UML Profile for BPMN 2 Processes

Date: August 19, 2011

Revised submission in response to UML Profile for BPMN Processes RFP (ad/10-06-01)
OMG Document Number: bmi/2011-08-04

Submitted by:
Axway
MEGA International
Model Driven Solutions
No Magic
Sparx Systems

Supported by:
Atego
Computer Sciences Corporation
oose Innovative Informatik GmbH
Softeam
The MITRE Corporation
U.S. Department of Defense
U.S. National Institute of Standards and Technology

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 page http://www.omg.org, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.htm).

Preface

About the Object Management Group

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. A catalog of all OMG Specifications is available from the OMG website at:
http://www.omg.org/technology/documents/spec_catalog.htm.
Specifications within the catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM).

Platform Specific Model and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

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. (as of January 16, 2006) at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
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 section 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: Italic text represents names defined in the specification or the name of a document, specification, or other publication.

0 Submission Material

0.1 Submission Preface

This section contains information specific to the OMG submission process and is not part of the proposed specification. The proposed specification starts with section 1. All sections are normative unless explicitly marked as informative.

The design rationale for the submission is presented in Section 1.

0.2 Submission Team

0.2.1 Submitters

Sylvain Astier
Axway
sastier@axway.com

Antoine Lonjon
META International
antoine.lonjon@mega.com

Cory Casanave
Model Driven Solutions
ed-s@modeldriven.com

Edita Mileviciene
No Magic
edita.mileviciene@nomagic.com

Sam Mancarella
Sparx Systems
sam.mancarella@sparxsystems.com.au

0.2.2 Supporters

Matthew Hause
matthew.hause@atego.com

James Odell
Computer Sciences Corporation
email@jamesodell.com

Tim Weilkiens
oose Innovative Informatik GmbH
tim.weilkiens@oose.de

Philippe Desfray
Softeam
philippe.desfray@softeam.fr

Fatma Dandashi
The MITRE Corporation
dandashi@mitre.org

Leonard Levine, Walter Okon
U.S. Department of Defense
leonard.levine@disa.mil, walt.okon@osd.mil

Conrad Bock
U.S. National Institute of Standards and Technology
conrad.bock@nist.gov

0.3 Resolution of Requirements

0.3.1 Mandatory Requirements

Requirement	Resolution
6.5.1 UML Profile for BPMN v2.0 Processes	
Submissions shall define a UML profile for BPMN that covers the Process Modeling and Process Execution conformance types, including the Descriptive, Analytic, and Common Executable conformance sub-classes. The defined profile shall:	This is addressed starting in Section 1.
1) use the semantics defined in BPMN 2,	The submission uses BPMN 2 semantics as the target of the mapping required in the next item.
2) provide a mapping between BPMN 2 semantics and the profiled UML semantics	The submission uses UML stereotypes to extend UML metamodel elements that have the equivalent semantics as the BPMN 2 element corresponding to the stereotype, or extends the corresponding UML elements with additional semantics to achieve this equivalence, or in a few cases, changes the UML semantics in limited ways to achieve this equivalence. Equivalent semantics means the way businesses function when following BPMN process or collaboration diagrams will be same whether the diagrams are captured using the BPMN metamodel or the profiled UML metamodel.
6.5.2 XSLT transformation between UML and BPMN process models.	
Submissions shall define an XSLT transform that transforms an instance of the BPMN v2.0 XSD to an XMI document that conforms to the UML Profile for BPMN v2.0, and shall also define the inverse XSLT transformation.	This will be defined after a UML XSD is adopted at OMG.
6.5.3 QVT transformation between UML and BPMN process models.	
Submissions shall define a QVT transformation where the source is BPMN v 2.0 and the target is the UML Profile for BPMN 2, and shall also define the inverse QVT transformation.	Example mappings are given in this submission. The complete mapping will be given in a later submission.
6.5.4 Submissions shall ensure that the XSLT and QVT transformations are consistent with each other, and with the profile.	See response to 6.5.2.
6.5.5 Integration into the rest of UML. For the portions of the UML metamodel that the profile is based on, submissions shall explain how the profile affects the use of those portions with the rest of UML.	The profile makes the fewest changes to UML semantics needed to ensure equivalence with BPMN semantics. The effect of the changes on the rest of UML is minimal, and will be covered in a later submission.

0.3.2 Optional Requirements

Requirement	Resolution
6.6.1 UPDM - UML profile for BPMN. A UPDM-BPMN mapping table is defined in UPDM v 2.0. The proposed profile may be consistent with this mapping table.	This requirement will be considered when UPDM contains a UPDM-BPMN mapping table.

0.3.3 Resolution of Discussion Items

Discussion Item	Resolution
6.7.1 Loss of information in transformations. Submissions shall discuss whether the transformations that they define result in loss of information and if so, the profile shall document that loss of information precisely and discuss how this loss of information can be managed.	This will be discussed in a later submission.
6.7.2 Semantics. Submissions shall discuss where UML users might expect a different semantics from BPMN process diagrams than similar diagrams in UML, and how that difference can be managed.	Any differences between the semantics of the UML metamodel elements and the semantics of the extensions is described for each extension, with guidelines on how to manage it.
6.7.3 Traceability from BPMN to UPDM 2.0 terminology. Submissions shall discuss whether the profile that they define results in deviation from the UPDM-BPMN mapping table defined in UPDM v 2.0. If so, the profile shall document that deviation precisely and discuss how this deviation can be mitigated.	This will be discussed when UPDM contains a UPDM-BPMN mapping table.
6.7.4 Traceability from BPMN process to SoaML terminology. Submissions shall discuss whether the profile that they define affects the relationship of BPMN and SoaML. If so, the profile shall document the effect precisely and discuss how negative effects can be mitigated.	There is no standard relationship between BPMN and soaML that could be affected by the adoption of this submission. In addition, the submission does not change BPMN or UML (on which soaML is based), so has no effect on the relationship between BPMN and soaML.

0.4 Evaluation Criteria

Criterion	Comment
Completeness and accuracy of the mapping between BPMN 2 semantics and the profiled UML semantics.	This will be evaluated when UPDM contains a mapping table.
The extent to which the UML profile enables models that use the profile to be visualized with BPMN process notation.	The profile supports the concepts of BPMN processes needed to use BPMN process notation.
The extent to which the transformations preserve information.	This will be evaluated in a later submission.
Clarity of the proposed specification for ease of reviewing its correctness and the purpose of implementing conforming modeling tools as discussed in sections 4.9.2, 5.1.4, and 5.2.4.	This will be evaluated in a later submission.
Ability to be reused within other profiles such as UPDM. Discuss the level of integration/linking between this and other UML extension profiles, such as UPDM, SoaML, SysML.	See the response to requirement 6.5.5.
If optional requirement 6.6.1 has been addressed, the extent to which the proposed profile is	See response to requirement 6.6.1.

consistent with a UML-BPMN mapping table defined in UPDM v 2.0.	
The degree to which other OMG standards are used as the basis of the specification.	The submission uses the standards relevant to the topic (BPMN, UML, and QVT).

0.5 Proof of Concept

The submitters of this specification have extensive experience in building graphic intensive software tools. This specification incorporates experience the submitters have gained so far in implementing UML Profiles for BPMN, and includes proven design principles. Implementations for this specification have started and will continue in parallel with the submission process and finalization.

0.6 Changes to Adopted OMG Specifications

This specification proposes no changes to adopted OMG specifications.

1 Scope

This specification enables modelers to use BPMN 2 process and collaboration notation to visualize UML activity and collaboration models. It extends the UML metamodel with a UML profile, including constraints, extensions, and modifications of UML semantics to ensure equivalence to BPMN semantics. Equivalent semantics ensures the way businesses function when following BPMN process or collaboration diagrams will be same whether the diagrams are captured using the BPMN metamodel or the profiled UML metamodel, as illustrated in Figure 1. The scope of the profile includes elements and relationships in BPMN that cover Process Modeling and Process Execution conformance types, including the Descriptive, Analytic, and Common Executable conformance sub-classes. It does not address other conformance types. The specification does not change BPMN notation, metamodel, or semantics.

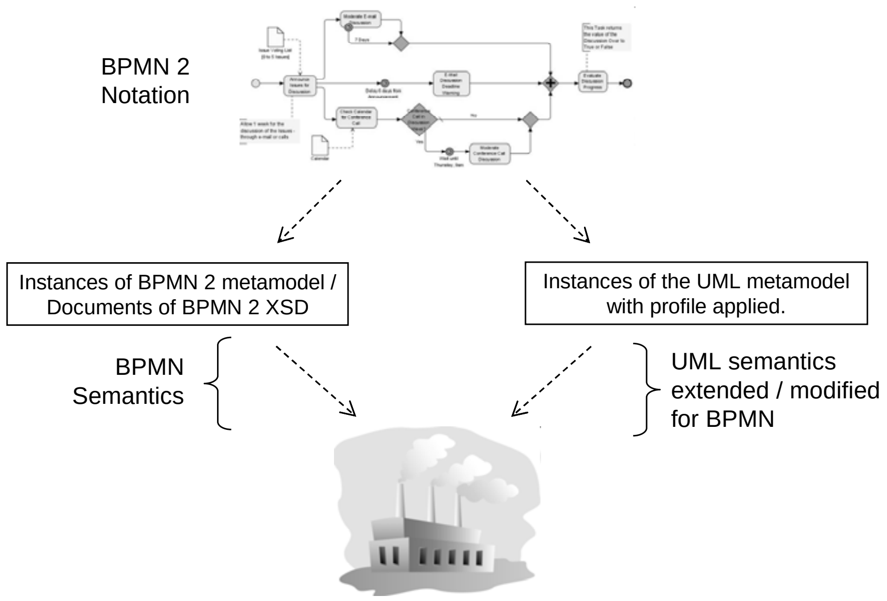

BPMN 2
Notation
Instances of BPMN 2 metamodel / Documents of BPMN 2 XSD
Instances of the UML metamodel with profile applied.
BPMN
Semantics
UML semantics
extended / modified
for BPMN

Figure 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. Stereotype bases and derived properties 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 constrains, extends, or modifies 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 or UML.

The stereotypes in the profile carry the abstract syntactic constraints from BPMN, and are not repeated here. For example, BPMN constraints on the number of sequence flows going out of event-based gateways, between values of the processType and isExecutable properties on processes, and between events and event definitions, apply to UML models when the corresponding stereotypes are applied. Applying BPMN constraints to a profiled model is facilitated by derived stereotype properties corresponding to BPMN. The derived properties also enable queries to the model based on BPMN property names, and provides a basis for mapping between BPMN and UML metaproperties.

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 predefined default values, if any. Other predefined values and implementation semantics are available in the BPMN specification, they are not restated here.

Abstract BPMN metaclasses are only supported in the profile when it simplifies the model, or to support the definition of constraints and derived properties.

2 Normative References

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 2.3 Specification (http://www.omg.org/spec/UML/2.3/)
- QVT 1.1 Specification (http://www.omg.org/spec/QVT/1.1/)

3 Informative References

There are no informative documents referenced in the specification.

4 Terms and Definitions

There are no new terms defined in this specification.

5 Symbols

There are no symbols defined in this specification.

6 Conventions

6.1 Stereotypes

Rectangles in the figures labeled «metaclass» are from the UML metamodel, while those labeled with «stereotype» and «enumeration» are from the profile. Primitive types, such as Boolean and String, are used from UML without stereotypes. 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. Metamodel figures show BPMN properties that are not derived. Derived properties are specified in separate subsections. The term "BPMN" used as a noun refers to the BPMN specification.

UML Stereotypes are Classes, with instances linked (applied) to instances of their base metaclass by Extensions, which are specialized associations. UML Extensions are navigable in both directions between instances of stereotypes and instances of metaclasses they are applied to, which means queries are efficient in returning an instance of a metaclass given an instance of a stereotype, and vice-versa. The names of these association ends in this profile are the concatenation of "base_" with the name of the metaclass. The names of these association ends in this profile are the concatenation of "extension_" with the name of the applied stereotype, with additional extensions for names of generalizations of the stereotype. UML Extensions are composite, with metaclass instances owning stereotype instances applied to them, which means deleting an instance of a metaclass will delete the instances of stereotypes applied to them, and instances of stereotypes can be applied to no more than one metaclass instance. Stereotypes in this profile extend the semantics of their extensions by deleting instances of metaclasses when instances of stereotypes applied to them are deleted (stereotype instances can be unlinked from instances of metaclasses before deleting stereotype instances to avoid deleting metaclass instances). This means when instances of metaclasses with stereotypes from the profile applied are deleted, the stereotype instances applied to the metaclass instance are deleted (due to extension composition), then the owned stereotype instances identified by links of any composite associations are deleted (due to composite association between stereotypes, if any), which in turn deletes the instances of a metaclass it is applied to (due to semantics of stereotypes in this profile).

The profile assumes specialized stereotypes can have bases that are specializations of bases of more general stereotypes, and defines constraints to ensure that specialized stereotypes are applied only to instances of their specialized bases. The profile uses abstract stereotypes with multiple bases when the specializations have bases that do not fall under one generalization. The profile defines constraints on the bases of specialized stereotypes to limit them to one base, which might be specializations of the base of more general stereotypes. The profile assumes the same stereotype can be applied multiple times to the same instance of its base, which means the base instance has multiple links of the extension association. The semantics of stereotypes is inherited to their specializations.

6.2 Mappings

Mappings from BPMN models to UML models with the profile applied are written in QVT Relational. The transformation accepts a BPMN model conforming to the BPMN metamodel, an initially empty UML model, and this profile as input, and populates the UML model with instances of the UML metamodel that have stereotypes applied to them. The BPMN metamodel and this profile are not modified by the transformation. The entire set of mappings in the profile is wrapped with the transformation statement below, which declares variables for the BPMN and UML models, and the profile.

```
transformation BPMN2UMLProfile( bpmn :  BPMN, uml :  UML, bpmnprofile : UML )
{
}
```

7 Core Structure

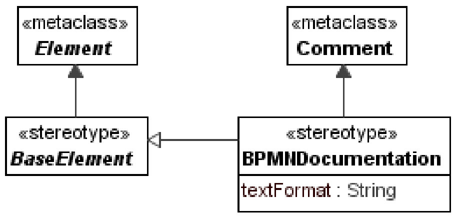


Figure 2: Base Elements

BPMN BaseElement generalizes almost other elements in BPMN, while UML Element generalizes all other elements in UML. They both can own user-specified text, and BPMNDocumentation includes specification of mime format. BPMN BaseElement is also used for extensibility, see below.

