

OPEN NETWORKING
FOUNDATION

UML Modeling Guidelines

Version 1.1
November 30, 2015

ONF TR-514



ONF Document Type: Technical Recommendation
ONF Document Name: UML Modeling Guidelines V1.1

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	16
5.4	Interfaces	19
5.4.1	«Interface» Notation	19
5.4.2	«Interface» Properties	19
5.5	Interface Operations	20
5.5.1	Operation Notation	21
5.5.2	Operation Properties	21
5.6	Operation Parameters	23
5.6.1	Parameter Notation	23
5.6.2	Parameter Properties	23
5.7	Notifications	25
5.7.1	Notification Notation	25
5.7.2	Notification Properties	25
5.8	Types	26
5.8.1	Type Notation	27
5.8.2	Type Properties	27
5.9	Qualifiers	28
6	UML Profile Definitions	29
6.1	Additional Properties Definitions	29
6.2	Modeling Lifecycle Definitions	32

7	Recommended Modeling Patterns	33
7.1	File Naming Conventions.....	33
7.2	Model Structure.....	34
7.2.1	Generic Model Structure.....	34
7.2.2	Model Structure	34
7.3	Flexible Attribute Assignment to Object Classes	35
7.4	Use of Conditional Packages.....	36
7.5	Use of XOR/Choice.....	37
7.5.1	Xor Constraint.....	37
7.5.2	«Choice».....	38
7.6	Diagram Guidelines	39

List of Figures

Figure 4.1:	Specification Architecture	8
Figure 5.1:	Graphical Notation for Object Classes.....	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.....	14
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.....	15
Figure 5.14:	Generalization Relationship Notation (normal and conditional).....	16
Figure 5.15:	Dependency Relationship Notation (normal and naming)	16
Figure 5.16:	Realization Relationship Notation.....	16
Figure 5.17:	Owner of a navigable Member End	17
Figure 5.18:	Potential Annotations for Associations	18
Figure 5.19:	Graphical Notation for «Interface».....	19
Figure 5.20:	Graphical Notation for «Interface» without Attributes Compartment	19
Figure 5.21:	«OpenModelInterface» Stereotype.....	20
Figure 5.22:	Graphical Notation for «Interface» with Operations.....	21
Figure 5.23:	«OpenModelOperation» Stereotype	22
Figure 5.24:	Graphical Notation for «Interface» with Operations and Parameters	23
Figure 5.25:	«OpenModelProperty» Stereotype	24
Figure 5.26:	«PassedByReference» Stereotype.....	25
Figure 5.27:	Graphical Notation for «Signal».....	25
Figure 5.28:	«OpenModelNotification» Stereotype	26
Figure 5.29:	Graphical Notation for «DataType»	27
Figure 5.30:	Graphical Notation for «Enumeration»	27
Figure 5.31:	Graphical Notation for «PrimitiveType»	27
Figure 5.32:	Potential Annotations for Data Types	28
Figure 6.1:	UML Artifact «Stereotypes».....	30
Figure 6.2:	Lifecycle «Stereotypes».....	33
Figure 7.1:	Core Model and Sub-Models	34
Figure 7.2:	Model Structure (snapshot)	35

Figure 7.3: Pre-defined Packages in a UML Module	35
Figure 7.4: Flexible Attribute Assignment to Object Classes	36
Figure 7.5: Enhancing Object Classes Using Conditional Packages.....	37
Figure 7.6: {xor} Notation	37
Figure 7.7: Information Model Element Example Using «Choice» Notation	38
Figure 7.8: Operations Model Element Example Using «Choice» Notation	39
Figure 7.9: Sink/Source/Bidirectional Termination Points Example Using «Choice» Notation	39

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»	30

Document History

Version	Date	Description of Change
1.0	March 13, 2015	Initial version
1.1	Nov. 30, 2015	Version 1.1

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 not specific to any SDO, technology or management protocol.

UML defines a number of basic model elements (UML artifacts). In order to assure consistent and harmonized information models, 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].

Summary of main changes between version 1.0 and 1.1

The following guidelines have been added:

- isAtomic property on operations
- «OpenModelNotification» stereotype
- realization association along with the «PruneAndRefactor» stereotype
- «Deprecated» lifecycle stereotype.

The requirement to use “Ref” and “List” in attribute/parameter/role names has been deprecated since the “Ref” property is already defined by the «PassedByReference» property and the “List” property is already defined by the multiplicity property.

The Guidelines are no longer ONF dependent; i.e, they can now be used as is by other SDOs.

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
JMS	Java Message Service
JSON	JavaScript Object Notation
JWG	Joint Working Group (TM Forum, 3GPP)
LCC	Lower Camel Case
LTP	Logical Termination Point
NA	Not Applicable
OMG	Object Management Group
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:

