

Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Studienarbeit Nr. 2124

**A Graphical Tool for
Modeling BPEL 2.0
processes**

David Schumm

Studiengang:	Informatik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dimka Karastoyanova
begonnen am:	01.06.2007
beendet am:	11.10.2007
CR-Klassifikation:	D.2.2, H.4.1, H.5.2

Table of Contents

Abstract	1
1. Introduction.....	2
1.1. Requirements	2
1.2. Notational Conventions.....	2
2. Comparison of BPEL 1.1 and BPEL 2.0.....	3
2.1. Static Analysis	3
2.2. Structure of a Business Process.....	3
2.2.1. Process Attributes.....	3
2.2.2. Document Linking	4
2.2.3. Conditional Linking	5
2.3. Partner Link Types, Partner Links, Partners and Endpoint References	6
2.3.1. Partner Link Types.....	6
2.3.2. Partner Links.....	6
2.3.3. Partners	7
2.3.4. Endpoint References	7
2.4. Data Handling	8
2.4.1. Variables	8
2.4.2. Assignment	11
2.5. Correlation	12
2.6. Basic Activities.....	13
2.6.1. Receive	13
2.6.2. Wait.....	14
2.6.3. Extension Activity.....	14
2.6.4. Terminate / Exit Activity	15
2.6.5. Rethrow Activity	15
2.7. Structured Activities	16
2.7.1. If / Switch Activity	16
2.7.2. While Activity	17
2.7.3. repeatUntil Activity	18
2.7.4. forEach Activity	18
2.8. Scopes.....	19
2.8.1. Scope Initialization	20
2.8.2. Message Exchange Handling	20
2.8.3. Process State Usage in Compensation Handlers.....	21
2.8.4. Invoking a Compensation Handler.....	21
2.8.5. Fault Handlers	22
2.8.6. Event Handlers	23
2.9. Extensions	25
2.10. Abstract Processes.....	26
2.10.1. Common Base	26
2.10.2. Opaque Language Extensions	26
2.10.3. Omission	26
2.10.4. Abstract Process Profiles.....	26
2.10.5. Observable Behavior	27
2.10.6. Process Template	27
3. Structure of the Graphical Editor	28
3.1. Eclipse, EMF and GEF	28
3.2. General Structure of the Graphical Editor.....	29
3.3. Structure of the Controller.....	31
3.4. Basic Structure of the Data Model	32

4. Analysis of the Extensibility of the Existing Tool.....	33
4.1. BPEL 1.1 Data Model Generation	33
4.2. BPEL 2.0 Data Model Generation	33
4.3. BPEL 1.1 Data Model Integration	33
4.4. BPEL 2.0 Data Model Integration	34
4.5. BPMN Integration	34
4.5.1. Mapping BPMN to BPEL.....	34
4.5.2. Mapping BPEL to BPMN.....	35
4.5.3. Integration Approaches.....	35
5. BPEL 2.0 model integration	37
5.1. Integration overview.....	37
5.2. Integration of new attributes	38
5.2.1. BPEL output generator	38
5.2.2. Property View Provider	38
5.2.3. Abstract Activity Interface	39
5.2.4. Abstract Activity Implementation.....	40
5.2.5. Activity Implementation.....	40
5.2.6. Model Descriptor.....	41
5.2.7. Model Descriptor Implementation	42
5.3. Integration of new constructs.....	43
5.3.1. BPEL output generator	43
5.3.2. Validate Edit Part	44
5.3.3. Edit Parts factory	45
5.3.4. Editor palette.....	46
5.3.5. Insertion of icons for the activity into project.....	46
5.3.6. Commands on the Edit Part.....	47
5.3.7. Display in the Property View	47
5.3.8. Registering at the Model Creation Factory	47
5.3.9. BPEL Model Factory.....	48
5.3.10. BPEL Model Factory implementation.....	48
5.3.11. Model Descriptor.....	48
5.3.12. Model Descriptor Implementation	49
5.3.13. Adapting the process definition.....	51
5.3.14. Adapting the process implementation.....	51
5.3.15. Generating activity classes	53
5.3.16. Adapting the activity definition	53
5.3.17. Adaptation of the activity implementation	53
5.4. Removal of attributes and elements	56
5.4.1. BPEL output generator	56
5.4.2. Related Construct Implementation.....	56
5.4.3. Model Descriptor.....	57
5.4.4. Model Descriptor Implementation	57
5.4.5. Optional modifications.....	58
5.5. Attributes and nested XML elements.....	59
5.5.1. BPEL output generator	59
5.6. Activity renaming	60
5.6.1. BPEL output generator	60
5.6.2. Editor palette.....	60
6. Accomplished BPEL 2.0 Extensions	61
6.1. New constructs	61
6.2. New attributes	62
6.3. Removed constructs	62
6.4. Removed attributes.....	63
6.5. Attributes and nested XML elements.....	63

6.6. Activity renaming	63
6.7. Bug-fixing and Additional Features	63
6.7.1. Attribute values	63
6.7.2. Dynamic link naming	64
6.7.3. Activity traversing at BPEL output generation	64
6.7.4. Activity type display in the Property View	65
7. Remaining BPEL 2.0 Extensions	66
7.1. New constructs	66
7.2. New attributes	66
7.3. Removal of attributes	66
7.4. Attributes and nested XML elements	67
7.5. Construct renaming	67
8. Discussion and Outlook	68
8.1. Discussion	68
8.2. Outlook	68
Appendices	70
References	70
Used Resources	73
Erklärung	74

Table of Listings

Listing 1: BPEL Process Attributes Schema	4
Listing 2: BPEL Process Attributes Example	4
Listing 3: Import Statement Schema	4
Listing 4: Import Statement Example	5
Listing 5: Conditional Linking Schema	5
Listing 6: Conditional Linking Example	5
Listing 7: Partner Link Type Schema	6
Listing 8: Partner Link Type Example	6
Listing 9: Partner Definition Schema in BPEL1.1	7
Listing 10: Partner Definition Example in BPEL 1.1	7
Listing 11: Endpoint References in Service Reference Containers Schema	8
Listing 12: Endpoint References in Service Reference Containers Example	8
Listing 13: Endpoint Reference Schema in WS-Addressing	8
Listing 14: New activity validate	9
Listing 15: Variable Access Example	9
Listing 16: PropertyAlias Schema	9
Listing 17: PropertyAlias Example	10
Listing 18: Variable In-line Initialization Example	10
Listing 19: fromParts Schema	10
Listing 20: toParts Schema	10
Listing 21: fromParts / toParts Example	10
Listing 22: Variable Assignment Schema	11
Listing 23: Variable Assignment Example	12
Listing 24: Correlation Schema	12
Listing 25: Correlation Example	12
Listing 26: Receive Schema	13
Listing 27: Receive Example	13
Listing 28: Wait Activity Schema	14
Listing 29: Wait Activity Example	14
Listing 30: Extension Activity Schema	15
Listing 31: Extension Activity Example	15
Listing 32: Terminate / Exit Schema	15
Listing 33: Terminate / Exit Example	15
Listing 34: Rethrow Activity Schema	16
Listing 35: Rethrow Activity Example	16
Listing 36: If / Switch Activity Schema	16
Listing 37: If / Switch Activity Example	17
Listing 38: While Activity Schema	17
Listing 39: While Activity Example	18
Listing 40: repeatUntil Activity Schema	18
Listing 41: repeatUntil Activity Example	18
Listing 42: forEach Activity Schema	19
Listing 43: forEach Activity Example	19
Listing 44: Scope Schema	20
Listing 45: Scope Example	20
Listing 46: Compensation Invocation Schema	21
Listing 47: Compensation Invocation Example	22
Listing 48: Default Fault Handler	23
Listing 49: Fault Handler Example	23
Listing 50: Default Compensation Handler	23
Listing 51: Compensation Handler Example	23
Listing 52: Default Termination Handler	23
Listing 53: Termination Handler Example	23

Listing 54: Event Handler Schema	24
Listing 55: Event Handler Example.....	24
Listing 56: Process Extension Schema	25
Listing 57: Process Extension Example.....	25
Listing 58: Observable Behavior Reference	27
Listing 59: Process Template Reference.....	27
Listing 60: Adaptation of BPEL output generator for attribute integration.....	38
Listing 61: Initialization of allowed attribute values array.....	38
Listing 62: Initialization of allowed attribute values	39
Listing 63: Getter function for the Property Descriptor.....	39
Listing 64: Getter function for the Property Values	39
Listing 65: Setter function for the Property Values	39
Listing 66: Abstract Activity Definition for getter and setter function of the attribute	39
Listing 67: Definition of default values for the attribute in the activity implementation	40
Listing 68: Implementation of the getter and setter function of the attribute	40
Listing 69: Implementation of the interface for the controller: eGet	40
Listing 70: Implementation of the interface for the controller: eSet.....	41
Listing 71: Implementation of the interface for the controller: eUnset.....	41
Listing 72: Implementation of the interface for the controller: elsSet.....	41
Listing 73: Definition of a feature number	41
Listing 74: Adding the static feature number to the implementation	42
Listing 75: Initial creation of the attribute in the model implementation	42
Listing 76: Modeling the attribute.....	42
Listing 77: Adding the function call for the validate subtree.....	43
Listing 78: Building the validate subtree	43
Listing 79: Edit Part for the validate activity	44
Listing 80: Edit Parts factory including the Edit Part for validate	45
Listing 81: Adding the activity to the editor palette	46
Listing 82: Extension of the delete command	47
Listing 83: Extension of the create command.....	47
Listing 84: Registering at the Model Creation Factory.....	48
Listing 85: Definition of the function for instance creation in the BPEL Model Factory.....	48
Listing 86: General instance creation function.....	48
Listing 87: Specific instance creation function	48
Listing 88: Registering the activity at the model descriptor.....	49
Listing 89: Model Descriptor Implementation.....	50
Listing 90: Create contents of the Model Descriptor Implementation	50
Listing 91: Initialize contents of the Mode Descriptor Implementation	50
Listing 92: Adapting the process definition	51
Listing 93: Adding getter and setter of the activity to the process implementation	51
Listing 94: Adapting the elInverseRemove function of the process implementation	52
Listing 95: Adapting eGet on the process implementation	52
Listing 96: Adapting eSet on the process implementation.....	52
Listing 97: Adapting eUnset on the process implementation.....	52
Listing 98: Adapting elsSet on the process implementation	53
Listing 99: Adapted activity defintion	53
Listing 100: New imports in the activity implementation	54
Listing 101: Adaptation the elInverseRemove function of the activity implementation	54
Listing 102: Adapting eGet on the process implementation	54
Listing 103: Adapting eSet on the process implementation.....	55
Listing 104: Adapting eUnset on the process implementation.....	55
Listing 105: Adapting elsSet on the process implementation.....	55
Listing 106: Adaptation of BPEL output generator for attribute removal.....	56
Listing 107: Removing an attribute from the data model: eGet	57
Listing 108: Removing the dynamic feature number	57

Listing 109: Exchanging the static feature number in the implementation.....	57
Listing 110: Removing initial creation of the attribute in the model implementation	58
Listing 111: Removing the modeling of the attribute.....	58
Listing 112: Removing attribute on related construct interface.....	58
Listing 113: Removing attribute on related construct implementation	58
Listing 114: Removing unused model provider functions	58
Listing 115: Removing unused model annotation.....	59
Listing 116: Transformation of the attribute <code>until</code> into a nested XML element	59
Listing 117: Construct naming in the BPEL output generator.....	60
Listing 118: Construct naming in the editor palette.....	60
Listing 119: BPEL output generation of boolean values	63
Listing 120: Dynamic link naming	64
Listing 121: Activity traversing for scope construct.....	65
Listing 122: Display of activity type in the Property View.....	65

Table of Figures

Figure 1: EMF and GEF.....	28
Figure 2: General Structure of the Graphical Editor	29
Figure 3: Integration of an Activity into the controller.....	31
Figure 4: Connections of an activity in the data model	32
Figure 5: Icon for new construct <code>validate</code>	46
Figure 6: New constructs on the editor palette	61
Figure 7: New construct <code>validate</code> nested inside a scope.....	61
Figure 8: New construct <code>validate</code> in BPEL code.....	61
Figure 9: New constructs <code>forEach</code> and <code>extensionActivity</code> in a process.....	61
Figure 10: New construct <code>forEach</code> and <code>extensionActivity</code> in BPEL code	62
Figure 11: New attribute <code>suppressJoinFailure</code> in Eclipse Property View.....	62
Figure 12: Attribute transformed into nested XML element on activity <code>wait</code>	63
Figure 13: Activity <code>terminate</code> renamed to <code>exit</code> in the editor palette	63
Figure 14: Dynamic link naming	64
Figure 15: Activity type display in the Property View	65

Abstract

Nowadays Web Services (WS) are the most prominent technology for solving the key problem facing businesses - the Application Integration (AI). To elaborate on this, both, Intra Enterprise Integration (Enterprise Application Integration, EAI) and Integration with Business Partners (Business Process Integration, BPI) can be achieved by loosely coupled applications using WS interfaces.

Here the Business Process Execution Language for Web Services (BPEL) comes into play. It enables the definition of business processes as coordinated sets of WS interactions (Orchestration) recursively into new aggregated Web Services. Furthermore BPEL may be used to define the external behavior of a service (through an `Abstract Process`) as well as the internal implementation (through an `Executable Process`) [ACKM04, pp. 284].

Although there is a variety of languages for service orchestration, such as the business process modeling language [BPML], the language BPEL4WS initially proposed in July 2002 by BEA, Microsoft and IBM [BPEL1.0] has emerged as de facto standard in this area. It has been transferred to OASIS for standardization and was released in April 2007 as BPEL 2.0.

The basis for this thesis is a graphical process modeling tool, implemented as an Eclipse-Plugin [Kapl06], which was designed to be compliant with BPEL 1.1 standard. The objective of this thesis is to extend this tool in order to be compliant with the OASIS BPEL 2.0 standard.

1. Introduction

1.1. Requirements

For graphical modeling of BPEL 2.0 processes [BPEL2.0], an existing tool [Kapl06] shall be analysed and extended. In the first step the specifications BPEL 1.1 and BPEL 2.0 shall be compared and differences shall be pointed out. In the second step different approaches for extending the existing modeling tool shall be analysed and the most appropriate one used, occurring bugs and incompleteness shall be resolved. Furthermore the possibilities for adapting the tool to support the Business Process Modeling Notation (see [BPMN1.0] and [BPMN2.0]) shall be investigated.

As a result of the modeling a `.bpe1` file will be generated, which is executable on a BPEL 2.0 compliant engine. The implementation of the graphical process modeler in the form of an Eclipse-Plugin shall be maintained. The already implemented support for Templates shall be maintained as in the version of the modeling tool presented in [Kapl06]. Beyond, the integration and implementation of an `extensionActivity` with two Attributes and two Variables is required.

1.2. Notational Conventions

"The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119", [RFC2119, p.1]:

The BPEL specification [BPEL2.0, p.9] uses an informal syntax to describe the XML grammar of the XML fragments, which will also be used in this document:

- `<-- description -->` is a placeholder for elements from some "other" namespace (like `##other` in XSD).
- Characters are appended to elements, attributes, and as follows:
 - `"?"` (0 or 1)
 - `"**"` (0 or more)
 - `"+"` (1 or more)The characters "[" and "]" are used to indicate that contained items are to be treated as a group with respect to the "?", "**", or "+" characters.
- Elements and attributes separated by "|" and grouped by "(" and ")" are meant to be syntactic alternatives.

2. Comparison of BPEL 1.1 and BPEL 2.0

In the following section the modifications that have been made to BPEL 1.1 during the standardization to BPEL 2.0 are described. Where necessary, code is listed to support a better understanding of new or modified constructs and syntactical changes. Note, that BPEL was first released as Version 1.0 in 2002; this comparison only regards the modifications since Version 1.1.

2.1. Static Analysis

BPEL 1.1 takes as general principle that compliant implementations may choose to perform static analysis in order to detect and reject process definitions that may have undefined semantics. Such analysis is necessarily pessimistic and therefore might in some cases prevent the use of processes that would not, in fact, create situations with undefined semantics, either in specific uses or in any use. [BPEL1.1, p.14]

This optional functionality of a BPEL implementation has been altered in the BPEL 2.0 standard to an absolute requirement:

“BPEL 2.0 takes it as a general principle that conformant implementations must perform basic static analysis [BPEL2.0, Appendix B] to detect and reject process definitions that fail any of those static analysis checks” [BPEL2.0, p.13].

2.2. Structure of a Business Process

2.2.1. Process Attributes

The following fragments of the top level attributes of a process have been changed:

1. The description of the attribute `suppressJoinFailure` has been enriched:

When this attribute is not specified for an activity, it implicitly inherits its value from its closest enclosing activity or from the `<process>` if no enclosing activity specifies this attribute [BPEL2.0, p.23].

2. The attribute `enableInstanceCompensation` has been removed [BPEL2.0, p.21].

3. A new attribute, `exitOnStandardFault` has been added:

"If the value of this attribute is set to `yes`, then the process must exit immediately as if an `<exit>` activity (formerly known as `<terminate>`) has been reached, when a BPEL 2.0 standard fault other than `bpel:joinFailure` is encountered. If the value of this attribute is set to `no`, then the process can handle a standard fault using a fault handler. The default value for this attribute is `no`. When this attribute is not specified on a `<scope>` it inherits its value from its enclosing `<scope>` or `<process>`.

If the value of `exitOnStandardFault` of a `<scope>` or `<process>` is set to `yes`, then a fault handler that explicitly targets the BPEL 2.0 standard faults must not be used in that scope. A process definition that violates this condition must be detected by static analysis and must be rejected by a conformant implementation" [BPEL2.0, p.23].

4. The attribute `abstractProcess` has been removed; instead this can be distinguished by the target namespace (`xmlns`): The syntax of Abstract Process has its own distinct target namespace [BPEL2.0, p.23].

5. “Constructs that require or allow queries or expressions provide the ability to override the default query/expression language for individual queries/expressions” [BPEL2.0, p.49], this corresponds to the `expressionLanguage` attribute of a process.

Listing 1: BPEL Process Attributes Schema

A.) [BPEL1.1, p.24]:

```
<process name="ncname" targetNamespace="uri"
  queryLanguage="anyURI"?
  expressionLanguage="anyURI"?
  suppressJoinFailure="yes|no"?
  enableInstanceCompensation="yes|no"?
  abstractProcess="yes|no"?
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
```

B.) [BPEL2.0, p.21]:

```
<process name="NCName" targetNamespace="anyURI"
  queryLanguage="anyURI"?
  expressionLanguage="anyURI"?
  suppressJoinFailure="yes|no"?
  exitOnStandardFault="yes|no"?
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
```

Listing 2: BPEL Process Attributes Example

A.) [BPEL1.1, p.98]:

```
<process name="loanApprovalProcess"
  targetNamespace="http://acme.com/loanprocessing"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:lns="http://loans.org/wsd/loan-approval"
  suppressJoinFailure="yes">
```

B.) [BPEL2.0, p.175]:

```
<process name="OrderingServiceProcess"
  targetNamespace="http://example.com/ordering/"
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/abstract"
  xmlns:ext="http://example.com/bpel/some/extension"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  abstractProcessProfile="http://docs.oasisopen.org/wsbpel/2.0/process/
    abstract/simple-template/2006/08"
  suppressJoinFailure="yes">
```

2.2.2. Document Linking

The modified `<import>` element is used within a BPEL 2.0 process to declare a dependency on external XML Schema or WSDL definitions: “Any number of `<import>` elements may appear as children of the `<process>` element. The new and mandatory `importType` attribute identifies the type of document being imported by providing an absolute URI that identifies the encoding language used in the document” [BPEL2.0, p.32]. “The value of the `importType` attribute of element `<import>` must be set to `http://www.w3.org/2001/XMLSchema` when importing XML Schema 1.0 documents and to `http://schemas.xmlsoap.org/wsd/` when importing WSDL 1.1 documents” [BPEL2.0, p.195]. Other URI values may be used as well.

Listing 3: Import Statement Schema

A.) BPEL1.1:

```
<import namespace="URI"
  location="URI"/>
```

B.) BPEL2.0:

```
<import importType="URI"
        location="URI"
        namespace="URI" />
```

Listing 4: Import Statement Example

A.) [BPEL1.1, p.16]:

```
<import namespace="http://manufacturing.org/xsd/purchase"
        location="http://manufacturing.org/xsd/purchase.xsd"/>
```

B.) [BPEL2.0, p.169]:

```
<import importType="http://schemas.xmlsoap.org/wsdl/"
        location="shippingLT.wsdl"
        namespace="http://example.com/shipping/partnerLinkTypes/" />
<import importType="http://schemas.xmlsoap.org/wsdl/"
        location="shippingPT.wsdl"
        namespace="http://example.com/shipping/interfaces/" />
<import importType="http://schemas.xmlsoap.org/wsdl/"
        location="shippingProperties.wsdl"
        namespace="http://example.com/shipping/properties/" />
```

2.2.3. Conditional Linking

In BPEL 2.0 the syntax for expressing Join and Transition Conditions has been altered from XML attributes to nested XML elements.

