IISOMI 514
UML Modeling Guidelines

Version 1.2
September 20, 2016

Work in progress!
Disclaimer

CREATIVE COMMONS ATTRIBUTION 4.0 INTERNATIONAL PUBLIC LICENSE

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, you are granted the Licensed Rights in consideration of your acceptance of these terms and conditions, and the Licensor grants Your such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions. For full details, explanations, and examples of the use of this License, please visit:

http://creativecommons.org/licenses/by/4.0/legalcode

Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

Open Networking Foundation
2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303
http://www.opennetworking.org
http://opensourcesdn.org
Content

Content......................................................................................................................................................... 3

List of Figures ............................................................................................................................................. 4

List of Tables ............................................................................................................................................... 6

Document History ....................................................................................................................................... 7

1 Introduction ......................................................................................................................................... 8

2 References ........................................................................................................................................... 8

3 Abbreviations ...................................................................................................................................... 8

4 Overview .............................................................................................................................................. 9
  4.1 Documentation Overview .............................................................................................................. 9
  4.2 Modeling approach ..................................................................................................................... 10
  4.3 General Requirements ................................................................................................................ 11

5 UML Artifact Descriptions ................................................................................................................ 11
  5.1 Classes ....................................................................................................................................... 11
    5.1.1 Class Notation .................................................................................................................. 12
    5.1.2 Class Properties ............................................................................................................... 12
  5.2 Attributes in Classes ................................................................................................................... 15
    5.2.1 Attribute Notation .............................................................................................................. 15
    5.2.2 Attribute Properties ........................................................................................................... 15
  5.3 Associations ................................................................................................................................ 19
    5.3.1 Association Notation ......................................................................................................... 19
    5.3.2 Association Properties ...................................................................................................... 22
  5.4 Interfaces .................................................................................................................................... 24
    5.4.1 «Interface» Notation ......................................................................................................... 24
    5.4.2 «Interface» Properties ...................................................................................................... 25
  5.5 Interface Operations ................................................................................................................... 26
    5.5.1 Operation Notation ........................................................................................................... 26
    5.5.2 Operation Properties ........................................................................................................ 26
  5.6 Operation Parameters ................................................................................................................. 28
    5.6.1 Parameter Notation .......................................................................................................... 28
    5.6.2 Parameter Properties ....................................................................................................... 29
  5.7 Notifications ................................................................................................................................ 31
    5.7.1 Notification Notation ......................................................................................................... 31
    5.7.2 Notification Properties ...................................................................................................... 31
5.8 Data Types.................................................................................................................................. 32
  5.8.1 Type Notation................................................................................................................... 32
  5.8.2 Type Properties................................................................................................................ 33
  5.8.3 UML Primitive Types........................................................................................................ 34
  5.8.4 Pre-defined Data Types.................................................................................................... 35

5.9 Qualifiers and Conditions............................................................................................................ 36

5.10 Use Cases.................................................................................................................................. 38

5.11 Activities...................................................................................................................................... 39

5.12 State Machines........................................................................................................................... 39

6  UML Profile Definitions..................................................................................................................... 40
  6.1 Additional Properties for individual UML artifacts ............................................................... 40
  6.2 Additional Properties for all UML artifacts.............................................................................. 44
    6.2.1 LifecycleState Property..................................................................................................... 45
    6.2.2 Reference Property .......................................................................................................... 47
    6.2.3 Example Property ............................................................................................................. 48

7  Recommended Modeling Patterns .................................................................................................. 48
  7.1 File Naming Conventions............................................................................................................ 48
  7.2 Model Structure........................................................................................................................... 48
    7.2.1 Generic Model Structure................................................................................................... 48
    7.2.2 Model Structure................................................................................................................ 49
  7.3 Flexible Attribute Assignment to Classes ................................................................................... 50
  7.4 Use of Conditional Packages...................................................................................................... 51
  7.5 Use of XOR/Choice..................................................................................................................... 52
    7.5.1 Xor Constraint................................................................................................................... 52
    7.5.2 «Choice»........................................................................................................................... 53
  7.6 Proxy Class Modeling................................................................................................................. 55
  7.7 Diagram Guidelines.................................................................................................................... 56
    7.7.1 Using Colors..................................................................................................................... 57
    7.7.2 Style Sheets...................................................................................................................... 57