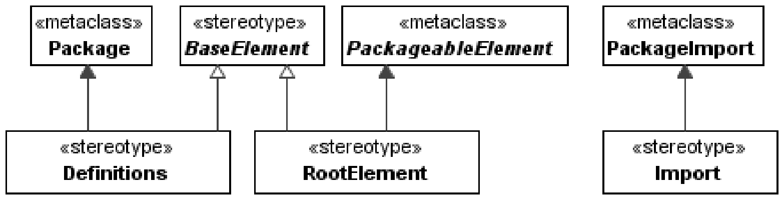


Figure 3: Definitions

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. 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 elements to be brought into definitions. UML PackageImport has the same purpose, but identifies external elements with packages,[1] providing derived properties for strings. Associations between definitions and imports in BPMN are captured as derived properties.

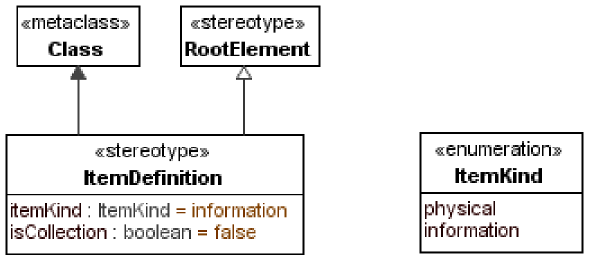


Figure 4: Item Definitions

BPMN ItemDefinition refers out of BPMN to specify structure, while UML uses Classes to 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.[2] BPMN isCollection tells whether the underlying data type is a collection, but the UML

[1] UML ElementImport imports individual elements in packages, but BPMN describes the external "elements" as documents, so the profile assumes importing packages is more appropriate.

[2] BPMN ItemDefinition supports references to technology-specific definitions as an alternative for specifying structure. UML is often used under a model-driven architecture for generating platform-specific formats. BPMN's import alternative for specifying structure is not supported in the profile.

metamodel does not have collection types, so the profile assumes isCollection = true means the item definition represents collections of instances of the class to which the ItemDefinition stereotype is applied.

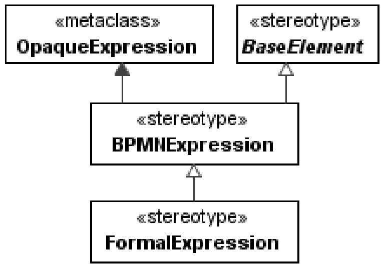


Figure 5: BPMN Expressions

UML OpaqueExpression and BPMN FormalExpression 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 BPMNExpression and FormalExpression.

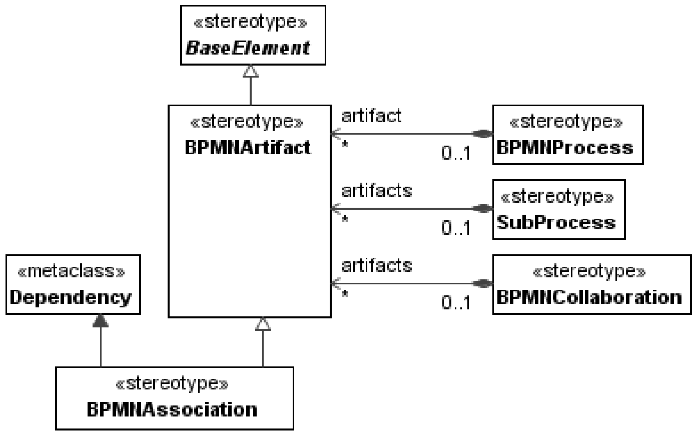


Figure 6: Artifacts

BPMN Artifacts are miscellaneous elements that can be included in processes, subprocesses and collaborations. Additional syntax and semantics is given in its specializations. BPMN Associations 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. The BPMNAssociation stereotype only supports unidirectional associations, which are the kind used in BPMN notation.

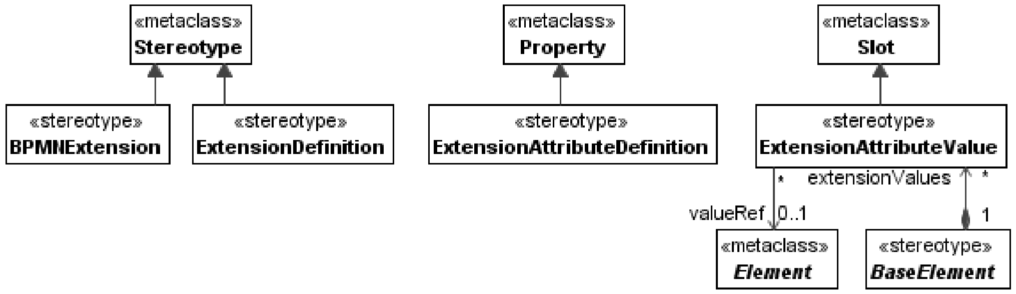


Figure 7: BPMN Extensions

BPMN Extension and ExtensionDefinition enable modelers to define properties with 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 InstanceSpecifications, which contain Slots specifying values for the properties defined in the classes. BaseElement stereotypes can own ExtensionAttributeValues that are applied to slots. These slots are owned by an instance specification describing instances of the stereotypes applied to the user model element with the BaseElement applied. The slots can use UML ValueSpecification to specify values for properties, or the ExtensionAttributeValue stereotype applied to them can specify values by referring to elements in the user model or other models. The ExtensionAttributeValue stereotype requires the actual instances of the applied stereotypes to conform to these instance specifications, giving semantics equivalent to BPMN's.

Derived Properties

- BaseElement::/documentation : BPMNDocumentation [*] = self.base_Element.ownedComment.extension_BPMNDocumentation
- BPMNDocumentation::/text : String = self.base_Element.ownedComment.body

- Definitions::/rootElements : RootElement [*] = self.base_Package.packagedElement.extension_RootElement
- RootElement::/definition : Definitions [0..1] = self.base_PackageableElement.owningPackage.extension_Definitions
- Import::/importType : String = location of language specification for imported element, sometimes a URI, derived from the contents or filename of the imported element.
- Import::/location :: String = Import::/location : String = self.base_PackageImport.importedPackage.URI
- Import::/namespace : String = self.base_PackageImport.importedPackage.name

- ItemDefinition::/structureRef = self.base_Class

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

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

- Definitions::/extensions : Extension [*] = self.base_Package./ownedStereotype .extension_Extension
- Extension::/definition : ExtensionDefinition = self.base_Stereotype.extension_ExtensionDefinition
- BaseElement:/extensionDefinitions : ExtensionDefinition [*] = stereotypes with ExtensionDefinition applied that have instances applied to the element
- ExtensionDefinition::/extensionAttributeDefinitions : ExtensionAttributeDefinition [*] = self.base_Steterotype.ownedAttribute.extension_ExtensionAttributeDefinition
- ExtensionAttributeDefinition::/type : String = self.base_Property.type./qualifiedName
- ExtensionAttributeDefinition::/isReference : Boolean = (self.base_Property./isComposite = false)
- ExtensionAttributeValue::/extensionAttributeDefinition :: ExtensionAttributeDefinition = self.appliedTo_Slot.owningInstance.classifier.extension_ExtensionAttributeDefinition

Constraints

- BPMNDocumentation may only be applied to Comments.
- Definitions may only be applied to Packages.
- RootElements may only be applied to PackageableElements.

- ItemDefinitions may only be applied to Classes.
- BPMNExpressions may only be applied to OpaqueExpression.
- BPMNAssociations may only be applied to Dependencies.
- BPMNExtension may only be applied to elements with ExtensionDefinition applied, and vice versa.

- SubProcess.artifact = instances of Artifact stereotype applied to dependencies that have client and suppliers directly contained by the subprocess (not nested subprocesses).
- SubProcess.artifact = instances of Artifact stereotype applied to dependencies that have client and suppliers directly contained by the subprocess (not nested subprocesses).
- Collaboration.artifact = instances of Artifact stereotype applied to dependencies that have client and suppliers directly contained collaboration.

- Stereotypes with Extension applied are based on Element.
- The extensionValues of a BaseElement 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 BaseElement.
- Slots with ExtensionAttributeValue applied are owned by instance specifications that are owned by the package containing the element having the applied ExtensionAttributeValue instance as the value of its extensionValues property.

Mappings

```
top relation DefinitionsToPackage
// The BPMN Definition is bound to the variable ‘d’. Its name is bound to the variable ‘dn’.
// ‘checkonly’ semantics ensures there are no side effects in the bpmn model.
// the ‘enforce’ semantics ensures that a corresponding Package is created in the uml model,
// and the Definition stereotype is applied to it, and the ‘name’ is copied.
// the ‘where’ clause calls the subsequent relations to transform the contained elements.
{ dn :  String;
   checkonly domain bpmn d : Definitions{
                        name  =  dn,
                        rootElement  =  re : RootElement{ } };
   enforce domain uml umlp : uml::Package{
                        name  =  dn,
                        packageableElement  =  pe : PackageableElement{ } };
   //apply Definitions stereotype to umlp
   enforce domain uml udef : bpmnprofile::Definition{
                        base_Package  =  umlp };
   where { RootElementToPackageableElement(re, pe); }
}

relation RootElementToPackageablElement
//map BPMN RootElement to UML PackageableElement
{ checkonly domain bpmn re : RootElement{
                                    name = ren,
                                    definition = d : Definition };
   enforce domain uml pe : uml::PackageableElement{
                                    name = ren,
                                    owningPackage = p : Package };
   when  { DefinitionToPackage(d, p); }
   where { ProcessToActivity(re, pe);
                ActivityToAction(re, pe); }
}
```

8 BPMN Processes

8.1 Processes and Global Tasks

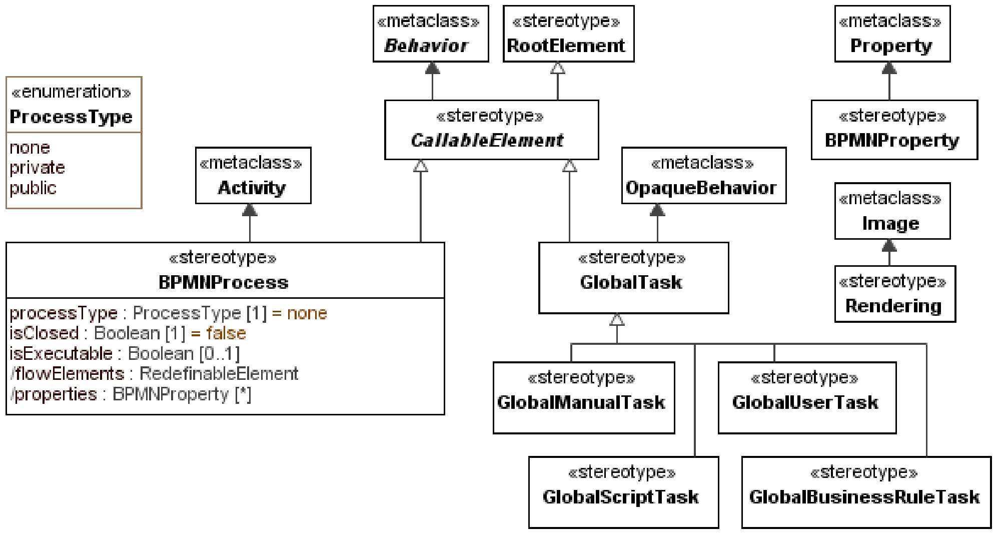

«metaclass» Behavior
«stereotype» RootElement
«metaclass» Property
«enumeration» ProcessType
none
private
public
«stereotype» CallableElement
«stereotype» BPMNProperty
«metaclass» Activity
«metaclass» OpaqueBehavior
«metaclass» Image
«stereotype» BPMNProcess
processType : ProcessType [1] = none
isClosed : Boolean [1] = false
isExecutable : Boolean [0..1]
/flowElements : RedefinableElement
/properties : BPMNProperty [*]
«stereotype» GlobalTask
«stereotype» Rendering
«stereotype» GlobalManualTask
«stereotype» GlobalUserTask
«stereotype» GlobalScriptTask
«stereotype» GlobalBusinessRuleTask

Figure 8: Processes and Global Tasks

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

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. UML Properties hold values of specified types for classes, including activities, providing a semantics equivalent to BPMN's. The BPMN isClosed property indicates whether valid process instances may carry out interactions, such as receiving messages and events, that are not specified in their models. If isClosed is true, then process instances are limited to interactions specified in their models, otherwise they are not. The Process 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 association 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 Process stereotype. One UML Activity generalizing another means occurrences (instances) of the specialized activity are occurrences of the general activity.

The BPMN isExecutable property indicates whether a private process is intended to have BPMN execution semantics at some point in development of the model. The profile provides a UML-based execution semantics that is equivalent to BPMN's, see discussion of semantics in Section 1.

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. See the /flowElements derived property.

See Sections 8.4, 8.5, and 9 for more about the Process and Subprocess stereotypes. GlobalManualTask, GlobalScriptTask, GlobalUserTask, and GlobalBusinessRuleTask have application semantics, see the BPMN specification.

Derived Properties

- Process::/flowElements = the stereotype instances applied to values of Activity::node, Activity::edge, and Activity::group.
- Process::/supports : Process = self.base_Activity./general.extension_Process (defined on Classifier).
- Process::/properties : BPMNProperty [*] = self.base_Activity.ownedAttribute.extension_BPMNProperty (defined on Class) values with the BPMNProperty applied.
- Property::/itemSubjectRef : ItemDefinition [0..1] = self.base_Property.type.extension_ItemDefinition
- GlobalScriptTask::/scriptFormat : String [*] {ordered} = self.base_OpaqueBehavior.language
- GlobalScriptTask::/script : String [*] {ordered, non-unique} = self.base_OpaqueBehavior.body
- GlobalUserTask::/implementation : String [*] {ordered, non-unique} = self.base_OpaqueBehavior.body (defaults to “##unspecified”, see BPMN specification)
- GlobalUserTask::/renderings : Image [*] = self.icon.extension_Image (defined on Stereotype)
- GlobalBusinessRuleTask::/implementation : String [*] {ordered, non-unique} = self.base_OpaqueBehavior.body (defaults to “##unspecified” , see BPMN specification)

Constraints

[1] Processes may only be applied to Activities.
[2] GlobalTasks may only be applied to OpaqueBehaviors.

[3] The classifier behaviors of UML Activities with the Process stereotype applied are the activities themselves.

