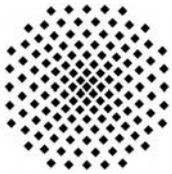

Universität Stuttgart

Fakultät Informatik, Elektrotechnik und Informationstechnik

Using Variability Descriptors to Describe Customizable SaaS Application Templates

Ralph Mietzner

Technischer Bericht (Report) 2008/01
January 18, 2008

Institut für Architektur von Anwendungssystemen

Universitätsstr. 38
70569 Stuttgart
Germany

CR: D.2.1, H.4.1

Summary

Customizable applications play an important role in software as a service (SaaS) scenarios. SaaS application providers want to exploit economies of scale by offering the same basic application to many customers. As customers have different requirements for the same type of application, SaaS vendors must offer so-called application templates that can be customized by their customers to be tailored exactly to their needs. Therefore variability points (i.e., points in an application template that can be customized) need to be made explicit and constraints for these variability points need to be specified. We introduce variability descriptors as a means to describe variability points for SaaS application templates independent of the artifacts (such as GUI components, workflows, configuration files, etc.) that make up the application.

1. Introduction

Software as a Service (SaaS) is a new delivery model for software. Software in a Software as a Service is hosted at a SaaS provider's data center and accessed by customers via the Internet. The advantage of this model for the customer over traditional "on premise" software (i.e., software that is run and maintained at the customer's own data center) is that the customer that wants to use the software does not need to maintain a data center but can outsource the whole application to a provider. The customer therefore does not need to make big initial investments in order to buy the necessary hardware and middleware infrastructure to run the application. From a customer's point of view the application can be used "on demand". The customer only pays for the application on a pay-per-use basis, i.e., only when he really needs the application.
The provider, however can exploit economies of scale by offering the same software to many customers and thus optimizing data center usage. Additionally, the required middleware and hardware infrastructure can be used to host different applications thus further increasing the utilization of the IT infrastructure. From a provider's point of view the resulting compute model is called "utility computing". Similar to a traditional utility, such as a water or electricity provider that provide water or electricity on demand a SaaS provider provides applications on demand. Users can subscribe and unsubscribe from these applications as they need them.
In the Application Service Provider (ASP) [20] model, one application is run for one customer on a dedicated server and middleware stack at an application service provider. The customer accesses the application via a network such as the Internet. Most of the time the applications, that are run in an ASP model are not even made to be accessed via the Internet in the first place, resulting in a lot of development effort to make them Web accessible. Since one application is run for one customer on a dedicated IT-Infrastructure ASPs face the same problems as traditional on-premise software. The underlying IT-Infrastructure must be provisioned for the peak load of that one customer resulting in a lot of resource overhead for the times where peak load is not reached. Additionally, the addition of new customers that want to use the same application results in the set up of a new instance of the application including the corresponding IT-infrastructure. Since the SaaS provider runs the same application for several customers on the same IT-infrastructure, the ASP model now scales. SaaS applications are applications that have been developed following a so-called multi-tenant architecture. Multi-tenant architecture means that several tenants (customers) can access the same application instance but the application behaves for each tenant as if it was a separate instance of that application. Thus two tenants A and B that are using the same SaaS application do not interfere with each other, especially tenant A cannot access the data from tenant B and vice versa. Using such a multi-tenant architecture results in applications where new tenants can be added and removed easily as they subscribe and unsubscribe to the application.
However, as different tenants have different requirements for an application the software must be configurable on a per-tenant basis. A SaaS software developer

therefore must annotate the software to be offered as a service with points that mark where the software can be customized by a tenant. We call these points, *variability points*, similar to those variability points (or variation points) that are introduced in software product line engineering. We call an application, that has been annotated with variability points, an *application template.* Tenants, can now start from that application template, customize it (i.e., bind the variability points) and then the application is run on the SaaS provider's infrastructure. A customized application template is called an *application solution*. Application solutions can be deployed in the data center of a SaaS provider. In this report we will focus on applications that are based on a service oriented architecture using Web service and workflow technology. In particular we consider applications that consist of a set of Web services that have been developed in any programming language. These Web services are orchestrated into business processes using WS-BPEL [11], the standard for the modeling and execution of business processes in the Web service world. In order to include people in the processes WS-Human Task [3] and BPEl4People [2] are used. Additionally the applications we consider have a user interface that is specified in the form of portlets and (X)HTML pages. Figure 1 illustrates the notion of templates and solutions, as well as the artifacts that we consider to be customizable in such an application in this report, namely the GUI artifacts, the workflows and configuration data for the services..

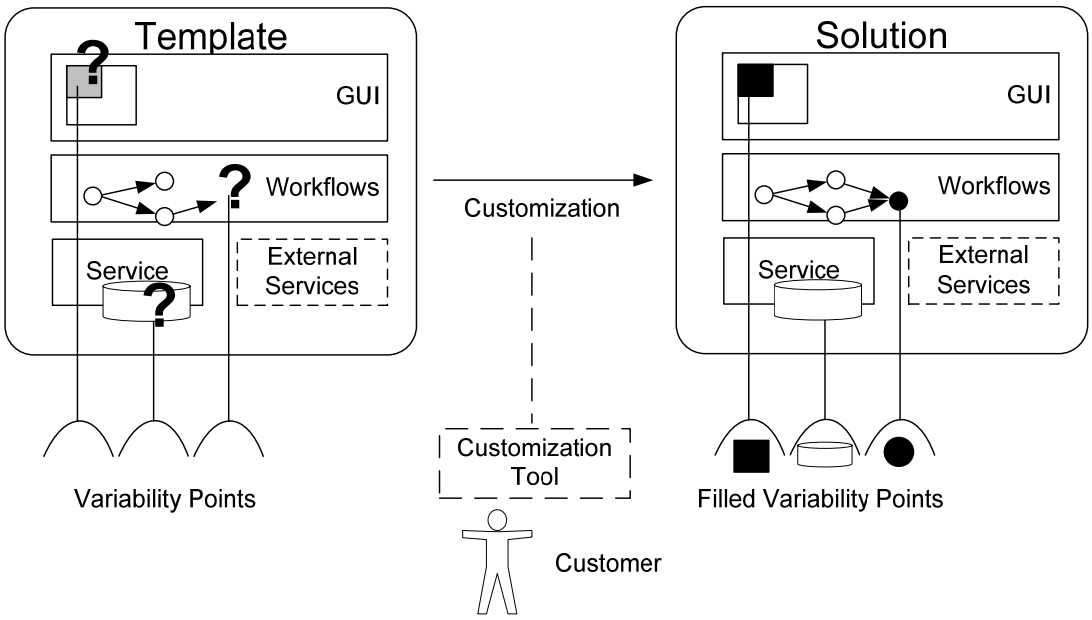

Figure 1: Transforming an application Template into a solution by customizing it

In this report we introduce a model and XML-based language to describe variability points in application templates, so-called *variability descriptors*. We describe the requirements for such a variability descriptor as well as the relation to software product line engineering.

