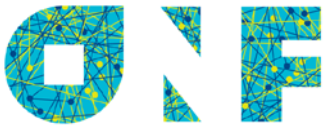


UML Modeling Guidelines

Version 1.0
March 13, 2015

ONF TR-514

ONF Document Type: Technical Recommendation
ONF Document Name: UML Modeling Guidelines V1.0

Disclaimer

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Any marks and brands contained herein are the property of their respective owners.

Open Networking Foundation
2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303
www.opennetworking.org

©2015 Open Networking Foundation. All rights reserved.

Open Networking Foundation, the ONF symbol, and OpenFlow are registered trademarks of the Open Networking Foundation, in the United States and/or in other countries. All other brands, products, or service names are or may be trademarks or service marks of, and are used to identify, products or services of their respective owners.

1 Introduction 6

2 References 6

3 Abbreviations 6

4 Overview 7

4.1 Documentation Overview 7
4.2 Modeling approach 8
4.3 General Requirements 9

5 UML Artifact Descriptions 9

5.1 Object Classes 9
5.1.1 Object Class Notation 9
5.1.2 Object Class Properties 10
5.2 Attributes in Object Classes 11
5.2.1 Attribute Notation 11
5.2.2 Attribute Properties 12
5.3 Associations 14
5.3.1 Association Notation 14
5.3.2 Association Properties 17
5.4 Interfaces 18
5.4.1 «Interface» Notation 18
5.4.2 «Interface» Properties 19
5.5 Interface Operations 20
5.5.1 Operation Notation 20
5.5.2 Operation Properties 20
5.6 Operation Parameters 22
5.6.1 Parameter Notation 22
5.6.2 Parameter Properties 23
5.7 Notifications 25
5.8 Types 25
5.8.1 Type Notation 25
5.8.2 Type Properties 26
5.9 Qualifiers 27

6 UML Profile Definitions 28

6.1 Additional ONF Properties Definitions 28
6.2 Modeling Lifecycle Definitions 30
6.3 ONF Open Model Profile Files 31

7 Recommended Modeling Patterns 31

 © Open Networking Foundation

7.1 File Naming Conventions 31
7.2 Model Structure 31
7.2.1 Generic Model Structure 31
7.2.2 ONF Specific Model Structure 32
7.3 Flexible Attribute Assignment to Object Classes 33
7.4 Use of Conditional Packages 34
7.5 Use of XOR/Choice 35
7.5.1 Xor Constraint 35
7.5.2 «Choice» 36
7.6 Diagram Guidelines 37

List of Figures

Figure 4.1: ONF Specification Architecture 8
Figure 5.1: Graphical Notation for Object Classes in ONF 9
Figure 5.2: Graphical Notation for Object Classes without Attributes Compartment 9
Figure 5.3: Graphical Notation for Object Classes with Attributes and Deprecated Operations Compartment 10
Figure 5.4: «OpenModelClass» Stereotype 10
Figure 5.5: Potential Choice Annotation for Object Classes 11
Figure 5.6: Graphical Notation for Object Classes with Attributes 12
Figure 5.7: «OpenModelAttribute» Stereotype 13
Figure 5.8: «PassedByReference» Stereotype 14
Figure 5.9: Bidirectional Association Relationship Notation 15
Figure 5.10: Unidirectional Association Relationship Notation 15
Figure 5.11: – Non-navigable Association Relationship Notation 15
Figure 5.12: Aggregation Association Relationship Notation 15
Figure 5.13: Composite Aggregation Association Relationship Notation 16
Figure 5.14: Papyrus Settings for Composite Aggregation 16
Figure 5.15: Generalization Relationship Notation (normal and conditional) 16
Figure 5.16: Dependency Relationship Notation (normal and naming) 16
Figure 5.17: Potential Annotations for Associations 18
Figure 5.18: Graphical Notation for «Interface» in ONF 19
Figure 5.19: Graphical Notation for «Interface» without Attributes Compartment 19
Figure 5.20: «OpenModelInterface» Stereotype 20
Figure 5.21: Graphical Notation for «Interface» with Operations 20
Figure 5.22: «OpenModelOperation» Stereotype 22
Figure 5.23: Graphical Notation for «Interface» with Operations and Parameters 23
Figure 5.24: «OpenModelParameter» Stereotype 24
Figure 5.25: «PassedByReference» Stereotype 24
Figure 5.26: Graphical Notation for «Signal» 25
Figure 5.27: Graphical Notation for «DataType» 25
Figure 5.28: Graphical Notation for «Enumeration» 26
Figure 5.29: Graphical Notation for «PrimitiveType» 26
Figure 5.30: Potential Annotations for Data Types 27
Figure 6.1: UML Artifact «Stereotypes» 28
Figure 6.2: Lifecycle «Stereotypes» 31
Figure 7.1: Core Model and Sub-Models 32
Figure 7.2: ONF Model Structure (snapshot) 33
Figure 7.3: Pre-defined Packages in a UML Module 33
Figure 7.4: Flexible Attribute Assignment to Object Classes 34
Figure 7.5: Enhancing Object Classes Using Conditional Packages 35
Figure 7.6: {xor} Notation 35

 © Open Networking Foundation

Figure 7.7: Information Model Element Example Using «Choice» Notation 36
Figure 7.8: Operations Model Element Example Using «Choice» Notation 37
Figure 7.9: Sink/Source/Bidirectional Termination Points Example Using «Choice» Notation 37

List of Tables

Table 5.1: Table 11.1/[3] – Collection Types for Properties 12
Table 6.1: UML Artifact Properties Defined in Complex «Stereotypes» 29

Document History

Version	Date	Description of Change
1.0	March 13, 2015	Initial version

 © Open Networking Foundation

1 Introduction

This Technical Recommendation defines the guidelines that have to be taken into account during the creation of a protocol-neutral UML (Unified Modeling Language) information model. These UML Modeling Guidelines are valid for the whole ONF; i.e., are not specific to any WG, technology or management protocol.

UML defines a number of basic model elements (UML artifacts). In order to assure a consistent ONF-wide harmonized information model, only a selected subset of these artifacts are used in the UML model guidelines in this document. The semantic of the selected artifacts is defined in [2].

The documentation of each basic model artifact is divided into three parts:

1. Short description
2. Graphical notation examples
3. Properties

The guidelines have been developed using the Papyrus open source UML tool [1].

2 References

[1] Papyrus Eclipse UML Modeling Tool (https://www.eclipse.org/papyrus/)
[2] Unified Modeling Language™ (UML®) (http://www.uml.org/)
[3] OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4
[4] 3GPP/TM Forum Model Alignment JWG: FMC Model Repertoire (ftp://ftp.3gpp.org/TSG_SA/WG5_TM/Ad-hoc_meetings/Multi-SDO_Model_Alignment/S5eMA20139.zip)

3 Abbreviations

CORBA	Common Object Request Broker Architecture
DS	Data Schema
FMC	Fixed-Mobile Convergence
HTTP	Hypertext Transfer Protocol
IM	Information Model
IMP	Information Modeling Project (ONF Services Area)
JMS	Java Message Service
JSON	JavaScript Object Notation
JWG	Joint Working Group (TM Forum, 3GPP)

 © Open Networking Foundation

LCC	Lower Camel Case
LTP	Logical Termination Point
NA	Not Applicable
NBIWG	NorthBound Interface WG (ONF)
OF	Open Flow
OMG	Object Management Group
OTWG	Optical Transport WG (ONF)
PM	Performance Monitoring
SDO	Standards Developing Organization
UCC	Upper Camel Case
UML	Unified Modeling Language
XML	Extensible Markup Language
WG	Working Group

4 Overview

4.1 Documentation Overview

This document is part of a series of Technical Recommendations. The location of this document within the documentation architecture is shown in Figure 4.1 below:

© Open Networking Foundation

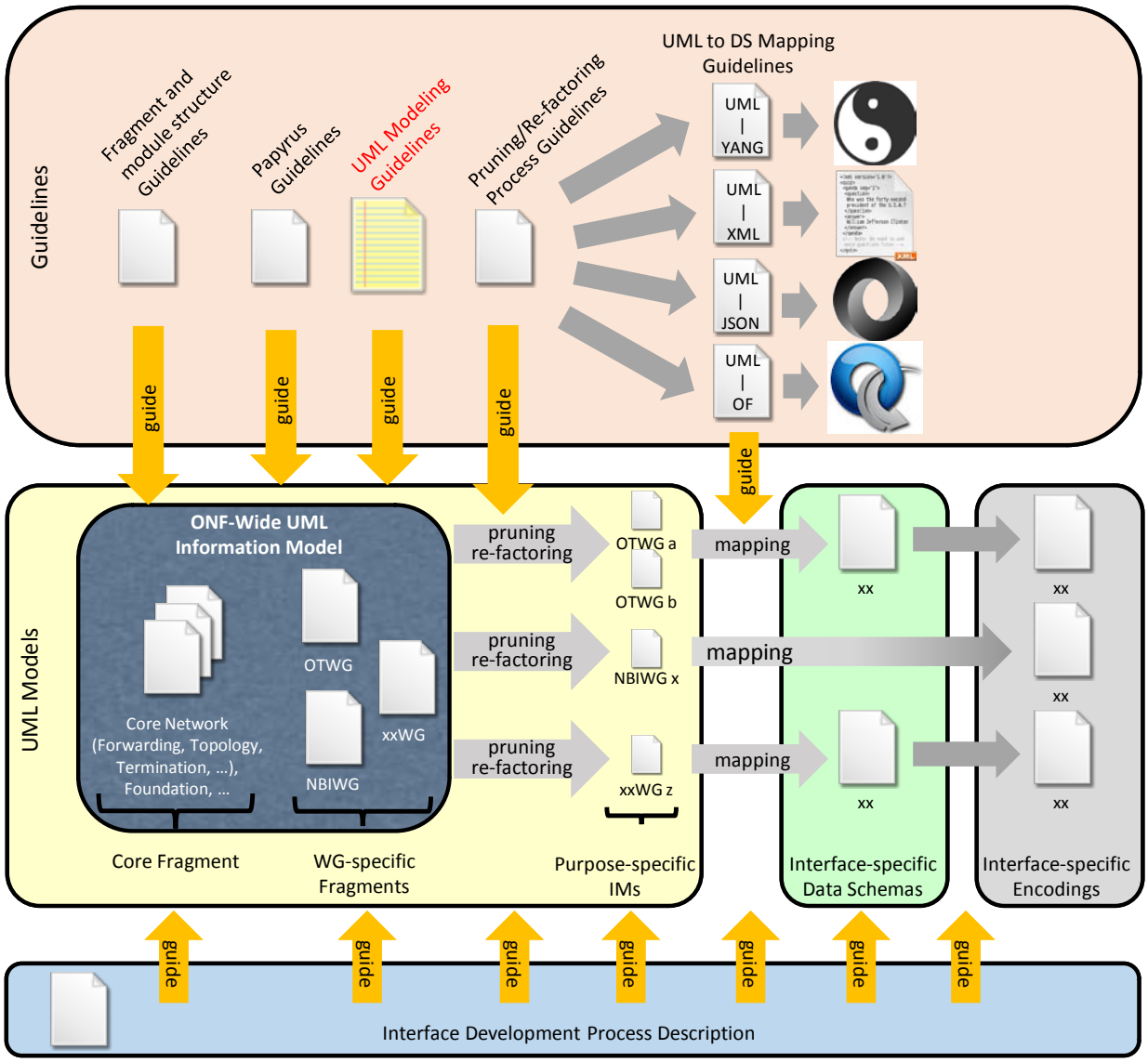


Figure 4.1: ONF Specification Architecture

4.2 Modeling approach

The information model is split into a static part and a dynamic part; i.e., data model is decoupled from operations model.