Mappings

```
relation ProcessToActivity
{ pn :  String;
  checkonly domain bpmn p : Process{
                                    name = pn,
                                    processType = pt : ProcessType{ },
                                    isClosed = ic : Boolean,
                                    isExecutable = ie : Boolean };
  enforce domain uml a : uml::Activity{ name = pn };
  //apply Process stereotype to a and map properties
  enforce domain uml umlp : bpmnprofile::Process {
          base_Activity  =  a,
          processType  =  pt,
          isClosed = is,
          isExecutable = ie };
  where { PropertyToProperty(p, a); }
}

relation PropertyToProperty
//map properties of Process to BPMNProperty stereotype
{ pn :  String;
  checkonly domain bpmn p : Process {
                                    property  =  prop : Property{ name  =  pn }};
  enforce domain uml a : uml::Activity { };
  enforce domain uml umlprop : uml::Property{ name = pn };
  //apply BPMNProperty stereotype to prop
  enforce domain uml prop : bpmnprofile::BPMNProperty {
                                    base_Property  =  umlprop };
  when { ProcessToActivity( p, a ); }
}
```

```
relation GlobalScriptTaskToOpaqueBehavior
//map GlobalScriptTask to OpaqueBehavior with GlobalScriptTask stereotype applied
{ checkonly domain bpmn gst : GlobalScriptTask {
                                        name  =  n : String,
                                        scriptFormat  =  sf : String,
                                        script = scr : String };
   enforce domain uml umlOB : uml::OpaqueBehavior {
                                        name = n,
                                        language = sf,
                                        body = scr };
   //apply GlobalScriptTask  stereotype
   enforce domain uml ugst : bpmnprofile::GlobalScriptTask {
                           base_OpaqueBehavior = umlOB };
}

relation GlobalUserTaskToOpaqueBehavior
{ checkonly domain bpmn gut : GlobalUserTask {
                                        name  =  n : String,
                                        implementation  =  i : String,
                                        renderings = r : Image };
   enforce domain uml umlOB : uml::OpaqueBehavior {
                                        name = n,
                                        body = i };
   //apply GlobalUserTask stereotype
   enforce domain uml ugut : bpmnprofile::GlobalUserTask {
                           base_OpaqueBehavior = umlOB,
                           icon = r };
}

relation GlobalBusinessRuleTaskToOpaqueBehavior
{ checkonly domain bpmn gbrt : GlobalBusinessRuleTask {
                                        name = n : String,
                                        implementation = i : String };
   enforce domain uml umlOB : uml::OpaqueBehavior  {
                                        name = n,
                                        body = i };
   //apply GlobalBusniessRuleTask stereotype
   enforce domain uml ugbrt : bpmnprofile::GlobalBusinessRuleTask {
                           base_OpaqueBehavior = umlOB };
}
```

8.2 Activities

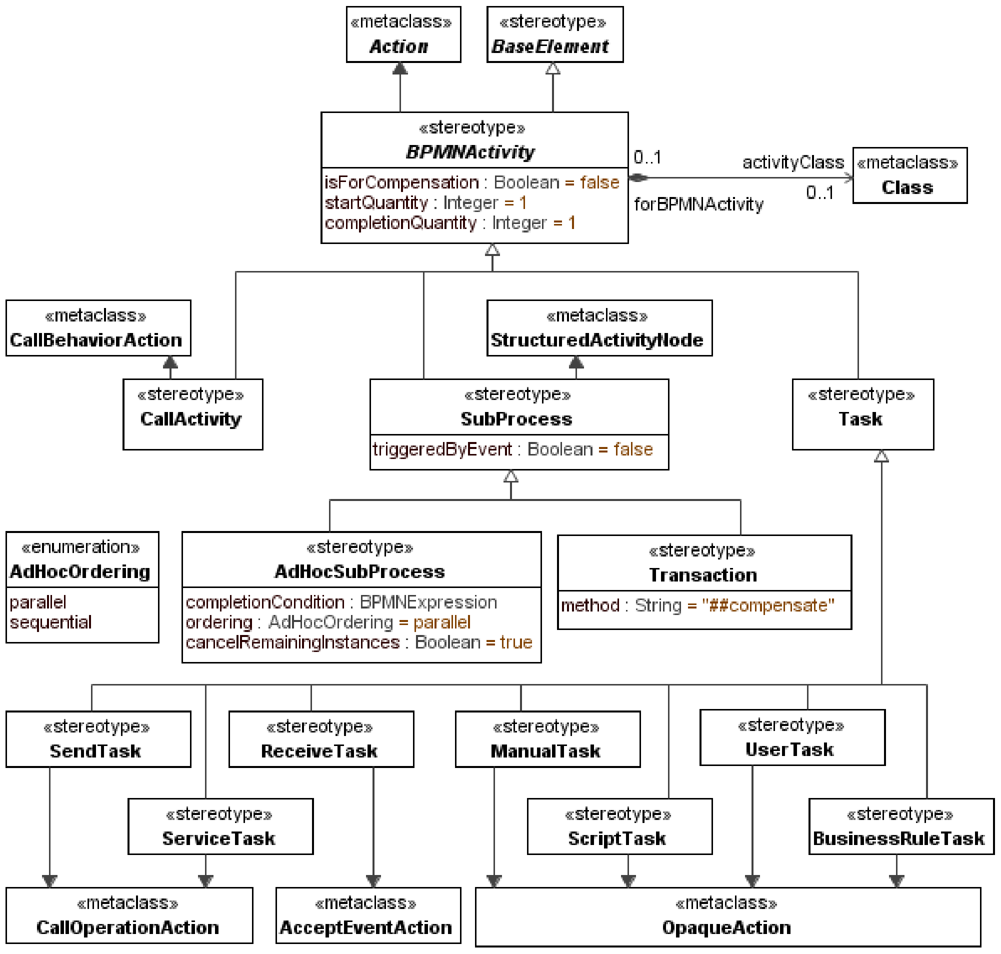

«metaclass» Action
«stereotype» BaseElement
«stereotype» BPMNActivity
isForCompensation : Boolean = false
startQuantity : Integer = 1
completionQuantity : Integer = 1
0..1
activityClass
0..1
forBPMNActivity
«metaclass» Class
«metaclass» CallBehaviorAction
«metaclass» StructuredActivityNode
«stereotype» CallActivity
«stereotype» SubProcess
triggeredByEvent : Boolean = false
«stereotype» Task
«enumeration» AdHocOrdering
parallel
sequential
«stereotype» AdHocSubProcess
completionCondition : BPMNExpression
ordering : AdHocOrdering = parallel
cancelRemainingInstances : Boolean = true
«stereotype» Transaction
method : String = "##compensate"
«stereotype» SendTask
«stereotype» ReceiveTask
«stereotype» ManualTask
«stereotype» UserTask
«stereotype» ServiceTask
«stereotype» ScriptTask
«stereotype» BusinessRuleTask
«metaclass» CallOperationAction
«metaclass» AcceptEventAction
«metaclass» OpaqueAction

Figure 9: Activities

There are several kinds of BPMN Activity, all supported in the profile by stereotypes on some kind of UML Action. See the constraints below for the kind of action each stereotype may be applied to.

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, which is expressed in the profile by BPMNActivity::startQuantity = *. For values of startQuantity other than *, the BPMNActivity stereotype modifies UML Action semantics to only require incoming control flows collectively to offer the number of tokens specified by startQuantity to start, giving an semantics equivalent to BPMN's.

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 completionQuantity other than 1, the BPMNActivity stereotype modifies UML Action semantics to offer the number of tokens specified by startQuantity 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 Section 8.1 about process instances). BPMN Activities can define properties that hold items of specified kinds activity instances. The BPMNActivity stereotype extends UML Actions with a class that has a distinct instance every time actions are carried out in an 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.

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 Figure 12 in Section 8.4.[3] In these cases, the SubProcess stereotype with a true value for the triggeredByEvent property modifies UML Activity and StructuredActivityNode semantics to be equivalent to BPMN's by offering tokens to non-interrupting structured activity nodes after they start,[4] and not replenishing tokens in accept event actions that start interrupting structured activity nodes. The Subprocess stereotype also modifies 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 AdHocSuprocess 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 completionCondition becomes true are terminated.[5] 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, see Section 8.4.[6]

See the SequenceFlow stereotype in Section 8.3 for other semantics of the BPMNActivity stereotype, and Section 8.4 about is the isForCompensation property. See Section 9 about BPMN SendTasks, ReceiveTasks, and ServiceTasks. ManualTask, ScriptTask, UserTask, and BusinessRuleTask have application semantics, see the BPMN specification.

Derived Properties

- BPMNActivity::/container : RedefinableElement the collected values of Action::activity and Action::inGroup. Constrained to elements stereotyped by BPMNActivity or SubProcess.
- BPMNActivity::properties : BPMNProperty [*] = self.base_BPMNActivity.activityClass.ownedAttribute extension_BPMNProperty
- BPMNActivity::/default : SequenceFlow [0..1] = instance of Sequence Flow stereotype applied to an outgoing control flow with guard = "else".
- CallActivity::/calledElement : CallableElement = self.base_CallBehaviorAction.behavior extension_CallableElement
- ScriptTask::/scriptFormat : String [*] {ordered} = self.base_OpaqueAction.language
- ScriptTask::/script : String [*] {ordered, non-unique} = self.base_OpaqueAction.body
- UserTask::/implementation : String [*] {ordered, non-unique} = self.base_OpaqueAction.body (defaults to "##unspecified", see BPMN specification)

[3] BPMN says the rest of the process is cancelled, which only applies to transactions. The profile assumes the rest of the process is terminated, rather than cancelled.

[4] Chapter 13 (Execution Semantics) of the BPMN 2 specification is not clear that non-interrupting event subprocesses can be started multiple times in parallel under the same process instance, but Section 10.2.5 (Sub-Processes) is. The UML isLocallyReentrant property of action is used to ensure parallelism for non-interrupting event subprocesses see Constraints.

[5] BPMN says remaining instances are cancelled, which only applies to transactions. The profile assumes the remaining instances are terminated, rather than cancelled.

[6] BPMN does not specify that changes made in a transaction do not conflict with changes made concurrently outside the transaction. Since transactional systems typically ensure such conflicts do not occur, implementations of the profile may require the UML mustIsolate property of structured activity nodes to be true when the Transaction stereotype is applied.

- UserTask::/renderings : Image [*] = self.icon extension_Image (defined on Stereotype)
- BusinessRuleTask::/implementation : String [*] {ordered, non-unique} = self.base_OpaqueAction.body (defaults to "##unspecified" , see BPMN specification)

Constraints

[1] CallActivites may only be applied to CallBehaviorActions.
[2] SubProcesses may only be applied to StructuredActivityNodes.
[3] Tasks that are not SendTasks, ReceiveTasks, or ServiceTasks may only be applied to OpaqueActions.

[4] BPMNActivity::startQuantity must be greater than zero.
[5] BPMNActivity::completionQuantity must be greater than zero.
[6] Task::body may not have any values.
[7] Task::language may not have any values.
[8] StructuredActivityNodes with SubProcess applied that have triggeredByEvent = true, and containing initial nodes with StartEvent applied that have isInterrupting = false, have isLocallyReentrant = true.

Mappings

```
relation ActivityToAction
{ checkonly domain bpmn ba : Activity { };
        enforce domain uml ua : uml::Action { }
        where
  {//map types of BPMN Actvity
        CallActivityToCallBehaviorAction( ba, ua );
        TaskToOpaqueAction( ba, ua );
        ManualTaskToOpaqueAction( ba, ua );
        ScriptTaskToOpaqueAction( ba, ua );
        UserTaskToOpaqueAction( ba, ua );
        BusinessRuleTaskToOpaqueAction( ba, ua );
  }
}

relation CallActivityToCallBehaviorAction
{ checkonly domain bpmn ba : CallActivity {
                callableElementRef = ca : CallableElement {
                name = cen }};
  enforce domain uml umlcba : uml::CallBehaviorAction {
                behavior = b : Behavior{ name = cen }};
  //apply CallActivity stereotype to ua
  enforce domain uml uca : bpmnprofile::CallActivity {
                base_CallBehaviorAction = umlcba };
}

relation ScriptTaskToOpaqueAction
{ checkonly domain bpmn st :  ScriptTask{
                                name = n : String;
                                scriptFormat = sf : String,
                                script = scr : String };
  enforce domain uml umla : uml::OpaqueAction {
                                name = n,
                                language = sf,
                                body = scr } ;
  //apply ScriptTask stereotype to umla
  enforce domain uml ua : bpmnprofile::ScriptTask {
                        base_OpaqueAction  =  umla };
}
```

```
relation UserTaskToOpaqueAction
{ checkonly domain bpmn ut :  UserTask{
                                        name = n : String,
                                        implementation = i : String,
                                        renderings = r : Image };
  enforce domain uml umla : uml::OpaqueAction {
                                        name = n,
                                        //map 'implementation' to 'body'
                                        body = i } ;
  //apply UserTask stereotype to umla
  enforce domain uml ua : bpmnprofile::UserTask {
                              base_OpaqueAction  =  umla,
                              //map 'renderings' to 'icon'
                              icon = r };
}

relation BusinessRuleTaskToOpaqueAction
{ checkonly domain bpmn brt :  BusinessRuleTask {
                                        name = n : String,
                                        implementation = i : String };
  enforce domain uml umla : uml::OpaqueAction {
                                        name = n,
                                        //map 'implementation' to 'body'
                                        body = i } ;
  //apply BusinessRuleTask stereotype to umla
  enforce domain uml ua : bpmnprofile:: BusinessRuleTask{
                              base_OpaqueAction  =  umla };
}
```

8.3 Sequence Flows

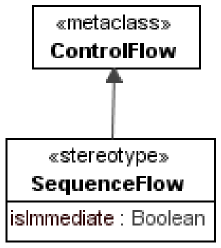


Figure 10: Sequence Flow

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). 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 Section 8.5. The BPMNActivity, InclusiveGateway, and ComplexGateway stereotypes modify the UML semantics of actions and fork nodes to offer a token to an “else” guard if none of the other outgoing control flows out of the same element accept any, giving a semantics equivalent to BPMN’s.

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 Process 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.

