs:choice>
      <xs:group ref="structural-variationpoint-group"/>
      <xs:group ref="parameter-variationpoint-group"/>
   </xs:choice>
</xs:group>
```

Listing 34 XML Schema for `variationpoint-type`

4.22.1 Description

The abstract class VariationPoint describes a variationpoint in an artifact.

4.22.2 Attribute bindingTime

The attribute bindingTime defines the binding time of a VariationPoint. For more information on the concept of binding times, see section 4.2.

- If a VariationPoint does not declare a BindingTime, then it is up to the binding process to define which binding time to use. For example, a process that uses a single binding time may not state an explicit binding time for its variation points.
- A VariationPoint may define more than one binding time. In this case, the attribute selected of the BindingTime elements decides which binding time is used in the actual binding process.
- If the VariabilityExchangeModel M which contains a VariationPoint v has the type VariationPointConfiguration, then let $s_1, s_2, \dots, s_n$ be the values of the attribute selected of the BindingTime attributes of v. Then the following conditions must hold:

  1. $\exists i \in \{1, \dots, n\}: eval(s_i) = true$
  2. $\forall j \ \in \{1, \dots, n\}, j \neq i: eval\left(s_j\right) = false$

  A consequence of the above condition is that if a VariationPoint in a VariationPointConfiguration has only a single BindingTime attribute b, then the attribute selected of b shall have the value $true$.

  How and when a value for the attribute selected is determined is beyond the scope of this document.

4.22.3 Attribute correspondingVariableArtifactElement

The attribute correspondingVariableArtifactElement of a VariationPoint v implements a reference to the artifact elements which correspond to v.

- Not all VariationPoints have a correspondingVariableArtifactElement.
- If a VariationPoint v has more than one correspondingVariableArtifactElements $c_1, \dots, c_n$ then the URIs of $c_1, \dots, c_n$ do not need to point to the same artifacts. That is, the URI attributes of $c_1, \dots, c_n$ may have different values for each c_i.

4.22.4 Notes

- The class VariationPoint inherits from the class Identifiable.
- There classes StructuralVariationPoint and ParameterVariationPoint inherit from the class VariationPoint.

4.23 VariationPointHierarchy `<variationpoint-hierarchy-type>`

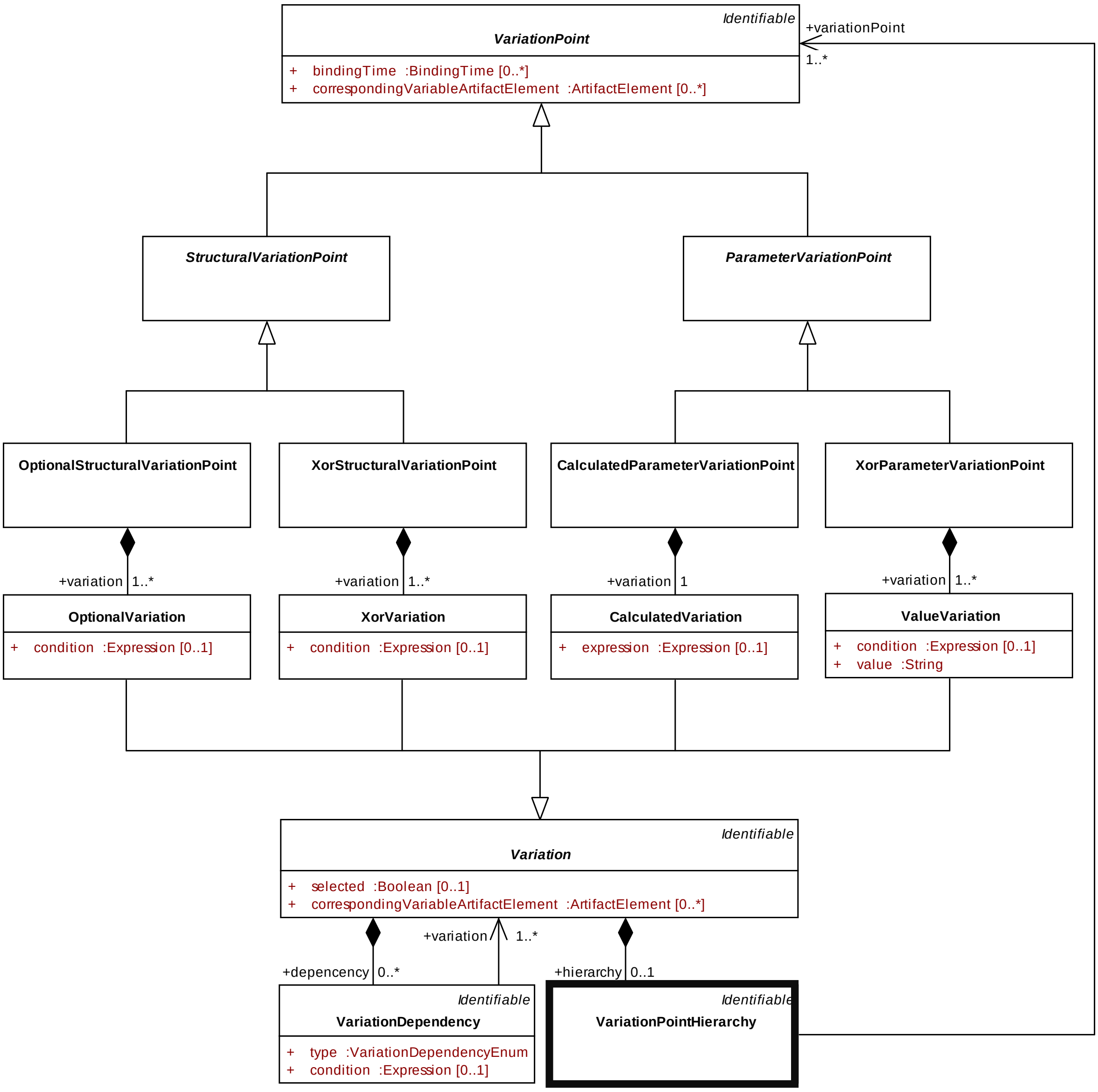

Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
+variationPoint
1..*
StructuralVariationPoint
ParameterVariationPoint
OptionalStructuralVariationPoint
XorStructuralVariationPoint
CalculatedParameterVariationPoint
XorParameterVariationPoint
+variation 1..*
+variation 1..*
+variation 1
+variation 1..*
OptionalVariation
+ condition :Expression [0..1]
XorVariation
+ condition :Expression [0..1]
CalculatedVariation
+ expression :Expression [0..1]
ValueVariation
+ condition :Expression [0..1]
+ value :String
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
+variation 1..*
+depencency 0..*
+hierarchy 0..1
Identifiable
VariationDependency
+ type :VariationDependencyEnum
+ condition :Expression [0..1]
Identifiable
VariationPointHierarchy

Figure 31 UML Diagram for class VariationPointHierarchy