8  Main Changes between Releases.................................................................................................... 57
  8.1 Summary of main changes between version 1.0 and 1.1........................................................ 57
  8.2 Summary of main changes between version 1.1 and 1.2........................................................ 57

List of Figures

Figure 4.1: Specification Architecture ............................................................................................. 10
Figure 5.1: Structural/Behavioral Features in UML 2.5 Metamodel................................................ 11
Figure 5.2: Graphical Notation for Classes....................................................................................... 12
Figure 5.3: Graphical Notation for Classes without Attributes Compartment .................................. 12
Figure 5.4: Graphical Notation for Classes with Attributes and Deprecated Operations Compartment ..... 12
Figure 5.38: Example of Use Case Diagram ................................................................. 38
Figure 5.39: Example Business Process Modeling ............................................................ 39
Figure 6.1: Required «Stereotypes» .................................................................................. 41
Figure 6.2: Optional «Stereotypes» .................................................................................... 42
Figure 6.3: Lifecycle «Stereotypes» .................................................................................... 46
Figure 6.4: OpenModelProfile LifecycleState State Machine .............................................. 47
Figure 6.5: Reference «Stereotype» .................................................................................... 48
Figure 6.6: Example «Stereotype» ...................................................................................... 48
Figure 7.1: Core Model and SubModels .............................................................................. 49
Figure 7.2: Model Structure (snapshot) ............................................................................... 50
Figure 7.3: Pre-defined Packages at the Bottom Level of each UML Model (Example) .......... 50
Figure 7.4: Flexible Attribute Assignment to Classes .......................................................... 51
Figure 7.5: Enhancing Classes Using Conditional Packages .............................................. 52
Figure 7.6: {xor} Notation .................................................................................................... 53
Figure 7.7: Information Model Element Example Using «Choice» Notation ......................... 54
Figure 7.8: Operations Model Element Example Using «Choice» Notation ......................... 54
Figure 7.9: Sink/Source/Bidirectional Termination Points Example Using «Choice» Notation ... 55
Figure 7.10: Proxy Class Modeling Example ...................................................................... 56

List of Tables
Table 5.1: Table 11.1/[3] – Collection Types for Properties .............................................. 16
Table 5.2: Attribute Property Dependencies ....................................................................... 18
Table 6.1: UML Artifact Properties Defined in Complex «Stereotypes» .............................. 43
### Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>March 13, 2015</td>
<td>Initial version</td>
</tr>
<tr>
<td>1.1</td>
<td>Nov. 30, 2015</td>
<td>Version 1.1</td>
</tr>
<tr>
<td>1.2</td>
<td>Sept. 20, 2016</td>
<td>Version 1.2</td>
</tr>
</tbody>
</table>
1 Introduction

This document 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 harmonious information models, only a selected subset of these artifacts is used in the UML model guidelines in this document. The semantic of the selected artifacts is defined in [2].

The guidelines of each basic model artifact are 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].

Note:
This version of the guidelines is still work in progress! Known open issues are marked in yellow and described by comments.

2 References


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
4 Overview

4.1 Documentation Overview

This document is part of a suite of guidelines. The location of this document within the documentation architecture is shown in Figure 4.1 below:
4.2 Modeling approach

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

Important note:
It is important to understand that the UML class diagrams always show only parts of the underlying model. E.g., classes shown without attributes do not mean that the class has no attribute, i.e., attributes could be hidden in a diagram. The full model can be viewed in its entirety through the UML tool (i.e., Papyrus; XMI codes in the .uml file) and a view of key details is provided in a data dictionary.
Also note that in this document, use of the term “Class” refers to a UML class, unless otherwise specified.

4.3 General Requirements

- UML 2.5 (Unified Modeling Language) is used for specifying the model.
- The model shall be management/control 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

Structural/behavioral features

The UML 2.5 specification [3] distinguishes between structural and behavioral features. The structural modeling is using Properties contained in Classes and the behavioral modeling is using Operations contained in Interfaces.