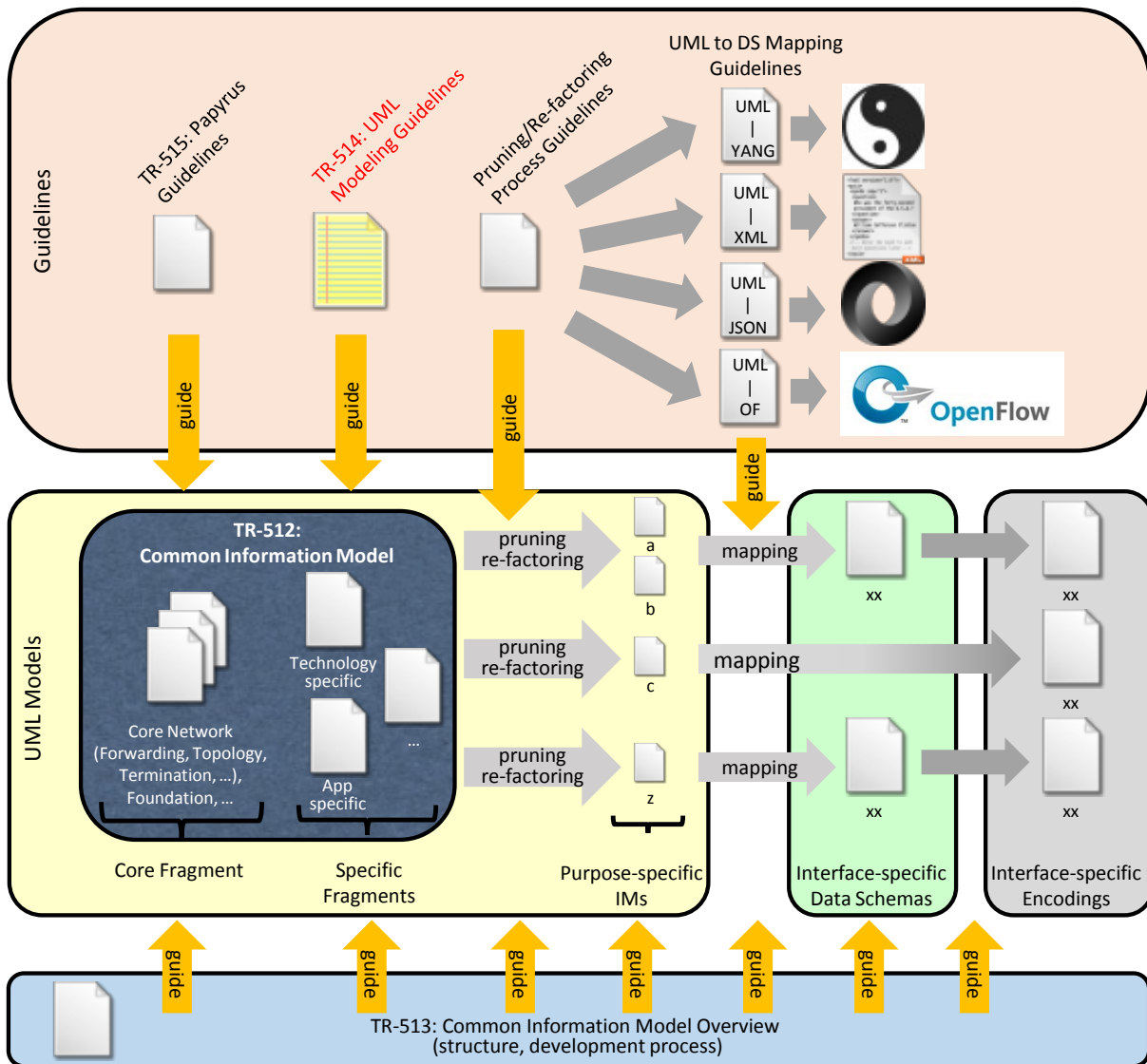


Figure 4.1: Specification Architecture

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.

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 static¹ representation of an entity, including properties and attributes; i.e., data model, the static part of the model.

5.1.1 Object Class Notation

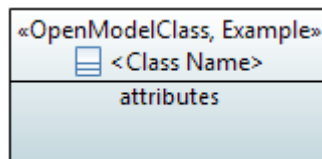


Figure 5.1: Graphical Notation for Object Classes

As highlighted in Figure 5.1, an object class is represented with a name compartment and an attributes compartment. The name compartment contains also the assigned lifecycle stereotypes. 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.

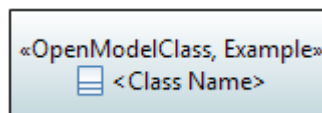


Figure 5.2: Graphical Notation for Object Classes without Attributes Compartment

¹ Not about operations acting on the entity.

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 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 since the static part and the dynamic part of the model are decoupled, the operation compartment, is not used and always hidden.

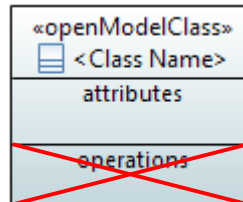




Figure 5.3: Graphical Notation for Object Classes with Attributes and Deprecated Operations Compartment

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 “Applied comments” field in Papyrus; i.e., the “Owned comments” field must not be used. The complete documentation should be written in a single comment; i.e., at most one “Applied comment”.
- 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 properties are defined in the «OpenModelClass» stereotype which extents ( Extension) by default (required) the «metaclass» Class:

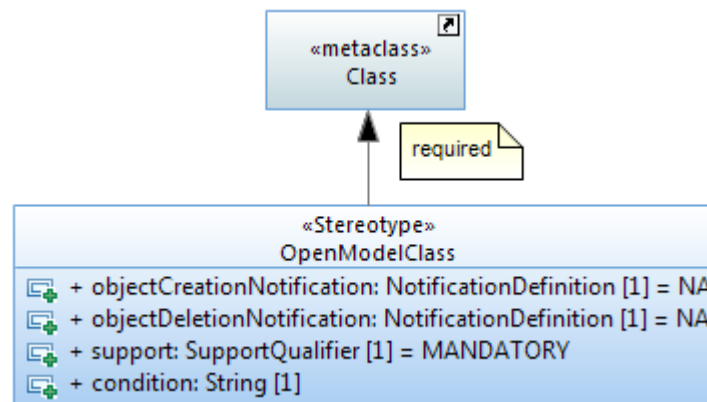


Figure 5.4: «OpenModelClass» Stereotype

- `objectCreationNotification` (only relevant in the purpose-specific modules of the 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 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.

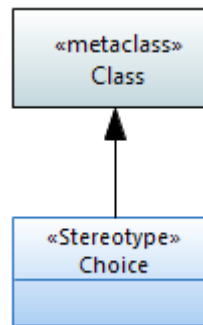


Figure 5.5: Potential Choice Annotation for Object Classes

- UML/Papyrus defined class properties that are not used:
 - Is leaf (default = false)
 - Is active (default = false)
 - Visibility (default = public)

