

TELECOMMUNICATION
STANDARDIZATION SECTORTD 406 Rev.1 (PLEN/15)

STUDY PERIOD 2013-2016

English only

Original: English

Question(s): 14/15

22 June - 3 July 2015

TD

Source: Editor G.7711/Y.1702

Title: Draft new Recommendation ITU-T G.7711/Y.1702 (ex G.gim) (for Consent, 3 July 2015)

This document contains the base text of new draft Recommendation G.7711 (ex. G.gim) “*Generic Protocol-Neutral Information Model for Transport Resources*” for consent

Document history:

Version	Date	Description
0.01	WD17r1 (9/2014 Shanghai)	Initial version. The proposal text in WD17 was accepted as the initial base text of G.gim, with the following updates from the discussion of WD18, WD05r1 (synchronization management), and WD12 (ECC management).
0.02	TD281/3 (11/2014)	<ul style="list-style-type: none"> • Update per the decision on C862, including moving the UML guideline to Annex A • Update the UML guideline so that it is not SDO-specific for the objective of an industry-wide Open Model (and companion Open Profile) • Provide the Papyrus file and data dictionary file of the model

Contact: Hing-Kam LAM
Alcatel-Lucent
USA

Tel: +1-732-331-3476
Email: kam.lam@alcatel-lucent.com

Contact: YUN Xiang
FiberHome
P.R.China

Tel: +86 27 59100125
Email: yunxig@fiberhome.com.cn

Contact: Scott Mansfield
Ericsson
USA

Tel: +1-724-931-9316
Email: scott.mansfield@ericsson.com

Attention: This is not a publication made available to the public, but an **internal ITU-T Document** intended only for use by the Member States of ITU, by ITU-T Sector Members and Associates, and their respective staff and collaborators in their ITU related work. It shall not be made available to, and used by, any other persons or entities without the prior written consent of ITU-T.

Version	Date	Description
0.03	WD31 (3/2015) Bundang	Updates per proposal in WD12 (ALU) <ul style="list-style-type: none"> • In Clause 1.1, update Figure 1-2 and add new sub-clause 1.1.4 • Describe Clauses 7.2 through 7.6 as various aspects of the Core Network Module • Update Annex A to align with xxx2015.018.01 • Accept all diff-marks To do: <ul style="list-style-type: none"> • Align the model files with xxx2014.408.03 and onf2015.074.03 • Align the profile with the Open Model Profile • Update for discussion on WD16 (ZTE) and WD21 (ZTE)
0.04	WD31r1 (3/3015) Bundang	Updates <ul style="list-style-type: none"> • Move the content of clause 7 into clause 6 (Overview) as the details is actually in clause 8 and use clause 7 as a place holder section for future areas • Align the merged clause 6 with onf2015.074.03 clause 7 • Update the new clause 7 for the Scheduling model, Synchronization management model, Logging model, Notification model, based on WD10r1 (CMCC), WD16 (ZTE), and WD21 (ZTE)
0.05 <u>(=1.0)</u>	TD406/P (7/2015) <u>TD406R1/P</u>	<u>Updates:</u> <ul style="list-style-type: none"> • <u>Further align with onf2015.074.06 ONF TR-512</u> • <u>Update Recommendation number to G.7711.</u> • <u>Updated per group review (shown with diff-marks)</u> • <u>Updated the data dictionary</u> • <u>Update the bibliography</u>

Draft new Recommendation ITU-T G.7711/Y.1702

Generic protocol-neutral information model for transport resources

Summary

Recommendation ITU-T G.7711/Y.1702 specifies a core information model of transport resources. The information model is applicable for the management and control of the transport network regardless of whether the transport networks utilize traditional OSS management, ASON control plane, or SDN controller to configure transport connectivity. The model is also applicable [to the management and control of the transport network] regardless of the technology of the underlying transport network. Furthermore, the applicability of the information model is independent upon the ultimate protocols that will be used in the management and control interfaces.