![Figure 5.1: Structural/Behavioral Features in UML 2.5 Metamodel]

5.1 Classes

Classes are used to convey a structural (often called static) representation of an entity, including properties and attributes; i.e., data model, the structural part of the model.

1 Not about operations acting on the entity.
5.1.1 Class Notation

As highlighted in Figure 5.2, a class is represented with a name compartment and an attributes compartment. It is recommended that 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 hidden to reduce clutter, in others only a subset of the attributes is exposed to focus attention on those attributes. 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 5.3.

It is recommended that the name compartment also show stereotypes for the class where relevant. When showing stereotypes, the compartment may 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 5.4, but since the structural part and the behavioral part of the model are decoupled, the operation compartment, is not used and always hidden.

5.1.2 Class Properties

A class has the following properties:

- Name
  Follows Upper Camel Case style (UCC). Each class in the model has a unique name. An example of Upper Camel Case: SubNetworkConnection.
• Documentation
  Contains a short definition. 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”.

  Figure 5.5: «OpenModelClass» Stereotype

2 Because of Papyrus tool reasons, you must not create comments directly in the class diagram and attach it by a link to the class. Such comments appear in applied comments field too, BUT they don’t appear in the gendoc output.

• 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; i.e., abstract classes will not be instantiated.

• Additional properties are defined in the «OpenModelClass» stereotype which extends ( Extension ) by default (required) the «metaclass» Class:
• objectCreationNotification (only relevant in the purpose-specific models of the information model; see Figure 4.1) Defines whether an object creation notification has to be sent when the instance is created.

• objectDeletionNotification (only relevant in the purpose-specific models of the information model; see Figure 4.1) Defines whether an object deletion notification has to be sent when the instance is deleted.

• support This property qualifies the support of the class class at the management interface. See definition in section 5.8.3.

• condition This property contains the condition for the condition-related support qualifiers.

• Choice This stereotype identifies a class as a choice between different alternatives.

![Figure 5.6: Potential Choice Annotation for Classes](image)

- **Uniqueness of a group of attribute values**

![Diagram of Uniqueness](image)

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

Attributes contain the properties of a class. Note that the roles of navigable association ends become an attribute in the class at the other associated end when this association end is owned by the classifier; see also “Role Type” property in section 5.3.2. Note: The association end can also be owned by the association itself in which case it does not become an attribute.

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.

```
+ attribute1: Integer [1] = 0
+ LikelyToChange: + attribute2: String [1..*] = <default value>
```

Figure 5.7: Graphical Notation for Classes with Attributes

5.2.2 Attribute Properties

An attribute has the following properties:

- Name
  Follows Lower Camel Case (LCC) style and is unique across all attribute names within the inheritance tree. An example of Lower Camel Case: subNetworkConnectionIdentifier.
  It is recommended that all Boolean typed attribute names start with ‘is’ (e.g., ‘isAbstract’), ‘must’ or a verb such as ‘has’ 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 definition. 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

---

3 In Papyrus an attribute is a property.
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

<table>
<thead>
<tr>
<th>Ordered</th>
<th>Unique</th>
<th>Collection type</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>true</td>
<td>Set</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>OrderedSet</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>Bag</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>Sequence</td>
</tr>
</tbody>
</table>

- **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:
partOfObjectKey
This property indicates if the attribute is part of the object key or not. Value = 0 (default) means the attribute is not part of the object key. Values > 0 indicate that the attribute is part of the object key and the value defines the order of the attribute in case the key is composed of more than one attribute. Attributes which are used as a key must be invariant (i.e., property isInvariant = true), must not be optional (i.e., the multiplicity must be [1] or [1..x]) and the multiplicity must be [1] after the Pruning&Refactoring process; i.e., a UML to Data Schema mapping tool must not get a list attribute which is part of the object identifier.

attributeValueChangeNotification (only relevant in the purpose-specific models 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
This property identifies if the value of the attribute can be changed after it has been created.

valueRange
This property identifies the allowed values for the attribute.

bitLength
This optional property defines the bit length of the attribute type; if applicable.