5.2 Attributes in Object Classes

Attributes contain the properties² 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:

```

|«<list of stereotypes>»| <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.

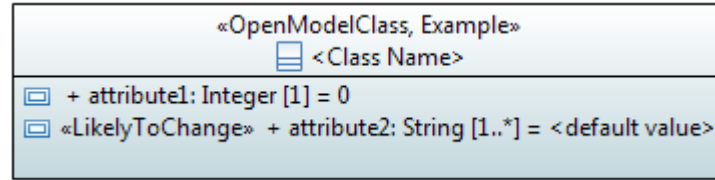


Figure 5.6: Graphical Notation for Object Classes with Attributes

5.2.2 Attribute Properties


An attribute has the following properties:

- **Name**
Follows Lower Camel Case (LCC) and is unique across all attribute names within the inheritance tree. An example of Lower Camel Case: subNetworkConnectionIdentifier. 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 “Applied comments” field in Papyrus; i.e., the “Owned comments” field must not be used. The complete documentation should be written in a single comment; i.e., at most one “Applied comment”.
- **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 false.
- **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.
- **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 properties are defined in the «OpenModelAttribute» stereotype which extends ( Extension) by default (required) the «metaclass» Property:

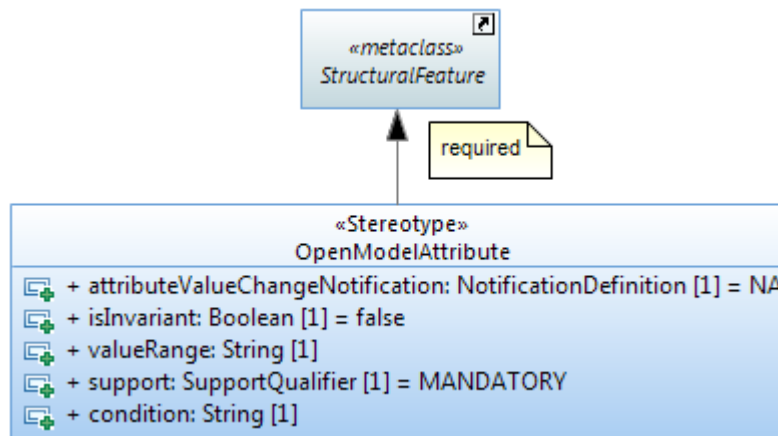


Figure 5.7: «OpenModelAttribute» Stereotype

- **attributeValueChangeNotification** (only relevant in the purpose-specific modules of the 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.
- **condition**
This property contains the condition for the condition-related support qualifiers.

- Other 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.

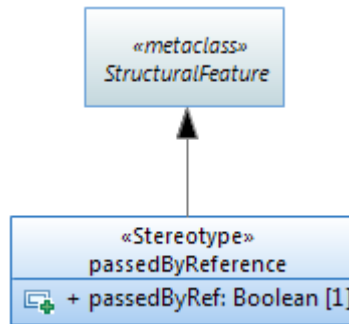


Figure 5.8: «PassedByReference» Stereotype

- UML/Papyrus defined attribute properties that are not used:
 - Is derived (default = false)
 - Is derived union (default = false)
 - Is leaf (default = false)
 - Is static (default = false)
 - Visibility (default = public)

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 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 “_classB” pointing to ClassB and vice versa.



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.

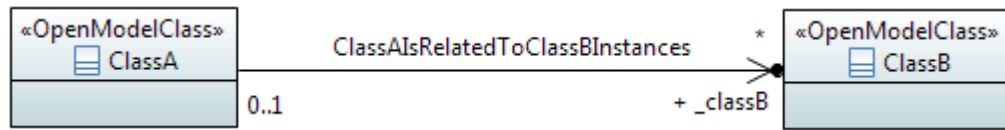


Figure 5.10: Unidirectional Association Relationship Notation

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.

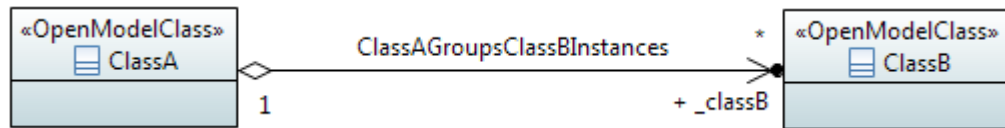


Figure 5.12: Aggregation Association Relationship Notation

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.

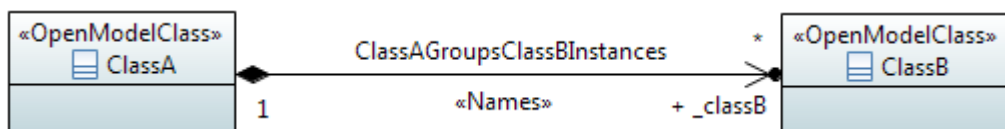


Figure 5.13: Composite Aggregation Association Relationship Notation

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.

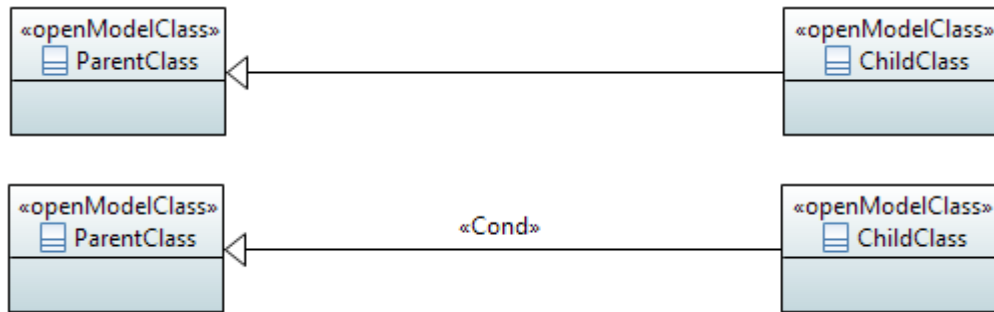