Keywords

<Optional>

Introduction

<Optional - This clause should appear only if it contains information different from Scope and Summary>

Draft new Recommendation ITU-T G.7711/Y.1702

Generic Protocol-Neutral Information Model for Transport Resources

1 Scope

An information model describes the things in a domain in terms of objects, their properties (represented as attributes), and their relationships. This Recommendation describes a core information model of transport resources. ~~This~~ information model is intended to be applicable for to the management and control of the transport network regardless of whether the transport networks utilize traditional OSS management [ITU-T G.7710], ASON control plane [ITU-T G.8080] or SDN controller to configure transport connectivity. The model is also intended to be applicable to the management and control of the transport network regardless of the transport technology of the underlying transport network. Furthermore, the applicability of the information model is independent upon the ultimate protocols that will be used in the management and control interfaces.

The core information model defined in this Recommendation can be used as the base for the extension of transport/control-technology-specific information models. Such extension will be specified in the technology-specific Recommendations, such as shown in Figure 1-1 below [ITU-T G.874.1] for OTN management, [ITU-T G.8052] for Carrier Ethernet management, ~~[ITU-T G.8152]~~ for MPLS-TP management, and [ITU-T G.7718.1] for ASON control management.

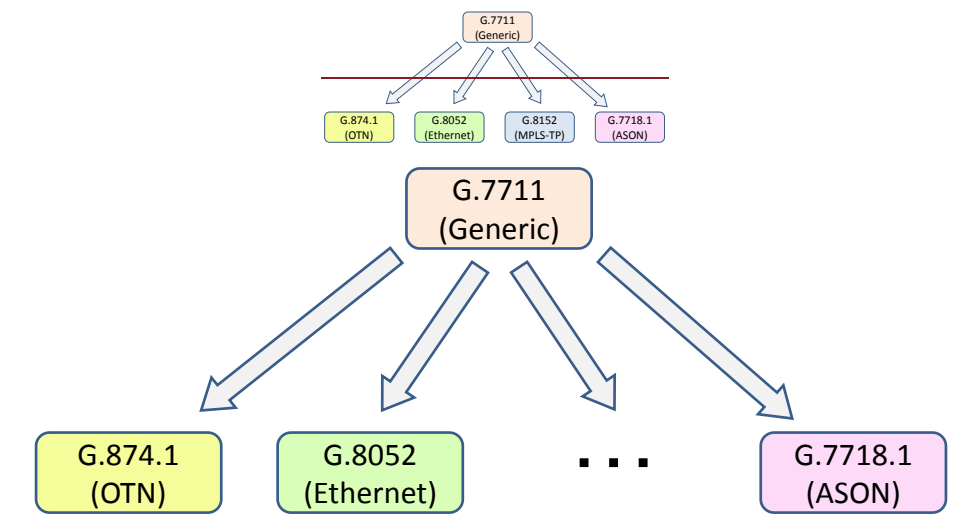


Figure 1-1 Example Information Model Extension

A uniform protocol-neutral core information model for traditional management, ASON control, and SDN control will ensure consistent OAM&P of the transport network. This will benefit the network operators and system/equipment vendors for enabling interoperability between SDN-controlled and traditionally-managed network domains and future migration from traditional management to SDN control.

Furthermore, it is essential that the information model be applicable to complex network elements (NEs) that may be deployed in current networks, which requires support of more than a simple nodal view. Examples of such NEs include:

- Multi-layer NEs with subnetworks at each layer with transitional links between the subnetworks:
- NEs that have their matrix partitioned (e.g. to model multiple ~~BLSR-MSPRING~~ terminations or to model connectivity restrictions) with "internal" links between the subnetworks.
- Distributed NEs (e.g., a PON) with a mediation function to allow management visibility of each of the "encapsulated" NEs

The complexity of these NEs makes it difficult to distinguish between the NE view and what is traditionally called the network view. The core information model thus encompasses both nodal view and network view of the transport resources.

1.1 Development and Use of the G.7711 Generic Information Model

Figure 1-2 below provides an overview of the Common Information Model (IM) and how the purpose specific IM views and data schema¹ are related to it. The term Data Schema (DS) in this document is used in the context of either (1) a specific protocol that is used to implement a purpose specific interface or (2) a programming language that is used to invoke a purpose specific application programming interface (API). Guidelines for the use of UML in the common IM, pruning and refactoring the common IM to provide a purpose specific view, and ultimately mapping to a data schema will also be provided.

¹ The term data schema is used instead of data model since the term data model is also used in a wider context.