```
<xs:complexType name="variationpoint-hierarchy-type">
   <xs:complexContent>
      <xs:extension base="identifiable-type">
         <xs:sequence>
            <xs:element name="variationpoint" minOccurs="1" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:attribute name="ref" type="xs:IDREF" use="required"/>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Listing 35 XML Schema for `variationpoint-hierarchy-type`

```
<variability-exchange-model type="variationpoint-description" id="model">
   <optional-structural-variationpoint id="vp1">
      <variation id="vp1v1">
         <condition type="single-feature-condition">Feature1</condition>
      </variation>
   </optional-structural-variationpoint>
   <optional-structural-variationpoint id="vp2">
      <variation id="vp2v1">
         <hierarchy id="vp2h1">
            <variationpoint ref="vp1"/>
         </hierarchy>
         <condition type="single-feature-condition">Feature1</condition>
      </variation>
   </optional-structural-variationpoint>
</variability-exchange-model>
```

Listing 36 XML Example for `variationpoint-hierarchy-type`

4.23.1 Description

Each Variation may contain a VariationPointHierarchy object. VariationPointHierarchy establishes a hierarchy among VariationPoints and Variations.

The hierarchy is a graph $G = (V, E)$ defined as follows:

- $V = \{v_1, v_2, \ldots, v_n\}$ where v_i is a VariationPoint in a *Variability Exchange Language* document[6].
- Let v_i be a VariationPoint which contains a Variation which contains a VariationPointHierarchy whose attribute ref refers to a VariationPoint v_j. Then $(v_i, v_j) \in E$.
- No two VariationPointHierarchy elements may refer to the same VariationPoints. Formally, the following condition shall hold: $(v_i, v_j) \in E \Rightarrow \forall k \neq i: (v_k, v_j) \notin E$.
- E must not contain circles, that is, there cannot be a sequence. Formally, the following condition shall hold: $(v_{i_1}, v_{i_2}), (v_{i_2}, v_{i_3}), \ldots, (v_{i_{k-2}}, v_{i_{k-1}}), (v_{i_{k-1}}, v_{i_k}) \in E$ with $i_1 = i_k$.

These conditions make sure that G is a tree or a set of trees.

4.23.2 Attribute variationPoint

The attribute variationPoint of a VariationPointHierarchy identifies the endpoint of a variationpoint hierarchy relation.

4.23.3 Notes

- The class VariationPointHierarchy inherits from Identifiable.

---

[6] Strictly speaking, V would be a set of nodes and there is bijective mapping between V and the set of elements of type `variationpoint-type` in the DOM of the Variability Exchange Language document. We use a simplified language for the sake of clarity here.

- In the XML Schema, the attribute variationPoint of VariationPointHierarchy is not implemented as a XML attribute but as a separate XML element named `variation` with an XML attribute `ref` that implements the actual reference. This is because variation has an upper multiplicity greater than one, but XML attributes are restricted to an upper multiplicity of 1.

4.24 VariationDependency `<variation-dependency-type>`

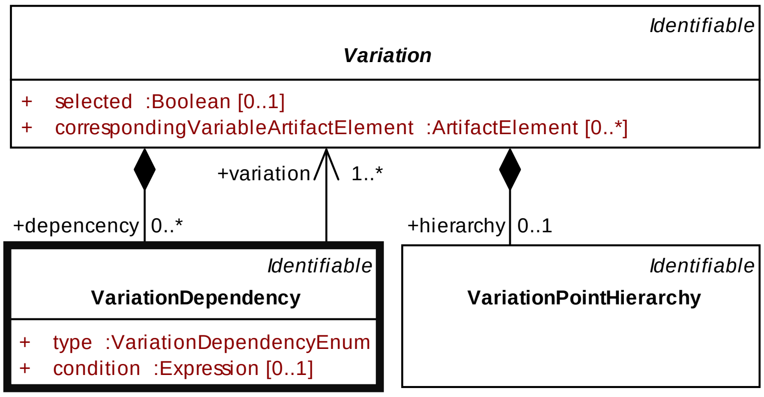

Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
+variation
1..*
+depencency
0..*
+hierarchy
0..1
Identifiable
VariationDependency
+ type :VariationDependencyEnum
+ condition :Expression [0..1]
Identifiable
VariationPointHierarchy