Figure 5.14: Generalization Relationship Notation (normal and conditional)

“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.

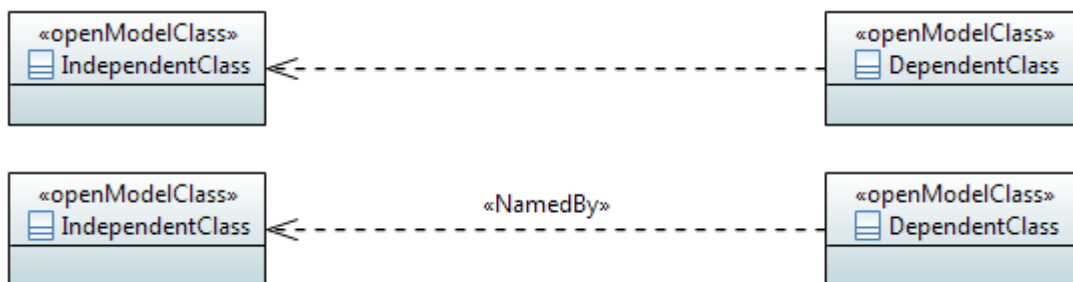


Figure 5.15: Dependency Relationship Notation (normal and naming)

The realization relationship along with the «PruneAndRefactor» stereotype indicates the association between a Core Model object class or relationship and the cloned Purpose Specific Model object class or relationship.

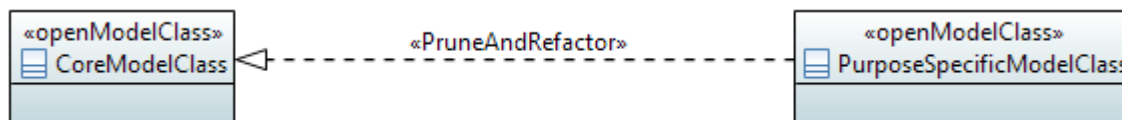


Figure 5.16: Realization Relationship Notation

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 “Applied comments” field in Papyrus; i.e., the “Owned comments” field must not be used. The complete documentation should be written in a single comment; i.e., at most one “Applied comment”.
- **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 (Member End) of the association.
Only navigable Member 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.
Note: The Owner of a navigable Member End has to be the Classifier to become an attribute in the object class.

Member End	
Name	<input type="text" value="_classB"/>
Owner	<input type="text" value="Classifier"/>
Navigable	<input checked="" type="radio"/> true <input type="radio"/> false

Figure 5.17: Owner of a navigable Member End

- **Role Multiplicity**
Identifies the number of object instances that can participate in an instance of the association.
- **Additional 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.
 - «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.
 - «PruneAndRefactor»
This «PruneAndRefactor»stereotype identifies that a realization association is used to identify pruning and refactoring.

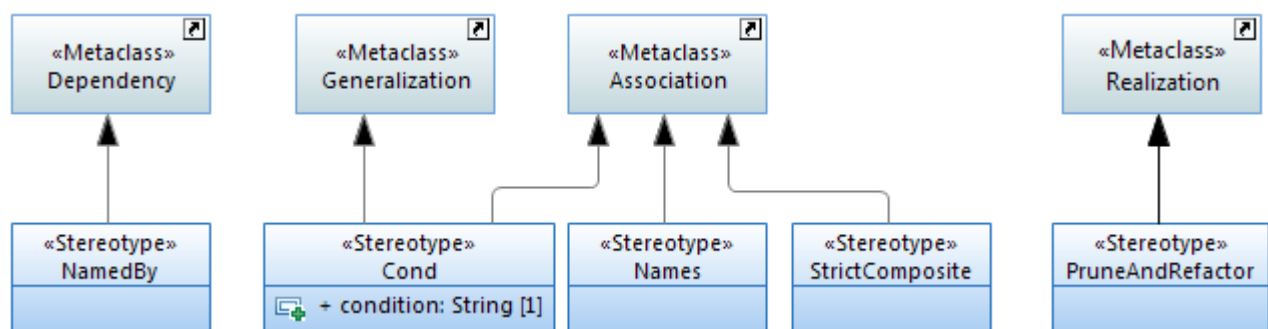


Figure 5.18: Potential Annotations for Associations

- UML/Papyrus defined attribute properties that are not used:
 - Visibility (default = public)

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 information model; see Figure 4.1.

5.4.1 «Interface» Notation

Interfaces are identified by the stereotype «Interface».

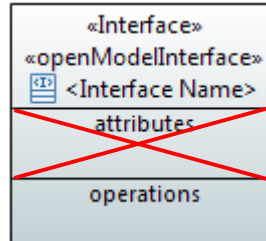


Figure 5.19: Graphical Notation for «Interface»

«Interfaces» usually have name, attributes and operations compartments. 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.

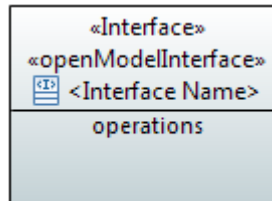



Figure 5.20: Graphical Notation for «Interface» without Attributes Compartment


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 “Applied

comments” field in Papyrus; i.e., the “Owned comments” field must not be used. The complete documentation should be written in a single comment; i.e., at most one “Applied comment”.