Listing 5: Conditional Linking Schema

A.) [BPEL1.1, p.31, p.67]

```
<invoke name="activity" joinCondition="bool-expr">
    <target linkName="ncname"/>
    <source linkName="ncname" transitionCondition="bool-expr"?/>*
</invoke>
```

B.) [BPEL2.0, p.31]

```
<targets?>
    <joinCondition expressionLanguage="anyURI"?>?
        bool-expr
    </joinCondition>
    <target linkName="NCName" />+
</targets>
<sources?>
    <source linkName="NCName">+
        <transitionCondition expressionLanguage="anyURI"?>?
            bool-expr
        </transitionCondition>
    </source>
</sources>
```

Listing 6: Conditional Linking Example

A.) [BPEL1.1, p.31, p.67]

```
<invoke name="settleTrade" joinCondition="bpws:getLinkStatus('buyToSettle') and
bpws:getLinkStatus('sellToSettle')">
    <target linkName="getBuyerInformation"/>
    <target linkName="getSellerInformation"/>
    <source linkName="toBuyConfirm"/>
    <source linkName="toSellConfirm"/>
</invoke>
```

B.) [BPEL2.0, p.31]

```
<invoke name="settleTrade" ...>
  <targets>
    <joinCondition>$buyToSettle and $sellToSettle</joinCondition>
    <target linkName="buyToSettle" />
    <target linkName="sellToSettle" />
  </targets>
  <sources>
    <source linkName="toBuyConfirm" />
    <source linkName="toSellConfirm" />
  </sources>
</invoke>
```

2.3. Partner Link Types, Partner Links, Partners and Endpoint References

2.3.1. Partner Link Types

The syntax of Partner Link Type has been altered, in the BPEL 2.0 standard the `portType` declaration is no longer a nested XML element, instead it is written as an XML attribute of the `<role>` tag.

Listing 7: Partner Link Type Schema

A.) [BPEL1.1, p.34]:

```
<plnk:partnerLinkType name="ncname">
  <plnk:role name="ncname">
    <plnk:portType name="qname"/>
  </plnk:role>
</plnk:partnerLinkType>
```

B.) [BPEL2.0, p.37]:

```
<plnk:partnerLinkType name="NCName">
  <plnk:role name="NCName" portType="QName" />
</plnk:partnerLinkType>
```

Listing 8: Partner Link Type Example

A.) [BPEL1.1, p.97]:

```
<plnk:partnerLinkType name="loanPartnerLinkType">
  <plnk:role name="loanService">
    <plnk:portType name="lns:loanServicePT"/>
  </plnk:role>
</plnk:partnerLinkType>
```

B.) [BPEL2.0, p.166]:

```
<plnk:partnerLinkType name="shippingLT">
  <plnk:role name="shippingService"
    portType="sif:shippingServicePT" />
  <plnk:role name="shippingServiceCustomer"
    portType="sif:shippingServiceCustomerPT" />
</plnk:partnerLinkType>
```

2.3.2. Partner Links

A new attribute is introduced in Partner Links, `initializePartnerRole`:

"The `initializePartnerRole` attribute specifies whether the BPEL 2.0 processor is required to initialise a `<partnerLink>`'s `partnerRole` value. The attribute has no affect on the `partnerRole`'s value after its initialization. The `initializePartnerRole` attribute must

not be used on a partner link that does not have a partner role. If the `initializePartnerRole` attribute is set to `yes` then the BPEL 2.0 processor must initialize the EPR of the `partnerRole` before that EPR is first utilized by the BPEL 2.0 process. An example would be when an EPR is used in an `<invoke>` activity. If the `initializePartnerRole` attribute is set to `no` then the BPEL 2.0 processor must not initialize the EPR of the `partnerRole` before that EPR is first utilized by the BPEL 2.0 process. If the `initializePartnerRole` attribute is omitted (see 2.10.3) then the partner role may be initialised by a BPEL 2.0 processor.

When `initializePartnerRole` is set to `yes`, the EPR value used in `partnerRole` Initialization is typically specified as a part of BPEL 2.0 process deployment or execution environment configuration. Hence, the `initializePartnerRole` attribute may be used as a part of process deployment contract" [BPEL2.0, p.37].

2.3.3. Partners

The (optional) `Partner` definitions have completely been removed from the new BPEL 2.0 specification. Partner definitions had the following intention: "A partner is defined as a subset of the partner links of the process. From the process perspective a partner definition introduces a constraint on the functionality that a business partner is required to provide" [BPEL1.1, p.24].

Listing 9: Partner Definition Schema in BPEL1.1

[BPEL1.1, p.35]:

```
<partner name="ncname" xmlns="URI">
  <partnerLink name="ncname"/>+
</partner>
```

Listing 10: Partner Definition Example in BPEL 1.1

[BPEL1.1, p.24]:

```
<partner name="SellerShipper" xmlns="http://schemas.xmlsoap.org/partner-link/">
  <partnerLink name="Seller"/>
  <partnerLink name="Shipper"/>
</partner>
```

2.3.4. Endpoint References

In BPEL 1.1 Endpoint References were only explained in principle and a reference to the corresponding Web Service standard, WS-Addressing [W3C04], was given, in order to keep the standard modular and composable [BPEL1.1, p. 36].

In the BPEL 2.0 standard the Schema is specified: Endpoint References associated with `partnerRole` and `myRole` of `<partnerLink>`s are manifested as service reference containers (`<sref:service-ref>`) in the new standard. This container is used as an envelope to wrap the actual Endpoint Reference value. The `Address` element is the only mandatory element and the omitted attribute `PortName` in the schema (see Listing 12:) provided in BPEL 2.0 is optional. Thus the schema complies with the WS-Addressing standard (see Listing 13:).

The `<sref:service-ref>` has an optional attribute called `reference-scheme` to denote the URI of the reference interpretation scheme of service endpoint, which is the child element of `<sref:service-ref>` [BPEL2.0, p.39].

Listing 11: Endpoint References in Service Reference Containers Schema

[BPEL2.0, p.39]:

```
<sref:service-ref reference-scheme="http://example.org">
  <foo:barEPR xmlns:foo="http://example.org">...</foo:barEPR>
</sref:service-ref>
```

Listing 12: Endpoint References in Service Reference Containers Example

[BPEL2.0, p.188]:

```
<partnerLink name="auctionRegistrationService"
  partnerLinkType="as:auctionHouseAuctionRegistrationServiceLT"
  myRole="auctionHouse"
  partnerRole="auctionRegistrationService" />
...
<assign>
  <copy>
    <from>
      <sref:service-ref>
        <addr:EndpointReference>
          <addr:Address>
            http://example.com/auction/
              RegistrationService/
          </addr:Address>
          <addr:ServiceName>
            as:RegistrationService
          </addr:ServiceName>
        </addr:EndpointReference>
      </sref:service-ref>
    </from>
    <to partnerLink="auctionRegistrationService" />
  </copy>
</assign>
```

Listing 13: Endpoint Reference Schema in WS-Addressing

[W3C04, 2.2]:

```
<wsa:EndpointReference>
  <wsa:Address>xs:anyURI</wsa:Address>
  <wsa:ReferenceProperties>... </wsa:ReferenceProperties> ?
  <wsa:ReferenceParameters>... </wsa:ReferenceParameters> ?
  <wsa:PortType>xs:QName</wsa:PortType> ?
  <wsa:ServiceName PortName="xs:NCName"?>xs:QName</wsa:ServiceName> ?
  <wsp:Policy> ... </wsp:Policy>*
</wsa:EndpointReference>
```

2.4. Data Handling

2.4.1. Variables

1. The new standard introduces the complex XML Schema type for usage, previously only simple XML Schema type was allowed besides WSDL message type and XML Schema element [BPEL2.0, p.45].

2. Furthermore a new Variable related activity is introduced, `<validate>`: “Values stored in variables can be mutated during the course of process execution. The `<validate>` activity can be used to ensure that values of variables are valid against their associated XML data definition, including XML Schema simple type, complex type, element definition and XML definitions of WSDL parts. The `<validate>` activity has a `variables` attribute, listing the variables to validate” [BPEL2.0, p.48].

Listing 14: New activity validate

```
<validate variables="BPELVariableNames" standard-attributes>
  standard-elements
</validate>
```

3. The usage patterns for XPath variable access have been clarified and simplified by leveraging "\$" syntax, e.g. `$myFooVar/lines/line[2]/text()` [BPEL2.0, pp.49]. In BPEL 1.1 variable access is performed using the `getVariableData` function.

Listing 15: Variable Access Example

A.) [BPEL 1.1, pp. 99]:

```
<variables>
  <variable name="request" messageType="lns:creditInformationMessage"/>
</variables>
<from expression="bpws:getVariableData('request','amount')"/>
<to variable="itemsShipped"/>
```

B.) [BPEL 2.0, p. 60]:

```
<xsd:element name="StatusContainer">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="statusDescription" type="xsd:string"
        form="qualified" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
...
<variable name="AStatus" element="e:StatusContainer" />
...
$AStatus/e:statusDescription <!-- Access to variable -->
```

4. "The notion of aliasing is introduced to map a property to a field in a specific message part or variable value. The property name becomes an alias for the message part and / or location, and can be used as such in expressions and assignments" [BPEL2.0, p.41].

In the section formerly named Message Properties, two new attributes for `propertyAlias` are introduced, called `element` and `type`, with mutual exclusive usage, see [BPEL2.0, p.43] and Listing 16: B.

Listing 16: PropertyAlias Schema

A.) [BPEL1.1, p.135]:

```
<element name="propertyAlias">
  <complexType>
    <attribute name="propertyName" type="QName" use="required"/>
    <attribute name="messageType" type="QName" use="required"/>
    <attribute name="part" type="NCName"/>
    <attribute name="query" type="string"/>
  </complexType>
</element>
```

B.) [BPEL2.0, pp. 42]:

```
<vprop:propertyAlias
  propertyName="QName"
  messageType="QName"?
  part="NCName"?
  type="QName"?
  element="QName"?>
  <vprop:query queryLanguage="anyURI"?>?
    queryContent
  </vprop:query>
</vprop:propertyAlias>
```

Listing 17: PropertyAlias Example

A.) [BPEL1.1, p.104]:

```
<bpws:propertyAlias
  propertyName="tns:auctionId"
  messageType="tns:sellerData"
  part="auctionId"/>
```

B.) [BPEL2.0, p.54]:

```
<vprop:propertyAlias
  propertyName="p:price"
  messageType="my:POMsg"
  part="poPart">
  <vprop:query>price</vprop:query>
</vprop:propertyAlias>
```

5. A variable can optionally be initialised by using an in-line from-spec [BPEL2.0, p.48].

Listing 18: Variable In-line Initialization Example

[BPEL2.0, p. 121]:

```
<variables>
  <variable name="V1" type="xsd:int">
    <from>0</from>
  </variable>
</variables>
```

6. The `<fromParts>` element in receiving activities like `<receive>`, `<onEvent>` etc. is used as an alternative to indicate that the data from a received message is to be directly copied to BPEL variables from a corresponding anonymous WSDL message variable. It may not be used together with the `variable` attribute of the activity. Similarly, the `<toParts>` element is used as an alternative to have data from BPEL variables directly copied into an anonymous WSDL message used by a sending activity like `<reply>`, `<invoke>`, etc. [BPEL2.0, p.88, p.91].

Listing 19: fromParts Schema

[BPEL2.0, p.22]:

```
<fromParts>?
  <fromPart part="NCName" toVariable="BPELVariableName" />+
</fromParts>
```

Listing 20: toParts Schema

[BPEL2.0, p.25]:

```
<toParts>?
  <toPart part="NCName" fromVariable="BPELVariableName" />+
</toParts>
```

Listing 21: fromParts / toParts Example

fromParts in Receive Activity:

```
<receive messageExchange="supplier"
  partnerLink="businessPartner"
  portType="businessPT"
  operation="update">
  <fromParts>
```

```

        <fromPart toVariable="updatedData" />
    </fromParts>
</receive>

```

fromParts and toParts in Invoke Activity:

```

<invoke name="purchase"
  partnerLink="Seller"
  portType="SP:Purchasing"
  operation="Purchase"/>
  <toParts>
    <toPart fromVariable="sendPO" />
  </toParts>
  <fromParts>
    <fromPart toVariable="getResponse" />
  </fromParts>
</invoke>

```

Without using fromParts and toParts:

```

<invoke name="purchase"
  partnerLink="Seller"
  portType="SP:Purchasing"
  operation="Purchase"
  inputVariable="sendPO"
  outputVariable="getResponse" />

```

2.4.2. Assignment

The `<assign>` activity is described more precise, and has been enriched with some features:

1. The optional `validate` attribute can be used with the `<assign>` activity. When `validate` is set to `yes`, the `<assign>` activity validates all the variables being modified by the activity against their schema definition [BPEL2.0, p.63].
2. A new extension is introduced, the `<extensionAssignOperation>` element. It is possible to include extensible data manipulation operations defined as extension elements under namespaces different from the WS-BPEL namespace [BPEL2.0, p.59, p.63]. The specification does not provide a detailed explanation, nor the intention of this extension.
3. The new optional `keepSrcElementName` attribute of the `<copy>` construct in an `<assign>` activity is used to specify whether the element name of the destination (as selected by the to-spec) will be replaced by the element name of the source (as selected by the from-spec) during the copy operation [BPEL2.0, p.63].
4. The copy mechanism, when combined with the default XPath 1.0 expression language, cannot perform complex XML transformations. To address this restriction in a portable fashion, a BPEL 2.0 processor must support the new standardized `bpel:doXsltTransform()` XPath 1.0 extension function [BPEL2.0, pp.63].
5. The syntax for copying from a literal value has changed, in BPEL 2.0 it has to be enveloped: `<from><literal>literal value</literal></from>` [BPEL2.0, p.63], formerly this was not necessary [BPEL1.1, p.43].

Listing 22: Variable Assignment Schema

```

<assign standard-attributes>
  standard-elements

```

```

        <copy>+
            from-spec
            to-spec
        </copy>
    </assign>

```

Listing 23: Variable Assignment Example

A.) [BPEL1.1, p.45]:

```

<assign>
    <copy>
        <from variable="c1"/>
        <to variable="c2"/>
    </copy>
</assign>

```

B.) [BPEL2.0, p.183]:

```

<assign>
    <copy>
        <from>
            <literal>yes</literal>
        </from>
        <to variable="approval" part="accept" />
    </copy>
</assign>

```

2.5. Correlation

The properties in Correlation sets have been enriched; the `initiate` attribute has a new legal value, `join`. When the `initiate` attribute is set to `join`, the related activity must attempt to initiate the correlation set, if the correlation set is not yet initiated [BPEL2.0, pp.76]. Also the naming of pattern values has been modified.

Listing 24: Correlation Schema

A.) [BPEL1.1, p.54]:

```

<correlations>?
    <correlation set="ncname"
        initiate="yes|no"?
        pattern="in|out|out-in"/>+
</correlations>

```

B.) [BPEL2.0, p.78]:

```

<correlations>
    <correlation set="NCName"
        initiate="yes|join|no"?
        pattern="request|response|request-response"? />+
</correlations>

```

Listing 25: Correlation Example

A.) [BPEL1.1, p.72]:

```

<correlations>
    <correlation set="PurchaseOrder"
        initiate="yes"
        pattern="out"/>
</correlations>

```

B.) [BPEL2.0, pp.81]:

```
<correlations>
  <correlation set="PurchaseOrder"
    initiate="yes"
    pattern="request" />
  <correlation set="Invoice"
    initiate="yes"
    pattern="response" />
</correlations>
```

2.6. Basic Activities

2.6.1. Receive

The new optional `messageExchange` attribute is used to disambiguate the relationship between `<receive>` and `<reply>` activities. The explicit use of `messageExchange` is needed only where the execution can result in multiple `<receive>`-`<reply>` pairs on the same `partnerLink` and operation [BPEL2.0, pp.93]. The value of the attribute is a user-defined `NCName`.

Listing 26: Receive Schema

A.) [BPEL1.1, p.27]:

```
<receive partnerLink="ncname"
  portType="qname"
  operation="ncname"
  variable="ncname"?
  createInstance="yes|no"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="ncname" initiate="yes|no"?>+
  </correlations>
</receive>
```

B.) [BPEL2.0, pp.25]:

```
<receive partnerLink="NCName"
  portType="QName"?
  operation="NCName"
  variable="BPELVariableName"?
  createInstance="yes|no"?
  messageExchange="NCName"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="NCName" initiate="yes|join|no"? />+
  </correlations>
  <fromParts>?
    <fromPart part="NCName" toVariable="BPELVariableName" />+
  </fromParts>
</receive>
```

Listing 27: Receive Example

A.) [BPEL1.1, p.83]:

```
<receive name="getOrder"
  partnerLink="buyer"
  portType="car"
  operation="order"
  variable="orderDetails"
  createInstance="yes" />
```

B.) [BPEL2.0]:

```
<messageExchanges>
  <messageExchange name="supplier"/>
  <messageExchange name="manufacturer"/>
</messageExchanges>
...
<receive messageExchange="supplier"
  partnerLink="businessPartner"
  portType="businessPT"
  operation="update" />
...
<receive messageExchange="manufacturer"
  partnerLink="businessPartner"
  portType="businessPT"
  operation="update" />
```

2.6.2. Wait

The attributes for the `<wait>` activity have syntactically changed into nested XML elements:

Listing 28: Wait Activity Schema

A.) [BPEL1.1]:

```
<wait until="time"/>
```

B.) BPEL2.0:

```
<wait>
  <until>time</until>
</wait>
```

Listing 29: Wait Activity Example

A.) [BPEL1.1, p.58]

```
<wait until="'2002-12-24T18:00+01:00'"/>
```

B.) [BPEL2.0, p.95]

```
<wait>
  <until>'2002-12-24T18:00+01:00'</until>
</wait>
```

Also the attribute `for` has been altered to a nested XML element, it can be used instead of `until`.

2.6.3. Extension Activity

A BPEL process definition can include new activities, which are not defined by the BPEL 2.0 specification, by placing them inside the new `<extensionActivity>` element. These activities are known as extension activities. An `<extensionActivity>` may be also a structured activity.

Listing 30: Extension Activity Schema

[BPEL2.0, p.95]:

```
<extensionActivity>
  <anyElementQName standard-attributes>
    standard-elements
  </anyElementQName>
</extensionActivity>
```

Listing 31: Extension Activity Example

BPEL2.0:

```
<extensionActivity>
  <ext:suspend/>
</extensionActivity>
```

2.6.4. Terminate / Exit Activity

The `<terminate>` activity has been renamed to `<exit>` activity. It is used to immediately end a business process instance. "In case an `<exit>` activity has been reached, no fault handling, compensation handling or termination handling is being executed" [BPEL2.0, p.96].

Listing 32: Terminate / Exit Schema

A.) [BPEL1.1, p.87]:

```
<terminate standard-attributes>
  standard-elements
</terminate>
```

B.) [BPEL2.0, p.96]:

```
<exit standard-attributes>
  standard-elements
</exit>
```

Listing 33: Terminate / Exit Example

A.) [BPEL1.1, p.81]:

```
<onMessage partnerLink="buyer"
  portType="car"
  operation="cancel"
  variable="cancelDetails">
  <terminate/>
</onMessage>
```

B.) [BPEL2.0, p.162]:

```
<faultHandlers>
  <catchAll>
    <exit />
  </catchAll>
</faultHandlers>
```

2.6.5. Rethrow Activity

The new `<rethrow>` activity is used in fault handlers to rethrow the fault they caught, i.e. the fault name and, where present, the fault data of the original fault. The fault is thrown to

the parent scope [BPEL2.0, p.127]. The `<rethrow>` activity can be used only within a fault handler (`<catch>` and `<catchAll>`). “Modifications to the fault data must be ignored by `<rethrow>`” [BPEL2.0, p.96]. For example, if the logic in a fault handler modifies the fault data and then calls `<rethrow>`, the original fault data would be rethrown and not the modified fault data.

Listing 34: Rethrow Activity Schema

[BPEL2.0, p.30]:

```
<rethrow standard-attributes>
  standard-elements
</rethrow>
```

Listing 35: Rethrow Activity Example

[BPEL2.0, p.132]:

```
<catchAll>
  <sequence>
    <compensate />
    <rethrow />
  </sequence>
</catchAll>
```

2.7. Structured Activities

2.7.1. If / Switch Activity

The `<if>` activity in BPEL 2.0 provides conditional behavior and replaces the BPEL 1.1 activity `<switch>`. The activity consists of an ordered list of one or more conditional branches defined by the `<if>` and optional `<elseif>` elements, followed by an optional `<else>` element [BPEL2.0, p.99].