unsigned
This optional property indicates if the attribute type is unsigned (value = true) or signed (value = false); if applicable, otherwise ignored.

encoding
This optional property defines the encoding of the attribute type; if applicable.
counter
This optional property defines the counter type of the attribute type; if applicable.

unit
This optional property contains a textual definition of the unit associated with the attribute value.
The spelling of the unit, including the ones beyond SI scope, shall be in accordance to the NIST Publication 811 “Guide for the Use of the International System of Units (SI)” (http://www.nist.gov/pml/pubs/sp811/index.cfm), section 9 “Rules and Style Conventions for Spelling Unit Names”.

support
This property qualifies the support of the attribute at the management interface. See definition in section 5.8.3.

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 a class defined as their type; i.e., association member ends owned by the class which became attributes. The stereotype is applied on a case by case basis.
The property defines that the attribute contains only the reference (name, identifier, address) of the referred instance(s) when being transferred across the interface. Otherwise the attribute contains the complete information of the instance(s) when being transferred across the interface.

![Figure 5.9: «PassedByReference» Stereotype](Image)

- 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)

<table>
<thead>
<tr>
<th>Table 5.2: Attribute Property Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadOnly</td>
</tr>
<tr>
<td>-----------</td>
</tr>
</tbody>
</table>

© Open Source SDN
### 5.3 Associations

Associations are defined between classes. Associations have association ends. An association end specifies the role that the class at that end performs.

#### 5.3.1 Association Notation

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

Figure 5.10 shows a bi-directional navigable association where each class has a pointer to the other. The association end 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.

![Bidirectional Association Relationship Notation](image)

**Figure 5.10: Bidirectional Association Relationship Notation**
Figure 5.11 shows a unidirectional association (shown with an open arrow at the target class) where only the source class has a pointer to the target class and not vice-versa.

Figure 5.12 shows a non-navigable association where none of the classes have a pointer to the other; i.e., such associations are just for illustration purposes. Non-navigable associations should have a name.

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 coordination called aggregate, in the object that represents the assembly.

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 all instances of ClassB related to an instance classA is tied to the lifecycle of the classA instance.

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

![Generalization Relationship Notation](image)

“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.

![Dependency Relationship Notation](image)
The realization relationship along with the «PruneAndRefactor» stereotype indicates the association between a Core Model class or relationship and the cloned Purpose Specific Model class or relationship.

![Figure 5.17: Realization Relationship Notation](image)

### 5.3.2 Association Properties

An association has the following properties:

- **Name**
  - Follows Upper Camel Case (UCC) style 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. In case of a long class name, it is also allowed to use a short form of the name.

- **Documentation**
  - Contains a short definition. 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) style 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 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 instance(s) when being transferred across the interface. Otherwise the role (becoming an attribute) contains the complete information of the 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 class.

---

**Figure 5.18: Owner of a navigable Member End**

- **Role Multiplicity**
  Identifies the number of class 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.
• 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 models of the information model; see Figure 4.1.

5.4.1 «Interface» Notation

Interfaces are identified by the stereotype «Interface».

«Interfaces» usually have name, attributes and operations compartments. The structural part and the behavioral 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.
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) style and is unique across all «Interface» names in the model.

- **Documentation**
  Contains a short definition. 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.

- **Abstract**
  Indicates if the «Interface» can be instantiated or is just used for inheritance.

  Additional properties are defined in the «OpenModelInterface» stereotype which extends ( ) by default (required) the «metaclass» Interface:

```plaintext
+ support: SupportQualifier [1] = MANDATORY
+ condition: String [1]
```

![Figure 5.22: «OpenModelInterface» Stereotype](image)

- **support**
  This property qualifies the support of the «Interface» at the management interface. See definition in section 5.8.3.

- **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 models of the information model; see Figure 4.1.

5.5.1 Operation Notation

![Figure 5.23: Graphical Notation for «Interface» with Operations](image)

5.5.2 Operation Properties

An operation has the following properties:

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

- **Documentation**
  Contains a short definition. 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**
  Lists 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:
isOperationIdempotent (Boolean)
This property defines if the operation is idempotent (true) or not (false).
Example: When an operation is going to create an 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 5.8.3.