Derived Properties

- SequenceFlow::/conditionExpression : BPMNExpression = self.base_ControFlow.guard extension_BPMNExpression (defined on ActivityEdge)
- SequenceFlow::/sourceRef = self.base_ControlFlow.source (defined on ActivityEdge)
- SequenceFlow::/targetRef = self.base_ControlFlow.target (defined on ActivityEdge)

Mappings

```
relation SequenceFlowToControlFlow
{ checkonly domain bpmn sf : SequenceFlow {
                              conditionExpression = ce:BPMNExpression,
                              sourceRef = sr:FlowNode,
                              targetRef = tr:FlowNode };
  enforce domain uml umlCF : uml::ControlFlow {
                              guard = uce,
                              source = usr,
                              target = utr };
  //apply SequenceFlow stereotype to umlCF
  enforce domain uml usf : bpmnprofile::SequenceFlow {
                      base_ControlFlow = umlCF };
  when
  {     BPMNExpressionToOpaqueExpression( ce, uce );
        BPMNFlowNodeToActivityNode( sr, usr );
        BPMNFlowNodeToActivityNode( tr, utr );
  }
}
```

8.4 Events

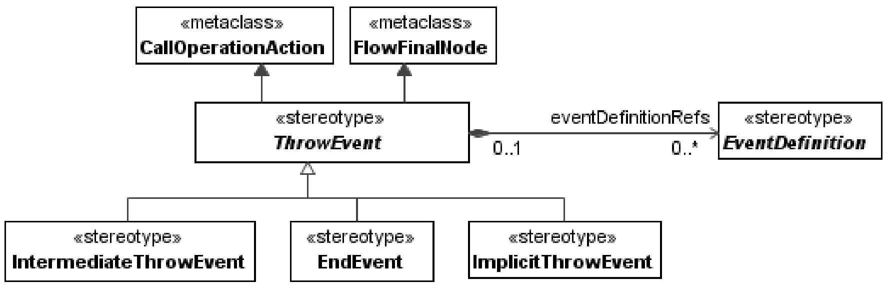


Figure 11: Throw Events

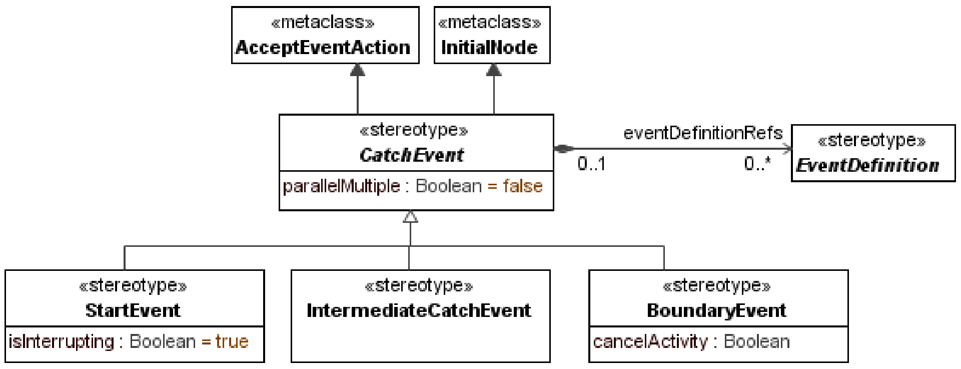


Figure 12: Catch Events

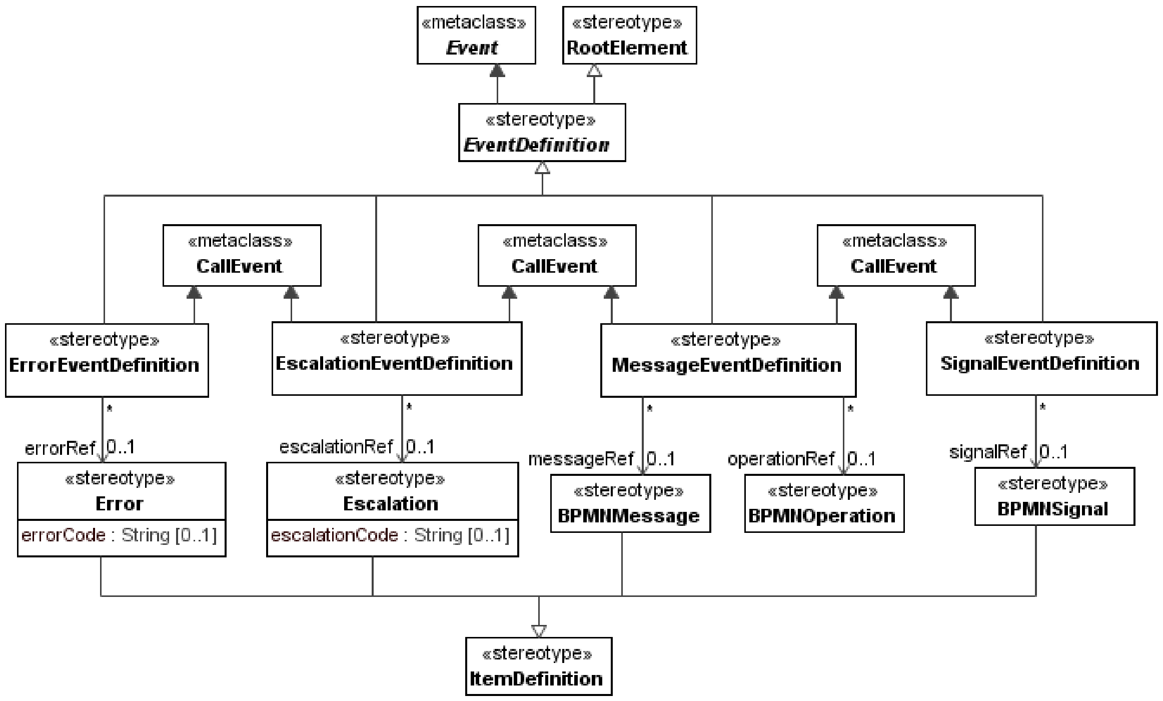

«metaclass» Event
«stereotype» RootElement
«stereotype» EventDefinition
«metaclass» CallEvent
«metaclass» CallEvent
«metaclass» CallEvent
«stereotype» ErrorEventDefinition
«stereotype» EscalationEventDefinition
«stereotype» MessageEventDefinition
«stereotype» SignalEventDefinition
errorRef 0..1
escalationRef 0..1
messageRef 0..1
operationRef 0..1
signalRef 0..1
«stereotype» Error
errorCode : String [0..1]
«stereotype» Escalation
escalationCode : String [0..1]
«stereotype» BPMNMessage
«stereotype» BPMNOperation
«stereotype» BPMNSignal
«stereotype» ItemDefinition

Figure 13: Event Definitions with Item Definitions

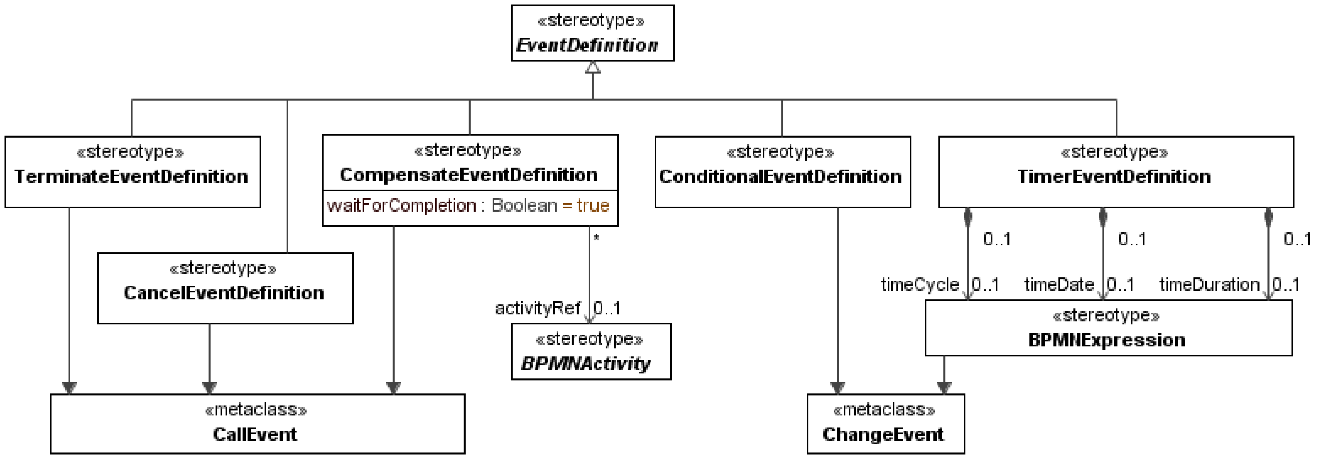

«stereotype» EventDefinition
«stereotype» TerminateEventDefinition
«stereotype» CompensateEventDefinition
waitForCompletion : Boolean = true
«stereotype» ConditionalEventDefinition
«stereotype» TimerEventDefinition
«stereotype» CancelEventDefinition
activityRef 0..1
«stereotype» BPMNActivity
timeCycle 0..1
timeDate 0..1
timeDuration 0..1
«stereotype» BPMNExpression
«metaclass» CallEvent
«metaclass» ChangeEvent

Figure 14: Other Event Definitions

The bases of the CatchEvent, ThrowEvent, and EventDefinition stereotypes reflect terminology differences between BPMN and UML. 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 packagable, 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 not a catch event.

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 Error, Escalation, Message, and Signal, that in turn refer to ItemDefinitions for the type of thing sent or received along with the event. The profile reflects this in the event definition stereotypes in

Figure 13, which refer to other stereotypes specialized from ItemDefinition, to simplify the model while still being equivalent to BPMN.[7] Throw event stereotypes in the profile provide equivalent semantics to other aspects of BPMN by extending UML CallOperationAction with equivalent effects, and constraining its target appropriately, passing items as UML Parameter values, if any (see below about operations for catching events):

- BPMN end events with no event definitions (owned or referenced) have no effect. UML FlowFinalNodes have equivalent semantics, and EndEvent is applied to provide BPMN's terminology.
- 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 Section 8.2 about BPMN Process and Activity instances), but terminate events are not thrown anywhere in the sense of having corresponding catch events.[8] UML CallOperations with a throw event stereotype applied, and TerminateEventDefinition applied to its trigger, call termination operations on the UML Activity instance or innermost UML StructuredActivityNode instance they are occurring in, which terminates all the actions in the activity or structured node recursively (see Section 8.2 about UML Activity and Action instances).
- 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,[9] and finally are thrown to cancel boundary events on that transaction, if any.[10] UML CallOperationActions with a throw event stereotype applied, and CancelEventDefinition applied to its trigger, call termination operations on the innermost instance of UML StructuredActivityNode with Transaction applied that the call operation action is occurring in, and undo any changes made during the structured node instance, then call operations on the process or activity instance the structured node instance was occurring in to notify them of the cancellation.
- BPMN error and escalation events are thrown to matching start error or escalation events in event subprocesses at the same level they are thrown, if any,[11] otherwise they are thrown to the innermost matching error or escalation event they are thrown from, including through call activities, which might be a boundary event or a start event of an event subprocess (see below about matching catch events).[12] 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.[13]

[7] BPMN does not explicitly require the item definitions for items thrown with events to have the same properties and values as the BPMN Errors and Escalations that refer to the item definitions, perhaps because BPMN is inconsistent about whether BPMN Errors, Escalations, and Signals specify the payload or the item definitions they reference. The profile assumes thrown items are the payload and have the properties of errors and escalations, so error and escalation codes are included in item properties if the modeler specifies them.

[8] Interrupting boundary events are catch events, but they catch non-termination events then terminate their associated activity instances.

[9] The BPMN method property on transactions is a string for specifying this with the possible values "##compensate", "##image", "##store", or a technology-specific URI (required for execution), but this property is not given semantics in BPMN. The BPMN protocol property identifies an underlying transaction protocol, and gives some semantics when other participants are involved, but the property is not available in the BPMN XSD. The profile provides abstract syntax for these properties, with implementation semantics, see the BPMN specification.

[10] BPMN is not clear about whether cancel events can be thrown from subprocesses or calls in transaction subprocesses. The profile assumes they can.

[11] BPMN omits event subprocesses in error and escalation propagation, but from the description of event subprocesses, it seems like they were intended to be included. The profile assumes event subprocess catch propagated error and escalation events.

[12] BPMN is inconsistent on the effect of not finding a matching error or escalation boundary event, saying it is unresolved, but also saying all activities are compensated in the subprocess in which the error is thrown. The profile assumes it is unresolved, and there is no effect due to uncaught events, including error and escalation events.

[13] BPMN does not specify that throw error and escalation events with event definitions that refer to Error or Escalation elements with error and escalation codes throw the code even when Error or Escalation do not

As error events are thrown, they have 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.[14] UML CallOperationActions with a throw event stereotype applied, and ErrorEventDefinition or EscalationEventDefinition applied to its trigger, call operations on the UML Activity or UML StructuredActivityNode instance that the call operation action is occurring in to notify them of the error or escalation. If the error or escalation event definition 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 operations. The classes have errorCode or escalationCode properties, typed by Strings, 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).[15] 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 process or activity instances the next level up with the same arguments, recursively, until an activity or structured node instance handles the notification, or a top-level process instance is reached. For error event definitions, activity or structured node instances that do not handle the notification operations are terminated before the next level is notified.

- BPMN compensation events attempt to reverse changes made during completed instances of activities identified by the event ahead of time,[16] if any. If the identified activity is a subprocess, then this is done with a 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).[17] If the throw compensation event does not identify an activity to be compensated, the compensation events are thrown to start compensation events in event subprocesses at the same level they are thrown from, if any.[18,19] 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 process 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

refer to item definitions. The profile assumes the codes are thrown, and generalizes Error and Escalation by ItemDefinition to reflect this semantics.

[14] BPMN does not explicitly require termination beyond the process instance or innermost subprocess instance in which the event is thrown, but the profile assumes termination occurs up to where the event is caught.

[15] The BPMN errorCode and escalationCode properties are required in the BPMN metamodel, but optional in the BPMN XSD. The profile assumes they are optional.

[16] BPMN Process and Activity instances are not active after they are complete, and might be considered not to exist at all, but their internal state at the time they complete is kept for reversing changes made by the instances when the were active (compensation). The profile assumes activity and processes exist in some form after they complete, at least to respond to compensation operation calls.

[17] BPMN explicitly disallows compensation activities associated with boundary compensation events from having access to the state of the process instance in which the compensated activity instance occurred, so it is unclear how compensation activities can reverse the effects of the compensated activity instance.

[18] BPMN gives compensation event subprocesses access to the the state of the process instance at the time of completion, but does not provide a way to pass this to the compensation activities identified by boundary compensation events, which will have no inputs due to being disallowed access to process instance state, see footnote 17.

[19] BPMN execution semantics omits handling compensation before a process is finished, and says that compensation start events for subprocesses are only triggered after the containing process is finished, but BPMN supports throw compensation intermediate events, and does not specify that throw compensation end events terminate their processes. The profile assumes compensation events thrown by processes before they are finished can compensate activities completed so far.

to reverse changes due to completed activity instances at the same level as the subprocess in the reverse order in which they occurred. 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,[20] 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 Process or Activity instance the call operation action is occurring in, unless it is occurring in a 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 Process and Subprocess stereotypes modify UML Activity and StructuredActivityNode semantics respectively to not offer tokens to actions with BPMNActivity applied that have a true value for the isForCompensation property.

- BPMN message events are thrown to particular external participants in collaborations, which involves sending items to those participants as messages (see Section 9 for more about throwing messages).
- Signals are thrown to all process instances, including those in other participants (in some implementations signals are thrown just to process instances subscribed to or accepting matching signals, the effect is the same, see below 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, are constrained to have no incoming edges to their target pins, and have their semantics modified to start without a value on their target pins, and call the signal operation on all UML Activity instances that support it, 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.