Listing 36: If / Switch Activity Schema

A.) [BPEL1.1, p.29]:

```
<switch standard-attributes>
  standard-elements
  <case condition="bool-expr">+
    activity
  </case>
  <otherwise>?
    activity
  </otherwise>
</switch>
```

B.) [BPEL2.0, p.99]:

```
<if standard-attributes>
  standard-elements
  <condition expressionLanguage="anyURI"?>bool-expr</condition>
  activity
  <elseif>+
    <condition expressionLanguage="anyURI"?>bool-expr</condition>
    activity
  </elseif>
  <else>?
    activity
  </else>
</if>
```

Listing 37: If / Switch Activity Example

A.) [BPEL1.1, p.60]:

```
<switch xmlns:inventory="http://supply-chain.org/inventory"
  xmlns:FLT="http://example.com/faults">
  <case condition="bpws:getVariableProperty(stockResult,level) > 100">
    <flow>
      <!-- perform fulfillment work -->
    </flow>
  </case>
  <case condition="bpws:getVariableProperty(stockResult,level) >= 0">
    <throw faultName="FLT:OutOfStock" variable="RestockEstimate"/>
  </case>
  <otherwise>
    <throw faultName="FLT:ItemDiscontinued"/>
  </otherwise>
</switch>
```

B.) [BPEL2.0, p.171]:

```
<if>
  <condition>
    bpel:getVariableProperty('shipRequest', 'props:shipComplete')
  </condition>
  <sequence>
    <invoke partnerLink="customer" operation="shippingNotice"
      inputVariable="shipNotice">
    </invoke>
  </sequence>
</if>
```

2.7.2. While Activity

The `<while>` activity provides for the repeated execution of the contained activities. In the BPEL 1.1 specification the `<while>` activity was described in a way, that nested activities are executed until the Boolean condition evaluates to true [BPEL1.1, p.60]. This has been clarified in BPEL2.0; the contained activity is now executed as long as the Boolean `<condition>` evaluates to true at the beginning of each iteration [BPEL2.0, p.99]. Also a part of the syntax has been altered: The condition attribute is now written as a nested XML element.

Listing 38: While Activity Schema

A.) [BPEL1.1, p.60]:

```
<while condition="bool-expr" standard-attributes>
  standard-elements
  activity
</while>
```

B.) [BPEL2.0, p.99]:

```
<while standard-attributes>
  standard-elements
  <condition expressionLanguage="anyURI"?>bool-expr</condition>
  activity
</while>
```

Listing 39: While Activity Example

A.) [BPEL1.1, p.60]:

```
<while condition="bpws:getVariableData(orderDetails) > 100">
  <scope>
    ...
  </scope>
</while>
```

B.) [BPEL2.0, p.100]:

```
<while>
  <condition>$orderDetails > 100</condition>
  <scope>...</scope>
</while>
```

2.7.3. repeatUntil Activity

The contained activity is executed until the given Boolean `<condition>` becomes true. The condition is tested after each execution of the body of the loop. In contrast to the `<while>` activity, the `<repeatUntil>` loop executes the contained activity at least once [BPEL2.0, p.100].

Listing 40: repeatUntil Activity Schema

[BPEL2.0, p.100]:

```
<repeatUntil standard-attributes>
  standard-elements
  activity
  <condition expressionLanguage="anyURI"?>bool-expr</condition>
</repeatUntil>
```

Listing 41: repeatUntil Activity Example

BPEL2.0:

```
<repeatUntil>
  <scope>...</scope>
  <condition>$orderDetails > 100</condition>
</repeatUntil>
```

2.7.4. forEach Activity

A new activity is introduced in BPEL 2.0, the serial or parallel `<forEach>`. By attaching an optional `<completionCondition>` it also provides for “at least N out of M” semantics [BPEL2.0, p.114]:

"The `<forEach>` activity will execute its contained `<scope>` activity exactly N+1 times where N equals the `<finalCounterValue>` minus the `<startCounterValue>`. If the value of the parallel attribute is “yes” then the activity is a parallel `<forEach>`. The enclosed `<scope>` activity must be concurrently executed N+1 times.

The `<forEach>` activity without a `<completionCondition>` completes when its’ entire child `<scope>`s have completed. The `<completionCondition>` element is optionally specified to prevent some of the children from executing (in the serial case), or to force early termination of some of the children (in the parallel case).

The `<branches>` element has an optional `successfulBranchesOnly` attribute with the default value of `no`. If the value of `successfulBranchesOnly` is `no`, all `<scope>`s which have completed (successfully or unsuccessfully) must be counted. If `successfulBranchesOnly` is `yes`, only `<scope>`s which have completed successfully must be counted" [BPEL2.0, pp.112].

Listing 42: forEach Activity Schema

A.) [BPEL2.0, p.112]:

```
<forEach counterName="BPELVariableName" parallel="yes|no"
  standard-attributes>
  standard-elements
  <startCounterValue expressionLanguage="anyURI"?>
    unsigned-integer-expression
  </startCounterValue>
  <finalCounterValue expressionLanguage="anyURI"?>
    unsigned-integer-expression
  </finalCounterValue>
  <completionCondition?>
    <branches expressionLanguage="anyURI"?
      successfulBranchesOnly="yes|no"?>
        unsigned-integer-expression
      </branches>
    </completionCondition>
  <scope ...>...</scope>
</forEach>
```

Listing 43: forEach Activity Example

A.) BPEL2.0:

```
<forEach counterName="counter" parallel="no">
  <startCounterValue>1</startCounterValue>
  <finalCounterValue>50</finalCounterValue>
  <completionCondition>
    <branches successfulBranchesOnly="yes">
      10
    </branches>
  </completionCondition>
  <scope>...</scope>
</forEach>
```

2.8. Scopes

Scopes became enriched by two new attributes, `isolated` and `exitOnStandardFault`:

1. The former `variableAccessSerializable` attribute [BPEL1.1, p.84] has been renamed to the `isolated` attribute of a scope. When set to `yes`, it provides control of concurrent access to shared resources: variables, partner links, and control dependency links. Such a scope is called an isolated scope. "Suppose two concurrent isolated scopes, S1 and S2, access a common set of variables and partner links (external to them) for read or write operations. The semantics of isolated scopes ensure that the results would be no different if all conflicting activities (read/write and write/write activities) on all shared variables and partner links were conceptually reordered so that either all such activities within S1 are completed before any in S2 or vice versa" [BPEL2.0, pp.143].

2. If the value of the `exitOnStandardFault` attribute on a scope is set to `yes`, then the process must exit immediately, as if an `<exit>` activity has been reached, when any BPEL 2.0 standard fault other than `bpel:joinFailure` reaches the scope. If the value of this attribute is set to `no`, then the process can handle a BPEL 2.0 standard fault using a fault handler [BPEL2.0, p.135].

Listing 44: Scope Schema

A.) [BPEL1.1, p.30]

```
<scope variableAccessSerializable="yes|no" standard-attributes>
```

B.) [BPEL2.0, p.132]

```
<scope isolated="yes|no"? exitOnStandardFault="yes|no"? standard-attributes>
```

Listing 45: Scope Example

A.) [BPEL1.1, p.70]

```
<scope variableAccessSerializable="yes">
  <faultHandlers>?
    ...
  </faultHandlers>
  <flow>
    <invoke
      ...
    </flow>
  </scope>
```

B.) BPEL2.0

```
<scope name="purchase" isolated="yes" exitOnStandardFault="yes">
  <targets>
    <target linkName="linkA" />
  </targets>
  <sources>
    <source linkName="linkB" />
  </sources>
  <invoke name="purchase" partnerLink="Seller" portType="SP:Purchasing"
    operation="Purchase" inputVariable="sendPO"
    outputVariable="getResponse" />
</scope>
```

2.8.1. Scope Initialization

In the previous version, BPEL 1.1, the Scope Initialization was not specified and thus implementation specific. In BPEL 2.0 the sequence of actions that are to be taken by the implementation have exactly been specified [BPEL2.0, pp.116].

2.8.2. Message Exchange Handling

When the primary activity and the event handlers of a `<scope>` complete, then all Web Service interactions dependent on partner links or message exchanges declared inside of the `<scope>` need to be completed. An orphaned inbound message activity (IMA) occurs when an IMA using a partner link or message exchange, declared in the completing `<scope>` or its descendants, remains open. In this case, the new standard fault `bpel:missingReply` must be thrown [BPEL2.0, pp.117].

2.8.3. Process State Usage in Compensation Handlers

In the BPEL 2.0 specification the Compensation Model has been extended:

“A compensation handler always uses the current state of the process at the time the compensation handler is executed. This state comes from its associated scope and all enclosing scopes, and includes the state of variables, partner links and correlation sets. Compensation handlers are able to both: read and write the values of all such data. Other parts of the process will see the changes made to shared data by compensation handlers, and conversely, compensation handlers will see changes made to shared data by other parts of the process” [BPEL2.0, pp.120].

Furthermore, “the process state consists of the current state of all scopes that have been started. This includes scopes that have completed successfully but for which the associated compensation handler has not been invoked. For successfully completed (but uncompensated) scopes, their state is kept at the time of completion. Such scopes are not running, yet they are still reachable. This is because their compensation handlers are still available, and therefore the execution of such scopes may continue during the execution of their compensation handlers, which can be thought of as an optional continuation of the behaviour of the associated scope. A scope may have been executed several times (e.g. in a `<while>` or in a `<forEach>`), so the state of the process includes the state of all successfully completed (and uncompensated) iteration instances of the scope. We refer to the preserved state of a successfully completed uncompensated scope as a *scope snapshot*” [BPEL2.0, pp.120].

2.8.4. Invoking a Compensation Handler

Compensation handlers are both in BPEL 1.1 and BPEL 2.0 defined by using the `<compensationHandler>` statement, yet the invocation mechanism differs:

In BPEL 2.0 a compensation handler can be invoked by using `<compensateScope>` or `<compensate>` (together referred to as the “compensation activities”) while in BPEL 1.1 only `<compensate>` is used, naming the targeted scope with the XML attribute `scope`.

“User-defined Fault / Compensation / Termination handlers may use `<compensateScope>` activities to compensate specific immediately enclosed scopes and / or `<compensate>` to compensate all immediately enclosed scopes in default order” [BPEL2.0, pp.123]. BPEL 2.0 furthermore specifies a new requirement: “Any repeated attempt to compensate immediately enclosed scopes is treated as executing an `<empty>` activity” [BPEL2.0, p.124].

Listing 46: Compensation Invocation Schema

A.) [BPEL1.1, p.72]:

```
<compensate scope="ncname"? standard-attributes>
  standard-elements
</compensate>
```

B.) [BPEL2.0, p.122]:

```
<compensateScope target="NCName" standard-attributes>
  standard-elements
</compensateScope>

<compensate standard-attributes>
  standard-elements
</compensate>
```

Listing 47: Compensation Invocation Example

A.)

[BPEL1.1, p.75]:

```
<compensate scope="RecordPayment"/>
```

[BPEL1.1, p.75]:

```
<compensate/>
```

B.)

[BPEL2.0, p.123]:

```
<compensateScope target="RecordPayment"/>
```

[BPEL2.0, p.121]:

```
<faultHandlers>
  <catch faultName="prefix:someFault">
    <compensate />
  </catch>
</faultHandlers>
```

2.8.5. Fault Handlers

1. A BPEL 2.0 process is allowed to rethrow the original fault caught by the nearest enclosing fault handler with the new `<rethrow>` activity. A `<rethrow>` activity is allowed to be used within any fault handler and only within a fault handler. Regardless of how a fault is caught and whether a fault handler modifies the fault data, a `<rethrow>` activity always throws the original fault data and preserves its type [BPEL2.0, pp.131].

2. The default fault handling mechanism was informally described in BPEL 1.1 [BPEL1.1, p.78], but BPEL 2.0 provides more precise details: “Whenever a `<catchAll>` fault handler (for any fault), `<compensationHandler>`, or `<terminationHandler>` is missing for any given `<scope>`, they must be implicitly created by the implementation” [BPEL2.0, pp.132], for the Default Handlers’ definitions see Listing 48:, Listing 50: and Listing 52:.

The way of handling such process models (e.g. adding the default fault handlers to the process model at deployment time) is not specified and thus implementation specific.

3. The Execution Order of Compensation activities has clearly been specified in BPEL 2.0 regarding also parallel activities [BPEL2.0, pp.132].

4. The new Termination handlers provide the ability for scopes to control the semantics of forced termination to some degree. The syntax is as follows:

“The forced termination of a scope begins by terminating its primary activity and all running event handler instances. Following this, the custom `<terminationHandler>` for the scope, if present, is run” [BPEL2.0, pp.136]. In the former BPEL 1.1 specifications this was handled by the fault handler with the standard `bpws:forcedTermination` fault [BPEL2.0, pp.79].

5. Enriched Fault Handling Model: Several new faults are introduced in BPEL 2.0, for a complete list see [BPEL2.0, pp.192] in comparison to [BPEL1.1, pp.112].

Listing 48: Default Fault Handler

[BPEL2.0, p.132]:

```
<catchAll>
  <sequence>
    <compensate />
    <rethrow />
  </sequence>
</catchAll>
```

Listing 49: Fault Handler Example

BPEL2.0:

```
<faultHandlers>
  <catch faultName="KnownIssue">
    <compensate />
  </catch>
  <catchAll>
    <rethrow />
  </catchAll>
</faultHandlers>
```

Listing 50: Default Compensation Handler

[BPEL2.0, p.132]:

```
<compensationHandler>
  <compensate />
</compensationHandler>
```

Listing 51: Compensation Handler Example

[BPEL2.0, p.145]:

```
<compensationHandler>
  <sequence name="undoQ Seq">
    ...
  </sequence>
</compensationHandler>
```

Listing 52: Default Termination Handler

[BPEL2.0, p.132]:

```
<terminationHandler>
  <compensate />
</terminationHandler>
```

Listing 53: Termination Handler Example

[BPEL2.0, p.136]:

```
<terminationHandler>
  activity
</terminationHandler>
```

2.8.6. Event Handlers

Event handling syntax and functionality has been modified and enriched in BPEL 2.0 [BPEL2.0, pp.137]:

1. The `onMessage` element has been renamed to `onEvent`.
2. Activities nested in the `onEvent` or `onAlarm` element have to be wrapped in a scope.
3. The `onAlarm` feature has additionally an optional `repeatEvery` expression: While the parent scope is active, the `<repeatEvery>` alarm event is created repeatedly each time the duration expires.

Listing 54: Event Handler Schema

A.) [BPEL1.1, pp.80]:

```
<eventHandlers>?
  <!-- there must be at least one onMessage or onAlarm handler -->
  <onMessage partnerLink="ncname" portType="qname" operation="ncname"
    variable="ncname"?>*
    <correlations>?
      <correlation set="ncname" initiate="yes|no">+
    </correlations>
    activity
  </onMessage>
  <onAlarm for="duration-expr"? until="deadline-expr"?>*
    activity
  </onAlarm>
</eventHandlers>
```

B.) [BPEL2.0, pp.137]:

```
<eventHandlers>?
  <onEvent partnerLink="NCName"
    portType="QName"?
    operation="NCName"
    ( messageType="QName" | element="QName" )?
    variable="BPELVariableName"?
    messageExchange="NCName"?>*
    <correlations>?
      <correlation set="NCName" initiate="yes|join|no"? />+
    </correlations>
    <fromParts>?
      <fromPart part="NCName" toVariable="BPELVariableName" />+
    </fromParts>
    <scope ...>...</scope>
  </onEvent>
  <onAlarm>*
    (
      <for expressionLanguage="anyURI"?>duration-expr</for>
      |
      <until expressionLanguage="anyURI"?>deadline-expr</until>
    )?
    <repeatEvery expressionLanguage="anyURI"?>?
      duration-expr
    </repeatEvery>
    <scope ...>...</scope>
  </onAlarm>
</eventHandlers>
```

Listing 55: Event Handler Example

A.) [BPEL1.1, pp.81]:

```
<eventHandlers>
  <onMessage partnerLink="buyer" portType="car" operation="cancel"
    variable="cancelDetails">
    <terminate/>
  </onMessage>
  ...
</eventHandlers>
```

B.) [BPEL2.0, pp.139]:

```
<eventHandlers>
  <onEvent partnerLink="travelAgency" portType="ns:agent"
    operation="travelUpdate" messageType="ns:travelStatsUpdate"
    variable="travelUpdate">
    <correlations>
      <correlation set="travelCode" initialize="no" />
      <correlation set="updateCode" initialize="yes" />
    </correlations>
    <scope name="S2">
      ...
    </scope>
  </onEvent>
</eventHandlers>
```

2.9. Extensions

1. BPEL 2.0 introduces a new extension directive to specify, which extension must be understood. If a BPEL 2.0 processor does not support one or more of the extensions with `mustUnderstand = yes`, then the process definition must be rejected.

Optional extensions are extensions which the BPEL 2.0 process may ignore. There is no requirement to declare any optional extensions. Optional extension can be declared using the extensions element with `mustUnderstand = no`.

Listing 56: Process Extension Schema

[BPEL2.0, p.164]

```
<process ...>
  ...
  <extensions?
    <extension namespace="anyURI" mustUnderstand="yes|no" />+
  </extensions>
  ...
</process>
```

Listing 57: Process Extension Example

[BPEL2.0, p.164]

```
<process ...>
  ...
  <extensions>
    <extension namespace="http://example.com/bpel/some/extension"
      mustUnderstand="no" />
  </extensions>
  ...
</process>
```

2. The optional `<documentation>` construct is applicable to any BPEL 2.0 extensible construct. Typically, the contents of `<documentation>` are for human-readable annotations. Example types for content are: plain text, HTML and XHTML [BPEL2.0, p.32].

2.10. Abstract Processes

The key distinction between public message exchange protocols and executable internal processes is that internal processes handle data in rich private ways that need not be described in public protocols [BPEL1.1, p.9].

2.10.1. Common Base

The common base is the “syntactic form” to which all BPEL 2.0 Abstract Processes must conform. The syntactic characteristics of the common base are [BPEL2.0, p.147]:

1. The new `abstractProcessProfile` attribute must exist. Its value refers to an existing profile definition.
2. All the constructs of Executable Processes are permitted. Thus, there is no fundamental expressive power distinction between Abstract and Executable Processes.
3. Certain syntactic constructs in BPEL 2.0 Executable Processes may be hidden, explicitly through the inclusion of opaque language extensions, and implicitly through omission.
4. An Abstract Process may omit the `createInstance` activity (`<receive>` or `<pick>`) that is mandatory for Executable BPEL 2.0 Processes.

2.10.2. Opaque Language Extensions

There are four opaque placeholders: expressions, activities, attributes and from-specs. A usage profile may restrict the kinds of opaque tokens allowed at its discretion. However, a usage profile must not expand allowable opacity above what is allowed by the common base [BPEL2.0, p.148].

2.10.3. Omission

Omission may be used as a shortcut to opacity, from hereon referred to as omission shortcut. The omission shortcut is exactly equivalent to representing the omitted artefact with an opaque value at the omitted location. Tokens must only be omitted where the location can be detected deterministically. To enforce this requirement, omissible tokens are restricted to all attributes, activities, expressions and from-specs which are both syntactically required by the Executable BPEL 2.0 XML Schema, and have no default value [BPEL2.0, p.151].

2.10.4. Abstract Process Profiles

A usage profile defines the necessary syntactic constraints and the semantics based on Executable BPEL 2.0 Processes for a particular use case for Abstract Processes. Every Abstract Process must identify the usage profile that defines its meaning. A profile is identified using a (mandatory) URI, which is referenced as value of the `abstractProcessProfile` attribute by all Abstract Processes belonging to this profile. This approach is extensible, new profiles can be defined as different areas are identified [BPEL2.0, p.147]. Two profiles are provided in the BPEL 2.0 specification, Observable Behaviour and Process Template.

2.10.5. Observable Behavior

The first usage profile is concerned with hiding internal processing of a business partner's process while capturing all the information required to describe, how the process interacts with its partners [BPEL2.0, pp.155]. The set of usage restrictions is derived from the original Abstract Process definition [BPEL1.1, pp.88]. The URI identifying this Abstract Process Profile is:

Listing 58: Observable Behavior Reference

<http://docs.oasis-open.org/wsbpel/2.0/process/abstract/ap11/2006/08>

2.10.6. Process Template

BPEL 2.0 defines an Abstract Process profile called Template Profile. This usage profile allows the definition of Abstract Processes which hide almost any arbitrary execution details and have explicit opaque extension points for adding behavior. These Abstract Processes allow process developers to complete execution details at a later stage – for example, adding conditions and defining endpoints for an Executable Completion [BPEL2.0, p.159]. The URI identifying this Abstract Process Profile is:

Listing 59: Process Template Reference

<http://docs.oasis-open.org/wsbpel/2.0/process/abstract/simpletemplate/2006/08>

3. Structure of the Graphical Editor

3.1. Eclipse, EMF and GEF

The Graphical BPEL Editor is built on top of several frameworks: It is implemented as an Eclipse plug-in and thus integrated into the Eclipse Workbench. It is also using the Eclipse Modeling Framework (EMF) for the generation and representation of the BPEL data model in code. For the graphical editing functionality it is utilizing the Graphical Editing Framework (GEF) which provides a rich platform for the graphical creation and modification of data models. Both, EMF and GEF are also implemented as Eclipse plug-ins.

Basic Elements of GEF applications are the so-called EditParts. Those elements define the mapping between the data model and their graphical representation. For each element in the data model that shall be graphically editable such an element has to be created. There are three kinds of EditParts:

- GraphicalEditParts which are used for representing activities.
- ConnectionEditParts which are used for representing links.
- TreeEditParts which are used for representing other constructs like `PartnerLinks` and `Variables`.