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 models 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.
5.6.2 Parameter Properties

A parameter has the following properties:

- **Name**
  Follows Lower Camel Case (LCC) style

- **Documentation**
  Contains a short definition. 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 «OpenModelParameter» stereotype which extends (Extension) by default ({required}) the «metaclass» Parameter:
• 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 5.8.3.

• 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 instance(s) when being transferred across the interface. Otherwise the parameter contains the complete information of the instance(s) when being transferred across the interface.

• 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 models 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

![Graphical Notation for «Signal»](image)

5.7.2 Notification Properties

A notification/signal has the following properties:

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

- **Documentation**
  Contains a short definition. 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**
  Defined in the «OpenModelNotification» stereotype which extends by default (required) the «metaclass» Signal:

  ![Diagram](image)
5.29 «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 5.8.3.

- **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 Data Types

Data Types are used as type definitions of attributes and parameters.

Data Types are divided into 3 categories:

- **(Complex) Data Types** (further structured; e.g., Host which combines ipAddress and domainName)
- **Primitive Types** (not further structured; e.g., Integer, MAC address).
- **Enumerations**

#### 5.8.1 Type Notation

- **<DataType>**
  - <Experimental>
  - <DataType Name>
  - «LikelyToChange» + attribute1: Boolean [1] = false
  - + attribute2: Integer [0..1] = 0

  Figure 5.30: Graphical Notation for «DataType»

Note: Default values may not be shown in all class diagrams.

- **<Enumeration>**
  - <Preliminary>
  - <Enumeration Name>
  - <Literal Name 1>
  - <Literal Name 2>
  - <Literal Name 3>

  Figure 5.31: Graphical Notation for «Enumeration»
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) style and is unique across all data type names defined in the whole model.

- **Documentation**
  Contains a short definition. 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 5.33: Potential Annotations for Data Types

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

5.8.3 UML Primitive Types

Papyrus already provides the following UML primitive types:

- `Boolean`
- `Integer`
- `Real`
- `String`
- `Unlimited Natural`

Figure 5.34: Primitive Types provided by Papyrus

Note that Papyrus also exposes the internal Eclipse eCore primitives which are not to be used in models.

The UML Primitive Types can be further restricted by the annotation of the following properties contained in the OpenModelAttribute stereotype (see definitions in section 5.2.2):

- bitLength
- unsigned
- encoding
- counter
For example: «UNSIGNED, LENGTH_8_BIT» Integer or «HexEncoded» String.

Note that common floating point types ‘float’ and ‘double’ are represented using the profile as below:

- Float (single-precision, 32-bit IEEE 754 floating point): «LENGTH_32_BIT» Real
- Double (double-precision, 64-bit IEEE 754 floating point): «LENGTH_64_BIT» Real.

### 5.8.4 Pre-defined Data Types

Additional common data types are defined in a separate model library which is imported to every UML model.

Note that model projects should not create their own primitive types. Requests for new primitive types should be made to the IISOMI team so they can be included in the standard Papyrus files and then available to all modeling teams.

Similar data types are grouped together to ease the search of the adequate data type by the model designer. The following groupings and containing data types are under discussion at the time of publication of these guidelines.

![Common Data Types Grouping](image)

**Address related Types**

- IpAddress
- Ipv4Address
- Ipv6Address
- IpAddressNoZone
- Ipv4AddressNoZone
- Ipv6AddressNoZone
- Ipv4Prefix
• Ipv6Prefix
• IpPrefix

```xml
<DataType>
  <Choice>
    <IpPrefix>
      [0..1]
    </IpPrefix>
    [0..1]
  </Choice>
</DataType>
```

- MacAddress

```xml
<PrimitiveType>
  <MacAddress>
    [1]
  </MacAddress>
</PrimitiveType>
```

Date and Time related Types
• Date
• Time

Domain Name and URI related Types
• DomainName

```xml
<DataType>
  <Host>
    [<IpAddress: IpAddress [0..1]]
    [<domainName: DomainName [0..1]]
  </Host>
</DataType>
```