Important note:

It is important to understand that the UML class diagrams always show only parts of the underlying model data base (data dictionary). E.g., object classes shown without attributes do not mean that the object class has no attributes, they could be hidden in a diagram. The complete model is contained in the data dictionary which contains all definitions.

© Open Networking Foundation

4.3 General Requirements

- The UML 2.4 (Unified Modeling Language) is used as a notation for the model.
- The model shall be protocol-neutral, i.e., not reflect any middleware protocol-specific characteristics (like e.g., CORBA, HTTP, JMS).
- The model shall be map-able to various protocol-specific interfaces.
  It is recommended to automate this mapping supported by tools.
- Traceability from each modeling construct back to requirements and use cases shall be provided whenever possible.

5 UML Artifact Descriptions

5.1 Object Classes

Object classes are used to convey a static[1] representation of an entity, including properties and attributes; i.e., data model, the static part of the model.

5.1.1 Object Class Notation

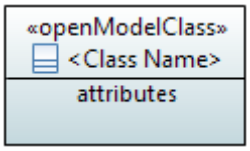


Figure 5.1: Graphical Notation for Object Classes in ONF

As highlighted in Figure 5.1, an object class is represented with a name compartment and an attributes compartment. The attributes compartment can be set in a diagram to not expose the attributes or to expose some or all of the attributes.

In some diagrams the attributes are not exposed so as to reduce clutter, in others only a subset of the attributes is exposed so as to focus attention. It is also possible to hide the attribute compartment of a class in the class diagrams where a large number of classes need to be shown, as depicted in Figure 5.2.

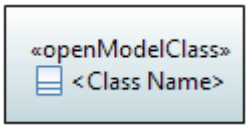


Figure 5.2: Graphical Notation for Object Classes without Attributes Compartment

[1] Not about operations acting on the entity.

 © Open Networking Foundation

The name compartment may also show stereotypes for the class where relevant. When showing stereotypes the compartment will include the stereotype «OpenModelClass» (as all classes in the ONF model have this stereotype by default) and may also include other stereotypes.

In the general UML definition a class may have name, attribute and operation compartments, as shown in Figure 5.3, but as in the ONF model the static part and the dynamic part of the model are decoupled, the operation compartment, is not used and always hidden.

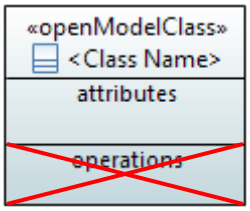


Figure 5.3: Graphical Notation for Object Classes with Attributes and Deprecated Operations Compartment

5.1.2 Object Class Properties

An object class has the following properties:

- Name
  Follows Upper Camel Case (UCC). Each class in the model has a unique name. An example of Upper Camel Case: SubNetworkConnection.
- Documentation
  Contains a short summary of the usage. The documentation is carried in the “Comments” field in Papyrus.
- Superclass(es)
  Inheritance and multiple inheritance may be used to deal with shared properties.
- Abstract
  Indicates if the object class can be instantiated or is just used for inheritance.
- Additional ONF properties are defined in the «OpenModelClass» stereotype which extents
  (Extension) by default (required) the «metaclass» Class:

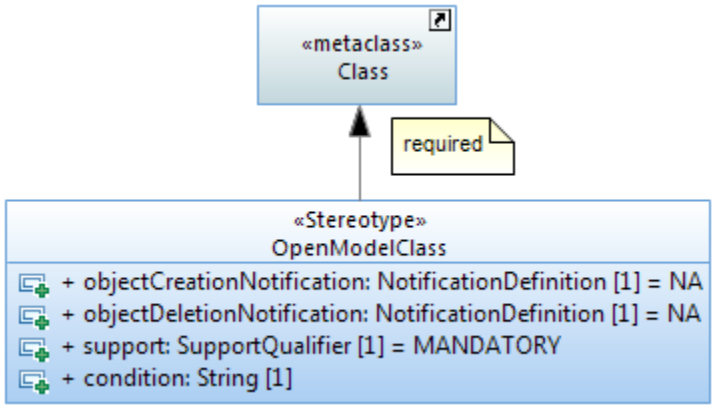


Figure 5.4: «OpenModelClass» Stereotype

 © Open Networking Foundation

- objectCreationNotification (only relevant in the purpose-specific modules of the ONF-wide Information Model; see Figure 4.1)
  Defines whether an object creation notification has to be sent when the object instance is created.
- objectDeletionNotification (only relevant in the purpose-specific modules of the ONF-wide Information Model; see Figure 4.1)
  Defines whether an object deletion notification has to be sent when the object instance is deleted.
- support
  This property qualifies the support of the object class at the management interface. See definition in section 5.9.
- condition
  This property contains the condition for the condition-related support qualifiers.
- Choice
  This stereotype identifies an object class as a choice between different alternatives.

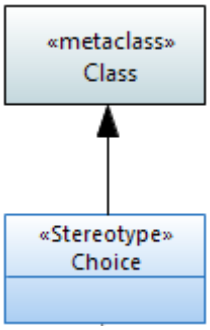


Figure 5.5: Potential Choice Annotation for Object Classes

- UML/Papyrus defined class properties that are not used in ONF:
  - Is leaf
  - Is active
  - Visibility

5.2 Attributes in Object Classes

Attributes contain the properties[2] of an object class. Note that the roles of navigable association ends become an attribute in the associated object class.

5.2.1 Attribute Notation

The notation is:

<visibility> <attribute name> : <attribute type> [<multiplicity>] = <default value>

Note: When no default is relevant or no default is defined, the "=" is not shown.

[2] In Papyrus an attribute is a property.

 © Open Networking Foundation

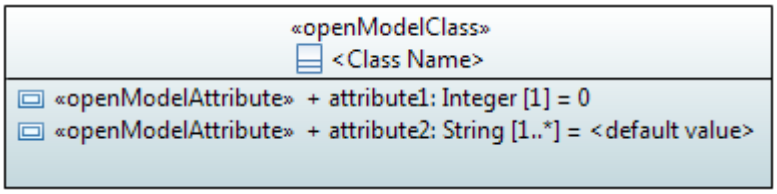


Figure 5.6: Graphical Notation for Object Classes with Attributes

5.2.2 Attribute Properties

An attribute has the following properties:

- Name
  Follows Lower Camel Case (LCC) and is unique across all attribute names in the model. An example of Lower Camel Case: subNetworkConnectionIdentifier. The plural form is "<attribute name>List".
  Boolean typed attribute names always start with a verb like ‘is’, 'must', etc. (e.g., ‘isAbstract’) and the whole attribute name must be composed in a way that it is possible to answer it by "true" or "false".
- Documentation
  Contains a short summary of the usage. The documentation is carried in the “Comments” field in Papyrus.
- Ordered
  For a multi-valued multiplicity; this specifies whether the values in an instantiation of this attribute are sequentially ordered; default is false.
- Unique
  For a multi-valued multiplicity, this specifies if the values of this attribute instance are unique (i.e., no duplicate attribute values); default is true.

  Excerpt from [3]: When Unique is true (the default), the collection of values may not contain duplicates. When Ordered is true (false being the default) the collection of values is ordered. In combination these two allow the type of a property to represent a collection in the following way:

Table 5.1: Table 11.1/[3] – Collection Types for Properties

Ordered	Unique	Collection type
false	true	Set
true	true	OrderedSet
false	false	Bag
true	false	Sequence

- Read Only
  If true, the attribute may only be read, and not changed by the client. The default value is

 © Open Networking Foundation

false. The state is dependent on the additional ONF attribute property writeable. I.e., if the attribute cannot be set/changed by the client, it is read only.

- Type
  Refers to a data type; see section 5.8.
- Default Value
  Provides the value that the attribute has to start with in case the value is not provided during creation, or already defined because of a system state.
  In cases where a default value makes no sense (e.g., if representing the state of the system) it is undefined which is shown by "--".
- Multiplicity (*, 1, 1..*, 0..1, …)
  Defines the number of values the attribute can simultaneously have.
  - * is a list attribute with 0, one or multiple values;
  - 1 attribute has always one value;
  - 1..* is a list attribute with at least one value;
  - 0..1 attribute may have no or at most one value;

  Default value is 1.
  Other values are possible; e.g., "2..17".
- Additional ONF properties are defined in the «OpenModelAttribute» stereotype which extents
  (Extension) by default (required) the «metaclass» Property:

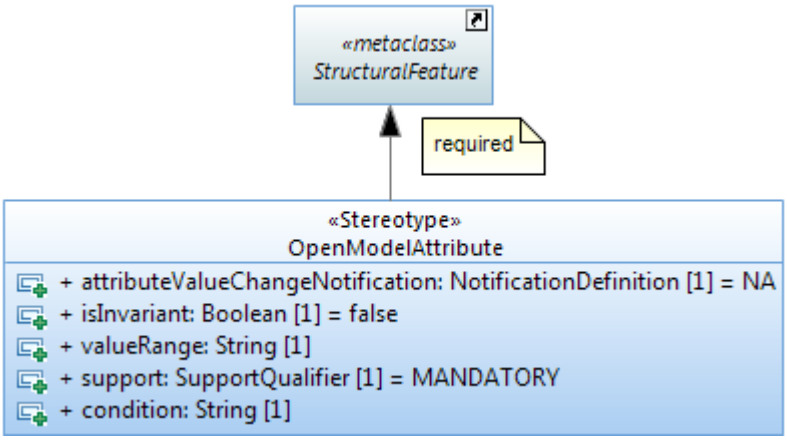


Figure 5.7: «OpenModelAttribute» Stereotype

- attributeValueChangeNotification (only relevant in the purpose-specific modules of the ONF-wide Information Model; see Figure 4.1)
  This property defines whether a notification has to be raised when the attribute changes its value or not.
- isInvariant
  Identifies if the value of the attribute can be changed after it has been created.
- valueRange
  Identifies the allowed values for the attribute.
- support
  This property qualifies the support of the attribute at the management interface. See definition in section 5.9.