The following figure illustrates the simplified connection of the data model with the graphical viewer using EditParts [Kap106, p.26].

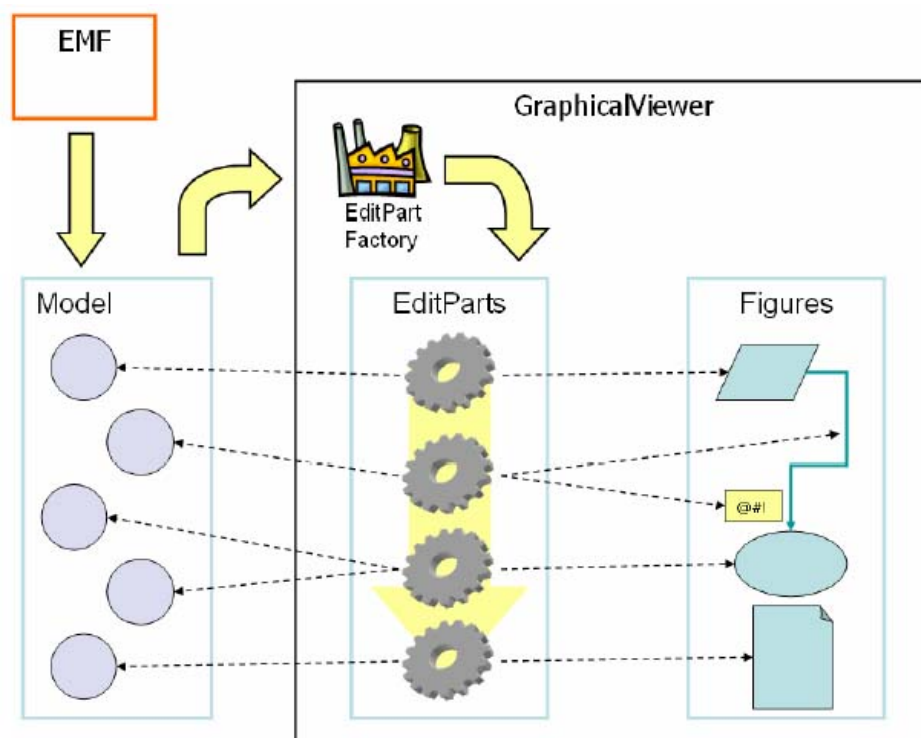


Figure 1: EMF and GEF

3.2. General Structure of the Graphical Editor

The graphical editor consists of a data model (generated with EMF), mapping elements (EditParts) and a graphical viewer (GEF). To be more precise the mapping elements are only part of a bigger structure, namely the controller. The figure below illustrates the general structure of the graphical editor.

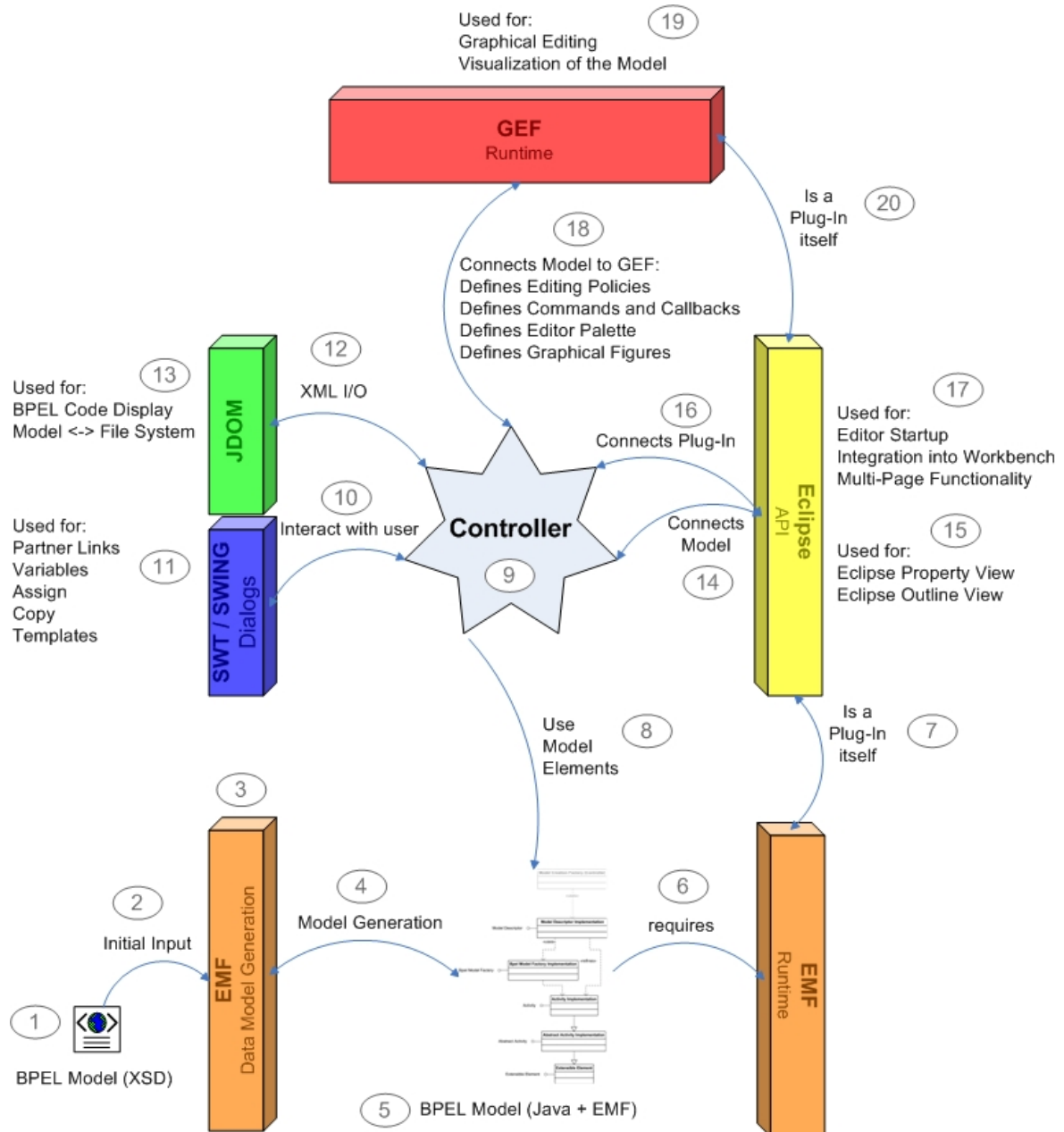


Figure 2: General Structure of the Graphical Editor

Walkthrough

1. For both, the BPEL 1.1 and BPEL 2.0 model XML Schema Definitions (XSD) are available.
2. The BPEL model (XSD) is used as initial input for the generation of the BPEL model in code.
3. For the generation of the model in code, the Eclipse Modeling Framework (EMF) is used. The EMF builds up an internal representation of the model, called Ecore model.
4. The Ecore model is the basis for the generation of the model in code. To certain extend - only Java Interfaces - the generated model can also be used as input for the Ecore model. Thus multiple cycles of code generation are possible.
5. The generated BPEL model consists of Java classes extending EMF components. The basic structure of the data model is discussed in 3.4.
6. These components have to be available during runtime.
7. The EMF is implemented as Eclipse plug-in; hence the whole development work of the BPEL model in code is performed using the Eclipse development platform.
8. The generated BPEL model provides a factory for the instantiation of model elements; this interface is used by the controller.
9. The controller connects the elements of the data model to the other components and frameworks. The detailed structure of the controller is discussed in 3.3.
10. Some interactions with the user can neither be realized with the Graphical Editing Framework (GEF) nor with Eclipse functionality.
11. For this kind of interactions Java SWT / SWING Dialogs are used, for example to import a Partner Link into the process model.
12. The controller has to deal with the transformation of the internal representation of the process model into XML (BPEL code) and the other way round.
13. For the accessing, manipulating and outputting XML data from Java code, e.g. outputting BPEL code or importing a `PartnerLink` WSDL, the JDOM framework is used. This is similar to the Document Object Model (DOM).
14. The controller connects the elements of the process model to the Eclipse API.
15. Firstly for the integration with the Eclipse Property View for providing access to activity attributes for example. Secondly for the integration with the Eclipse Outline View for providing access to other constructs like `PartnerLink` and `Variable`.
16. The controller is implemented as Eclipse plug-in and has to connect itself to the Eclipse development platform.
17. These connections provide the automatic startup of the graphical editor when a file with the extension `.bpel` is opened in Eclipse and also the integration into the workbench which is required for all Eclipse plug-ins. The graphical editor makes usage of further Eclipse features like the multi-page functionality for switching between the graphical representation of the BPEL process and its BPEL code.
18. As already described in 3.1 the controller connects the elements of the data model to the Graphical Editing Framework using Edit Parts. These however are not the only connection, also editing policies, commands and according call backs, the content of the editor palette and instructions for graphical display of elements have to be defined.
19. The Graphical Editing Framework provides the graphical platform for the visual creation and modification of BPEL process models.
20. This framework is also implemented as an Eclipse plug-in.

3.3. Structure of the Controller

The controller consists of various components that connect the data model to other the frameworks. In this thesis the components that connect an Activity are most important, so the following description omits other aspects like for example the integration of the editor as Eclipse plug-in. For a complete description of the controller please refer to [Kapl06].

- The Activity Edit Part defines the mapping of the Activity from the data model to the Graphical Editing Framework.
- The Edit Part Factory instantiates the Edit Part of the Activity
- The Editor Palette is required to make the Activity accessible within the Graphical Editor. It provides a palette of grouped activities with a label and an icon.
- The command classes define the call backs for events (e.g. Add Activity) in the Graphical Editing Framework.
- The commands are using Edit Policies that declare how the command shall be handled for a specific activity. For example the ActivityXYLayoutEditPolicy declares, how movements of activities onto other activities are reflected in the data model, e.g. by nesting the Activity into a Scope.
- The Bpel Output Generator transforms the Activity into Bpel code using the Java Document Object Model Package for building up the XML tree structure.
- The Property View Provider makes the properties of the Activity accessible in the Eclipse Property View.
- The Model Creation Factory is the connection between the controller and the data model. It is delegating the request for the creation of elements to the data model.

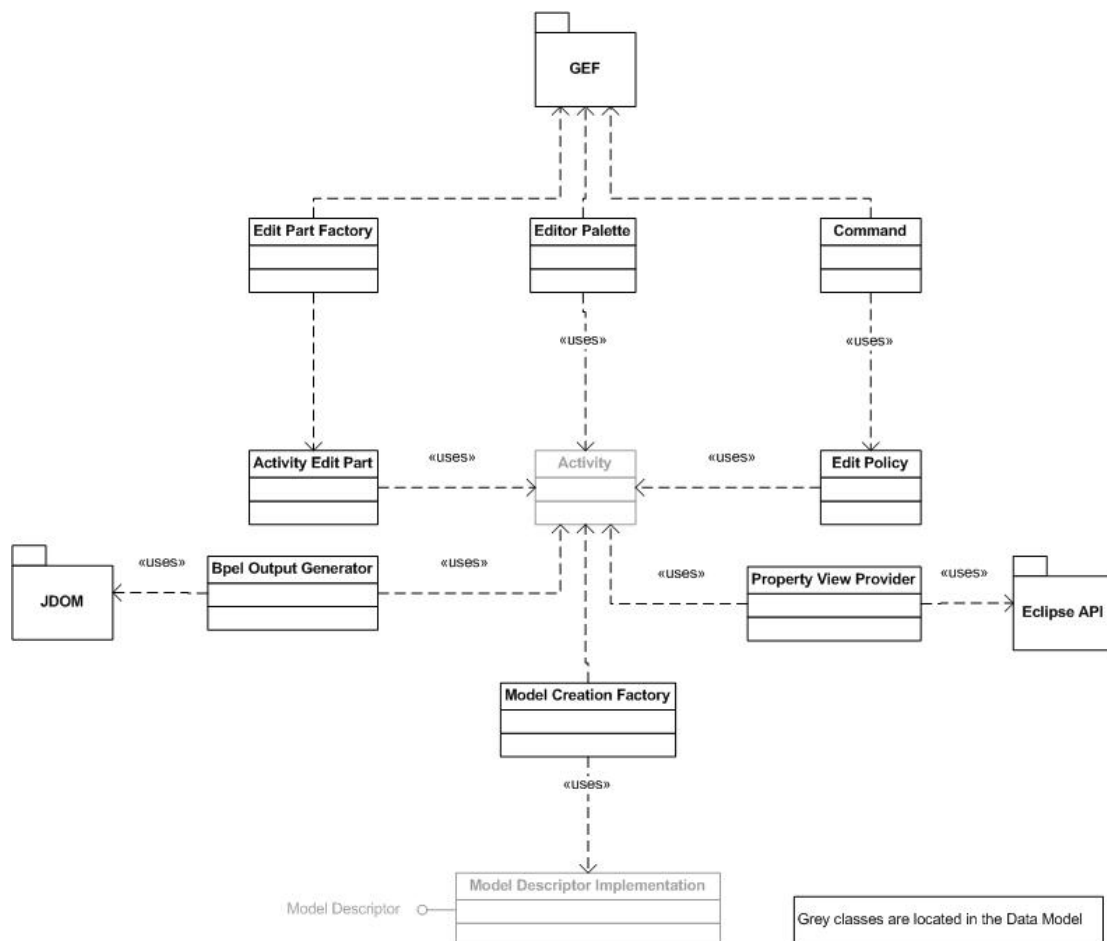


Figure 3: Integration of an Activity into the controller

3.4. Basic Structure of the Data Model

The data model is the representation of the Bpel Model in code. The data model is generated using the Eclipse Modeling Framework (EMF) and is based on Java. It makes additional usage of generic EMF classes like EAttribute and EClass. These EMF classes are used for internal representation of the data model during runtime. The EMF code generation produces the following class structure:

“For each EClass in the EPackage, an interface is generated in the base package, and a Java class that implements it is generated in the impl package. If the EClass inherits from another EClass, then the generated interface and implementation extend the interface and implementation generated for the supertype” [MDGW04, p.46].

The Model Descriptor “contains accessors for the meta objects to represent each class, each feature of each class, each enum and each data type” [EMF JavaDoc]. For example the Property View Provider from the controller accesses the Model Descriptor to find out, which attributes a certain class has and what their display name is.

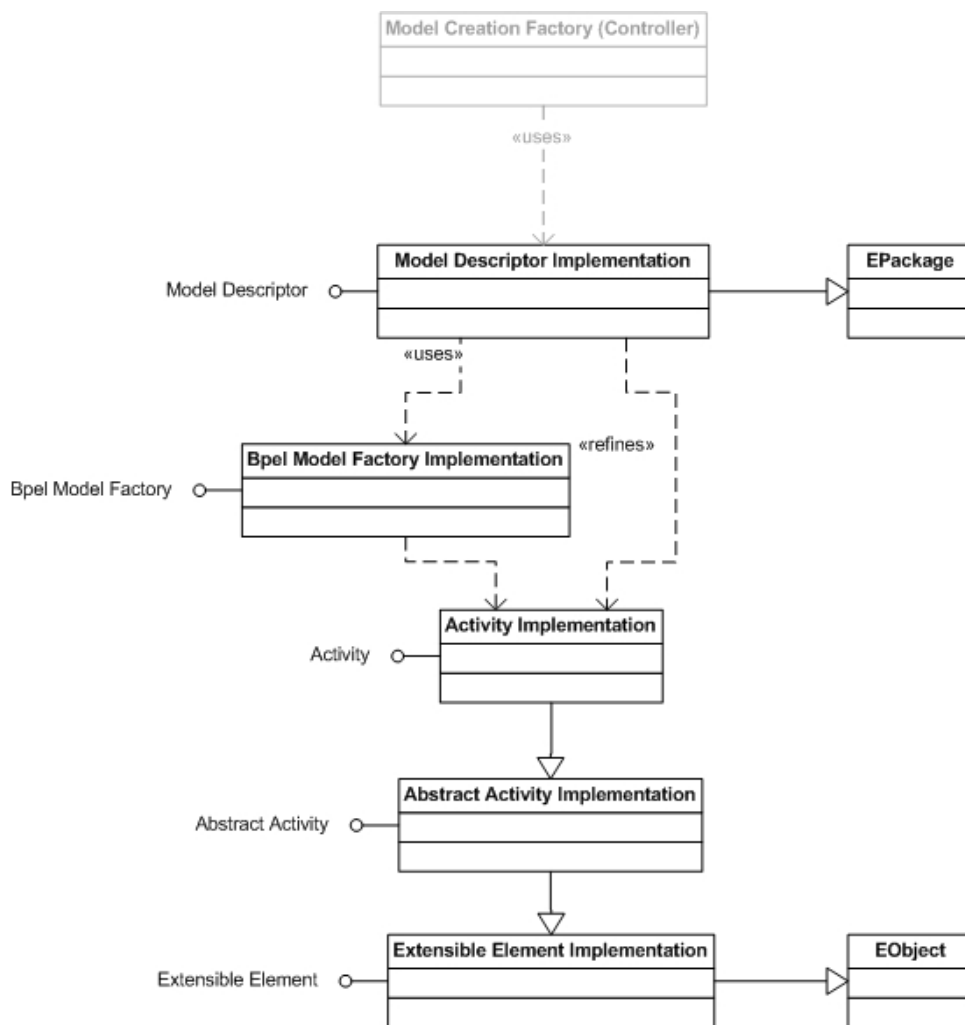


Figure 4: Connections of an activity in the data model

4. Analysis of the Extensibility of the Existing Tool

4.1. BPEL 1.1 Data Model Generation

“The Eclipse Modeling Framework (EMF) is designed to ease the design and implementation of a structured model. The Java framework provides a code generation facility in order to keep the focus on the model itself and not on its implementation details. The key concepts underlying the framework are: meta-data, code generation, and default serialization” [MDGW04, p.3].

The Eclipse Modeling Framework has been used during the implementation of the prior version of this editor, which this thesis is based on. For the code generation of the editor the BPEL 1.1 XML Schema Definitions (XSD) have been used as initial input, see [BPEL1.1XSD].

After the initial code generation the generated interface classes have been modified to support Extensions like `Evaluate`, `Find & Bind` and the support for `BPEL Templates` [Kapl06, pp. 7]. These interface classes have been used for a second cycle of code generation as described in [Kapl06, pp. 34]. After the second cycle the generated skeletons have partially been implemented.

4.2. BPEL 2.0 Data Model Generation

The OASIS consortium has provided the XML Schema Definition for BPEL 2.0 executable processes (see [BPEL2.0XSD]). This enables the generation of a BPEL 2.0 data model using the Eclipse Modeling Framework.

After the initial code generation the generated interface classes can be modified to support Extensions like `Evaluate`, `Find & Bind` and the support for `BPEL Templates` [Kapl06, pp. 7] like in the prior version of the editor. These interface classes can also be used for a second cycle of code generation as described in [Kapl06, pp. 34].

The generated BPEL 2.0 model files provide the skeletons for the model but they are missing the implementation of language elements existing in BPEL1.1 and which have already been provided by the former version of the editor. Nevertheless this implementation could be accomplished with tolerable effort, thus it would be theoretically possible to exchange the underlying data model of the editor completely.

The various adaptations of the generated BPEL 1.1 data model in [Kapl06] and the multitude of syntactical changes from BPEL 1.1 to BPEL 2.0 make the merge of the available BPEL 1.1 data model and the BPEL 2.0 data model hardly possible.

4.3. BPEL 1.1 Data Model Integration

Based on the knowledge about building graphical editors using EMF and GEF [MDGW04] and plug-in development, it turns out that the real development work lies in connecting the generated data model with the Graphical Editing Framework. In other words, the development of the controller and its embedding into an eclipse plug-in are the main tasks.

The controller of the former version of the editor is not generic and hard-wired with the underlying data model. It is also not embedded into a plug-in instead it is the plug-in itself. The actual development work of the editor was the architecture and implementation of this controller.

4.4. BPEL 2.0 Data Model Integration

The investigation of the BPEL 1.1 data model integration identifies two options on how to integrate the BPEL 2.0 model into the editor:

The first option is to redevelop the editor using the BPEL 2.0 data model. Some code can be reused but most of the development work has to be performed again. This option would also allow using a more generic approach during the development of the controller. In this case an MDA approach would bring benefits, however, the requirements in the project have been to extend a particular existing modeling tool.

The other option is to integrate the BPEL 2.0 data model into the controller piece by piece. The first section of this thesis provides the background information that is needed to follow this approach. This option is challenging as the BPEL 1.1 data model with its extensions and its integration with the controller on the basis of an eclipse plug-in is quite complex. This option has been chosen in this thesis and is described and demonstrated in detail (see the following section).

4.5. BPMN Integration

The Business Process Management Initiative of the Object Management Group (OMG) has developed a standard Business Process Modeling Notation (BPMN):

“The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation“, see [BPMN1.0, p.1].

