Date: July 2014

UML Profile for BPMN Processes

Version 1.0

OMG Document Number: formal/2014-07-01
Standard document URL: http://www.omg.org/spec/BPMNProfile/1.0
Machine Consumable Files:
Normative:
http://www.omg.org/spec/BPMNProfile/20121112/BPMNProfile.xmi
http://www.omg.org/spec/BPMNProfile/20121112/BPMNToUMLProfile.qvt
http://www.omg.org/spec/BPMNProfile/20121112/UMLProfileToBPMN.qvt
http://www.omg.org/spec/BPMNProfile/20121112/BPMNToUMLProfile.xslt
http://www.omg.org/spec/BPMNProfile/20121112/UMLProfileToBPMN.xslt

---

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE

MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE.
IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 109 Highland Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

IMM®, MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , MOF™ , OMG Interface Definition Language (IDL)™ , and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (*http://www.omg.org/report_issue.htm*).

Table of Contents

Preface iii
1 Scope 1
2 Conformance 2
3 References 2
3.1 Normative 2
3.2 Non-normative 2
4 Terms and Definitions 3
5 Symbols and Acronyms 3
6 Additional Information 3
6.1 Changes to Adopted OMG specifications 3
6.2 Acknowledgments 3
7 Conventions 5
8 Other Material 7
8.1 Derived Properties 7
8.2 Transformations 7
9 Core Structure 9
9.1 Foundation and Infrastructure 9
9.1.1 Base Elements 9
9.1.2 Definitions 9
9.1.3 Extensions 10
9.1.4 External Relationships 10
9.1.5 Derived Properties and Constraints 11
9.2 Common Elements 13
9.2.1 Expressions 13
9.2.2 Artifacts 14
9.2.3 Derived Properties and Constraints 16
10 Processes 19
10.1 Processes and Global Tasks 19
10.1.1 Flow Elements 19
10.1.2 Processes and Global Tasks 19
10.1.3 Derived Properties and Constraints 21
10.2 Activities 23

10.2.1 Illustration 23
10.2.2 Derived Properties and Constraints 25
10.3 Sequence Flows 27
10.3.1 Illustration 27
10.3.2 Derived Properties and Constraints 27
10.4 Events 28
10.4.1 Events and Event Definitions 28
10.4.2 Throw Events 30
10.4.3 Catch Events 33
10.4.4 Derived Properties and Constraints 35
10.5 Gateways 38
10.5.1 Parallel and Exclusive Gateways 39
10.5.2 Event-based Gateways 39
10.5.3 Complex Gateways 40
10.5.4 Inclusive Gateways 40
10.5.5 Derived Properties and Constraints 41
10.6 Data 41
10.6.1 Item Aware Elements and Data Associations 41
10.6.2 Input Output Specifications and Data Objects 42
10.6.3 Input Sets and Output Sets 43
10.6.4 Properties as Item Aware Elements 44
10.6.5 Data Stores 44
10.6.6 Derived Properties and Constraints 45
10.7 LoopCharacteristics 49
10.7.1 Derived Properties and Constraints 51
10.8 Lanes and Resources 52
10.8.1 Derived Properties and Constraints 54
11 Collaborations 57
11.1 Collaborations and Conversations 57
11.1.1 Collaborations 57
11.1.2 Participants 57
11.1.3 Message Flows 58
11.1.4 Conversations 59
11.1.5 Derived Properties and Constraints 61
11.2 Operations, Interfaces, and Related Tasks 64
11.2.1 Operations and Interfaces 64
11.2.2 Related Tasks 66
11.2.3 Derived Properties and Constraints 67
11.3 Correlation 68
11.3.1 Derived Properties and Constraints 70

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from this URL:

http://www.omg.org/spec

Specifications are organized by the following categories:

Business Modeling Specifications

Middleware Specifications

- **CORBA/IIOP**
- **Data Distribution Services**
- **Specialized CORBA**

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

- **UML, MOF, CWM, XMI**
- **UML Profile**

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

- **CORBAServices**
- **CORBAFacilities**

OMG Domain Specifications

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

Signal and Image Processing

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Avenue
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/report_issue.htm*.

1 Scope

This specification enables modelers to use BPMN 2 process and collaboration notation as a concrete syntax for UML activity and collaboration models. It extends the UML metamodel with a UML profile, including extensions of UML semantics to ensure equivalence to BPMN semantics. Equivalent semantics ensures businesses following BPMN process or collaboration diagrams will function the same way whether the diagrams are captured using the BPMN metamodel or the profiled UML metamodel, as illustrated in Figure 1.1. The scope of the profile includes elements and relationships in BPMN in the Process Modeling and Process Execution conformance types, including the Descriptive, Analytic, and Common Executable conformance sub-classes, see Clause 2. It does not address other conformance types. The specification does not change BPMN notation, metamodel, or semantics.

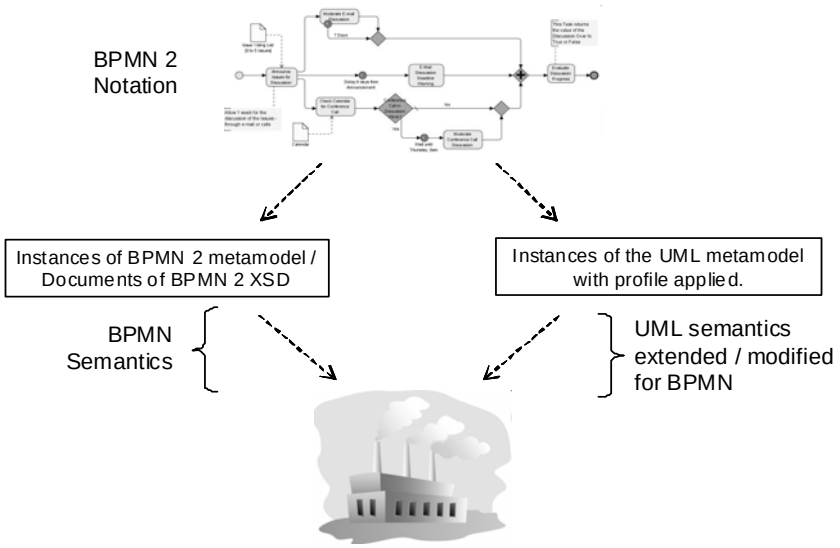

Figure 1.1 - Equivalent Semantics

The profile acts as a mapping between concepts in the BPMN and UML specifications, but BPMN and UML are only described as needed to explain the mapping. Any differences between the descriptions of BPMN and UML in this specification and the BPMN or UML specifications is unintentional, and should be taken as errors in this specification, rather than changing BPMN or UML. Stereotypes give the syntactic portion of the mapping, which carries along the semantic portion when UML and BPMN semantics are equivalent, and is only explained further as needed for clarity. When UML and BPMN semantics are different, it is explained how applying the profile extends UML semantics to be equivalent to BPMN semantics. Applying the profile does not change the semantics of BPMN or UML, only of elements in the particular UML model to which the profile is applied. Any differences in semantics between profiled model elements and BPMN semantics is unintentional, and should be taken as errors in this specification, rather than changing BPMN.

Some BPMN concepts have semantics in their application, for example, manual tasks and public processes. These constrain how a modeler applies the concepts to their domain, which BPMN captures in the name of the concept and an informal textual description. The profile includes these concepts with the same names as BPMN. The application semantics is available in the BPMN specification, it is not restated here. The same applies to BPMN properties that have their semantics in implementation. These constrain how the model is implemented, which BPMN captures sometimes

with predefined values and informal textual descriptions. The profile includes these properties with the same names as BPMN, and the same predefined default values, if any. Other predefined values and implementation semantics are available in the BPMN specification, they are not restated here.

The stereotypes in the profile carry the abstract syntax constraints of BPMN, which are not repeated here. For example, BPMN constrains the number of sequence flows going out of event-based gateways, as well as the combined values of the processType and isExecutable properties on processes, and the relationships between events and event definitions. Constraints corresponding to those in BPMN apply to UML models when the profile is applied. Checking BPMN constraints in profiled models is facilitated by derived stereotype properties corresponding to BPMN properties. The derived properties also enable model queries using BPMN property names, and provide a portion of the mapping between BPMN and UML properties. Properties with the same name and type in BPMN and UML are not included in derived properties.

2 Conformance

This profile has conformance points corresponding to the BPMN Process Modeling conformance type, including its Descriptive, Analytic, and Common Executable sub-classes, and the BPMN Process Execution conformance type. The profile's conformance points are:

- Process Modeling and subpoints Descriptive, Analytic, and Common Executable require support for applying and unapplying stereotypes corresponding to the BPMN elements required by the corresponding BPMN Process Modeling conformance type and it Descriptive, Analytic, and Common Executable subclasses (see BPMN for enumeration of these elements). This conformance point also requires import and export of profiled models that are executed.
- Process Execution requires execution of profiled models according to the semantics specified for the stereotypes, which is based on UML's and equivalent to BPMN's, see Clause 1. This conformance point also requires import of profiled models that are executed.

The profile's Process Modeling and Process Execution conformance points are independent. Each may be supported without the other, or they may be supported together.

3 References

3.1 Normative

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply:

- BPMN 2.0 Specification (http://www.omg.org/spec/BPMN/2.0/)
- UML 2.4.1 Specification (http://www.omg.org/spec/UML/2.4.1/)
- QVT 1.1 Specification (http://www.omg.org/spec/QVT/1.1/)
- XML Schema 1.0 (http://www.w3.org/XML/Schema/)

3.2 Non-normative

There are no non-normative documents referenced in the specification.

4 Terms and Definitions

There are no new terms defined in this specification.

5 Symbols and Acronyms

There are no symbols defined in this specification.

6 Additional Information

6.1 Changes to Adopted OMG specifications

This specification does not change any other OMG specification.

6.2 Acknowledgments

The following companies submitted this specification:

- META International
- Model Driven Solutions
- No Magic
- Sparx Systems

The following companies supported this specification:

- Atego
- Axway
- Computer Sciences Corporation
- Oose Innovative Informatik GmbH
- Softeam
- The MITRE Corporation
- U.S. Department of Defense
- U.S. National Aeronautics and Space Administration
- U.S. National Institute of Standards and Technology

7 Conventions

In models:

- Rectangles in the figures labeled “metaclass” notate elements from the UML metamodel, while those labeled with “stereotype” and “enumeration” notate elements of the profile.
- Primitive types, such as Boolean and String, are used from UML without special labels.
- The prefix “BPMN” is used for stereotype names when there is a clash with UML names. Enumeration literal names are uncapitalized, per UML convention, even when they are capitalized in BPMN.

In paragraph text:

- The term “BPMN” used as a noun in this specification refers to the BPMN specification. The term “BPMN” used as an adjective, as in “BPMN Activity,” refers the concept in BPMN, rather than the stereotype in this profile.
- Metaclass names are capitalized when preceded by their language, as in “UML Activities,” otherwise they are not capitalized (unless at the beginning of a sentrence) and spaces are inserted between words in the name, as in “catch event.”
- Stereotype names are followed by “stereotype” or “applied.” They are capitalized, except when using the name of an abstract stereotype to refer to its concrete specializations, in which case they are not capitalized (unless at the beginning of a sentrence) and spaces are inserted between words in the pluralized name, as in “throw event stereotypes.”
- Metaclass and stereotype names that are pluralized or preceded by “a(n)” or “the” refer to instances of the metaclasses and stereotypes.
- Property names are spelled and capitalized as they appear in their metaclasses or stereotypes, as in “textFormat,” and are always used as an adjective for “property,” as in “the value of the textFormat property.”

See 10.4.1 about the convention for referring to events by their event definitions.

In natural language text in derived properties and constraints the above rules apply, except:

- Metaclass names are not preceded with “UML” and stereotype names are not followed by “stereotype.” Only UML metaclass and stereotype names are used derived properties and constraints, not BPMN metaclass names.
- Metaclass and stereotype names are capitalized when they are singular and not preceded by “a(n)” or “the,” and when they are at the beginning of a sentence, otherwise they are not capitalized and spaces are inserted between words in the name.
- Property names are used as nouns to refer to their values.

8 Other Material

8.1 Derived Properties

The profile includes derived stereotype properties corresponding to BPMN properties that can be derived from UML properties. They support model queries based on BPMN property names, enabling a profiled UML model to respond in the same way as a BPMN model. Derived properties are specified in separate sub clauses of each clause. Stereotype figures do not show derived BPMN properties, only properties that are added to UML to reduce clutter in the stereotype diagrams. Derived properties are only defined when the corresponding BPMN and UML properties are different in some respect.

8.2 Transformations

This document provides mappings between BPMN models and UML models with the profile applied, expressed in stereotypes, properties and associations between stereotypes, and natural language descriptions of these. The mappings are formalized as transformations in both directions defined in separate machine-readable artifacts using QVT Relational and XML Schema (XSD). The transformations from BPMN to UML accept as input conforming instances of the BPMN metamodel, or an XML document conforming to the BPMN XSD, and this profile then creates conforming instances of the UML metamodel with stereotypes applied to them. The transformations from UML to BPMN accept as input conforming instances of the UML metamodel with stereotypes applied to them, and this profile then creates conforming instances of the BPMN metamodel, or an XML document conforming to the BPMN XSD. The source BPMN models must be syntactically correct, and source UML elements with stereotypes applied must obey the corresponding abstract syntax constraints of BPMN, see Clause 1. The BPMN metamodel, BPMN XSD, UML metamodel, and the profile are not modified by these transformations.

9 Core Structure

9.1 Foundation and Infrastructure

9.1.1 Base Elements

Figure 9.1 illustrates BPMN Base Elements.

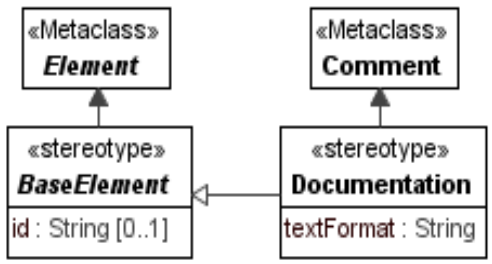

Figure 9.1 - Base Elements

BPMN BaseElement generalizes almost all other elements in BPMN, while UML Element generalizes all other elements in UML. They both can have user-specified text, and Documentation includes specification of mime format using the textFormat property. BPMN BaseElement is also used for extensibility, see 9.1.4.

9.1.2 Definitions

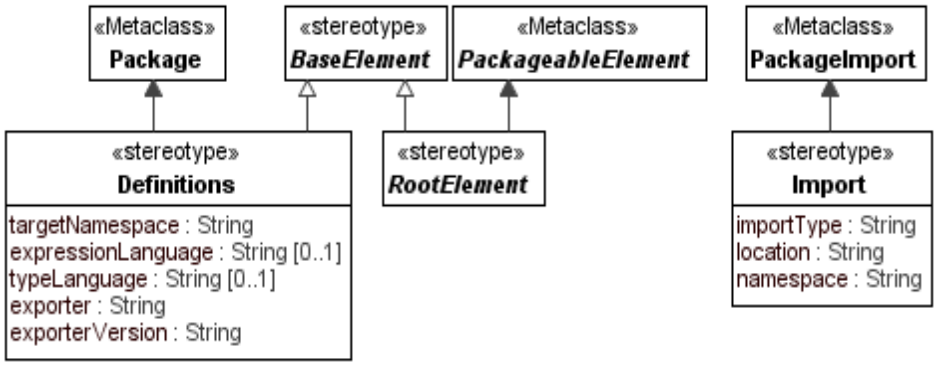

Figure 9.2 - Definitions

BPMN Definitions contain models, but only to organize them, rather than to specify anything about how businesses function based on those models. UML Packages have the same purpose. The targetNamespace and typeLanguage properties of BPMN Definitions do not have meaning in UML, but are included in the profile to retain information from BPMN tools not based on UML, and generating platform-specific languages from UML if applicable. The expressionLanguage property of BPMN Definitions is used as the language for expressions, when no other is specified, see FormalExpression in 9.2.1. The exporter and exporterVersion properties of BPMN Definitions give information about

the tool that generated the definitions model. BPMN RootElement generalizes all elements that can be contained by definitions. UML PackagedElement similarly generalizes all elements that can be contained by packages. BPMN Import uses strings to identify external models to be brought into definitions. UML PackageImport has the same purpose. The importType, location, and namespace properties of BPMN Import do not have meaning in UML, but are included in the profile to retain information from BPMN tools not based on UML, and generating platform-specific languages from UML if applicable.

9.1.3 Extensions

Figure 9.3 illustrates BPMN Extensions.

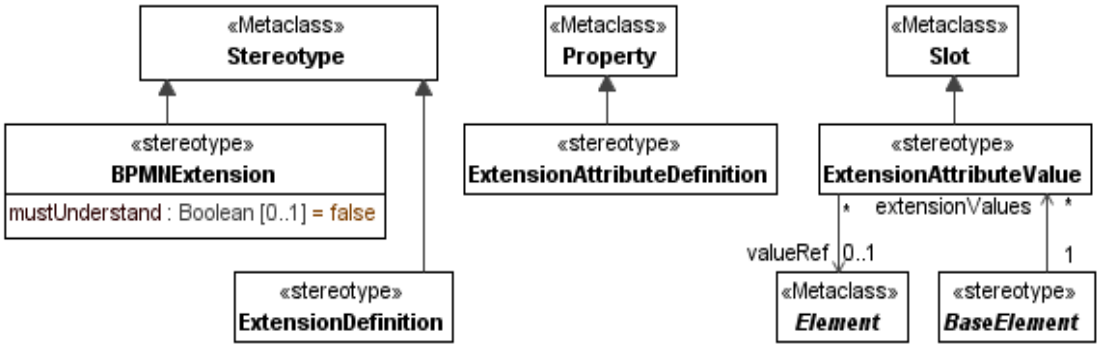

Figure 9.3 - BPMN Extensions

BPMN Extension and BPMN ExtensionDefinition enable modelers to define properties with BPMN ExtensionAttributeDefinitions that can be given values for elements in their models. Extension definitions can be shared by model elements. UML Stereotypes have the same purpose, and can be restricted to apply only to certain kinds of model elements in the UML language. UML Stereotypes are classes instantiated when they are applied to elements in user models, where the instances have values for the properties defined by the stereotype. UML does not capture instances of classes directly, but can specify instances with instance specifications, which contain slots specifying values for the properties defined in the classes. The BaseElement stereotype has an extensionValues property for referring to ExtensionAttributeValues stereotypes that are applied to slots. The slots are owned by an instance specification describing instances of the stereotypes applied to the same user model element the BaseElement stereotype is applied to. The slots can use UML ValueSpecification to specify values for properties, or the ExtensionAttributeValue stereotype applied to the slots can specify values with the valueRef property by referring to elements in the user model or other models. The ExtensionAttributeValue stereotype requires the instances of the applied stereotypes to conform to these instance specifications, giving semantics equivalent to BPMN's. The mustUnderstand property of BPMN Extension has implementation semantics, see the BPMN specification.

9.1.4 External Relationships

Figure 9.4 illustrates External Relationships.

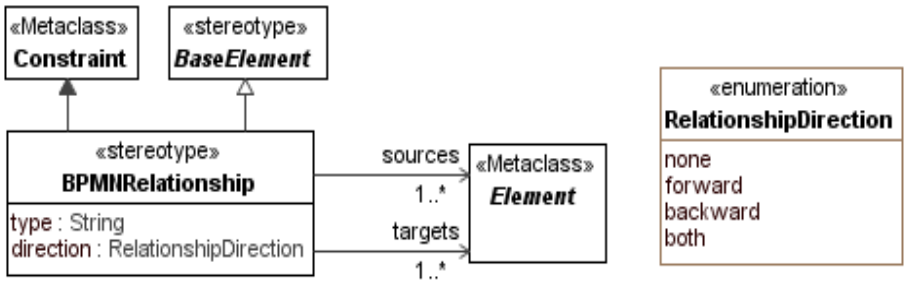

Figure 9.4 - External Relationships

BPMN Relationships link to BPMN model elements with elements from other models, including ones not defined with BPMN. The profiles support this with a BPMNRelationship stereotype extending UML Constraint to refer to the related elements, and the type and direction properties to specify more about the relationship, giving semantics equivalent to BPMN's.

9.1.5 Derived Properties and Constraints

9.1.5.1 BaseElement

Derived properties

- /documentation : Documentation [*] = self.base_Element.ownedComment.extension_ Documentation
- /extensionDefinitions : ExtensionDefinition [*] = self.base_Element.extension_ anystereotype.extension_ExtensionDefinition
- /incoming : BPMNAssociation [*] = self.base_BaseElement.supplierDependency. extension_BPMNAssociation
- /outgoing : BPMNAssociation [*] = self.base_BaseElement.clientDependency. extension_BPMNAssociation

Constraints

[1] The extensionValues of a base element must be applied to slots that are owned by an instance specification with a classifier that is one of the stereotypes with ExtensionDefinition applied that are the extensionDefinitions of the base element.

9.1.5.2 BPMNExtension

Derived properties

- /definition : ExtensionDefinition = self.base_Stereotype.extension_ ExtensionDefinition

Constraints

[1] BPMNExtension may only be applied to elements with ExtensionDefinition applied, and vice versa.