 © Open Networking Foundation

- condition
  This property contains the condition for the condition-related support qualifiers.
- Other ONF properties:
  - passedByReference
    This property shall only be applied to attributes that have an object class defined as their type; i.e., on a case by case basis.
    The property defines that the attribute contains only the reference (name, identifier, address) of the referred object instance(s) when being transferred across the interface. Otherwise the attribute contains the complete information of the object instance(s) when being transferred across the interface.

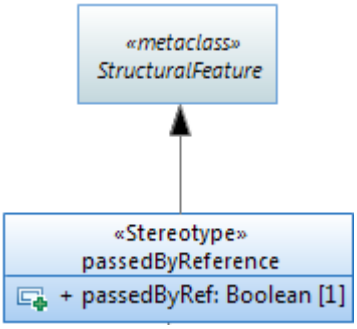


Figure 5.8: «PassedByReference» Stereotype

- UML/Papyrus defined attribute properties that are not used in ONF:
  - Is derived
  - Is derived union
  - Is leaf
  - Is static
  - Visibility

5.3 Associations

Associations are defined between object classes. Associations have association-ends. The association ends specify the role that the object at one end of a relationship performs.

5.3.1 Association Notation

The following examples show the different kinds of associations that are used in the ONF model.

Figure 5.9 shows a bi-directional navigable association where each object class has a pointer to the other. The role name becomes the name of the corresponding attribute. I.e., in the example: ClassA will have an attribute named "_classBRefList" pointing to ClassB and vice versa.

 © Open Networking Foundation

Figure 5.9: Bidirectional Association Relationship Notation

Figure 5.10 shows a unidirectional association (shown with an open arrow at the target object class) where only the source object class has a pointer to the target object class and not vice-versa.

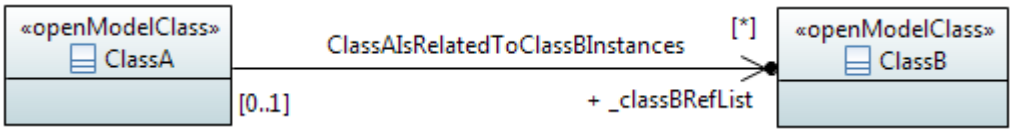


Figure 5.10: Unidirectional Association Relationship Notation

Figure 5.11 shows a uni-directional non-navigable association where each object class does not have a pointer to the other; i.e., such associations are just for illustration purposes.

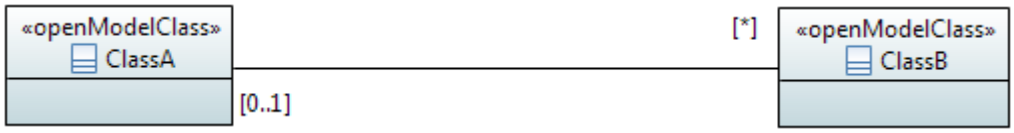


Figure 5.11: – Non-navigable Association Relationship Notation

An aggregation is a special type of association in which objects are assembled or configured together to create a more complex object. Aggregation protects the integrity of an assembly of objects by defining a single point of control called aggregate, in the object that represents the assembly.

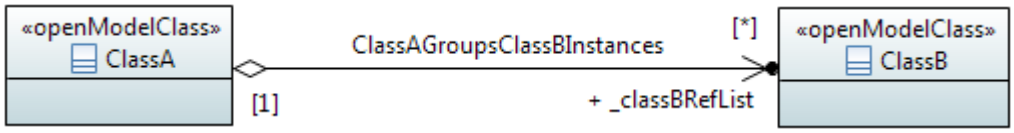
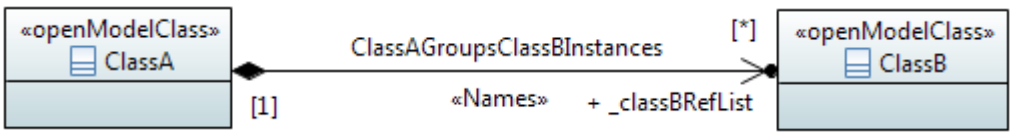


Figure 5.12: Aggregation Association Relationship Notation

A composite aggregation association is a strong form of aggregation that requires a part instance be included in at most one composite at a time. If a composite is deleted, all of its parts are deleted as well; i.e., the lifecycle of ClassB is tied to the lifecycle of ClassA.

Note: In the example below, ClassA names ClassB instances; defined by the «Names» stereotype.



 © Open Networking Foundation

Figure 5.13: Composite Aggregation Association Relationship Notation

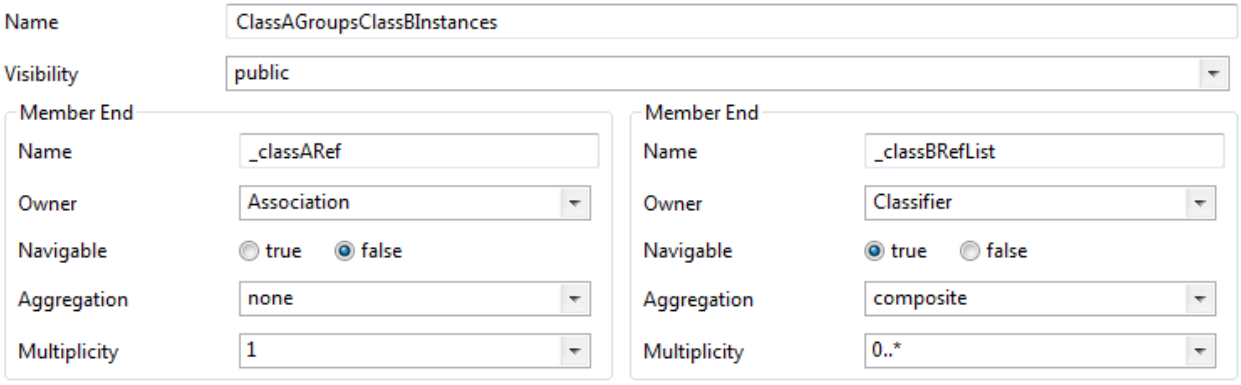


Figure 5.14: Papyrus Settings for Composite Aggregation

A generalization association indicates a relationship in which one class (the child) inherits from another class (the parent). A generalization relationship may be conditional, identified by the «Cond» stereotype.

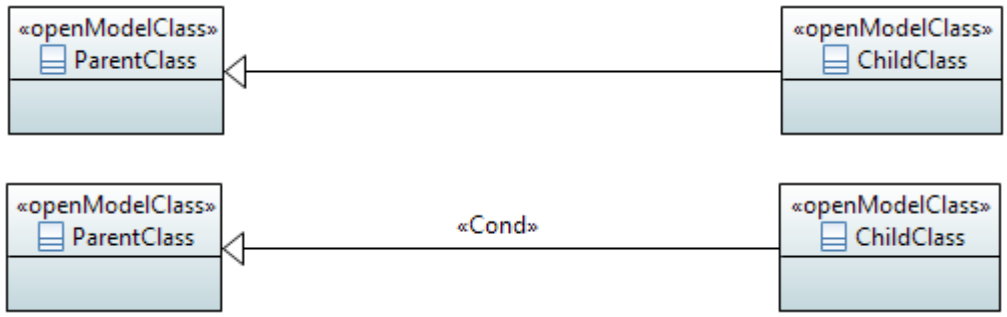


Figure 5.15: Generalization Relationship Notation (normal and conditional)

"*A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation. This means that the complete semantics of the depending elements is either semantically or structurally dependent on the definition of the supplier element(s)...*", an extract from [2].
A dependency relationship may define naming identified by the «NamedBy» stereotype.

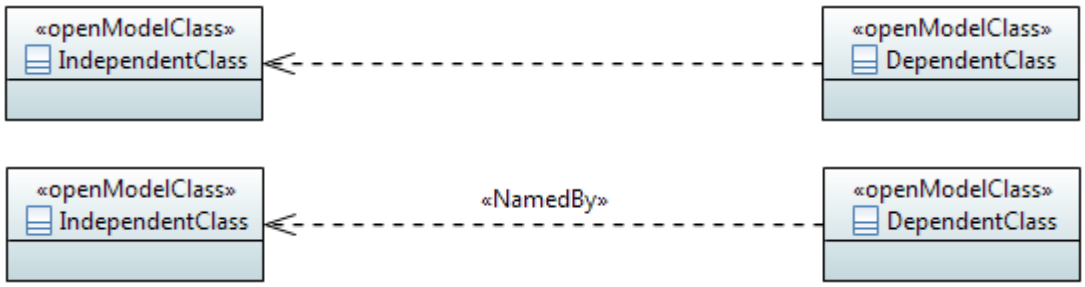


Figure 5.16: Dependency Relationship Notation (normal and naming)

 © Open Networking Foundation

5.3.2 Association Properties

An association has the following properties:

- Name
  Follows Upper Camel Case (UCC) and is unique across all association names defined in the whole model.
  The format is "<Class1Name><VerbPhrase><Class2Name>" where the verb phrase creates a sequence that is readable and meaningful.
- Documentation
  Contains a short summary of the usage. The documentation is carried in the “Comments” field in Papyrus.
- Abstract
  Associations which are just for explanation to the reader of the model are defined as "abstract". Their ends are not navigable and have no role names. Abstract associations must not be taken into account in a protocol specific implementation.
- Type
  The following types are used:
  - inheritance,
  - simple association,
  - composition,
  - aggregation.
- Role Name
  Follows Lower Camel Case (LCC) with an underscore “_” prefix and identifies the role that the object plays at this end of the association. The plural form is "_<role name>List". Only navigable association ends have role names and follow the definitions made for attributes in section 5.2.
- Role Type
  The type of the role is fixed to the object class attached to the association end. Therefore it is important to define the type as passedByReference or passedByValue. The «PassedByReference» stereotype identifies that the role (becoming an attribute) that has the stereotype associated, contains only the reference (name, identifier, address) to the referred object instance(s) when being transferred across the interface. Otherwise the role (becoming an attribute) contains the complete information of the object instance(s) when being transferred across the interface.
- Role Multiplicity
  Identifies the number of object instances that can participate in an instance of the association.
- Additional ONF properties:
  - «Names»
    The «Names» stereotype identifies that the association is used to define the naming.
  - «NamedBy»
  - The «NamedBy» stereotype identifies that a dependency relationship is used to define naming.