Figure 32 UML Diagram for class VariationDependenxy

```
<xs:complexType name="variation-dependency-type">
   <xs:complexContent>
      <xs:extension base="identifiable-type">
         <xs:sequence>
            <xs:element name="variation"
                        minOccurs="1" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:attribute name="ref" type="xs:IDREF" use="required"/>
               </xs:complexType>
            </xs:element>
            <xs:element name="condition"
                        type="expression-type"
                        minOccurs="0" maxOccurs="1"/>
         </xs:sequence>
         <xs:attribute name="type"
                       type="variation-dependency-enum"
                       use="required"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Listing 37 XML Schema for `variation-dependency-type`

```
<variability-exchange-model type="variationpoint-description" id="model">
   <optional-structural-variationpoint id="vp1">
      <variation id="vp1v1">
         <condition type="single-feature-condition">Feature1</condition>
      </variation>
   </optional-structural-variationpoint>
   <optional-structural-variationpoint id="vp2">
      <variation id="vp2v1">
         <depencency type="conflicts" id="vp2d1">
            <variation ref="vp1v1"/>
         </depencency>
         <condition type="single-feature-condition">Feature1</condition>
      </variation>
      <variation id="vp2v2">
         <condition type="single-feature-condition">Feature2</condition>
      </variation>
      <variation id="vp2v3">
         <condition type="single-feature-condition">Feature3</condition>
      </variation>
   </optional-structural-variationpoint>
</variability-exchange-model>
```

Listing 38 XML example for `variation-dependency-type`

4.24.1 Description

A VariationDependency defines a dependency between Variation objects. Each Variation may have an arbitrary number of dependencies on other Variations. There are two types of variations: requires and conflicts.

If a Variation aggregates more than one VariationDependency, then all dependencies must be fulfilled.

4.24.2 Attribute type

The attribute type of VariationDependency defines the type of a dependency. There are two types of dependencies:

- requires
- conflicts

The enumeration VariationDependencyEnum defines the values that are allowed for the attribute type.

4.24.3 Attribute variation

The attribute variation of a VariationDependency defines the target of a dependency.

4.24.4 Attribute condition

The optional attribute condition of a VariationDependency defines a condition under which the relation that is defined by the VariationDependency is effective.

4.24.5 Formal Definition

- Let v be a Variation which contains a VariationDependency d, and let $v_1, v_2, \ldots, v_k$ be the Variation to which the attribute variation of d refers, and let t be the value of the attribute type of d. Furthermore, let c be the content of the attribute condition of v.

  Then the condition of the VariationDependency d, $\mathrm{condition}(d)$ is defined as follows:

  - If the attribute type of d is requires, then

    $$\mathrm{condition}(d) = (v \Rightarrow (c \land (v_1 \lor v_2 \lor \ldots \lor v_k))$$

  - If the attribute type of d is conflicts, then

    $$\mathrm{condition}(d) = (v \Rightarrow (c \land (\neg v_1 \land \neg v_2 \land \ldots \land \neg v_k))$$

- Let $d_1, d_2, \ldots, d_n$ be the VariationDependency objects contained by a Variation v. Then the condition of v, $\mathrm{condition}(v)$ is defined as follows:

  $$\text{condition}(v) = \text{condition}(d_1) \land \text{condition}(d_2) \land \ldots \land \text{condition}(d_k)$$

- Let $c_1, c_2, \ldots, c_n$ be the conditions of all Variation objects in a *Variability Exchange Language*. Then the following condition shall hold:

  $$c_1 \land c_2 \land \ldots \land c_n$$

4.24.6 Notes

- The class VariationDependency inherits from Identifiable.
- In the XML Schema, the attribute variation is not implemented as a XML attribute but as a separate XML element named `variation` with an XML attribute `ref` that implements the actual reference. This is because variation has an upper multiplicity greater than one, but XML attributes are restricted to an upper multiplicity of 1 (that is, an XML element may not have multiple elements with the same name).

4.25 VariationDependencyEnum

`<variation-dependency-enum>`

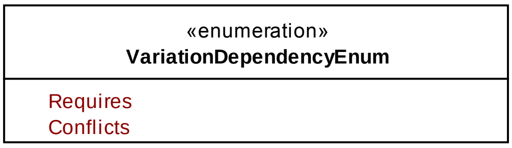


Figure 33 UML Diagram for class VariationDependenyEnum