[2] Stereotypes with BPMNExtension applied are based on Element.

[3] Stereotypes with BPMNExtension applied are owned by packages with Definitions applied.

9.1.5.3 BPMNRelationship

Derived properties

- /definition : Definition = self.base_Constraint.context.extension_Definitions

Constraints

[1] Constraints with BPMNRelationship applied must be owned by packages with Definitions applied.

[2] Constraints with BPMNRelationship applied must have constrainedElements that are the same as the union of the sources and targets of the applied BPMNRelationship.

9.1.5.4 Definitions

Derived properties

- /extensions : Extension [*] = self.base_Package./ownedStereotype.extension_Extension
- /imports : Import [*] = self.base_Package.packageImport.extension_Import
- /relationships : BPMNRelationship [*] = self.base_ Package./ownedRule.extension_ BPMNRelationship
- /rootElements : RootElement [*] = self.base_Package.packagedElement.extension_ RootElement

Constraints

None

9.1.5.5 Documentation

Derived properties

- /text : String = self.base_Comment.body

Constraints

None

9.1.5.6 ExtensionAttributeDefinition

Derived properties

- /isReference : Boolean = (self.base_Property./isComposite = false)
- /type : String = self.base_Property.type./qualifiedName

Constraints

None

9.1.5.7 ExtensionAttributeValue

Derived properties

- /extensionAttributeDefinition : ExtensionAttributeDefinition = self.base_Slot.owningInstance. classifier.extension_ExtensionAttributeDefinition

- /value : Element [0..1] = self.base_Slot.value

Constraints

[1] Slots with ExtensionAttributeValue applied are owned by instance specifications that are owned by the package containing the element having the applied ExtensionAttributeValue instance as its extensionValues.

9.1.5.8 xtensionDefinition

Derived properties

- /extensionAttributeDefinitions : ExtensionAttributeDefinition [*] = self.base_Steterotype. ownedAttribute.extension_ExtensionAttributeDefinition

Constraints

None

9.1.5.9 Import

Derived properties

- /definition : Definitions = self.base_PackageImport.importingNamespace.extension_Definitions

Constraints

[1] PackageImports with Import applied must be owned by packages with Definitions applied.

9.1.5.10 RootElement

Derived properties

- /definition : Definitions [0..1] = self.base_PackageableElement.owningPackage. extension_Definitions

Constraints

None

9.2 Common Elements

9.2.1 Expressions

Figure 9.5 illustrates BPMN Expressions.

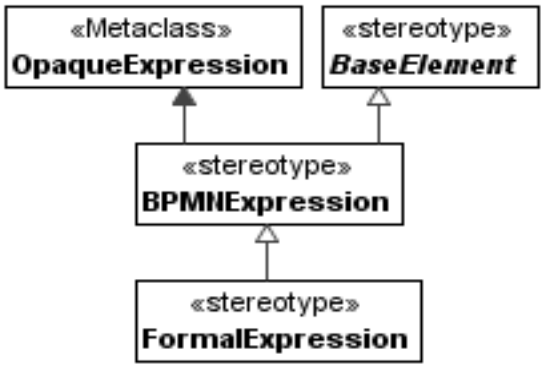

Figure 9.5 - BPMN Expressions

BPMN FormalExpression and UML OpaqueExpression have properties of the same name, but the UML properties have unlimited upper multiplicity, and the body property is limited to strings, rather than general elements as in BPMN. BPMN Expression is intended for natural language expressions, but does not provide properties to capture this. The profile uses the properties of UML OpaqueExpression for both the BPMNExpression and FormalExpression stereotypes.

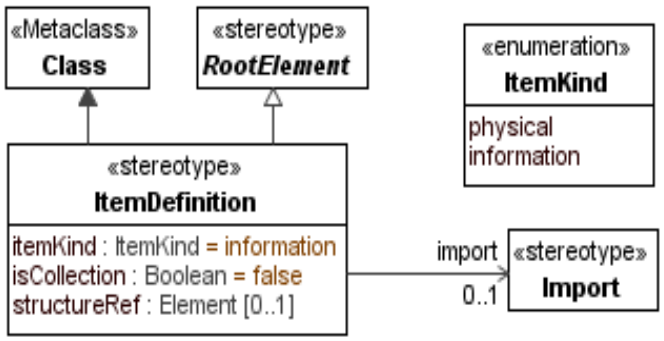

Figure 9.6 - Item Definitions

BPMN ItemDefinition refers out of BPMN to specify structure, while UML Classes specify structure. The ItemDefinition stereotype applies to UML Classes, rather than referring to them, providing an appropriate base for the stereotype, better integration with the rest of UML, and a semantics equivalent to BPMN's. The structureRef property is included in the profile to retain information from BPMN tools not based on UML, and to generate platform-specific languages from UML if applicable. The BPMN isCollection property tells whether the underlying data type is a collection, but the UML metamodel does not have collection types, so classes with the ItemDefinition stereotype applied and a value of true for the isCollection property represent collections of instances of the class. BPMN ItemDefinition has an import property identifying a BPMN Import (see 9.1.2) specifying where the item definition is imported from, if it is imported, and the profile extends classes to support this.

9.2.2 Artifacts

Figure 9.7 illustrates Artifacts.

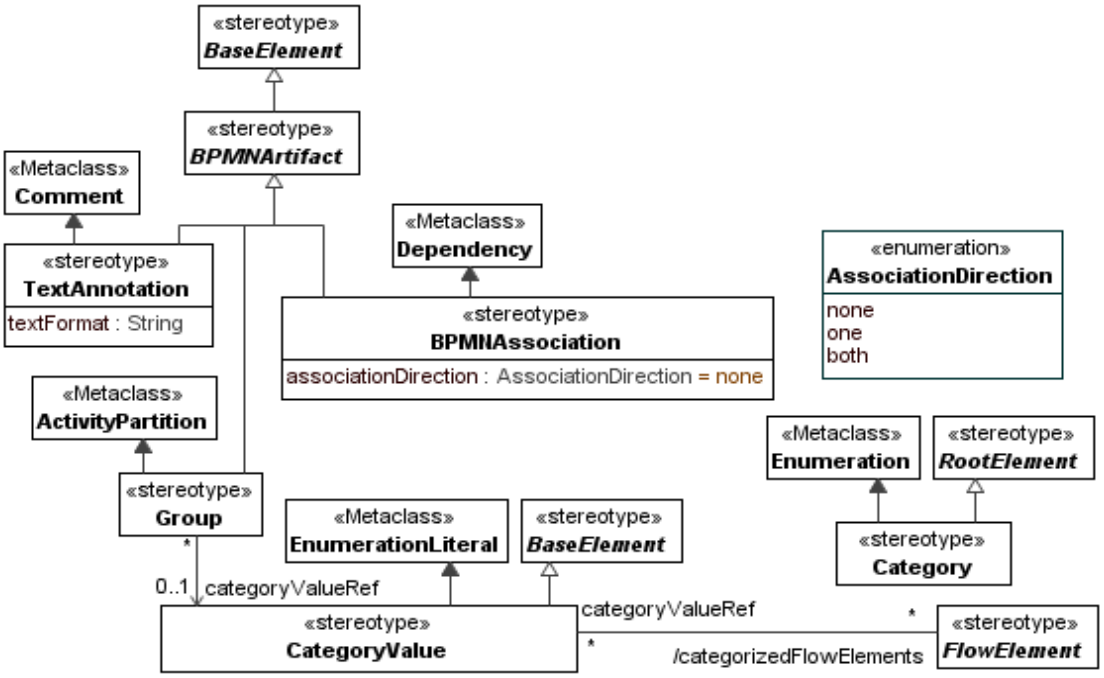

Figure 9.7 - Artifacts

BPMN Artifacts are miscellaneous elements that can be included in processes, subprocesses, and collaborations (see the /artifacts derived property on the corresponding stereotypes). BPMN gives additional syntax and semantics for the various kinds of artifacts.

BPMN Associations are artifacts that link elements in BPMN models to provide semantics that varies with the kinds of elements being connected. UML Dependencies indicate that some elements require others to be completely specified, which is general enough to cover BPMN Associations. BPMN Associations can indicate whether they are unidirectional, bidirectional, or have no direction. UML Dependencies are unidirectional from the elements that require others to those others, which is how unidirectional BPMN Associations are used. UML Dependency direction can be ignored for bidirectional and nondirectional BPMN Associations. The BPMNAssociation stereotype extends UML Dependencies with a property for specifying BPMN AssociationDirection. BPMN TextAnnotations are modeler-provided comments on elements identified by BPMN Associations to those elements, in a format specified by the textFormat property. UML Comments refer directly to the elements they describe, and do not specify a format. The TextAnnotation stereotype extends UML Comments with a property for text format.

BPMN Groups are artifacts that visually identify elements of processes for modeler-defined purposes. They are visually similar to BPMN Lanes in this respect, see 10.8, but in the underlying model identifies these elements by their relationships to categories, rather than directly as lanes do. BPMN Categories have BPMN CategoryValues, which process elements and groups can be related to. BPMN Groups visually identify the process elements that include the category values of the groups. BPMN Groups can also include elements from multiple processes for participants in the same collaboration, whereas BPMN Lanes can only include elements from one process, see 11.1 about participants and collaborations. UML ActivityPartitions identify activity elements for modeler-defined purposes, and the Group stereotype extends them to identify UML EnumerationLiterals with the CategoryValue stereotype applied. These literals are specified in UML Enumerations with the Category stereotype applied, providing equivalent abstract syntax as BPMN (the semantics in UML and BPMN are both modeler-defined).

9.2.3 Derived Properties and Constraints

9.2.3.1 BPMNAssociation

Derived properties

- /sourceRef : BaseElement = self.base_Dependency.client.extension_BaseElement
- /targetRef : BaseElement = self.base_Dependency.supplier.extension_BaseElement

Constraints

[1] Dependencies with BPMNAssociation applied must be owned by the innermost package containing the suppliers and clients of the dependencies.

9.2.3.2 BPMNExpression

Derived properties

- /assignment = inverse of Assignment::/to and Assignment::/from.

Constraints

None

9.2.3.3 Category

Derived properties

- /categoryValue : CategoryValue [*] = base_Enumeration.ownedLiteral.extension_ CategoryValue

Constraints

None

9.2.3.4 CategoryValue

Derived properties

- /categorizedFlowElements : FlowElement [*] = self.opposite(categoryValueRef). base_ActivityPartition.node&edge.extension_FlowElement

Constraints

None

9.2.3.5 FormalExpression

Derived properties

- /evaluatestoTypeRef : ItemDefinition = self.base_OpaqueExpression.type. extension_ItemDefinition (type defined on TypedElement).

Constraints

None

9.2.3.6 Group

Derived properties

Constraints

[1] Activity partitions with Group applied do not represent anything.

[2] Group may only be applied to activity partitions that have no subpartitions and that are direct partitions of activities that have BPMNProcess applied.

9.2.3.7 ItemDefinition

Derived properties

None

Constraints

None

9.2.3.8 TextAnnotation

Derived properties

- /text : String = self.base_Comment.body

Constraints

[1] Comments with TextAnnotation applied are ownedComments of an element with BPMNProcess, SubProcess, or BPMNCollaboration applied that has the text annotation included in its artifacts.

10 Processes

10.1 Processes and Global Tasks

10.1.1 Flow Elements

Figure 10.1 illustrates BPMN Flow Elements.

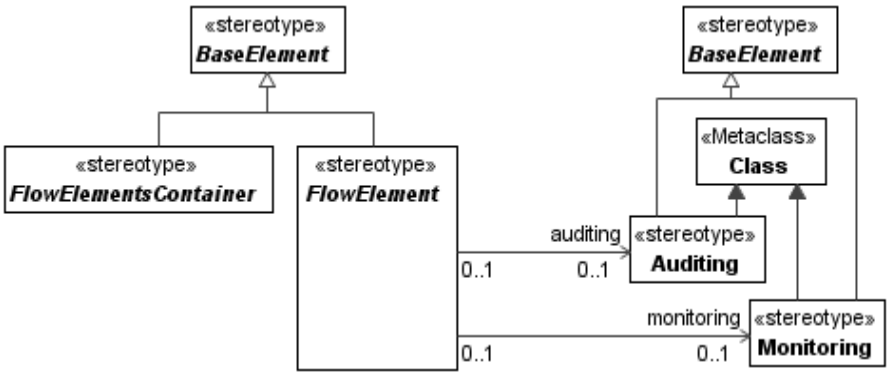

Figure 10.1 - FlowElements

BPMN Processes have a single abstraction for all of things they contain, while UML does not. UML activity elements are divided into nodes and edges. The abstract FlowElement stereotype applies to both nodes and edges, and types the / flowElements derived property of the BPMNProcess stereotype. BPMN FlowElementsContainers are an abstraction including contents of choreographies, which are not included in the profile. The FlowElementsContainer stereotype is included to support derived properties.

BPMN Auditing and Monitoring are for examining process instances and other execution instances that have occurred (auditing) or are occurring (monitoring). See 10.1.2 about process instances and 10.2 and 10.4 about other execution instances. The Auditing and Monitoring stereotypes extend UML Classes for modelers to specify operations and properties for auditing and monitoring execution instances. The instances of auditing and monitoring classes can be related to execution instances, or be the same as the execution instances, as determined by the modeler and implementation. For example, auditing and monitoring classes might generalize UML Activities with the BPMNProcess stereotype applied, providing operations and properties on each activity instance, or they might classify activity instances, which has the same effect, or they might have instances separate from activity instances but related to them by modeler or implementation defined links.

10.1.2 Processes and Global Tasks

Figure 10.2 illustrates BPMN processes and global tasks.

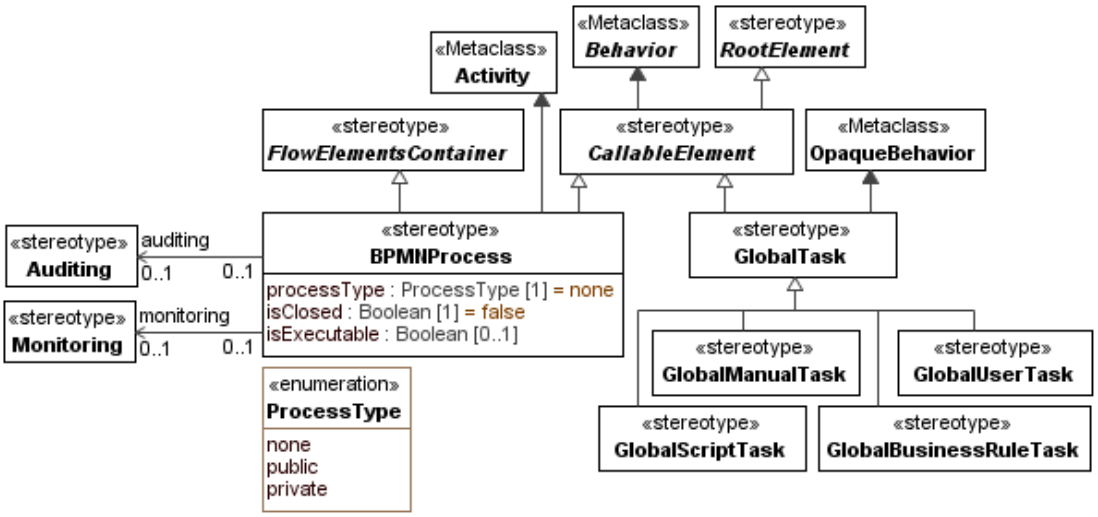

Figure 10.2 - Processes and Global Tasks

A BPMN process instance is a single occurrence of a process model being carried out, whether manually, semi-automatically, or executed fully automatically. Valid process instances follow their process model, invalid ones do not. Similarly, UML Activities are special kinds of classes, with instances being valid occurrences of activities as they are carried out, as BPMN process instances are. BPMN Processes specify properties that hold items of specified kinds in process instances, see 10.6.4. See 10.4.3.1 and 11.2.2 about when process instances are started, and 10.4.2.3 about when process instances are terminated.

The BPMN isClosed property of BPMN Processes indicates whether valid process instances may carry out interactions, such as receiving messages and events, that are not specified in their models. If the value of the isClosed property is true, then process instances are limited to interactions specified in their models, otherwise they are not. The BPMNProcess stereotype extends UML Activity semantics to use the isClosed property to determine whether valid activity instances may carry out interactions, such as receiving operations and other events, that are not specified in their models, giving a semantics equivalent to BPMN's. The BPMN supports property between process models indicates when all occurrences of one process model are intended to be valid for another. UML provides equivalent semantics with generalization between activities, as expressed by the /supports derived property of the BPMNProcess stereotype. One UML Activity generalizing another means occurrences (instances) of the specialized activity are occurrences of the general activity.

The BPMN processType property of BPMN Processes has application semantics, see the BPMN specification.

The BPMN isExecutable property of BPMN Processes indicates whether the modeler believes a private process is specified completely enough to support BPMN execution semantics. The profile provides a UML-based execution semantics that is equivalent to BPMN's, see Clause 1.

See 10.4, 10.5, and Clause 11 for more about the BPMNProcess and Subprocess stereotypes. BPMN GlobalManualTask, GlobalScriptTask, GlobalUserTask, and GlobalBusinessRuleTask have application semantics, see the BPMN specification. See 10.1.2 for more about the GlobalUserTask stereotype.

10.1.3 Derived Properties and Constraints

10.1.3.1 CallableElement

Derived properties

- /ioSpecification : InputOutputSpecification [0..1] = self.extension_ InputOutputSpecification
- /supportedInterfaceRefs : BPMNInterface [*] = self.base_Behavior.interfaceRealization.contract. extension_BPMNInterface (interfaceRealization defined on BehavioredClassifier)

Constraints

None

10.1.3.2 FlowElement

Derived properties

- /categoryValueRef : CategoryValue [*] = inverse of CategoryValue:: /categorizedFlowElements
- /container : FlowElementsContainer = inverse of FlowElementsContainer:: /flowElements.

Constraints

None

10.1.3.3 FlowElementsContainer

Derived properties

- /flowElements : FlowElement [*] = self.base_Activity.node&edge&group. extension_FlowElement when applied to Activities, and self.base_StructuredActivityNode. node&edge_extension_FlowElement when applied to StructuredActivityNodes.
- /laneSets : LaneSet [*] = self.base_Activity.partition.extension_LaneSet or self.base_ StructuredActivityNode.extension_SubProcess.hasLaneSets, whichever has a value.

Constraints

None

10.1.3.4 GlobalBusinessRuleTask

Derived properties

- /implementation : String [*] {ordered, non-unique} = self.base_ OpaqueBehavior.body (defaults to "##unspecified" , see BPMN specification)

Constraints

None

10.1.3.5 GlobalScriptTask

Derived properties

- /script : String [*] {ordered, non-unique} = self.base_OpaqueBehavior.body
- /scriptFormat : String [*] {ordered} = self.base_OpaqueBehavior.language

Constraints

None

10.1.3.6 GlobalTask

Derived properties

- /resources : ResourceRole [*] = self.base_OpaqueBehavior.ownedAttribute.extension_ ResourceRole, for properties that have ResourceRole applied (ownedAttribute defined on Class)

Constraints

None

10.1.3.7 GlobalUserTask

Derived properties

- /implementation : String [*] {ordered, non-unique} = self.base_OpaqueBehavior. body (defaults to “##unspecified,” see BPMN specification)
- /renderings : Rendering [*] = self.icon.extension_Rendering. (icon defined on Stereotype)

Constraints

None

10.1.3.8 BPMNProcess

Derived properties

- /artifact : Artifact [*] = union of self.base_Activity.node&edge.clientDependency& supplierDependency.extension_BPMNAssociation, self.base_Activity.partition.extension_Group, self.base_Activity.ownedComment.extension_TextAnnotation, self.base_Activity. node&edge.ownedComment.extension_TextAnnotation
- /properties : BPMNProperty [*] = self.base_Activity.ownedAttribute.bpmnProperty (ownedAttribute defined on Class).
- /resources : ResourceRole [*] = self.base_Behavior.ownedAttribute.extension_ResourceRole, for properties that have ResourceRole applied (ownedAttribute defined on Class)
- /supports : BPMNProcess = self.base_Activity./general.extension_BPMNProcess (/general defined on Classifier)

Constraints

[1] Activities with BPMNProcess applied that have a definitionalCollaboration must be one of the /processRefs of the definitionalCollaboration.

[2] The classifierBehaviors of activities with BPMNProcess applied are the activities themselves.

10.2 Activities

10.2.1 Illustration

Figure 10.3 illustrates BPMN Activities.

