Abstract:
The Variability Exchange Language (VEL) enables the exchange of variability information among tools for variant management tools and systems development tools. VEL eliminates the cost of building customized interfaces by defining a standard way for information to be exchanged among corresponding tools. Using VEL, a variant management tool is able to read the variability
from a development tool and pass configurations of selected system features to a development tool.

By defining a common variability data interface that can be implemented by both the development tools and the variant management tools, VEL enables a continuous development process for variable systems and more flexible use of tools.

Status:
This document was last revised or approved by the OASIS Variability Exchange Language (VEL) TC on the above date. The level of approval is also listed above. Check the “Latest stage” location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/vel/technical.

Technical Committee members should send comments on this specification to the TC’s email list. Others should send comments to the TC by using the “Send A Comment” button on the TC’s web page at https://www.oasis-open.org/committees/vel/.

This specification is provided under the Non-Assertion Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC’s web page (https://www.oasis-open.org/committees/vel/ipr.php).

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product’s prose narrative document(s), the content in the separate plain text file prevails.

Citation format:
When referencing this specification the following citation format should be used:

Notices

Copyright © OASIS Open 2019. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the “OASIS IPR Policy”). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an “AS IS” basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS’ procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see https://www.oasis-open.org/policies-guidelines/trademark for guidance.
# Table of Contents

1 Introduction ............................................................................................................................ 7
  1.1 Overview ............................................................................................................................ 7
    1.1.1 Section Prolog ........................................................................................................... 7
    1.1.2 Variants Management, System Variability, and Variation Points ......................... 7
    1.1.3 Variability View and Variants Management Tools .................................................. 8
  1.2 Terminology ....................................................................................................................... 9
    1.2.1 Key words .................................................................................................................. 9
    1.2.2 Definitions ................................................................................................................ 10
    1.2.3 Key concepts ............................................................................................................. 10
  1.3 Normative References ....................................................................................................... 10
  1.4 Non-Normative References .............................................................................................. 10

2 Overview of the Variability Exchange Language ................................................................. 11
  2.1 Section Prolog ................................................................................................................... 11
  2.2 VariationPoint and Variation ......................................................................................... 11
  2.3 Variation Point Descriptions versus Variation Point Selections ....................................... 12
  2.4 Binding ............................................................................................................................. 13
  2.5 Common Concepts ......................................................................................................... 13
  2.6 API ................................................................................................................................... 13
  2.7 Example ........................................................................................................................... 13

3 Variability Exchange Language Class Reference ............................................................... 16
  3.1 ArtifactElement .............................................................................................................. 16
    3.1.1 Description ............................................................................................................... 16
    3.1.2 Specification ........................................................................................................... 16
    3.1.3 XMLSerialization ................................................................................................. 16
  3.2 BindingTime ....................................................................................................................... 17
    3.2.1 Description ............................................................................................................ 17
    3.2.2 Specification ........................................................................................................... 18
    3.2.3 XMLSerialization ................................................................................................. 19
  3.3 BindingTimeEnum ............................................................................................................. 20
    3.3.1 Description ............................................................................................................ 20
    3.3.2 Specification ........................................................................................................... 20
    3.3.3 XMLSerialization ................................................................................................. 22
  3.4 Expression ........................................................................................................................ 22
    3.4.1 Description ............................................................................................................ 22
    3.4.2 Specification ........................................................................................................... 22
    3.4.3 XMLSerialization ................................................................................................. 23
  3.5 ExpressionTypeEnum ....................................................................................................... 24
    3.5.1 Description ............................................................................................................ 24
    3.5.2 Specification ........................................................................................................... 24
    3.5.3 XMLSerialization ................................................................................................. 25
  3.6 Identifiable ....................................................................................................................... 26
    3.6.1 Description ............................................................................................................ 26
    3.6.2 Specification ........................................................................................................... 27
    3.6.3 XMLSerialization ................................................................................................. 28
  3.7 KeyValuePair ..................................................................................................................... 29
    3.7.1 Description ............................................................................................................ 29
    3.7.2 Specification ........................................................................................................... 29
    3.7.3 XMLSerialization ................................................................................................. 30
  3.8 ParameterVariation ........................................................................................................... 31
    3.8.1 Description ............................................................................................................ 31
    3.8.2 Specification ........................................................................................................... 31
    3.8.3 XMLSerialization ................................................................................................. 32
  3.9 ParameterVariationPoint ................................................................................................. 32
    3.9.1 Description ............................................................................................................ 32
    3.9.2 Specification ........................................................................................................... 33
Appendixes

A An Annex ................................................................. 56
B A Normative Annex (Normative) .................................... 57
C A Non-normative Annex (Non-Normative) ...................................... 58
D An Informative Annex (Informative) ...................................... 59
E (normative) An ISO-normative Annex .................................... 60
F (informative) An ISO-informative Annex ................................... 61
1 Introduction

1.1 Overview

1.1.1 Section Prolog

VEL is an interoperability standard that enables the exchange of variability information among variant management tools and systems development tools. 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 VEL a common interface is defined that can be implemented by both the development tools and the variants management tool, thus VEL eliminates the cost of building customized interfaces by defining a standard way for information to be exchanged between tools.

1.1.2 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, and 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 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.

1.1.3 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.

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.
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, “Use case for the Variability Exchange Language”. 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.

1.2 Terminology

1.2.1 Key words

The key words must, must not, required, shall, shall not, should, should not, recommended, may, and optional are to be interpreted as described in [RFC 2119]. Note that for reasons of style, these words are not capitalized in this document.
1.2.2 Definitions

term
  Definition
term
  Definition

1.2.3 Key concepts

concept
  Definition
concept
  Definition
concept
  Definition

1.3 Normative References


1.4 Non-Normative References

TODO
2 Overview of the Variability Exchange Language

2.1 Section Prolog

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, “An Overview of the Variability Exchange Language”). In the following we immediately use the class names from the meta-model presented in Chapter Section 3, “Variability Exchange Language Class Reference” 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, “An Overview of the Variability Exchange Language”.

A detailed specification of all classes is provided in Chapter Section 3, “Variability Exchange Language Class Reference”.

Figure 2. An Overview of the Variability Exchange Language

<table>
<thead>
<tr>
<th>Identifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>VariabilityExchangeModels</td>
</tr>
<tr>
<td>+ version: Unsigned Integer {readOnly}</td>
</tr>
<tr>
<td>+ models 0..*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Identifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>VariabilityExchangeModel</td>
</tr>
<tr>
<td>+ type: VariabilityExchangeModelTypeEnum {readOnly}</td>
</tr>
<tr>
<td>+ uri: UniformResourceIdentifier [0..1] {readOnly}</td>
</tr>
</tbody>
</table>

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.