The BPMN throw end events above that do not explicitly terminate the process or subprocess instance they are thrown from have no effect on that instance (at least not before the event is caught). Instances without end events that terminate complete when all activities in the instance are finished, and no catch events are waiting. UML Activities have the same semantics. BPMN ThrowEvents do not wait for events to be received wherever they are sent before completing, except for compensation events with event definitions indicating the throw event should wait for compensation to complete before finishing. The throw event stereotypes applied to UML CallOperationAction reflect this aspect of BPMN semantics by requiring call operation actions to be asynchronous, except when CompensateEventDefinition is applied to their triggers with a true value for the waitForCompletion property, whereupon the call operation action is synchronous. BPMN conditional and time events are only caught, not thrown, see below about catch events. See Section 8.7 about BPMN ImplicitThrowEvents.

[20] Footnote 16 applies to UML Activity and Action instances as well.

BPMN catch events accept thrown events with the same or equivalent event definitions,[21] which includes referenced item definitions (BPMN StartEvents without event definitions are catch events, but do not actually catch events, they indicate where to start new process instances). 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 (action instances do not support signals, see above about invoking the operations when throwing events),[22] The profile does not specify the names of these operations, and neither does the modeler, they are defined by simulation, enactment, and execution engines supporting the profile, or standards these engines choose to adopt. 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 modify UML AcceptEventAction semantics to only accept events for calls where the actual argument is an instance of the class referred to by the event definition stereotype applied to the trigger of the action, or one of its subclasses, giving semantics equivalent to BPMN's, and integrating it with UML's for support specialization of UML Class, which the ItemDefinition stereotype is based on. See Section 9 about catching messages.

BPMN StartEvents with event definitions at the top level of a process are able to instantiate processes when events of particular kinds occur.[23] The Process stereotype modifies UML Activity semantics to be equivalent to BPMN's by behaving as if tokens were offered to accept event actions directly under activities that are not instantiated, with StartEvent applied, and that have event definitions, and instantiating the activity when the event occurs, giving semantics equivalent to BPMN's.[24] 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, modify UML AcceptEventAction semantics to only accept events for calls where the actual argument either has no value for the errorCode or escalationCode property, or the same value as the errorCode and escalationCode property of the value of the errorRef or escalationRef property of the event defintion, giving semantics equivalent to BPMN's. BPMN StartEvents with compensation event definitions in event subprocesses that are in subprocesses are able to recover the state of the containing subprocess as it was when it was completed, 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, modifies UML Activity and StructuredActivityNode semantics to be equivalent to BPMN's by behaving as if tokens were offered to the structured activity node under the activity instance in which the containing structured activity instance occurred, giving semantics equivalent to BPMN's. BPMN IntermediateCatchEvents only catch events that occur after the process flow reaches them[25] and before it leaves them, while UML AcceptEventActions accepts events that occur anytime before the action ends, including before it starts. The profile modifies UML event pool semantics to discard events that no action can accept at the time they are put in the pool, giving semantics equivalent to BPMN's. BoundaryEvent

[21] BPMN is not clear on whether a throw and catch require the same event definition instance, or only equivalent ones (same kind and content). The profile assumes it could be either.

[22] The profile does not specify the names of these operations, and neither does the modeler, they are defined by simulation, enactment, and execution engines supporting the profile, or standards these engines choose to adopt.

[23] See footnote 34 in Section 9 about instantiating processes by receiving messages.

[24] For processes with multiple start events, BPMN is inconsistent about whether each start event initiates its own process instance, or each process instance requires all start events to arrive before it completes. The profile assumes each start event initiates its own process instance, and the Process stereotype modifies UML Activity semantics for starting an activity to put tokens in only one of the initial nodes or accept event actions with StartEvent applied.

[25] This was not entirely agreed among BPMN 2 Finalization Task Force members, but the BPMN specification is fairly clear, see Section 10.4.6 (Handling Events), subheading Handling Events within normal Sequence Flow, and the execution semantics in Section 13.4.2 (Intermediate Events).

stereotypes may only be applied to accept event actions in interruptible regions. For interrupting boundary events the outgoing edges are interrupting, otherwise they are non-interrupting (boundary events are interrupting if the cancelActivity property is true, but this refers to termination of the activity, rather than cancellation in the transactional sense, see Section 8.2). 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 associated BPMNActivity action 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 above.

BPMN catch events with ConditionalEventDefinition and TimerEventDefinition are for noticing changes in conditions and time. The corresponding event definition 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. The TimerEventDefinition stereotype extends UML ChangeEvent semantics to detect the arrival of a particular time with 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. This provides an equivalent semantics to BPMN's for timer event definitions.

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, catch event stereotypes modify the semantics of UML AcceptEventAction to require events for all of its triggers to arrive for the action to complete, providing semantics equivalent to BPMN's. The BPMN isInterrupting property on start events is for event subprocesses, see Section 8.2.

Derived Properties

- CatchEvent::/eventDefinitionRefs : EventDefinitions [*] = instances of EventDefinition applied to events of the triggers of an accept event action to which the stereotype is applied that are not owned by stereotypes applied to the action.
- BoundaryEvent::/attachedToRef : BPMNActivity = instance of BPMNActivity applied to action in same interruptible region as the event.
- BPMNActivity::/boundaryEventRefs : BoundaryEvent [*] = instances of BoundaryEvent applied to accept event actions in same interruptible region as the BPMNActivity action.
- ThrowEvent::/eventDefinitionRefs : EventDefinition [*] = instances of EventDefinition applied to the events of the triggers of a call operation action to which the stereotype is applied that are not owned by the stereotype applied to the action.

- ConditionalEventDefinition::/condition : BPMNExpression = self.base_ChangeEvent.changeExpression extension_BPMNExpression

- Error::/structureRef [0..1] = self.base_Class
- BPMNMessage::/itemRef : ItemDefinition [0..1] = self.base_Class.extension_ItemDefinition
- Escalation::/structureRef : [0..1] = self.base_Class
- BPMNSignal::/structureRef [0..1] = self.base_Class

Constraints

[1] IntermediateThrowEvents and ImplicitThrowEvents may only be applied to CallOperationActions.
[2] End event stereotypes with no event definitions (owned or referenced) may only be applied to FlowFinalNodes, otherwise they may only be applied to CallOperationActions.
[3] StartEvents with no event definitions (owned or referenced) may only be applied to InitialNodes, otherwise,it may only be applied to AcceptEventActions.
[4] IntermediateCatchEvents and BoundaryEvents may only be applied to AcceptEventActions.

[5] CatchEvent.eventDefinitions must be the same or a subset of the catch event stereotypes applied to events of the triggers of an accept event action to which the stereotype is applied.
[6] AcceptEventActions with CatchEvent Applied must have isUnmarshall = false.
[7] ThrowEvent.eventDefinitions must be the same as the throw event stereotypes applied to events of the triggers of a call operation action to which the stereotype is applied.
[8] AcceptEventActions with catch event stereotypes applied must have triggers that have a specialization of EventDefinition applied.
[9] AcceptEventActions with BoundaryEvent applied must be directly contained by interruptible regions.
[10] Operations invoked by CallOperationActions with throw event stereotypes applied must have no methods.
[11] Operations invoked by CallOperationActions with throw event stereotypes applied that have event definitions (owned or referenced) with ErrorEventDefinition, EscalationEventDefinition, or SignalEvent Definition applied have one in parameter with no type, and multiplicity 1, otherwise the operations have no in parameters.
[12] The operations invoked by CallOperationActions with throw event stereotypes applied must have no out or return parameters, except if they have event definitions 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.
[13] CallOperationActions with throw event stereotypes applied must have isSynchronous = false, except when CompensateEventDefinition is applied to their triggers with waitForCompletion = true, whereupon isSynchronous = true.
[14] CallOperationActions with throw event stereotypes applied, and a trigger with SignalEventDefinition applied must have no incoming edge to their target pins.
[15] CallOperationActions 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.

[16] ErrorEventDefinitions, MessageEventDefinitions, EscalationEventDefinitions, SignalEventDefinitions, and CompensateEventDefinitions may only be applied to CallEvents.
[17] ConditionalEventDefinitions and TimerEventDefinitions may only be applied to ChangeEvents.

[18] ChangeEvents with TimerEventDefinition applied have changeExpression = timeCycle, timeDate, or timeDuration, whichever has a value.

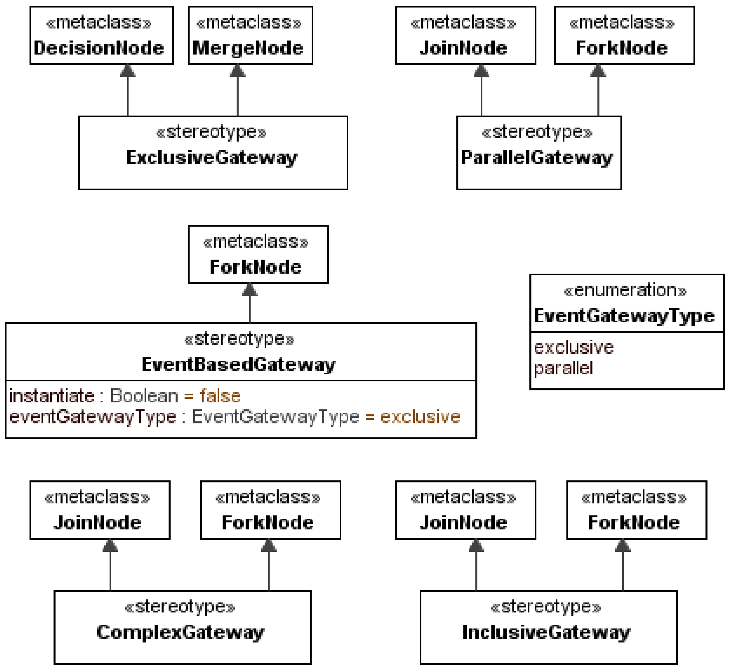
Mappings

```
relation FlowNodeToActivityNode
{ checkonly domain bpmn fn : FlowNode { };
  enforce domain uml an : uml::ActivityNode { };
  where {
        //map Events
        StartEventToInitialNode( fn, an );
        IntermediateCatchEventToAcceptEventAction( fn, an );
        BoundaryEventToAcceptEventAction( fn, an );
        //map Gateways
        //map Activities
  }
}

relation StartEventToInitialNode
//map BPMN StartEvent with no EventDefinitions to UML InitialNode with StartEvent stereotype //
applied.
{ checkonly domain bpmn se : StartEvent {
                              isInterrupting = ii : Boolean };
  enforce domain uml umlIN : uml::InitialNode { };
  //apply StartEvent stereotype to umlIN
  enforce domain uml usev : bpmnprofile::StartEvent {
                        base_InitialNode = umlIN,
                        isInterrupting = ii };
}

relation IntermediateCatchEventToAcceptEventAction
{ checkonly domain bpmn ice : IntermediateCatchEvent {
                                name = n : String,
                                //inherited attribute
                                parallelMultiple = pm : Boolean };
  enforce domain uml umlAEA : uml::AcceptEventAction {
                                name = m };
  //apply stereotype and copy attributes
  enforce domain uml uice : bpmnprofile::IntermediateCatchEvent {
                        base_AcceptEventAction = umlAEA
                        parallelMultiple = pm };
}
```

8.5 Gateways



«metaclass» DecisionNode
«metaclass» MergeNode
«metaclass» JoinNode
«metaclass» ForkNode
«stereotype» ExclusiveGateway
«stereotype» ParallelGateway
«metaclass» ForkNode
«enumeration» EventGatewayType
exclusive
parallel
«stereotype» EventBasedGateway
instantiate : Boolean = false
eventGatewayType : EventGatewayType = exclusive
«metaclass» JoinNode
«metaclass» ForkNode
«metaclass» JoinNode
«metaclass» ForkNode
«stereotype» ComplexGateway
«stereotype» InclusiveGateway

Figure 15: Gateways

The bases of the gateway stereotypes reflect differences in the models of BPMN gateways and UML control nodes. BPMN Gateways can be converging, diverging, both, or neither (except for event-based gateways, which are not converging), while UML JoinNodes and MergeNodes have one exactly 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 stores this as separate fork and join nodes, while BPMN stores it as a single gateway that is both converging and diverging.

BPMN Event-based Gateways wait for one of the events immediately following them to occur, after which tokens flow past that event (see exception for process instantiation below). The EventBasedGateway stereotype requires the accept event actions immediately after it to be contained in an interruptible region, and the outgoing edges from them to 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.

The BPMN instantiate property on event-based gateways indicates whether new process instances are created when events immediately following the gateway occur (see Section 8.1 about process instances). Event-based gateways that instantiate processes have no incoming flows. In this case, the EventBasedGateway stereotype loosens syntactic constraints on UML ForkNodes to enable them to have no incoming edges, and the Process stereotype modifies UML Activity semantics to be equivalent to BPMN's by behaving as if tokens were offered to accept event actions following forks with EventBasedGateway applied and no incoming edges when the activity is not instantiated, and instantiating the activity when one of the events is accepted following the forks.

The BPMN eventGatewayType property is only relevant when instantiating processes, and changes the above semantics for parallel event-based gateways to require all events immediately following the gateway to occur before the process instance completes. In this case, the EventBasedGateway stereotype prohibits the accept event actions immediately following it from being in an interruptible region. 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.

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

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 ComplexEventGateway stereotype extends the semantics of UML JoinNodes to be equivalent to BPMN's when the joinSpec property has the value "BPMNInclusive", 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,
- does not include the join node.

See the SequenceFlow stereotype for other semantics of the ExclusiveGateway, InclusiveGateway, and ComplexGateway stereotypes.