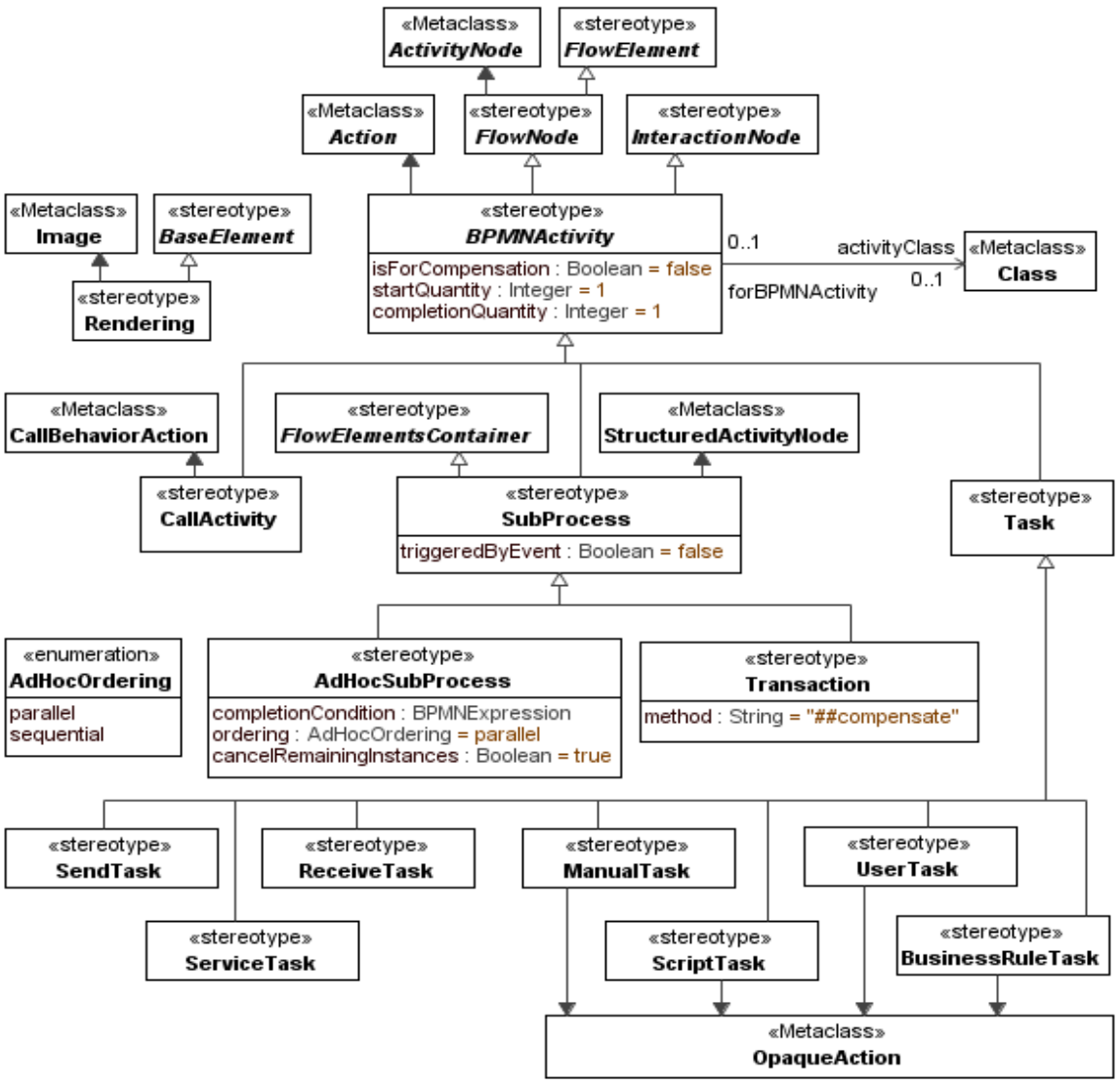

Figure 10.3 - Activities

BPMN Activities typically require only one incoming token to start, as expressed in the default for the startQuantity property, while UML Actions require all incoming control flows to offer a single token to start. For BPMN Activities with multiple incoming sequence flows and a value 1 for the startQuantity property, the corresponding UML Actions with a

specialization of the BPMNActivity stereotype applied must have single incoming control flow with a merge node as source, where the merge node has incoming control flows with the SequenceFlow stereotype applied. This provides a semantics equivalent to BPMN's by enabling the action to start with one incoming token. For BPMN Activities with multiple incoming sequence flows and a value other than 1 for the startQuantity property, the corresponding UML Actions with a specialization of the BPMNActivity stereotype applied must have single incoming control flow with a join node as source, where the join node has incoming control flows with the SequenceFlow stereotype applied, and has an opaque expression as the value of the joinSpec property, where the opaque expression body property has the value "StartQuantity" and the language property has the value "BPMNProfile." This provides a semantics equivalent to BPMN's by enabling the action to start with the required number of incoming tokens.

BPMN Activities typically emit one token to each outgoing sequence flow, as expressed in the default for the completionQuantity property, as do UML Actions to their outgoing control flows. For values of the completionQuantity property other than 1, the BPMNActivity stereotype extends UML Action semantics to offer an additional number of tokens equal to one less than the value of the completionQuantity property to each outgoing control flow on completion, giving a semantics equivalent to BPMN's.

A BPMN activity instance is a single occurrence of an activity being carried out within a specific process instance (see 10.1.2 about process instances). BPMN Activities can define properties that hold items of specified kinds. The BPMNActivity stereotype extends UML Actions with an activityClass property identifying a class that has a distinct instance every time actions are carried out in a UML activity instance. UML Properties hold values of specified types in class instances, including the classes with the BPMNActivity stereotype applied, providing a semantics equivalent to BPMN's. The Auditing and Monitoring stereotypes extend UML Classes for modelers to specify operations and properties for auditing and monitoring activity instances, see 10.1.1.

BPMN event subprocesses are subprocesses with no incoming sequence flows with start events that begin the subprocess under certain circumstances, as indicated by the triggeredByEvent property, optionally terminating the rest of the containing process, as indicated by the isInterrupting property of start events, see 10.4.3 and 10.2. The SubProcess stereotype with a value of true for the triggeredByEvent property extends UML Activity and StructuredActivityNode semantics to be equivalent to BPMN's by offering tokens to non-interrupting structured activity nodes after they start, and removing tokens accepted by accept event actions that start interrupting structured activity nodes that are executing or have been executed already in the containing activity execution. The Subprocess stereotype also extends the semantics of UML AcceptEventActions starting an interrupting structured activity node to terminate the activities outside the containing structured activity node when it accepts an event.

BPMN AdHocSubprocesses are subprocesses without complete sequencing and data flow between activities. The completionCondition property of the AdHocSubprocess stereotype determines when the structured activity node will end, the ordering property determines whether the actions in the structured activity node are carried out in parallel or sequentially, and the cancelRemainingInstances property determines whether actions carried out at the time the value of the completionCondition becomes true are terminated. This provides semantics equivalent to BPMN adhoc subprocesses.

BPMN Transactions are subprocesses that respond to cancel events by undoing any changes made during the transaction before cancellation. The method property of transactions has implementation semantics, see the BPMN specification. See 10.4.2.4 for events used with transactions.

BPMN UserTasks and BPMN GlobalUserTasks support modeler-defined icons (renderings), as do UML stereotypes (images). Modelers can specialize the UserTask and GlobalUserTask stereotypes to add images. BPMN CallActivities calling global user tasks with renderings also display the rendering. The CallActivity stereotype extends the semantics of UML Stereotypes to display images of specialized global user task stereotypes applied to called behaviors.

See the SequenceFlow stereotype in 10.3 for other semantics of the BPMNActivity stereotype, and 10.4.2.6 about the isForCompensation property (see 11.2.2 about BPMN SendTasks, ReceiveTasks, and ServiceTasks). BPMN ManualTask, ScriptTask, UserTask, and BusinessRuleTask have application semantics, see the BPMN specification.

BPMN Activities can be used in collaborations as interaction nodes, see 11.1.3 about the InteractionNodes stereotype.

10.2.2 Derived Properties and Constraints

10.2.2.1 BPMNActivity

Derived properties

- /boundaryEventRefs : BoundaryEvent [*] = instances of BoundaryEvent applied to accept event actions in same interruptible region as the action with BPMNActivity applied.
- /dataInputAssociations [*] : DataInputAssociation = self.base_Action./input.incoming .extension_DataInputAssociations
- /dataOutputAssociations [*] : DataOutputAssociation = self.base_Action./output. outgoing.extension_DataOutputAssociations
- /default : SequenceFlow [0..1] = self.base_BPMNActivity.outgoing.outgoing. guard="else".extension_SequenceFlow
- /ioSpecification : InputOutputSpecification [0..1] = self.extension_InputOutputSpecification
- /loopCharacteristics : LoopCharacteristics [0..1] = self.base_Action.inStructuredNode. extension_LoopCharacteristics
- /properties : BPMNProperty [*] = self.base_BPMNActivity.activityClass. ownedAttribute.bpmnProperty (ownedAttribute defined on Class).
- /resources : ResourceRole [*] = self.activityClass.ownedAttribute.extension_ ResourceRole

Constraints

[1] Actions with specializations of BPMNActivity applied may have no more than one incoming control flow.

[2] completionQuantity must be greater than zero.

[3] startQuantity must be greater than zero.

10.2.2.2 BusinessRuleTask

Derived properties

- /implementation : String = self.base_OpaqueAction.body.first (defaults to "##unspecified", see BPMN specification)

Constraints

None

10.2.2.3 CallActivity

Derived properties

- /calledElementRef : CallableElement = self.base_CallBehaviorAction.behavior. extension_CallableElement

Constraints

None

10.2.2.4 FlowNode

Derived properties

- /incoming : SequenceFlow [*] = self.base_ActivityNode.incoming.extension_ SequenceFlow
- /outgoing : SequenceFlow [*] = self.base_ActivityNode.outgoing.extension_ SequenceFlow

Constraints

None

10.2.2.5 ScriptTask

Derived properties

- /script : String [0..1] = self.base_OpaqueAction.body.first
- /scriptFormat : String [0..1] = self.base_OpaqueAction.language.first

Constraints

None

10.2.2.6 SubProcess

Derived properties

- /artifact : Artifact [*] = union of (self.base_StructuredActivityNode.node&edge. clientDependency&supplierDependency.extension_BPMNAssociation) and self.base_StructuredActivityNode.ownedComment. extension_TextAnnotationcies with BPMNAssociation applied.

Constraints

[1] StructuredActivityNodes with SubProcess applied that have triggeredByEvent = true, and containing initial nodes with StartEvent applied that have isInterrupting = false, have isLocallyReentrant = true.

10.2.2.7 Task

Derived properties

None

Constraints

[1] body may not have any values.

[2] language may not have any values.

[3] Task may only be applied to OpaqueActions (this does not apply to specializations of Task).

10.2.2.8 UserTask

Derived properties

- /implementation : String = self.base_OpaqueAction.body.first (defaults to "##unspecified", see BPMN specification)
- /renderings : Rendering [*] = self.icon.extension_Rendering (icon defined on Stereotype)

Constraints

None

10.3 Sequence Flows

10.3.1 Illustration

Figure 10.4 illustrates BPMN Sequence Flows.

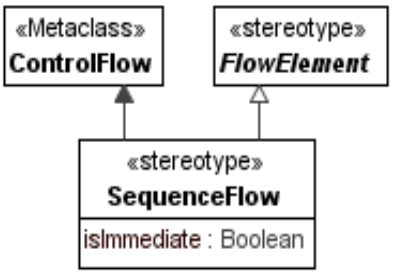

Figure 10.4 - Sequence Flows

One of the BPMN Sequence Flows that go out of activities and exclusive, inclusive, and complex gateways can be indicated as receiving a token if none of the other outgoing sequence flows from the same element do (the default sequence flow). For exclusive gateways with default sequence flows, UML Decision Nodes provide an equivalent semantics with outgoing edges that have an “else” guard, as expressed by the /default derived property of the ExclusiveGateway stereotype, see 10.5.1. For BPMN Activities, inclusive gateways, and complex gateways with multiple outgoing sequence flows and a default sequence flow, the corresponding actions and fork nodes with BPMNActivity, InclusiveGateway, or ComplexGateway stereotypes applied must have a single outgoing control flow targeting a decision node that has two outgoing control flows, one of which has an “else” guard and the other a fork node as target. The fork node must have outgoing control flows with the SequenceFlow stereotype applied. This provides a semantics equivalent to BPMN’s by offering a token along the “else” guard if none are accepted on the control flows going out of the fork.

The BPMN isImmediate property on sequence flows indicates whether activities not in a process model may occur between elements connected by the sequence flow (see discussion of process instances in the BPMNProcess stereotype). The SequenceFlow stereotype extends UML ControlFlow semantics to use the isImmediate property in determining whether actions not in an activity model may occur between elements connected by a stereotyped control flow, giving a semantics equivalent to BPMN’s.

10.3.2 Derived Properties and Constraints

10.3.2.1 SequenceFlow

Derived properties

- /conditionExpression [0..1] : BPMNExpression = self.base_ControFlow.guard. extension_BPMNExpression (guard defined on ActivityEdge)

- /sourceRef : FlowNode = self.base_ControlFlow.source.extension_FlowNode (source defined on ActivityEdge)
- /targetRef : FlowNode = self.base_ControlFlow.target.extension_FlowNode (target defined on ActivityEdge)

Constraints

None

10.4 Events

10.4.1 Events and Event Definitions

Figure 10.5 illustrates BPMN Events and Event Definitions.

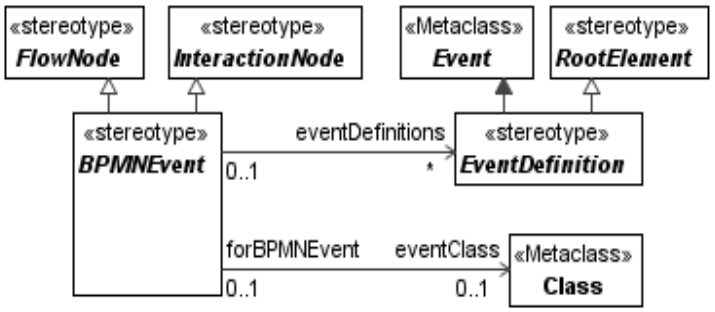

Figure 10.5 - Events and Event Definitions

BPMN Events are part of process flow (flow elements), while BPMN EventDefinitions are owned or referenced by flow elements. UML Actions are part of activity flow, while UML Events are packageable, and referenced indirectly by UML AcceptEventActions. BPMN informally calls event definitions the “triggers” of an event, which are received or noticed by BPMN CatchEvents, and sent or raised by BPMN ThrowEvents. UML Triggers are owned by UML AcceptEventActions to refer to UML Events that the action can accept. By convention in this specification, BPMN Events are identified by their event definitions, for example, a throw event with a message event definition is called a “message throw event,” or “message event” if it is clear from context that it is a throw event. BPMN Events can be used in collaborations as interaction nodes, see 11.1.3 about the InteractionNodes stereotype.

A BPMN event instance is a single occurrence of an event within a specific process instance (see 10.1.2 about process instances). BPMN Events can define properties that hold items of specified kinds. The BPMNEvent stereotype an eventClass property identifying a class that has a distinct instance every time events occur in a UML activity instance. UML Properties hold values of specified types in class instances, including the classes with the BPMNEvent stereotype applied, providing a semantics equivalent to BPMN’s. The Auditing and Monitoring stereotypes extend UML Classes for modelers to specify operations and properties for auditing and monitoring event instances, see 10.1.1.

The specialized event definitions shown in Figure 10.6, Figure 10.7, and Figure 10.8 are explained in the context of throwing and catching events in 10.4.2 and 10.4.3, respectively.

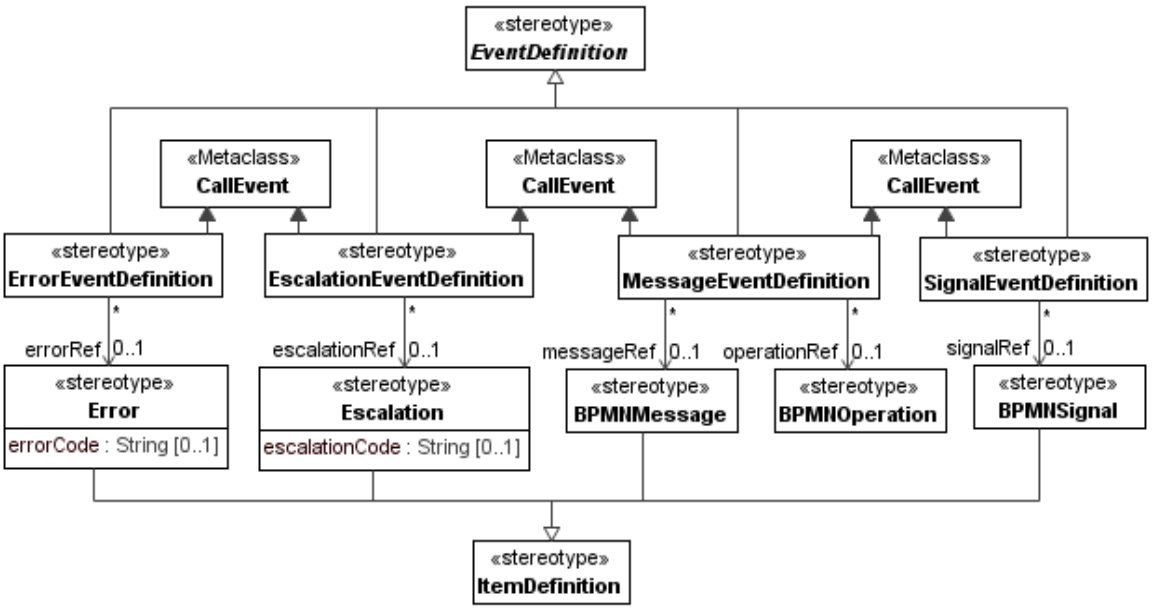

Figure 10.6 - Event Definitions with Item Definitions

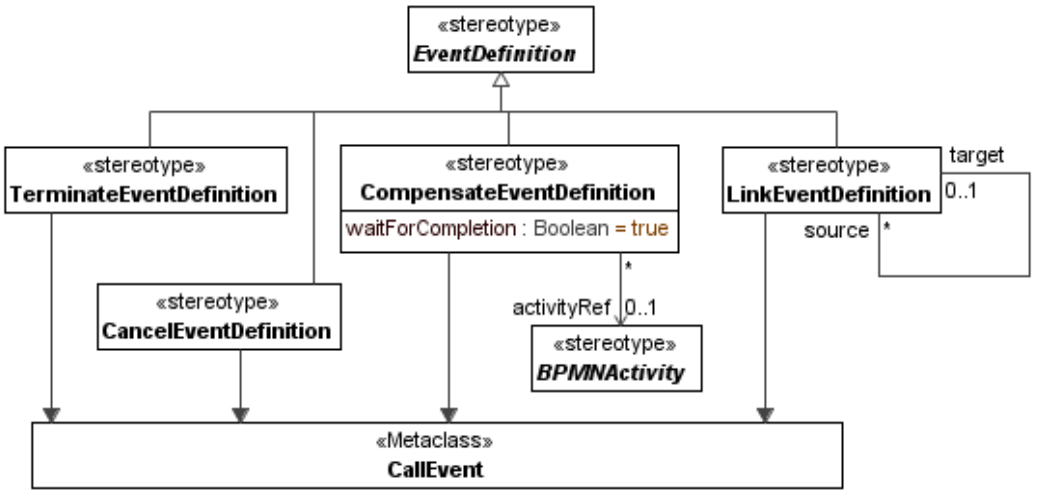

Figure 10.7 - Other Event Definitions extending Call Events without Item Definitions

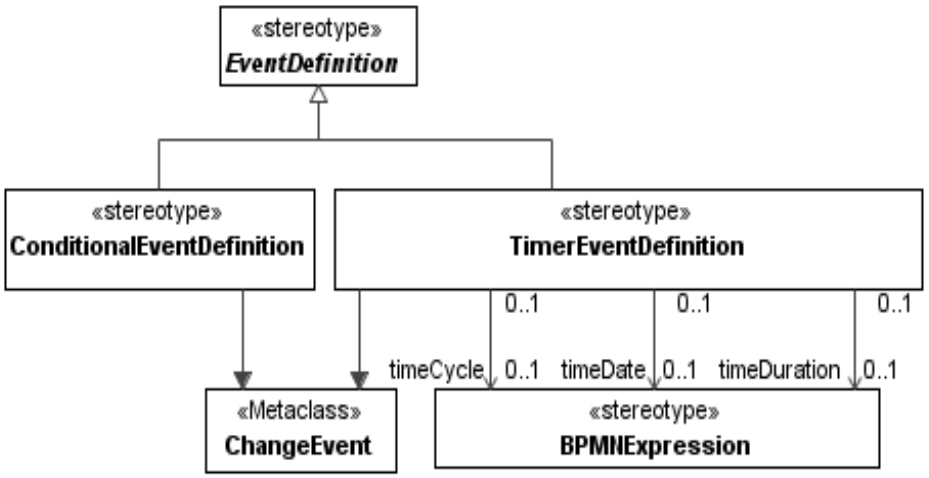

Figure 10.8 - Event Definitions Extending Change Events

10.4.2 Throw Events

Figure 10.9 illustrates BPMN Throw Events.