2.2 VariationPoint and Variation

As shown in Figure 2, “An Overview of the Variability Exchange Language”, we distinguish between two different kinds of VariationPoint:

VEL-v1.0-csprd01
Standards Track

Copyright © OASIS Open 2020. All rights reserved.
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 (variableArtifact), 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.

2.3 Variation Point Descriptions versus Variation Point Selections

A VariabilityExchangeModel as defined in Figure 2, “An Overview of the Variability Exchange Language” 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.
2.4 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 (ParameterVariation).

In a variation point description (see section Section 2.3, “Variation Point Descriptions versus Variation Point Selections”) 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.

2.5 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. Identifiable also provide a way to attach application-specific data (SpecialData) to elements in the Variability Exchange Language.

2.6 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.

2.7 Example

Figure 3. Example source code with C-preprocessor directives

```c
1. #if A
2.   /* code active if A is defined */
3. #if B
4.   /* code active if A and B is defined */
5. endif
6. /* code active if A is defined */
7. #else
8.   /* code active if A is not defined */
9. #endif
```

To demonstrate the applicability of the Variability Exchange Language for the exchange of variability information, we show as an example a simple source code section in Figure Figure 3, “Example
source code with C-preprocessor directives, in which the C-preprocessor (cpp) is employed to realize variability, and the extract of that variability defined according to Variability Exchange Language (see Figure 4, “Example source code with C-preprocessor directives”). We could have used further artifact types like requirements, UML models, tests, etc. but for the sake of simplicity and understandability we opted for C-source code using the cpp.

Figure 4. Example source code with C-preprocessor directives

```
1. ...
2. <variability-exchange-model type="variationpoint-description" id="model" url="file:///c/example.c">
3.  <xor-structural-variationpoint id="vp1">
4.   <corresponding-variable-artifact-element type="src-lines">
5.    <src-lines>1-9</src-lines>
6.   </corresponding-variable-artifact-element>
7.   <variation id="vp1v1"/>
8.   <condition type="single-feature-condition">A</condition>
9.   <corresponding-variable-artifact-element type="src-lines">
10.  <src-lines>2-6</src-lines>
11.  </corresponding-variable-artifact-element>
12.  </hierarchy id="vp2h1">
13.  <variationpoint ref="vp2"/>
14.  </hierarchy>
15.  </variation>
16.  <variation id="vp1v2">
17.   <corresponding-variable-artifact-element type="src-lines">
18.    <src-lines>8</src-lines>
19.   </corresponding-variable-artifact-element>
20.  </variation>
21.  </xor-structural-variationpoint>
22.  <optional-structural-variationpoint id="vp2">
23.   <corresponding-variable-artifact-element type="src-lines">
24.    <src-lines>3-5</src-lines>
25.   </corresponding-variable-artifact-element>
26.   <variation id="vp1v1"/>
27.   <condition type="single-feature-condition">B</condition>
28.   <corresponding-variable-artifact-element type="src-lines">
29.    <src-lines>4</src-lines>
30.   </corresponding-variable-artifact-element>
31.  </variation>
32.  </optional-structural-variationpoint>
33.  </variability-exchange-model>
```
To note, the cpp is a stand-alone tool for text processing, which, although initially invented for C, is not limited to a specific language and can be used for arbitrary text and source code transformations leading e.g. to conditional code compilation. The cpp tool works on the basis of directives (a.k.a. macros) that control syntactic program transformations. The directives supported by the cpp tool can be divided into four classes: file inclusion, macro definition, macro substitution, and conditional inclusion.

In our example, we only use the macros \texttt{A} and \texttt{B} and conditional inclusion mechanisms. The source code comments in Figure 3, “Example source code with C-preprocessor directives” explain how the cpp will transform the code depending on the definition of the macros. From an abstract point of view, the code contains two variation points and three variations, which is reflected according to the Variability Exchange Language in the exchange format definition in Figure 4, “Example source code with C-preprocessor directives”. The first variation point spans the source lines 1-9 and contains two alternative variations. The variation point’s corresponding elements of the artifact – in this case exactly the source lines – are represented in the exchange format as well. Regarding the variations, the first one (lines 2-6) will be selected if macro \texttt{A} is defined. Otherwise the second variation (line 8) gets selected. Assuming that the macro names are identically with features or at least there exists a mapping from a feature to a macro name, then the \textit{condition} in the eighth line in Figure 4, “Example source code with C-preprocessor directives” is the equivalent to the first source code line.

The [XMLmind] variation point (lines 3-5) is nested within the first variation of the first variation point, constituting a variation point hierarchy. Within the corresponding exchange format, the variation points are not nested but the nesting information is covered by the definition in the lines 15-17 in Figure 4, “Example source code with C-preprocessor directives”. There, the nested variation point is referenced by its \textit{id}, resulting in a tree-like structure at the end.
3 Variability Exchange Language Class Reference

3.1 ArtifactElement

3.1.1 Description

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

3.1.2 Specification

Figure 5. UML Diagram for class ArtifactElement

<table>
<thead>
<tr>
<th>ArtifactElement</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ type: String [0..1] {readOnly}</td>
</tr>
<tr>
<td>+ uri: UniformResourceIdentifier [0..1] {readOnly}</td>
</tr>
</tbody>
</table>

Attribute uri

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

The URI of an ArtifactElement should conform to the definition of Uniform Resource Locators as specified in ??? [FIXME].

Although the URI of ArtifactElement is optional, it is recommended to supply an URI instead of additional attributes whenever possible.

Attribute type

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

The attribute type is a string, so any user-defined type identifiers can be used.

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

3.1.3 XML Serialization

3.1.3.1 XML Schema

Figure 6. XML Schema for ArtifactElement

```xml
<xs:complexType name="ArtifactElement">
    <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>
```
In the XML schema, the type ArtifactElement allows arbitrary XML child elements. This is implemented by using the xs:any element (see Figure 6, “XML Schema for ArtifactElement”), 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.

### 3.1.3.2 Examples

**Example 2. “Example for ArtifactElement using artifact-specific XML elements”** shows a Variation, whose corresponding variable artifact element is a Simulink block with the Identifier 12.