Derived Properties

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

Constraints

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

8.6 Data

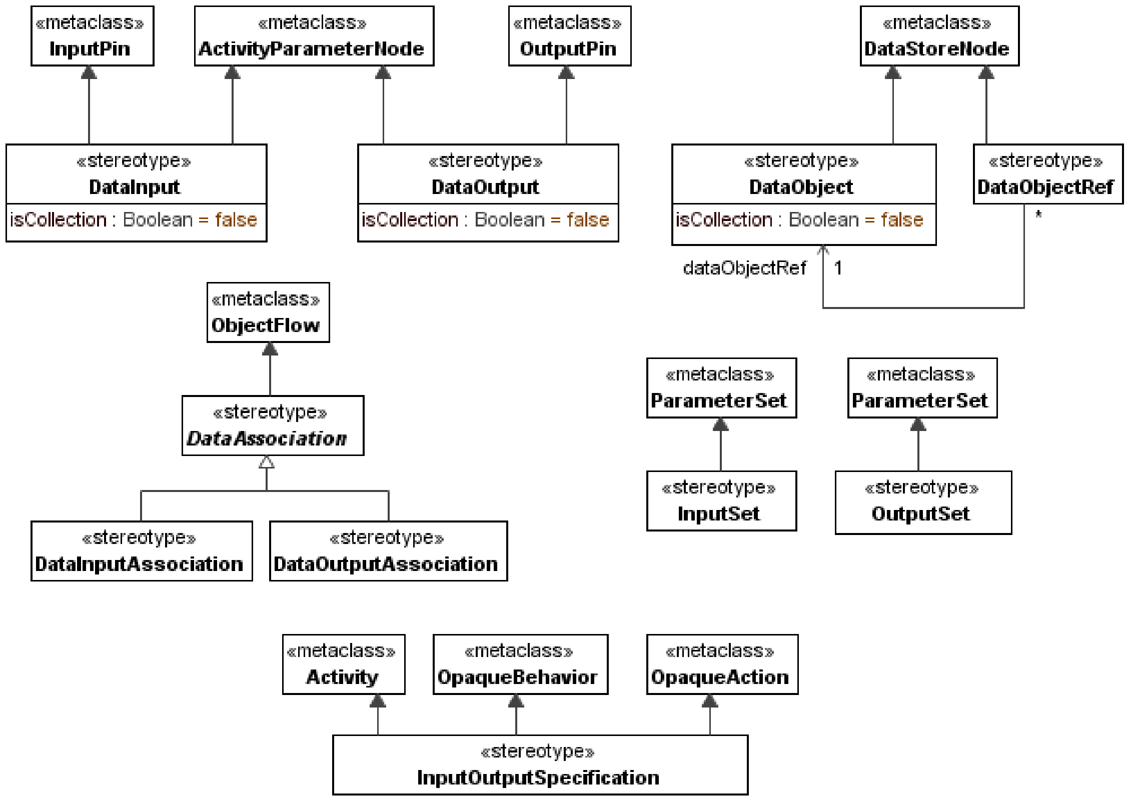

«metaclass» InputPin
«metaclass» ActivityParameterNode
«metaclass» OutputPin
«metaclass» DataStoreNode
«stereotype» DataInput
isCollection : Boolean = false
«stereotype» DataOutput
isCollection : Boolean = false
«stereotype» DataObject
isCollection : Boolean = false
«stereotype» DataObjectRef
*
dataObjectRef
1
«metaclass» ObjectFlow
«stereotype» DataAssociation
«stereotype» DataInputAssociation
«stereotype» DataOutputAssociation
«metaclass» ParameterSet
«metaclass» ParameterSet
«stereotype» InputSet
«stereotype» OutputSet
«metaclass» Activity
«metaclass» OpaqueBehavior
«metaclass» OpaqueAction
«stereotype» InputOutputSpecification

Figure 16: Data

Data flow modeling in BPMN and UML use different terminology, but take similar approaches. Both require matching input and output specifications on elements within a flow (activities in BPMN and actions in UML) and reusable global elements (processes and global tasks in BPMN and behaviors in UML). BPMN models this more efficiently, using the same specification structure for the matching specifications (InputOutputSpecification), while in UML they are different. This is reflected in multiple bases for the DataInput and DataOutput stereotypes (pins are on actions in UML, and activity parameter nodes are on behaviors). BPMN does not notate data input and output on activities, while UML has an optional notation for pins. BPMN always interposes another data element when flowing items between data outputs of one activity and data inputs of another (BPMN DataObjects), while UML has the option 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 hold items while the process is occurring. Items flowing into BPMN DataObjects replace any that are there already, and values flowing out of them leave duplicates behind. The DataObject

BPMN and UML support grouping of inputs and outputs into alternative sets, 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). The profile does not extend UML to support alternative sets of inputs and outputs on elements within a flow (actions). These groupings in BPMN specify which inputs and outputs are optional, and also specify which inputs can be accepted during execution, rather that at the beginning of execution, and which outputs can be provided during execution, rather that at the end of execution ("whileExecuting" inputs and outputs in BPMN, and streaming parameters in UML). These characteristics in UML are specified on the inputs and outputs directly, 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.

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.

BPMN supports any kind of assignment to be specified on data associations, even ones unrelated to the data association. Since data associations already assign values from their targets from their sources, and have expressions to specify transformation of those values (see derived properties), the profile does not support assignments on data associations.

Derived Properties

- InputSet::/dataInputRefs : DataInput [*] = self.base_ParameterSet.parameter extension_DataInput
- InputSet::/optionalInputRefs : DataInput [*] = instances of DataInput applied to self.base_ParameterSet.parameters with multiplicity lower = 0
- InputSet::/whileExecutingInputRefs : DataInput [*] = instances of DataInput applied to self.base_ParameterSet.parameters with isStreaming = true.
- OutputSet::/dataOutputRefs : DataOutput [*] = self.base_ParameterSet.parameter extension_DataOutput
- OutputSet::/optionalOutputRefs : DataOutput [*] = instances of DataOutput applied to self.base_ParameterSet.parameters with multiplicity lower = 0
- OutputSet::/whileExecutingOutputRefs : DataOutput [*] = instances of DataOutput with instances applied to self.base_ParameterSet.parameters with isStreaming = true
- DataInput::/itemSubjectRef : ItemDefinition [0..1] = self.base_ObjectNode.type extension_ItemDefinition (defined on TypedElement)
- DataInput::/inputSetRefs : InputSet [*] = self.base_Parameter.parameterSet extension_InputSet
- DataInput::/inputSetWithOptional : InputSet [*] = self.base_Parameter.parameterSet extension_InputSet if the parameter multiplicity lower = 0, empty otherwise.
- DataInput::/inputSetWithWhileExecuting : InputSet [*] = self.base_Parameter.parameterSet extension_InputSet if the parameter isStreaming = true, empty otherwise.
- DataOutput::/itemSubjectRef : ItemDefinition [0..1] = self.base_ObjectNode.type extension_ItemDefinition (defined on TypedElement)
- DataOutput::/outputSetRefs : OutputSet [*] = self.base_Parameter.parameterSet extension_OutputSet

- DataOutput::/outputSetWithOptional : OuputSet [*] = self.base_Parameter.parameterSet extension_OutputSet if the parameter multiplicity lower = 0, empty otherwise.
- DataOutput::/outputSetWithWhileExecuting : OutputSet [*] = self.base_Parameter.parameterSet extension_OutputSet if the parameter isStreaming = true, empty otherwise.
- InputOutputSpecification::/dataInputs : DataInput [*] = self.base_Behavior.ownedParameter extension_DataInput when applied to behaviors, and self.base_Action./input extension_DataInput when applied to actions.
- InputOutputSpecification::/dataOutputs : DataOutput [*] = self.base_Behavior.ownedParameter extension_DataOutput when applied to behaviors, and self.base_Action./output extension_DataOutput when applied to actions.
- InputOutputSpecification::/inputSets : InputSet [1..*] = self.base_Behavior.ownedParameterSet extension_InputSet when applied to behaviors, and self.base_CallBehaviorAction.behavior.ownedParameterSet extension_InputSet when applied to call behavior actions.
- InputOutputSpecification::/outputSets : OutputSet [1..*] = self.base_Behavior.ownedParameterSet extension_OutputSet when applied to behaviors, and self.base_CallBehaviorAction.behavior.ownedParameterSet extension_OutputSet when applied to call behavior actions, otherwise empty.
- CallableElement::/ioSpecification :: InputOutputSpecification [0..1] = self extension_InputOutputSpecification
- Task::/ioSpecification :: InputOutputSpecification [0..1] = self extension_InputOutputSpecification
- DataObjectRef::/dataState : State [0..1] = self.base_DataStoreNode.inState (defined on ObjectNode)
- BPMNActivity ::/dataInputAssociations [*] : DataInputAssociation = self.base_Action./input.incoming extension_DataInputAssociations
- BPMNActivity ::/dataOutputAssociations [*] : DataOutputAssociation = self.base_Action./output.outgoing extension_DataOutputAssociations
- ThrowEvent ::/dataInputAssociation [*] : DataInputAssociation = self.base_Action./input.incoming extension_DataInputAssociations
- CatchEvent ::/dataOutputAssociation [*] : DataOutputAssociation = self.base_Action./output.outgoing extension_DataOutputAssociations
- DataAssociation::/transformation : FormalExpression [0..1] = self.base_ObjectFlow.transformation extension_FormalExpression

Constraints

[1] DataInputAssociations may only be applied to ObjectFlows that have a DataStoreNode as source, and an InputPin as target.

[2] DataOutputAssociations may only be applied to ObjectFlows that have an OutputPin as source and a DataStoreNode as target.

[3] DataStoreNodes with DataObjectRef applied must have the same type, multiplicity, and ordering, and uniqueness as the DataStoreNodes with DataObject applied they refer to.

[4] ObjectNodes with DataObject, DataObjectRef, DataInput, DataOutput applied have upper bound = 1 if isCollection = false, and upperbound = * otherwise.

[5] ActivityParameterNodes with DataInput stereotype applied must have direction = in or inout.

[6] ActivityParameterNodes with DataOutput stereotype applied must have direction = out, inout, or return.

[7] The Parameters in ParameterSets with InputSet applied must all have direction = in.

[8] The Parameters in ParameterSets with OutputSet applied must all have direction = out or return.

8.7 LoopCharacteristics

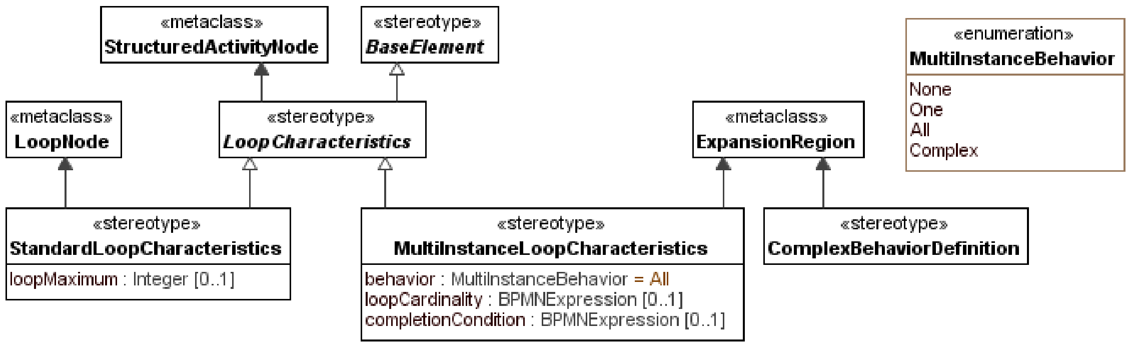

«metaclass»
StructuredActivityNode
«stereotype»
BaseElement
«enumeration»
MultiInstanceBehavior
None
One
All
Complex
«metaclass»
LoopNode
«stereotype»
LoopCharacteristics
«metaclass»
ExpansionRegion
«stereotype»
StandardLoopCharacteristics
loopMaximum : Integer [0..1]
«stereotype»
MultiInstanceLoopCharacteristics
behavior : MultiInstanceBehavior = All
loopCardinality : BPMNExpression [0..1]
completionCondition : BPMNExpression [0..1]
«stereotype»
ComplexBehaviorDefinition

Figure 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.[26] BPMN SequenceFlows linked to the BPMN Activity translate under the profile to control flows linked to the loop node or expansion region.[27]

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 StandardLoopCharacteristics. Loop nodes supports a condition test before or after each iteration, and the StandardLoopCharacteristics stereotype extends UML LoopNode semantics to prevent more than the number of iterations specified by the loopMaximum property from occurring, providing a semantics equivalent to BPMN's.

BPMN MultiInstanceLoopCharacteristics adds to BPMN Activities an indication of whether the executions of the activity happen sequentially, like they do in BPMN StandardLoopCharacteristics, or in parallel. UML ExpansionRegions support this, and 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 MultiInstanceLoopCharacteristics. 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 ExpansionRegion with the completionCondition property, and specializes UML ExpansionRegion semantics to terminate itself and actions occurring in it 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. The first is an expression that is evaluated just before the activity begins to determine how many times to execute it. The MultiInstanceLoopCharacteristics stereotype

[26] BPMN execution semantics refers to these as the "outer" and "inner" activities, respectively, even though they are captured as one activity in the BPMN abstract syntax.
[27] Translation of BPMN DataAssociations linked to looping BPMN Activities are not affected by the UML LoopNode or UML ExpansionRegion wrapper (the data associations translate to object flows linked to pins of the action inside the loop or region). BPMN does not give any special semantics for this case, so the profile assumes the "inner" activity (see footnote 26) accepts and provides data like any other BPMN Activity, which is equivalent to UML semantics.

modifies the abstract syntax of UML ExpansionRegion to not require an input, and when there is no input modifies the semantics of the action to evaluate the loopCardinality property before it begins to determine the number of times to execute its contents. The second way BPMN MultiInstanceLoopCharacteristics specifies 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.[28] 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.[29] 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 events to throw after iterations are complete under various conditions:[30]

- 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.

- 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.

- 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.

If the event definitions of the ImplicitThrowEvents are not specified by the modeler, they are assumed to be BPMN Signals with the properties specified by BPMN for multi-instance activity instances.[31]

Derived Properties
- BPMNActivity::/loopCharacteristics : LoopCharacteristics [0..1] = self.base_Action.inStructuredNode.extension_LoopCharacteristics
- StandardLoopCharacteristics::/testBefore : Boolean = self.base_LoopNode.isTestedFirst
- StandardLoopCharacteristics::/loopCondition : BPMNExpression= self.base_LoopNode.test.value.extension_BPMNExpression
- MultiInstanceLoopCharacteristics::/isSequential : Boolean = (mode = iterative)

[28] BPMN is inconsistent in how it models and describes the inputs of BPMN Tasks with MultiInstanceLoopCharacteristics, primarily because some of the specification refers to "outer" and "inner" activities (see footnote 26) while some does not, as well as because of seemingly inadvertant errors in the description of MultiInstanceLoopCharacteristics. The profile assumes BPMN Tasks with MultiInstanceLoopCharacteristics that loop over collections take individual elements of the collection as input, with data associations from other elements that provide collections.
[29] The comments of footnote 28 apply here also. The profile assumes BPMN Tasks with MultiInstanceLoopCharacteristics provide individual elements of collections as output, with data associations to other elements that accept collections.
[30] BPMN requires ImplicitThrowEvents only for the third case, but the profile assumes them in the first two cases also, to reuse throw event stereotype structure and semantics.
[31] BPMN gives this default only for the first two cases above, but the profile assumes it in the third case also, for uniformity.