© Open Networking Foundation

- «Cond»
  The «Cond» stereotype identifies that the association is conditional. The condition is also provided.
- «StrictComposite»
  The «StrictComposite» stereotype can only be applied to associations with a composite end (i.e., composite aggregation association). It means that the content of the composed classes is part of the parent class and has no opportunity for independent lifecycle. The composed classes are essentially carrying attributes of the parent class where the composite is used to provide grouping of similar properties. The composed classes just provide groups of attributes for the parent class; i.e., they are abstract and cannot be instantiated.
  Whereas in an association with a composite end that is not StrictComposite the composed class is a part that has a restricted independent lifecycle. In this case an instance of the composed class can be created and deleted in the context of the parent class and should be represented as a separate instance from the parent in an implementation. This is especially true where there is a recursive composition. It is possible that in some cases the composed instance could move from one parent to another so long as it exists with one parent only at all points of the transaction. This move is not meaningful for a class associated via a StrictComposite association.

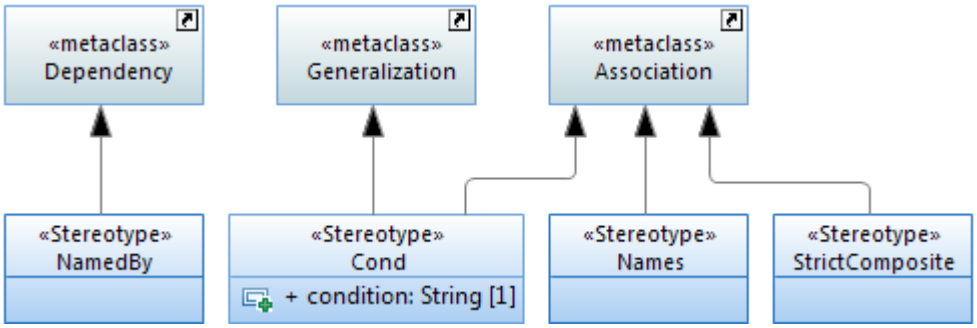


Figure 5.17: Potential Annotations for Associations

- UML/Papyrus defined attribute properties that are not used in ONF:
  - Visibility

5.4 Interfaces

An «Interface» is used to group operations, i.e., models the dynamic part of the model. Groupings of operations can be used to modularize the functionalities of the specification.

Note: Interfaces (and operations) may only be defined in the purpose-specific modules of the ONF-wide Information Model; see Figure 4.1.

5.4.1 «Interface» Notation

Interfaces are identified by the stereotype «Interface».

 © Open Networking Foundation

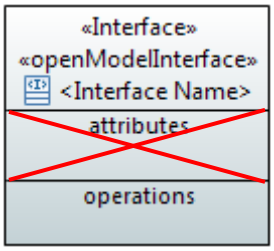


Figure 5.18: Graphical Notation for «Interface» in ONF

«Interfaces» usually have name, attributes and operations compartments. In the ONF model the static part and the dynamic part of the model are decoupled. Therefore, the attributes compartment is not used and always empty. It is also possible to hide the attributes compartment in the interface diagrams.

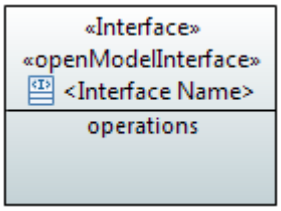


Figure 5.19: Graphical Notation for «Interface» without Attributes Compartment

Note: The graphical notation of an «Interface» may show an empty operation compartment so as to reduce clutter even if the «Interface» has operations.

5.4.2 «Interface» Properties

An «Interface» has the following properties:

- Name
  Follows Upper Camel Case (UCC) and is unique across all «Interface» names in the model.
- Documentation
  Contains a short summary of the usage. The documentation is carried in the “Comments” field in Papyrus.
- Superinterface(s)
  Inheritance and multiple inheritance may be used.
- Abstract
  Indicates if the «Interface» can be instantiated or is just used for inheritance.
- Additional ONF properties are defined in the «OpenModelInterface» stereotype which extents
  (Extension) by default (required) the «metaclass» Interface:

 © Open Networking Foundation

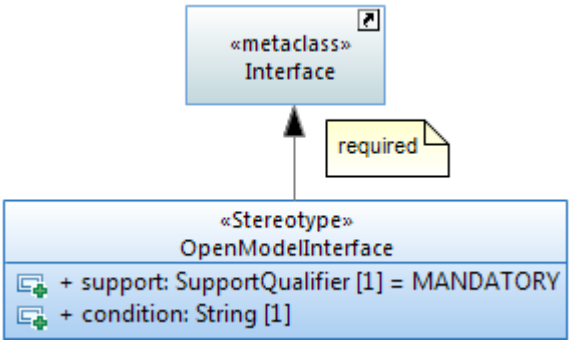


Figure 5.20: «OpenModelInterface» Stereotype

- support
  This property qualifies the support of the «Interface» at the management interface. See definition in section 5.9.
- condition
  This property contains the condition for the condition-related support qualifiers.

- UML/Papyrus defined interface properties that are not used in ONF:
  - Is leaf
  - Visibility

5.5 Interface Operations

Operations can be defined within an «Interface». An «Interface» must have at least one operation.

Note: Operations may only be defined in the purpose-specific modules of the ONF-wide Information Model; see Figure 4.1.

5.5.1 Operation Notation

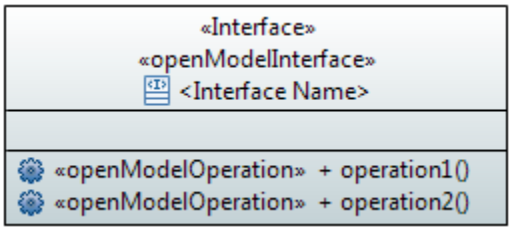


Figure 5.21: Graphical Notation for «Interface» with Operations

5.5.2 Operation Properties

An operation has the following properties:

- Name
  Follows Lower Camel Case (LCC) and is unique across all operation names defined in the whole model.

 © Open Networking Foundation

- Documentation
  Contains a short summary of the usage. The documentation is carried in the “Comments” field in Papyrus.
- Pre-condition(s)
  This property defines the conditions that have to be true before the operation can be started (i.e., if not true, the operation will not be started at all and a general “precondition not met” error will be returned, i.e., exception is raised).
- Post-condition(s)
  This property defines the state of the system after the operation has been executed (if successful, or if not successful, or if partially successful).
  Note that partially successful post-condition(s) can only be defined in case of non-atomic operations.
  Note that when an exception is raised, it should not be assumed that the post-condition(s) are satisfied.
- Parameter(s)
  See section 5.6.
- Operation Exceptions
  List the allowed exceptions for the operation.
  The model uses predefined exceptions which are split in 2 types:
  - generic exceptions which are associated to all operations by default
  - common exceptions which needs to be explicitly associated to the operation.

  Note: These exceptions are only relevant for a protocol neutral information model. Further exceptions may be necessary for a protocol specific information model.

  Generic exceptions:

  - Internal Error: The server has an internal error.
  - Unable to Comply: The server cannot perform the operation. Use Cases may identify specific conditions that will result in this exception.
  - Comm Loss: The server is unable to communicate with an underlying system or resource, and such communication is required to complete the operation.
  - Invalid Input: The operation contains an input parameter that is syntactically incorrect or identifies an object of the wrong type or is out of range (as defined in the model or because of server limitation).
  - Not Implemented: The entire operation is not supported by the server or the operation with the specified input parameters is not supported.
  - Access Denied: The client does not have access rights to request the given operation.

  Common exceptions:

  - Entity Not Found: Is thrown to indicate that at least one of the specified entities does not exist.
  - Object In Use: The object identified in the operation is currently in use.
  - Capacity Exceeded: The operation will result in resources being created or activated beyond the capacity supported by the server.

 © Open Networking Foundation

    - Not In Valid State: The state of the specified object is such that the server cannot perform the operation. In other words, the environment or the application is not in an appropriate state for the requested operation.
    - Duplicate: Is thrown if an entity cannot be created because an object with the same identifier/name already exists.
- Additional ONF properties are defined in the «OpenModelOperation» stereotype which extents
  (Extension) by default (required) the «metaclass» Operation:

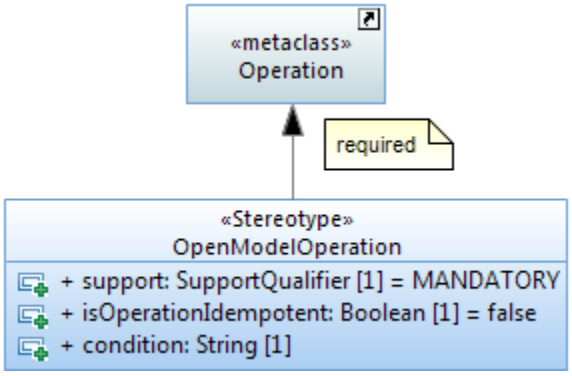


Figure 5.22: «OpenModelOperation» Stereotype

    - isOperationIdempotent (Boolean)
      Defines if the operation is idempotent (true) or not (false).
    - support
      This property qualifies the support of the operation at the management interface. See definition in section 5.9.
    - condition
      This property contains the condition for the condition-related support qualifiers.
- UML/Papyrus defined operation properties that are not used in ONF:
    - Is leaf
    - Is query
    - Is static

5.6 Operation Parameters

Parameters define the input and output signals of an operation.