**Example 1. Example for ArtifactElement using URIs**

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

**Example 2. Example for ArtifactElement using artifact-specific XML elements**

```xml
<structural-variationpoint id="vp2" type="optional">
  <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>
</structural-variationpoint>
```

### 3.2 BindingTime

#### 3.2.1 Description

The *binding time* of a variation point describes how the associated variability is resolved. 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.

---

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

**Figure 7. UML Diagram for class** BindingTime

<table>
<thead>
<tr>
<th>BindingTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ name: BindingTimeEnum</td>
</tr>
<tr>
<td>+ selected: Boolean [0..1]</td>
</tr>
<tr>
<td>+ condition: Expression [0..1]</td>
</tr>
</tbody>
</table>

**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 generation (binding time CodeGenerationTime, see Section 3.3, “BindingTimeEnum”) or just deactivated during startup (binding time PostBuild). 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 and let \( s_1, \ldots, s_n \) be the values of the selected attributes of the BindingTimes of \( v \). Then the following conditions shall hold:

  1. \( \exists i. (i \in \{1, \ldots, n\} \land s_i = \text{true}) \)
  2. \( \forall j. (j \in \{1, \ldots, n\} \land j \neq i \Rightarrow s_j = \text{false}) \)
  3. \( \exists i. (i \in \{1, \ldots, n\} \land s_i = \text{true} \land (\forall j. (j \in \{1, \ldots, n\} \land j \neq i \Rightarrow s_j = \text{false})) \)
- If a BindingTime has both an attribute selected \( s \) and an attribute condition \( c \), then the following condition shall hold:

\[
\text{eval}(c) = s
\]  

**Attribute name**

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

**Attribute condition**

[TODO: Text with formulas missing]

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 if a VariationPoint has more than one BindingTime.

See attribute selected for more information how condition is used to select a binding time.
3.2.3 XML Serialization

3.2.3.1 XML Schema

Figure 8. XML Schema for BindingTime

```xml
<xs:complexType name="BindingTime">
  <xs:sequence>
    <xs:element name="name" type="BindingTimeEnum"/>
    <xs:element name="condition" type="Expression"
      minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="selected" type="xs:boolean" use="optional"/>
</xs:complexType>
```

3.2.3.2 Examples

Example 3. XML Example for BindingTime in a variationpoint-description

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

Example 4. XML Example for BindingTime in a variationpoint-configuration

```xml
<variability-exchange-model type="variationpoint-configuration" id="model">
  <structural-variationpoint id="vp1" type="optional">
    <bindingtime>
      <name>preprocessor-time</name>
    </bindingtime>
    <variation id="vp1v1">
      <condition type="single-feature-condition">Feature1</condition>
    </variation>
  </structural-variationpoint>
  <structural-variationpoint id="vp2" type="optional">
    <bindingtime selected="false">
      VEL-v1.0-csprd01 13 November 2020
    </bindingtime>
  </structural-variationpoint>
</variability-exchange-model>
```
3.3 BindingTimeEnum

3.3.1 Description

The enumeration BindingTimeEnum defines the list of possible binding times to be used in BindingTime.

3.3.2 Specification

Figure 9. UML Diagram for enumeration BindingTimeEnum

<table>
<thead>
<tr>
<th>enumeration BindingTimeEnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequirementsTime</td>
</tr>
<tr>
<td>BluePrintDerivationTime</td>
</tr>
<tr>
<td>ModelConstructionTime</td>
</tr>
<tr>
<td>ModelSimulationTime</td>
</tr>
<tr>
<td>CodeGenerationTime</td>
</tr>
<tr>
<td>PreprocessorTime</td>
</tr>
<tr>
<td>CompileTime</td>
</tr>
<tr>
<td>LinkTime</td>
</tr>
<tr>
<td>FlashTime</td>
</tr>
<tr>
<td>PostBuild</td>
</tr>
<tr>
<td>PostBuildLoadable</td>
</tr>
<tr>
<td>PostBuildSelectable</td>
</tr>
<tr>
<td>RunTime</td>
</tr>
</tbody>
</table>

Following binding times can be used:

RequirementsTime

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

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.

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.
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.

CodeGenerationTime

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

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.

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.

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.

FlashTime

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

PostBuild

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

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.

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.

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.

User-defined binding times

If none of the binding times defined above are suitable, the user can define own binding times. For that, the user-defined binding-time identifier has to be prefixed with `x:`.
3.3.3 XML Serialization

3.3.3.1 XML Schema

Figure 10. XML Schema for bindingtime-enum

```xml
<xs:simpleType name="EnumerationExtension">
    <xs:restriction base="xs:string">
        <xs:pattern value="x:\S+/"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="BindingTimeBaseEnum">
    <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>
<xs:simpleType name="BindingTimeEnum">
    <xs:union memberTypes="BindingTimeBaseEnum EnumerationExtension"/>
</xs:simpleType>
```

[TODO: describe how extended enumeration is realized in schema]

3.4 Expression

3.4.1 Description

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

- "Genuine" expressions which may return any kind of value. These are represented by the type PVSCLExpression [FIXME].

- Constraints, which may only return Boolean values. These are represented by the SingleFeatureExpression, AndFeatureExpression and OrFeatureExpression. A constraint may also be of type PVSCLExpression [FIXME]; in this case the return value must be of type Boolean.

3.4.2 Specification

Figure 11. UML Diagram for class Expression

<table>
<thead>
<tr>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ type: ExpressionTypeEnum</td>
</tr>
<tr>
<td>+ datatype: String [0..1]</td>
</tr>
</tbody>
</table>

Technically, an Expression is a string whose syntax is determined by the attribute type.
An expression shall not be an empty string.

Attribute type

The attribute type defines the kind of expression. There are following predefined kinds of expressions:

- SingleFeatureCondition
- AndFeatureCondition
- OrFeatureCondition

Additionally, user-defined types can be used by prefixing the user-specific expression type identifier by x:. The individual expression types are explained in detail in Section 3.5, “ExpressionTypeEnum”.

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.