2. Related Work and Motivation

[5] introduce four SaaS maturity levels, relating to the fact on how well the application scales to many customers. In level 0 an application is specifically run for one customer at a SaaS provider. This level corresponds to the traditional ASP level. Starting from level 1 the SaaS application is customizable via configuration on a per-tenant basis but still in level 1 one instance of the software serves only one customer. In level 2 only a single instance of the SaaS application serves multiple tenants. Separate configuration metadata is used to configure the application to the need of the particular tenants. In order to achieve level 3, the SaaS application is developed as a single instance multi-tenant application and several instances are run in a load-balanced farm. In level 3 load is balanced via a tenant-load-balancer that spreads the load over the various servers the application runs on.

In the field of software engineering, software reuse is important topic. Software product families as one possibility for software reuse [12], [13] have been under extensive research in the software engineering community.
Variation points have been introduced in [6] and are one of the key concepts for software product lines to express variability. Variation points, often also called variability points or points of variability, allow the specification of points in a software design that can be customized in order to create new product line members. This is a similar problem to the problem we face in customizable SaaS applications, where different tenants can create different "solution versions" of an application template by filling the variability points offered by this template.

WS-BPEL employs a similar concept to variability points in its abstract processes. Abstract processes allow the specification of so-called "opaque tokens". The notion of opaque tokens in abstract BPEL is very similar to the notion of variability points in product family engineering. These opaque tokens are variability points that are left open by explicitly marking them as opaque or implicitly by omitting them. Opaque tokens need to be defined ("bound") before the process becomes executable and can be deployed in a BPEL engine. In BPEL the "binding" of opaque tokens and the refinement of abstract processes into executable processes is called executable completion.

The BPEL specification defines two profiles for abstract processes. The first profile, the profile for observable behavior is intended to describe the communication of a service (that can for example be a BPEL process) with the outside. The second profile, the template profile describes a template process with certain points of variability. Both profiles differ in the locations, where opaque tokens are permitted. Furthermore different kinds of modifications are permitted during executable completion of the different abstract profiles. For example, during executable completion of an abstract process of the observable behavior profile new activities can be added everywhere where BPEL permits it, whereas during the executable completion of a template profile abstract process new activities can only be added as replacement for previously defined opaque

activities. Opaque tokens are a way in BPEL to specify variability points, however they lack fundamental capabilities needed in a SaaS scenario. It is for example not possible to specify dependencies between opaque tokens or to specify alternative values for an opaque token. Dependencies between variability points are an important mechanism to specify complex configuration possibilities in a product line [7].
In related work the Pesoa project coined the term "process family engineering" for families of business processes and introduced a notation how to model these families in BPMN [15]. Flexible Workflows and workflow variants have been extensively researched within the context of run-time process variants [18], configurable workflow models in [1] and configurable EPCs (C-EPCs) in [14]. An XML exchange format to express these C-EPCs is introduced in [9]. In [8] a formal model to define customizations for BPEL processes is introduced. Our approach differs from these approaches as it is not focused on process models but on the documents that make up an application. Our approach therefore natively supports not only processes but other parts of an application such as configuration files, interface descriptions and deployment descriptors. Compared to [15] and C-EPCS where alternatives are modeled in BPMN or EPCs, we introduce variability points on the implementation level. Variabilities on the implementation level (in our case BPEL) are necessary to generate applications that can be directly deployed in corresponding middleware (i.e., BPEL engines). Furthermore we introduce a concept how the introduced variability descriptors can be mapped to a customization process model. An instance of this customization process model can then be used by a customer to generate a new solution out of the template.
The Reusable Asset Specification (RAS) [10] defines a model and format to describe reusable assets including variability points similar to the one we present here, however the notation is fairly generic and therefore is not directly suitable for the specification of SaaS application templates. Furthermore, it is more geared towards design-time rather than customization for SaaS scenarios. However, the variability point description element in a RAS manifest file contains an optional `reference` attribute. "The optional `reference` attribute points to an external document that could further explain the variability point." [10]. The variability descriptors described in this report can be integrated into RAS using this reference mechanism.

Schematron [19] is a language that can be used to specify constraints on XML documents that go further than those that can be introduced with XML Schema. Schematron could be used to describe points of variability for XML documents. However, Schematron does not allow to specify dependencies between the single constraints and the approach is limited to XML files while our approach offers dependencies on variabilities and introduces extension points to deal with non-XML files as well.

3. Introductory Example

TaaS (Travel as a Service) is a travel agency that offers a travel booking service in a SaaS business model to small and medium sized enterprises. The travel

booking service contains a WS-BPEL booking process that is essentially comprised of the following simplified steps.

1. The user submits a travel request via a web front-end
2. A request-for-approval is sent via e-mail to the manager of the user
3. Upon successful approval the trip is booked via a third-party travel agency
4. The itinerary is sent to the user