- Uri

Identifier related Types
• ObjectIdentifier
• ObjectIdentifier128

Protocol Field related Types
• Dscp
• IpVersion
• IpV6FlowLabel
• PortNumber

String related Types
• DottedQuad
• Uuid

### 5.9 Qualifiers and Conditions

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 model element shall be supported.
- Definition of O (Optional) qualification:
The model element may, but needs not to, be supported.
- Definition of CM (Conditional-Mandatory) qualification:
The model element shall be supported under certain conditions. If the specified conditions are met then the model element shall be supported.
- Definition of CO (Conditional-Optional) qualification:
The model element may, but needs not to, be supported under certain conditions. If the specified conditions are met then the model element may, but needs not to, be supported. If the specified conditions are not met then the model element shall be supported.
- Definition of C (Conditional) qualification:
Used for model elements 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 model element 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 model element shall be supported. If all the specified constraints are met and are related to optional, then the model element may, but needs not to, be supported. If all the specified constraints are met and are related to "no support", then the model element shall not be supported.

The condition property contains the condition for the condition-related support qualifiers (CM, CO, C). Often different conditional UML artifacts share the same condition. It is therefore recommended to group such conditions within a model based on the supported features. The grouping is provided by the first line of the condition string which must contain the name of the group; i.e., all condition strings of the UML artifacts which share the same condition have the same text in their first line. The second and further lines may contain an explanation of the condition.

![Figure 5.36: Conditional Class Example](image-url)
5.10 Use Cases

Use case diagrams define actors in a system and the defined behavior over a specific interface. The actor is the entity that is invoking the behavior over the interface. In the diagram below, the actor is a stick figure representing a business application that is given a “name” which must be specified in Upper Camel Case (UCC). The use cases, or the defined behavior invoked over an interface, are defined in the “ovals” and specified in their “names” in Upper Camel Case (UCC) also. The tabular format which defines the input, output, description, etc. of a use case is only found in the Interface Profile Specification and is not present in the UML model.
5.11 Activities

Activities defined in UML are used for business process modeling. The primary artifacts used in modeling business processes are as follows:

- **Activity Compartment**: Defines the boundary of the process being defined
- **Activity Partition**: Defines a partitioning boundary of the process
- **Initial Node**: Defines the start of the business process
- **Opaque Action**: Defines an individual process within an activity
- **Control Flow**: Defines the flow control between processes
- **Decision Node**: Defines a decision point between processes
- **Flow Final Node**: Defines the endpoint of a process flow
- **Accept Event Action**: Defines the received event from another component
- **Data Store Node**: Defines the information owned by the component that runs this activity

Other artifacts may be required based upon the business process being defined. The following diagram illustrates as an example the overall Product Lifecycle and Service Lifecycle processes as defined in MEF 50.

![Figure 5.39: Example Business Process Modeling](image)

5.12 State Machines

State machines define state transitions and triggers that must occur for the transitions to take place. The primary artifacts used in modeling state machines are as follows:
• State Machine Compartment  Defines the boundary of the state machine
• Region  Defines a region within a state machine
• Initial State  Defines the initial state
• Transition  Defines the trigger for a state transition to occur
• State  Defines a given state within the state machine
• Final State  Defines the final, or end state

Other artifacts may be required based upon the state machine being defined.

As an example, see the state machine of the Lifecycle Stereotypes in Figure 6.4.

6  UML Profile Definitions

6.1  Additional Properties for individual UML artifacts

Section 5 has already described the additional properties for each UML artifact. All defined stereotypes are shown as an overview in Figure 6.1, Figure 6.2 and Table 6.1 below.
Figure 6.1: Required «Stereotypes>
This stereotype can only be applied to associations with a composite end (i.e., composite aggregation association). 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.

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.

This property shall only be applied to attributes or parameters that have an object class as their type.

The stereotype identifies that the attribute or the parameter that has the stereotype associated, contains only the identifier(s) of the referred object instance(s) when being transferred across the interface.

Otherwise the attribute/parameter contains the complete information of the object instance(s) when being transferred across the interface.