3.4.3 XML Serialization

3.4.3.1 XML Schema

Figure 12. XML Schema for Expression

```xml
<xs:complexType name="Expression">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="type" type="ExpressionLanguageEnum" use="required"/>
      <xs:attribute name="datatype" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

In the XML representation, the actual expression is contained in the inner text of the expression or condition element².

3.4.3.2 Examples

Example 5. XML Example for Expression

```xml
<structural-variationpoint id="vp1" type="xor">
  <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>
</structural-variationpoint>
```

²For simplicity and consistency, XML elements of type Expression are always named expression or condition.
3.5 ExpressionTypeEnum

3.5.1 Description

The enumeration ExpressionTypeEnum defines the possible values for the attribute type of the class Expression.

3.5.2 Specification

Figure 13. UML Diagram for class ExpressionTypeEnum

Following the semantics of the predefined kinds of expressions are described:

SingleFeatureCondition

A SingleFeatureCondition is a type of expression that models a Boolean condition whose literal is a single Feature.

The example in Example 5, “XML Example for Expression” translates to the Boolean expression

\[ Feature_1 \] (2)

Formally, if a SingleFeatureCondition references the feature \( f_i \) then this translates into the Boolean expression

\[ \text{eval}(f_i) \] (3)

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

The datatype for an Expression of type SingleFeatureCondition should be Boolean.

See also Section 3.5.3.2.2, “Syntax for SingleFeatureCondition” on how single features are represented in XML.
AndFeatureCondition

An AndFeatureCondition is a type of expression that models a Boolean condition whose literals are features, and which are connected by a Boolean AND. The example in Example 5, “XML Example for Expression” translates to the Boolean expression

\[ Feature_2 \land Feature_3 \land Feature_4 \] (4)

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 3.5.3.2.3, “Syntax for AndFeatureCondition and OrFeatureCondition” on how features are represented in XML.

OrFeatureCondition

An OrFeatureCondition is a type of expression that models a Boolean condition whose literals are features, and which are connected by a Boolean OR. The example in Example 5, “XML Example for Expression” translates to the Boolean expression

\[ Feature_5 \land Feature_6 \] (5)

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 3.5.3.2.3, “Syntax for AndFeatureCondition and OrFeatureCondition” on how features are represented in XML.

User-defined expression types

Additionally, user-defined expression types can be used by prefixing the user-specific expression type identifier by x:

[TODO]

3.5.3 XML Serialization

3.5.3.1 XML Schema

Figure 14. XML Schema for ExpressionTypeEnum

```xml
<xs:simpleType name="EnumerationExtension">
  <xs:restriction base="xs:string">
    <xs:pattern value="x:\S+"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ExpressionLanguageBaseEnum">
  <xs:restriction base="xs:string">
    <xs:enumeration value="single-feature-condition"/>
    <xs:enumeration value="and-feature-condition"/>
    <xs:enumeration value="or-feature-condition"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ExpressionLanguageEnum">
  <xs:union memberTypes="ExpressionLanguageBaseEnum EnumerationExtension"/>
</xs:simpleType>
```
3.5.3.2 Representation of expressions and features in XML

3.5.3.2.1 Features

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 representation, 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.

3.5.3.2.2 Syntax for SingleFeatureCondition

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 whose attribute type has the value single-feature-condition must match to the above pattern.

3.5.3.2.3 Syntax for AndFeatureCondition and OrFeatureCondition

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

```
\s*[a-zA-Z_][a-zA-Z0-9_]*,\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.

3.6 Identifiable

3.6.1 Description

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

Figure 15. UML Diagram for class Identifiable

| Identifiable
| + name: string [0..1] {readOnly}
| + id: Identifier {readOnly}

| SpecialData
| + name: String [0..1]

| KeyValuePair
| + key: String

| Value
| + type: String [0..1]
| + value: String

Attribute id

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

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 XML elements $e_1$ and $e_2$ with $i_1$ equals $i_2$. Then $e_1$ and $e_2$ are the same elements.

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

[FIXME] 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).

[FIXME] 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.

Attribute name

The attribute name of an Identifiable provides a human readable name for an element. It is recommended 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.
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.

### 3.6.3 XML Serialization

#### 3.6.3.1 XML Schema

**Figure 16. XML Schema for Identifiable**

```xml
<xs:complexType name="Identifiable" abstract="true">
  <xs:sequence>
    <xs:element name="special-data" type="SpecialData"
                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>
```

**Figure 17. Example use of Identifiable in the XML Schema**

```xml
<xs:complexType name="VariationPoint" abstract="true">
  <xs:complexContent>
    <xs:extension base="Identifiable">
      ...
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

In the XML Schema, Identifiable 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.

In XML representation, 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 element that is Identifiable. This is consistent with the semantics defined for id in Section 3.6.2, “Specification”.

#### 3.6.3.2 Examples

**Example 6. XML Example for Identifiable**

```xml
<structural-variationpoint id="vp1" name="optional variationpoint"
                           type="optional">
  <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">
    ...
  </variation>
</structural-variationpoint>
```
3.7 **KeyValuePair**

3.7.1 **Description**

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`.

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

3.7.2 **Specification**

*Figure 18. UML Diagram for class KeyValuePair*

**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 `KeyValuePair` objects whose attribute `key` have the same value.

**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.

**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. [FIXME]
3.7.3 XML Serialization

3.7.3.1 XML Schema

Figure 19. XML Schema for KeyValuePair

<xs:complexType name="KeyValuePair">
  <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>

As shown in Example 7, "XML Example for KeyValuePair", a key-value pair is implemented by the XML elements key and value, which are enclosed by a data element. 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.

3.7.3.2 Examples

Example 7. XML Example for KeyValuePair

<structural-variationpoint id="vp1" name="optional variationpoint" type="optional">
  <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>
</structural-variationpoint>

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

3.8.1 Description

A ParameterVariation defines a value for the corresponding artifact element of a ParameterVariationPoint. The artifact element in question is referenced by its attribute variableArtifact (see Attribute variableArtifact).

The value is defined either using a constant or a calculation Expression.

The class ParameterVariation inherits from the abstract class Variation.

3.8.2 Specification

Figure 20. UML Diagram for class ParameterVariation

```
<table>
<thead>
<tr>
<th>VariationPoint</th>
<th>Identifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ type: VariationPointTypeEnum</td>
<td></td>
</tr>
<tr>
<td>+ bindingTime: BindingTime [0..*]</td>
<td></td>
</tr>
<tr>
<td>+ variableArtifact: ArtifactElement [0..*]</td>
<td></td>
</tr>
</tbody>
</table>

ParameterVariationPoint

<table>
<thead>
<tr>
<th>ParameterVariation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ variation 1..*</td>
</tr>
</tbody>
</table>