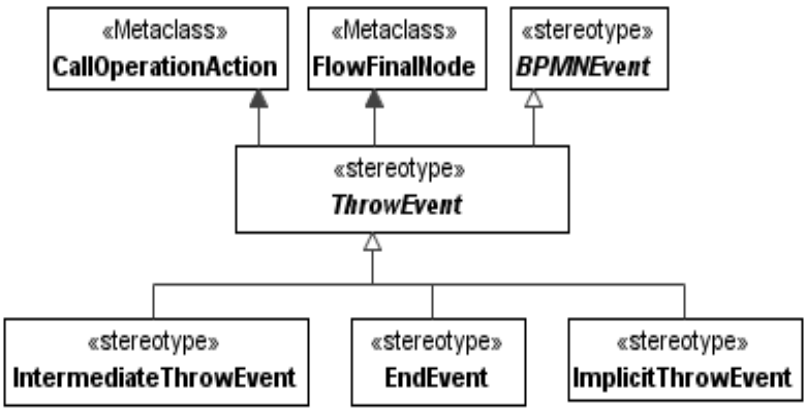

Figure 10.9 - Throw Events

BPMN ThrowEvents vary in the effects they have before events are thrown, how specific they are about their targets, and in the items that are thrown, if any. Items thrown are specified with EventDefinitions referring to BPMN Errors, Escalations, BPMNMessages, and Signals, that in turn refer to specialized ItemDefinitions for the type of thing sent or received along with the event. The profile reflects this in the event definition stereotypes in Figure 10.9, which refer to other stereotypes specialized from ItemDefinition, to simplify the model while still being equivalent to BPMN. Throw event stereotypes in the profile provide equivalent semantics to other aspects of BPMN by extending UML CallOperationActions with equivalent effects, determining its target appropriately, and passing items as UML Parameter values, if any (see 10.4.3 about operations for catching events).

BPMN ThrowEvents do not wait for events to be received before completing, except for compensation events, see 10.4.2.6. BPMN ThrowEvents with multiple event definitions throw all their events at once. In the multiple event definition case, UML CallOperationActions with throw event stereotypes applied have outgoing control flows to the same join node, providing semantics equivalent to BPMN's by proceeding past the fork only when all events have arrived. If the throw event is not an end event, the UML CallOperationActions have outgoing control flows to the same join node.

BPMN conditional and timer events are only caught, not thrown, see 10.4.3.3 and see 10.7 about BPMN ImplicitThrowEvents.

10.4.2.1 End Events without Event Definitions

BPMN EndEvents with no event definitions have no effect. They are not thrown anywhere in the sense of having corresponding catch events. UML FlowFinalNodes have equivalent semantics, and the EndEvent stereotype is applied to provide BPMN's terminology.

10.4.2.2 Link Events

BPMN link events are thrown to a single matching link catch event in the same process. Matching link events can be specified by the source / target association between BPMN LinkEventDefinitions when matching throw and catch events do not refer to the same link event definition, or when there are multiple link catch events that refer to the same event definition in the same process. UML CallOperationActions with a throw event stereotype applied, and LinkEventDefinition applied to its trigger, call an operation on the UML Activity or UML StructuredActivityNode instance that the call operation action is occurring in to notify it of the link event.

10.4.2.3 Terminate Events

BPMN terminate events permanently stop the process instance or innermost subprocess instance they are thrown in, which stops the activity instances under that process or subprocess instance recursively (see 10.1.2 and 10.2 about BPMN process and activity instances). Terminate events are not thrown anywhere in the sense of having corresponding catch events. UML CallOperations with a throw event stereotype applied, and the TerminateEventDefinition stereotype applied to its trigger, call a termination operation on the UML activity instance or innermost UML structured activity node instance they are occurring in, which terminates all the actions in the activity or structured node instance recursively (see 10.1.2 and 10.2 about UML activity and action instances).

BPMNProcess instances without terminate events are considered complete when all activities in the instance are finished, and no catch events are waiting (see 10.4.3 about catch events). UML Activities have the same semantics.

10.4.2.4 Cancel Events

BPMN cancel events have the same effect as terminate events on the innermost transaction subprocess instance they are thrown from, then undo any changes made during that transaction instance, and finally are thrown to cancel boundary catch events on that transaction, if any. UML CallOperationActions with a throw event stereotype applied, and the CancelEventDefinition stereotype applied to its trigger, call a termination operation on the innermost instance of UML StructuredActivityNode with the Transaction stereotype applied that the call operation action is occurring in, and undo any changes made during the structured node instance, then call an operation on the UML activity or structured node instance the transaction structured node instance was occurring in to notify them of the cancellation.

10.4.2.5 Error and Escalation Events

BPMN error and escalation events are thrown to matching error or escalation start events in event subprocesses at the same level they are thrown, if any, otherwise they are thrown to the innermost matching error or escalation catch event they are thrown from, including through call activities, which might be a boundary event or a start event of an event subprocess (see 10.4.3 about matching catch events). If the event definitions specify BPMN Errors or Escalations that refer to item definitions, then items conforming to these definitions are thrown with the event. As error events are thrown, they have the same effect as terminate events in the process and subprocess instances up to where the event is caught (except they do not terminate event subprocesses that catch them). If they are not caught, they have the effect of terminate events in process and subprocess instances all the way to the top-level process instance.

UML CallOperationActions with a throw event stereotype applied, and the ErrorEventDefinition or EscalationEventDefinition stereotype applied to its trigger, call an operation on the UML activity or UML structured activity node instance that the call operation action is occurring in to notify it of the error or escalation. If ErrorEventDefinition or EscalationEventDefinition stereotypes applied to triggers of call operation actions specify item definitions in the errorRef or escalationRef properties, then instances of the classes with those item definitions applied are passed as arguments of the operation. The classes have errorCode or escalationCode properties, typed by String, with multiplicity [0..1], read only, and default values equal to the values of the errorCode or escalationCode property of the event definition stereotypes (read only properties are given default values on instantiation that cannot be changed). If the activity or structured node instances do not handle the notification (“catch the event”), as indicated by returning a true value, error or escalation notification operations are called on activity or structured node the next level up with the same arguments, recursively, until an activity or structured node instance handles the notification, or a top-level activity instance is reached. For error events, activity or structured node instances that do not handle the notification operations are terminated before the next level is notified.

10.4.2.6 Compensation Events

BPMN compensation events attempt to reverse changes made during completed instances of activities identified by the event ahead of time, if any. If the identified activity is a subprocess, then this is done with compensation event subprocesses in the activity, if any, otherwise reversal is done with activities identified by boundary compensation events on the activity, using associations, if any (boundary compensation events do not actually catch events, the activity they are attached to is finished by the time compensation occurs). If the throw compensation event does not identify an activity to be compensated, the compensation events are thrown to start compensation catch events in event subprocesses at the same level they are thrown from, if any. If there is no compensation event subprocess, throw compensation events attempt to reverse changes made during completed activity instances at the same level of the activity instance the events are thrown from, in the reverse order in which the activity instances occurred, unless the events are thrown from a compensation event subprocess, whereupon the throw compensation events attempt to reverse changes due to completed activity instances at the same level as the subprocess in the reverse order in which they occurred. Throw compensation events can wait for events to be received wherever they are sent before completing if needed.

CallOperations with a throw event stereotype applied, and a compensation event definition, call compensation operations on the completed instances of UML Action identified by the activityRef property, if any, which attempt to reverse changes made during completed instances of actions. If the identified action is a structured activity node, then a token is offered to a structured activity node with SubProcess applied in the action, and a true value for triggeredByEvent property, containing a UML AcceptEventAction with StartEvent applied and triggered by a UML CallEvent with CompensationEventDefinition applied, if any, otherwise reversal is done by offering tokens to actions that have BPMNActivity applied and a true value for the isForCompensation property, that are suppliers of UML Dependencies that have as clients UML AcceptEventActions with BoundaryEvent applied in the same UML InterruptibleRegion as the compensated action, if any. If the applied throw event stereotype does not identify an action to compensate with the activityRef property, a token is offered to a structured activity node with SubProcess applied at the same level as the call

operation action, and a true value for triggeredByEvent property, containing a UML AcceptEventAction with StartEvent applied and triggered by a UML CallEvent with CompensationEventDefinition applied. If there is no such structured activity node, compensation operations are called on completed action instances at the same level of UML activity or UML structured activity node instance the call operation action is occurring in, unless it is occurring in an instance of UML StructuredActivityNode with Subprocess applied, and a true value for triggeredByEvent property, containing a UML AcceptEventAction with StartEvent applied and triggered by a UML CallEvent with CompensationEventDefinition applied, whereupon compensation operations are called on completed action instances at the same level as the structured activity node in the reverse order in which they occurred. The BPMNProcess and Subprocess stereotypes extend UML Activity and StructuredActivityNode semantics respectively to remove tokens accepted by actions with BPMNActivity applied that have a true value for the isForCompensation property. The call operation actions for compensation are asynchronous, except when the CompensateEventDefinition stereotype is applied to their triggers with a true value for the waitForCompletion property, whereupon the call operation action is synchronous.

10.4.2.7 Message Events

BPMN message events are thrown to particular external participants in collaborations, which involves sending items to those participants as messages (see 10.4.2 for more about throwing messages).

10.4.2.8 Signal Events

Signals are thrown to all process instances, including those in other participants (some implementations might throw signals just to process instances subscribed to or accepting matching signals, the effect is the same, see 10.4.3 about matching catch events). If the event definitions specify BPMN Signals that refer to item definitions, then items conforming to these definitions are thrown with the event. UML CallOperationActions with a throw event stereotype applied, and a signal event definition, have value pins as their target pins, where the value property of the pin is an opaque expression with body “BPMNProcessInstance” and language “BPMNProfile,” indicating it returns all process instances that support the signal operation, providing semantics equivalent to BPMN’s. If the signal event definition stereotypes applied to the triggers of call operation actions specify item definitions in the signalRef properties, then instances of the classes with those item definitions applied are passed as arguments of the operations.

10.4.3 Catch Events

Figure 10.10 illustrates BPMN Catch Events.

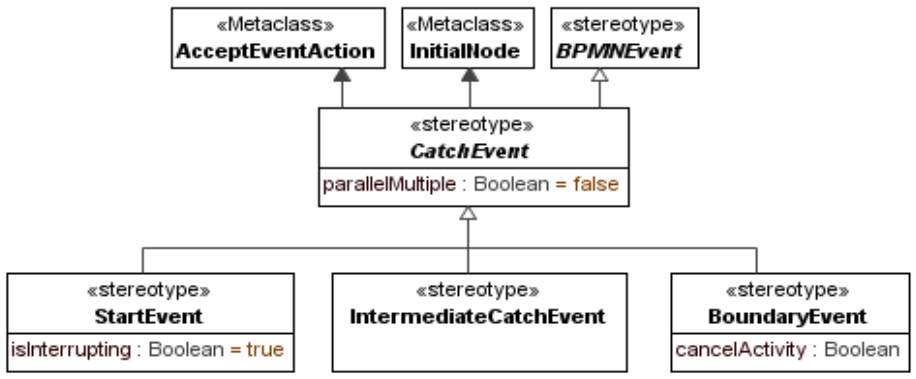

Figure 10.10 - Catch Events

BPMN catch events accept events thrown with the same or equivalent event definitions, including referenced item definitions, except for link events below. Catch event stereotypes may only be applied to UML AcceptEventActions, except for start events that have no event definitions, which may only be applied to UML InitialNodes. UML activity and action instances support one operation for each concrete event definition stereotype, except for the LinkEventDefinition stereotype, where they support a separate operation for each instance of the stereotype that defines an event in the activity or action (action instances do not support signals, see 10.4.2 about invoking the operations when throwing events). The names of these operations are defined by simulation, enactment, and execution engines supporting the profile, or standards these engines choose to adopt, rather than the profile or modelers. UML AcceptEventActions with catch event stereotypes applied have triggers referring to UML CallEvents that refer to those operations, which means the actions only accept calls to those operations. Operations for errors, escalations, and signals have one argument, which is untyped, the other operations have none. For operations that accept an argument, the catch event stereotypes extend UML AcceptEventAction semantics by removing tokens accepted when the actual argument is not an instance of the class referred to by the event definition stereotype applied to the trigger of the action, or one of its subclasses, and returning the event to the pool from which it was drawn. This gives semantics equivalent to BPMN's, and integrates it with UML's for support specialization of UML Class, which the ItemDefinition stereotype is based on. See 11.2.2 about catching messages.

The BPMN parallelMultiple property on catch events indicates whether all of the event definitions of the event are required to trigger the event, or only one of them. In the parallel-multiple case, multiple UML AcceptEventActions with catch event stereotypes applied and one trigger per action have outgoing control flows to the same join node, providing semantics equivalent to BPMN's by proceeding past the join only when all events have arrived. For catch events that are not start events, the corresponding UML AcceptEventActions have incoming control flows from the same fork node.

10.4.3.1 Start Events

BPMN StartEvents without event definitions indicate where to start new process instances, they do not actually catch events.

BPMN StartEvents with event definitions at the top level of a process instantiate processes when events of particular kinds occur. The BPMNProcess stereotype extends UML Activity semantics to be equivalent to BPMN's by instantiating activities that directly contain accept event actions with StartEvent applied with event definitions when those events occur, and removing tokens from accept event actions with StartEvent applied that are not for the event that instantiated the process (these tokens are also removed for initial nodes with StartEvent applied, and any of them can be the one retaining a token).

BPMN StartEvents with error or escalation event definitions that specify errors or escalations with values for their errorCode or escalationCode properties only catch error or escalation events with the same codes. If the start events do not specify codes, they catch any error or escalation event. UML AcceptEventActions with StartEvents applied, and triggered by UML CallEvents with ErrorEventDefinition or EscalationEventDefinition applied, extend UML AcceptEventAction semantics by removing tokens accepted by accept event actions when the actual argument has a different value from the errorCode and escalationCode property of the value of the errorRef or escalationRef property of the triggering UML Event, and returning the event to the pool from which it was drawn, giving semantics equivalent to BPMN's.

BPMN StartEvents with compensation event definitions in event subprocesses that are in subprocesses attempt to reverse changes made during completed instances of the containing subprocess, and instantiate the event subprocess. The Subprocess stereotype with a true value for the isForCompensation property, containing a UML AcceptEventAction with StartEvent applied and triggered by a UML CallEvent with CompensationEventDefinition applied, extends UML Activity and StructuredActivityNode semantics to be equivalent to BPMN's by offering a token to the structured activity node in

the activity instance in which the containing structured activity instance occurred, giving semantics equivalent to BPMN's.

The BPMN isInterrupting property on start events is for event subprocesses, see 10.4.3 and 10.2.

10.4.3.2 Intermediate Catch Events

BPMN IntermediateCatchEvents with incoming sequence flows only catch events that occur after the process flow reaches them and before it leaves them, while UML AcceptEventActions with incoming control flows accept events that occur anytime before the action ends, including before it starts. The BPMNProcess stereotype extends UML event pool semantics by discarding events that no action can accept at the time they are put in the pool, giving semantics equivalent to BPMN's.

BPMN link intermediate catch events with event definitions that are targets of links via the source / target association between BPMN LinkEventDefinitions only catch events thrown from events with event definitions that are sources of the same links, even if the catch and throw events refer to the same link event definitions. IntermediateCatchEvent stereotypes with LinkEventDefinition stereotypes that are targets of links via the source / target association extend the semantics of UML AcceptEventActions by removing tokens accepted by accept event actions for events that do not have LinkEventDefinitions applied that are source of the same links, providing a semantics equivalent to BPMN's.

The BoundaryEvent stereotype may only be applied to accept event actions in interruptible regions. If the value of the cancelActivity property is true, the outgoing edges of the accept event actions are interrupting, otherwise they are non-interrupting (this is termination of the action, rather than cancellation in the transactional sense, see 10.2 and 10.4.2.4). Exactly one action with a specialization of BPMNActivity applied must be in the interruptible region along with the accept event actions, and have an outgoing interrupting edge to a target outside the region. This provides a semantics equivalent to BPMN BoundaryEvents by waiting for the accept event actions in the interruptible region only while the action with BPMNActivity applied is occurring, and terminating that action and the accept event actions when an event is accepted by an action with an outgoing interruptible edge, or waiting for more events after an event is accepted by an action with a non-interrupting outgoing edge. Error and escalation code matching apply to error and escalation boundary events in the same way as start events in both BPMN and the profile, see 10.4.3.1.

10.4.3.3 Conditional and Timer Catch Events

BPMN catch events with conditional event definitions and timer event definition are for noticing changes in conditions and time. The ConditionalEventDefinition and TimerEventDefinition stereotypes are applied to UML ChangeEvents. This provides equivalent semantics of BPMN's for conditional events, as expressed in the /condition derived property of the ConditionalEventDefinition stereotype. Change events with the TimerEventDefinition stereotype applied have opaque expressions as the values of their changeExpression property, where the opaque expression body property has the value "BPMNTimerEvent" and the language property has the value "BPMNProfile." This extends UML ChangeEvent semantics to detect the arrival of a particular time with the timeDate property, the passing of a particular amount of time with the timeDuration property, and the arrival of a recurring time with the timeCycle property, whichever has a value, equivalent to the semantics of BPMN's for timer event definitions.

10.4.4 Derived Properties and Constraints

10.4.4.1 BPMNEvent

Derived properties

- /eventDefinitionRefs : EventDefinition [*] = instances of EventDefinition that are not eventDefinitions of the

stereotype and, if the stereotype is a throw event, are applied to call events for the same operation as the one specified by the call operation action to which the stereotype is applied, or, if the stereotype is a CatchEvent, are applied to events of the triggers of the accept event action to which the stereotype is applied.

- /properties : BPMNProperty [*] = self.base_BPMNEvent.eventClass. ownedAttribute. bpmnProperty (ownedAttribute defined on Class)

Constraints

[1] The eventDefinitions of concrete specializations of BPMNEvent, if the specialization is a throw event, must be applied to call events for the same operation as the one specified by the call operation action to which the specialized BPMNEvent is applied, or, if the stereotype is a catch event, must be applied to events of the triggers of the accept event action to which the specialized BPMNEvent is applied.

10.4.4.2 BPMNMessage

Derived properties

- /itemRef : ItemDefinition [0..1] = self.base_Class.extension_ItemDefinition

Constraints

None

10.4.4.3 BPMNSignal

Derived properties

Constraints

None

10.4.4.4 BoundaryEvent

Derived properties

- /attachedToRef : BPMNActivity = instance of BPMNActivity applied to action in same interruptible region as the event.

Constraints

None

10.4.4.5 CatchEvent

Derived properties

- /dataOutputAssociation [*] : DataOutputAssociation = self.base_Action./output.outgoing. extension_DataOutputAssociations
- /dataOutputs [*] : DataOutput = self.base_Action./output. extension_DataOutput

Constraints

[1] AcceptEventActions with catch event stereotypes applied must have isUnmarshall = false.

[2] AcceptEventActions with catch event stereotypes applied must have triggers that have a specialization of EventDefinition applied.

10.4.4.6 ConditionalEventDefinition

Derived properties

- /condition : BPMNExpression = self.base_ChangeEvent.changeExpression.extension_ BPMNExpression

Constraints

None

10.4.4.7 Error

Derived properties

None

Constraints

None

10.4.4.8 Escalation

Derived properties

None

Constraints

None

10.4.4.9 ImplicitThrowEvent

Derived properties

None

Constraints

[1] ImplicitThrowEvent may only be applied to call operation actions.

10.4.4.10 StartEvent

Derived properties

None

Constraints

[1] Start events with no event definitions (eventDefinitions or /eventDefinitionsRefs) may only be applied to initial nodes, otherwise, they may only be applied to accept event actions.

10.4.4.11 ThrowEvent

Derived properties

- /dataInputAssociation [*] : DataInputAssociation = self.base_Action./input.incoming.extension_DataInputAssociations
- /dataInputs [*] : DataInput = self.base_Action./input.extension_DataInput

Constraints

[1] Call operation actions with throw event stereotypes applied must have isSynchronous = false, except when CompensateEventDefinition is applied to their triggers with waitForCompletion = true, whereupon isSynchronous = true.

[2] Call operation actions with throw event stereotypes applied, and a trigger with ErrorEventDefinition, EscalationEventDefinition, or SignalEventDefinition applied that refer to classes with Error, Escalation, or BPMNSignal applied, respectively, have a single argument input pin typed by those classes, and multiplicity 1, otherwise the call operation actions have no input pins.

[3] Call operation actions with throw event stereotypes applied, and a trigger with SignalEventDefinition applied must have no incoming edge to their target pins.

[4] Operations invoked by call operation actions with throw event stereotypes applied must have no methods.

[5] Operations invoked by call operation actions with throw event stereotypes applied that have event definitions (eventDefinitions or /eventDefinitionsRefs) with ErrorEventDefinition, EscalationEventDefinition, or SignalEvent Definition applied have one in parameter with no type, and multiplicity 1, otherwise the operations have no in parameters.

[6] The operations invoked by call operation actions with throw event stereotypes applied must have no out or return parameters, except if they have event definitions (eventDefinitions or /eventDefinitionsRefs) that are events with ErrorEventDefinition or EscalationEventDefinition applied, whereupon the operations have no out parameters and exactly one return parameter typed by Boolean, with multiplicity 1.