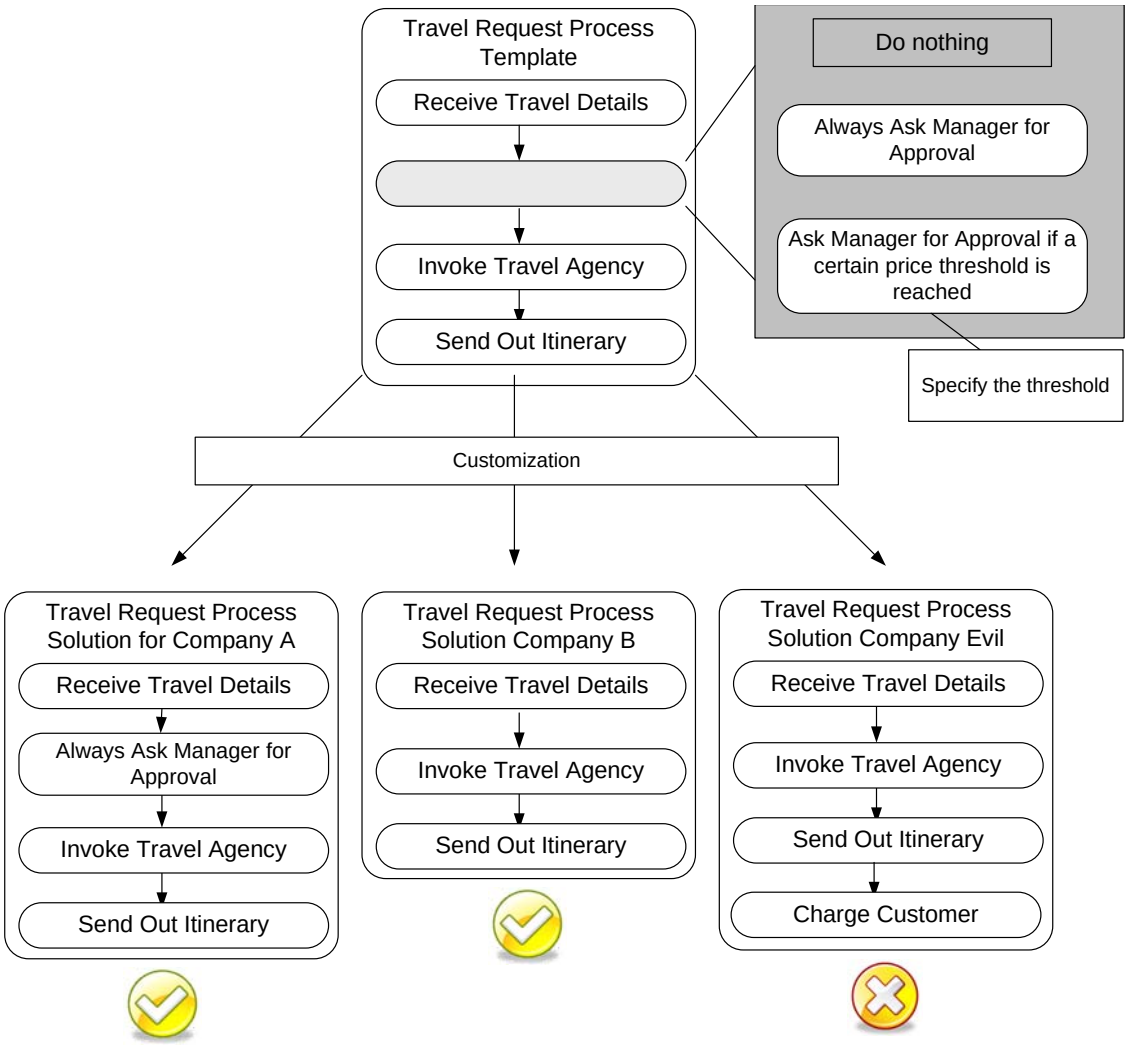

Figure 2: Taas Travel Request Process Template and possible derived Solutions

The requirements for such a process might be quite different for different companies. One company for example needs approval by a manager only if the travel costs exceed a certain sum, while another company permits its employees to book any trip while others always need manager approval. However, all these different processes can be derived from the same basic abstract application template by defining concrete values for variability points offered in the template.

Figure 2 shows the Travel Request Process template including a variability point at the second activity that allows to either specify that nothing is done here or that the manager always needs to confirm the travel request or that a certain price threshold is needed for manager confirmation. In case the manager approval is needed after a certain threshold the threshold can be specified. As the

example shows one decision at a variability point might trigger other decisions. Figure 2 also shows different solutions for the template. Whereas the first two solutions (company A and company B) are valid customizations, the third solution (company Evil) is an invalid customization since the process document has been modified at a location where no variability point is present.
Customers that want to use the travel booking service for their enterprise can register over the web for the service. They are then guided through the customization process by a wizard, after customization the travel booking service the created solution specific for that company is automatically deployed in the TaaS data center. We show in section 6 how this scenario can be specified with the variability descriptors we introduce in this report.

4. Variability Descriptors

Derived from [4],[12],[7],[15] and the example above we capture the following requirements for variability points:

1. Explicitly mark a variability point in order to tell the customizer where customization is needed and allowed.
2. Describe constraints for a variability point. Constraints specify allowed values for the variability point.
3. Describe dependencies between two variability points in order to allow the specification of an order in which variability points need to be bound. Furthermore conditions must be described that allow to activate and deactivate certain possible values for a variability point dependent on the values filled in during the binding of another variability point.

A straightforward means to define variability descriptors would be to address these requirements as extensions directly in the files making up the application. For example, one could add variability points via opaque tokens and extensions in a BPEL file. However, our approach is to provide a separate variability-descriptor XML document that describes the variability points and their properties and points into the file to be customized. The advantage of the second approach is that for different customers, different variability descriptors can be introduced. For example certain alternatives might be only allowed for premium customers.
Similar to WS-Policy [16] where policies can be provided as a separated WS-Policy Attachment document we attach a variability descriptor to a SaaS application. This approach also allows variability attachments to be attached to documents and programming languages that have not been developed with extensibility in mind and that do not allow the specification of variability descriptors in their own language. Additionally using an external variability descriptor allows describing variability point dependencies for variability points in different documents of different types. For example one can describe a variability point in a BPEL file where the customization has impact on a variability point in a deployment descriptor file for that BPEL process.

4.1 Basic Elements of a Variability Descriptor

The variability descriptor consists of a set of variability points and dependencies. A variability point describes a single variability point including locators that

point into documents and possible values and restrictions for the binding of this variability point. In the dependency section of the variability descriptor the dependencies between the variability points declared in the variability descriptor are specified. Figure 3 shows the basic elements of a variability descriptor. The XML-Schema for the variability descriptor document can be found in Appendix A. An XML document, describing a complete variability descriptor is shown in Section 6.

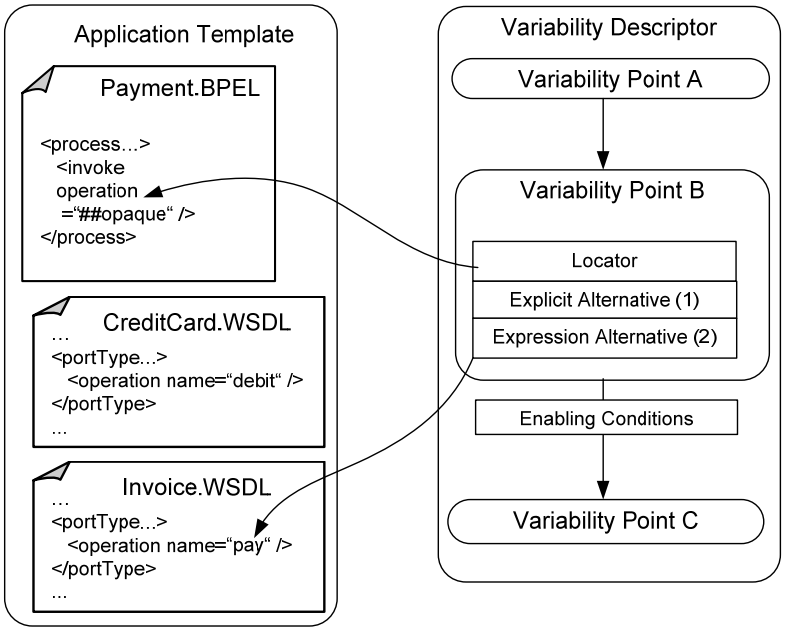

Figure 3: Sample Application artifacts and basic elements of a variability descriptor

The top-level element variability descriptor is serialized to XML as shown in Listing 1. The `variabilityDescriptor` element is the top-level element of the variability descriptor document.