ParameterVariation

| + expression: Expression [0..1] |
| + value: String |

Variation

```

Attribute expression

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

The attribute expression of a ParameterVariation may evaluate to an arbitrary value. Which values are allowed, depends on the artifact elements which are referenced by the attribute variableArtifact (see Attribute variableArtifact).

[TODO: Roles of expression in description, configuration, partial config]

Attribute value

The attribute value of a ParameterVariation 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 ParameterVariation is defined by the artifact element that is associated with ParameterVariation (see variableArtifact, Attribute variableArtifact).

[TODO: Roles of value in description, configuration, partial config]

### 3.8.3 XML Serialization

#### 3.8.3.1 XML Schema

Figure 21. XML Schema for `ParameterVariation`

```xml
<xs:complexType name="ParameterVariation">
    <xs:complexContent>
        <xs:extension base="Variation">
            <xs:sequence>
                <xs:element name="expression" type="Expression" minOccurs="0" maxOccurs="1"/>
                <xs:element name="value" type="xs:string" minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

#### 3.8.3.2 Examples

**Example 8. XML example for ParameterVariation with alternative constant values**

```xml
<parameter-variationpoint id="vp1" type="xor">
    <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>
</parameter-variationpoint>
```

**Example 9. XML example for ParameterVariation with a calculation expression**

```xml
<parameter-variationpoint id="vp1" type="optional">
    <variation id="vp1v1">
        <expression type="x:pvscl">6*9</expression>
    </variation>
</parameter-variationpoint>
```

### 3.9 ParameterVariationPoint

#### 3.9.1 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 class ParameterVariationPoint inherits from the class VariationPoint.

3.9.2 Specification

Figure 22. UML Diagram for class ParameterVariationPoint

<table>
<thead>
<tr>
<th>VariationPoint</th>
<th>Identifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ type: VariationPointTypeEnum</td>
<td></td>
</tr>
<tr>
<td>+ bindingTime: BindingTime [0..*]</td>
<td></td>
</tr>
<tr>
<td>+ variableArtifact: ArtifactElement [0..*]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ParameterVariationPoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ variation 1..*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ParameterVariation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ expression: Expression [0..1]</td>
</tr>
<tr>
<td>+ value: String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifiable</td>
</tr>
<tr>
<td>+ condition: Expression [0..1]</td>
</tr>
<tr>
<td>+ selected: Boolean [0..1]</td>
</tr>
</tbody>
</table>

The artifact elements are referenced by the attribute variableArtifact of the VariationPoint and the attribute variableArtifact of its Variation objects (see the classes VariationPoint and Variation in Figure 22, "UML Diagram for class ParameterVariationPoint")

Each ParameterVariationPoint object contains one or more ParameterVariation objects, representing possible values. During the binding process, the attribute condition of each ParameterVariation object is evaluated. The condition may evaluate to true for only one ParameterVariation:

[TODO: Is only xor allowed here?]

Let v be an ParameterVariationPoint and let \(s_1, \ldots, s_n\) be the values of the selected attributes of the ParameterVariation objects of v. Then the following conditions shall hold:

1. \(\exists i. (i \in \{1, \ldots, n\} \land s_i = \text{true})\)

2. \(\forall j. (j \in \{1, \ldots, n\} \land j \neq i \Rightarrow s_j = \text{false})\)
3. \( \exists i \in \{1, \ldots, n\} \land s_i = \text{true} \land \left( \forall j \in \{1, \ldots, n\} \land j \neq i \Rightarrow s_j = \text{false} \right) \) 

[TODO: Do we need this?] The class ParameterVariationPoint 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 ParameterVariationPoint evaluates a list of Boolean expressions and selects the one which returns true.

### 3.9.3 XML Serialization

#### 3.9.3.1 XML Schema

*Figure 23. XML Schema for ParameterVariationPoint*

```xml
<xs:complexType name="ParameterVariationPoint">
  <xs:complexContent>
    <xs:extension base="VariationPoint">
      <xs:sequence>
        <xs:element name="variation" type="ParameterVariation" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

#### 3.9.3.2 Examples

**Example 10. XML Example for ParameterVariationPoint with mutual exclusiv constant values**

```xml
<parameter-variationpoint id="vp1" type="xor">
  <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>
</parameter-variationpoint>
```

**Example 11. XML Example for ParameterVariationPoint with a calculation**

```xml
<parameter-variationpoint id="vp1" type="optional">
  <variation id="vp1v1">
    <expression type="x:pvscl">6*9</expression>
  </variation>
</parameter-variationpoint>
```
3.10 SpecialData

3.10.1 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.

3.10.2 Specification

Figure 24. UML Diagram for class `SpecialData`

![UML Diagram for SpecialData](image)

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.

[TOREMOVE?] Any application that deals with variability information read from an artifact via methods `exportVariabilityExchangeModels` or `getConfiguration` shall not read or write the information contained in `SpecialData` if its name is unknown to the application.

[TOREMOVE?] If an application reads variability information from an artifact via methods `exportVariabilityExchangeModels` or `getConfiguration`, then changes this information, and later uses the methods `importVariabilityExchangeModels` or `setConfiguration` 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.
3.10.3 XML Serialization

3.10.3.1 XML Schema

Figure 25. XML Schema for SpecialData

```
<xs:complexType name="SpecialData">
    <xs:sequence>
        <xs:element name="data" type="KeyValuePair"
            minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="optional"/>
</xs:complexType>
```

3.10.3.2 Examples

Example 12. XML Example for SpecialData