10.4.4.12 TimerEventDefinition

Derived properties

Constraints

[1] Change events with TimerEventDefinition applied have an opaque expression as the value of changeExpression, where the opaque expression body property has the value “BPMNTimerEvent” and the language property has the value “BPMNProfile.”

10.5 Gateways

Figure 10.11 illustrates BPMN Gateways.

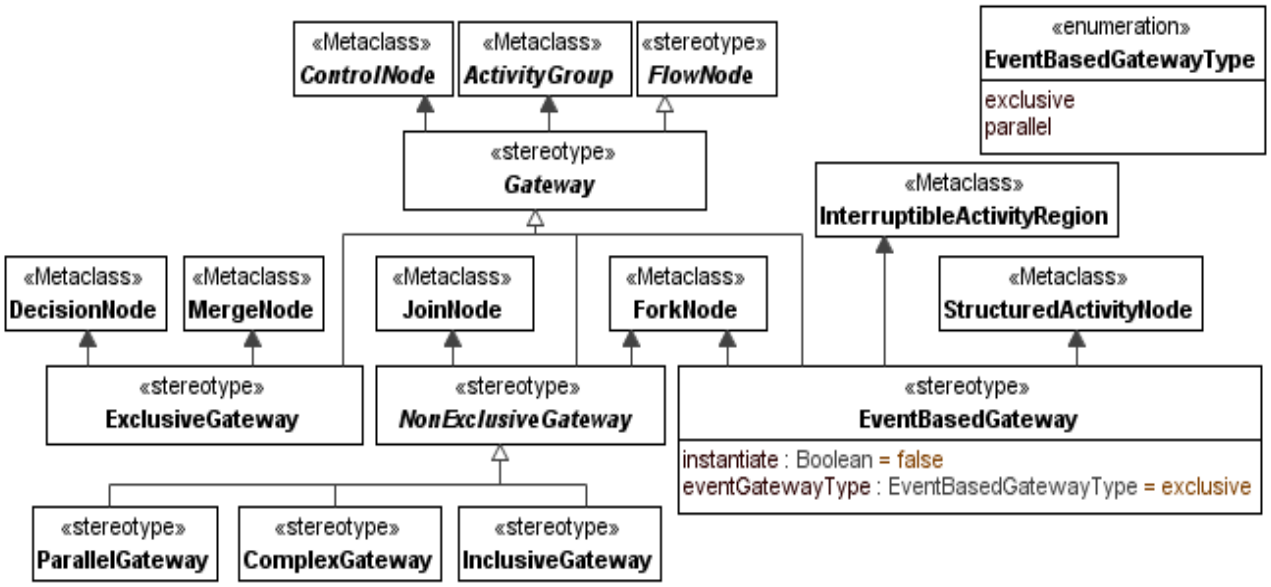

Figure 10.11 - Gateways

The bases of the gateway stereotypes reflect differences in the models of BPMN Gateways and UML ControlNodes. BPMN Gateways can be converging, diverging, both, or neither (except for event-based gateways, which are not converging), while UML JoinNodes and MergeNodes have exactly one outgoing edge, ForkNodes and DecisionNodes have exactly one incoming edge (decision nodes have an optional additional incoming edge not used in this profile). The profile does not support gateways that are neither converging nor diverging. ForkNodes or JoinNodes can be created for these, then reclassified or replacing them when the direction is determined. UML notation supports combining fork and join nodes into one graphical symbol, as BPMN does, but UML models store this as separate fork and join nodes, while BPMN stores it as a single gateway that is both converging and diverging. See 10.3 for other semantics of ExclusiveGateway, ComplexGateway, and InclusiveGateway in addition to 10.5.1, 10.5.3, and 10.5.4.

10.5.1 Parallel and Exclusive Gateways

The semantics of BPMN ParallelGateways and UML ForkNodes are equivalent, as are the semantics of BPMN ExclusiveGateways and UML DecisionNodes and MergeNodes.

10.5.2 Event-based Gateways

BPMN EventBased Gateways wait for one of the events immediately following them to occur, after which tokens flow past that event. The BPMN instantiate property on event-based gateways indicates whether new process instances are created when events immediately following the gateway occur (see 10.1.2 about process instances). When the value of the instantiate property is

- false, the corresponding EventBasedGateway stereotype is applied to UML Forks. These forks must have outgoing control flows with the SequenceFlow stereotype applied and targeting accept event actions (see 10.4.3) that are contained in an interruptible region, and the outgoing edges from the actions must be interrupting and target elements outside the region. This gives a semantics equivalent to BPMN's by terminating the accept event actions that are not the first to receive an event.

- true, the BPMN EventBased Gateway has no incoming sequence flows. The EventBasedGateway stereotype and forks are not used in this case and the accept event actions have no incoming edges. The BPMN eventGatewayType property is only relevant when instantiating processes. A value of Parallel requires all events immediately following the gateway to occur before the process instance completes, while a value of Exclusive requires exactly one. For parallel and instantiating event-based gateways, the corresponding accept event actions are in an structured node. This gives a semantics equivalent to BPMN's by requiring all accept event actions immediately following the fork to receive an event before the activity instance completes. For exclusive and instantiating event-based gateways, the corresponding accept event actions are contained in an interruptible region, and the outgoing edges from the actions must be interrupting and target elements outside the region, providing a semantics equivalent to BPMN's by terminating the accept event actions that are not the first to receive an event.

10.5.3 Complex Gateways

The BPMN property activationCondition only applies to converging complex gateways, and is equivalent to the joinSpec property of UML JoinNodes, as expressed by the /activationCondition derived property of the ComplexGateway stereotype.

10.5.4 Inclusive Gateways

Converging BPMN Inclusive Gateways consume a token from each of their incoming sequence flows that has one when at least one incoming sequence flow has at least one token, and for every path of sequence flows to the gateway that

- starts with a sequence flow having a token,
- ends with a sequence flow that has no token, and
- does not include the gateway.

There is also a path of sequence flows to the gateway that

- starts with the same sequence flow as the first path,
- ends with a sequence flow coming into the gateway that has a token, and
- does not include the gateway.

The InclusiveGateway stereotype extends the semantics of UML JoinNodes to be equivalent to BPMN's when the joinSpec property has as a value an opaque expression with body "BPMNInclusive" and language "BPMNProfile," which is required when the stereotype is applied to join nodes. In this case, join nodes accept all the tokens offered by incoming edges when at least one incoming edge is offering at least one token, and for every path of control flows to the join node that

- starts with a control flow that is offered a token its own conditions can accept,
- ends with a control flow that is not offered a token, or is offered a token its own conditions cannot accept, and
- does not include the join node.

There is also a path of control flows to the join node that

- starts with the same control flow as the first path,
- ends with a control flow coming into the join node that is offered a token its own conditions can accept, and
- does not include the join node.

10.5.5 Derived Properties and Constraints

10.5.5.1 ComplexGateway

Derived properties

- /activationCondition : BPMNExpression [0..1] = self.base_JoinNode.joinSpec.extension_ BPMNExpression
- /default : SequenceFlow [0..1] = instances of SequenceFlow applied to an outgoing control flow with guard = “else.”

Constraints

[1] JoinNodes with ComplexGateway applied must have joinSpec = “BPMNInclusive.”

10.5.5.2 ExclusiveGateway

Derived properties

- /default : SequenceFlow [0..1] = self.base_DecisionNode.outgoing. guard="else".extension_SequenceFlow

Constraints

None

10.5.5.3 InclusiveGateway

Derived properties

- /default : SequenceFlow [0..1] = instances of SequenceFlow applied to an outgoing control flow with guard = “else.”

Constraints

None

10.6 Data

10.6.1 Item Aware Elements and Data Associations

Figure 10.12 illustrates Item Aware Elements and Data Associations.

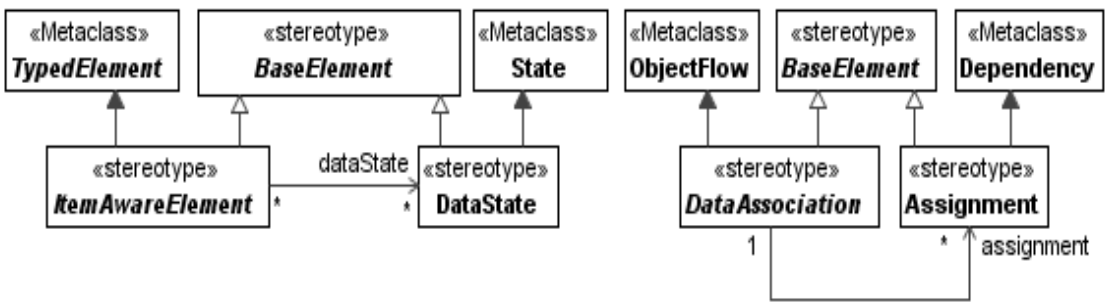

Figure 10.12 - Item Aware Elements and Data Associations

Data flow modeling in BPMN and UML use different terminology, but take similar approaches. Both are based on elements that identify individual things when processes are carried out (BPMN ItemAwareElements and UML TypedElements, specifically UML ObjectNodes in activities, including UML ActivityParameterNodes, InputPins, OutputPins, and DataStoreNodes, see 10.6.2 and 10.6.5), and links between these elements to enable “flow” of identified items or values (BPMN DataAssociations and UML ObjectFlows). BPMN ItemAwareElements and UML TypedElement can specify the kinds of things they identify, in terms of BPMN ItemDefinitions (see 9.2.2) and UML Types (including classes), respectively. Items flowing into BPMN ItemAwareElements typically replace any that are there already, and items flow out of them all at once and leave duplicates behind. Item aware element stereotypes extending UML ObjectNodes extend their semantics to remove tokens from the node before accepting tokens, and retain duplicate tokens for any accepted offers they make (this is already the semantics of data store nodes), providing semantics equivalent to BPMN’s. BPMN Assignments give pairs of expressions for getting data out of the source end of associations and placing it in the target. UML Dependencies with the Assignment stereotype applied have as client and supplier UML OpaqueExpressions with the BPMNExpression stereotype applied.

All BPMN ItemAwareElements can specify a single state that data must be in, where the state is specified as a string. UML has a state machine model supporting multiple states per object, and typed elements used in activities (UML ObjectNodes) can specify which of these states their tokens must be in, while the ItemAwareElement stereotype extends UML TypedElement to support specifying states for parameters with the DataInput or DataOutput stereotypes applied for opaque behaviors with global task stereotypes applied (see 10.6.2 about global tasks and 10.6.2 about parameters). The profile uses UML state machines to specify states rather than strings.

10.6.2 Input Output Specifications and Data Objects

Figure 10.13 illustrates Input Output Specifications and Data Objects.

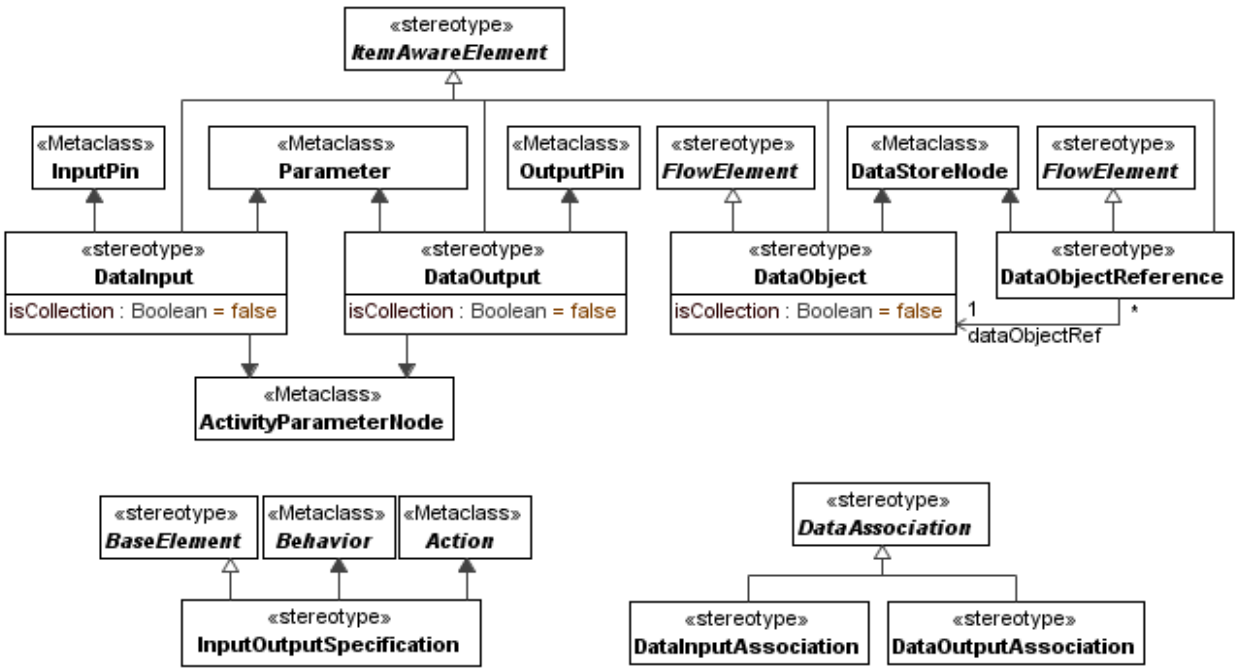

Figure 10.13 - Input Output Specifications and Data Objects

BPMN and UML both require matching input and output specifications when global elements (BPMN Processes and GlobalTasks, and UML Behaviors, including UML Activities and OpaqueBehaviors, see 10.1.2) are reused by elements within a flow (BPMN Activities and UML Actions, see 10.2). BPMN uses the same specification structure for the matching specifications (BPMN InputOutputSpecification), while in UML they are different. This is reflected in multiple bases for the DataInput and DataOutput stereotypes (UML Pins and ActivityParameterNodes, which are object nodes on actions and activities, respectively, and UML Parameters, which are typed elements on behaviors, specifically opaque behaviors). DataInput and DataOutput stereotypes are applied to UML Parameters only on opaque behaviors extended by global task stereotypes. BPMN GlobalTasks cannot contain data associations between their data inputs and outputs, and UML ObjectFlows cannot link parameters. BPMN does not notate data input and output on activities, while UML has an optional notation for pins shown attached to actions. The isCollection properties of the DataInput, DataOutput, and DataObject stereotypes (see below) have the same effect as the isCollection property of the ItemDefinition stereotype applied to the type of the UML TypedElement those stereotypes are applied to.

BPMN always interposes another data element when items flow between data outputs of one activity and data inputs of another (BPMN DataObjects and DataObjectReferences), while UML has the option for values to flow directly from output pins of one action to input pins of another (this option is not used when the profile is applied). BPMN DataObjects and DataObjectReferences are very similar, but data objects cannot specify states, data object references cannot specify item definitions, and the items that data object references identify during execution must be the same as those identified by the data object they refer to. The DataObjectReference stereotype requires data store nodes to have the same tokens as the data store node they refer to, providing a semantics equivalent to BPMN's. BPMN Activities own the data input associations and data output associations linked to them, while UML Actions do not own the object flows linked to their pins. This does not affect semantics. The InputOutputSpecification stereotype extends UML elements in flows (actions) and reusable global elements (activities and opaque behaviors) to define derived properties from BPMN for inputs, outputs, and alternative input and output sets.

10.6.3 Input Sets and Output Sets

Figure 10.14 illustrates BPMN Input Sets and Output Sets.

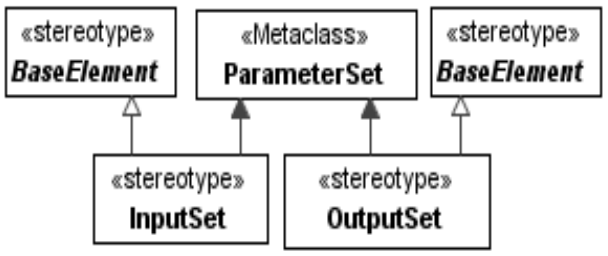

Figure 10.14 - Input Sets and Output Sets

BPMN and UML support grouping of inputs and outputs into alternative sets (BPMN InputSets and OutputSets, and UML ParameterSets), where only one of the sets of inputs and one of the sets of outputs are used in each execution. BPMN supports this on elements within a flow (activities) and reusable global elements (processes and global tasks), while UML only supports it on reusable global elements (behaviors, including opaque behaviors), but not elements within a flow (actions). Parameter sets of behaviors called by UML CallActions with the CallActivity stereotype applied provide semantics equivalent to BPMN BPMN InputSets and OutputSets on call activities, because in BPMN these must match the called element (see BPMN InputOutputSpecification above), and BPMN GlobalTasks with multiple input or output sets are equivalent to UML CallActions calling opaque behaviors with a global task stereotype applied and parameter sets

with the InputSet and OutputSet stereotypes applied. The profile does not support BPMN InputSets and OutputSets on tasks that are not global. BPMN InputSets and OutputSets specify which inputs and outputs are optional, and also specify which inputs can be accepted during execution, rather than at the beginning of execution, and which outputs can be provided during execution, rather than at the end of execution (whileExecuting inputs and outputs). These characteristics in UML are specified on the inputs and outputs directly (parameter lower multiplicity and streaming), rather than the sets, so cannot be different depending on the sets chosen during execution, as they can in BPMN. The profile does not support these characteristics on alternative sets of inputs and outputs. See 11.2.1 for more about input and output sets.

10.6.4 Properties as Item Aware Elements

Figure 10.15 illustrates Properties as Item Aware Elements.

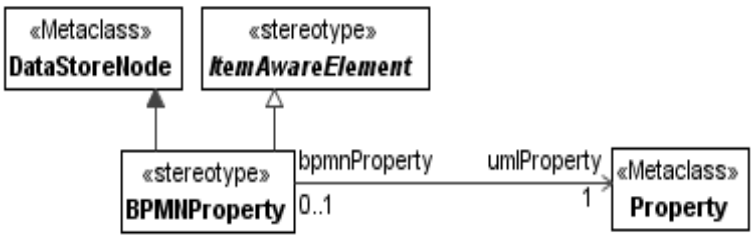

Figure 10.15 - Properties as Item Aware Elements

BPMN Properties as item aware elements can be used with data associations to provide or accept items from other item aware elements. UML Properties hold values of specified types for classes, including UML Activities, and activity and event classes of BPMNActivity and BPMNEvent stereotypes (see 10.2 and 10.4.1), but cannot be used with object flows. The BPMNProperty stereotype extends UML DataStoreNodes to ensure the tokens they contain are for exactly the values of the UML Property associated with the stereotype while the containing activity or structured activity node is carried out, and as tokens in the data store node and values of the UML Property change. This means tokens are added or removed as values of the UML Property change, and vice versa, providing a semantics equivalent to BPMN's.

10.6.5 Data Stores

Figure 10.16 illustrates BPMN Data Stores.

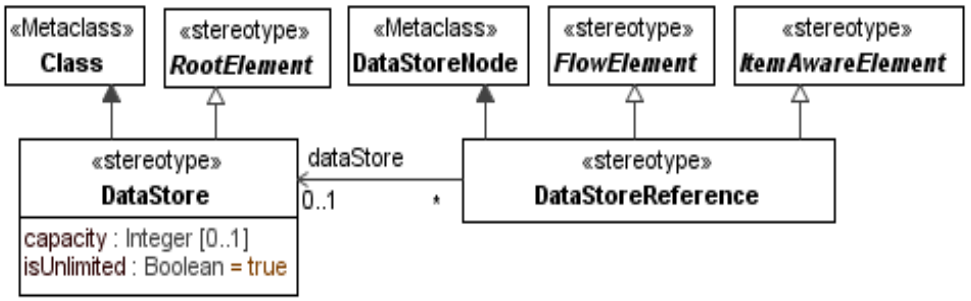

Figure 10.16 - Data Stores

BPMN DataStores identify items that exist independently of processes, including when no processes are being carried out that use the items. BPMN DataStores are item aware elements specifying the definition of items that are stored, but unlike other item aware elements, they are root elements (contained in definitions, see 9.2.2 and 9.1.2 about item definitions, definitions, and root elements). The ItemAwareElement stereotype extends UML TypedElements, which identify values, but always in the context of elements other than UML Packages, which is extended by a root element stereotype. The DataStore stereotype extends UML Classes, to enable data stores to be packaged and still specify a type. The extended classes have one generalization with ItemDefinition applied that corresponds to the item definition of the BPMN DataStore, see the /itemSubjectRef derived property of the DataStore stereotype. The instances of the specialized class (the one with DataStore applied) are instances of the general class (the one with ItemDefinition applied) that are “in” the store. BPMN does not specify what is measured by the capacity property of data stores, or its units, and does not specify the effect of a full data store on flow of items in a process. The capacity property is only used is the values of the isUnlimited property is false. The profile includes these properties with semantics equivalent to BPMN’s, as far as it is specified.

BPMN DataStoreReferences are process elements that are also item aware elements, enabling data associations to assign items to data stores or find out the items in the data stores (see 10.6.1 on the semantics of data associations). The DataStoreReference stereotype extends UML DataStoreNodes to ensure the tokens they contain while the containing activity or structured activity node is carried out are for exactly the instances of the class with the associated DataStore stereotype applied, even as tokens in the data store node and instances of the class change. This means tokens are added or removed as instances of the class change, and vice versa, providing a semantics equivalent to BPMN’s. In particular, objects removed as instances of the data store class might remain classified by the class with ItemDefinition applied directly generalizing the data store class. They still exist, but not in the data store.