A hybrid editor that supports BPMN and BPEL would be a great improvement for the collaboration of both, business analysts and technical developers. Therefore the possibilities for integrating BPMN into the existing tool are investigated in the following.

4.5.1. Mapping BPMN to BPEL

The specification BPMN 1.0 describes extensively (see [BPMN1.0, pp.137]) how parts of a Business Process Diagram can be mapped to BPEL 1.1 processes. One of the essential disparities is mentioned in the preamble:

“A Business Process Diagram can be made up of a set of (semi-) independent components, which are shown as separate *Pools*. Thus, there is not a specific mapping to the diagram itself. Rather, there are separate mappings to each of the Pools that are in the diagram. That is, each Pool in the diagram, if it is a “white box” that contains process elements, will map to an individual BPEL4WS [BPEL1.1] *process*. However, in the course of mapping the contents of the Process, there may be one or more derived *processes* necessary to handle complex behavior, such as looping. The attributes of “black box” Pools will also be used in determining specific BPEL4WS [BPEL1.1] elements, such as *partnerLink*” [BPMN1.0, p.137].

As a conclusion it can be stated that a procedure for mapping a BPMN 1.0 Business Process Diagram to BPEL 1.1 *processes* and *elements* has already been specified from the OMG, see [BPMN1.0, p.137], and demonstrated, see [Whit05]. The mapping has not yet been specified for BPEL 2.0 but as the specification for BPMN 2.0 is ongoing, it can be expected, that support for BPEL 2.0 mapping will be considered, see [BPMN2.0, p.24].

Although this procedure has been specified and demonstrated in various examples, it is also described as “intrinsically complex to map the diagrams to BPEL processes because of the structural disparity between BPMN and BPEL. BPEL is a block structured language overall, even though a flow with links in BPEL can be more flexible. In contrast, BPMN is a constrained, but relative free form graph. Structurally, BPMN can be a super-set of BPEL. There are no fundamental difficulties in mapping a BPEL process to an isomorphic BPMN diagram. In other words, any BPEL process can be visualized as a BPMN diagram without rearranging the flows. But it is not always possible to map a BPMN diagram directly to an isomorphic BPEL process”, see [Gao06, p.1].

4.5.2. Mapping BPEL to BPMN

The procedure for mapping BPEL 2.0 `processes` and `elements` to a BPMN 1.0 Business Process Diagram has not been provided by the organization that has developed the BPEL 2.0 standard, OASIS. However there exists a proprietary solution [eClarus] for this problem, which is partially described in a related white paper, see [Gao06]. As the procedure, which is used in this solution, has not been released to the public and probably will not be in near future it would be necessary to elaborate on this before integration into any tool can take place.

4.5.3. Integration Approaches

The prior investigation identifies three main approaches for integrating BPMN with BPEL in general, with respect to the enormous benefits of graphical editing technologies:

- Developing a graphical editor for BPMN, that provides the transformation into BPEL `processes` and `elements` and supports preferably their visualization.
- Developing a graphical editor for BPEL `processes` and `elements`, that provides transformation into BPMN and its visualization.
- Developing a graphical editor for BPMN and BPEL offering Round-Trip-Engineering with respect to certain constraints and limitations.

The first approach has already been implemented by various proprietary solutions, yet the open source projects are just at the beginning. The most promising project in this area is the Eclipse SOA Tools Platform (STP) that will contain a BPMN editor [STP07]. An interface for the generation of BPEL code is planned for this project but not assigned to contributors yet. This approach can not be implemented in the existing graphical BPEL editor, as only BPEL model instances and its data structures can be processed and stored. All additional (only BPMN related) information could not be taken into account.

The second approach is developing a graphical editor for BPEL `processes` and `elements` that provides transformation into BPMN and its visualization is the most feasible for the existing editor. For realization of this approach first the mapping rules from BPEL 2.0 into BPMN 1.0 would have to be defined. Afterwards a read-only BPMN visualization-tab could be integrated into the editor, where BPMN objects according to the BPEL model instance can be displayed. Daniel Lee [Lee03] describes how to use the java library *Draw2d* to display user-defined graphical objects in a GEF environment, this also applies here. The BPMN visualization-tab is described as read-only, as the existing editor is only able to process and store BPEL model instances due to the underlying BPEL data model.

The third approach is developing a graphical editor for BPMN and BPEL, offering Round-Trip-Engineering with respect to certain constraints and limitations is undisputably the most challenging one. It has already been implemented by [eClarus] in a proprietary solution for Round-Trip-Engineering BPMN and BPEL (1.1). For realization of this approach an editor has to be developed, that is capable of modeling and processing both models, BPEL and BPMN. Additionally, BPMN related constraints and limitations would have to be defined in order to allow only constructs that can be mapped to BPEL. Finally the mapping procedures from BPEL 2.0 to BPMN 1.0 and the other way round have to be defined and implemented. The existing graphical editor is not appropriate for such an extension.

5. BPEL 2.0 model integration

5.1. Integration overview

In order to preserve the existing controller and the already integrated extensions and conducted implementation work, the data model files are not exchanged completely. For some of the integration tasks, the generation of the data model is a prerequisite and has to be performed beforehand. These are the classes of integration:

New attributes

For additional attributes like `suppressJoinFailure` on any activity (see 2.2.1) integration can be achieved by adapting the graphical editor (editor palette and property provider), the BPEL output generator and the data model accordingly.

New construct

For completely new constructs like `validate` (see 2.4.1) it is feasible to integrate the generated BPEL 2.0 model files into the existing data model of the editor and adapt the various parts of the controller accordingly.

New elements

New elements with single occurrence like `documentation` (see 2.9) can be integrated like new attributes, while elements with more than one occurrence like `fromParts` (see 2.4.1) have to be integrated like new constructs.

Removed constructs

For the removal of constructs like `Partners` (see 2.3.3) integration can be achieved by adapting the existing data model provider and the editor palette accordingly. Although the construct `Partners` has been removed from the BPEL 2.0 specification it will not be removed from the editor as required for further development of the tool. Since no other constructs or activities were removed from the former BPEL 1.1 specification during the standardization by OASIS, the removal of elements is not described in detail.

Removed attributes and construct elements

For the removal of attributes like the `process` attribute `abstractProcess` (see 2.2.1) integration can be achieved by adapting the BPEL output generator and the existing data model accordingly. The procedure of removing elements is the same.

Attributes and nested XML elements

For the transformation of attributes into nested XML elements like `until` in the activity `wait` (see 2.6.2) and the other way round like `portType` in `partnerLinks` (see 2.3) integration can be achieved by adapting the BPEL output generator of the editor accordingly.

Constructs naming

For the change of the naming like from `Terminate` to `Exit` (see 2.6.4) integration can be achieved by adapting the editor palette and the BPEL output module of the editor accordingly.

Distinguishing between the Controller and the Data Model

The adaptations that are made affect in most of the described integration classes both, the controller and the data model. They can easily be distinguished by the package name:

Package name of the controller: `org.xmlsoap.schemas.ws.bpeleditor`

Package name of the data model: `org.xmlsoap.schemas.ws.bpelmodel`

5.2. Integration of new attributes

As an example of the integration of a new attribute `suppressJoinFailure` (see 2.2.1) on the activity `invoke` is described in detail. Actually, the attribute was already included in the specification of BPEL 1.1, as a (optional) `standard-attribute` for all activities and is still included in the specification of BPEL 2.0, but it was not implemented for all activities in the former version of the editor. The parts of the integration are described top-down, beginning on top at the BPEL output the editor produces going down to its actual implementation in the data model. As data type for the attribute the type `String` is preferred to `boolean` because the native Java `boolean` type does not support having no value. It is important to have the option that no value is set in order to be able to distinguish whether the attribute has to be generated in the BPEL output or not.

5.2.1. BPEL output generator

The BPEL output generator is the unit that produces the BPEL code out of the BPEL process model. Each type of construct has a related function to build its subtree in the JDOM tree, in this example the function `buildInvokeSubtree`.

Listing 60: Adaptation of BPEL output generator for attribute integration

```
File: org.xmlsoap.schemas.ws.bpeleditor.xml.BpelBuilder.java  
Function: buildInvokeSubtree  
  
//optional: supressJoinFailure  
if ((invokeActivity.getSuppressJoinFailure() != null) &&  
    (invokeActivity.getSuppressJoinFailure() != "")){  
    invokeElement.setAttribute(new Attribute("suppressJoinFailure",  
        invokeActivity.getSuppressJoinFailure()));  
}
```

Explanation:

- `invokeActivity` is the instance of the invoke activity class `TInvokeActivityImpl`
- `invokeElement` is the element in the JDOM tree that is being built
- The attribute name in the generate output is "suppressJoinFailure"
- The attribute is only generated, if its value is defined (`!= null`) and if its value is not an empty string (`!= ""`)
- `getSuppressJoinFailure` is a getter function of the invoke activity class `TInvokeActivity` which is implemented by `TInvokeActivityImpl`. The function returns the value of the new attribute.

5.2.2. Property View Provider

For some attributes it is feasible to limit the allowed values the user can set. For the `suppressJoinFailure` attribute the values "yes" and "no" are defined by the specification. A third value, the empty string "", has to be added to allow the user to completely unset the attribute. This value limitation is implemented in the Property Source Descriptor class, which provides additional functionality for the Eclipse Property View.

Listing 61: Initialization of allowed attribute values array

```
File: org.xmlsoap.schemas.ws.bpeleditor.model.EObjectPropertySource.java  
Function: class EObjectPropertySource  
  
//Creating a new ArrayList for the allowed values  
ArrayList suppressJoinFailureValues = new ArrayList();
```


Listing 62: Initialization of allowed attribute values

```
File: org.xmlsoap.schemas.ws.bpeleditor.model.EObjectPropertySource.java  
Function: EObjectPropertySource  
  
//Add values to Attribute Array  
suppressJoinFailureValues.add("yes");  
suppressJoinFailureValues.add("no");  
suppressJoinFailureValues.add("");
```

Listing 63: Getter function for the Property Descriptor

```
File: org.xmlsoap.schemas.ws.bpeleditor.model.EObjectPropertySource.java  
Function: getPropertyDescriptors  
  
//Iteration over all attributes  
...  
else if (attr.getName() == "suppressJoinFailure"){  
    ComboBoxPropertyDescriptor desc = new ComboBoxPropertyDescriptor(  
        Integer.toString(attr.getFeatureID()), attr.getName(),  
        (String[])suppressJoinFailureValues.toArray(new  
            String[suppressJoinFailureValues.size()]);  
    desc.setCategory(groupName);  
    descriptors.add(desc);  
}
```

Listing 64: Getter function for the Property Values

```
File: org.xmlsoap.schemas.ws.bpeleditor.model.EObjectPropertySource.java  
Function: getPropertyValue  
  
else if (feature.getName().equals("suppressJoinFailure")){  
    //get the value of the attribute instance  
    String value = (String)object.eGet(feature);  
    //return position in array for Property View  
    result = new Integer(suppressJoinFailureValues.indexOf(value));  
}
```

Listing 65: Setter function for the Property Values

```
File: org.xmlsoap.schemas.ws.bpeleditor.model.EObjectPropertySource.java  
Function: setPropertyValue  
  
else if (feature.getName() == "suppressJoinFailure"){  
    //Sets the value on the attribute instance (as String)  
    object.eSet(feature, suppressJoinFailureValues.get(  
        ((Integer)value).intValue());  
}
```

5.2.3. Abstract Activity Interface

At this point the adaptations of the controller are complete and those of the data model begin:

In this case - adding a standard-attribute - the general class `TActivity` (interface definition) and `TActivityImpl` (interface implementation) can be adapted, as its functions are getting inherited by all activities. In any other case all of the following adaptations have to be conducted directly in the related classes, e.g. `TInvoke` and `TInvokeImpl`.

Listing 66: Abstract Activity Definition for getter and setter function of the attribute

```
File: org.xmlsoap.schemas.ws.bpelmodel.TActivity.java  
Function: getSuppressJoinFailure; setSuppressJoinFailure
```

```
//Defintion of the suppressJoinFailure attribute getter and setter
String getSuppressJoinFailure();
void setSuppressJoinFailure(String value);
```

Explanation:

The interface description has to include the getter and setter functions of the attribute.

5.2.4. Abstract Activity Implementation

In order to stay consistent with the generated code the integration is performed in the same manner as how the code generator (EMF) would act: The abstract activity implementation class `TActivityImpl` implements the getter and setter function for the common activity attribute, and defines a default setting and the variable itself.

Listing 67: Definition of default values for the attribute in the activity implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TActivityImpl.java
Function: class TActivityImpl

//Setting the default value of the attribute supressJoinFailure
protected static final String SUPPRESS_JOIN_FAILURE_EDEFAULT = null;
protected String suppressJoinFailure = SUPPRESS_JOIN_FAILURE_EDEFAULT;
```

Listing 68: Implementation of the getter and setter function of the attribute

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TActivityImpl.java
Function: getSuppressJoinFailure; setSuppressJoinFailure

//Attribute supressJoinFailure getter and setter implementation
public void setSuppressJoinFailure(String newSupressJoinFailure) {
    suppressJoinFailure = newSupressJoinFailure;
}

public String getSuppressJoinFailure() {
    return suppressJoinFailure;
}
```

5.2.5. Activity Implementation

The activity implementation contains the functionality for accessing the attribute. Four kinds of functions are provided, `eGet`, `eSet`, `elsSet` und `eUnset`. Those are functions for dynamic access to activity features and references. Parameters for these functions are the feature IDs from the Model Descriptor. These functions overwrite those of the abstract activity implementation.

Listing 69: Implementation of the interface for the controller: `eGet`

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TInvokeImpl.java
Function: eGet

switch (eDerivedStructuralFeatureID(eFeature)) {
    .
    .
    .
    //supressJoinFailure Attribute dynamic get
    case BpelmodelPackage.TACTIVITY__SUPPRESS_JOIN_FAILURE:
        return getSuppressJoinFailure();
}
```

Listing 70: Implementation of the interface for the controller: eSet

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TInvokeImpl.java  
Function: eSet  
  
switch (eDerivedStructuralFeatureID(eFeature)) {  
    .  
    .  
    //supressJoinFailure attribute dynamic set  
    case BpelmodelPackage.TACTIVITY_SUPPRESS_JOIN_FAILURE:  
        setSuppressJoinFailure((String) newValue);  
        return;  
}
```

Listing 71: Implementation of the interface for the controller: eUnset

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TInvokeImpl.java  
Function: eUnset  
  
switch (eDerivedStructuralFeatureID(eFeature)) {  
    .  
    .  
    //suppressJoinFailure dynamic unset  
    case BpelmodelPackage.TACTIVITY_SUPPRESS_JOIN_FAILURE:  
        setSuppressJoinFailure(SUPPRESS_JOIN_FAILURE_EDEFAULT);  
        return;  
}
```

Listing 72: Implementation of the interface for the controller: eIsSet

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TInvokeImpl.java  
Function: eIsSet  
  
switch (eDerivedStructuralFeatureID(eFeature)) {  
    .  
    .  
    //suppressJoinFailure dynamic eIsSet  
    case BpelmodelPackage.TACTIVITY_SUPPRESS_JOIN_FAILURE:  
        return suppressJoinFailure != null;  
}
```

5.2.6. Model Descriptor

The Model Descriptor “contains accessors for the meta objects to represent each class, each feature of each class, each enum and each data type” [EMF JavaDoc].

Listing 73: Definition of a feature number

```
File: org.xmlsoap.schemas.ws.bpelmodel.BpelmodelPackage.java  
Function: class BpelmodelPackage  
  
//suppressJoinFailure EAttribute definition  
EAttribute getActivity SuppressJoinFailure();  
  
//New Feature number  
int TACTIVITY_SUPPRESS_JOIN_FAILURE = TEXTENSIBLE_ELEMENTS_FEATURE_COUNT + 14;  
  
//Feature number count incremented, was: 14  
int TACTIVITY_FEATURE_COUNT = TEXTENSIBLE_ELEMENTS_FEATURE_COUNT + 15;
```

Explanation:

Each feature that is used in the graphical editor has a feature number, which is unique in its scope, so the feature `TACTIVITY__SUPPRESS_JOIN_FAILURE` is the only feature within the

type `TActivity` with the number 14 (plus an offset). For iterations over all features the controller uses the variable `TACTIVITY_FEATURE_COUNT`, which has to be incremented for each added feature.

For each of the activities, there is one feature count variable. This variable represents the information how many features (attributes or class references) an activity has.

When a `standard-attribute`, that is valid for all activities, is added, the abstract activity feature count has to be incremented. In any other case, the feature count of the specific activity has to be incremented.

5.2.7. Model Descriptor Implementation

The Model Descriptor Implementation provides the access to the feature that has been declared in its interface. The code which was produced by the Eclipse Modeling Framework does not make use of this declaration; it uses its static value instead.

Listing 74: Adding the static feature number to the implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.BpelmodelPackageImpl.java  
Function: init  
  
//Implementation of the suppressJoinFailure attribute  
public EAttribute getTActivity SuppressJoinFailure() {  
    return (EAttribute)tActivityEClass.getEStructuralFeatures().get(14);  
}
```

Explanation:

It is unclear why the Eclipse Modeling Framework generates static references, although the variable `TACTIVITY__SUPPRESS_JOIN_FAILURE` is accessible from within this context. In order to stick to the style of the generated code a static reference is used here also. The number is the same as the one added to `TACTIVITY__SUPPRESS_JOIN_FAILURE` in the class loader of the interface.

Listing 75: Initial creation of the attribute in the model implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.BpelmodelPackageImpl.java  
Function: createPackageContents  
  
createEAttribute(tActivityEClass, TACTIVITY__SUPPRESS_JOIN_FAILURE);
```

Listing 76: Modeling the attribute

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.BpelmodelPackageImpl.java  
Function: initializePackageContents  
  
//Attribute suppressJoinFailure initialization  
initEAttribute(getTActivity SuppressJoinFailure(),  
   .ecorePackage.getEString(),  
    "suppressJoinFailure",  
    "", //may not be null  
    0,  
    1,  
    TActivity.class,  
    !IS_TRANSIENT, !IS_VOLATILE, IS_CHANGEABLE, !IS_UNSETTABLE,  
    !IS_ID, !IS_UNIQUE, !IS_DERIVED, !IS_ORDERED);
```

Explanation:

The generic function `initEAttribute` is used to initialize the datatype and its behavior in the editor. The first parameter, `getTActivity SuppressJoinFailure()`, is a function call of the attribute constructor. The second attribute defines the datatype of the attribute. In this

case the type `ecorePackage.getEString()` is used, a datatype to hold strings. The third parameter is the internal name. The fourth is its initial value. The following numbers are upper and lower bounds for numbered data types. Next is the parent class of the attribute. The other parameters are additional settings for the Property View of the graphical editor. For a complete description of the addition settings see the IBM redbook for GEF and EMF [MDGW04, pp.24].

5.3. Integration of new constructs

As an example for completely new constructs the integration of the activity `validate` (see 2.4.1) is described in detail. `Validate` is a basic activity with one attribute and no functionality for nesting other constructs like for example the new activity `forEach`. The integration of this activity is already quite complex and has thus been chosen as example for demonstration.

5.3.1. BPEL output generator

The BPEL output generator produces the BPEL code out of a process instance. Each construct has an according function for building its specific subtree.

Listing 77: Adding the function call for the validate subtree

```
File: org.xmlsoap.schemas.ws.bpeleditor.xml.BpelBuilder.java
Function: buildActivitySubtree

import org.xmlsoap.schemas.ws.bpelmodel.TValidate;

//Insertion of new activity Validate
if (activity instanceof TValidate) {
    buildValidateSubtree(root, (TValidate)activity);
}
```

Explanation:

The function `buildActivitySubtree` calls the according function to generate BPEL output for the specific type, in this case for a `validate` activity. Note that class-casting has to be used extensively in the controller for the usage of the generated data model.

Listing 78: Building the validate subtree

```
File: org.xmlsoap.schemas.ws.bpeleditor.xml.BpelBuilder.java
Function: buildValidateSubtree

//Insertion of new Activity Validate
Element validateElement = new Element("validate", ns);

if (!validateActivity.eCrossReferences().isEmpty()) {
    getSourceLinkNames(validateElement, (TActivity)validateActivity);
    getTargetLinkNames(validateElement, (TActivity)validateActivity);
}

// Standard attribute: name
if ((validateActivity.getName() != null) &&
    !(validateActivity.getName().equals(""))){
    validateElement.setAttribute(new Attribute("name",
        validateActivity.getName()));
}

//optional: suppressJoinFailure
if ((validateActivity.getSuppressJoinFailure() != null) &&
    !(validateActivity.getSuppressJoinFailure().equals(""))){
```