- Superinterface(s)
Inheritance and multiple inheritance may be used.
- Abstract
Indicates if the «Interface» can be instantiated or is just used for inheritance.
- Additional properties are defined in the «OpenModelInterface» stereotype which extends ( Extension) by default (required) the «metaclass» Interface:

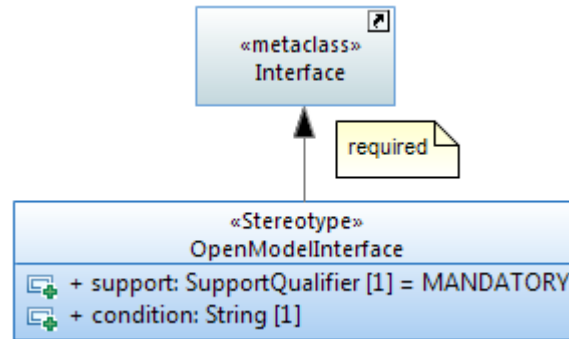


Figure 5.21: «OpenModelInterface» Stereotype

- 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:
 - Is leaf (default = false)
 - Visibility (default = public)

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 information model; see Figure 4.1.

5.5.1 Operation Notation

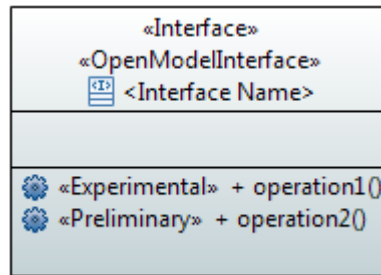


Figure 5.22: Graphical Notation for «Interface» with Operations

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.
- Documentation
Contains a short summary of the usage. The documentation is carried in the “Applied comments” field in Papyrus; i.e., the “Owned comments” field must not be used. The complete documentation should be written in a single comment; i.e., at most one “Applied comment”.
- 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.
- **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 properties are defined in the «OpenModelOperation» stereotype which extends ( **Extension**) by default (required) the «metaclass» Operation:

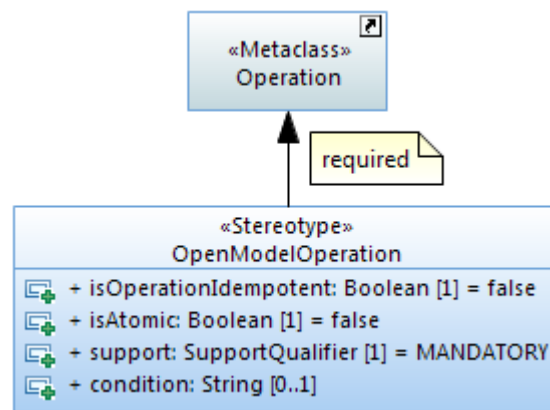


Figure 5.23: «OpenModelOperation» Stereotype

- **isOperationIdempotent (Boolean)**
This property defines if the operation is idempotent (true) or not (false).
Example: When an operation is going to create an object instance which does already exist, an idempotent operation would return success and a non-idempotent operation would return an exception.
- **isAtomic (Boolean)**
This property identifies if the operation is best effort or is successful / not successful as a whole.

- 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:
 - Is leaf (default = false)
 - Is query (default = false)
 - Is static (default = false)

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 information model; see Figure 4.1.

5.6.1 Parameter Notation

The notation is:

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

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

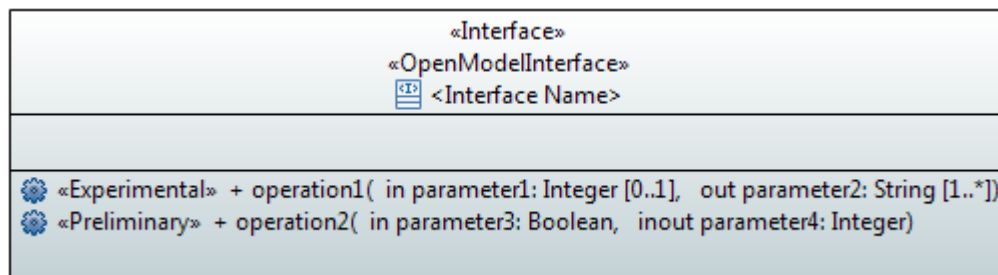



Figure 5.24: Graphical Notation for «Interface» with Operations and Parameters

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 “Applied comments” field in Papyrus; i.e., the “Owned comments” field must not be used. The