This stereotype identifies an object class or a data type as a choice between different alternatives.

This stereotype defines a data type used for an operation exception.

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 the used as is and the specialization is simply offered to more easily illustrate an application of the generalized class).

This optional stereotype contains a reference upon which the UML artifact is based. A reference to a standard is preferred.
<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Name of property</th>
<th>Type</th>
<th>Allowed values</th>
<th>Default value</th>
<th>Associated to metaclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenModelClass</td>
<td>objectCreation Notification</td>
<td>enumeration</td>
<td>NO, YES, NA</td>
<td>NA</td>
<td>Class</td>
</tr>
<tr>
<td></td>
<td>objectDeletion Notification</td>
<td>enumeration</td>
<td>NO, YES, NA</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>support</td>
<td>enumeration</td>
<td>MANDATORY OPTIONAL CONDITIONAL_ MANDATORY CONDITIONAL_ OPTIONAL CONDITIONAL</td>
<td>MANDATORY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>condition</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenModelAttribute</td>
<td>partOfObjectKey</td>
<td>Integer</td>
<td>0,1,2,3,...</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>attributeValue Change Notification</td>
<td>enumeration</td>
<td>NO, YES, NA</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>isInvInvariant</td>
<td>Boolean</td>
<td>true/false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td></td>
<td>valueRange</td>
<td>string</td>
<td></td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bitLength</td>
<td>BitLength</td>
<td>NA LENGTH_8_BIT LENGTH_16_BIT LENGTH_32_BIT LENGTH_64_BIT</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned</td>
<td>Boolean</td>
<td>true/false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td></td>
<td>encoding</td>
<td>Encoding</td>
<td>NA BASE_64 HEX OCTET</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>counter</td>
<td>Counter</td>
<td>NA COUNTER GAUGE ZERO_COUNTER</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unit</td>
<td>String</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>support</td>
<td>enumeration</td>
<td>MANDATORY OPTIONAL CONDITIONAL_ MANDATORY CONDITIONAL_ OPTIONAL CONDITIONAL</td>
<td>MANDATORY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>condition</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stereotype</td>
<td>Name of property</td>
<td>Type</td>
<td>Allowed values</td>
<td>Default value</td>
<td>Associated to metaclass</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------</td>
<td>---------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>OpenModelInterface</td>
<td>support</td>
<td>enumeration</td>
<td>MANDATORY OPTIONAL CONDITIONAL MANDATORY CONDITIONAL OPTIONAL CONDITIONAL</td>
<td>MANDATORY</td>
<td>Interface</td>
</tr>
<tr>
<td></td>
<td>condition</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenModelOperation</td>
<td>isOperationIdempotent</td>
<td>Boolean</td>
<td>true/false</td>
<td>false</td>
<td>Operation</td>
</tr>
<tr>
<td></td>
<td>isAtomic</td>
<td>Boolean</td>
<td>true/false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td></td>
<td>support</td>
<td>enumeration</td>
<td>MANDATORY OPTIONAL CONDITIONAL MANDATORY CONDITIONAL OPTIONAL CONDITIONAL</td>
<td>MANDATORY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>condition</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenModelParameter</td>
<td>valueRange</td>
<td>string</td>
<td></td>
<td>NA</td>
<td>Parameter</td>
</tr>
<tr>
<td></td>
<td>support</td>
<td>enumeration</td>
<td>MANDATORY OPTIONAL CONDITIONAL MANDATORY CONDITIONAL OPTIONAL CONDITIONAL</td>
<td>MANDATORY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>condition</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenModelNotification</td>
<td>triggerCondition</td>
<td>String</td>
<td></td>
<td></td>
<td>Signal</td>
</tr>
<tr>
<td></td>
<td>support</td>
<td>enumeration</td>
<td>MANDATORY OPTIONAL CONDITIONAL MANDATORY CONDITIONAL OPTIONAL CONDITIONAL</td>
<td>MANDATORY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>condition</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 6.2 Additional Properties for all UML artifacts

This section defines the additional properties that are associated to all UML artifacts.
6.2.1 LifecycleState Property

All UML artifacts (packages, classes, attributes, interfaces, operations, parameters, data types, associations and generalizations) are appended with one of the following lifecycle states:

- **Deprecated**
  - This stereotype indicates that the entity may become obsolete in the near future. It may still be used in new implementation.
  - The entity should be kept in this state for at least one further release. The team has to decide on a case by case basis when to move it to Obsolete.