```
<xs:simpleType name="variation-dependency-enum">
   <xs:restriction base="xs:string">
      <xs:enumeration value="requires"/>
      <xs:enumeration value="conflicts"/>
   </xs:restriction>
</xs:simpleType>
```

Listing 39 XML Schema for `variation-dependency-enum`

4.25.1 Description

The enumeration VariationDependencyEnum defines which values are allowed for the attribute type of VariationDependency. Currently, this enumeration defines two values:

- requires
- conflicts

For more information see VariationDependency.

4.26 XorParameterVariationPoint `<xor-parameter-variationpoint-type>`

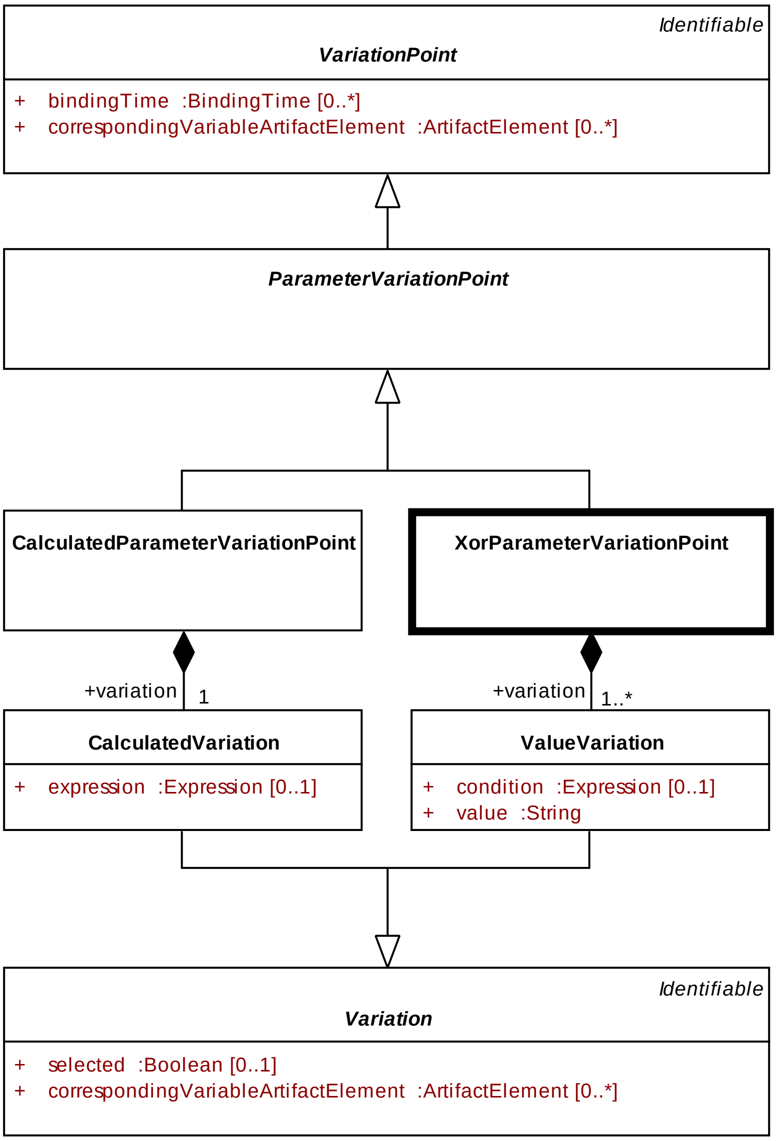

Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
ParameterVariationPoint
CalculatedParameterVariationPoint
XorParameterVariationPoint
+variation
1
+variation
1..*
CalculatedVariation
+ expression :Expression [0..1]
ValueVariation
+ condition :Expression [0..1]
+ value :String
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 34 UML Diagram for class XorParameterVariationPoint

```
<xs:complexType name="xor-parameter-variationpoint-type">
   <xs:complexContent>
      <xs:extension base="variationpoint-type">
         <xs:sequence>
            <xs:element name="variation"
                        type="value-variation-type"
                        maxOccurs="unbounded" />
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Listing 40 XML Schema for `xor-parameter-variationpoint-type`

```
<xor-parameter-variationpoint id="vp1">
   <variation id="vp1v1">
      <condition type="single-feature-condition">Feature1</condition>
      <value>1</value>
   </variation>
   <variation id="vp1v2">
      <condition type="single-feature-condition">Feature2</condition>
      <value>2</value>
   </variation>
   <variation id="vp1v3">
      <condition type="single-feature-condition">Feature3</condition>
      <value>3</value>
   </variation>
</xor-parameter-variationpoint>
```

Listing 41 XML Example for `xor-parameter-variationpoint-type`

4.26.1 Description

A XorParameterVariationPoint contains a number of ValueVariation objects. During the binding process, the attribute condition of each ValueVariation objects is evaluated. As described in section 4.16.2, it is guaranteed that if all ValueVariation objects have a condition attribute, then there is exactly one ValueVariation whose condition evaluates to $true$. The attribute value of this ValueVariation is then used to set the value of the associated artifact element.

4.26.2 Notes

- The class XorParameterVariationPoint inherits from the class ParameterVariationPoint.
- The class `XorParameterVariationPoint` is modelled after the switch statement in the programming languages C or Java; it selects a single value from a list of values. The difference is that a `switch` in C or Java first evaluates a Boolean expression and then compares the result to a list of constants, while XorParameterVariationPoint evaluates a list of Boolean expressions and selects the one which returns $true$.