complete documentation should be written in a single comment; i.e., at most one “Applied comment”.

- **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 properties are defined in the «OpenModelProperty» stereotype which extends ( Extension) by default ({required}) the «metaclass» Parameter:**

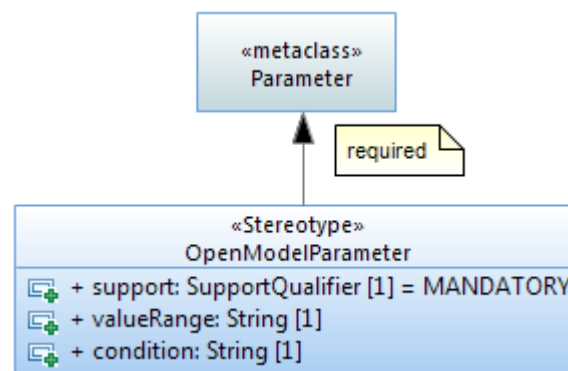


Figure 5.25: «OpenModelProperty» Stereotype

- **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 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.

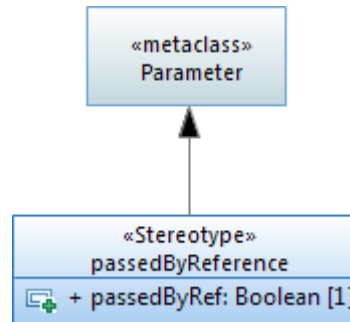


Figure 5.26: «PassedByReference» Stereotype

- UML/Papyrus defined parameter properties that are not used:
 - Is exception (default = false)
 - Is stream (default = false)
 - Is unique (default = true)
 - Visibility (default = public)

5.7 Notifications

Note: Notifications may only be defined in the purpose-specific modules of the 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».

5.7.1 Notification Notation

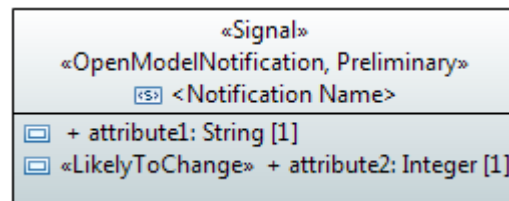



Figure 5.27: Graphical Notation for «Signal»

5.7.2 Notification Properties

A notification/signal  has the following properties:

- Name
Follows Upper Camel Case (UCC). Each notification/signal in the model has a unique name. An example of Upper Camel Case: ObjectCreationNotification.

- **Documentation**
Contains a short summary of the usage. The documentation is carried in the “Applied comments” field in Papyrus; i.e., the “Owned comments” field must not be used. The complete documentation should be written in a single comment; i.e., at most one “Applied comment”.
- **Superclass(es)**
Inheritance and multiple inheritance may be used to deal with shared properties.
- **Abstract**
Indicates if the notification/signal can be instantiated or is just used for inheritance.
- **Additional properties are defined in the «OpenModelNotification» stereotype which extends ( Extension) by default (required) the «metaclass» Signal:**

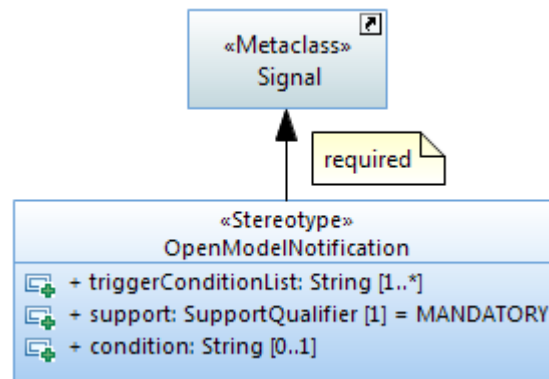


Figure 5.28: «OpenModelNotification» Stereotype

- **triggerConditionList**
This property provides the list of conditions that cause the notification.
- **support**
This property qualifies the support of the notification/signal 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 class properties that are not used:**
 - Is leaf (default = false)
 - Visibility (default = public)

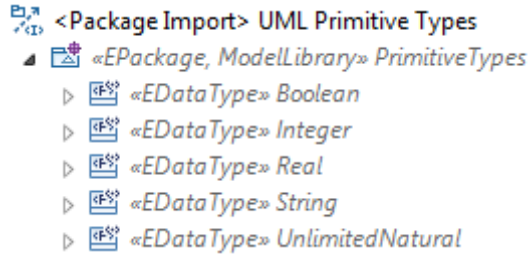
5.8 Types

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

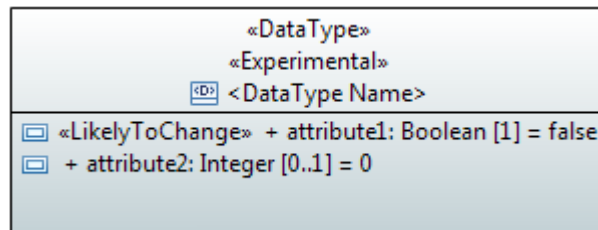


Figure 5.29: Graphical Notation for «DataType»

Note: Default values may not be shown in any class diagram.