- **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.

- **Mature**
  - This stereotype indicates that the entity is fully developed and can be used in implementations without any constraints.

- **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.
  - The entity should be kept in the model at least for one further release. The team has to decide on a case by case basis when to remove it from the model.

- **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.

**Rules:**

One and only one lifecycle state has to be associated to every UML artifact.
It is recommended that every new UML artifact is initially annotated with the “Experimental” lifecycle stereotype.
The following state machine diagram shows the defined state transitions.
6.2.2 Reference Property

A reference can be defined for all UML artifacts. This is an optional property which contains a reference upon which the artifact is based. A reference to a standard is preferred.

The reference property is defined in the Reference stereotype and extent the Element Metaclass.
6.2.3 Example Property

This is an optional property which can be defined for all UML artifacts. It is defined as a stereotype and 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).

7 Recommended Modeling Patterns

7.1 File Naming Conventions

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

Each Model can be optionally organized into multiple submodels. Each Model or each of its constituent submodels is further divided – at the bottom level of the hierarchy – into packages containing associations, diagrams, imports, object classes, rules and type definitions. Submodels 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.
Each model is then divided at the bottom level into a set of pre-defined packages. Note: Not all pre-defined packages need to be established in a particular model instance. Additional packages can be added when needed. Figure 7.3 shows the pre-defined packages at the bottom level of the CoreNetworkModel.

7.3 Flexible Attribute Assignment to Classes

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

For case (a), attributes are grouped in classes called “Pacs” and are associated to the base 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.

→ 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” classes and are associated to the base 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 instance (created on the basis of the corresponding 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 instance from the LTP afterwards without impacting the life cycle of the base LTP instance.

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

Figure 7.4: Flexible Attribute Assignment to Classes

7.4 Use of Conditional Packages

Conditional packages are used to enhance (core) classes / interfaces with additional attributes / operations on a conditional basis. The attributes / operations are defined in special classes called packages.
Package names follow the same rules as defined for 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.
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, OpenModelClass, InformationObjectClass» named SubstituteObjectClass. This scenario indicates that only one of the three classes named Alternative1ObjectClass, Alternative2ObjectClass, Alternative3ObjectClass shall be realized.

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

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

7.6 Proxy Class Modeling

There are cases where an attribute or parameter may contain different kinds of classes. This would require an attribute/parameter per kind of class. In order to reduce the number of attributes/parameters it is recommended to define a proxy class and let a single attribute/parameter point to this class. The different kinds of classes must be inherited from the proxy class. All real subclasses inheriting from the abstract superclass (proxy) must have the same object key.
7.7 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.
7.7.1 Using Colors

tba

7.7.2 Style Sheets

Classes in class diagrams should not show the "nestedclassifiers" and "operations" compartments. Data Types in class diagrams should not show the "operations" compartment.

The following style sheet fulfill these requirements:

![CompartmentRestrictions.css](image)

Attributes in class diagrams should only show name, type, multiplicity and defaultValue. Attributes in class diagrams should not show the stereotype “OpenModelAttribute”.

The following style sheet fulfill these requirements:

![NoStereotypesDiagram.css](image)

8 Main Changes between Releases

8.1 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.

8.2 Summary of main changes between version 1.1 and 1.2

- Document moved to Open Source SDN
- Using UML Version 2.5 as basis.
• Further properties added to OpenModelAttribute stereotype:
  o partOfObjectKey
  o bitLength
  o unsigned
  o encoding
  o counter.
• Table 5.2 on attribute property dependencies added.
• Sections on Use Cases (5.10), Activities (5.11) and State Machines (5.12) added.
• Section 7.7.2 on style sheets for class diagrams added.
• Section 7.6 on proxy class modeling added.
• Element metaclass extended by an optional reference stereotype.