Attributes:

The optional `expressionLanguage` attribute defines the language in which expressions (such as those used for locators or expression alternatives) are specified. We use the mechanism used in BPEL [11] to identify these languages via a URI.
The default URI is `urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0` denoting that XPath is used as the expression language. Individual elements can overwrite the `expressionLanguage` specified in the top-level `variabilityDescriptor` element. The mandatory `targetNamespace` attribute defines the namespace for which this variability document is declared.

Elements:

The variability descriptor element contains two sub-elements `variabilityPoints` and `dependencies` that describe a list of variability points and dependencies respectively. The `variabilityPoints` element must contain at least one

`variabilityPoint` element whereas the `dependencies` element can also contain zero `dependency` elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<variabilityDescriptor
  xmlns="http://www.iaas.uni-stuttgart.de/schemas/VD"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.iaas.uni-stuttgart.de/schemas/VD VD.xsd"
  expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"
  targetNamespace="http://www.iaas.uni-stuttgart.de/schemas/taasVD”>
  <variabilityPoints>
    <variabilityPoint name="VP1">
     …
    </variabilityPoint>
    <variabilityPoint name="VP2">
      …
    </variabilityPoint>
  </variabilityPoints>
  <dependencies>
    <dependency>
      …
    </dependency>
  </dependencies>
</variabilityDescriptor>
```

Listing 1: XML representation of the variability descriptor element

4.2 Variability Points

A variability point element describes a variability point. It consists of a set of locators that point into locations in a document where the variability point is meant to be. The second part of the variability point declares the possible values that are allowed to fill the variability point. We call these the *alternatives*. Listing 2 shows the XML representation of a variability point element.

Attributes:

The variability point element has one mandatory attribute, the `name` attribute. The `name` attribute uniquely identifies a variability point within the namespace of the variability descriptor it is declared in. Therefore two variability points declared in the same variability descriptor cannot have the same name. Corresponding tooling must enforce this.

Elements:

A variability point element has three sub-elements:
An optional `import` element containing an expression. The expression points to a variability point declaration in another variability descriptor document. The `import` element denotes a textual inclusion of the variability point the expression points to. The import element has one attribute “`expressionLanguage`” that can be used to overwrite the expression language set by the top-level variability

descriptor document. For example an XPath expression (including the `document()` function introduced in XSLT [17]) can be used to point to a variability point declaration in another variability descriptor document. In case a variability point is imported via the import element, the locators and variability elements of the variability point extend the imported variability point.

A mandatory `locators` element representing a list of locators. The `locators` element can have zero or more `locator` sub-elements. In case the `import` element is omitted for a variability point it must have at least one `locator` specified.

A mandatory `alternatives` element representing a list of alternative values for that variability point. The `alternatives` element can contain zero or more `alternative` elements representing values that can be used to fill the variability point. In case the `import` element is omitted the variability point must have at least one alternative specified.

```
<variabilityPoint name="VP1">
   <locators>
     <locator>
      document("myProcess.bpel")//bpel:opaqueActivity[@name="payment"]
     </locator>
   </locators>
   <alternatives>
     <alternative name="Alternative1">
      …
     </alternative>
     <alternative name="Alternative2">
      …
     </alternative>
     <alternative name="Alternative3">
      …
     </alternative>
   </alternatives>
 </variabilityPoint>
```

Listing 2: XML representation of the variability point element

4.3 Locators

Locators describe how a variability descriptor is attached to a variability point. A `locator` element contains an expression that specifies a pointer into a specific location or several locations in a document where the variability is located. Listing 2 shows the XML representation of a locator element that contains an XPath expression that points to an opaque activity inside a BPEL document.

Attributes:

The `locator` element has one optional attribute: `expressionLanguage`. The `expressionLanguage` attribute denotes the expression language in which the expression contained in the locator is specified. In case the attribute is omitted the `expressionLanguage` from the top-level `variabilityDescriptor` element is assumed. The standard expression language is XPath. Again the `document()`

function introduced in XSLT [17] can be used to point into the respective documents.

Elements:

The `locator` element is an extensible element; therefore arbitrary sub-elements from other namespaces can be nested in the locator element. This feature can be used for example to describe locators that can point into text-files. Listing 3 shows an example of a text locator pointing into line 3 of the database.properties file.

```
<locator
   expressionLanguage="urn:iaas:uni_stuttgart:variability:textLocator">
   <tl:textLocator
      xmlns:tl="http://www.iaas.uni-stuttgart.de/schemas/VDTextLocator">
      <tl:document>database.properties</tl:document>
      <tl:line>3</tl:line>
   </tl:textLocator>
</locator>
```

Listing 3: Example of a text locator

A locator based on XPath can reference multiple nodes in an XML document; for example, all non-specified partner link attributes of invoke activities. Furthermore, it is allowed to specify multiple locators for one variability point. Referencing multiple locations in one or more documents allows the filling of variation points that have the same set of alternatives in one step.
In case of an XPath based locator it must be possible to unambiguously reference the node or the set of nodes the locator points to. For example in case the locator points into a BPEL document, activities cannot be unambiguously referenced since they do not contain a unique id attribute. Therefore sometimes it might be necessary to add unique ids to extensible documents in order to allow them to be referenced by an XPath locator.

4.4 Alternatives

Alternatives are a means to specify one or more possible values for a variability point. This can either be done explicitly by specifying the possible values for the variability point, or implicitly based on an expression that retrieves the possible values from another part of the same or a different document. In order to allow a part of a document referenced by a variability point to be specified as optional, i.e. to allow that the variability point can be filled with an empty string, the empty alternative is introduced. Free alternatives denote that a variability point can be filled by any user input. The last type of alternatives, the locator alternative, allows specifying that the values of the locations inside a document, referenced by the locators of the variability point are alternative values too. Each variability point can have a set of alternatives that is made up of arbitrary combinations of alternatives from the different alternative types. Listing 4 shows the XML representation of an alternative.

Attributes:

Each alternative has a `name` attribute that identifies it uniquely within its variability point. Additionally an alternative can be the default alternative indicated through the value of the optional `default` attribute. If the `default` attribute is omitted, the alternative is not a default alternative. Each variability point can have at most one default alternative. Listing 4 shows the standard attributes for alternatives

Elements:

An alternative can have one of the following sub-elements:

- `emptyAlternative`, denoting that the alternative is an empty alternative,
- `explicitAlternative`, denoting that the alternative is an explicitly declared alternative,
- `expressionAlternative`, denoting that the alternative depends on an expression,
- `freeAlternative`, denoting that the alternative depends on user input and
- `locatorAlternative`, denoting that the values referenced by a locator are also possible values for the variability point

```
<alternative name="Alternative1" default="true">
  …
</alternative>
```

Listing 4: XML representation of an alternative

4.4.1 Empty alternatives

Empty alternatives denote that the variability point can be filled with an empty string. These alternatives can be used to mark optional parts in a document. A variability point can have at most one empty alternative. The empty alternative can also be the default alternative. Listing 5 shows the XML representation of an empty alternative.