```

        validateElement.setAttribute(new Attribute("suppressJoinFailure",
eliminateNull(validateActivity.getSuppressJoinFailure())));
    }

    if ((validateActivity.getVariables() != null) &&
        !(validateActivity.getVariables().equals(""))) {
        validateElement.setAttribute(new Attribute("variables",
eliminateNull(validateActivity.getVariables())));
    }

    root.addContent(validateElement);

```

Explanation:

Building the tree for the validate activity has two main parts. The first part is the calculation of the links from and to the activity, done by the function `getSourceLinkNames` and `getTargetLinkNames`. Afterwards the attributes and elements of the construct are processed, in this case the attributes `name`, `suppressJoinFailure` and `variables`. If the activity would have nested constructs then a call of the `buildActivitySubtree` would be necessary, this is for this basic activity not the case.

5.3.2. Validate Edit Part

This editor is, among other frameworks, based on the Graphical Editing Framework (GEF). This framework provides functions for graphically editing any part of a model instance that is connected by the controller: each class in the model needs a so-called Edit Part to be accessible by this framework.

Listing 79: Edit Part for the validate activity

```

File: org.xmlsoap.schemas.ws.bpeleditor.editparts.ValidateEditPart.java

public class ValidateEditPart extends ActivityEditPart
    implements NodeEditPart, Adapter {
    protected AdapterFactoryLabelProvider labelProvider;
    protected AdapterFactory adapterFactory;
    private IPropertySource propertySource = null;
    private Notifier target;

    public ValidateEditPart(TValidate activity) {
        super(activity);
        setModel(activity);
    }

    protected ActivityFigure getActivityFigure() {
        return (ActivityFigure) getFigure();
    }

    protected void refreshVisuals() {
        getActivityFigure().setName(getActivity().getName());
        Point loc = new Point(getActivity().getX(), getActivity().getY());
        Dimension size = new Dimension((int) getActivity().getWidth(),
                                        (int) getActivity().getHeight());
        Rectangle r = new Rectangle(loc, size);
        if (getParent() instanceof GraphicalEditPart)
            ((GraphicalEditPart) getParent()).setLayoutConstraint(this,
                                                                    getFigure(), r);
    }

    public boolean isAdapterForType(Object type) {
        return type.equals( getModel().getClass() );
    }

    public Notifier getTarget() {
        return target;
    }

    public void setTarget(Notifier newTarget) {

```

```

        target = newTarget;
    }

    public Object getAdapter(Class key) {
        if (IPropertySource.class == key) {
            return getPropertySource();
        }
        return super.getAdapter( key );
    }

    protected IPropertySource getPropertySource() {
        if( propertySource == null ) {
            propertySource = new EObjectPropertySource(getActivity());
        }
        return propertySource;
    }

    protected IFigure createFigure() {
        //Create an Object for the item Provider
        Object image = BpelmodelEditPlugin.INSTANCE.getImage(
            "full/obj16/TValidate");
        //Cast Object into Image
        Image img = ExtendedImageRegistry.getInstance().getImage(image);
        //Create new label with image
        Label l = new Label(img);
        ActivityFigure figure = new ActivityFigure(l);
        return figure;
    }

    protected AdapterFactoryLabelProvider getLabelProvider() {
        adapterFactory = new BpelmodelItemProviderAdapterFactory();
        if (labelProvider == null)
            labelProvider = new AdapterFactoryLabelProvider(adapterFactory);
        return labelProvider;
    }
}

```

Explanation:

- The class constructor `ValidateEditPart` binds the activity model to Edit Part.
- `getActivityFigure` is the standard getter function for the activity figure/icon. The call is delegated to the class `ActivityEditPart`.
- The function `refreshVisuals` is used for redrawing and rescaling of the graphical object, especially when it is nested in other objects. Therefore `Draw2d` functions are being used.
- The function `isAdapterForType` is used for identifying the object when the type is unknown in some part of the controller.
- The functions `getTarget` and `setTarget` are used for positioning of the graphical object on the editor pane.
- The function `getPropertySource` provides the functionality of the activity for being accessible in the eclipse Property View; also the function `getAdapter` is used in this context.
- `createFigure` is a function provided for the integration into the Graphical Editing Framework, it provides the figure that is drawn on the editor pane.

5.3.3. Edit Parts factory

All Edit Parts are instantiated in the Edit Parts factory, where the `validate` Edit Part has to be registered.

Listing 80: Edit Parts factory including the Edit Part for validate

```

File: org.xmlsoap.schemas.ws.bpelmodel.editparts.GraphicalEditPartsFactory.java
Function: createEditPart

import org.xmlsoap.schemas.ws.bpelmodel.TValidate;

```

```

. . .
//Insertion of new activity Validate
else if(obj instanceof TValidate) {
    ValidateEditPart p = new ValidateEditPart((TValidate)obj);
    p.setShell(shell);
    return p;
}

```

5.3.4. Editor palette

The editor palette is the feature of the graphical editor where the user can select constructs to include into a process model. The constructs are displayed with an icon, a display name and a ToolTip, that gives a more detailed description when the mouse cursor is over the icon. To make the activity validate accessible within the editor palette it has to be registered accordingly.

Listing 81: Adding the activity to the editor palette

```

File: org.xmlsoap.schemas.ws.bpeleditor.editor.ProcessPaletteRoot.java
Function: ProcessPaletteRoot

import org.xmlsoap.schemas.ws.bpelmodel.TValidate;
. . .

PaletteDrawer symbols = new PaletteDrawer("Activities",null);
add(symbols);
...
//New activity Validate
entry = new CombinedTemplateCreationEntry(
    "Validate",
    "Create Validate Activity",
    TValidate.class,
    new ModelCreationFactory(TValidate.class),
    BpelEditorPlugin.getDefault().getImageDescriptor(
        "icons/full/obj16/TValidate.gif"),
    BpelEditorPlugin.getDefault().getImageDescriptor(
        "icons/full/obj16/TValidate.gif"));
symbols.add(entry);

```

Explanation:

The editor palette is divided into several sections like Activities or Structured Activities. The activity validate is a basic activity and thus inserted into the Activities section. The class `CombinedTemplateCreationEntry` is constructed with all relevant parameters for the entry in the palette: the name, the ToolTip, the class type, the data model and the icons for displaying the activity on the editor palette and on the editor pane.

5.3.5. Insertion of icons for the activity into project

The icons used by the Graphical Editing Framework have to be designed and inserted into the project. This refers to the graphic file `TValidate.gif` (16 x 16 pixels, transparent GIF) which has to be inserted into the project `BpelEditor` in the folder `/full/obj16/` (for the editor palette) and in the project `BpelModel.edit` in the folder `/icons/full/obj16` (for the editor pane) (see also Listing 81:).



Figure 5: Icon for new construct validate

5.3.6. Commands on the Edit Part

When a new construct is integrated into the editor the commands on the Edit Parts have to be extended. This extension provides the functionality for creating and deleting the activity.

Listing 82: Extension of the delete command

```
File: org.xmlsoap.schemas.ws.bpeleditor.editpolicy.ActivityComponentEditPolicy  
Function: createDeleteCommand  
  
import org.xmlsoap.schemas.ws.bpelmodel.TValidate;  
. . .  
  
//Delete command support for new activity validate  
if (getHost().getModel() instanceof TValidate) {  
    deleteCmd.setActivity((TValidate) getHost().getModel());  
}
```

Listing 83: Extension of the create command

```
File: org.xmlsoap.schemas.ws.bpeleditor.editpolicy.ProcessXYLayoutPolicy.java  
Function: getCreateCommand  
  
import org.xmlsoap.schemas.ws.bpelmodel.TValidate;  
. . .  
  
if (newObjectType == TValidate.class) {  
    create = new CreateElementCommand();  
    create.setActivity((TValidate) request.getNewObject());  
}
```

5.3.7. Display in the Property View

For correctly displaying the activity in the eclipse Property View it is necessary to extend the class `EObjectPropertySource` and register the activity at the `getGroupName` function. This class extends the object meta-data provided by the Model Descriptor. For example it is providing enumerations of attribute values as combo boxes in the Property View. It also groups the attributes of an activity into BPEL attributes and View attributes.

```
File: org.xmlsoap.schemas.ws.bpeleditor.model.EObjectPropertySource  
Function: getGroupName  
  
. . .  
else if (. . . || className.equals("TValidate") . . . ) {  
    groupName = "Activity: " + className.substring(1, className.length());  
}  
return groupName;
```

Explanation:

When the activity is registered in this function, it will be displayed in the Property View in the scheme `Activity: Validate` followed by a list of its attributes.

5.3.8. Registering at the Model Creation Factory

The Model Creation Factory is the class that answers requests from the Graphical Editing Framework for the creation of instances of classes of the data model. The request is redirected to the factory of the data model, `BpelmodelFactory`.

Listing 84: Registering at the Model Creation Factory

```
File: org.xmlsoap.schemas.ws.bpelmodel.model.ModelCreationFactory.java
Function: getNewObject
import org.xmlsoap.schemas.ws.bpelmodel.TValidate;
. . .

BpelmodelFactory factory = bpelmodelPackage.getBpelmodelFactory();
. . .
//Inserted new Activity Validate
else if (targetClass.equals(TValidate.class)) {
    result = factory.createTValidate();
}
return result;
```

5.3.9. BPEL Model Factory

Now the adaptations of the controller are complete and those of the data model begin: The BPEL Model Factory is described by the interface `BpelmodelFactory`. In this class the function to create instances of the data class `TValidate` is defined.

Listing 85: Definition of the function for instance creation in the BPEL Model Factory

```
File: org.xmlsoap.schemas.ws.bpelmodel.BpelmodelFactory.java
TValidate createTValidate();
```

5.3.10. BPEL Model Factory implementation

The BPEL Model Factory implementation provides one function for creating any instance of the data model, `create`. The creation of the specific class is redirected to the specific instance creation function `createTValidate`, where the class is actually instantiated.

Listing 86: General instance creation function

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.BpelmodelFactoryImpl.java
Function: create

switch (eClass.getClassifierID()) {
    //Inserted new activity Validate
    case BpelmodelPackage.TVALIDATE: return createTValidate();
```

Listing 87: Specific instance creation function

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.BpelmodelFactoryImpl.java
Function: createTValidate

TValidateImpl tValidate = new TValidateImpl();
return tValidate;
```

5.3.11. Model Descriptor

Each feature that is used in the graphical editor has a feature number, which is unique in its scope. When a new activity is integrated, many new features have to be added, one for each explicit (e.g. `validate: name`) or implicit (e.g. `validate: width`) feature. Additionally the activity has to be registered as a feature of the `process` and the activity container to enable the nesting of the activity into activity containers like `flow` and the `process` itself. Also the functions for accessing the class instance, its attributes or elements and references are defined in the model descriptor.

Listing 88: Registering the activity at the model descriptor

```
File: org.xmlsoap.schemas.ws.bpelmodel.BpelmodelPackage.java
Function: interface BpelmodelPackage

//Registering new activity Validate on activity container
int TACTIVITY CONTAINER VALIDATE = 14;
//Was before insertion: int TACTIVITY CONTAINER FEATURE COUNT = 14;
int TACTIVITY CONTAINER FEATURE COUNT = 15;

//Registering new activity Validate on process
int TPROCESS VALIDATE = TEXTENSIBLE ELEMENTS FEATURE COUNT + 28;
//Was before insertion: 28, now:
int TPROCESS FEATURE COUNT = TEXTENSIBLE ELEMENTS FEATURE COUNT + 29;

//Defining new Activity: Validate
//Last Value: 59
int TVALIDATE = 60;
//General
int TVALIDATE ANY = TACTIVITY ANY;
int TVALIDATE ANY ATTRIBUTE = TACTIVITY ANY ATTRIBUTE;
int TVALIDATE WIDTH = TACTIVITY WIDTH;
int TVALIDATE HEIGHT = TACTIVITY HEIGHT;
int TVALIDATE SOURCE CONNECTIONS = TACTIVITY SOURCE CONNECTIONS;
int TVALIDATE TARGET CONNECTIONS = TACTIVITY TARGET CONNECTIONS;
int TVALIDATE ACTIVITY = TACTIVITY ACTIVITY;
int TVALIDATE LOCATION = TACTIVITY LOCATION;
int TVALIDATE X = TACTIVITY X;
int TVALIDATE Y = TACTIVITY Y;
int TVALIDATE NAME = TACTIVITY NAME;
int TVALIDATE TARGETS = TACTIVITY TARGET;
int TVALIDATE SOURCES = TACTIVITY SOURCE;
int TVALIDATE BP EL TEMPLATE = TACTIVITY BP EL TEMPLATE;
int TVALIDATE PROCESS = TACTIVITY PROCESS;
//Features
int TVALIDATE VARIABLES = TACTIVITY FEATURE COUNT + 0;
int TVALIDATE FEATURE COUNT = TACTIVITY FEATURE COUNT + 1; //Has 1 feature

//Validate References:
EReference getActivityContainer Validate();
EReference getTProcess Validate();
EClass getTValidate();
EAttribute getTValidate_Variables();
```

Explanation:

- The activity `validate` is registered at the `activity container` and the `process` as a feature whereas the total number of features has to be incremented.
- Each feature of the activity has to be registered at the model descriptor; all of the standard features like `width`, `height`, `name` etc. are already implemented by the abstract activity interface `TActivity` so no new feature numbers have to be set for these features.
- New features - attributes or elements of the activity - have to be registered using a new feature number with the offset `TACTIVITY_FEATURE_COUNT` and also the total number of features of the activity has to be defined the same way. Note that for the integration of structured activities class references have to be listed before attributes.
- The definitions of references contain the `process`, the `activity container`, the actual activity class and all of its non-standard features.

5.3.12. Model Descriptor Implementation

The functions for the integration of the activity into the model are directly related to the feature definitions in the model descriptor.

Listing 89: Model Descriptor Implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.BpelmodelPackageImpl.java
Function: class BpelmodelPackageImpl

import org.xmlsoap.schemas.ws.bpelmodel.TValidate;

//Insertion for new Activity Validate
private EClass tValidateEClass = null;

//Insertions for activity validate
public EClass getTValidate(){
    return tValidateEClass;
}

public EAttribute getTValidate Variables() {
    return (EAttribute)tValidateEClass.getEStructuralFeatures().get(0);
}

public EReference getTActivityContainer Validate(){
    //Adapt Structural Feature number for new activity Validate
    return (EReference)
        tActivityContainerEClass.getEStructuralFeatures().get(14);
}

public EReference getTProcess Validate() {
    //Adapt structural Feature number for new activity Validate
    return (EReference)tProcessEClass.getEStructuralFeatures().get(28);
}
```

Explanation:

Any reference or attribute is identified by its feature number defined in the Model Descriptor. The fact that the feature numbers from the Model Descriptor are not referenced here but rather statically inserted by the code generator is strange. It leads back to the implementation of the Eclipse Modeling Framework (EMF) that has been used to generate this code. In order to stay compliant with the generated code this form has been retained.

The contents of the Model Descriptor are first created (`createPackageContents`) and afterwards initialized (`initializePackageContents`) in a separate function.

Listing 90: Create contents of the Model Descriptor Implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.BpelmodelPackageImpl.java
Function: createPackageContents

//Insertions for new activity Validate
createEReference(tActivityContainerEClass, TACTIVITY_CONTAINER_VALIDATE);
createEReference(tProcessEClass, TPROCESS_VALIDATE);
tValidateEClass = createEClass(TVALIDATE);
createEAttribute(tValidateEClass, TVALIDATE_VARIABLES);
```

Listing 91: Initialize contents of the Model Descriptor Implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.BpelmodelPackageImpl.java
Function: initializePackageContents

//Add Supertype to class
tValidateEClass.getESuperTypes().add(this.getTActivity());
//Activity Container Reference

initEReference(getTActivityContainer Validate(), this.getTValidate(), null,
    "validate", null, 0, 1, TActivityContainer.class, !IS_TRANSIENT, !IS_VOLATILE,
    IS_CHANGEABLE, IS_COMPOSITE, !IS_RESOLVE_PROXIES, !IS_UNSETTABLE, IS_UNIQUE,
    !IS_DERIVED, IS_ORDERED);

initEReference(getTProcess Validate(), this.getTValidate(), null, "validate",
    null, 0, 1, TProcess.class, !IS_TRANSIENT, !IS_VOLATILE, IS_CHANGEABLE,
```

```
IS_COMPOSITE, !IS_RESOLVE_PROXIES, !IS_UNSETTABLE, IS_UNIQUE, !IS_DERIVED,
IS_ORDERED);

initEClass(tValidateEClass, TValidate.class, "TValidate", !IS_ABSTRACT,
!IS_INTERFACE, IS_GENERATED_INSTANCE_CLASS);

initEAttribute(getTValidate Variables(),.ecorePackage.getEString(), "variables",
null, 0, 1, TValidate.class, !IS_TRANSIENT, !IS_VOLATILE, IS_CHANGEABLE,
!IS_UNSETTABLE, !IS_ID, !IS_UNIQUE, !IS_DERIVED, IS_ORDERED);
```

5.3.13. Adapting the process definition

The process definition `TProcess` has to be adapted for the reference of the new activity. Note that also the activity container definition `TActivityContainer` has to be adapted in the same manner.

Listing 92: Adapting the process definition

```
File: org.xmlsoap.schemas.ws.bpelmodel.TProcess.java
Function: interface TProcess

//Insertion for new activity validate
TValidate getValidate();
void setValidate(TValidate value);
```

5.3.14. Adapting the process implementation

The getter and setter function for the validate activity have to be integrated in the process implementation `TProcessImpl`. This integration contains the protected variable `validate`, the getter function `getValidate` and two setter functions, `basicSetValidate` (only for objects) and `setValidate` (for all types). Note that also the activity container implementation `TActivityContainerImpl` has to be adapted in the same manner.

Listing 93: Adding getter and setter of the activity to the process implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TProcessImpl.java
Function: class TProcessImpl

//Insertion for new activity validate
protected TValidate validate = null;

public TValidate getValidate() {
    return validate;
}

public NotificationChain basicSetValidate(TValidate newValidate,
                                           NotificationChain msgs) {
    TValidate oldValidate = validate;
    validate = newValidate;
    if (eNotificationRequired()) {
        ENotificationImpl notification = new ENotificationImpl(this,
            Notification.SET, BpelmodelPackage.TPROCESS_VALIDATE,
            oldValidate, newValidate);
        if (msgs == null) msgs = notification; else msgs.add(notification);
    }
    return msgs;
}

public void setValidate(TValidate newValidate) {
    if (newValidate != validate) {
        NotificationChain msgs = null;
        if (validate != null)
            msgs = ((InternalEObject)validate).eInverseRemove(this,
                EOPPOSITE_FEATURE_BASE -
                BpelmodelPackage.TPROCESS_VALIDATE, null, msgs);
```

```

        if (newValidate != null)
            msgs = ((InternalEObject)newValidate).eInverseAdd(this,
                EOPPOSITE_FEATURE_BASE -
                BpelmodelPackage.TPROCESS_VALIDATE, null, msgs);
        msgs = basicSetValidate(newValidate, msgs);
        if (msgs != null) msgs.dispatch();
    }
    else if (eNotificationRequired())
        eNotify(new ENotificationImpl(this, Notification.SET,
            BpelmodelPackage.TPROCESS_VALIDATE,
            newValidate, newValidate));
}

```

The function `eInverseRemove` has to be adapted for the integration of the activity. According to its Javadoc documentation this function “removes the object at the other end of a bidirectional reference from the appropriate feature and returns accumulated notifications.” For details see the JavaDoc on `org.eclipse.emf.ecore.InternalEObject`.

Listing 94: Adapting the `eInverseRemove` function of the process implementation

```

File: org.xmlsoap.schemas.ws.bpelmodel.impl.TProcessImpl.java
Function: eInverseRemove

switch (eDerivedStructuralFeatureID(featureID, baseClass)) {
    . . .
    //Inserted for new activity validate
    case BpelmodelPackage.TPROCESS_VALIDATE:
        return basicSetValidate(null, msgs);
}

```

Next the functions for dynamic access to process features and references have to be adapted, `eGet`, `eSet`, `eUnset` and `eIsSet`. This adaptation is similar to the integration of a new attribute.

Listing 95: Adapting `eGet` on the process implementation

```

File: org.xmlsoap.schemas.ws.bpelmodel.impl.TProcessImpl.java
Function: eGet

switch (eDerivedStructuralFeatureID(eFeature)) {
    . . .
    //Inserted for new activity validate
    case BpelmodelPackage.TPROCESS_VALIDATE:
        return getValidate();
}

```

Listing 96: Adapting `eSet` on the process implementation

```

File: org.xmlsoap.schemas.ws.bpelmodel.impl.TProcessImpl.java
Function: eSet

switch (eDerivedStructuralFeatureID(eFeature)) {
    . . .
    case BpelmodelPackage.TPROCESS_VALIDATE:
        setValidate((TValidate)newValue);
        return;
}

```

Listing 97: Adapting `eUnset` on the process implementation

```

File: org.xmlsoap.schemas.ws.bpelmodel.impl.TProcessImpl.java
Function: eIsSet

switch (eDerivedStructuralFeatureID(eFeature)) {
    . . .
    case BpelmodelPackage.TPROCESS_VALIDATE:
        setValidate((TValidate)null);
        return;
}

```

Listing 98: Adapting elsSet on the process implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TProcessImpl.java
Function: elsSet

switch (eDerivedStructuralFeatureID(eFeature)) {
    . . .
    case BpelmodelPackage.TPROCESS__VALIDATE:
        return validate != null;
}
```

5.3.15. Generating activity classes

The activity definition (here: `TValidate`) and the activity implementation (here: `TValidateImpl`) are generated using the Eclipse Modeling Framework (EMF). Input for the code generation is the BPEL 2.0 XML Schema Definition for executable processes [BPEL2.0XSD] provided by the OASIS Organization. In the first step the Java interfaces of the data model are generated and adapted for usage in the Graphical Editing Framework (GEF) as described in [Kapl06, pp. 34]. This adaptation includes the extension by the attributes `width`, `height`, `x`, `y` and lists for the graphical display of the connections between activities, `SourceConnections` and `TargetConnections`. For this activity the data type of the attribute `variables` has also been changed from `List` to `String` for the purpose of simplification of the already complex example. The adapted java interfaces are afterwards used as input for the second and final code generation as described in [Kapl06, p.36].