```
<structural-variationpoint id="vp1" name="optional variationpoint"
    type="optional">
    <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>
</structural-variationpoint>
```

3.11 Structural Variation

3.11.1 Description

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

The class StructuralVariation inherits from the class Variation.
3.11.2 Specification

Figure 26. UML Diagram for class StructuralVariation

Attribute `variableArtifact`

The attribute `variableArtifact` of a Variation \( v \) implements a reference to the artifact elements which correspond to \( v \).

The attribute `variableArtifact` is optional.

If a Variation \( v \) has more than one `variableArtifacts` \( c_1, \ldots, c_n \) the the URIs of \( c_1, \ldots, c_n \) do not need to point to the same artifacts. That is, the URI attributes of \( c_1, \ldots, c_n \) may have different values for each \( c_i \).
3.11.3 XML Serialization

3.11.3.1 XML Schema

Figure 27. XML Schema for \textit{StructuralVariationPoint}

\begin{verbatim}
<xs:complexType name="StructuralVariation">
  <xs:complexContent>
    <xs:extension base="Variation">
      <xs:sequence>
        <xs:element name="variable-artifact"
          type="ArtifactElement"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
\end{verbatim}

3.11.3.2 Examples

TODO

3.12 StructuralVariationPoint

3.12.1 Description

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

The class \textit{StructuralVariationPoint} inherits from the class \textit{VariationPoint}.
3.12.2 Specification

Figure 28. UML Diagram for class StructuralVariationPoint

The artifact elements are referenced by the attribute variableArtifact of the VariationPoint and the attribute variableArtifact of its Variation objects (see the classes VariationPoint and Variation in Figure 28, “UML Diagram for class StructuralVariationPoint”)

Each StructuralVariationPoint object contains one or more StructuralVariation objects. During the binding process, the attribute condition is evaluated for each StructuralVariation. As described in Section 3.16, “Variation”, it is guaranteed that if all StructuralVariation objects have a condition attribute, then, depending on the type of the VariationPoint, a) any number, b) at least one, or c) exactly one of those conditions evaluates to true. The artifact elements that correspond to the StructuralVariation, whose attribute condition evaluates to false, is then removed or set inactive.
3.12.3 XML Serialization

3.12.3.1 XML Schema

Figure 29. XML Schema for StructuralVariationPoint

```xml
<xs:complexType name="StructuralVariationPoint">
  <xs:complexContent>
    <xs:extension base="VariationPoint">
      <xs:sequence>
        <xs:element name="variation" type="StructuralVariation"
                    minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

3.12.3.2 Examples

Example 13. XML Example for StructuralVariationPoint with one optional variation

```xml
<structural-variationpoint id="vp1" type="optional">
  <variation id="vp1v1">
    <condition type="single-feature-condition">Feature1</condition>
  </variation>
</structural-variationpoint>
```

Example 14. XML Example for StructuralVariationPoint with multiple optional variations

```xml
<structural-variationpoint id="vp2" type="optional">
  <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>
</structural-variationpoint>
```

Example 15. XML Example for StructuralVariationPoint with mutual exclusive variations

```xml
<structural-variationpoint id="vp1" type="xor">
  <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>
</structural-variationpoint>
```
3.13 VariabilityExchangeModel

3.13.1 Description

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

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

3.13.2 Specification

Figure 30. UML Diagram for class VariabilityExchangeModel

<table>
<thead>
<tr>
<th>VariabilityExchangeModel</th>
<th>Identifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ version: Unsigned Integer {readOnly}</td>
<td></td>
</tr>
</tbody>
</table>

+ models 0..*

<table>
<thead>
<tr>
<th>VariabilityExchangeModel</th>
<th>Identifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ type: VariabilityExchangeModelTypeEnum {readOnly}</td>
<td></td>
</tr>
<tr>
<td>+ uri: UniformResourceIdentifier [0..1] {readOnly}</td>
<td></td>
</tr>
</tbody>
</table>

Attribute **type**

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

- If the value of **type** is VariationPointDescription, then the attribute **selected** of all Variations (see Attribute **selected**) and BindingTimes (see Section 3.2, “BindingTime”) has no effect and shall be omitted.

- If the value of **type** is VariationPointConfiguration, then the attribute **selected** of all Variations (see Attribute **selected**) and BindingTimes (see Section 3.2, “BindingTime”) is not optional, and the attribute **expression** of [FIXME] CalculatedVariation must contain a constant.

- If the value of **type** is VariationPointPartialConfiguration, then [TODO]

See also Attribute **selected**.

Attribute **uri**

The attribute **uri** of a VariabilityExchangeModel defines the Uniform Resource Locator (URI, see (???)) of the artifact that is associated with the VariabilityExchangeModel.
3.13.3 XML Serialization

3.13.3.1 XML Schema

Figure 31. XML Schema for VariabilityExchangeModel

```xml
<xs:complexType name="VariabilityExchangeModel">
  <xs:complexContent>
    <xs:extension base="Identifiable">
      <xs:sequence>
        <xs:group ref="variationpoint-group"
          minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="type"
          type="VariabilityExchangeModelTypeEnum" use="required"/>
        <xs:attribute name="uri" type="xs:anyURI" use="optional"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

3.13.3.2 Examples

TODO

3.14 VariabilityExchangeModels

3.14.1 Description

VariabilityExchangeModels is the top level object of a Variability Exchange Language document.

3.14.2 Specification

Figure 32. UML Diagram for class VariabilityExchangeModels

<table>
<thead>
<tr>
<th>VariabilityExchangeModels</th>
<th>Identifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ version: Unsigned Integer (readOnly)</td>
<td></td>
</tr>
</tbody>
</table>

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). If both version numbers are equal, the document should be accepted\(^5\).

The attribute `version` of `VariabilityExchangeModels` is read-only.

### 3.14.3 XML Serialization

#### 3.14.3.1 XML Schema

**Figure 33. XML Schema for VariabilityExchangeModels**

```xml
<xs:complexType name="VariabilityExchangeModels">
  <xs:complexContent>
    <xs:extension base="Identifiable">
      <xs:sequence>
        <xs:element name="version" type="xs:unsignedInt"/>
        <xs:element name="variability-exchange-model"
                    type="VariabilityExchangeModel"
                    minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

In the XML representation, `VariabilityExchangeModels` is the root element of the XML document object.

#### 3.14.3.2 Examples

TODO

### 3.15 VariabilityExchangeModelTypeEnum

#### 3.15.1 Description

The enumeration `VariabilityExchangeModelTypeEnum` differentiates between the three flavors of `VariabilityExchangeModel` objects:

1. VariationPointDescription
2. VariationPointConfiguration
3. VariationPointPartialConfiguration

See the class `VariabilityExchangeModel` for more details.

---

\(^{5}\)Of course, the document might still be rejected later for another reason, for example a data type mismatch.
3.15.2 Specification

Figure 34. UML Diagram for class `VariabilityExchangeModelTypeEnum`

<table>
<thead>
<tr>
<th>«enumeration»</th>
<th><code>VariabilityExchangeModelTypeEnum</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>VariationPointDescription</td>
<td></td>
</tr>
<tr>
<td>VariationPointConfiguration</td>
<td></td>
</tr>
<tr>
<td>VariationPointPartialConfiguration</td>
<td></td>
</tr>
</tbody>
</table>

3.15.3 XML Serialization

3.15.3.1 XML Schema

Figure 35. XML Schema for `VariabilityExchangeModelTypeEnum`

```xml
<xs:simpleType name="VariabilityExchangeModelTypeEnum">
  <xs:restriction base="xs:string">
    <xs:enumeration value="variationpoint-description"/>
    <xs:enumeration value="variationpoint-configuration"/>
    <xs:enumeration value="variationpoint-partial-configuration"/>
  </xs:restriction>
</xs:simpleType>
```

3.15.3.2 Examples

TODO

3.16 Variation

3.16.1 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 two classes that derive from `Variation`, namely `StructuralVariation` and `ParameterVariation`.