```
<alternative name="Alternative1">
  <emptyAlternative/>
</alternative>
```

Listing 5: XML representation of an alternative

4.4.2 Explicitly declared alternatives

Explicitly declared alternatives are a means to specify alternatives whose values are specified at design time. Listing 6 shows the XML representation of an explicit alternative. The explicit alternative contains a piece of BPEL code that can be used to fill the variability point the alternative belongs to. Since the explicitly declared alternative contains an `xsd:any` sub-element of multiplicity `unbounded` and is of type `mixed`, arbitrary XML nodes and text can be placed inside the `explicitAlternative` element.

```
<alternative name="Alternative2">
  <explicitAlternative>
     <bpel:invoke
        xmlns:bpws="http://docs.oasis-open.org/
                    wsbpel/2.0/process/executable"
        xmlns:lns="http://www.iaas.uni-stuttgart.de/someWSDL"
        partnerLink="assessor" portType="lns:riskAssessmentPT"
        operation="check" inputVariable="request"
        outputVariable="risk">
     </bpws:invoke>
  </explicitAlternative>
</alternative>
```

Listing 6: XML representation of an explicit alternative

Expression based alternatives

Expression based alternatives specify alternatives whose values are determined by an expression. Expression based alternatives permit to specify an expression that references a set of values in the variability descriptor document or other documents that can be used to fill the variability point. Listing 7 shows how to define an alternative pointing a BPEL document that allows the names of all BPEL partner links of a specific scope to be possible values for a variability point. As with the locators we allow different expression languages to describe expression based alternatives. The language that is used to describe the expression based alternative is declared in the optional `expressionLanguage` attribute. In case the attribute is omitted the expression language of the top-level variability descriptor element is used which is XPath per default. The most convenient method for XML-based documents is the XPath based expression alternative that is shown in Listing 7. Other expression based alternatives can be plugged-in. For example, a regular expression alternative can be used to define expressions on string-based documents.
Expression based alternatives might return more than one possible value to fill a variability point.

```
<alternative name="Alternative3">
  <expressionAlternative>
document("myProcess.bpel")//bpws:scope[@name="payment"]/bpws:partnerLinks
  </expressionAlternative>
</alternative>
```

Listing 7: XML representation of an expression based alternative

Free Alternatives

Free alternatives are a way to denote that a customizer can fill the variability point with any free text. However, a corresponding tool must check that the free text entered by a customizer corresponds to the data type of the variability point. I.e. if a variability point points to an attribute that is specified as an integer, the tool must allow a customizer to enter only an integer value as a free text. Optionally constraints on the allowed values can be added via expressions. Again arbitrary expression languages can be added via the optional `expressionLanguage` attribute. We show an XPath expression in Listing 8. The alternative presented

in Listing 5 denotes that a value must be entered that is between 1 and 10000. In order to allow arbitrary XPath expressions we introduce the variable `$value` that represents the value entered by the user. Tooling must then replace the `$value` variable with the value entered, during constraint checking.

```
<alternative name="Alternative4">
  <freeAlternative>$value &gt;=1 and $value &lt;= 10000</freeAlternative>
</alternative>
```

Listing 8: XML representation of a free alternative

Locator Alternatives

Locator alternatives are a means to denote that the values referenced by the locator(s) of the variability point the alternative is declared in, are valid values for the variability point. For example if the locator of a variability point points to an operation attribute inside an invoke activity inside a BPEL document that already has an operation name assigned to it, a locator alternative for that variability point denotes, that this operation name is also an alternative value for that operation attribute. In case a locator points to several artifacts inside a document or different documents or several locators are specified for a variability point, the semantics of the locator alternative is as follows:
In case a customizer selects the locator alternative, the values referenced by the locators are not touched in the original documents during the customization. This mechanism allows for example to define the values in a document affected by a variability point as default values by using the locator alternative and by setting this alternative's default attribute to true. Listing 9 shows the XML representation of a locator alternative.

```
<alternative name="Alternative5">
  <locatorAlternative/>
</alternative>
```

Listing 9: XML representation of a locator alternative

4.4 Dependencies

In order to describe dependencies between different variability points the dependency element is introduced. A dependency element can have multiple sources and one target. The sources are the variability points that the target variability point is dependent on. During customization, these variability points must be bound or disabled before the target variability point can be bound. It is not allowed to have a circle in the dependencies, i.e., a variability point cannot be the (transitive) target of a dependency if it is contained in the set of (transitive) sources of the dependency. Dependencies contain so-called enabling conditions that further specify the dependency. Listing 10 shows the XML representation of a dependency.

```
<dependency>
  <sources>
    <variabilityPointRef>tvd:VP1</variabilityPointRef>
  </sources>
  <target>
    <variabilityPointRef>tvd:VP2</variabilityPointRef>
  </target>
  <enablingConditions>
     <enablingCondition>
       …
     </enablingCondition>
     <enablingCondition>
       …
     </enablingCondition>
  </enablingConditions>
</dependency>
```

Listing 10: XML representation of a dependency

Elements

A dependency element has three sub-elements. The `sources` element represents a list of source variability points; the `target` element contains the target variability point. Both the `sources` and the `target` element contain `variabilityPointRef` sub-elements. The target element however, must contain exactly one `variabilityPointRef` element whereas the sources element can contain zero or more `variabilityPointRef` elements. A `variabilityPointRef` element contains the name of a variability point that identifies one of the variability points declared in the variability descriptor.
The `enablingConditions` element contains zero or more enabling conditions specified via `enablingCondition` elements that define conditions under which certain alternatives of the target variability point are enabled.

4.5 Enabling Conditions

In some cases a simple dependency is not enough, an “enabling condition” that enables alternatives of a variability point based on a condition is needed. For example, an alternative might only be enabled if another alternative has been taken in a variability point that has been bound before.

Elements:

An `enablingCondition` element has two sub-elements, the `condition` element specifying the condition that must be true in order to enable the alternatives specified in the `enabledAlternatives` list that is represented via the `enabledAlternatives` element.
The mandatory condition element contains the condition. As with all other expression elements in the variability descriptor, the condition can be in any expression language. The expression language is specified via the optional `expressionLanguage` attribute. In case the attribute is omitted the expression

language of the top-level `variabilityDescriptor` element applies which is XPath by default.
The mandatory `enabledAlternatives` element contains zero or more `alternativeRef` elements that reference an alternative of the target variability point of the dependency the `enablingCondition` is located in. The `alternativeRef` element can only contain the `QName` of an alternative, if that alternative is declared in the target variability point of the dependency the enabling condition is declared in. In case no `enabledAlternative` element is contained in the `enabledAlternatives` list the semantics during customization are as follows: The variability point is considered disabled and must not be bound by the customizer.

Listing 11 shows the XML representation of two enabling conditions. In case alternative `"Alternative1"` was selected for variability point `"VP1"` alternatives `"alt1"` and `"alt2"` are selected for the target variability point, whereas otherwise (denoted by the empty condition) only alternative `"alt2"` is enabled and can be selected by the customizer.