- MultiInstanceLoopCharacteristics::/loopDataInputRef [0..1] = self.base_ExpansionRegion.inputElement.incoming.source
- MultiInstanceLoopCharacteristics::/loopDataOutputRef [0..1] = self.base_ExpansionRegion.outputElement.outgoing.target
- MultiInstanceLoopCharacteristics::/inputDataItem : DataInput = self.base_ExpansionRegion.node.input.extension_DataInput for the node with BPMNActivity applied.
- MultiInstanceLoopCharacteristics::/outputDataItem : DataOutput = self.base_ExpansionRegion.node.output.extension_DataOutput for the node with BPMNActivitity applied.
- MultiInstanceLoopCharacteristics::/oneBehaviorEventRef : EventDefinition [0..1] = self.base_ExpansionRegion.node.eventDefinitions.extension_EventDefinition for the node with ImplicityThrowEvent applied, if any.
- MultiInstanceLoopCharacteristics::/noneBehaviorEventRef : EventDefinition [0..1] = self.base_ExpansionRegion.node.eventDefinitions.extension_EventDefinition for the node with ImplicitThrowEvent applied, if any.
- MultiInstanceLoopCharacteristics::/complexBehaviorDefinition : ComplexBehaviorDefinition [*] = self.base_ExpansionRegion.extension_ ComplexBehaviorDefinition
- ComplexBehaviorDefinition::/condition : FormalExpression = instance of FormalExpression applied to guard of sequence flow going out of decision node corresponding to the complex behavior definition that comes into a call operation action with ImplictThrowEvent applied.
- ComplexBehaviorDefinition::/event : ImplicitThrowEvent [0..1] = instance of ImplicitThrowEvent applied to call operation action targeted by sequence flow coming out of decision node corresponding to the complex behavior definition.

Constraints

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

[2] ComplexBehaviorDefinitions may only be applied to elements that have MultiInstanceLoopCharacteristics applied.

[3] LoopNodes with the StandardLoopCharacteristics stereotype applied have an empty setPart, one action in the bodyPart, which has a specialization of BPMNActivity applied, and one ValueSpecificationAction as test.

[4] ExpansionRegions with MultiInstanceLoopCharacteristics applied must have an iterative or parallel mode.

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

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

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

[8] ExpansioRegions with MultiInstanceLoopCharacteristics applied and an inputElement must contain an object flow from the input element to the input pin of the action it contains with BPMNActivity applied.

[9] ExpansionRegions 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.

[10] OpaqueExpressions must have FormalExpression applied when they are owned by sequence flows coming into call operation actions with ImplicitThrowEvent applied in expansion regions with ComplexBehaviorDefinition applied.

8.8 Lanes and Resources

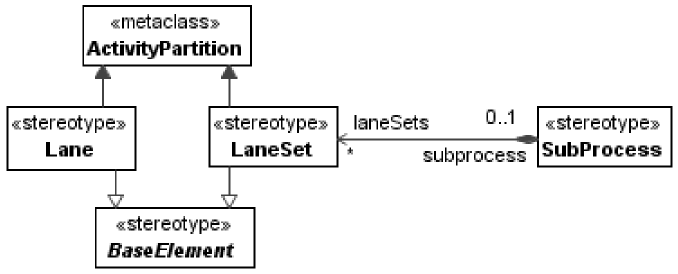


Figure 18: Lanes and LaneSets

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. The LaneSet stereotype requires UML ActivityPartitions to have their isDimension property set to true, and removes the UML constraint that dimension partitions cannot be contained in other partitions.

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 elments in the same way, but not own them. The Lane stereotype extends UML ActivityPartition to own 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.

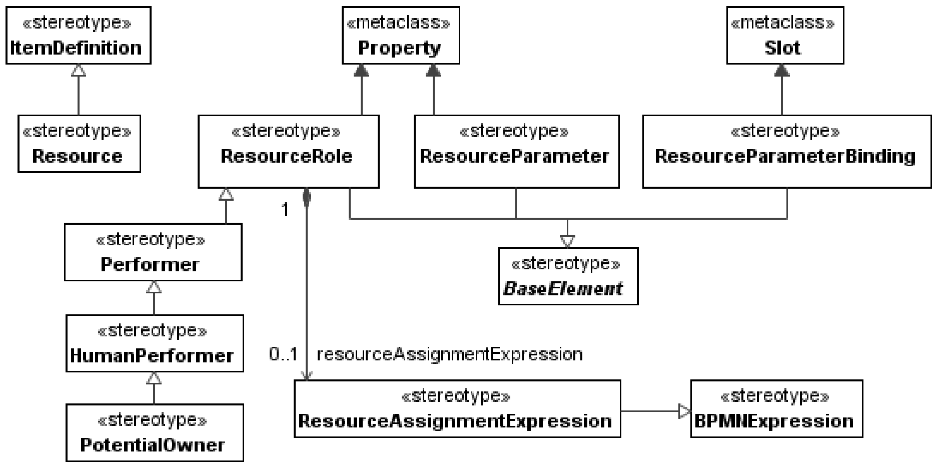

«stereotype» ItemDefinition
«metaclass» Property
«metaclass» Slot
«stereotype» Resource
«stereotype» ResourceRole
«stereotype» ResourceParameter
«stereotype» ResourceParameterBinding
1
«stereotype» Performer
«stereotype» BaseElement
«stereotype» HumanPerformer
0..1
resourceAssignmentExpression
«stereotype» PotentialOwner
«stereotype» ResourceAssignmentExpression
«stereotype» BPMNExpression

Figure 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 typically 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 Sections 8.1 and 8.2), 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 stereotyped by ResourceParameterBinding 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.

Derived Properties

- LaneSet::/lanes : Lane [*] = self.base_ActivityPartition.subpartition.extension_Lane
- Lane::/laneSet : Lane = self.base_ActivityPartition.superPartition.extension_Lane
- LaneSet::/parentLane : Lane [*] = self.base_ActivityPartition.superpartition.extension_Lane
- Lane::/childLaneSet : Lane = self.base_ActivityPartition.subPartition.extension_Lane
- Lane::/flowNodeRefs [*] = self.base_ActivityPartition.node
- Process::/laneSets : LaneSet [*] = self.base_Activity.partition.extension_LaneSet
- LaneSet::/flowElementsContainer [0..1] = self.base_ActivityPartition.inActivity.extension_Process (defined on ActivityGroup) or LaneSet.subprocess, whichever has a value.
- Lane::/partitionElementRef : BaseElement [0..1] = self.base_ActivityPartition.represents.extension_BaseElement

- CallableElement::/resources : ResourceRole [*] = self.base_Behavior.ownedAttribute.extension_ResourceRole (defined on Class)
- ResourceRole::/process : Process [0..1] = self.base_Property.class.extension_Process
- BPMNActivity::/resources : ResourceRole [*] = self.activityClass.ownedAttribute.extension_ResourceRole
- ResourceRole::/resourceRef : Resource [0..1] = self.base_Property.type.extension_Resource (defined on TypedElement)
- Resource::/resourceParameters : ResourceParameter [*] = self.base_Class.ownedAttribute.extension_ResourceParameter
- ResourceParameter::/type : ItemDefinition [0..1] = self.base_Property.type.extension_ItemDefinition (defined on TypedElement)
- ResourceParameter::/isRequired : Boolean = self.base_Property.lower > 0 (defined on MultiplicityElement)
- ResourceParameterBinding::/parameterRef : ResourceParameter = self.base_Slot.definingFeature.extension_ResourceParameter
- ResourceParameterBinding::/expression : BPMNExpression = self.base_Slot.value.extension_BPMNExpression
- ResourceRole:/resourceParameterBindings : ResourceParameterBinding [*] = self.base_Property.defaultValue.slot
- ResourceAssignmentExpression::/expression : BPMNExpression = self

Constraints

[1] Lanes and LaneSets may only be applied to ActivityPartitions.

[2] ActivityPartitions with LaneSet stereotype applied must have isDimension=true.
[3] On Partitions with LaneSet applied, either the inActivity property must have a value, or the subprocess property of the lane set must have a value, but not both.
[4] Lane.partitionElement must be included in ActivityPartition.represents

[5] ResourceRoles and ResourceParameters may only be applied to Properties.
[6] ResourceParameterBindings may only be applied to Slots.

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

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

9 Collaborations

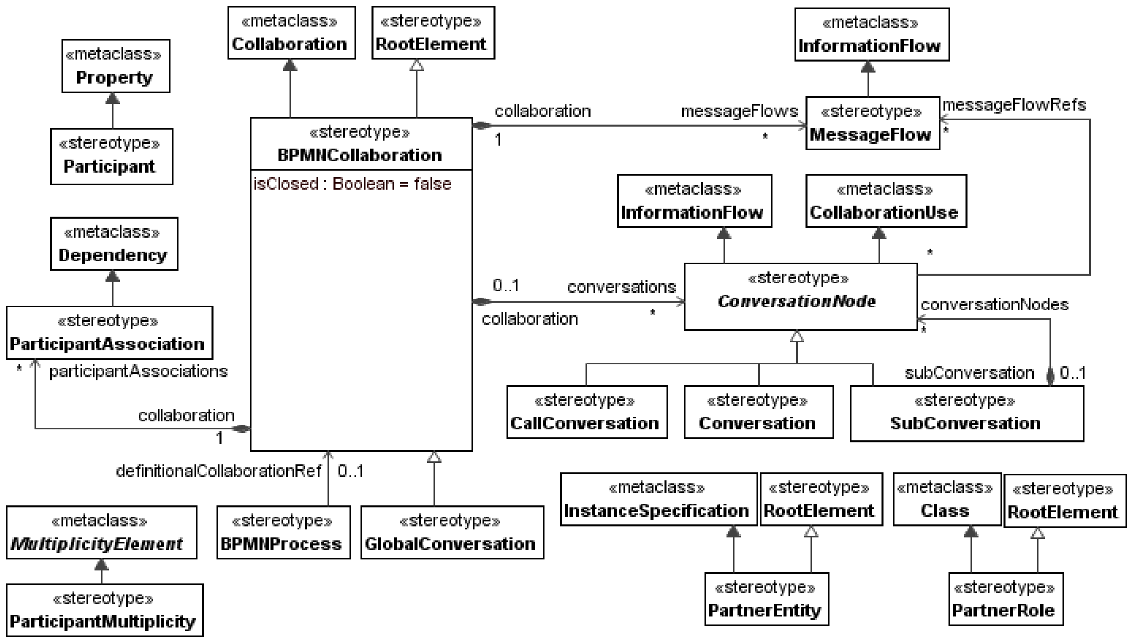

«metaclass» Property
«stereotype» Participant
«metaclass» Collaboration
«stereotype» RootElement
«stereotype» BPMNCollaboration
isClosed : Boolean = false
«metaclass» InformationFlow
«stereotype» MessageFlow
collaboration 1
messageFlows *
messageFlowRefs *
«metaclass» InformationFlow
«metaclass» CollaborationUse
«stereotype» ConversationNode
0..1 conversations *
collaboration
conversationNodes *
subConversation 0..1
«metaclass» Dependency
«stereotype» ParticipantAssociation
* participantAssociations
collaboration 1
«stereotype» CallConversation
«stereotype» Conversation
«stereotype» SubConversation
definitionalCollaborationRef 0..1
«metaclass» MultiplicityElement
«stereotype» ParticipantMultiplicity
«stereotype» BPMNProcess
«stereotype» GlobalConversation
«metaclass» InstanceSpecification
«stereotype» RootElement
«stereotype» PartnerEntity
«metaclass» Class
«stereotype» RootElement
«stereotype» PartnerRole

Figure 20: Collaborations

BPMN Collaborations and UML Collaborations share the same name and purpose. They both help specify interactions between processes or behaviors that can be carried out by entities or objects participating in the collaboration. Neither BPMN and UML Collaborations capture all interaction details, they focus on the flows between participants, using grouping and reuse of flows to scale to complicated interactions. The time sequence of 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.

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 things that participate, and UML Properties are in always in exactly one classifier. The Participant stereotype indicates which properties of UML Collaborations are participants, providing semantics equivalent to BPMN's. BPMN PartnerEntities are individual things, such as WalMart or particular persons, while BPMNPartnerRoles are the kinds of things that play roles, such as buyers or sellers.[32] Partner entities and roles can refer to participants, specifying the individual things or kinds of things that play 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 have default values to specify individual things that will be values of the properties, and also have types to specify the kinds of things those individuals can be, 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. In this case, BPMN Collaborations specify the

[32] BPMN does not specify what PartnerRoles are roles of. The profile assumes they are not roles of collaborations, since those are BPMN Participants. The profile takes them to be types of things that participate in collaborations, possibly of properties of other classes, such as organization types.

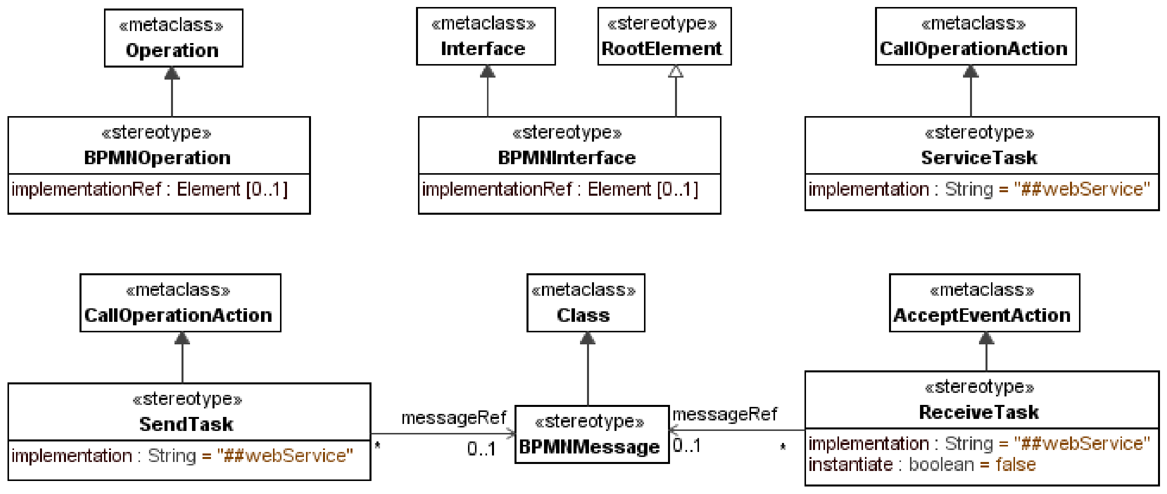
interactions of processes directly, and the BPMNCollaboration stereotype requires the participant properties of the collaboration to have UML Activities as types, providing a semantics equivalent to BPMN's (see Section 8.1 about activities as types).