10.6.6 Derived Properties and Constraints

10.6.6.1 Assignment

Derived properties

- /from : BPMNExpression = self.base_ Dependency.supplier.extension_BPMNExpression
- /to : BPMNExpression = self.base_Dependency.client.extension_BPMNExpression

Constraints

[1] Dependencies with Assignment applied must be owned by the innermost package containing the data association referring to the instance of Assignment applied to the dependency.

[2] Dependencies with the Assignment applied must have exactly one client and one supplier, and both must have FormalExpression applied.

[3] Opaque expressions with FormalExpression applied must not be clients or suppliers of more than one dependency with Assignment applied.

[4] Opaque expressions with FormalExpression applied that are clients or suppliers of dependencies with Assignment applied are owned by the same package that owns the data association referring to the instance of Assignment applied to the dependency.

10.6.6.2 BPMNProperty

Derived properties

None

Constraints

[1] The value of umlProperty must be a property of the activity containing the data store node to which the stereotype is applied, or of an activityClass of a BPMNActivity (or eventClass of a BPMNEvent) applied to an action contained by that activity.

10.6.6.3 DataAssociation

Derived properties

- /sourceRef : ItemAwareElement = self.base_ObjectFlow.source.extension_ ItemAwareElement
- /targetRef : ItemAwareElement = self.base_ObjectFlow.target.extension_ ItemAwareElement
- /transformation : FormalExpression [0..1] = self.base_ObjectFlow.transformation. extension_FormalExpression

Constraints

None

10.6.6.4 DataInput

Derived properties

- /inputSetRefs : InputSet [*] = self.(base_ActivityParameterNode. Parameter or base_ Parameter).parameterSet.extension_InputSet
- /inputSetWithOptional : InputSet [*] = self.(base_ ActivityParameterNode.parameter or base_Parameter, with multiplicity lower = 0).parameterSet.extension_InputSet
- /inputSetWithWhileExecuting : InputSet [*] = self.(base_ActivityParameterNode.parameter or base_Parameter, with isStreaming = true).parameterSet.extension_InputSet
- DataInputs and DataOutputs cannot be applied to the same elements at the same time.

Constraints

[1] Activity parameter nodes with DataInput applied must have parameters with direction = in.

[2] Data inputs can be applied to parameters only on opaque behaviors extended by global task stereotypes.

[3] Object nodes with DataInput applied have upper bound = 1 if isCollection = false, and upperbound = * otherwise.

[4] Parameters with DataInput applied must be on opaque behaviors with a specialization of GlobalTask applied, and vice versa.

10.6.6.5 DataInputAssociation

Derived properties

None

Constraints

[1] DataInputAssociation may only be applied to object flows that have a data store node as source, and an input pin as target.

10.6.6.6 DataObject

Derived properties

None

Constraints

[1] DataObject and DataStoreReference cannot be applied to the same element at the same time.

[2] Object nodes with DataObject applied have upperbound = 1 if isCollection = false, and upperbound = * otherwise.

10.6.6.7 DataObjectReference

Derived properties

None

Constraints

[1] Data store nodes with DataObjectReference applied must have the same type, multiplicity and ordering, and uniqueness as the data store nodes with DataObject applied they refer to.

[2] Object nodes with DataObjectReference applied have upper bound = 1 if isCollection = false, and upperbound = * otherwise.

10.6.6.8 DataOutput

Derived properties

- /outputSetRefs : OutputSet [*] = self.(base_ActivityParameterNode.parameter or base_ Parameter).parameterSet.extension_OutputSet
- /outputSetWithOptional : OutputSet [*] = self.(base_ ActivityParameterNode.parameter or base_Parameter with multiplicity lower = 0).parameterSet.extension_OutputSet
- /outputSetWithWhileExecuting : OutputSet [*] = self.(base_ActivityParameterNode. parameter or base_Parameter with isStreaming = true).parameterSet.extension_OutputSet

Constraints

[1] Activity parameter nodes with DataOutput applied must have parameters with direction = out or return.

[2] DataOutput and DataInput cannot be applied to the same element at the same time.

[3] DataOutput and DataInput can be applied to parameters only on opaque behaviors with a specialization of GlobalTask applied.

[4] Object nodes with DataOutput applied have upper bound = 1 if isCollection = false, and upperbound = * otherwise.

10.6.6.9 DataOutputAssociation

Derived properties

None

Constraints

[1] DataOutputAssociation may only be applied to object flows that have an output pin as source and a data store node as target.

10.6.6.10 DataState

Derived properties

- /itemSubjectRef : ItemDefinition [0..1] = self.base_Class.generalization.extension_ ItemDefinition

Constraints

[1] States with DataState applied that are dataStates of item aware element stereotypes applied to object nodes must be values of the inState property of the object nodes, and vice versa.

10.6.6.11 DataStoreReference

Derived properties

None

Constraints

[1] DataObject and DataStoreReference cannot be applied to the same element at the same time.

[2] Data store nodes with DataStoreReference applied are typed by the class with ItemDefinition applied that directly generalizes the class with DataStore applied that is the data store associated with the DataStoreReference. (self.type = self.extension_DataStoreReference.dataStore.base_ Class.generalization)

10.6.6.12 InputOutputSpecification

Derived properties

- /dataInputs : DataInput [*] = self.(base_Behavior.ownedParameter.(for parameters owned by Activities, opposite(ActivityParameterNode::parameter.))extension_DataInput or self.base_Action./input.extension_DataInput
- /dataOutputs : DataOutput [*] = self.(base_Behavior.ownedParameter.(for parameters owned by Activities, opposite(ActivityParameterNode::parameter.))extension_DataOutput or self.base_Action./ output.extension_DataOutput
- /inputSets : InputSet [1..*] = self.(base_Behavior.ownedParameterSet. extension_InputSet
- /outputSets : OutputSet [1..*] = self.base_Behavior.ownedParameterSet. extension_OutputSet

Constraints

[1] InputOutputSpecification may only be applied to elements that have a specialization of CallableElement or BPMNActivity applied.

10.6.6.13 InputSet

Derived properties

- /dataInputRefs : DataInput [*] = inverse of DataInput::/inputSetRefs
- /optionalInputRefs : DataInput [*] = inverse of DataInput::/inputSetWithOptional

- /whileExecutingInputRefs : DataInput [*] = inverse ofDataInput::/ inputSetWithWhileExecuting

Constraints

[1] Parameters in parameter sets with InputSet applied must all have direction = in.

10.6.6.14 ItemAwareElement

Derived properties

- /itemSubjectRef : ItemDefinition [0..1] = self.base_TypedElement.type. extension_ItemDefinition

Constraints

None

10.6.6.15 OutputSet

Derived properties

- /dataOutputRefs : DataOutput [*] = inverse of DataOutput::/outputSetRefs
- /optionalOutputRefs : DataOutput [*] = inverse of DataOutput::/outputSetWithOptional
- /whileExecutingOutputRefs : DataOutput [*] = inverse DataOutput::/ outputSetWithWhileExecuting

Constraints

[1] Parameters in parameter sets with OutputSet applied must all have direction = out or return.

10.7 LoopCharacteristics

Figure 10.17 illustrates Loop Characteristics.

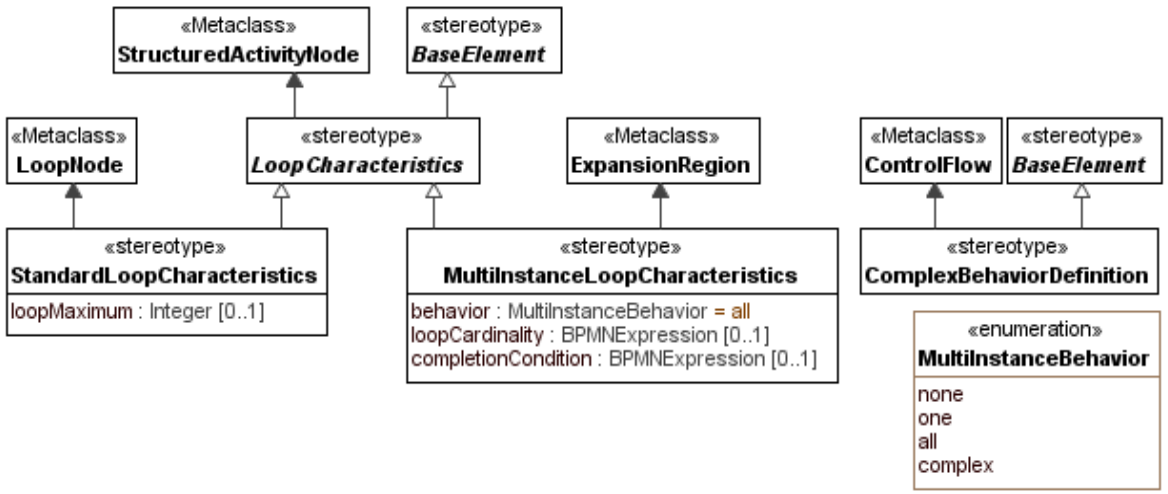

Figure 10.17 - Loop Characteristics

BPMN and UML support looping without depending on flows to create cycles, sometimes called “structured” loops. BPMN and UML support these in two ways, the first for sequential execution (BPMN StandardLoopCharacteristics and UML LoopNode), and the second for parallel execution (BPMN MultiInstanceLoopCharacteristics and UML ExpansionRegion). The profile uses loop nodes and expansion regions containing actions corresponding to the standard or multi-instance looping BPMN Activity. BPMN SequenceFlows linked to the BPMN Activity translate under the profile to control flows linked to the loop node or expansion region.

BPMN StandardLoopCharacteristics adds to BPMN Activities a condition to test before or after each iteration indicating whether to stop looping, and a maximum number of iterations allowed to occur, specified ahead of time as an integer. UML LoopNodes are UML StructuredActions, enabling them to contain other actions to be iterated over, which in the profile is a single one corresponding to the BPMN Activity with standard loop characteristics. Loop nodes supports a condition test before or after each iteration, and the StandardLoopCharacteristics stereotype extends UML LoopNode semantics to remove the loop node’s token when the number of completed iterations is equal to the value of the loopMaximum property, providing a semantics equivalent to BPMN’s.

BPMN MultiInstanceLoopCharacteristics adds to BPMN Activities an indication of whether the executions of the activity happen sequentially, as they do in BPMN StandardLoopCharacteristics, or in parallel. UML ExpansionRegions support this also. They are UML StructuredActions, enabling them to contain other actions to be iterated over, which in the profile is a single action corresponding to the BPMN Activity with multi-instance loop characteristics. BPMN MultiInstanceLoopCharacteristics supports a condition under which the loop will be interrupted, which might happen in the middle of a loop, not just before or after each iteration as in BPMN StandardLoopCharacteristics. The MultiInstanceLoopCharacteristics stereotype extends UML ExpansionRegions with the completionCondition property, and extends UML ExpansionRegion semantics by removing the expansion region’s token when the condition becomes true, providing a semantics equivalent to BPMN’s.

BPMN MultiInstanceLoopCharacteristics also adds to BPMN Activities two ways to specify how many executions of the activity will occur:

1. An expression that is evaluated just before the activity begins to determine how many times to execute it. When the MultiInstanceLoopCharacteristics stereotype is applied with a value for the loopCardinality property, the UML ExpansionRegion has one input ExpansionNode with no incoming or outgoing edges, and its semantics is extended to evaluate the expressions in the loopCardinality property before it begins to determine the size of the collection placed in the input. Expansion regions execute their contents once for every element of the collection, providing a semantics equivalent to BPMN’s.

2. Specifying how many executions of the activity will occur is to accept a collection, and execute the activity once for each element of the collection, providing that element as input to the activity. UML ExpansionRegion supports this with ExpansionNodes used as inputs that accept collections through object flows from actions outside the region, and provides individual elements of the collections through object flows to actions inside the region, executing those actions once for every element of the collection. BPMN MultiInstanceLoopCharacteristics also provides a way for each execution of a BPMN Activity to add an element to a collection that is output when looping is finished. UML ExpansionRegion supports this with ExpansionNodes on ExpansionRegions used as outputs, accepting elements of collections though object flows from actions inside the region, and providing the collections through object flows to actions outside the region when the region is finished.

BPMN MultiInstanceLoopCharacteristics on BPMN Activities can also specify when to throw implicit throw events after iterations are complete, depending on the value of the behavior property:

- all: After each execution of the activity. The profile supports this with a second action in the expansion region, which has the ImplicitThrowEvent stereotype applied, and is executed after the action with BPMNActivity applied.

- one: Only after the first execution of the activity. The profile supports this with a second action in the expansion region, which has the ImplicitThrowEvent stereotype applied. It is executed after the action with BPMNActivity applied, but preceded by a decision node to ensure the second action only occurs on the first execution.
- complex: After executions of the activity only under certain conditions. The profile supports this with additional actions in the expansion region, which have the ImplicitThrowEvent stereotype applied. They are executed in parallel after the action with BPMN Activity applied, but each additional action is preceded by a decision node to ensure it only occurs on under the specified conditions. Control flows in the expansion region have ComplexBehaviorDefinition applied when they are going out of the decision node targeting actions with the ImplicitThrowEvent stereotype applied.
- none: No implicit throw events are thrown. The expansion region contains no actions with the ImplicitThrowEvent stereotype applied.

If the event definitions of the BPMN ImplicitThrowEvents are not specified by the modeler, they are BPMN Signals with the properties specified by BPMN for multi-instance activity instances. UML CallOperationActions with an ImplicitThrowEvent applied can have triggers with SignalEventDefinition applied that refers to a UML Signal with the properties specified by BPMN, provided a semantics equivalent to BPMN.

10.7.1 Derived Properties and Constraints

10.7.1.1 ComplexBehaviorDefinition

Derived properties

- /condition : FormalExpression = self.base_ControlFlow.guard. extension_FormalExpression
- /event : ImplicitThrowEvent [0..1] = self.based_ControlFlow.target. extension_ImplicitThrowEvent

Constraints

[1] ComplexBehaviorDefinition may only be applied to control flows that are in expansion regions with MultiInstanceLoopCharacteristics applied, and that have decision nodes as sources and actions with ImplicitThrowEvent applied as targets.

10.7.1.2 MultiInstanceLoopCharacteristics

Derived properties

- /complexBehaviorDefinition : ComplexBehaviorDefinition [*] = self.base_ExpansionRegion.edge.extension_ComplexBehaviorDefinition
- /inputDataItem : DataInput = self.base_ExpansionRegion.node. input.extension_DataInput for the node with BPMNActivity applied.
- /isSequential : Boolean = (mode = iterative)
- /loopDataInputRef : ItemAwareElement [0..1] = self.base_ ExpansionRegion.inputElement.incoming.source
- /loopDataOutputRef : ItemAwareElement [0..1] = self.base_ ExpansionRegion.outputElement.outgoing.target
- /noneBehaviorEventRef : EventDefinition [0..1] = self.base_ ExpansionRegion.node.eventDefinitions.extension_EventDefinition for the node with ImplicitThrowEvent applied, if any.

- /oneBehaviorEventRef : EventDefinition [0..1] = self.base_ ExpansionRegion.node.eventDefinitions.extension_EventDefinition for the node with ImplicitThrowEvent applied, if any.
- /outputDataItem : DataOutput = self.base_ExpansionRegion.node. output.extension_DataOutput for the node with BPMNActivitity applied.

Constraints

[1] Expansion regions with MultiInstanceLoopCharacteristics applied and an inputElement must contain an object flow from the inputElement to the input pin of the action with BPMNActivity applied contained in the expansion region.

[2] Expansion regions with MultiInstanceLoopCharacteristics applied and an outputElement must contain an object flow to the output element from the output pin of the action it contains with BPMNActivity applied.

[3] Expansion regions with MultiInstanceLoopCharacteristics applied have exactly one inputElement if loopCardinality is empty, otherwise it has no inputElements.

[4] Expansion regions with MultiInstanceLoopCharacteristics applied have one outputElement if the action with BPMNActivity applied it contains has an output, otherwise it has no outputElements.

[5] Expansion regions with MultiInstanceLoopCharacteristics applied must contain one action with BPMNActivity applied, and any other actions must have ImplicitThrowEvent applied.

[6] Expansion regions with MultiInstanceLoopCharacteristics applied must have an iterative or parallel mode.

[7] MultiInstanceLoopCharacteristics may only be applied when Task or SubProcess is also applied.

[8] Opaque expressions must have FormalExpression applied when they are owned by control flows coming into call operation actions with ImplicitThrowEvent applied in expansion regions with ComplexBehaviorDefinition applied.

10.7.1.3 StandardLoopCharacteristics

Derived properties

- /loopCondition : BPMNExpression= self.base_LoopNode.test.value.extension_BPMNExpression
- /testBefore : Boolean = self.base_LoopNode.isTestedFirst

Constraints

[1] StandardLoopCharacteristics may only be applied when Task or SubProcess is also applied.

[2] Loop nodes with StandardLoopCharacteristics applied have an empty setPart, one action in the bodyPart, which has a specialization of BPMNActivity applied, and one value specification action as test, the result output pin of which is the decider of the LoopNode.

10.8 Lanes and Resources

Figure 10.18 illustrates BPMN Lanes and LaneSets.

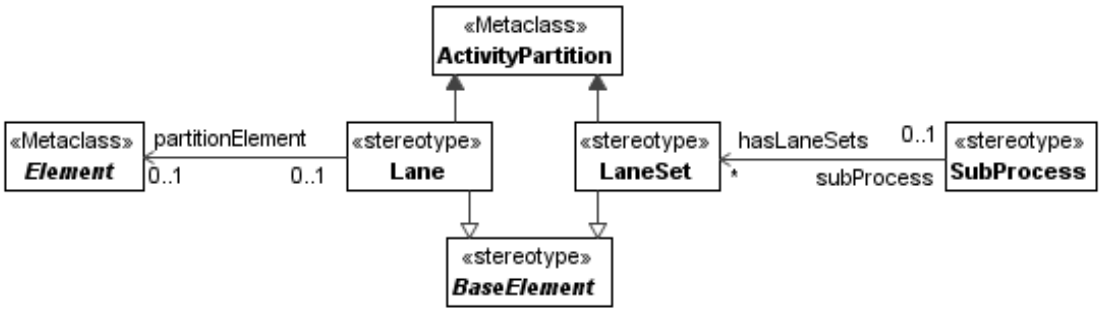

Figure 10.18 - Lanes and LaneSets

BPMN Lanes group elements of processes or subprocesses to express modeler-defined relationships to other parts of the model. BPMN LaneSets group lanes in modeler-defined ways. UML ActivityPartitions do the same for elements of activities, and other activity partitions, except they are only available on activities, not structured activity nodes. The SubProcess stereotype extends UML StructuredActivityNodes with the laneSets property, to provide partitions in structured activity nodes, equivalent to BPMN Lanes in subprocesses. BPMN LaneSets directly under BPMN Subprocesses are owned by the subprocesses, rather than lanes the subprocesses are in. LaneSet stereotypes referred to by SubProcess stereotypes are applied to activity partitions owned by the activity, rather than activity partitions the structured activities are in. BPMN Lanes contain at most one lane set to define sublanes. The LaneSet and Lane stereotypes are applied to the same activity partition on sublanes, to simplify the model.

BPMN Lanes specify other parts of the model that process elements in the lane have a relationship to, which can be owned by the lane or not. UML Activities can refer to other model elements in the same way, but not own them. The Lane stereotype extends UML ActivityPartitions with the partitionElement property to refer to other elements of the model that activity elements in the partition have a relationship to, which are required to be among those that it references in UML.