4.27 XorStructuralVariationPoint `<xor-structural-variationpoint-type>`

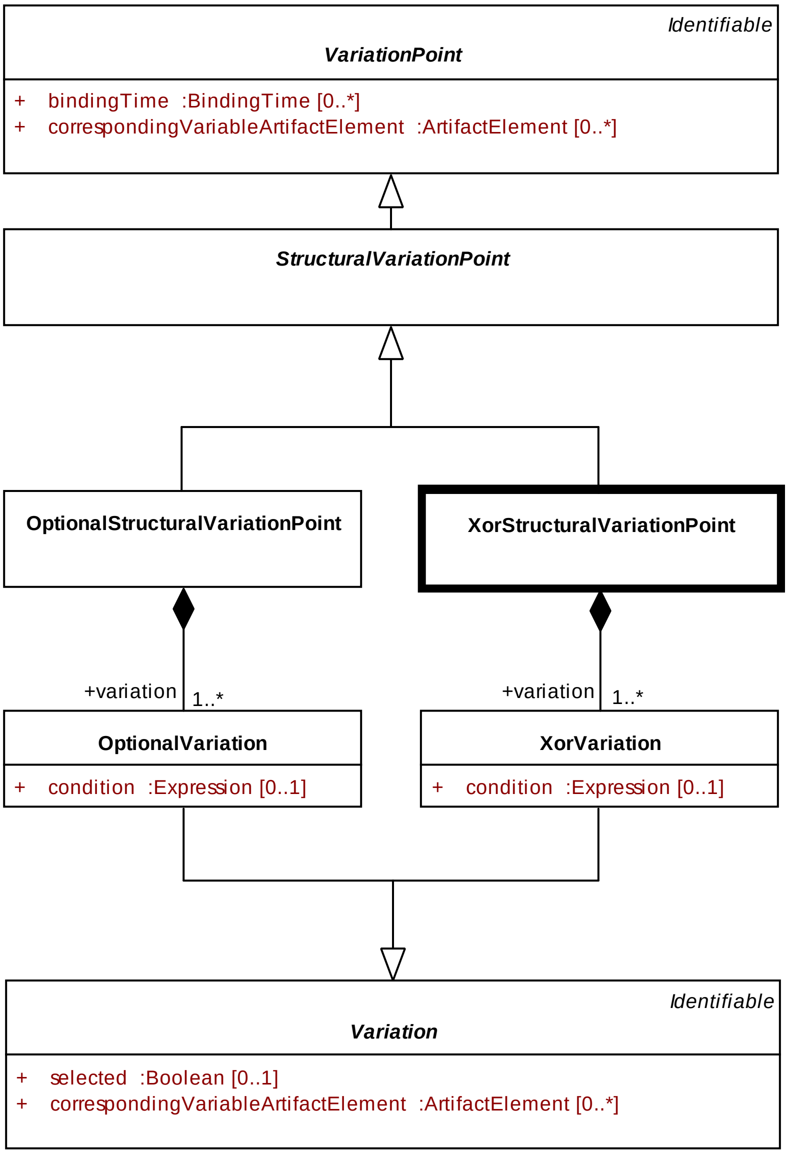

Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
StructuralVariationPoint
OptionalStructuralVariationPoint
XorStructuralVariationPoint
+variation
1..*
+variation
1..*
OptionalVariation
+ condition :Expression [0..1]
XorVariation
+ condition :Expression [0..1]
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 35 UML Diagram for XorStructuralVariationPoint

```
<xs:complexType name="xor-structural-variationpoint-type">
    <xs:complexContent>
        <xs:extension base="variationpoint-type">
            <xs:sequence maxOccurs="unbounded">
                <xs:element name="variation" type="xor-variation-type"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Listing 42 XML Schema for `xor-structural-variationpoint-type`