The class `Variation` inherits from the class `Identifiable`.

3.16.2 Specification

Figure 36. UML Diagram for class `Variation`

<table>
<thead>
<tr>
<th><code>Variation</code></th>
<th><code>Identifiable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>+ condition: Expression [0..1]</td>
<td></td>
</tr>
<tr>
<td>+ selected: Boolean [0..1]</td>
<td></td>
</tr>
</tbody>
</table>

Attribute `selected`

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 VariationPointPartialConfiguration, then every Variation \( v \) in \( M \) may have an attribute selected.

If the attribute type of a VariabilityExchangeModel \( M \) has the value VariationPointConfiguration or VariationPointPartialConfiguration, and the attribute selected of a Variation \( v \) contained by \( M \) has the value \( \text{true} \), 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 or VariationPointPartialConfiguration, and the attribute selected of a Variation \( v \) contained by \( M \) has the value \( \text{false} \), then \( v \) is not a member of the variation point configuration defined by \( M \).

If the attribute type of a VariabilityExchangeModel \( M \) has the value VariationPointPartialConfiguration, and the attribute selected of a Variation \( v \) contained by \( M \) is not set, then \( v \) is potentially a member of the variation point configuration defined by \( M \).

**Attribute condition**

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

When evaluated, the attribute condition of a Variation 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 a Variation has an attribute condition \( c \) and an attribute selected \( s \), then the following condition shall hold:

\[
eval(c) = s
\]  

[TODO: formulas missing, see 4.16.2] Let \( ??? \) be the conditions of all the Variation objects that are contained in a given VariationPoint. Then the following conditions must hold

1. \( ??? \)
2. \( ??? \)

If a Variation has an attribute condition \( ??? \) and an attribute selected \( ??? \), then the following condition shall hold:

[TODO \( \text{eval}(c) = s \) ]
3.16.3 XML Serialization

3.16.3.1 XML Schema

Figure 37. XML Schema for Variation

```xml
<xs:complexType name="Variation" abstract="true">
  <xs:complexContent>
    <xs:extension base="Identifiable">
      <xs:sequence>
        <xs:element name="hierarchy" type="VariationPointHierarchy"
                     minOccurs="0" maxOccurs="1"/>
        <xs:element name="dependency" type="VariationDependency"
                     minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="condition" type="Expression"
                     minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="selected" type="xs:boolean"
                     use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

3.17 VariationDependency

3.17.1 Description

A VariationDependency defines a dependency between Variation objects. Each Variation may have an arbitrary number of dependencies to other Variations. There are two predefined types of variations: requires and conflicts. Additionally, user-defined types can be used.

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

The class VariationDependency inherits from Identifiable.

3.17.2 Specification

Figure 38. UML Diagram for class VariationDependency

Attribute `type`

The attribute `type` of VariationDependency defines the type of a dependency. There are two predefined types of dependencies:
• requires
• conflicts

Additionally, used-defined dependency types can be used, by prefixing any user-defined identifier with \textit{x:}.

The enumeration \texttt{VariationDependencyEnum} defines the values that are allowed for the attribute \texttt{type}.

\textbf{Attribute variation}

The \textit{attribute variation} of a \texttt{VariationDependency} defines the target of a dependency.

\textbf{Attribute condition}

The optional attribute \textit{condition} of a \texttt{VariationDependency} defines a condition under which the relation that is defined by the \texttt{VariationDependency} is effective.

Formally, the semantics of a variation dependency is defined as follows:

Let \( v \) be a \texttt{Variation}, which contains a \texttt{VariationDependency} \( d \), and let \( v_1, \ldots, v_k \) be the \texttt{Variation} to which the \textit{attribute variation} of \( d \) refers, and let \( t \) be the value of the \textit{attribute type} of \( d \). Furthermore, let \( c \) be the content of the \textit{attribute condition} of \( v \).

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

- If the \textit{attribute type} of \( d \) is \textit{required}, then \( \text{condition}(d) = v \Rightarrow c \wedge v_1 \vee \ldots \vee v_k \)
- If the \textit{attribute type} of \( d \) is \textit{conflicts}, then \( \text{condition}(d) = v \Rightarrow c \wedge \neg v_1 \wedge \ldots \wedge \neg v_k \)

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

\[ \text{condition}(d) = \text{condition}(d_1) \wedge \ldots \wedge \text{condition}(d_n) \]  

(7)

Let \( c_1, \ldots, c_n \) be the conditions of all \texttt{Variation} objects in a \texttt{Variability Exchange Language}. Then the following condition shall hold:

\[ c_1 \wedge \ldots \wedge c_n \]  

(8)
3.17.3 XML Serialization

3.17.3.1 XML Schema

Figure 39. XML Schema for VariationDependency

```xml
<xs:complexType name="VariationDependency">
  <xs:complexContent>
    <xs:extension base="Identifiable">
      <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"
          minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="type" type="VariationDependencyEnum"
        use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

In the XML Schema, the attribute variation is not implemented as an 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).

3.17.3.2 Examples

Example 16. XML Example for VariationDependency

```xml
<variability-exchange-model type="variationpoint-description" id="model">
  <structural-variationpoint id="vp1" type="optional">
    <variation id="vp1v1">
      <condition type="single-feature-condition">Feature1</condition>
    </variation>
  </structural-variationpoint>
  <structural-variationpoint id="vp2" type="optional">
    <variation id="vp2v1">
      <dependency type="conflicts" id="vp2d1">
        <variation ref="vp1v1"/>
      </dependency>
      <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>
  </structural-variationpoint>
</variability-exchange-model>
```
3.18 VariationDependencyEnum

3.18.1 Description

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

- requires
- conflicts
- x:* [TODO]

For more information see VariationDependency.