The resulting classes for the activity `validate`, `TValidate` and `TValidateImpl` have to be imported into the according package (`org.xmlsoap.schemas.ws.bpelmodel` and `org.xmlsoap.schemas.ws.bpelmodel.impl`) in the `BpelModel` project. Afterwards they have to be adapted to the current package structure.

5.3.16. Adapting the activity definition

The activity definition has to be adapted to the current package structure. When it was generated it was located in the package `org.open.oasis.docs.wsbpel._2._0.process.executable.impl`. After renaming and omitting the comments added by the code generator the activity definition is quite clear.

Listing 99: Adapted activity definition

```
File: org.xmlsoap.schemas.ws.bpelmodel.TValidate.java

package org.xmlsoap.schemas.ws.bpelmodel;
public interface TValidate extends TActivity {
    String getVariables();
    void setVariables(String value);
}
```

5.3.17. Adaptation of the activity implementation

In the same way the activity implementation has to be adapted, the package has to be renamed from the new BPEL 2.0 package naming to the former BPEL 1.1 naming:

BPEL 2.0: `org.open.oasis.docs.wsbpel._2._0.process.executable.impl`

BPEL 1.1: `org.xmlsoap.schemas.ws.bpelmodel.impl`

Afterwards the references to `ExecutablePackage` have to be replaced by references to `BpelmodelPackage`, this can be done by using the find and replace function of eclipse. After this the import statements have to be extended:

Listing 100: New imports in the activity implementation

```
import org.xmlsoap.schemas.ws.bpelmodel.BpelTemplate;
import org.xmlsoap.schemas.ws.bpelmodel.BpelmodelPackage;
import org.xmlsoap.schemas.ws.bpelmodel.TValidate;
```

In the function `eInverseRemove` some changes are necessary for the integration into the existing model.

Listing 101: Adaptation the `eInverseRemove` function of the activity implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TValidateImpl.java
Function: eInverseRemove

switch (eDerivedStructuralFeatureID(featureID, baseClass)) {
    //Should be implemented Abstract
    //case BpelmodelPackage.TVALIDATE DOCUMENTATION:
    //return ((InternalEList)getDocumentation()).basicRemove(otherEnd, msgs);
    case BpelmodelPackage.TVALIDATE TARGETS:
        return ((InternalEList)getTarget()).basicRemove(otherEnd, msgs);
    case BpelmodelPackage.TVALIDATE SOURCES:
        return ((InternalEList)getSource()).basicRemove(otherEnd, msgs);
}
```

Explanation:

- The `documentation` element should be implemented as an attribute of the abstract `TActivity` class and should thus be inherited by all activities.
- The notation of `target` and `source` has to be adapted to the old notation; in the new standard links are enveloped into `targets` and `sources` (see 2.2.3).

Just like with the adaptation of the process implementation the functions for accessibility of the activity features and references have to be adapted, i.e. `eGet`, `eSet`, `eUnset` and `eIsSet`. In the following listing only the changes are shown that have to be made, other code is not affected.

Listing 102: Adapting `eGet` on the process implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TValidateImpl.java
Function: eGet

switch (eDerivedStructuralFeatureID(eFeature)) {
    //Should be implemented Abstract
    //case BpelmodelPackage.TVALIDATE DOCUMENTATION:
    //    return getDocumentation();
    case BpelmodelPackage.TVALIDATE TARGETS:
        return getTarget();
    case BpelmodelPackage.TVALIDATE SOURCES:
        return getSource();
    case BpelmodelPackage.TACTIVITY SUPPRESS JOIN FAILURE:
        return getSuppressJoinFailure();
    case BpelmodelPackage.TVALIDATE BPEL TEMPLATE:
        if (resolve) return getBpelTemplate();
        return basicGetBpelTemplate();
    case BpelmodelPackage.TACTIVITY JOIN CONDITION:
        return getJoinCondition();
}
```

Explanation:

- The `documentation` element is commented out in all functions for dynamic access.
- Also the notation of `target` and `source` has to be adapted in those functions
- `suppressJoinFailure` is an attribute (see 2.2.1), which has already been modeled as an attribute of the abstract activity `TActivity` (see 5.2) and must hence be implemented in this function.
- The support for BPEL templates, see [Kapl06, pp.9] is maintained.
- `joinCondition` has been modeled as an attribute of the abstract activity `TActivity` in the prior version of the editor and must hence be implemented in this function.

The other functions for accessibility have according modifications and are listed here for completeness.

Listing 103: Adapting eSet on the process implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TValidateImpl.java
Function: eSet

switch (eDerivedStructuralFeatureID(eFeature)) {
    //Should be implemented Abstract
    //case BpelmodelPackage.TVALIDATE DOCUMENTATION:
    //    getDocumentation().clear();
    //    getDocumentation().addAll((Collection)newValue);
    //    return;
    case BpelmodelPackage.TVALIDATE TARGETS:
        getTarget().clear();
        getTarget().addAll((Collection)newValue);
    case BpelmodelPackage.TVALIDATE SOURCES:
        getSource().clear();
        getSource().addAll((Collection)newValue);
    case BpelmodelPackage.TACTIVITY SUPPRESS JOIN FAILURE:
        setSuppressJoinFailure((String) newValue);
        return;
    case BpelmodelPackage.TVALIDATE BPEL TEMPLATE:
        setBpelTemplate((BpelTemplate)newValue);
        return;
    case BpelmodelPackage.TACTIVITY JOIN CONDITION:
        setJoinCondition((String)newValue);
        return;
}
```

Listing 104: Adapting eUnset on the process implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TValidateImpl.java
Function: eUnset

switch (eDerivedStructuralFeatureID(eFeature)) {
    //Should be implemented Abstract
    //case BpelmodelPackage.TVALIDATE DOCUMENTATION:
    //    getDocumentation().clear();
    //    return;
    case BpelmodelPackage.TVALIDATE TARGETS:
        getTarget().clear();
        return;
    case BpelmodelPackage.TVALIDATE SOURCES:
        getSource().clear();
        return;
    case BpelmodelPackage.TACTIVITY SUPPRESS JOIN FAILURE:
        setSuppressJoinFailure(SUPPRESS JOIN FAILURE EDEFAULT);
        return;
    case BpelmodelPackage.TVALIDATE BPEL TEMPLATE:
        setBpelTemplate((BpelTemplate)null);
        return;
    case BpelmodelPackage.TACTIVITY JOIN CONDITION:
        setJoinCondition(JOIN CONDITION EDEFAULT);
        return;
}
```

Listing 105: Adapting elsSet on the process implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TValidateImpl.java
Function: eIsSet

switch (eDerivedStructuralFeatureID(eFeature)) {
    //Should be implemented Abstract
    //case BpelmodelPackage.TVALIDATE DOCUMENTATION:
    //    return documentation != null && !documentation.isEmpty();
    case BpelmodelPackage.TVALIDATE TARGETS:
        return target != null && !target.isEmpty();
    case BpelmodelPackage.TVALIDATE SOURCES:
        return source != null && !source.isEmpty();
}
```

```

case BpelmodelPackage.TACTIVITY SUPPRESS JOIN FAILURE:
    return suppressJoinFailure != null;
case BpelmodelPackage.TVALIDATE BPEL TEMPLATE:
    return bpelTemplate != null;
case BpelmodelPackage.TACTIVITY JOIN CONDITION:
    return JOIN CONDITION EDEFAULT == null ? joinCondition != null :
        !JOIN CONDITION EDEFAULT.equals(joinCondition);
}

```

5.4. Removal of attributes and elements

As an example the removal of the `process` attribute `abstractProcess` (see 2.2.1) is described in detail. First the modification of the BPEL output generator is explained and consecutively the changes that have to be made in the data model. The removal of an (XML) element is not described in detail, as the procedure is same with the one for removing an attribute. In principle the removal of an attribute is almost the inverse of the addition of an attribute (see 5.2), so almost the same steps have to be made in reverse order.

5.4.1. BPEL output generator

The BPEL output generator is the unit that produces the BPEL code from the BPEL process instance. Each type of construct has a related function to build its sub tree in the JDOM tree, which has to be modified accordingly. In this case, a `process` attribute, it refers to the function `buildProcessSubtree`:

Listing 106: Adaptation of BPEL output generator for attribute removal

```

File: org.xmlsoap.schemas.ws.bpeleditor.xml.BpelBuilder.java
Function: buildProcessSubtree

//removing the attribute abstractProcess
//rootElement.setAttribute(new Attribute("abstractProcess",
//eliminateNull(String.valueOf(process.isAbstractProcess()))));

```

Explanation:

The function `buildProcessSubtree` prepares the root element of the tree: `process`. The variable `process`, which provides the data value by the function `isAbstractProcess`, is implemented by the class `TProcessImpl` which is typed as the interface `TProcess`.

5.4.2. Related Construct Implementation

Again, the adaptations of the controller are complete and those of the data model begin: The parent construct implementation, in case of the `abstractProcess` attribute this is the `process` construct implementation `TProcessImpl`, contains the functionality for accessing the attribute. Four kinds of functions for the accessibility are provided, `eGet`, `eSet`, `elsSet` und `eUnset`. Each has a `switch` construct to distinguish the requested feature which is represented by a number that is unique in its scope. As the feature (`TPROCESS__ABSTRACT_PROCESS`) has been removed the according case statement has to be deleted or commented also, see `eGet` as example:

Listing 107: Removing an attribute from the data model: eGet

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TProcessImpl.java  
Function: eGet  
  
switch (eDerivedStructuralFeatureID(eFeature)) {  
    .  
    .  
    .  
    //Removing the attribute abstractProcess  
    //case BpelmodelPackage.TPROCESS_ABSTRACT_PROCESS:  
    //return isAbstractProcess() ? Boolean.TRUE : Boolean.FALSE;  
}
```

5.4.3. Model Descriptor

Listing 108: Removing the dynamic feature number

```
File: org.xmlsoap.schemas.ws.bpelmodel.BpelmodelPackage.java  
Function: class BpelmodelPackage  
  
//Removing the definition of the attribute abstractProcess  
//EAttribute getTProcess AbstractProcess();  
  
//Removing the feature number of the attribute abstractProcess or set to 0  
//int TPROCESS_ABSTRACT_PROCESS = TEXTENSIBLE_ELEMENTS_FEATURE_COUNT + 24;  
  
//Exchange of the feature number with the last feature, it was 28 before  
int TPROCESS_TARGET_NAMESPACE = TEXTENSIBLE_ELEMENTS_FEATURE_COUNT + 24;  
  
//Feature number count decremented, it was 29 before:  
int TPROCESS_FEATURE_COUNT = TEXTENSIBLE_ELEMENTS_FEATURE_COUNT + 28;
```

Explanation:

Each feature that is used in the graphical editor has a feature number, which is unique in its scope, so the feature `TPROCESS_ABSTRACT_PROCESS` is the only feature within the type `TProcess` with the number 24 (plus an offset). For iterations over all features the graphical editor uses the variable `TPROCESS_FEATURE_COUNT`, which has to be decremented for each removed feature. Each construct has its own feature count. As there may not be a gap in between, the last feature takes the place of the removed feature.

Note: when the feature number is being removed, the optional modifications (see 5.4.5) have to be made. As an alternative it can be set to 0.

5.4.4. Model Descriptor Implementation

The model descriptor implementation contains functions for creating, initializing and preparing the attribute for accessibility by the controller. The attribute has to be removed from these functions for consistency with the model descriptor.

Listing 109: Exchanging the static feature number in the implementation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.BpelmodelPackageImpl.java  
Function: init  
  
//Removed attribute abstractProcess  
//public EAttribute getTProcess AbstractProcess() {  
//    return (EAttribute)tProcessEClass.getEStructuralFeatures().get(24);  
//}  
  
public EAttribute getTProcess TargetNamespace() {  
    //Exchanged feature number because the attribute  
    //abstractProcess was removed, was before: 28
```

```

        return (EAttribute)tProcessEClass.getEStructuralFeatures().get(24);
    }

```

Listing 110: Removing initial creation of the attribute in the model implementation

File: org.xmlsoap.schemas.ws.bpelmodel.impl.BpelmodelPackageImpl.java
Function: createPackageContents

```

//createEAttribute(tProcessEClass, TPROCESS__ABSTRACT_PROCESS);

```

Listing 111: Removing the modeling of the attribute

File: org.xmlsoap.schemas.ws.bpelmodel.impl.BpelmodelPackageImpl.java
Function: initializePackageContents

```

//Removing initialization for the attribute abstractProcess
//initEAttribute(getTProcess AbstractProcess(),.ecorePackage.getEBoolean(),
//    "abstractProcess", "yes", 1, 1, TProcess.class, !IS TRANSIENT,
//    !IS VOLATILE, IS CHANGEABLE, IS UNSETTABLE, !IS ID, !IS UNIQUE,
//    !IS DERIVED, IS ORDERED);

```

5.4.5. Optional modifications

In order to stay consistent with the generated code, also the basic functions, that provide the functionality for the attribute, can be removed.

Listing 112: Removing attribute on related construct interface

File: org.xmlsoap.schemas.ws.bpelmodel.TProcess.java

```

// boolean isAbstractProcess();
// void setAbstractProcess(boolean value);
// void unsetAbstractProcess();
// boolean isSetAbstractProcess();

```

Listing 113: Removing attribute on related construct implementation

File: org.xmlsoap.schemas.ws.bpelmodel.impl.TProcessImpl

```

// protected static final boolean ABSTRACT_PROCESS EDEFAULT = false;
// protected boolean abstractProcess = ABSTRACT_PROCESS EDEFAULT;
// protected boolean abstractProcessESet = false;
// public boolean isAbstractProcess()
// public void setAbstractProcess(boolean newAbstractProcess)
// public void unsetAbstractProcess()
// public boolean isSetAbstractProcess()

```

Listing 114: Removing unused model provider functions

File: org.xmlsoap.schemas.ws.bpelmodel.provider.TProcessItemProvider

```

// protected void addAbstractProcessPropertyDescriptor()

// public void notifyChanged(Notification notification) {
    . . .
    switch (notification.getFeatureID(TProcess.class)) {
        //Removing attribute abstractProcess
        //case BpelmodelPackage.TPROCESS__ABSTRACT_PROCESS:

```

```
// . . .

public List getPropertyDescriptors(Object object) {
    //Removing attribute abstractProcess
    //addAbstractProcessPropertyDescriptor(object);
}
```

Explanation:

The generated model provider code for describing properties has not been used in the graphical editor. To avoid compiler warnings and errors this code has to be removed.

Listing 115: Removing unused model annotation

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.BpelmodelPackageImpl.java

//public EAttribute getTProcess AbstractProcess()
//addAnnotation (getTProcess AbstractProcess(), source, new String[] {
//    "kind", "attribute", "name", "abstractProcess" });
```

5.5. Attributes and nested XML elements

For the transformation of attributes into nested XML elements like `until` in the activity `wait` (see 2.6.2) and the other way round like `portType` in `partnerLinkTypes` (see 2.3) integration can be achieved by adapting the BPEL output generator of the editor accordingly. In this section, as an example the transformation from attribute into nested XML element is described, the other way round is alike. This procedure does not require a modification of the data model.

5.5.1. BPEL output generator

Listing 116: Transformation of the attribute `until` into a nested XML element

```
File: org.xmlsoap.schemas.ws.bpeleditor.xml.BpelBuilder.java
Function: buildWaitSubtree

//Transform attribute "until" into element "until"
//if ((waitActivity.getUntil() != null) &&
//    !(waitActivity.getUntil().equals(""))){
//    waitElement.setAttribute(new Attribute("until", waitActivity.getUntil()));
//}

if ((waitActivity.getUntil() != null) && !(waitActivity.getUntil().equals(""))){
    Element untilElement = new Element("until", ns);
    untilElement.addContent(waitActivity.getUntil());
    waitElement.addContent(untilElement);
}
```

Explanation:

During the build-up of the JDOM tree that is used for the BPEL output generation the process tree is parsed. The data types `Attribute` and `Element` are treated by the output generator as an XML attribute and an XML element. The function `setAttribute` (`addContent`) inserts the data value into the attribute (element). The namespace `ns` is the `xmlns` namespace of the related process.

5.6. Activity renaming

As an example of providing new names to constructs the activity `Exit` (see 2.6.4), formerly known as `Terminate`, is described in detail. It is sufficient to change the naming at those parts of the editor, where the construct name is (graphically) displayed to the user on the one hand and where the final BPEL output is generated on the other. A change of file names and internal naming is not necessary.

5.6.1. BPEL output generator

The BPEL output generator is the unit that produces the BPEL code from the BPEL process instance. Each type of construct has a related function to build its subtree in the JDOM tree, in this example the function `buildTerminateSubtree`.

Listing 117: Construct naming in the BPEL output generator

```
File: org.xmlsoap.schemas.ws.bpeleditor.xml.Bpelbuilder.java  
Function: buildTerminateSubtree  
  
Element terminateElement = new Element("exit", ns);
```

Explanation:

The function to create a new XML element in the output is performed by the constructor of the JDOM class `element`. The two parameters are the name for the output XML element and the namespace of the XML element, `ns` is the namespace of the related `process` (see 2.2.1). In the output the namespace is not generated, as it is equal to the default namespace `xmlns`, which is already declared in the `process` element.

5.6.2. Editor palette

The editor palette is the feature of the graphical editor where the user can select constructs that shall be included in the process. The constructs are displayed with an icon, a display name and a ToolTip that gives a more detailed description when the mouse cursor is over the icon.

Listing 118: Construct naming in the editor palette

```
File: org.xmlsoap.schemas.ws.bpeleditor.editor.ProcessPaletteRoot.java  
Function: ProcessPaletteRoot  
  
entry = new CombinedTemplateCreationEntry(  
    "Exit", //Display name on the editor palette  
    "Create Exit Activity", //ToolTip  
    TTerminate.class,  
    new ModelCreationFactory(TTerminate.class),  
    BpelEditorPlugin.getDefault().getImageDescriptor(  
        "icons/full/obj16/TTerminate.gif"),  
    BpelEditorPlugin.getDefault().getImageDescriptor(  
        "icons/full/obj16/TTerminate.gif"));  
symbols.add(entry);
```

Explanation:

The `entry` variable is passed to the GEF function that visualizes the editor palette. In the function `ProcessPaletteRoot` the items of the editor palette are prepared.