Note: Operations and their parameters may only be defined in the purpose-specific modules of the ONF-wide Information Model; see Figure 4.1.

5.6.1 Parameter Notation

The notation is:

<visibility> <direction> <parameter name> : <parameter type> [<multiplicity>] = <default value>

 © Open Networking Foundation

Note: When no default is relevant or no default is defined, the "=" is not shown

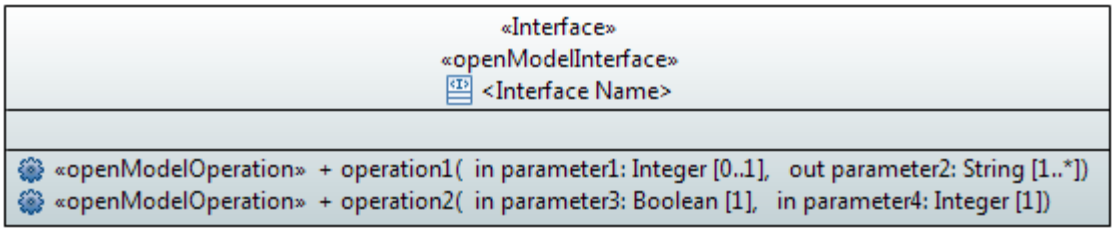


Figure 5.23: Graphical Notation for «Interface» with Operations and Parameters

5.6.2 Parameter Properties

A parameter has the following properties:

- Name
  Follows Lower Camel Case (LCC)
- Documentation
  Contains a short summary of the usage. The documentation is carried in the "Comments" field in Papyrus.
- Direction
  Parameters can be defined as:
  - input parameters
  - output parameters
  - in out parameters
- Type
  Refers to a data type.
  Note that a list of parameters can also be combined in a complex data type.
- Default Value
  Defines the value that the parameter has in case the value is not provided. If it is mandatory to provide a value, the default value is set to NA.
- Is Ordered
  Defines for a multi-valued parameter that the order of the values is significant.
- Multiplicity
  Defines the number of values the parameter can simultaneously have.
- Additional ONF properties are defined in the «OpenModelParameter» stereotype which extents
  (Extension) by default ({required}) the «metaclass» Parameter:

 © Open Networking Foundation

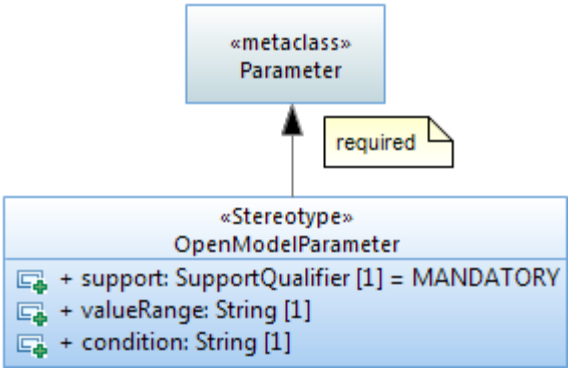


Figure 5.24: «OpenModelParameter» Stereotype

- valueRange
  Identifies the allowed values for the parameter.
- support
  This property qualifies the support of the parameter at the management interface. See definition in section 5.9.
- condition
  This property contains the condition for the condition-related support qualifiers.

- Other ONF properties:
  - passedByReference
    This property shall only be applied to parameters that have an object class defined as their type; i.e., on a case by case basis.
    The property defines if the attribute contains only the reference (name, identifier, address) to the referred object instance(s) when being transferred across the interface. Otherwise the parameter contains the complete information of the object instance(s) when being transferred across the interface.

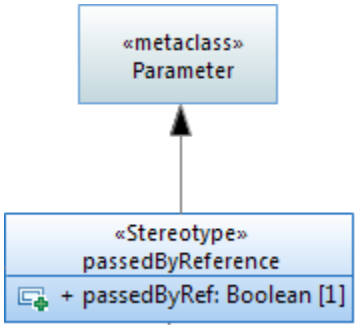


Figure 5.25: «PassedByReference» Stereotype

- UML/Papyrus defined parameter properties that are not used in ONF:
  - Is exception
  - Is stream
  - Is unique
  - Visibility

 © Open Networking Foundation

5.7 Notifications

Note: Notifications may only be defined in the purpose-specific modules of the ONF-wide Information Model; see Figure 4.1.

The UML «Signal» artifact is used to define the content of a notification. The information is defined in the attributes of the «Signal».

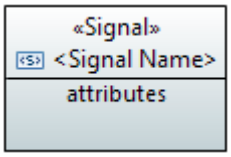


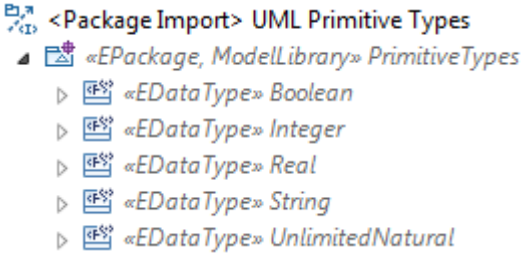
Figure 5.26: Graphical Notation for «Signal»

5.8 Types

Types are used as type definitions of attributes and parameters.

Data Types are divided into 3 categories:
- dataType
- enumeration
- primitiveType

Papyrus already provides the following UML primitive types:



5.8.1 Type Notation

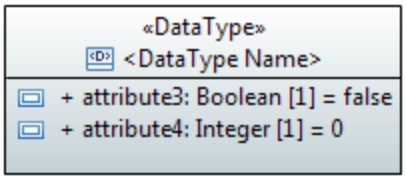


Figure 5.27: Graphical Notation for «DataType»

© Open Networking Foundation

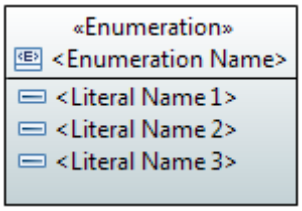


Figure 5.28: Graphical Notation for «Enumeration»

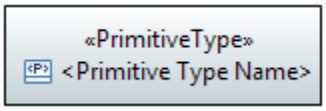


Figure 5.29: Graphical Notation for «PrimitiveType»

5.8.2 Type Properties

A type has the following properties:

- Category
  Three categories are used in the model:
  - dataType
  - enumeration
  - primitive
- Name
  Follows Upper Camel Case (UCC) and is unique across all data type names defined in the whole model.
- Documentation
  Contains a short summary of the usage. The documentation is carried in the “Comments” field in Papyrus.
- Data type attributes (only in dataTypes)
  Follow the definitions made for attributes in section 5.2 with the following exceptions:
  - the isInvariant property can be ignored and is fixed to "true"
  - the notification property can be ignored and is fixed to "NA".
- Enumeration literals (only in enumerations)
  The name contains only upper case characters where the words are separated by "_".
- Additional ONF properties
  - Choice
    This stereotype identifies a data type as a choice between different alternatives.
  - Exception
    This stereotype defines a data type used for an operation exception.

 © Open Networking Foundation

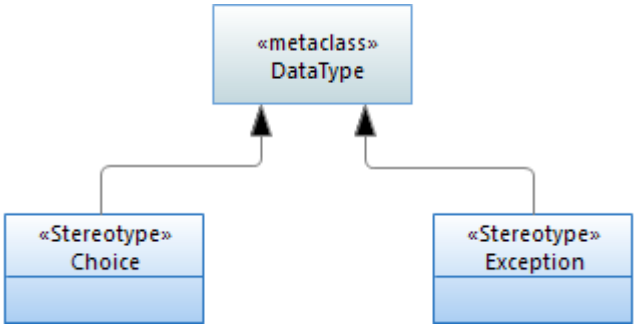


Figure 5.30: Potential Annotations for Data Types

- UML/Papyrus defined attribute properties that are not used in ONF:
  - Is abstract
  - Is leaf

5.9 Qualifiers

This clause defines the qualifiers applicable for model elements specified in this document, e.g., the «OpenModelClass» (see section 5.1.2), and the «OpenModelAttribute» (see section 5.2.2). The qualifications are M, O, CM, CO, C and 'SS'. Their meanings are specified in this section. This type of qualifier is called Support Qualifier.

- Definition of M (Mandatory) qualification:
  The capability shall be supported.
- Definition of O (Optional) qualification:
  The capability may or may not be supported.
- Definition of CM (Conditional-Mandatory) qualification:
  The capability shall be supported under certain conditions, specifically:
  When qualified as CM, the capability shall have a corresponding constraint defined in the specification. If the specified constraint is met then the capability shall be supported.
- Definition of CO (Conditional-Optional) qualification:
  The capability may be supported under certain conditions, specifically:
  When qualified as CO, the capability shall have a corresponding constraint defined in the specification. If the specified constraint is met then the capability may be supported.
- Definition of C (Conditional) qualification:
  Used for items that have multiple constraints. Each constraint is worded as a condition for one kind of support, such as mandatory support, optional support or "no support". All constraints must be related to the same kind of support. Specifically:
  Each item with C qualification shall have the corresponding multiple constraints defined in the specification. If all specified constraints are met and are related to mandatory, then the item shall be supported. If all the specified constraints are met and are related to optional, then the item may be supported. If all the specified constraints are met and are related to "no support", then the item shall not be supported.

  Note: This qualifier should only be used when absolutely necessary, as it is more complex to implement.

 © Open Networking Foundation

- Definition of SS (SS Conditional) qualification:
  The capability shall be supported by at least one but not all solutions.
- Definition of '-' (no support) qualification:
  The capability shall not be supported.

6 UML Profile Definitions

6.1 Additional ONF Properties Definitions

Section 5 has already described the additional ONF properties for each UML artifact. All defined stereotypes are shown as an overview in Figure 6.1 and Table 6.1 below.