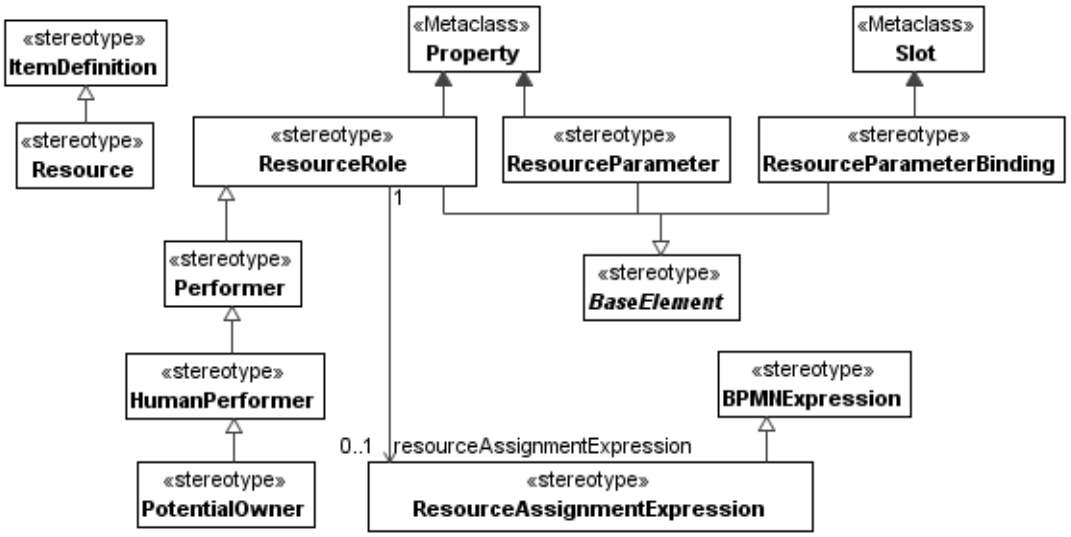

Figure 10.19 - Resources and Resource Roles

The relationships between process elements in a lane and other parts of the model are up to modelers, but they are often relationships to resources, which are expressed with BPMN ResourceRoles. BPMN Resources are the kinds of things that play resource roles. BPMN Processes, GlobalTasks, and Activities can have resource roles. The ResourceRole stereotype identifies UML Properties on activities, opaque behaviors, and activity classes of BPMNActivities (see 10.1.2, 10.2, and 10.6.4), with the type of the property specifying the kinds of resources that can play the role, providing a semantics equivalent to BPMN's. The specializations of the ResourceRole stereotype have application semantics, see the BPMN specification. BPMN ResourceParameters specify properties of resources, while BPMN ResourceParameterBindings specify values for those properties when the resources play particular resource roles. The ResourceParameter stereotype identifies resource parameter properties on UML Classes that have the Resource stereotype applied, providing a semantics equivalent to BPMN's. Resource parameter properties can have UML InstanceSpecifications as default values, where the instance specifications have UML Slots with ResourceParameterBinding applied that specify a value for a resource parameter, as expressed by the /parameterRef and /expression derived properties on the ResourceParameterBinding stereotype, providing a semantics equivalent to BPMN's. BPMN ResourceAssignmentExpression is an expression that deployment platforms can use to identify an individual resource instances at runtime to play resource roles. BPMN does not define how the platform uses this expression, and the profile does not either.

10.8.1 Derived Properties and Constraints

10.8.1.1 Lane

Derived properties

- /childLaneSet : LaneSet = self.base_ActivityPartition.extension_LaneSet
- /flowNodeRefs : FlowNode [*] = self.base_ActivityPartition.node.extension_FlowNode
- /laneSet : LaneSet = self.base_ActivityPartition.superPartition.extension_LaneSet
- /partitionElementRef : BaseElement [0..1] = self.base_ActivityPartition.represents.extension_ BaseElement that are not included in Lane::partitionElement.

Constraints

[1] partitionElement must be included in ActivityPartition.represents.

10.8.1.2 LaneSet

Derived properties

- /flowElementsContainer : FlowElementsContainer [0..1] = inverse of FlowElementsContainer:: /laneSets
- /lanes : Lane [*] = self.base_ActivityPartition.subpartition.extension_Lane
- /parentLane : Lane [*] = self.base_ActivityPartition.extension_Lane

Constraints

[1] Activity partitions with LaneSet applied where the ActivityPartition is owned by an Activity must have isDimension=true.

[2] On Activity partitions with LaneSet applied. If the subprocess property of the lane set has a value, then the inActivity property must have a value.

10.8.1.3 Resource

Derived properties

- /resourceParameters : ResourceParameter [*] = self.base_Class.ownedAttribute.extension_ ResourceParameter, for properties with ResourceParameter applied.

Constraints

None

10.8.1.4 ResourceAssignmentExpression

Derived properties

- /expression : BPMNExpression = self

Constraints

None

10.8.1.5 ResourceParameter

Derived properties

- /isRequired : Boolean = self.base_Property.lower > 0 (lower defined on MultiplicityElement)
- /type : ItemDefinition [0..1] = self.base_Property.type.extension_ItemDefinition (type defined on TypedElement)

Constraints

[1] Properties with ResourceParameter applied must be owned by classes with the Resource stereotype applied.

10.8.1.6 ResourceParameterBinding

Derived properties

- /expression : BPMNExpression = self.base_Slot.value.extension_ BPMNExpression
- /parameterRef : ResourceParameter = self.base_Slot.definingFeature. extension_ResourceParameter

Constraints

[1] Slots with ResourceParameterBinding applied must be owned by instance specifications that are the default values of properties with ResourceRole applied, and that have the type of the property as classifier.

10.8.1.7 ResourceRole

Derived properties

- /process : BPMNProcess [0..1] = self.base_Property.class.extension_BPMNProcess
- /resourceParameterBindings : ResourceParameterBinding [*] = self.base_Property. defaultValue.slot
- /resourceRef : Resource [0..1] = self.base_Property.type.extension_Resource (type defined on TypedElement)

Constraints

None

11 Collaborations

11.1 Collaborations and Conversations

11.1.1 Collaborations

Figure 11.1 illustrates BPMN Collaborations.

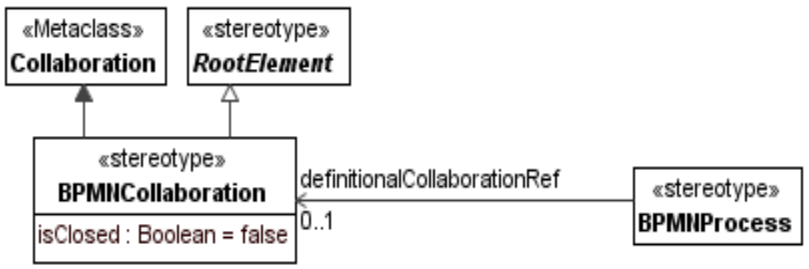

Figure 11.1 - BPMN Collaborations

BPMN Collaborations and UML Collaborations share the same name and purpose. They both help specify interactions (message exchange) between processes or behaviors that can be carried out by entities or objects participating in the collaboration. Neither BPMN nor UML Collaborations capture all interaction details, they focus on message flows between participants (see 11.1.2 and 11.1.3), using grouping and reuse of flows for scaling to complicated interactions (see 11.1.4). The time sequence of message flows and the conditions under which they occur is captured in other parts of the languages (choreography in BPMN, and the behavior models in UML). However, BPMN Processes can be included in collaborations, providing a way to specify time sequence and conditions for flows between participants by using time sequence and conditions for process elements that send and receive messages.

As with processes, when BPMN Collaborations are carried out, it is expected they will follow their models, whereupon they are valid, otherwise they are invalid. The BPMN isClosed property indicates whether message flows not specified in a collaboration can validly occur when the collaboration is carried out. If the value of isClosed is true, then messages flows are limited to those specified in the models, otherwise they are not. The BPMNCollaboration stereotype extends UML Collaboration semantics to use the isClosed property to determine whether collaborations may be validly carried out with message flows not specified in their models, giving a semantics equivalent to BPMN's.

BPMN Processes use collaborations to specify how they interact with other processes or external entities. The same process might appear in multiple collaborations, but no more than one is identified as being included in the definition of each process, as expressed by the BPMN definitionalCollaborationRef property of processes. The BPMNProcess stereotype extends UML Activities with a property of the same name to identify such collaborations.

11.1.2 Participants

Figure 11.2 illustrates BPMN Participants.

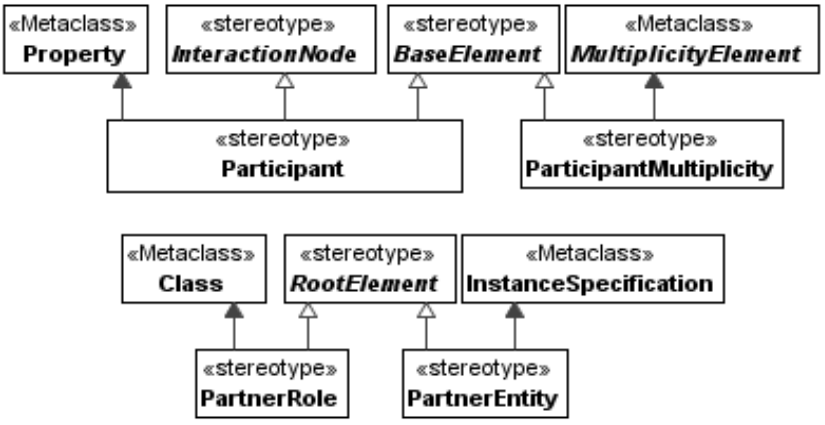

Figure 11.2 - Participants

BPMN Participants are roles in collaborations, rather than things or kinds of things that participate in collaborations, so each BPMN Participant is in exactly one collaboration. UML Collaborations are classifiers that have properties to identify participating things, and UML Properties are always in exactly one classifier. The Participant stereotype indicates which properties of UML Collaborations identify participating things, providing semantics equivalent to BPMN's. BPMN PartnerEntities are individual things, such as WalMart or particular persons, while BPMN PartnerRoles are the kinds of things that play roles, such as buyers or sellers, rather than roles in the collaboration. Partner entities and roles can refer to participants (roles in collaborations), specifying individual things or kinds of things playing those roles in collaborations. The PartnerEntity stereotype constrains UML InstanceSpecification semantics to specify exactly one instance, providing a semantics equivalent to BPMN's. UML Properties can have read-only default values specifying individual things that will be values of the properties, using instance specifications with the PartnerEntity stereotype applied, and they can also have types to specify the kinds of individuals, using classes with PartnerRole stereotype applied, providing semantics equivalent to BPMN's. BPMN PartnerEntities and PartnerRoles are optional for BPMN Participants, as are UML property defaults and types for UML Properties.

BPMN Participants can refer to processes that exchange messages. BPMN Processes appear visually inside participants, but can actually occur within or due to individual things specified by partner entities or partner roles of the participants. In this case, UML Properties with the Participant stereotype applied have read-only default instance specifications with the PartnerEntity stereotype applied and/or types with the PartnerRole stereotype applied, and classifiers of the default instance specifications and/or types of participant properties have UML Activities with the BPMNProcess stereotype applied as their classifier behaviors, providing a semantics equivalent to BPMN's (UML classifier behaviors occur within or due to individual things being classified). BPMN Participants can refer to processes without specifying partner entities or roles in which the processes will occur. In this case, participant properties with the Participant stereotype applied are typed by UML Activities with the BPMNProcess stereotype applied (see 10.1.2 about activities as types). Combined with the constraint that UML Activities with the BPMNProcess stereotype applied are their own classifier behaviors (see 10.1.2), actions can accept operations messages directed to the containing activity, providing a semantics equivalent to BPMN's.

11.1.3 Message Flows

Figure 11.3 illustrates BPMN Message Flows.

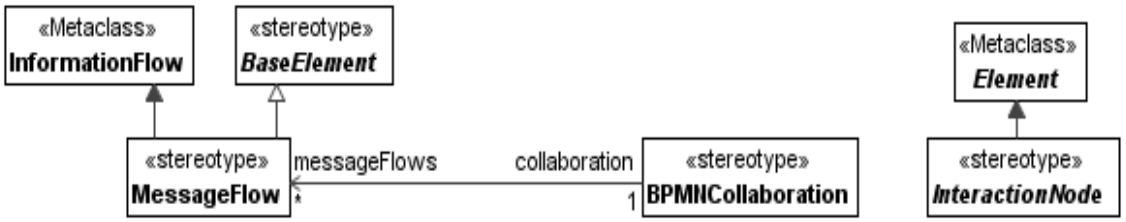

Figure 11.3 - Message Flows

BPMN MessageFlows specify messages transmitted between participants, or elements of processes that send and receive messages. BPMN Messages are the kinds of things that are transmitted along message flows. UML InformationFlows specify the transmission of things between almost any kind of UML element. Contrary to the name, the things transmitted can be anything classifiable, including physical things. The kind of thing transmitted by an information flow is specified with the UML conveyed property. The BPMN Collaboration stereotype extends UML Collaborations to refer to message flows with the messageFlows property. The BPMNMessage stereotype is a specialization of ItemDefinition (see 10.4.1) and specifies the kinds of things that flow on information flows, as expressed by the /messageRef derived property, providing semantics equivalent to BPMN's. The InteractionNode stereotype generalizes elements that can be on the ends of information flows with the MessageFlow stereotype applied (see 10.2, 10.4.1, and 11.1.2).

11.1.4 Conversations

Figure 11.4 illustrates BPMN Conversations.

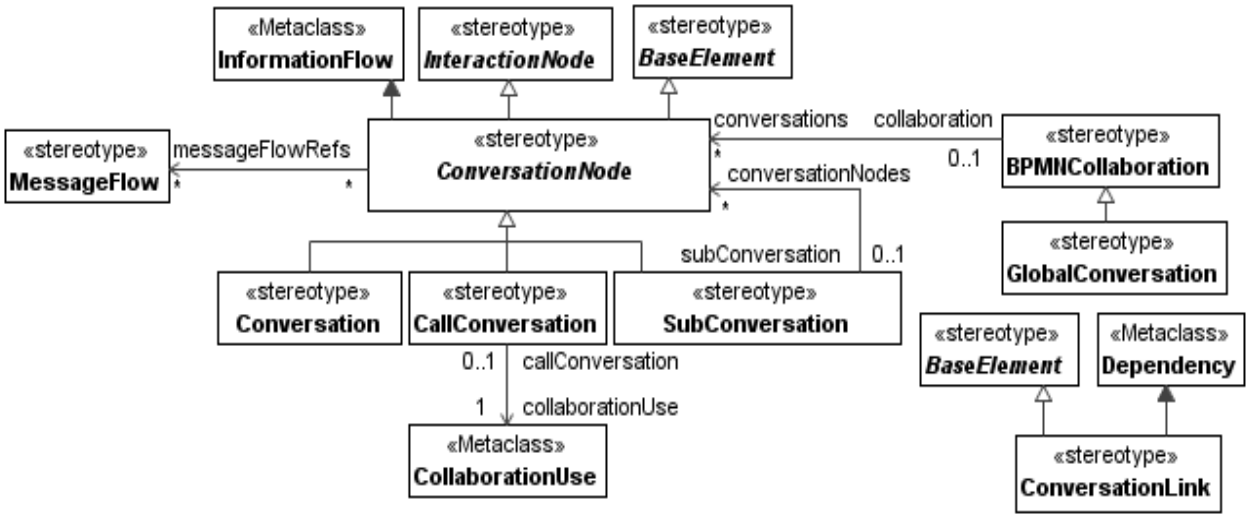

Figure 11.4 - Conversations

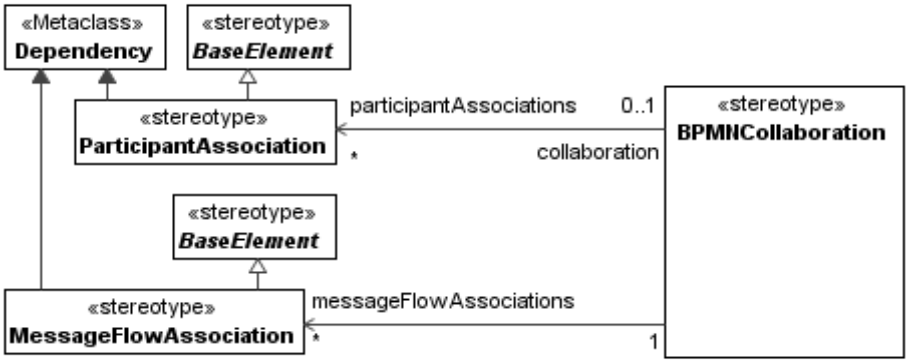

Figure 11.5 - Participant Associations and Message Flow Associations

BPMN ConversationNodes group message flows analogously to BPMN CallActivities and Subprocesses grouping process flow elements. They can be shown in an expanded view exposing their grouped message flows, or in a collapsed view with BPMN ConversationLinks between the nodes and participants or process elements in participants. The ConversationNode stereotype extends UML InformationFlows to refer to grouped message flows. The ConversationLink stereotype extends UML Dependency with sources that are information flows that have the ConversationNode stereotype applied, targets that are the sources and targets of the information flows, and one source and target per dependency.

The three kinds of conversation for grouping message flows in BPMN Collaborations are analogous to the kinds of activities in processes for grouping other activities (BPMN CallActivities, Task, and Subprocesses):

1. BPMN CallConversations reuse other collaborations, using BPMN ParticipantAssociations to link participants in the "caller" with participants in the reused collaboration. UML Collaboration does the same with UML CollaborationUse, linking participants by dependencies identified by the collaboration use as role bindings. BPMN does not give a semantics to associating participants between collaborations, but it is typically read as requiring the individuals acting as the associated participants to be the same when the calling collaboration is carried out. UML collaboration role bindings have an equivalent semantics. BPMN MessageFlowAssociations link message flows in collaborations to message flows in reused collaborations, indicating that the transfer of items along the associated message flows will be the same when the calling collaboration is carried out. The BPMN Collaboration stereotype extends UML Collaborations with a messageAssociations property to refer to message flow associations for this purpose.

2. The Conversation stereotype extends UML InformationFlows with a messageFlowRefs property to identify the message flows of the containing collaboration, providing a grouping construct equivalent to BPMN's.

3. BPMN SubConversations group message flows as conversations do, as well as containing any of the three kinds of conversation. The SubConversation stereotype extends UML InformationFlows as the Conversation stereotype does, and also with a conversationNodes property to refer to any of the other three kinds of conversation, providing a construct equivalent to BPMN's.

BPMN Conversations and SubConversations group message flows owned by the containing collaboration, possibly sharing message flows across conversations.

When a BPMN Process in a collaboration calls another process, the participants of the caller's collaboration are associated with participants in the definitional collaboration of the reused process. The BPMNCollaboration stereotype extends UML Collaborations with a participantAssociations property to refer to participant associations for this purpose.

BPMN does not give a semantics to the three kinds of conversation, but they are typically read as specifying that the message flows in them, or in collaborations they reuse, occur when the conversations do. The same message flow is in multiple conversations is typically read as specifying a single transmission occurs even though there are two conversations. The profile provides equivalent semantics, either through UML CollaborationUse, or by extension of UML InformationFlows.

11.1.5 Derived Properties and Constraints

11.1.5.1 CallConversation

Derived properties

- /calledCollaborationRef : BPMNCollaboration [0.1] = self.collaborationUse.type. extension_BPMNCollaboration
- /participantAssociations : ParticipantAssociations [*] = self.collaborationUse. roleBinding.extension_ParticipantAssociation

Constraints

[1] Collaboration uses with CallConversation applied must be collaborationUses of the collaboration with BPMNCollaboration applied that has the call conversation included in its conversations, and vice versa.

11.1.5.2 Collaboration

Derived properties

- /artifact : Artifact [*] = self.base_Collaboration.ownedComment.extension_ TextAnnotation
- /conversationLinks : ConversationLinks [*] = self.conversations.base_InformationFlow. clientDependency.extension_ConversationLink
- /participants : Participant [*] = self.base_Collaboration.ownedAttribute.extension_ Participant (ownedAttribute defined on Class)

Constraints

None

11.1.5.3 ConversationLink

Derived properties

- /collaboration : Collaboration = inverse of Collaboration::/conversationLinks
- /sourceRef : InteractionNode = self.base_ Dependency.client.extension_ InteractionNode
- /targetRef : InteractionNode = self.base_Dependency.target.extension_ InteractionNode

Constraints

[1] Dependencies with ConversationLink applied must be owned by the innermost package containing the collaboration with BPMNCollaboration applied that has a conversation node included in its conversations that is applied to the client of the dependency.

[2] Dependencies with the ConversationLink applied must have exactly one client and one supplier, where the client must have ConversationNode applied and the supplier must have a specialization of InteractionNode applied.

[3] The targetRefs of conversation links that have the same sourceRef must all be different and collectively be the same set as the union of self.base_InformationFlow.informationSource. extension_InteractionNode and self.base_InformationFlow.informationTarget. extension_InteractionNode.

11.1.5.4 ConversationNode

Derived properties

- /participantRefs : Participant [2..*] = self.base_InformationFlow. informationSource and informationTarget.extension_Participant or extension_Participant of the collaboration role containing the information source or target.

Constraints

[1] Information flows with ConversationNode applied must be owned by the innermost package containing the collaboration with BPMNCollaboration applied that has the conversation node included in its conversations.

11.1.5.5 InteractionNode

Derived properties

- /incomingConversationLinks : ConversationLink [*] = inverse of ConversationLink:: targetRef
- /outgoingConversationLinks : ConversationLink [*] = inverse of ConversationLink:: /sourceRef

Constraints

None

11.1.5.6 MessageFlow

Derived properties

- /messageRef : BPMNMessage [0..1] = self.base_InformationFlow.conveyed.extension_ BPMNMessage
- /sourceRef : InteractionNode = self.base_InformationFlow.informationSource. extension_InteractionNode
- /targetRef : InteractionNode = self.base_InformationFlow.informationTarget. extension_InteractionNode

Constraints

None

11.1.5.7 MessageFlowAssociation

Derived properties

- /innerMessageFlowRef : MessageFlow = self.base_Dependency.supplier. extension_MessageFlow

Constraints

[1] Dependencies with MessageFlowAssociation applied must be owned by the innermost package containing the collaboration with BPMNCollaboration applied that has the message flow association included in its messageFlowRefs.

[2] Dependencies with MessageFlowAssociation applied must have exactly one client and one supplier, both of which must have MessageFlow applied.