```
      <enablingConditions>
        <enablingCondition>
          <condition>
            selectedAlternative("tvd:VP1","tvd:Alternative1")
          </condition>
          <enabledAlternatives>
            <alternativeRef>tvd:alt1</alternativeRef>
            <alternativeRef>tvd:alt2</alternativeRef>
          </enabledAlternatives>
        </enablingCondition>
        <enablingCondition>
          <condition></condition>
          <enabledAlternatives>
            <alternativeRef>tvd:alt2</alternativeRef>
          </enabledAlternatives>
        </enablingCondition>
```

Listing 12: XML representation of enabling conditions

The semantics of the enabling condition is the following: In case the condition is true, the set of activities declared in the enabled alternatives can be used to fill a variability point by a customizer.
We define an ordering over the enabling conditions in a dependency, which affects the evaluation semantics. The first condition is the one evaluated first, if the condition is true the set of variants that are marked as enabled in that condition can be selected for the variability point. If the condition evaluates to false the next condition is evaluated. An empty condition denotes that the corresponding alternatives are enabled regardless of any condition; it corresponds to the else branch of traditional programming language's if-then-else constructs. In case all enabling conditions evaluate to false (and no empty condition is present) the variability point is considered disabled and does not need to be bound.
In order to facilitate the specification of XPath based enabling conditions we introduce some shortcut XPath functions:

- `variabilityPoint(QName)` – This function points to the variability point that is specified with the respective QName. A compliant implementation must

replace the `variabilityPoint` function with the locators of the variability point that is specified with the QName.

- `selectedAlternative(variabilityPointQName,alternativeQName)` – In order to allow constraints to be specified on alternatives and other variability points we introduce a new function that denotes that the variant with the name of a variability point with the name `variabilityPointQName` has been taken. The expression `selectedAlternative("vp1:vp1","vp1:alt3")` denotes that the alternative `"alt3"` at variability point `"vp1"` of namespace `"vp1"` has been selected. A compliant variability descriptor generation tool must enforce that this function can only be added to variability points that are dependent on the variability point with the given id. Otherwise deadlocks could arise where a customization tool waits for the evaluation of an enabling condition that references a variability point that will only be bound after the enabling condition has been evaluated.

5 Customization Validation

In order to being able to actually run the customized application a tool must ensure that the customization (i.e. the binding of the variability points) performed by the customizer is valid. Valid means that all variability points have been filled with values that are allowed. Therefore after customization, all constraints imposed on a variability point by an expression alternative must evaluate to true. Additionally variability points for which alternatives are specified, must have been filled with one of the alternatives, otherwise the executable completion is not valid. In order to correctly validate the constraints, a tool must consider the dependencies and validate the variability points in the order of the dependencies. Which means that the tool must start to validate all independent variability points first (i.e., those that are not dependent on any other variability points) and then follow the dependencies. The variability points that are not activated (because of enabling conditions that all evaluate to false) must not be validated. We assume in this report that all solutions that can be produced by doing a valid executable completion lead to an application that itself can be run on the corresponding middleware. For example all valid customizations of the TaaS process will produce a BPEL process that can be executed by a WS-BPEL engine. Generation of templates that ensure this is out of the scope for this report and is subject to future work.

6. Example Variability Descriptor

The following variability descriptor describes the variability descriptor for the example in Section 3. It includes two variability points that operate on a BPEL file (`"taasProcess.bpel"`). The first variability point points to an opaque activity (`"managerApproval"`) and allows a customizer:

- To delete it, using alternative `"noManagerApproval"`, an empty alternative.

- To replace it with an assign activity and a BPEL4People people activity. The assign activity sets the threshold for the price for which the manager approval is needed. The two activites are connected with a link that contains a transition condition that disables the people activity containing the task for the manager to approve if the price is less than the threshold. This is alternative "managerApprovalWThreshold", an explicit alternative.
- To replace it with a BPEL4People people activity that contains a task asking the manager for approval, using the explicit alternative "managerApprovalWithoutThreshold"

Having bound variability point "VP1" the customizer can now bind variability point "VP2" which is connected to "VP1" via a dependency. In case the customizer chose "managerApprovalWThreshold" in the variability point "VP1" he must now decide between two alternatives:

- Alternative "defaultThreshold" which is a locator alternative, denoting that the default threshold of 1000 applies (as defined in the explicit alternative from "VP1") or
- Alternative "customThreshold" which is a free alternative that allows to enter any integer value greater than 0.

Otherwise (in case the customizer has chosen one of the two other alternatives in variability point "VP1") he cannot specify the threshold. No alternatives are enabled in this case which denotes that the variability point is disabled and does not need to be bound.

```
<?xml version="1.0" encoding="UTF-8"?>
<variabilityDescriptor
  xmlns="http://www.iaas.uni-stuttgart.de/schemas/VD"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:tvd="http://www.iaas.uni-stuttgart.de/schemas/taasVD"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:b4p="http://www.example.org/BPEL4People"
  xmlns:htd="http://www.example.org/WS-HT"
  xsi:schemaLocation="http://www.iaas.uni-stuttgart.de/schemas/VD VD.xsd "
  expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"
  targetNamespace="http://www.iaas.uni-stuttgart.de/schemas/taasVD">
  <variabilityPoints>
    <variabilityPoint name="VP1">
      <locators>
        <locator> document("taasProcess.bpel")//
                  bpel:opaqueActivity[@name="managerApproval"]
        </locator>
      </locators>
      <alternatives>
        <alternative name="noManagerApproval">
          <emptyAlternative />
        </alternative>
        <alternative name="managerApprovalWThreshold">
          <explicitAlternative>
            <bpel:flow>
              <bpel:links>
                <bpel:link name="L1" />
              </bpel:links>
              <bpel:assign name="assignApprovalThreshold">
                <bpel:sources>
                  <bpel:source linkName="L1">
                    <bpel:transitionCondition>
```