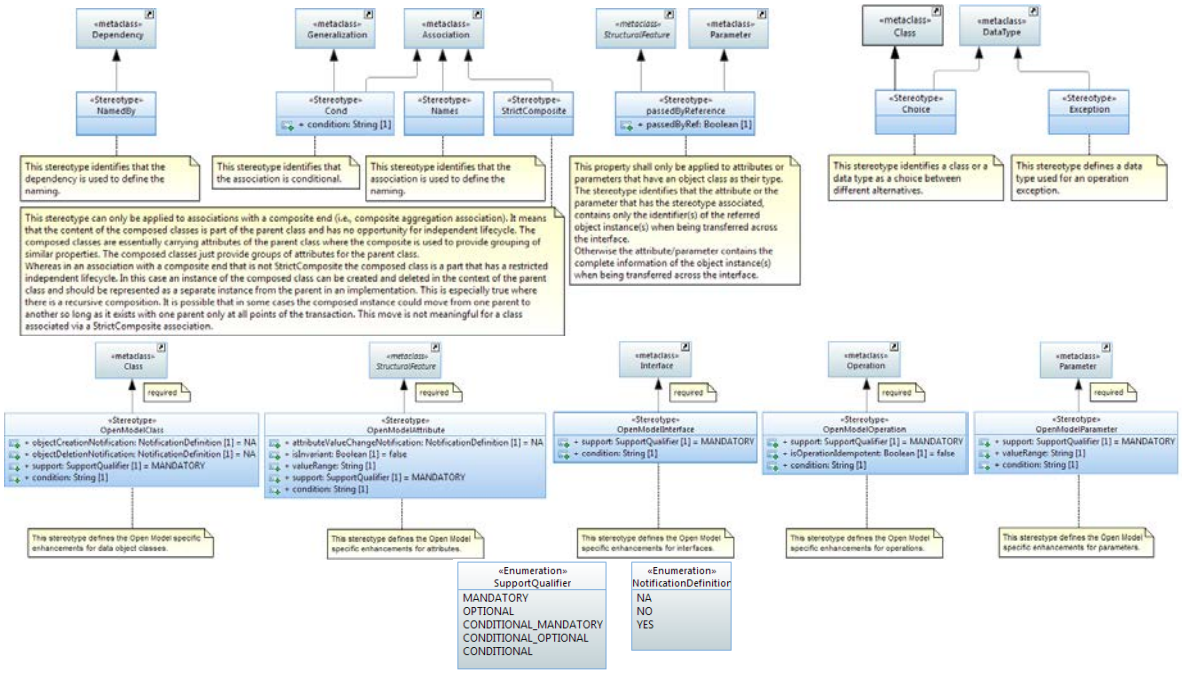


Figure 6.1: UML Artifact «Stereotypes»

 © Open Networking Foundation

Table 6.1: UML Artifact Properties Defined in Complex «Stereotypes»

Stereotype	Name of property	Type	Allowed values	Default value	Associated to metaclass
OpenModelClass	objectCreation Notification	enumeration	NO, YES, NA	NA	Class
	objectDeletion Notification	enumeration	NO, YES, NA	NA	
	support	enumeration	MANDATORY OPTIONAL CONDITIONAL_ MANDATORY CONDITIONAL_ OPTIONAL CONDITIONAL	MANDATORY	
	condition	string			
OpenModelAttribute	attributeValue Change Notification	enumeration	NO, YES, NA	NA	Property
	isInvariant	Boolean	true/false	false	
	valueRange	string		NA	
	support	enumeration	MANDATORY OPTIONAL CONDITIONAL_ MANDATORY CONDITIONAL_ OPTIONAL CONDITIONAL	MANDATORY	
	condition	string			
OpenModelInterface	support	enumeration	MANDATORY OPTIONAL CONDITIONAL_ MANDATORY CONDITIONAL_ OPTIONAL CONDITIONAL	MANDATORY	Interface
	condition	string			
OpenModelOperation	isOperationIdempotent	Boolean	true/false	false	Operation
	support	enumeration	MANDATORY OPTIONAL CONDITIONAL_ MANDATORY CONDITIONAL_ OPTIONAL CONDITIONAL	MANDATORY	
	condition	string			

© Open Networking Foundation

Stereotype	Name of property	Type	Allowed values	Default value	Associated to metaclass
OpenModelParameter	valueRange	string		NA	Parameter
	support	enumeration	MANDATORY OPTIONAL CONDITIONAL_MANDATORY CONDITIONAL_OPTIONAL CONDITIONAL	MANDATORY	
	condition	string			

6.2 Modeling Lifecycle Definitions

The UML artifacts (packages, classes, attributes, interfaces, operations, parameters, data types, associations and generalizations) can be appended with the following modeling lifecycle states:

- Example
  This state indicates that the entity is NOT to be used in implementation and is in the model simply to assist in the understanding of the model (e.g., a specialization of a generalized class where the generalized class is to be used as is and the specialization is simply offered to more easily illustrate an application of the generalized class).
- Experimental
  This state indicates that the entity is at a very early stage of development and will almost certainly change. The entity is NOT mature enough to be used in implementation.
- Faulty
  This state indicates that the entity should not be used in new implementation and that attempts should be made to remove it from existing implementation as there is a problem with the entity. An update to the model with corrections will be released.
- LikelyToChange
  This state indicates that although the entity may be mature, work in the area has indicated that change will be necessary (e.g., there are new insights in the area or there is now perceived benefit to be had from further rationalization). The entity can still be used in implementation but with caution.
- Obsolete
  This state indicates that the entity should not be used in new implementation and that attempts should be made to remove it from existing implementation.
- Preliminary
  This state indicates that the entity is at a relatively early stage of development and is likely to change, but is mature enough to be used in implementation.

© Open Networking Foundation

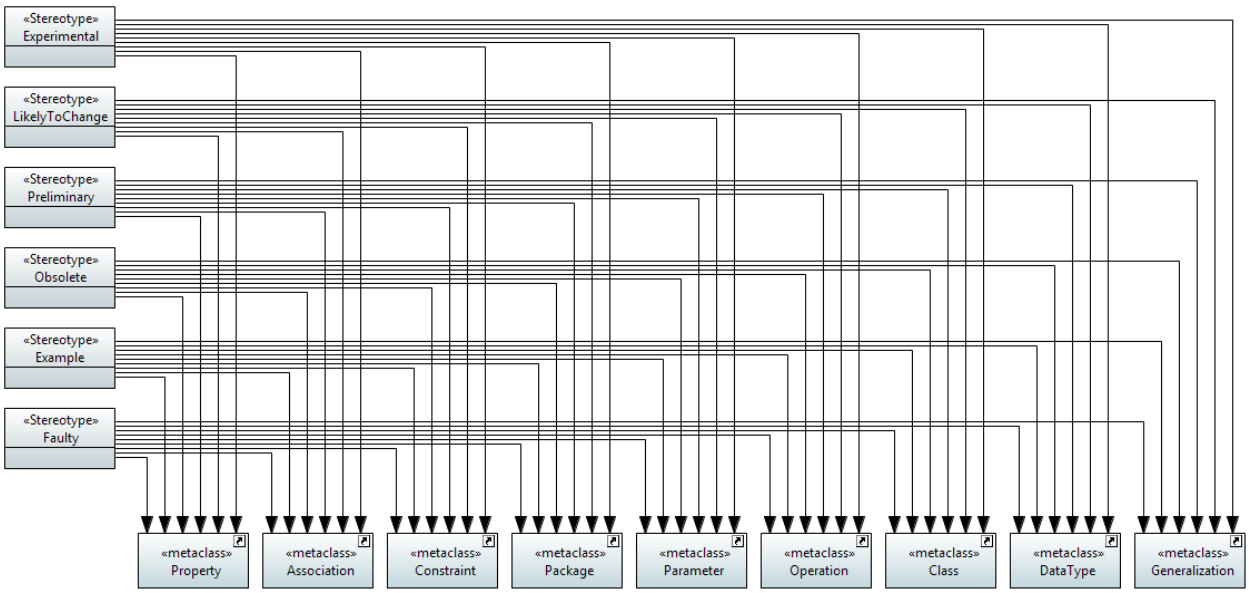


Figure 6.2: Lifecycle «Stereotypes»

6.3 ONF Open Model Profile Files

OpenModelProfile_Kepler_v0.0.1.zip

OpenModelProfile_Luna_v0.0.1.zip

Date: Feb. 20th 2015

7 Recommended Modeling Patterns

7.1 File Naming Conventions

tba

7.2 Model Structure

7.2.1 Generic Model Structure

Figure 7.1 shows a generic Information Model containing a core model and various sub-models A, B, C structured by packages:

© Open Networking Foundation

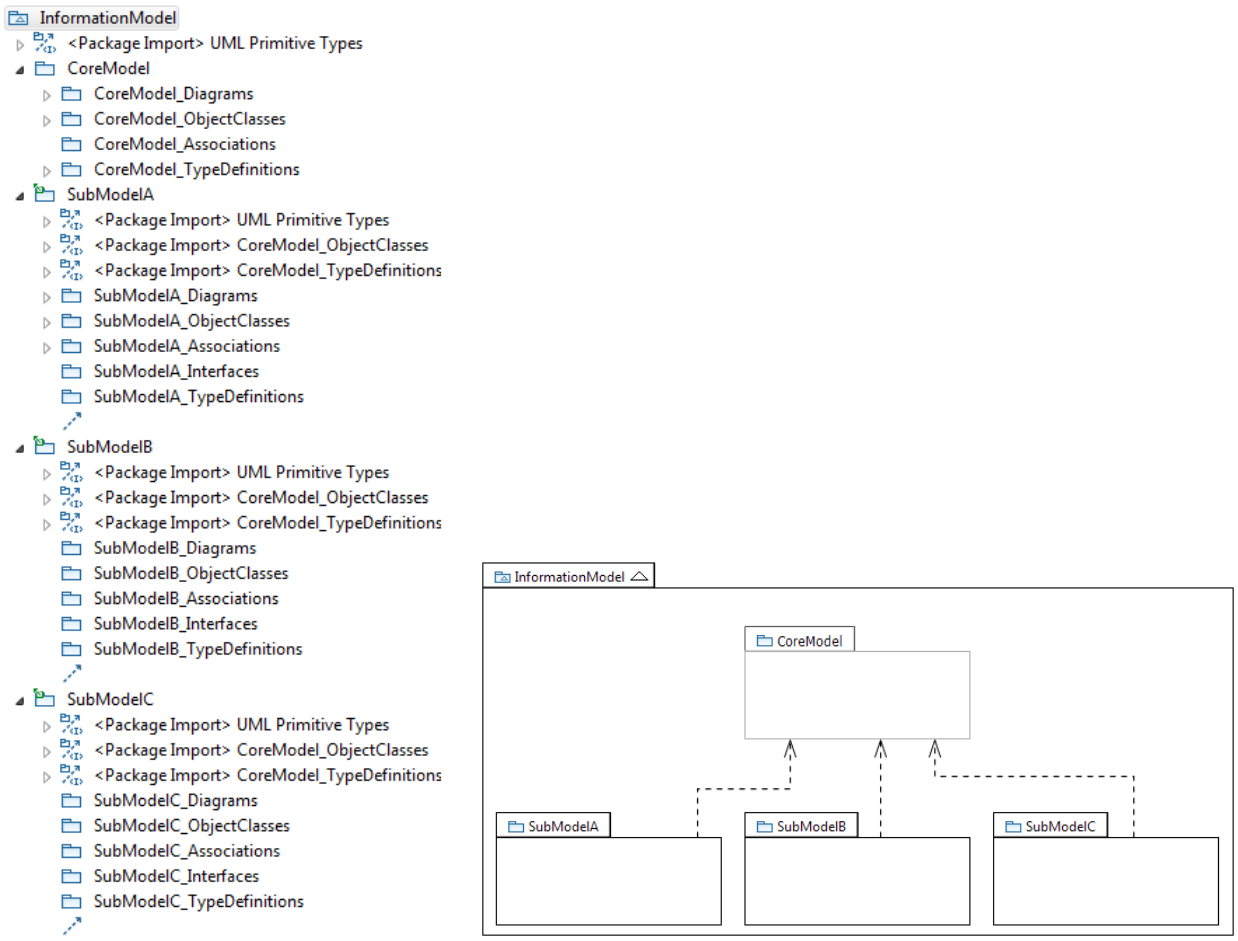


Figure 7.1: Core Model and Sub-Models

Note:
Figure 7.1 shows only the schematic structure of the core and sub-models as necessary for these guidelines.

Each Model can be optionally organized into multiple modules. Each Model or each of its constituent modules is further divided into packages containing associations, diagrams, object classes, rules and type definitions. Sub-models may contain in addition packages for (UML-) interfaces (and their operations) and notifications.

7.2.2 ONF Specific Model Structure

The ONF Information Model is structured into a Common Information Model and additional Specific Views which are grouped according to the working groups' structure. The ONF-wide Information Model will be defined in different WGs; see Figure 4.1. The IMP Modeling team defines the generic functions and the individual WGs will add their specific enhancements to the model.

Each working group view can be further divided into individual UML modules.
Figure 7.2 presents the current structure of the model.

 © Open Networking Foundation

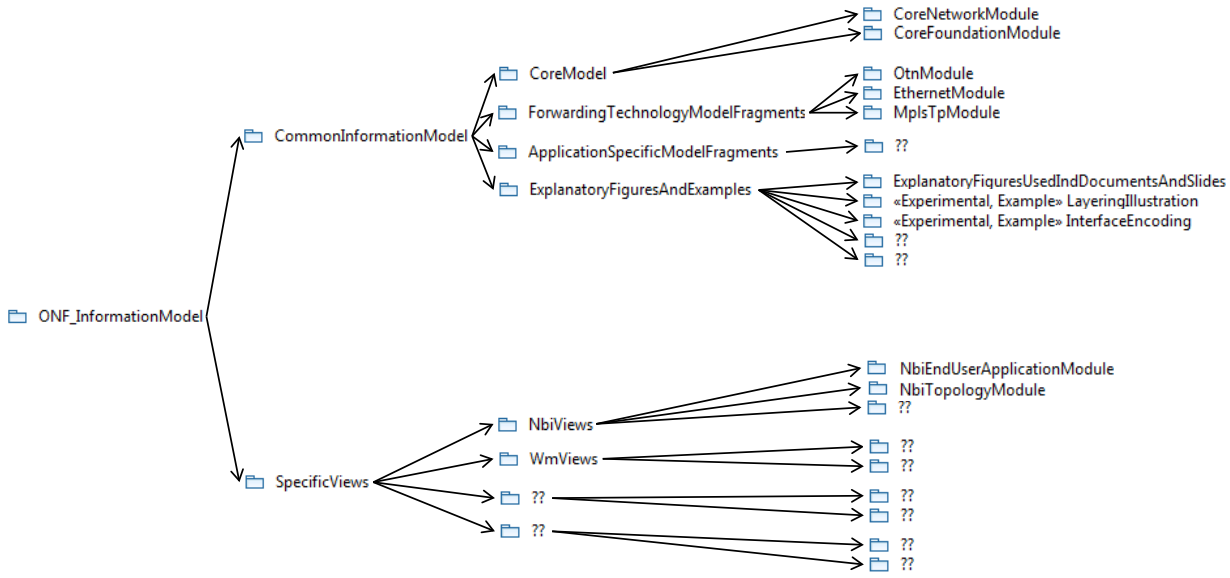


Figure 7.2: ONF Model Structure (snapshot)

Each module is divided into a set of pre-defined packages. Not all packages need to be established. Figure 7.3 shows the pre-defined packages.

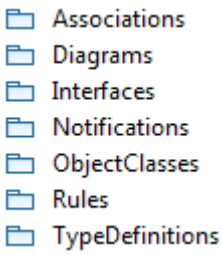


Figure 7.3: Pre-defined Packages in a UML Module

Additional packages can be added when needed.

7.3 Flexible Attribute Assignment to Object Classes

Since it is not possible to add attributes once an object instance has been created, it is necessary to differentiate case (a) where attributes are assembled before the object instance is created, and case (b) where further attributes (functions) are added after the object instance is created.

For case (a), attributes are grouped in object classes called “Pacs” and are associated to the base object class using a conditional composition association (see section 7.3.1 below).

An example for (a) is a specific LTP instance which has specific Pacs associated, based on the functions that this LTP supports. Once the LTP is created, it is no longer possible to add further attributes or remove attributes.

 © Open Networking Foundation

→ Object instances are (automatically) created as an assembly of the base object plus a list of Pacs (depending on the supported functionality).

For case (b), attributes are grouped in “normal” object classes and are associated to the base object class using a composition association.

An example for (b) is a specific, already existing LTP instance which will be configured to do performance monitoring (PM). In this case an additional PM object instance (created on the basis of the corresponding object class (i.e., not Pac)) is separately instantiated and associated to the already existing LTP. Note that it is also possible to remove the PM object instance from the LTP afterwards without impacting the life cycle of the base LTP instance.

→ Object instances are created on an explicit request and associated to already existing object instances (depending on the requested additional functionality).

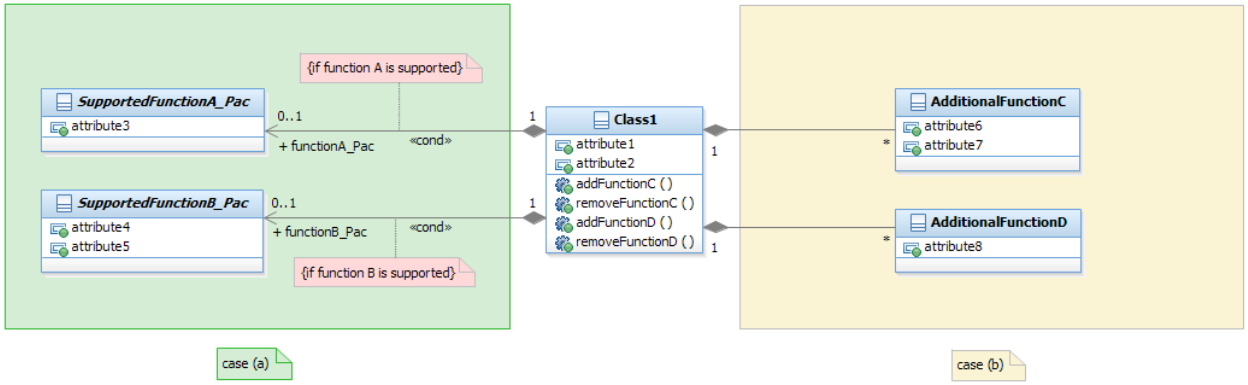
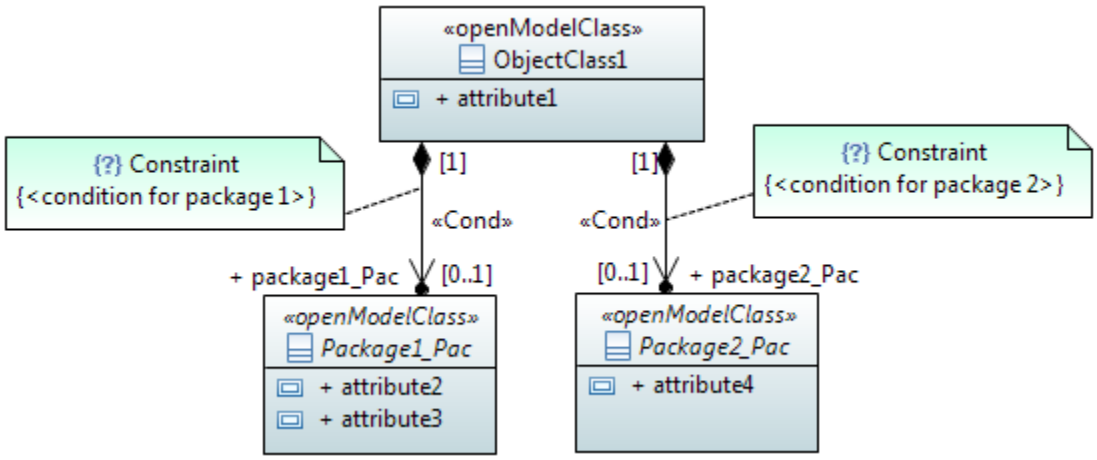


Figure 7.4: Flexible Attribute Assignment to Object Classes

7.4 Use of Conditional Packages

Conditional packages are used to enhance (core) object classes / interfaces with additional attributes / operations on a conditional basis. The attributes / operations are defined in special object classes called packages.



 © Open Networking Foundation

Figure 7.5: Enhancing Object Classes Using Conditional Packages

Package names follow the same rules as defined for object classes; i.e., UCC. The name ends with the suffix "_Pac".
The role name of the navigable end pointing to the package follows the same rules as defined for attributes; i.e., LCC. The name ends with the suffix "_Pac".