BPMN MessageFlows specify messages transmitted between participants, or elements of processes that send and receive messages. BPMN Messages specify the kind 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 MessageFlow stereotype constrains information flows, defines derived properties, and loosens the lower multiplicity of the conveyed property to zero, to provide abstract syntax and semantics equivalent to BPMN's. The BPMN Collaboration stereotype extends UML Collaborations to own information flows with the messageFlows property. The Message stereotype is a specialization of ItemDefinition (see Section 8.4) 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 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 CallConversations reuse other collaborations, associating the participants in the "caller" with the participants in the reused collaboration. UML Collaboration does the same with UML CollaborationUse, linking participants with dependencies that the collaboration use identifies as role bindings. BPMN does not give a semantics to associating participants between collaborations, but it is typically read as requiring the individuals playing the roles specified by the associated participants to be the same when the calling collaboration is carried out. UML provides equivalent semantics for collaboration role bindings. BPMN Conversations group message flows owned by the containing collaboration, possibly sharing message flows across conversations. 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. 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 own any of the other three kinds of conversation, providing a construct equivalent to BPMN's. 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 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 from UML CollaborationUse, or by extension of UML InformationFlows.

BPMN Processes use collaborations to specify how processes 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 the process, as expressed by the BPMN definitionalCollaborationRef property of processes. The Process stereotype extends UML Activities with a property of the same name to identify such collaborations. When a BPMN Process in a collaboration calls another processes, 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 own dependencies for this purpose.

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 can occur when the collaboration is carried out. If 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 carried out with message flows not specified in their models, giving a semantics equivalent to BPMN's.



«metaclass» Operation
«stereotype» BPMNOperation
implementationRef : Element [0..1]
«metaclass» Interface
«stereotype» RootElement
«stereotype» BPMNInterface
implementationRef : Element [0..1]
«metaclass» CallOperationAction
«stereotype» ServiceTask
implementation : String = "##webService"
«metaclass» CallOperationAction
«stereotype» SendTask
implementation : String = "##webService"
messageRef
*
0..1
«metaclass» Class
«stereotype» BPMNMessage
messageRef
0..1
*
«metaclass» AcceptEventAction
«stereotype» ReceiveTask
implementation : String = "##webService"
instantiate : boolean = false

Figure 21: Operations, Interfaces, and Related Tasks

BPMN and UML Collaborations are typically used in conjunction with operations and interfaces, which specify capabilities that collaborators offer to each other. BPMN Operations and Interfaces 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. They 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 BPMNMessage applied, and to have exactly one input parameter and at most one output, to match BPMN restrictions. Interfaces in both languages group operations that are typically used together. 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, while UML Interfaces can be supported only by behaviored classifiers, including activities and opaque behaviors, although UML interfaces can be the type of UML properties. The Participant stereotype loosens the constraint on UML Properties that they only have one type, and extends UML Property semantics by requiring property values to be classified by all of a property's types. This enables properties with the Participant stereotype applied to be typed by multiple interfaces, as well as classes and activities. Combined with the constraint that UML Activities with the Process stereotype applied are their own classifier behaviors, actions can accept operations directed to the containing activity, providing semantics equivalent to BPMN's.

BPMN SendTask, ServiceTasks, and ThrowEvents with message event definitions (see Sections 8.2 and 8.4) can call operations on things outside the containing process, typically other participants in collaborations. BPMN SendTasks and throw message events can specify outgoing messages[33] without specifying operations, or vice versa, while ServiceTasks can only specify outgoing messages via operations. This is reflected in the profile by the messageRef associations from SendTask and MessageEventDefinition, but not ServiceTask. The participants receiving operation calls are identified by message flows going out of the send task, service task, or throw event. SendTask, ServiceTask, or throw event stereotypes with message event definitions, applied to UML CallOperationActions require their

[33] BPMN requires messages to be sent on all outgoing message flows for a send task to complete, and does not give an exception to this when combined with an operation. The SendTask and throw event stereotypes with message event definitions extend the semantics of UML CallOperationAction to target all of the participants identified by outgoing message flows.

targets to be values of properties with Participant applied, as determined by outgoing information flows with MessageFlow applied, if any, 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 ReceiveTask and catch events with MessageEventDefinitions (see Sections 8.2 and 8.4) 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 SendTask and throw event stereotypes constrain the semantics of UML AcceptEventAction to accept events from participants determined by the incoming message flows. BPMN ReceiveTask and MessageEventDefinitions can specify incoming messages without specifying operations, and vice-versa, which is reflected in the profile by the messageRef associations from ReceiveTask and MessageEventDefinition. The BPMN instantiate property on receive tasks indicates whether new process instances are created when messages of particular kinds arrive (see Section 8.1 about process instances).[34] In this case, the Process stereotype modifies UML Activity semantics to be equivalent to BPMN's by behaving as if tokens were in accept event actions with ReceiveTask applied and no incoming edges when the activity is not instantiated, and instantiating the activity when the messages arrive.

The BPMN implementation properties on SendTask, ReceiveTask, and ServiceTask identify implementation technologies outside of BPMN. The implementationRef properties on Interface and Operation identify elements modeled outside BPMN, but do not constrain them. The SendTask, ReceiveTask, ServiceTask, Interface, and Operation stereotypes extend their base classes with these properties, with implementation semantics, see the BPMN specification.

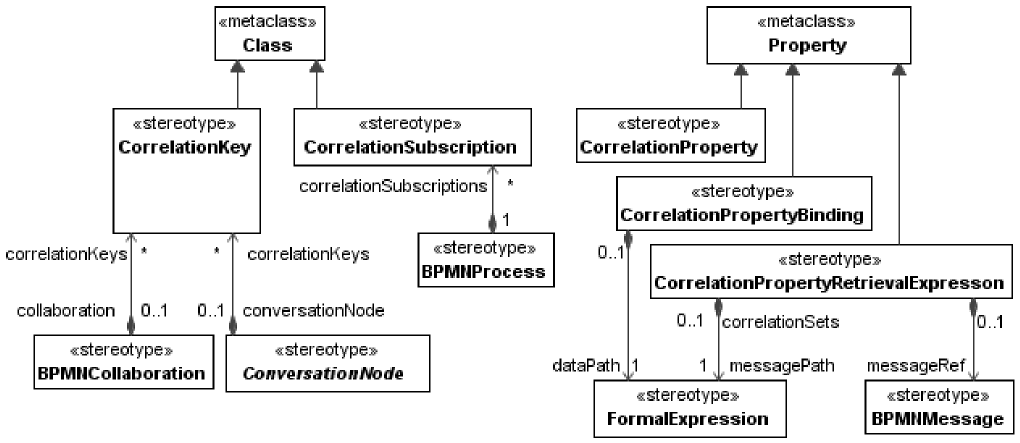

«metaclass» Class
«metaclass» Property
«stereotype» CorrelationKey
«stereotype» CorrelationSubscription
«stereotype» CorrelationProperty
«stereotype» CorrelationPropertyBinding
«stereotype» CorrelationPropertyRetrievalExpresson
«stereotype» BPMNProcess
«stereotype» BPMNCollaboration
«stereotype» ConversationNode
«stereotype» FormalExpression
«stereotype» BPMNMessage
correlationSubscriptions *
1
correlationKeys *
* correlationKeys
collaboration 0..1
0..1 conversationNode
0..1
dataPath 1
0..1 correlationSets
1 messagePath
messageRef
0..1

Figure 22: Correlation

Messages coming into participants have correlation information to determine which process instances they should be routed to (see Section 9.1 about process instances). BPMN Collaborations and the three kinds of conversation use CorrelationKeys to specify which CorrelationProperties of incoming messages they match process instances against. BPMN does not support message properties, so properties of correlation keys

[34] BPMN does not specify where messages arrive to instantiate processes. They cannot arrive at process instances, because they are not created yet. The profile assumes they arrive at a participant that contains the process.

have CorrelationPropertyRetreivalExpressions to specify how to extract property values from messages. The CorrelationKey and Message stereotypes can specify properties because they are based on UML Class, but properties can only be shared between classes via generalization. The profile assumes classes with CorrelationKey and Message applied either form a taxonomy for property sharing, or properties are matched by name, providing syntax equivalent to BPMN's. Usually mapping from UML Classes to message formats is part of a model-driven architecture for generating platform-specific technology, but PropertyRetrievalExpression stereotypes can be applied to the same properties as CorrelationProperty to provide syntax equivalent to BPMN's.

Incoming messages supply initial values for correlation properties to match against later incoming messgages for each process instance, but processes can also specify values to match against incoming messages. BPMN processes can have CorrelationSubscriptions with CorrelationPropertyBindings specifying expressions that determinine property values for matching against particular keys. The CorrelationSubscription stereotype is based on UML Class, and can have properties with the CorrelationProperty and CorrelationPropertyBinding stereotypes applied to specify expressions used to match process instances against incoming message properties. Classes with CorrelationSubscription applied are generalized by other classes with CorrelationKey applied, providing syntax equivalent to BPMN's.

Derived Properties

- Collaboration::/participants : Participant [*] = self.base_Collaboration.ownedAttribute.external_Participant (defined on Class)
- Participant::/processRef : Process [0..1] = self.base_Property.type.extension_Process if type is an activity, or Property.type.classifierBehavior.extension_Process if type is a BehavioredClassifier that is not an activity.
- ParticipantMultiplicity.minimum : Integer = self.base_MultiplicityElement.lower
- ParticipantMultiplicity.mximum : Integer = self.base_MultiplicityElement.upper when the value is an integer, empty otherwise.
- MessageFlow::/sourceRef = self.base_InformationFlow.informationSource
- MessageFlow::/targetRef = self.base_InformationFlow.informationTarget
- MessageFlow::/messageRef : Message [0..1] = self.base_InformationFlow.conveyed.extension_Message
- ConversationNode::/participantRefs : Participant [2..*] = instances of Participant stereotype applied to the informationSource and informationTarget of the information flow
- CallConversation::/calledCollaborationRef : BPMNCollaboration [0.1] = self.base_Collaboration.type.extension_CallConversation
- CallConversation::/participantAssociations : ParticipantAssociations [*] = self.base_CollaborationUse.roleBinding.extension_ParticipantAssociation
- PartnerEntity::/participantRef : Participant [*] = instances of Participant stereotype applied to properties the instance specification is the default value of.
- PartnerRole::/participantRef : Participant [*] = instances of Participant stereotype applied to properties the class is the type of.
- Participant::/partnerEntityRef : PartnerEntity [*] = inverse of PartnerEntity::/participantRef.
- Participant::/partnerRoleRef : PartnerRole [*] = inverse of PartnerRole::/participantRef.
- ParticipantAssociation::/innerParticipantRef : Participant = self.base_Dependency.supplier.extension_ParticipantAssociation
- ParticipantAssociation::/outerParticipantRef : Participant = self.base_Dependency.clients.extension_ParticipantAssociation

- BPMNInterface::/operations : BPMNOperation [*] = self.base_Interface.ownedOperation.extension_BPMNOperation
- BPMNOperation::/inMessageRef : BPMNMessage = instance of BPMNMessage applied to the type of the first parameter in self.base_Operation.ownedParameter with direction=in.
- BPMNOperation::/outMessageRef : BPMNMessage = instance of BPMNMessage applied to the type of the first parameter in self.base_Operation.ownedParameter with direction=return.

- BPMNOperation ::/errorRef : Error [*] = self.base_Operation.raisedException (defined on BehavioralFeature)
- CallableElement::/supportedInterfaceRefs : BPMNInterface [*] = self.base_Behavior.interfaceRealization.contract.extension_BPMNInterface (defined on BehavioredClassifier and InterfaceRealization)
- BPMNInterface::/callableElements : CallableElement [*] = self.base_Interface.interfaceRealization. implementingClassifier.extension_CallableElement.
- Participant::/interfaceRefs : BPMNInterface [*] = self.base_Property.type.extension_BPMNInterface
- SendTask::/operationRef : BPMNOperation [0..1] = self.base_CallOperationAction.operation.extension_BPMNOperation
- ServiceTask::/operationRef : BPMNOperation [0..1] = self.base_CallOperationAction.operation.extension_BPMNOperation
- ReceiveTask::/operationRef : BPMNOperation [0..1] = self.base_AcceptEventAction.trigger.operation.extension_BPMNOperation

- CorrelationKey::/correlationPropertyRef : CorrelationProperty[*] = self.base_Class.ownedAttribute.extension_CorrelationProperty
- CorrelationProperty::/type : ItemDefinition [0..1] = self.base_Property.type.extension_ItemDefinition (defined on TypedElement)
- CorrelationProperty::/correlationPropertyRetreivalExpression : CorrelationPropertyRetreivalExpression [1..] = self.base_Property. .extension_CorrelationPropertyRetreivalExpression.
- CorrelationSubscription::/correlationKeyRef : CorrelationKey [1] = self.base_Class.generalization.general.extension_CorrelationKey (defined on Classifier)
- CorrelationSubscription::/correlationPropertyBinding : CorrelationPropertyBinding [*] = self.base_Class.ownedAttribute.extension_CorrelationPropertyBinding
- CorrelationPropertyBinding::/correlationPropertyRef : CorrelationProperty [1] = self.base_Property.extension_CorrelationProperty

Constraints

[1] ParticipantMultiplicity may only applied to elements that have Participant applied, and vice versa.
[2] CallConversations may only be applied to ConversationUses.
[3] Conversations and SubConversations may only be applied to InformationFlows.

[4] Properties with the Participant stereotype applied must be owned by collaborations with BPMNCollaboration applied.
[5] InformationFlows with the MessageFlow stereotype applied must have exactly one informationSource and one informationTarget, both of which must have either the Participant, a BPMNActivity, CatchEvent, or ThrowEvent stereotype applied.
[6] Dependencies with the ParticipantAssociation applied must have exactly one client and one supplier, both of which must have the Participant stereotype applied.
[7] Activities with the Process stereotype applied and a definitional collaboration must be one of the values of the processRef derived property of the collaboration.

[8] ServiceTasks and SendTasks may only be applied to CallOperationActions.
[9] ReceiveTasks may only be applied to AcceptEventActions.

[10] Operations with the BPMNOperation stereotype applied must have exactly one parameter with direction=in, and the parameter must be typed by a class with Message applied that is the same as the value of the /inMessageRef property, and have multiplicity 1.
[11] Operations with the BPMNOperation stereotype applied and a value for the /outMessageRef property must have no more than one parameter with direction=return, and the parameter must be typed by a class with Message applied that is the same as the value of the /outMessageRef property, and have multiplicity 1.

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

[13] CallOperationActions with the SendTask or ServiceTask stereotypes applied must have isSynchronous = false, except when ServiceTasks have a value for the /outMessageRef property, whereupon isSynchronous = true.

[14] CorrelationKeys must be owned by either BPMNCollaborations or ConversationNodes.

[15] The types of Properties with CollaborationProperty applied must have ItemDefinition applied.

[16] CorrelationProperties with the same name that do not inherit from each other have copies of each others CorrelationPropertyRetrievalExpressions.