```
                    $NewMessageRequest/price &lt;= $ApprovalThreshold
                  </bpel:transitionCondition>
                </bpel:source>
              </bpel:sources>
              <bpel:copy>
                <bpel:from>
                  <bpel:literal>1000</bpel:literal>
                </bpel:from>
                <bpel:to variable="ApprovalThreshold" />
              </bpel:copy>
            </bpel:assign>
            <bpel:extensionActivity>
              <b4p:peopleActivity name="askManagerForApproval"
                inputVariable="request" outputVariable="decision"
                isSkipable="yes">
                <bpel:targets>
                  <bpel:target linkName="L1" />
                </bpel:targets>
                <htd:task name="managerApprovalTask">...</htd:task>
                ...
              </b4p:peopleActivity>
            </bpel:extensionActivity>
          </bpel:flow>
        </explicitAlternative>
      </alternative>
      <alternative name="managerApprovalWithoutThreshold">
        <explicitAlternative>
          <bpel:extensionActivity>
            <b4p:peopleActivity name="askManagerForApproval"
              inputVariable="request" outputVariable="decision"
              isSkipable="yes">
              <htd:task name="managerApprovalTask">...</htd:task>
              ...
            </b4p:peopleActivity>
          </bpel:extensionActivity>
        </explicitAlternative>
      </alternative>
    </alternatives>
  </variabilityPoint>
  <variabilityPoint name="VP2">
    <locators>
      <locator>document("taasProcess.bpel")//bpel:
               assign[@name="assignApprovalThreshold"]
               /bpel:copy/bpel:from/bpel:literal
      </locator>
    </locators>
    <alternatives>
      <alternative name="defaultThreshold" default="true">
        <locatorAlternative />
      </alternative>
      <alternative name="customThreshold">
        <freeAlternative>$value &gt;=0</freeAlternative>
      </alternative>
    </alternatives>
  </variabilityPoint>
</variabilityPoints>
<dependencies>
  <dependency>
    <sources>
      <variabilityPointRef>tvd:VP1</variabilityPointRef>
    </sources>
    <target>
```