```
<xor-structural-variationpoint id="vp1">
   <variation id="vp1v1" name="Alternative 1">
      <condition type="single-feature-condition">Feature1</condition>
   </variation>
   <variation id="vp1v2" name="Alternative 2">
      <condition type="single-feature-condition">Feature2</condition>
   </variation>
   <variation id="vp1v3" name="Alternative 3">
      <condition type="single-feature-condition">Feature3</condition>
   </variation>
</xor-structural-variationpoint>
```

Listing 43 XML Example for `xor-structural-variationpoint-type`

4.27.1 Description

A XorStructuralVariationPoint contains one or more XorVariations. Its purpose is to choose exactly one of several alternative Variations.

During the binding process, the attribute condition is evaluated for each XorVariation. As described in section 4.28.2, it is guaranteed that if all XorVariation objects have a condition attribute, then exactly one of those conditions evaluates to $true$. The artifact element that corresponds to the XorVariation whose attribute condition evaluates to true is then removed or set inactive.

4.27.2 Notes

- The class XorStructuralVariationPoint inherits from the class StructuralVariationPoint.

4.28 XorVariation <xor-variation-type>

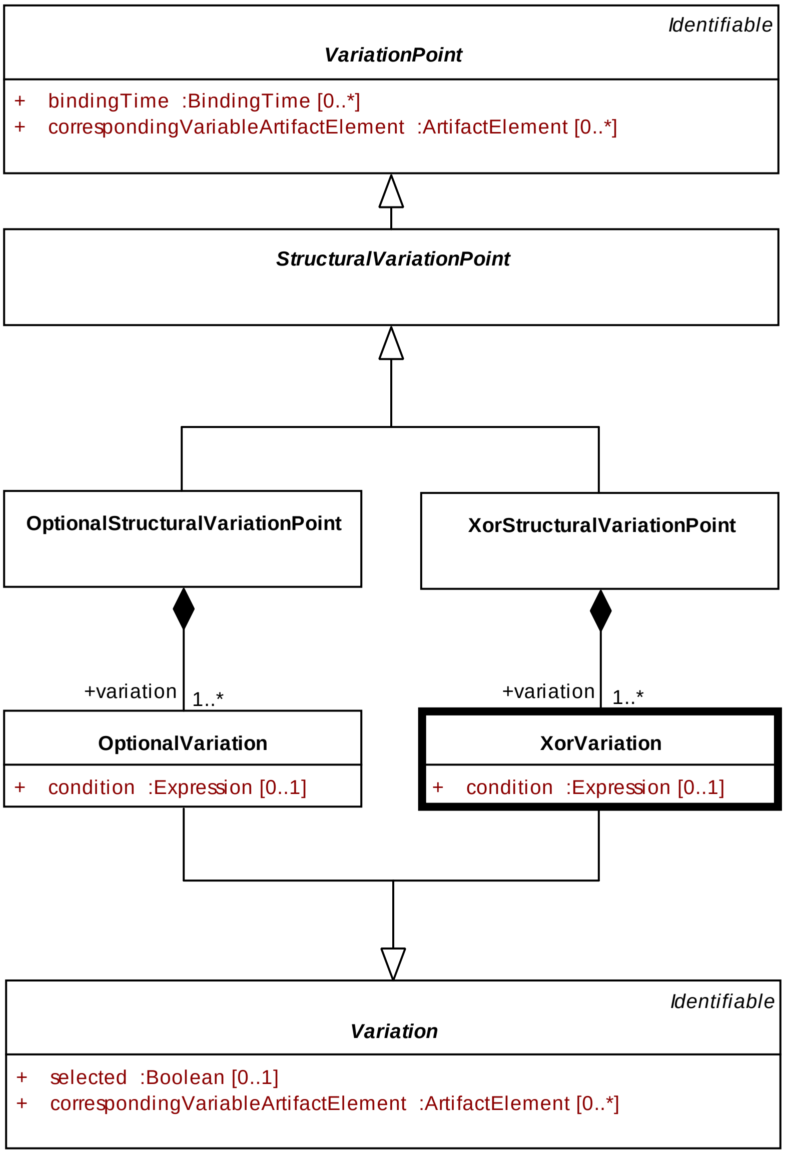

Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
StructuralVariationPoint
OptionalStructuralVariationPoint
XorStructuralVariationPoint
+variation
1..*
+variation
1..*
OptionalVariation
+ condition :Expression [0..1]
XorVariation
+ condition :Expression [0..1]
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]

Figure 36 UML Diagram for XorVariation

```
<xs:complexType name="xor-variation-type">
   <xs:complexContent>
      <xs:extension base="variation-type">
         <xs:sequence>
            <xs:element name="condition"
                        type="expression-type"
                        minOccurs="0"
                        maxOccurs="1"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Listing 44 XML Schema for `xor-variation-type`

```
<xor-structural-variationpoint id="vp1">
   <variation id="vp1v1" name="Alternative 1">
      <condition type="single-feature-condition">Feature1</condition>
   </variation>
   <variation id="vp1v2" name="Alternative 2">
      <condition type="single-feature-condition">Feature2</condition>
   </variation>
   <variation id="vp1v3" name="Alternative 3">
      <condition type="single-feature-condition">Feature3</condition>
   </variation>
</xor-structural-variationpoint>
```

Listing 45 XML Example for `xor-variation-type`

4.28.1 Description

A XorVariation is a kind of variation that choses Variation out of several alternative Variations.

- Let v be a XorStructuralVariationPoint which and let $s_1, \ldots, s_n$ be the values of the selected attributes of the XorVariations which are aggregated by v. Then the following conditions shall hold:
    1. $\exists i \in \{1, \ldots, n\}: s_i = true$
    2. $\forall j \in \{1, \ldots, n\}, i \neq j: s_i = false$

4.28.2 Attribute condition

- When evaluated, the attribute condition of a XorVariation shall return a *Boolean* value. That is, its datatype attribute (if present) should be a Boolean or a data type which can be converted into a Boolean.
- Let $c_1, c_2, \ldots, c_n$ be the conditions of all the XorVariations that are contained in a given XorStructuralVariationPoint. Then the following conditions must hold
    1. $\exists j \in \{1, \ldots, n\}: eval(c_j) = true$
    2. $\forall i \in \{1, \ldots, n\}, i \neq j: eval(c_i) = false$
- If a XorVariation has an attribute condition c and an attribute selected s (inherited from Variation), then the following condition shall hold:

$$eval(c) = s$$

4.28.3 Binding

Each XorStructuralVariationPoint contains one or more XorVariations. When a XorStructuralVariationPoint gets bound, the attribute condition of each XorVariation is evaluated. For only one XorVariation the condition shall evaluate to $true$. For all other XorVariations, the artifact elements that are referenced by its attribute correspondingVariableArtifactElement (see section 4.21.3) get deleted or set to inactive.

- A XorStructuralVariationPoint can only be bound when all its ValueVariations have a condition.