7.5 Use of XOR/Choice

7.5.1 Xor Constraint

7.5.1.1 Description

"A Constraint represents additional semantic information attached to the constrained elements. A constraint is an assertion that indicates a restriction that must be satisfied by a correct design of the system. The constrained elements are those elements required to evaluate the constraint specification…", an extract from 9.6.1 Constraint of [3].

For a constraint that applies to two elements such as two associations, the constraint shall be shown as a dashed line between the elements labeled by the constraint string (in braces). The constraint string, in this case, is xor.

7.5.1.2 Example

The figure below shows a ServerObjectClass instance that has relation(s) to multiple instances of a class from the choice of ClientObjectCLass_Alternative1, ClientObjectClass_Alternative2 or ClinetObjectCLass_Alternative3.

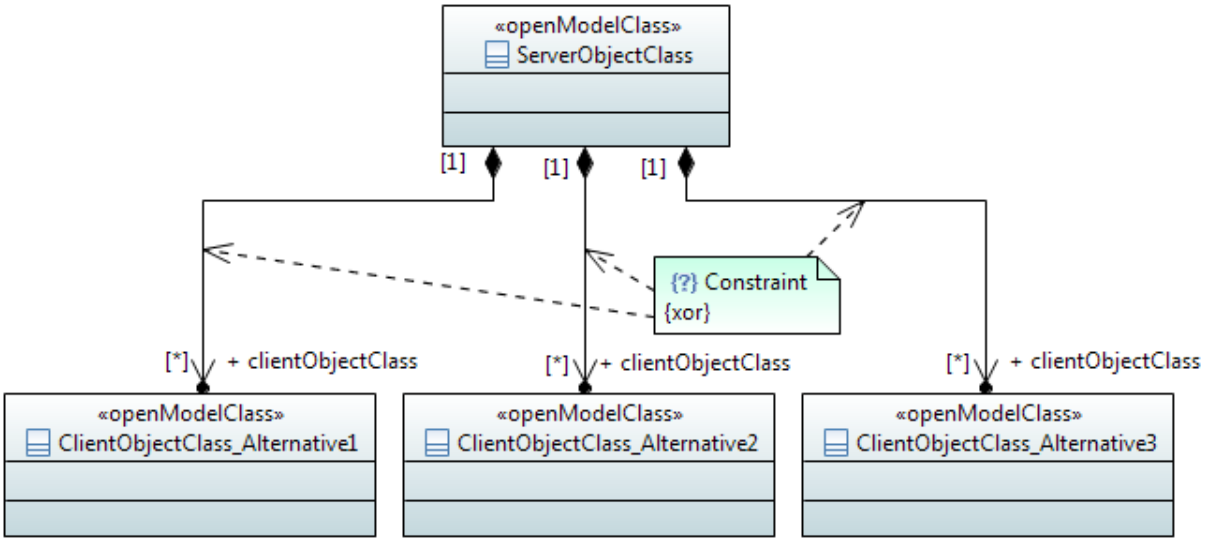


Figure 7.6: {xor} Notation

7.5.1.3 Name style

It has no name so there is no name style.

© Open Networking Foundation

7.5.2 «Choice»

7.5.2.1 Description

The «Choice» stereotype represents one of a set of classes (when used as an information model element) or one of a set of data types (when used as an operations model element).

This stereotype property, e.g., one out of a set of possible alternatives, is identical to the {xor} constraint (see 7.4.1).

7.5.2.2 Example

Sometimes the specific kind of class cannot be determined at model specification time. In order to support such scenario, the specification is done by listing all possible classes.

The following diagram lists 3 possible classes. It also shows a «Choice, InformationObjectClass» named SubstituteObjectClass. This scenario indicates that only one of the three «InformationObjectClass» named Alternative1ObjectClass, Alternative2ObjectClass, Alternative3ObjectClass shall be realized.

The «Choice» stereotype represents one of a set of classes when used as an information model element.

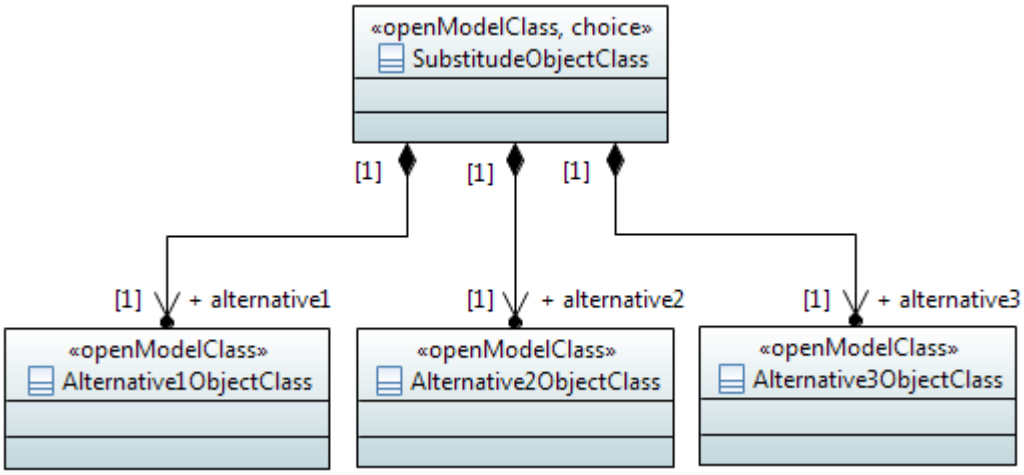


Figure 7.7: Information Model Element Example Using «Choice» Notation

Sometimes the specific kind of data type cannot be determined at model specification time. In order to support such scenario, the specification is done by listing all possible data types.

The following diagram lists 2 possible data types. It also shows a «Choice» named ProbableCause. This scenario indicates that only one of the two «DataType» named IntegerProbableCause, StringProbableCause shall be realized.

The «Choice» stereotype represents one of a set of data types when used as an operations model element.

 © Open Networking Foundation

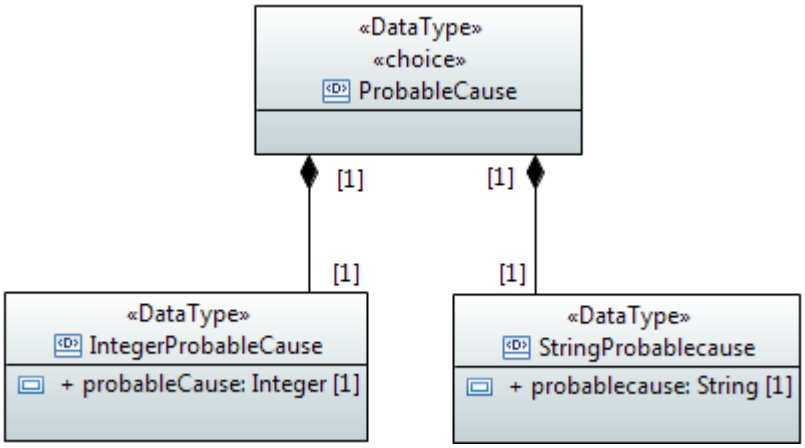


Figure 7.8: Operations Model Element Example Using «Choice» Notation

Sometimes models distinguish between sink/source/bidirectional termination points. A generic class which comprises these three specific classes can be modeled using the «Choice» stereotype.

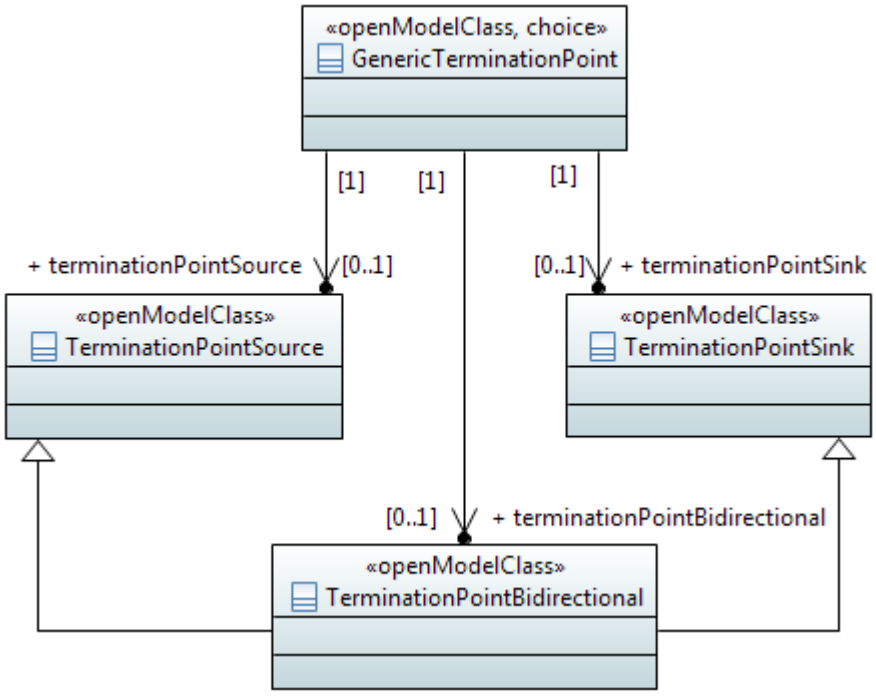


Figure 7.9: Sink/Source/Bidirectional Termination Points Example Using «Choice» Notation

7.5.2.3 Name style

For «Choice» name, use the same style as «OpenModelClass» (see 5.1.2).

7.6 Diagram Guidelines

 © Open Networking Foundation

Classes and their relationships shall be presented in class diagrams.

Interfaces and their operations shall be presented in class diagrams.

It is recommended to create:

- An overview class diagram containing all classes related to a specific management area:
  - The class name compartment should contain the location of the class definition (e.g. "Qualified Name").
  The class attributes should show the "Signature" (see section 7.3.45 of [2] for the signature definition).
- A separate inheritance class diagram in case the overview diagram would be overloaded when showing the inheritance structure (Inheritance Class Diagram).
- A class diagram containing the user defined data types (Type Definitions Diagram).
- Additional class diagrams to show specific parts of the specification in detail.
- State diagrams for complex state attributes.
- State transition diagrams for attributes with defined value transitions.
- Activity diagrams for operations with high complexity.

 © Open Networking Foundation