6. Accomplished BPEL 2.0 Extensions

6.1. New constructs

Activities:

- validate (see 2.4.1)
- extensionActivity (see 2.6.3)
- forEach (see 2.7.4)

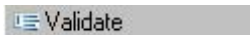


Figure 6: New constructs on the editor palette

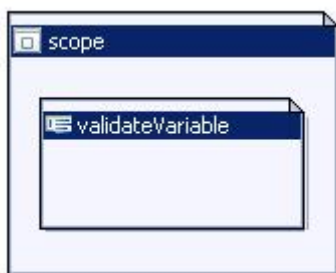


Figure 7: New construct `validate` nested inside a scope

```
<scope name="scope">
  <validate name="validateVariable" variables="itemsShipped" />
</scope>
```

Figure 8: New construct `validate` in BPEL code

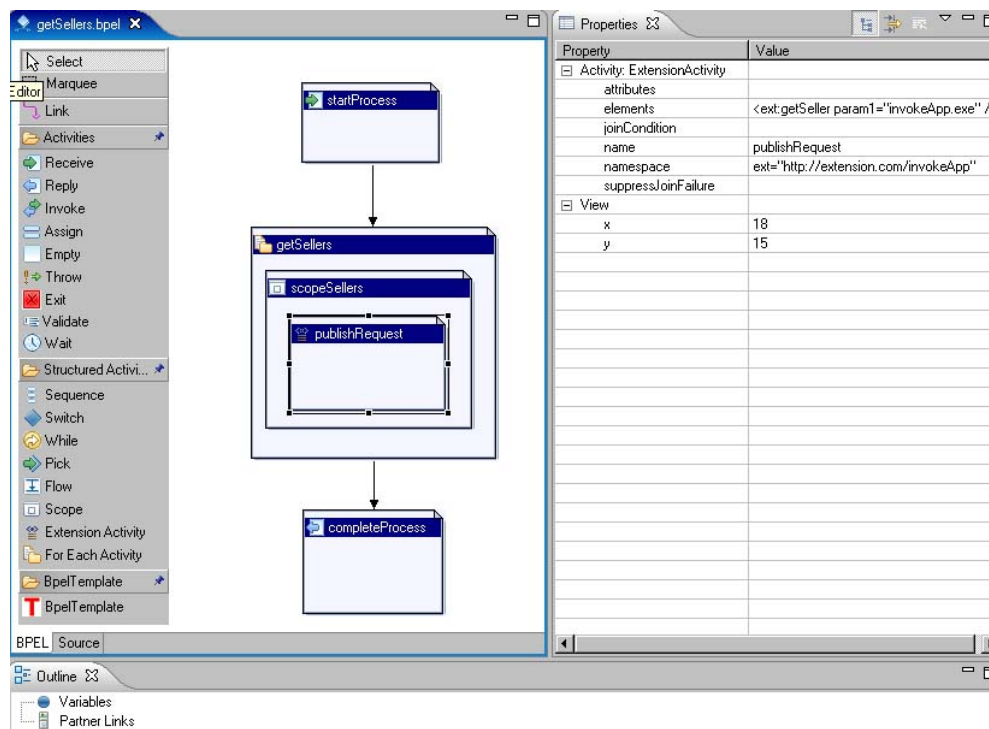


Figure 9: New constructs `forEach` and `extensionActivity` in a process

```

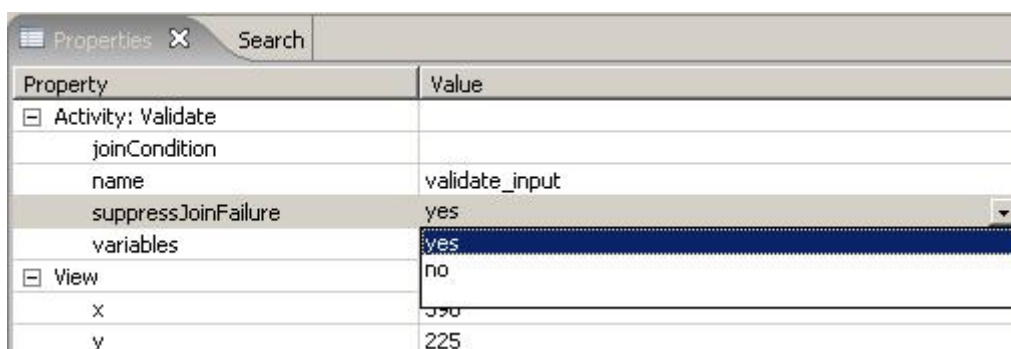
<forEach name="getSellers" counterName="countReplies" parallel="yes">
  <sources>
    <source linkName="getSellers-to-completeProcess" />
  </sources>
  <targets>
    <target linkName="startProcess-to-getSellers" />
  </targets>
  <startCounterValue>1</startCounterValue>
  <finalCounterValue>100</finalCounterValue>
  <completionCondition>
    <branches successfulBranchesOnly="yes">50</branches>
  </completionCondition>
  <scope name="scopeSellers">
    <extensionActivity name="publishRequest">
      <ext:getSeller xmlns:ext="http://extension.com/invokeApp" param1="invokeApp.exe" />
    </extensionActivity>
  </scope>
</forEach>

```

Figure 10: New construct `forEach` and `extensionActivity` in BPEL code

6.2. New attributes

- `suppressJoinFailure` (see 2.2.1) on:
 - `exit`
 - `invoke`
 - `scope`
 - `flow`
 - `assign`
 - `empty`
 - `pick`
 - `receive`
 - `reply`
 - `sequence`
 - `switch`
 - `throw`
 - `validate`
 - `wait`
 - `while`
- `xmlns` on the construct `process` (see 2.2.1)



Property	Value
Activity: Validate	
joinCondition	
name	validate_input
suppressJoinFailure	yes
variables	yes
View	no
x	350
y	225

Figure 11: New attribute `suppressJoinFailure` in Eclipse Property View

6.3. Removed constructs

No constructs have been removed, neither from the data model nor from the graphical editor.

6.4. Removed attributes

- `abstractProcess` on the `process` construct (see 2.2.1)

6.5. Attributes and nested XML elements

- Attribute `transitionCondition` on links (see 2.2.3)
- Nesting of element `source` on links into `sources` (see 2.2.3)
- Nesting of element `target` on links into `targets` (see 2.2.3)
- Attribute `joinCondition` on activities (see 2.2.3)
- Attribute `for` on the activity `Wait` (see 2.6.2)
- Attribute `until` on the activity `Wait` (see 2.6.2)
- Attribute `condition` on the activity `while` (see 2.7.2)

```
<wait name="wait">
  <for>P1Y2M3D</for>
</wait>
```

Figure 12: Attribute transformed into nested XML element on activity `wait`

6.6. Activity renaming

The following constructs have been renamed:

- `exit`, was formerly named `terminate` (see 2.6.4)



Figure 13: Activity `terminate` renamed to `exit` in the editor palette

6.7. Bug-fixing and Additional Features

6.7.1. Attribute values

Attributes in BPEL that stand for `boolean` use "yes" and "no" instead of "true" and "false" in both standards. The existing data model partly uses the native Java `boolean` type for modeling these kinds of attributes. To comply with the BPEL standard, the `boolean` values have to be transformed during the BPEL output generation. This example fixes the wrong display of the values of the `createInstance` attribute (see 2.6.1) in the `receive` activity.

Listing 119: BPEL output generation of boolean values

```
File: org.xmlsoap.schemas.ws.bpelEditor.xml.BpelBuilder.java
Function: buildReceiveSubtree

String createInstance = String.valueOf(receiveActivity.isCreateInstance());
if (createInstance.equals("true")) {
    createInstance = "yes";
}
else {
    createInstance = "no";
}
receiveElement.setAttribute(new Attribute("createInstance", createInstance));
```

Explanation:

The function `isCreateInstance` of the object `receiveActivity` provides the boolean data value of the attribute. The `receiveElement` object is the XML element prepared for output where `createInstance` is set as an attribute.

6.7.2. Dynamic link naming

In the prior version of the editor links were not given a name dynamically. This deficiency has been addressed using the following algorithm and implementation, respectively. The name returned by the link object is dynamically calculated according to the schema “<fromActivity>-to-<toActivity>” as long as it is not overwritten.

Listing 120: Dynamic link naming

```
File: org.xmlsoap.schemas.ws.bpelmodel.impl.TLinkImpl.java
Function: getName

if (!(target == null) && !(source == null) && ((name == "") || (name == null))) {
    return source.getName() + "-to-" + target.getName();
}
else {
    return name;
}
```

Explanation:

The variable `name` is the variable of the link object that holds the actual name of the link. The objects `target` and `source` are the constructs that this link connects. Both provide the function `getName` which is used for the dynamic link naming at initialization. After a user-defined link name has been set, the dynamic naming is inactive. After deleting the link name it is active again.



Figure 14: Dynamic link naming

6.7.3. Activity traversing at BPEL output generation

For some constructs the traversing of nested activities was not yet implemented. This has been resolved for the `scope` and `while` constructs. Without this traversing the generated BPEL output would not contain the nested activities or constructs. The following example shows the modifications for the `scope` construct.

Listing 121: Activity traversing for scope construct

File: org.xmlsoap.schemas.ws.bpeleditor.xml.BpelBuilder.java
Function: buildScopeSubtree

```
//Traversing the Subactivities
Element linksElement = null;
if (!scopeActivity.eContents().isEmpty()) {
    for (int i=0; i<scopeActivity.eContents().size(); i++) {
        if (scopeActivity.eContents().get(i) instanceof TActivity) {
            buildActivitySubtree(scopeElement,
                (TActivity)scopeActivity.eContents().get(i));
        }
        if (scopeActivity.eContents().get(i) instanceof TLink) {
            if (linksElement == null) {
                linksElement = new Element("links",ns);
                scopeElement.addContent(linksElement);
            }
            buildLinkSubtree(linksElement,
                (TLink)scopeActivity.eContents().get(i));
        }
    }
}
```

Explanation:

scopeActivity is a scope object in a BPEL model instance. Its function eContents provides a list of nested activities or constructs. For each contained activity or construct the according subtree builder function is called by buildActivitySubtree. The function buildLinkSubtree builds up the subtree for links from and to this scope.

6.7.4. Activity type display in the Property View

In the prior version of the editor an activity type could only be distinguished by its icon; this has been resolved by displaying the className without the preceding "T" in the name in the Property View. The Property View displayed only the string "Activity" before.

Listing 122: Display of activity type in the Property View

File: org.xmlsoap.schemas.ws.bpeleditor.model.EObjectPropertySource
Function: getGroupName
. . .
else if (className.equals("TReply") || className.equals("TReceive")) . . . {
 groupName = "Activity: " + className.substring(1, className.length());
}
return groupName;

Property	Value
[-] Activity: ExtensionActivity	
attributes	
elements	<ext:invokeApp param1="getSellers.exe"/>
joinCondition	
name	publishRequest
namespace	ext="http://www.extension.com/invokeApp"
suppressJoinFailure	
[-] View	
x	17
y	14

Figure 15: Activity type display in the Property View

7. Remaining BPEL 2.0 Extensions

Due to the high complexity of the generated code on the one hand and the time schedule on the other, additional modifications constructs and features can still be added to the tool. The following list of extensions and modifications describes the remaining differences from BPEL 1.1 to BPEL 2.0. Note that the tool was not yet entirely BPEL 1.1 compliant, and maybe the underlying BPEL 1.1 constructs that these modifications require are not yet implemented.

7.1. New constructs

Activities:

- `rethrow` in fault handlers (see 2.6.5)
- `repeatUntil` (see 2.7.3)
- `compensateScope` (see 2.8.4)
- default fault handlers (see 2.8.5)

Other:

- `import` on the construct `process` (see 2.2.2)
- Service reference containers (`<sref:service-ref>`) (see 2.3.4)
- In-line variable initialization (see 2.4.1)
- `fromParts` in receiving activities (see 2.4.1)
- `toParts` in sending activities (see 2.4.1)
- `extensionAssignOperation` on `assign` (see 2.4.2)
- `extensions` on the construct `process` (see 2.9)
- `abstractProcess` profiles (see 2.10)

7.2. New attributes

- `exitOnStandardFault` on the construct `process` (see 2.2.1)
- `expressionLanguage` on constructs that allow or require expressions (see 2.2.1)
- `initializePartnerRole` on `partnerLinks` (see 2.3.2)
- `element` on `propertyAlias` (see 2.4.1)
- `type` on `propertyAlias` (see 2.4.1)
- `validate` on `assign` (see 2.4.2)
- `keepSrcElementName` on `copy` in `assign` (see 2.4.2)
- `messageExchange` on `receive` (see 2.6.1)
- `isolated` on `scope` (see 2.8)
- `exitOnStandardFault` on `scope` (see 2.8)
- `repeatEvery` on `onAlarm` (see 2.8.6)
- `mustUnderstand` on `extensions` (see 2.9)
- `documentation` on all constructs (see 2.9)
- Custom namespaces on `process`

7.3. Removal of attributes

- `enableInstanceCompensation` on the construct `process` (see 2.2.1)

7.4. Attributes and nested XML elements

- `portType` on `partnerLinks` (see 2.3.1)
- `from` in `copy` (see 2.4.2)
- `to` in `copy` (see 2.4.2)
- activities nested in `onEvent` or `onAlarm` elements have to be wrapped in a scope (see 2.8.6)

7.5. Construct renaming

- `switch` and subelements are renamed into `if` and according subelements (see 2.7.1)
- `onMessage` is renamed into `onEvent` in events handlers (see 2.8.6)

8. Discussion and Outlook

8.1. Discussion

Modeling of business processes is gaining more and more importance within the framework of business process management. An essential factor for the successful modeling of processes is the use of tools, which offer sufficient functionality and permit user-friendly and intuitive interactions. The motivation of this thesis is to develop a modeling tool that supports graphical process modeling in BPEL 2.0. The tool is an extension of an existing modelling BPEL1.1 tool that did not completely support modelling of all language constructs.

On account of this, the specifications BPEL 1.1 and BPEL 2.0 were compared and the differences have been identified. The structure and the extensibility of the existing tool have been analyzed and necessary changes have partially been implemented, including the required integration of the `extensionActivity`. Also the possibilities for supporting BPMN have been discussed.

The analysis of this tool combined with the experience of how it can be adapted to the new standard raise a general discussion about the usage of code generators for the development of applications. The code, which has been generated with the Eclipse Modeling Framework (EMF) for this tool to model the BPEL 1.1 data model is very complex and hard to maintain manually. Much of the generated code is either redundant or unused and the whole code needs refactoring. Unfortunately, the generated code also does not implement any debugging or exception-handling functionality. These characteristics have been discovered, when the modifications on the data model for BPEL 2.0 compliance have been made. For example, the integration of one additional attribute required enormous effort whereas manual architecture could enormously reduce this complexity.

It was shown, that re-generation of the code for the data model is quite simple, but it has also become apparent that the controller is hard-coded with the data model and as well as with the visualization component - the Graphical Editing Framework (GEF). This does not allow exchanging the underlying data model completely.

One question that arises is whether one should give up maintainability in favor of development speed. Another question is how much effort may or can be invested into the development of a generic controller. Both questions are open research topics in the area of Model Driven Architecture (MDA).

8.2. Outlook

The Graphical Modeling Framework (GMF) provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF" [GMF06, "About"]. Yet GMF is not fully generic and with this framework much code is getting generated, too. The problem of implementing model changes for particular projects still needs to be investigated.

A far more complex approach would be the development of a completely generic editor. This should enable the creation and modification of arbitrary XML based languages like BPEL. Input for the editor could be a data model defined as XSD or XSI (XML Metadata Interchange). However, the data model would have to be extended by semantics.

For example, a link in BPEL needs to be described in the XSD in such a way, that it becomes clear to the editor, that the link connects two constructs. The idea about this approach emerged during the work on the graphical BPEL editor presented in this thesis. To the best of our knowledge, this approach has not been proposed or implemented yet.

The new technologies and unexplored approaches sound promising and so a decision on the further development of this tool has to be made in favor of one of the following possibilities:

The first option is to go on and accomplish the remaining BPEL 2.0 extensions that are listed in section 7 and use BPMN as described in 4.5.3.

The other option is the creation of an editor from scratch using a newer technology like GMF or a completely different framework like the SOA Tools Platform (STP), which provides a BPMN editing framework that has an unimplemented interface for BPEL export, as described in 4.5.3.

The investigation and implementation of a generic model driven editor could be a possible solution as well.

In conclusion, at the time the specification of BPEL 2.0 was completed, various Web Service standard works, such as from WSDL 1.1 [W3C01] to WSDL 2.0 and WS-Addressing [W3C04], were ongoing and not ready for consideration for BPEL 2.0. Future versions of BPEL 2.0 may provide support for these standards [BPEL2.0, p.12]. Also the BPMN 2.0 standard works [BPMN2.0] are in progress and may provide support for BPEL 2.0. A BPEL modeling tool must be extended with such support as well.

Appendices

References

[ACKM04]

Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju:
Web Services – Concepts, Architectures and Applications
Springer, 2004

[BPEL1.0]

Francisco Burbera, Yaron Goland, Johannes Klein, Frank Leymann, Dieter Roller, Satish Thatte, Sanjiva Weerawarana:
Business Process Execution Language for Web Services Version 1.0
July 2002
<ftp://www6.software.ibm.com/software/developer/library/BPEL.2.01.pdf>

[BPEL1.1]

Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana:
Business Process Execution Language for Web Services Version 1.1
May 2003
<http://www-128.ibm.com/developerworks/library/specification/BPEL.2.0/>

[BPEL2.0]

Alexandre Alves, Assaf Arkin, Sid Askary, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri and Alex Yiu:
OASIS Web Services Business Process Execution Language Version 2.0, Comitee Specification
January 2007
<http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.pdf>

[BPEL4People]

Ashish Agrawal, Mike Amend, Manoj Das, Mark Ford, Chris Keller, Matthias Kloppmann, Dieter König, Frank Leymann, Ralf Müller, Oracle Gerhard Pfau, Karsten Plösser, Ravi Rangaswamy, Alan Rickayzen, Michael Rowley, Patrick Schmidt, Ivana Trickovic, Alex Yiu, Matthias Zeller:
BPEL Extension for People Version 1.0
June 2007
<http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/>

[BPML]

Assaf, Arkin:
Business process modeling language 1.0
June 2002
http://www.omg.org/technology/documents/br_pm_spec_catalog.htm

[BPMN1.0]

Object Management Group:
Business Process Model and Notation 1.0, Final Adopted Specification
February 2006
<http://www.bpmn.org/Documents/BPMN.2-0.RFP.07-06-05.pdf>

[BPMN2.0]

Object Management Group:

Business Process Model and Notation (BPMN) 2.0 Request For Proposal

June 2006

[http://www.bpmn.org/Documents/BPMN 2-0 RFP 07-06-05.pdf](http://www.bpmn.org/Documents/BPMN%202-0%20RFP%2007-06-05.pdf)

[eClarus]

eClarus Business Process Modeler for SOA Architects

2005 - 2006

<http://www.eclarus.com/pdf/DS-SOA-05-06-v1-0.pdf>

[Evde06]

John Evdemon:

What's new in BPEL 2.0

August 2006

www.oasis-open.org/committees/download.php/20266/whats_new_in_bpel_2.0.ppt

[Gao06]

Yi Gao:

BPMN-BPEL Transformation and Round Trip Engineering

May 2006

http://www.eclarus.com/pdf/BPMN_BPEL_Mapping.pdf

[GMF06]

Eclipse Graphical Modeling Framework (GMF)

June 2007

<http://www.eclipse.org/gmf>

[inno05]

innoQ:

Web Service Standards overview

September 2005

<http://www.innoq.com/soa/ws-standards/poster/>

[Kapl06]

Institut für Architektur von Anwendungssystemen, Michael Kaplan:

Graphisches BPEL Modellierungstool für parametrisierte Prozesse und Templates

Diploma thesis, No. 2439,

June 2006

[KCHK04]

Martin Keen, Jonathan Cavell, Sarah Hill, Chee Keong Kee, Wendy Neave, Bradley Rumph, Hoang Tran:

BPEL4WS Business Processes with WebSphere Business Integration:

Understanding, Modeling, Migrating

December 2004

<http://www.redbooks.ibm.com/abstracts/sg246381.html>

[Lee03]

Daniel Lee:

Display a UML Diagram using Draw2D

August 2004

<http://www.eclipse.org/articles/Article-GEF-Draw2d/GEF-Draw2d.html>

[LeRo00]

Frank Leymann, Dieter Roller:
Production Workflow - Concepts and Techniques
Prentice Hall, 2000

[MDGW04]

Bill Moore, David Dean, Anna Gerber, Gunnar Wagenknecht, Philippe Vanderheyden:
Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework
February 2004
<http://www.redbooks.ibm.com/abstracts/sg246302.html>

[RFC2119]

Scott Bradner:
Key words for use in RFCs to Indicate Requirement Levels, RFC 2119
Harvard University, March 1997
<http://www.ietf.org/rfc/rfc2119.txt>

[STP07]

SOA Tools Platform Project (STP):
STP BPMN
2007
<http://www.eclipse.org/stp/bpmn/>

[WeCu06]

Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, Donald F. Ferguson:
Web Services Platform Architecture
Prentice Hall, 2006

[Whit05]

Stephen A. White:
Using BPMN to Model a BPEL Process
February 2005
<http://www.bpmn.org/Documents/Mapping BPMN to BPEL Example.pdf>

[W3C01]

W3C:
Web Services Description Language (WSDL) 1.1
March 2001
<http://www.w3.org/TR/wsdl>

[W3C04]

W3C:
Web Services Addressing (WS-Addressing) Member Submission
August 2004
<http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/>

Used Resources

[BPELValidator]

Active Endpoints OnDemand BPEL 2.0 Validation

Version: August 2007

Website: http://www.activebpel.org/BPEL_Validator/

[BPEL1.1XSD]

XML Schema Definitions by Mircosoft, IBM, BEA, SAP and Siebel

March 2003

<http://schemas.xmlsoap.org/ws/2003/03/business-process/>

[BPEL2.0XSD]

XML Schema Definitions by OASIS for executable BPEL processes

April 2007

http://docs.oasis-open.org/wsbpel/2.0/OS/process/executable/ws-bpel_executable.xsd

[Eclipse]

Eclipse Development Platform

Version: 3.1.1

<http://www.eclipse.org/>

[EMF]

Extended Modeling Framework

Version: 2.1.1

<http://www.eclipse.org/modeling/emf/>

[GEF]

Graphical Editing Framework

Version: 3.1.1

<http://www.eclipse.org/gef/>

[J2SDK]

Java 2 Software Development Kit

Version: 1.4.2 release 14

<http://java.sun.com/j2se/1.4.2/download.html>

[Jigloo]

Jigloo Java UI Builder

Version: 3.9.5

<http://www.cloudgarden.com/jigloo/>

[Kdiff3]

Kdiff3 File Comparison

Version: 0.9.92

<http://kdiff3.sourceforge.net/>

[XMLMarker]

XML Editor

Version 1.1

<http://symbolclick.com/>

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und nur die angegebenen Hilfsmittel verwendet habe.

(David Schumm)