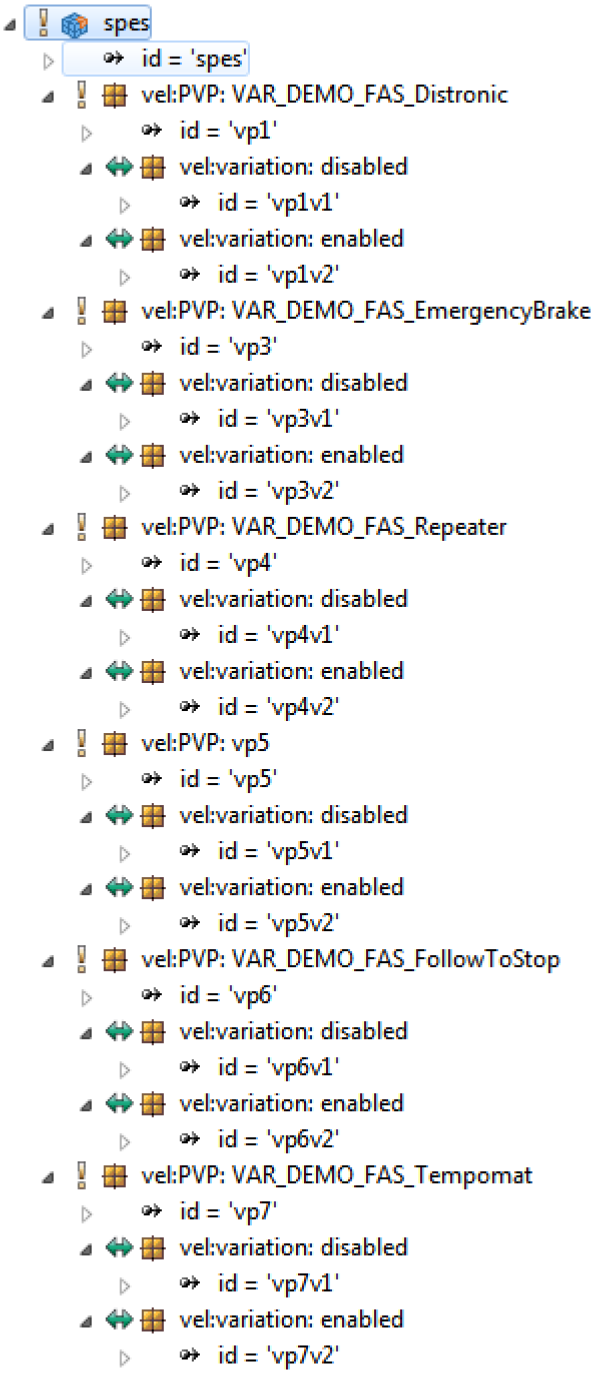
4.28.4 Notes

- The class XorVariation inherits from the class Variation.

5 Example: Importing into pure::variants

```xml
<?xml version="1.0" encoding="UTF-8"?>
<variability-exchange-models id="root" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../VariabilityExchangeLanguage.xsd">
    <version>1</version>
    <capability>
        <import-variability-exchange-model>true</import-variability-exchange-model>
        <export-variability-exchange-model>true</export-variability-exchange-model>
        <get-configuration>true</get-configuration>
        <set-configuration>true</set-configuration>
    </capability>
    <variability-exchange-model type="variationpoint-description" id="spes">
        <xor-parameter-variationpoint id="vp1" name="VAR_DEMO_FAS_Distronic">
            <variation id="vp1v1" name="disabled">
                <value>0</value>
            </variation>
            <variation id="vp1v2" name="enabled">
                <value>1</value>
            </variation>
        </xor-parameter-variationpoint>
        <xor-parameter-variationpoint id="vp3" name="VAR_DEMO_FAS_EmergencyBrake">
            <variation id="vp3v1" name="disabled">
                <value>0</value>
            </variation>
            <variation id="vp3v2" name="enabled">
                <value>1</value>
            </variation>
        </xor-parameter-variationpoint>
        <xor-parameter-variationpoint id="vp4" name="VAR_DEMO_FAS_Repeater">
            <variation id="vp4v1" name="disabled">
                <value>0</value>
            </variation>
            <variation id="vp4v2" name="enabled">
                <value>1</value>
            </variation>
        </xor-parameter-variationpoint>
        <xor-parameter-variationpoint id="vp5">
            <variation id="vp5v1" name="disabled">
                <value>0</value>
            </variation>
            <variation id="vp5v2" name="enabled">
                <value>1</value>
            </variation>
        </xor-parameter-variationpoint>
        <xor-parameter-variationpoint id="vp6" name="VAR_DEMO_FAS_FollowToStop">
            <variation id="vp6v1" name="disabled">
                <value>0</value>
            </variation>
            <variation id="vp6v2" name="enabled">
                <value>1</value>
            </variation>
        </xor-parameter-variationpoint>
        <xor-parameter-variationpoint id="vp7" name="VAR_DEMO_FAS_Tempomat">
            <variation id="vp7v1" name="disabled">
                <value>0</value>
            </variation>
            <variation id="vp7v2" name="enabled">
                <value>1</value>
            </variation>
        </xor-parameter-variationpoint>
    </variability-exchange-model>
</variability-exchange-models>
```