3.18.2 Specification

Figure 40. UML Diagram for class VariationDependencyEnum

3.18.3 XML Serialization

3.18.3.1 XML Schema

Figure 41. XML Schema for VariationDependencyEnum

```xml
<xs:simpleType name="EnumerationExtension">
    <xs:restriction base="xs:string">
        <xs:pattern value="x:\S+"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="VariationDependencyBaseEnum">
    <xs:restriction base="xs:string">
        <xs:enumeration value="requires"/>
        <xs:enumeration value="conflicts"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="VariationDependencyEnum">
    <xs:union memberTypes="VariationDependencyBaseEnum EnumerationExtension"/>
</xs:simpleType>
```

3.19 VariationPoint

3.19.1 Description

The abstract class VariationPoint describes a variation point in an artifact.

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

### 3.19.2 Specification

**Figure 42. UML Diagram for class VariationPoint**

```
+ variabilityExchangeModels
  + models 0..*

+ variabilityExchangeModel
  + type: VariabilityExchangeModelTypeEnum (readOnly)
  + uri: UniformResourceIdentifier [0..1] (readOnly)
  + variationPoint 0..*

+ variationPoint
  + type: VariationPointTypeEnum
  + bindingTime: BindingTime [0..*]
  + variableArtifact: ArtifactElement [0..*]
```

**Attribute type**

The attribute **type** restricts the number of **Variation** objects, which may be set during the binding process. Currently, three types are defined:

**Optional**

Any number of **Variation** objects may be set to this **VariationPoint**.

Or

At least one **Variation** object shall be set to this **VariationPoint**.

Xor

Exactly one **Variation** object shall be set to this **VariationPoint**.

**Attribute bindingTime**

The attribute **bindingTime** defines the binding time of a **VariationPoint**. For more information on the concept of binding times, see Section 3.2, “BindingTime”.

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 $s_1, \ldots, s_n$ be the values of the attribute selected of the BindingTime attributes of $v$. Then the following conditions must hold:

1. $\exists i. (i \in \{1, \ldots, n\} \land \text{eval}(s_i) = \text{true})$

2. $\forall j. (j \in \{1, \ldots, n\} \land j \neq i \Rightarrow \text{eval}(s_j) = \text{false})$

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.

Attribute variableArtifact

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

Not all VariationPoints have a variableArtifact.

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

3.19.3 XML Serialization

3.19.3.1 XML Schema

Figure 43. XML Schema for VariationPoint

```xml
<xs:complexType name="VariationPoint" abstract="true">
  <xs:extension base="Identifiable">
    <xs:sequence>
      <xs:element name="bindingtime" type="BindingTime"
                   minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="variable-artifact"
                   type="ArtifactElement"
                   minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="type" type="VariationPointTypeEnum"
                  use="required"/>
  </xs:extension>
</xs:complexType>
```

3.20 VariationPointHierarchy

3.20.1 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:
• $V = \{v_1, ..., v_n\}$ 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: $(v_i, v_j) \in E \Rightarrow \forall k \neq i \Rightarrow (v_i, v_j) \notin E$.

• $E$ must not contain cycles, that is, there cannot be a sequence. Formally, the following condition shall hold: $(v_{i_1}, v_{i_2}), (v_{i_3}, v_{i_4}), \ldots, (v_{i_{k-2}}, v_{i_{k-1}}), (v_{i_{k-1}}, v_{i_k}) \in E$ with $i_1 = i_k$.

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

The class VariationPointHierarchy inherits from Identifiable.

\(^6\)Strictly speaking, $V$ would be a set of nodes and there is a bijective mapping between $V$ and the set of elements of type VariationPoint in the [FIXME] DOM of the Variability Exchange Language document. We use a simplified language for the sake of clarity here.
3.20.2 Specification

Figure 44. UML Diagram for class VariationPointHierarchy

The attribute variationPoint of a VariationPointHierarchy identifies the endpoint of a variation-point hierarchy relation.
3.20.3 XML Serialization

3.20.3.1 XML Schema

Figure 45. XML Schema for VariationPointHierarchy

```xml
<xs:complexType name="VariationPointHierarchy">
  <xs:complexContent>
    <xs:extension base="Identifiable">
      <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>
```

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

3.20.3.2 Examples

Example 17. XML Example for VariationPointHierarchy

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

3.21 VariationPointTypeEnum

3.21.1 Description

The enumeration `VariationPointTypeEnum` defines which values are allowed for the attribute `type` of `VariationPoint`. Currently, this enumeration defines three values:

- optional
- or
• xor

For more information see VariationPoint.

3.21.2 Specification

Figure 46. UML Diagram for class VariationPointTypeEnum

<table>
<thead>
<tr>
<th>«enumeration»</th>
<th>VariationPointTypeEnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>optional</td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
</tr>
<tr>
<td>xor</td>
<td></td>
</tr>
</tbody>
</table>

3.21.3 XML Serialization

3.21.3.1 XML Schema

Figure 47. XML Schema for VariationDependencyEnum

```xml
<xs:simpleType name="VariationPointTypeEnum">
  <xs:restriction base="xs:string">
    <xs:enumeration value="optional"/>
    <xs:enumeration value="or"/>
    <xs:enumeration value="xor"/>
  </xs:restriction>
</xs:simpleType>
```
Appendix A An Annex

Appendices are distinguished from sections.
Appendix B A Normative Annex (Normative)

An annex or section with role="normative".
Appendix C A Non-normative Annex (Non-Normative)

An annex or section with role="non-normative".
Appendix D An Informative Annex (Informative)

An annex or section with role="informative".
Appendix E (normative) An ISO-normative Annex

An annex or section with role="iso-normative".
Appendix F (informative) An ISO-informative Annex

An annex or section with role="iso-informative".
Appendix G Acknowledgments (Non-Normative)

In a typical OASIS work product one might wish to list committee participants in a non-normative annex (markup shown above in the normative annex example) using wording along the line of “The following individuals have participated in the creation of this specification and are gratefully acknowledged:”

- Mary Baker, Associate Member
- Jane Doe, Example Corporation Member
- John Able, Other Example Corporation Member

Note that the itemized list uses `spacing="compact"` to remove the space between list items in the printed result, not the HTML result.
Appendix H Revision History

Typically the revision history is removed from a finalized specification or note.

<table>
<thead>
<tr>
<th>Revision X.Y</th>
<th>DD Mmmmmm YYYY</th>
<th>abc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Details of the revision</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Revision X.Y</td>
<td>DD Mmmmmm YYYY</td>
<td>abc</td>
</tr>
<tr>
<td>Details of the revision</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>