```
        <variabilityPointRef>tvd:VP2</variabilityPointRef>
      </target>
      <enablingConditions>
        <enablingCondition>
          <condition>
             selectedAlternative("tvd:VP1","tvd:managerApprovalWThreshold")
          </condition>
          <enabledAlternatives>
            <alternativeRef>tvd:defaultThreshold</alternativeRef>
            <alternativeRef>tvd:customThreshold</alternativeRef>
          </enabledAlternatives>
        </enablingCondition>
        <enablingCondition>
          <condition></condition>
          <enabledAlternatives />
        </enablingCondition>
      </enablingConditions>
    </dependency>
  </dependencies>
</variabilityDescriptor>
```

7 Conclusions and Future Work

In this report we introduce a model and an XML format to describe variability descriptors that can be used to annotate SaaS applications with variability points. We describe the concept of variability points that can be connected via dependencies to describe complex customization options for SaaS applications. The concept of locators is introduced to attach a variability point that is specified in a variability descriptor document to a document that needs to be customized. Such documents can be XML files, in case the XPath based locator specified in this document is used or can be any other format in case other locators are specified that extend the basic framework we introduce here. In order to describe the values permitted to fill a variability point we introduce alternatives. Alternatives describe allowed values for the filling of a variability point. In order to describe constraints and complex dependencies between variability points we introduce the concept of dependencies and enabling conditions. All these elements of a variability descriptor can be extended by users to incorporate their requirements. Therefore we introduce an extensible framework that offers plugging points where extensions to the basic framework described in this report can be made.

6 References

[1] Van der Aalst W. M. P, Dreiling A., Gottschalk F., Rosemann M., Jansen-Vullers M. H.: Configurable Process Models as a Basis for Reference Modeling. Business Process Management Workshops 2005: 512-518

[2] Active Endpoints, Adobe, BEA, IBM, Oracle, SAP AG, *WS-BPEL Extension for People specification v1.0*,

http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/ (2007)

[3] Active Endpoints, Adobe, BEA, IBM, Oracle, SAP AG, *WS-BPEL Extension for People specification v1.0*, http://www.ibm.com/developerworks/webservices/library /specification/ws-bpel4people/, 2007

[4] Bosch, J. (2000), *Design and use of software architectures: adopting and evolving a product-line approach*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

[5] F. Chong, G. Carraro. "Building Distributed Applications Architecture Strategies for Catching the Long Tail", MSDN architecture center, 2006. http://msdn2.microsoft.com/en-us/library/aa479069.aspx

[6] Jacobson, I., Griss, M., Jonsson, P.: *Software Reuse: Architecture, Process and Organization for Business Success.* Addison-Wesley, 1997

[7] Jaring, M. & Bosch, J., "Variability Dependencies in Product Family Engineering", *in Software Product-Family Engineering,* pp. 81-97. Springer Berlin / Heidelberg (2003)

[8] Lazovik A., Ludwig H., "Managing Process Customizability and Customization: Model, Language and Process," *Web Information Systems Engineering – WISE 2007*, 2007

[9] Mendling, J., Recker, J., Rosemann, M., and van der Aalst, W. (2005). "Towards the Interchange of Configurable EPCs: An XML-based Approach for Reference Model Configuration," *Proceedings of the Workshop on Enterprise Modelling and Information Systems Architectures.* University of Klagenfurt, Austria 2005,

[10] Object Management Group (OMG), *Reusable Asset Specification*, Version 2.2,2005

[11] Organization for the Advancement of Structured Information Standards (OASIS), *Web Services Business Process Execution Language Version 2.0,* 2007

[12] Parnas, D. L., *On the Design and Development of Program Families, IEEE Transactions on Software Engineering SE-2(1), 1-9.* 1976

[13] Pohl, K.; Böckle, G., van der Linden, F. J. *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer, 2005.

[14] Rosemann, M. and van der Aalst, W. M. P. "A configurable reference modelling language." *Inf. Syst. 32*, 1Mar. 2007, 1-23.

[15] Schnieders, A. and Puhlmann, F., "Variability Mechanisms in E-Business Process Families," *in BIS*, 2006 pp. 583-601.

[16] W3C Member Submission, *Web Services Policy 1.2 - Framework*, http://www.w3.org/Submission/WS-Policy/ (2006)

[17] W3C Recommendation, *XSL Transformations Version 1.0*, http://www.w3.org/TR/xslt

[18] Sadiq, S. W., Orlowska, M. E., Sadiq, W., "Specification and validation of process constraints for flexible workflows." *Inf. Syst.* 30, 5, 2005, 349-378.

[19] Schematron "A language for making assertions about patterns found in XML documents" Specification available at http://www.schematron.com/spec.html

[20] Tao L., "Shifting Paradigms with the Application Service Provider Model" *IEEE Computer, 34(10)*, pp. 32-39, 2001.

Appendix A: Variability Descriptor Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  targetNamespace="http://www.iaas.uni-stuttgart.de/schemas/VD"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.iaas.uni-stuttgart.de/schemas/VD"
  elementFormDefault="qualified" blockDefault="#all">
  <xsd:annotation>
    <xsd:documentation>
      Schema for Variability Descriptors Last Edited: 11.1.2008
    </xsd:documentation>
  </xsd:annotation>
  <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd" />

  <xsd:element name="variabilityDescriptor"
    type="VariabilityDescriptorType">
  </xsd:element>

  <xsd:complexType name="ExtensibleElementsType" mixed="true">
    <xsd:sequence>
      <xsd:element ref="documentation" minOccurs="0"
        maxOccurs="unbounded" />
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
        maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="documentation" type="DocumentationType" />

  <xsd:complexType name="DocumentationType" mixed="true">
    <xsd:sequence>
      <xsd:any processContents="lax" minOccurs="0"
        maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="source" type="xsd:anyURI" />
    <xsd:attribute ref="xml:lang" />
  </xsd:complexType>

  <xsd:complexType name="VariabilityDescriptorType" mixed="true">
    <xsd:complexContent>
      <xsd:extension base="ExtensibleElementsType">
        <xsd:sequence>
          <xsd:element name="variabilityPoints"
            type="VariabilityPointListType" maxOccurs="1"
            minOccurs="1">
          </xsd:element>
          <xsd:element name="dependencies" type="DependencyListType"
            maxOccurs="1" minOccurs="1">
          </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="expressionLanguage" type="xsd:anyURI"
          default="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
        </xsd:attribute>
        <xsd:attribute name="targetNamespace"
                       type="xsd:anyURI"></xsd:attribute>
      </xsd:extension>
```

```
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="DependencyListType">
    <xsd:sequence>
      <xsd:element name="dependency" type="DependencyType"
        maxOccurs="unbounded" minOccurs="0">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="VariabilityPointListType">
    <xsd:sequence>
      <xsd:element name="variabilityPoint" type="VariabilityPointType"
        maxOccurs="unbounded" minOccurs="1">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="DependencyType">
    <xsd:sequence>
      <xsd:element name="sources" type="SourceListType"></xsd:element>
      <xsd:element name="target" type="TargetType"></xsd:element>
      <xsd:element name="enablingConditions"
        type="EnablingConditionListType">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="VariabilityPointType" mixed="true">
    <xsd:complexContent>
      <xsd:extension base="ExtensibleElementsType">
        <xsd:sequence>
          <xsd:element name="import" type="ExpressionType" maxOccurs="1"
            minOccurs="0">
          </xsd:element>
          <xsd:element name="locators" type="LocatorListType">
          </xsd:element>
          <xsd:element name="alternatives" type="alternativeListType">
          </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:NCName"
                       use="required"></xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="LocatorListType">
    <xsd:sequence>
      <xsd:element name="locator" type="ExpressionType"
        maxOccurs="unbounded" minOccurs="0">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="alternativeListType">
    <xsd:sequence>
      <xsd:element name="alternative" type="AlternativeType"
        maxOccurs="unbounded" minOccurs="0">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
```

```
<xsd:complexType name="ExpressionType" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##other" processContents="lax" minOccurs="0"
      maxOccurs="unbounded">
    </xsd:any>
  </xsd:sequence>
  <xsd:attribute name="expressionLanguage" type="xsd:anyURI"
    default="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
  </xsd:attribute>
</xsd:complexType>

<xsd:complexType name="LocatorType" mixed="true">
  <xsd:complexContent>
    <xsd:extension base="ExtensibleElementsType">
      <xsd:sequence>
        <xsd:element name="xPathLocator" type="ExpressionType">
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="AlternativeType" mixed="true">
  <xsd:complexContent>
    <xsd:extension base="ExtensibleElementsType">
      <xsd:choice>
        <xsd:element name="explicitAlternative"
          type="ExplicitAlternativeType">
        </xsd:element>
        <xsd:element name="freeAlternative" type="ExpressionType">
        </xsd:element>
        <xsd:element name="expressionAlternative"
          type="ExpressionType">
        </xsd:element>
        <xsd:element name="emptyAlternative"
          type="EmptyAlternativeType">
        </xsd:element>
        <xsd:element name="locatorAlternative"
          type="LocatorAlternativeType">
        </xsd:element>
        <xsd:any namespace="##other" processContents="lax"></xsd:any>
      </xsd:choice>
      <xsd:attribute name="name" type="xsd:NCName"
                     use="required"></xsd:attribute>
      <xsd:attribute name="default" type="xsd:boolean" use="optional"
        default="false">
      </xsd:attribute>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ExplicitAlternativeType" mixed="true">
  <xsd:sequence>
    <xsd:any processContents="lax" namespace="##other" minOccurs="0"
      maxOccurs="unbounded">
    </xsd:any>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="EmptyAlternativeType"></xsd:complexType>
```

```
  <xsd:complexType name="LocatorAlternativeType"></xsd:complexType>

  <xsd:complexType name="TargetType" mixed="true">
    <xsd:complexContent>
      <xsd:extension base="ExtensibleElementsType">
        <xsd:sequence>
          <xsd:element ref="variabilityPointRef" maxOccurs="1"
            minOccurs="1">
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="SourceListType" mixed="true">
    <xsd:complexContent>
      <xsd:extension base="ExtensibleElementsType">
        <xsd:sequence>
          <xsd:element ref="variabilityPointRef" maxOccurs="unbounded"
            minOccurs="1">
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="variabilityPointRef" type="xsd:QName"></xsd:element>

  <xsd:complexType name="EnablingConditionListType">
    <xsd:sequence>
      <xsd:element name="enablingCondition" type="EnablingConditionType"
        maxOccurs="unbounded" minOccurs="0">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="EnablingConditionType">
    <xsd:sequence>
      <xsd:element name="condition" type="ExpressionType"></xsd:element>
      <xsd:element name="enabledAlternatives"
        type="EnabledAlternativeListType">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="EnabledAlternativeListType" mixed="true">
    <xsd:complexContent>
      <xsd:extension base="ExtensibleElementsType">
        <xsd:sequence>
          <xsd:element name="alternativeRef" type="xsd:QName"
            maxOccurs="unbounded" minOccurs="0">
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```