Listing 46 The Automotive SPES Demonstrator as a Variability Exchange Language document



```
spes
  id = 'spes'
  vel:PVP: VAR_DEMO_FAS_Distronic
    id = 'vp1'
    vel:variation: disabled
      id = 'vp1v1'
    vel:variation: enabled
      id = 'vp1v2'
  vel:PVP: VAR_DEMO_FAS_EmergencyBrake
    id = 'vp3'
    vel:variation: disabled
      id = 'vp3v1'
    vel:variation: enabled
      id = 'vp3v2'
  vel:PVP: VAR_DEMO_FAS_Repeater
    id = 'vp4'
    vel:variation: disabled
      id = 'vp4v1'
    vel:variation: enabled
      id = 'vp4v2'
  vel:PVP: vp5
    id = 'vp5'
    vel:variation: disabled
      id = 'vp5v1'
    vel:variation: enabled
      id = 'vp5v2'
  vel:PVP: VAR_DEMO_FAS_FollowToStop
    id = 'vp6'
    vel:variation: disabled
      id = 'vp6v1'
    vel:variation: enabled
      id = 'vp6v2'
  vel:PVP: VAR_DEMO_FAS_Tempomat
    id = 'vp7'
    vel:variation: disabled
      id = 'vp7v1'
    vel:variation: enabled
      id = 'vp7v2'
```

Figure 37 pure::variants model for the Automotive SPES Demonstrator from Listing 46.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<variability-exchange-models id="root" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../VariabilityExchangeLanguage.xsd">
    <version>1</version>
    <capability>
        <import-variability-exchange-model>true</import-variability-exchange-model>
        <export-variability-exchange-model>true</export-variability-exchange-model>
        <get-configuration>true</get-configuration>
        <set-configuration>true</set-configuration>
    </capability>
    <variability-exchange-model type="variationpoint-description" id="model1">
        <special-data name="Metatdata">
            <data>
                <key>CreationDate</key>
                <value type="xs:date">2014-10-29</value>
            </data>
        </special-data>
        <optional-structural-variationpoint id="vp1">
            <variation id="vp1v1">
                <condition type="single-feature-condition">Feature1</condition>
            </variation>
        </optional-structural-variationpoint>
        <xor-structural-variationpoint id="vp2">
            <corresponding-variable-artifact-element type="simulink">
                <simulink>42</simulink>
            </corresponding-variable-artifact-element>
            <variation id="vp2v1">
                <condition type="single-feature-condition">Feature2</condition>
            </variation>
            <variation id="vp2v2">
                <condition type="single-feature-condition">Feature3</condition>
            </variation>
        </xor-structural-variationpoint>
        <calculated-parameter-variationpoint id="vp3">
            <variation id="vp3v1">
                <expression type="pvscl-expression">6*9</expression>
            </variation>
        </calculated-parameter-variationpoint>
        <xor-parameter-variationpoint id="vp4">
            <variation id="vp4v1">
                <condition type="and-feature-condition">Feature10, Feature11</condition>
                <value>41</value>
            </variation>
            <variation id="vp4v2">
                <condition type="or-feature-condition">
                    Feature 12,Feature13
                </condition>
                <value>42</value>
            </variation>
            <variation id="vp4v3">
                <condition type="or-feature-condition">Feature14,Feature15</condition>
                <value>43</value>
            </variation>
        </xor-parameter-variationpoint>
        <optional-structural-variationpoint id="vp5">
            <variation id="vp5v1">
                <condition type="single-feature-condition">Feature1</condition>
            </variation>
        </optional-structural-variationpoint>
        <optional-structural-variationpoint id="vp6">
            <variation id="vp6v1">
                <hierarchy id="vp6h1">
                    <variationpoint ref="vp5"/>
                </hierarchy>
                <condition type="single-feature-condition">Feature2</condition>
            </variation>
        </optional-structural-variationpoint>
        <optional-structural-variationpoint id="vp7" name="Variationpoint7">
            <bindingtime selected="true">
                <name>compile-time</name>
            </bindingtime>
            <variation id="vp7v1" name="Variation 1 of Variationpoint7">
            <depencency type="requires" id="vp7v1d1">
                <variation ref="vp6v1"></variation>
            </depencency>
                <condition type="single-feature-condition">Feature7</condition>
            </variation>
        </optional-structural-variationpoint>
    </variability-exchange-model>
</variability-exchange-models>
```

Listing 47 A sample XML file illustrating various features of the Variability Exchange Language

6 Bibliography

[1] SPES EC5, „Begriffsdefinitionen für die SPES EC 5," 2013. [Online]. Available: https://svnbroy.informatik.tu-muenchen.de/spes_xt/EC_QT/EC5_WiederverwendungVarianten/2%20-%20Arbeitsdokumente/Begriffsdefinitionen%20&%20Glossar/EC5%20Begriffsdefinitionen.docx

[2] pure•systems GmbH, "pure::variants User's Guide," 2013. [Online]. Available: http://www.pure-systems.com/

[3] RFC3986, "Uniform Resource Identifier (URI): Generic Syntax" http://tools.ietf.org/html/rfc3986

[4] „Extensible Markup Language (XML) 1.1 (Second Edition)," 2006. [Online]. Available: http://www.w3.org/TR/2006/REC-xml11-20060816/.

[5] pure•systems GmbH, "PVSCL", 2013. [Online]. Available: http://www.pure-systems.com/.

[6] The Internet Engineering Task Force RCF 2119, https://www.ietf.org/rfc/rfc2119.txt

[7] Object Constraint Language, http://www.omg.org/spec/OCL/

[8] AUTOSAR, http://www.autosar.org/