11.1.5.8 Participant

Derived properties

- /interfaceRefs : BPMNInterface [*] = self.base_Property.type. interfaceRealization.contract.extension_BPMNInterface (interfaceRealization defined on BehavioredClassifier) if there a type, otherwise, if there is a default value, self.base_Property.defaultValue.classifier.interfaceRealization.contract. extension_BPMNInterface.
- /participantMultiplicity : Participant [0..1] = self.base_Property.extension_ ParticipantMultiplicity
- /partnerEntityRef : PartnerEntity [*] = self.base_Property.defaultValue.instance. extension_PartnerEntity
- /partnerRoleRef : PartnerRole [*] = self.base_Property.type.extension_ PartnerRole
- /processRef : BPMNProcess [0..1] = self.base_Property.type.extension_BPMNProcess if type is an activity, or self.base_Property.type.classifierBehavior.extension_BPMNProcess if type is a behaviored classifier that is not an activity, or self.base_Property.defaultValue.classifier. classifierBehavior.extension_BPMNProcess if there is a default value.

Constraints

[1] Properties with Participant applied are read-only when they have a default value that is an instance specification with PartnerEntity applied.

[2] Properties with Participant applied cannot be typed by an activity with BPMNProcess applied and have a default value that is an instance specification with a classifier having a classifier behavior that is an activity with BPMNProcess applied.

[3] Properties with Participant applied must be ownedAttributes of collaborations with BPMNCollaboration applied.

11.1.5.9 ParticipantAssociation

Derived properties

- /innerParticipantRef : Participant = self.base_Dependency.supplier.extension_ Participant
- /outerParticipantRef : Participant = self.base_Dependency.client.extension_ Participant

Constraints

[1] Dependencies with ParticipantAssociation applied must be owned by the innermost package containing the collaboration with BPMNCollaboration applied that has the ParticipantAssociation instance applied to the dependency included in its participantAssociations, or the innermost package containing the collaboration use owning the roleBinding to which the ParticipantAssociation is applied.

[2] Dependencies with ParticipantAssociation applied must have exactly one client and one supplier, both of which must have Participant applied.

11.1.5.10 ParticipantMultiplicity

Derived properties

- /maximum : Integer = self.base_MultiplicityElement.upper the value is an integer, empty otherwise.
- /minimum : Integer = self.base_MultiplicityElement.lower

Constraints

[1] ParticipantMultiplicity may only applied to elements that have Participant applied.

11.1.5.11 PartnerEntity

Derived properties

- /participantRef : Participant [*] = inverse of Participant::/partnerEntityRef

Constraints

None

11.1.5.12 PartnerRole

Derived properties

- /participantRef : Participant [*] = inverse of Participant::/partnerRoleRef

Constraints

None

11.2 Operations, Interfaces, and Related Tasks

11.2.1 Operations and Interfaces

Figure 11.6 illustrates BPMN Operations and Interfaces.

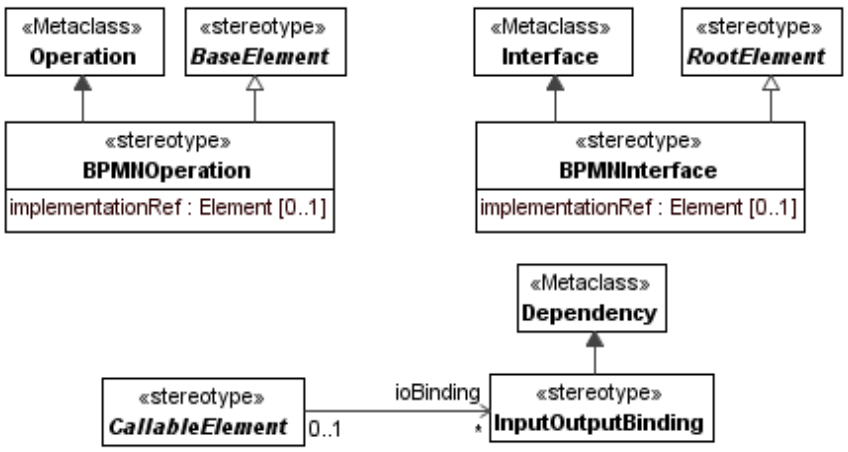

Figure 11.6 - Operations and Interfaces

BPMN and UML Collaborations are typically used in conjunction with operations and interfaces, which specify capabilities that collaborators offer to each other. BPMN and UML Operations and Interfaces share the same names and purposes, though they vary in how widely they can be applied in each language. Operations in both languages specify that something can be done, without saying exactly how. BPMN and UML Interfaces group operations that are typically used together, and then the operations are supported together by elements supporting interfaces.

BPMN and UML Operations can require certain kinds of things be available before the operation can start, and commit to provide certain kinds of things when the operation is finished (BPMN Operations require something to be available before starting, while UML is not restricted this way). BPMN Operations use messages to specify the things they need to start and the things provided when finished, with exactly one message to start, and optionally one message provided on completion, while UML Operations are not restricted this way. The BPMNOperation stereotype constrains parameters of UML Operations to be typed by classes with the BPMNMessage stereotype applied, and to have exactly one input parameter and at most one output, to match BPMN restrictions.

Typical operation implementation languages do not support input and output sets (see 10.6.3 about input and output sets). BPMN InputOutputBindings specify a separate operation to handle each combination of input and output set that might occur for any particular call activity instance, for called elements that have operations and multiple input sets or output sets (see 10.1.2 and 10.2 about callable elements, call activities, and activity instances, and below about specifying operations for callable elements). UML Dependencies with InputOutputBinding applied have two clients, one with InputSet applied and one with OutputSet applied, and one client, which has BPMNOperation applied. UML Operations with BPMNOperation applied supported by UML Behaviors with a callable element stereotype applied can be invoked by CallOperationActions with CallActivity applied targeting the behaviors (as classes), pins typed by a class with BPMNMessage applied, and one argument pin and at most one result pin.

BPMN Operations are only specified within interfaces, while UML Operations can be specified in interfaces and classes. BPMN Interfaces can be supported by processes, global tasks, and participants. BPMN Participants referring to (visually containing) processes might have interfaces supported on the participants, processes, or both. BPMN PartnerEntities and PartnerRoles do not support interfaces, but can refer to participants that do. UML Interfaces can be supported by behaviored classifiers, including activities and opaque behaviors as extended by BPMNProcess and global task stereotypes, and by classes, but not by properties, except indirectly through their default values and types (see 11.1.2 about property typing, default values, and classifiers in the following). UML Properties with the Participant stereotype applied:

- with no links to elements corresponding to processes, partner entities, or partner roles, can be typed by a single interface, indicating the individual things playing that role in the collaboration support the interface, or can be typed by a class with no features supporting multiple interfaces, indicating individual things playing that role in the collaboration support those interfaces.
- with no links to elements corresponding to partner entities or partner roles can be typed by UML Activities with the BPMNProcess stereotype applied and supporting interfaces with the BPMNInterface stereotype applied. Combined with the constraint that UML Activities with the BPMNProcess stereotype applied are their own classifier behaviors, actions can accept messages arriving via operations directed to the containing activity, providing semantics equivalent to BPMN's.
- with links corresponding to processes and partner roles, can be typed by classes with the PartnerRole stereotype applied that support interfaces with the BPMNInterface stereotype applied and having UML Activities as classifier behaviors.
- can have read-only default instance specifications with the PartnerEntity stereotype applied and/or types with the PartnerRole stereotype applied, and classifiers of the default instance specifications and/or types of participant properties supporting interfaces with the BPMNInterface stereotype applied.

The implementationRef property of the BPMN Interface and Operation identify elements modeled outside BPMN, but do not constrain them. The BPMNInterface and BPMNOperation stereotypes extend their base classes with this property, with implementation semantics, see the BPMN specification.

11.2.2 Related Tasks

Figure 11.7 illustrates BPMN Related Tasks.

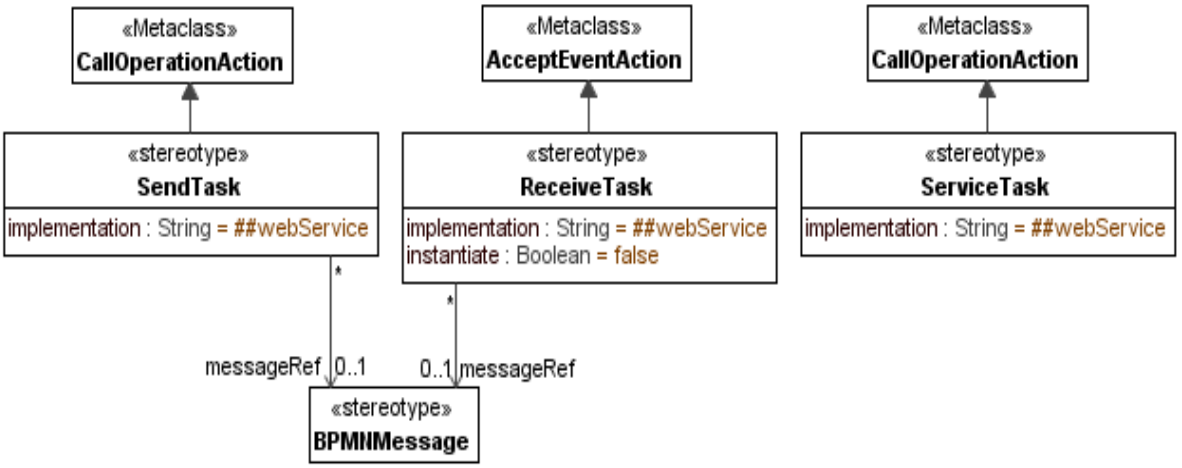

Figure 11.7 - Related Tasks

BPMN SendTasks, ServiceTasks, and ThrowEvents with message event definitions (see 10.2 and 10.4.2) can call operations on things outside the containing process, typically other participants in collaborations. BPMN SendTasks and throw message events can specify outgoing messages without specifying operations, or vice versa, while BPMN ServiceTasks can only specify outgoing messages via operations. This is reflected in the profile by the messageRef associations from the SendTask and MessageEventDefinition stereotypes, but not the ServiceTask stereotype. The participants receiving operation calls are identified by message flows going out of the send task, service task, or throw event. UML CallOperationActions with SendTask, ServiceTask, or throw event stereotypes applied that have message event definitions require their targets to be values of properties with Participant applied, as determined by outgoing information flows with MessageFlow stereotype, if any, their pins typed by a class with BPMNMessage applied, and one argument pin and at most one result pin, providing semantics equivalent to BPMN's. BPMN SendTasks and throw message events do not wait for messages to be received wherever they are sent before completing (if they invoke operations with out messages, those messages are received by catch events or receive tasks later in the process, in containing processes, or other processes entirely). BPMN ServiceTasks will wait for a reply message to arrive before completing only if the operation they invoke will provide a message when finished. UML CallOperationActions can wait for a reply or not, regardless of whether or not the operation is supposed to send one. The SendTask and throw event stereotypes with message event definitions require call operation actions to be asynchronous, and the ServiceTask stereotype does also, except when the invoked operation has a return parameter, whereupon the call operation action is synchronous, providing semantics equivalent to BPMN's.

BPMN ReceiveTasks and catch events with message event definitions (see 10.2 and 10.4.3) accept operations calls from things outside the containing process, typically other participants in collaborations. The participants sending operation calls are determined by message flows into the receive task or throw event. The ReceiveTask and catch event stereotypes extend UML AcceptEventAction semantics by removing tokens accepted when the participants are not specified by the

incoming message flows, and returning the event to the pool from which it was drawn. BPMN ReceiveTasks with message event definitions can specify incoming messages without specifying operations, and vice-versa, which is reflected in the profile by the messageRef properties of the ReceiveTask and MessageEventDefinition stereotypes. The BPMN instantiate property on receive tasks indicates whether new process instances are created when messages of particular kinds arrive (see 10.1.2 about process instances). BPMNProcess stereotypes with a value of true for the instantiate property extend UML Activity semantics to be equivalent to BPMN's by instantiating activities that directly contain accept event actions with ReceiveTasks applied and no incoming edges when the messages arrive.

The implementation properties on BPMN SendTask, ReceiveTask, and ServiceTask identify implementation technologies outside of BPMN. The SendTask, ReceiveTask, and ServiceTask stereotypes extend their base classes with this property, with implementation semantics, see the BPMN specification.

11.2.3 Derived Properties and Constraints

11.2.3.1 BPMNInterface

Derived properties

- /callableElements : CallableElement [*] = self.base_Interface.interfaceRealization. implementingClassifier.extension_CallableElement.
- /operations : BPMNOperation [*] = self.base_Interface.ownedOperation.extension_ BPMNOperation

Constraints

None

11.2.3.2 BPMNOperation

Derived properties

- /errorRef : Error [*] = self.base_Operation.raisedException (raisedException defined on BehavioralFeature)
- /inMessageRef : BPMNMessage = instance of BPMNMessage applied to the type of the first parameter in self.base_Operation.ownedParameter with direction=in.
- ioBinding [*] = inverse of InputOutputBinding::/operationRef.
- outMessageRef [0..1]: BPMNMessage = instance of BPMNMessage applied to the type of the first parameter in self.base_Operation.ownedParameter with direction=return.

Constraints

[1] Operations with BPMNOperation applied must have either one or two parameters, and no parameters with direction=inout or direction=out.

[2] Operations with BPMNOperation applied must have no more than one parameter with direction=return, and the parameter must be typed by a class with BPMNMessage applied, and have multiplicity 1.

[3] Operations with BPMNOperation applied must have one parameter with direction=in, and the parameter must be typed by a class with BPMNMessage applied, and must have multiplicity 1.

11.2.3.3 InputOutputBinding

Derived properties

- inputDataRef = self.base_InputOutputBinding.client.extension_InputSet
- operationRef = self.base_InputOutputBinding.supplier.extension_ BPMNOperation
- outputDataRef = self.base_InputOutputBinding.client.extension_OutputSet

Constraints

[1] Dependencies with InputOutputBinding applied must be owned by the innermost package containing the element with a callable element applied that has the input output binding included in its ioBindings.

[2] Dependencies with the InputOutputBinding applied must have exactly two clients and one supplier, and one of the clients must have InputSet applied, while the other has OutputSet applied, and the supplier must have BPMNOperation applied.

11.2.3.4 ReceiveTask

Derived properties

- /operationRef : BPMNOperation [0..1] = self.base_AcceptEventAction.trigger.operation. extension_BPMNOperation

Constraints

[1] AcceptEventActions with ReceiveTask applied must have call events as triggers.

11.2.3.5 SendTask

Derived properties

- /operationRef : BPMNOperation [0..1] = self.base_CallOperationAction.operation. extension_BPMNOperation
- /operationRef : BPMNOperation [0..1] = self.base_CallOperationAction.operation. extension_BPMNOperation

Constraints

[1] CallOperationActions with SendTask applied must have isSynchronous = false, except when ve a value for the /outMessageRef property, whereupon isSynchronous = true.

11.2.3.6 ServiceTask

Derived properties

- /operationRef : BPMNOperation [0..1] = self.base_CallOperationAction.operation. extension_BPMNOperation

Constraints

[1] Call operation actions with ServiceTask applied must have isSynchronous = false, except when service tasks have a value for the /outMessageRef property, whereupon isSynchronous = true.

11.3 Correlation

Figure 11.8 illustrates BPMN Correlation.

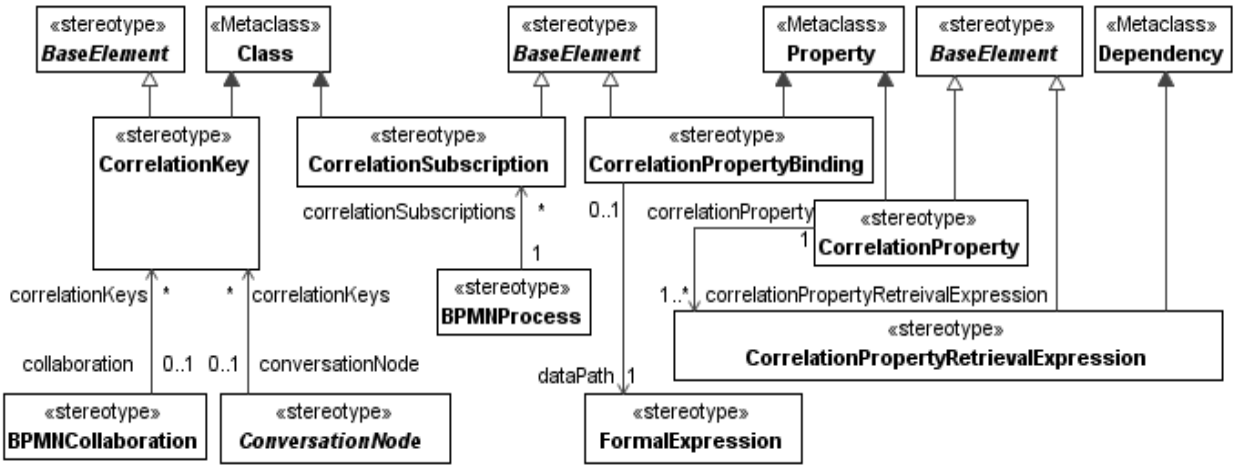

Figure 11.8 - Correlation

Messages coming into participants have correlation information to determine which process instances they should be routed to (see 10.1.2 about process instances). BPMN Collaborations and the three kinds of conversation use BPMN CorrelationKeys to specify which BPMN CorrelationProperties of incoming messages they match process instances against. BPMN does not support message properties, so properties of correlation keys have BPMN CorrelationPropertyRetrievalExpressions to specify how to extract property values from messages. The CorrelationKey and BPMNMessage stereotypes can specify properties because they are based on UML Class, but properties can only be shared between classes via generalization. Classes with CorrelationKey and BPMNMessage applied must form a taxonomy for property sharing, or properties must be matched by name. Instances of UML Classes with the BPMNMessage stereotype applied that are sent to values of properties with the Participant stereotype applied, are routed to instances of activities with the BPMNProcess stereotype applied according to correlation semantics equivalent to BPMN's. Mapping from UML Classes to message formats is usually part of a model-driven architecture for generating platform-specific technology, but CorrelationProperty stereotypes applied to UML Properties can refer to CorrelationPropertyRetrievalExpression stereotypes applied to UML Dependencies that have as clients OpaqueExpressions with the FormalExpression stereotype applied, and as suppliers Classes with the BPMNMessage stereotype applied to provide syntax equivalent to BPMN's, and to guide platform-specific generators.

Incoming messages supply initial values for correlation properties to match against messages arriving later for each process instance, and processes can also specify values to match against incoming messages. BPMN processes can have BPMN CorrelationSubscriptions with BPMN CorrelationPropertyBindings specifying expressions that determine property values for matching against particular keys. The CorrelationSubscription stereotype is based on UML Class and these classes are generalized by other classes with the CorrelationKey stereotype applied. Subscription classes redefine inherited correlation key properties to apply CorrelationPropertyBinding stereotypes. These stereotype instances have dataPath properties specifying expressions used to get information from activity instances to match against properties of incoming messages (where the activity instances are of activities with the BPMNProcess stereotype applied that refers to instances of the CorrelationSubscription stereotype). Incoming messages are routed to matching activity instances according to correlation semantics equivalent to BPMN's.

11.3.1 Derived Properties and Constraints

11.3.1.1 CorrelationProperty

Derived properties

None

Constraints

[1] Correlation properties with the same name that do not inherit from each other have copies of each others' correlationPropertyRetrievalExpressions.

[2] Properties with CorrelationProperty applied must be ownedAttributes of classes with CorrelationKey applied.

[3] The types of properties with CorrelationProperty applied must have ItemDefinition applied.

11.3.1.2 CorrelationPropertyBinding

Derived properties

None

Constraints

[1] Opaque expressions with FormalExpression applied that are dataPaths of correlation property bindings are owned by the same package that owns the correlation property binding.

[2] Opaque expressions with FormalExpression applied must not be dataPaths of more than one correlation property binding.

[3] Properties with CorrelationPropertyBinding applied must be ownedAttributes of classes with CorrelationSubscription applied, and must redefine ownedAttributes of classes with CorrelationKey applied that are generalizations of the class owning the property with CorrelationPropertyBinding applied.

11.3.1.3 CorrelationPropertyRetrievalExpression

Derived properties

None

Constraints

[1] Dependencies with CorrelationPropertyRetrievalExpression applied must be owned by the innermost package containing the property with CorrelationProperty applied that has the correlation property retrieval expression included in its correlationPropertyRetrievalExpression.

[2] Dependencies with CorrelationPropertyRetrievalExpression applied must have exactly one client and one supplier, where the client must have FormalExpression applied and the supplier must have BPMNMessage applied.

[3] Opaque expressions with FormalExpression applied must not be suppliers of more than one dependency with CorrelationPropertyRetrievalExpression applied.

[4] Opaque expressions with FormalExpression applied that are suppliers of dependencies with CorrelationPropertyRetrievalExpression applied are owned by the same package that owns the correlation property retrieval expression.

11.3.1.4 CorrelationSubscription

Derived properties

None

Constraints

[1] Classes with CorrelationSubscription applied must be direct specializations of classes with CorrelationKey applied.