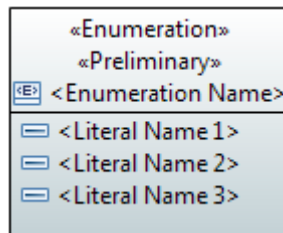


Figure 5.30: Graphical Notation for «Enumeration»

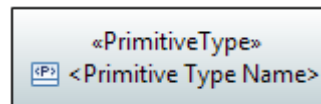


Figure 5.31: Graphical Notation for «PrimitiveType»

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 “Applied comments” field in Papyrus; i.e., the “Owned comments” field must not be used. The complete documentation should be written in a single comment; i.e., at most one “Applied comment”.
- 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 properties
 - Choice
This stereotype identifies a data type as a choice between different alternatives; see also section 7.5.
 - Exception
This stereotype defines a data type used for an operation exception.

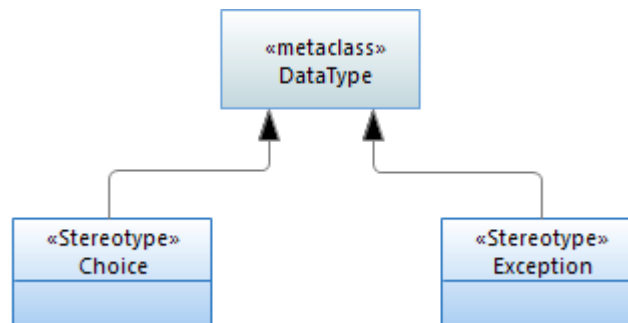


Figure 5.32: Potential Annotations for Data Types

- UML/Papyrus defined attribute properties that are not used:
 - Is abstract (default = false)
 - Is leaf (default = false)

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 and C. 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.

6 UML Profile Definitions

6.1 Additional Properties Definitions

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

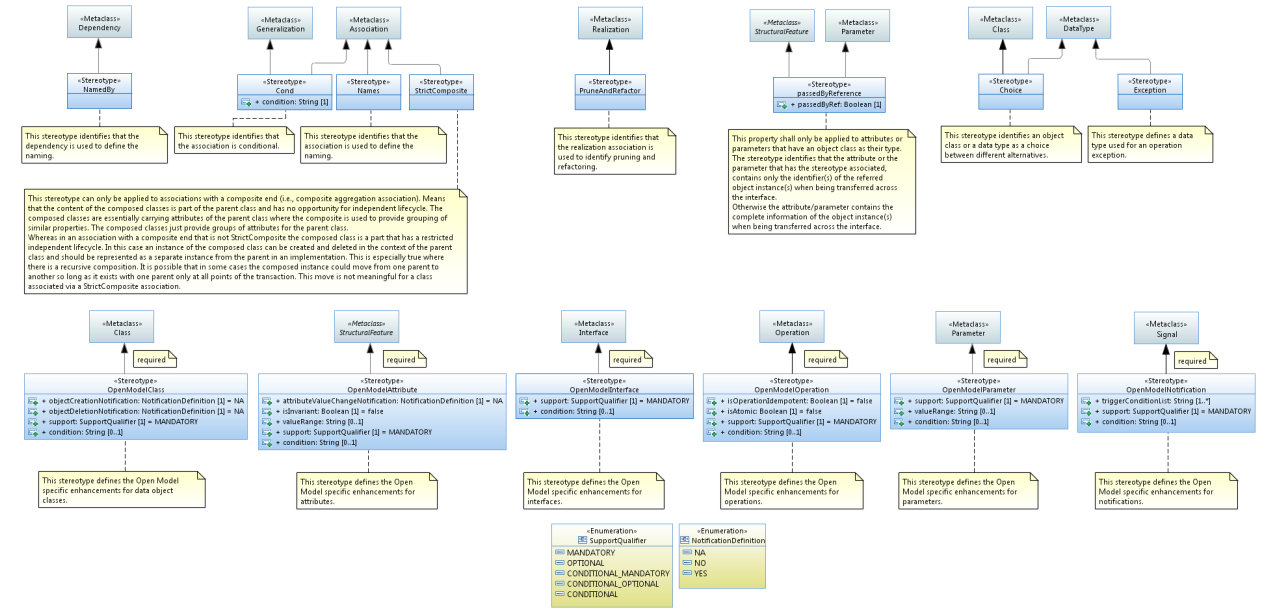


Figure 6.1: UML Artifact «Stereotypes»

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

Stereotype	Name of property	Type	Allowed values	Default value	Associated to metaclass
OpenModelClass	objectCreationNotification	enumeration	NO, YES, NA	NA	Class
	objectDeletionNotification	enumeration	NO, YES, NA	NA	
	support	enumeration	MANDATORY OPTIONAL CONDITIONAL MANDATORY_OPTIONAL CONDITIONAL_OPTIONAL CONDITIONAL	MANDATORY	
	condition	string			
OpenModelAttribute	attributeValueChangeNotification	enumeration	NO, YES, NA	NA	Property
	isInvariant	Boolean	true/false	false	
	valueRange	string		NA	

Stereotype	Name of property	Type	Allowed values	Default value	Associated to metaclass
	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
	isAtomic	Boolean	true/false	false	
	support	enumeration	MANDATORY OPTIONAL CONDITIONAL_ MANDATORY_ CONDITIONAL_ OPTIONAL CONDITIONAL	MANDATORY	
	condition	string			
OpenModelParameter	valueRange	string		NA	Parameter
	support	enumeration	MANDATORY OPTIONAL CONDITIONAL_ MANDATORY_ CONDITIONAL_ OPTIONAL CONDITIONAL	MANDATORY	
	condition	string			
OpenModelNotification	triggerConditionList	String			Signal
	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:

- **Deprecated**
This stereotype indicates that the entity may become obsolete in the near future. It may still be used in new implementation.
- **Example**
This stereotype 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 stereotype 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 stereotype 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 stereotype 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 stereotype 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 stereotype 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.

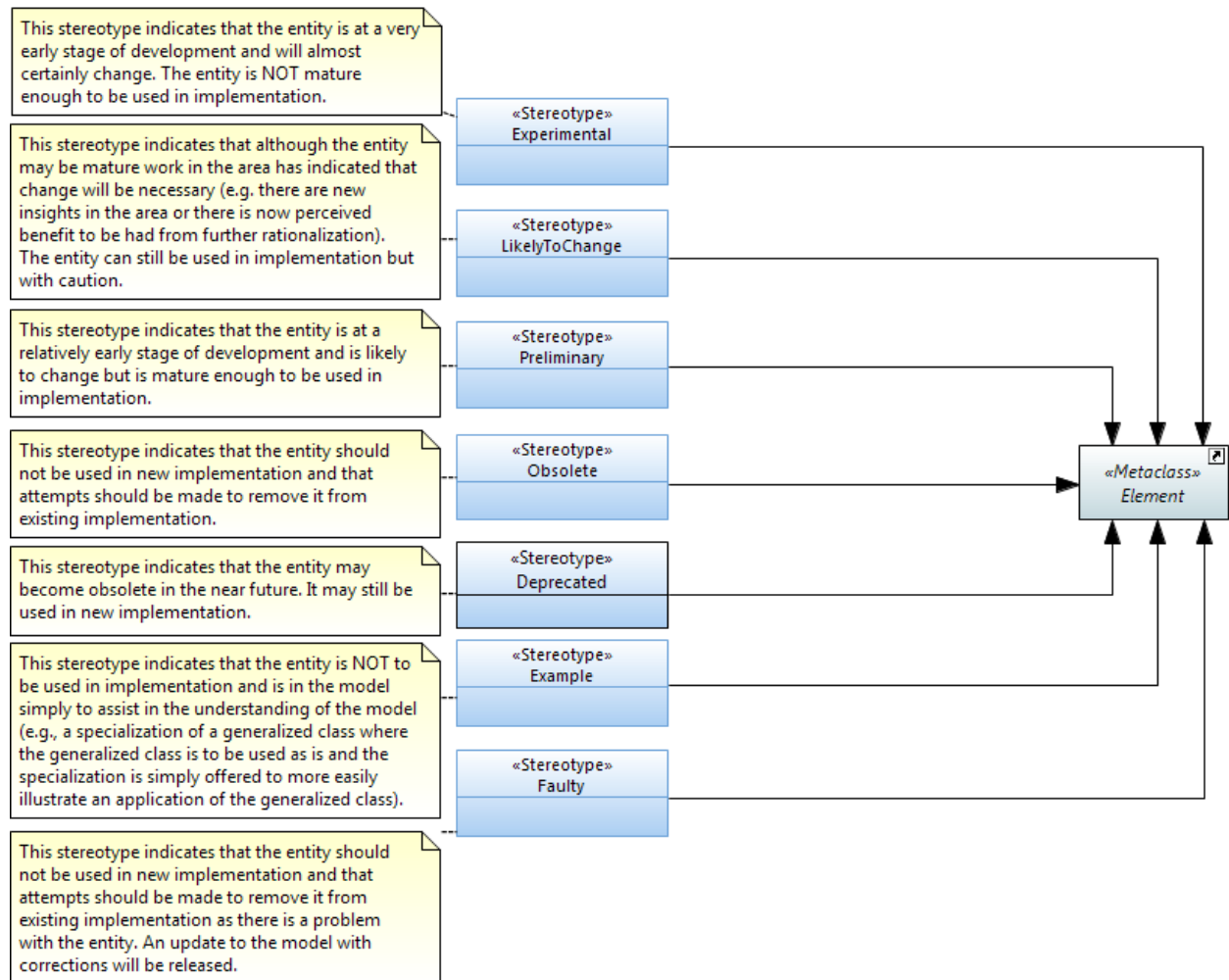


Figure 6.2: Lifecycle «Stereotypes»

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:

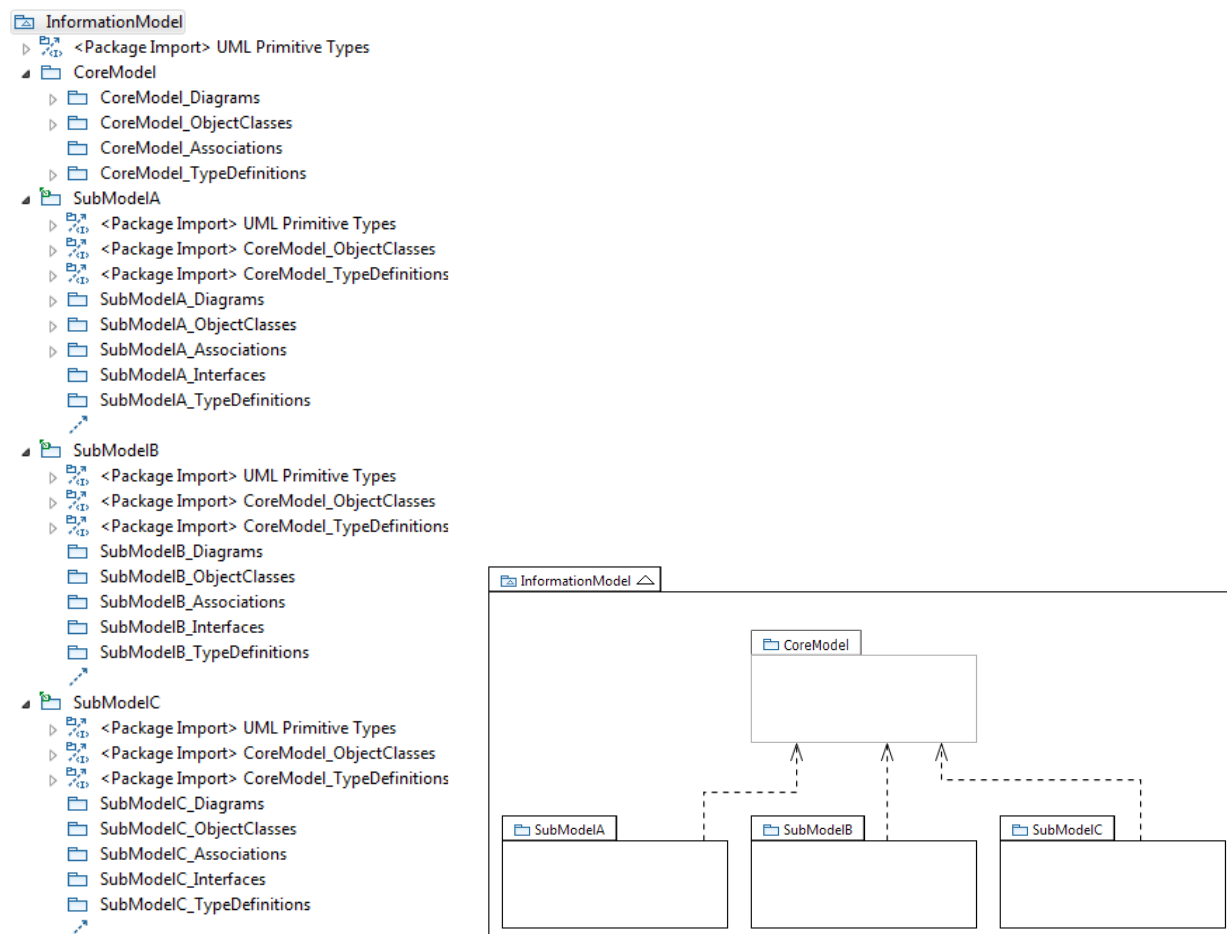


Figure 7.1: Core Model and Sub-Models

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 Model Structure

The Information Model is structured into a Common Information Model and additional Specific Views which are based on the Core Model. Specific models may also be added by other SDOs.

A Core Modeling team (with members from many SDOs) defines and maintains the generic functions in the Core Model.

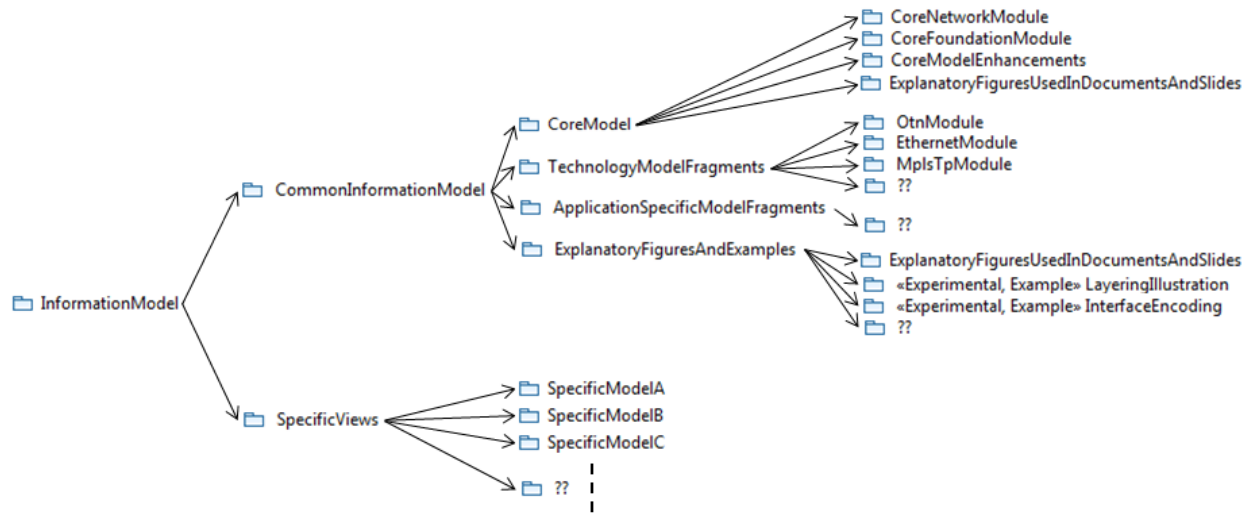


Figure 7.2: Model Structure (snapshot)

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.

- ▢ Associations
- ▢ Diagrams
- ▢ Interfaces
- ▢ Notifications
- ▢ ObjectClasses
- ▢ Rules
- ▢ TypeDefinitions

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 7.4 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.

→ 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).

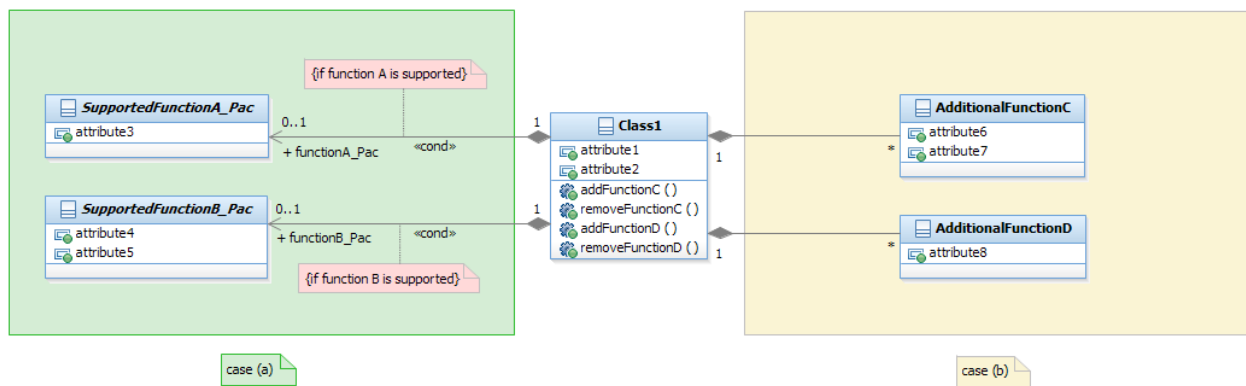


Figure 7.4: Flexible Attribute Assignment to Object Classes

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.

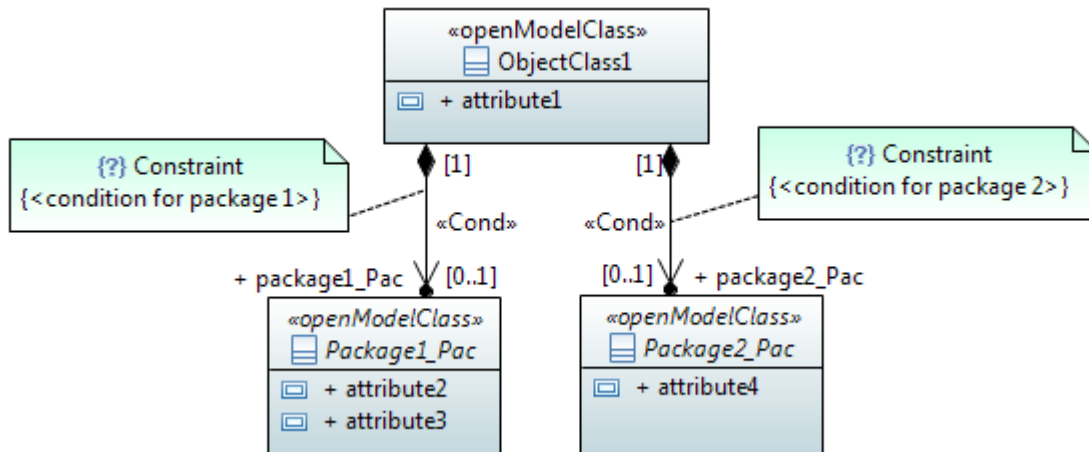


Figure 7.5: Enhancing Object Classes Using Conditional Packages

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 ClientObjectClass_Alternative3.

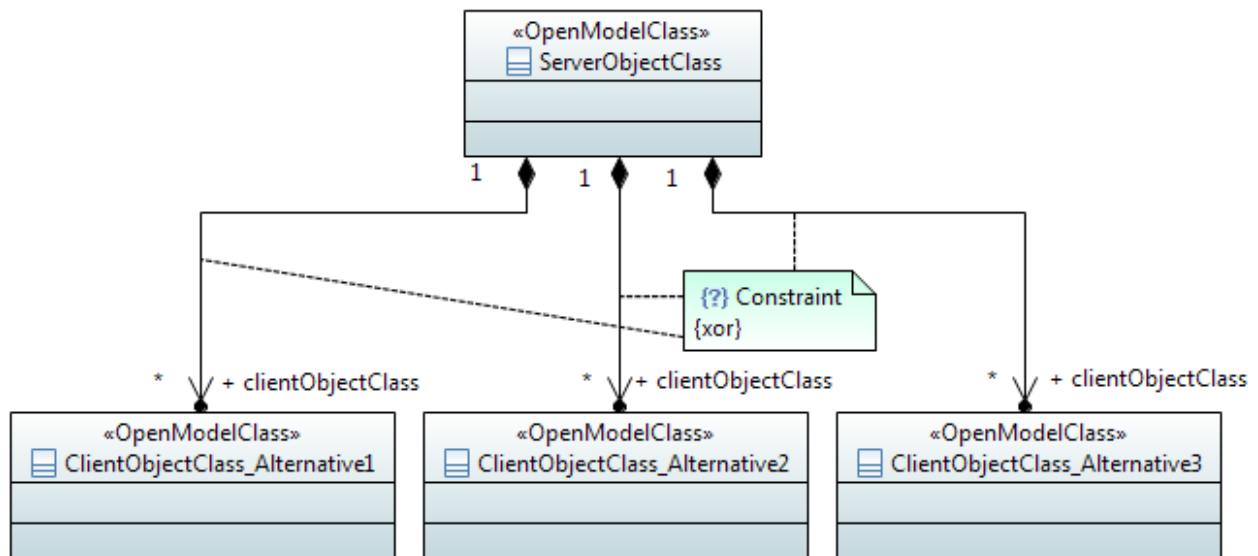


Figure 7.6: {xor} Notation

7.5.1.3 Name style

It has no name so there is no name style.

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 7.5.1).

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.

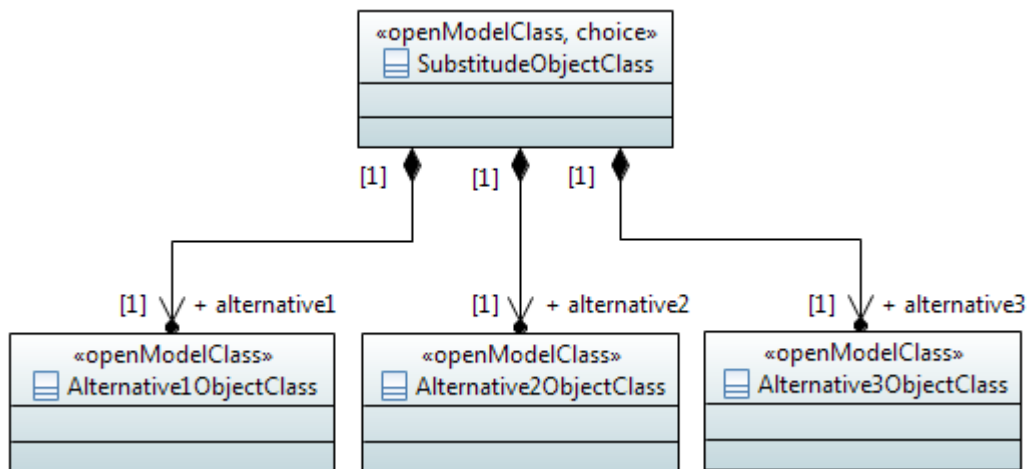


Figure 7.7: Information Model Element Example Using «Choice» Notation

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.

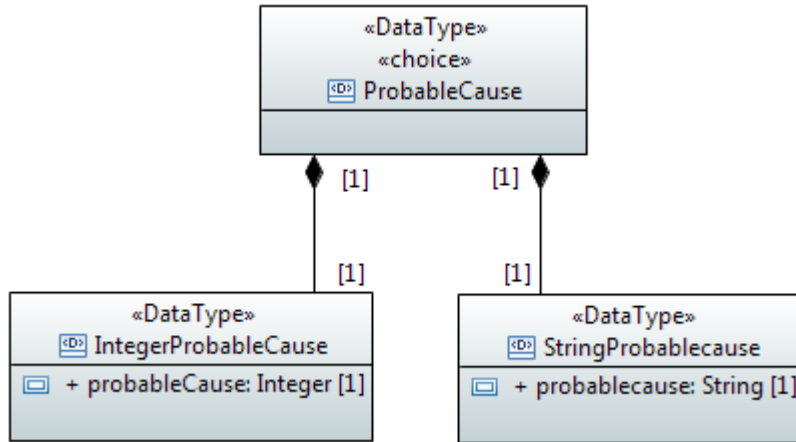


Figure 7.8: Operations Model Element Example Using «Choice» Notation

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.

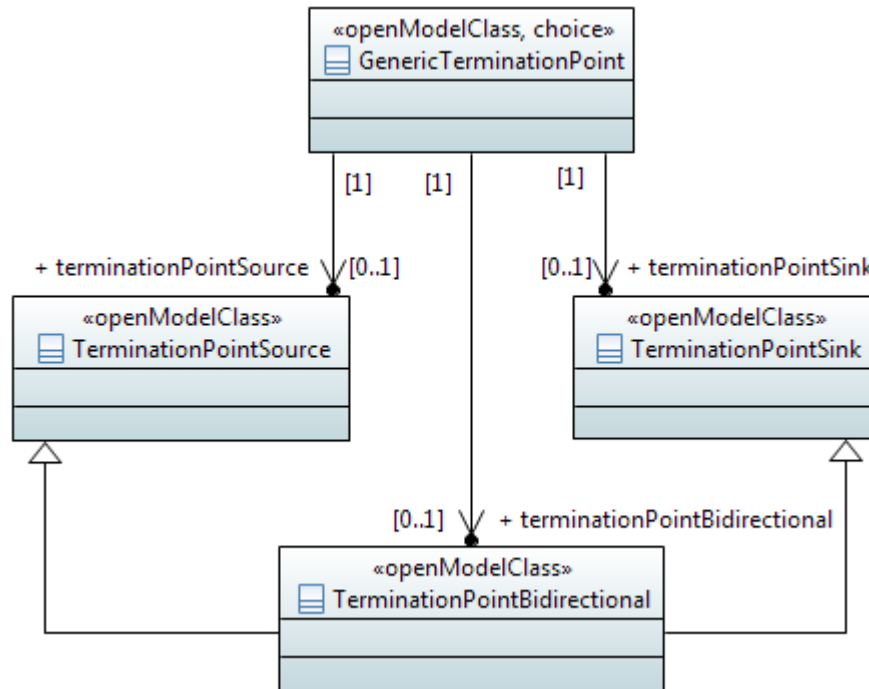


Figure 7.9: Sink/Source/Bidirectional Termination Points Example Using «Choice» Notation

7.5.2.3 Name style

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

7.6 Diagram Guidelines

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.