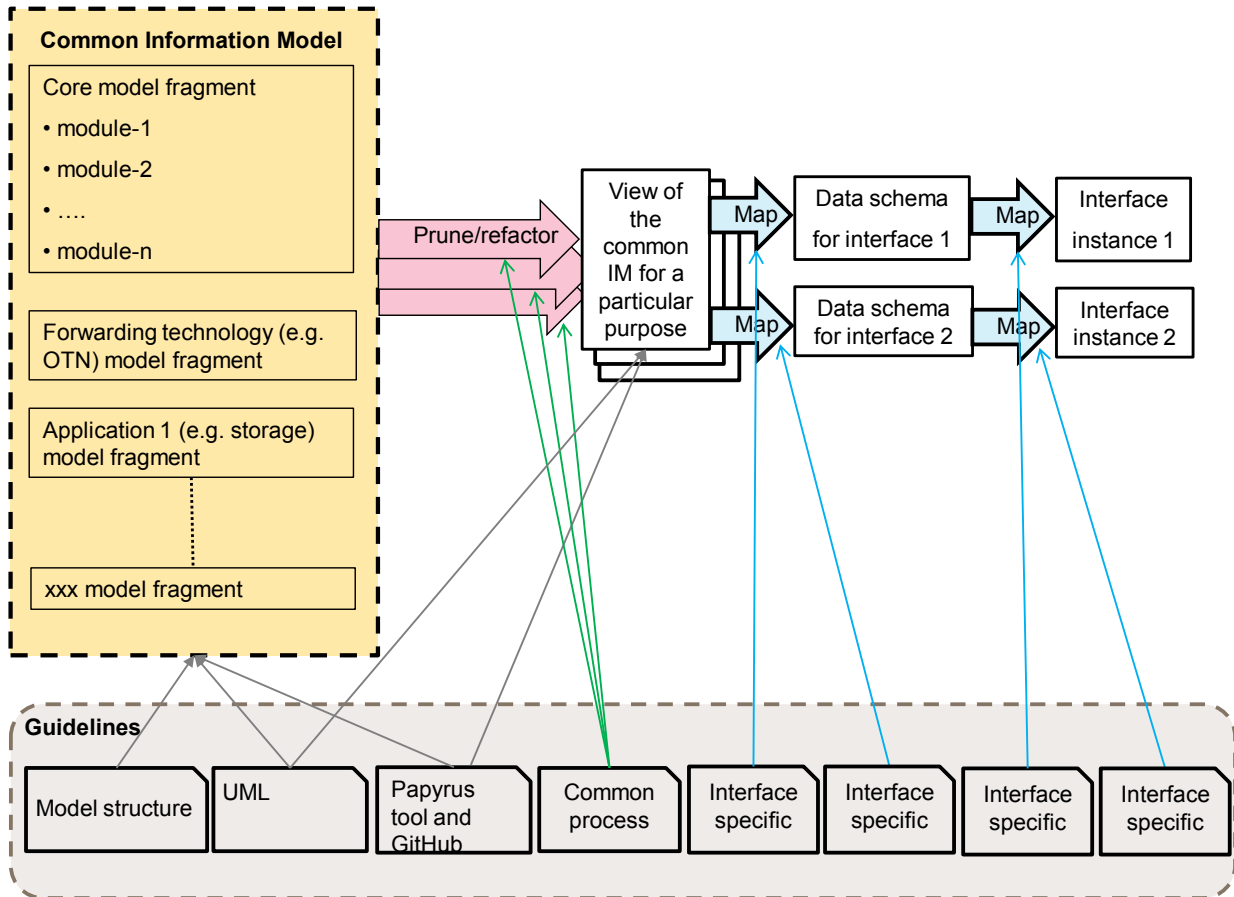


Figure 1-2 Methodology of IM and DS Development

1.1.1 Common IM

An information model describes the things in a domain in terms of objects, their properties (represented as attributes), and their relationships. The Common IM should be expressed in UML and include all of the artefacts (object classes, attributes, relationships, etc.) that are necessary to describe the generic and domains for the technologies/applications being developed.

It will be necessary to continually expand and refine the Common IM over time as new forwarding technologies, capabilities and applications are encompassed and new insights are gained.

To allow these extensions to be made in a seamless manner, the Common IM will be structured into a generic model (G.7711) and a number of models which are specific to the forwarding technologies (such as OTN in G.874.1, Ethernet in G.8052, ~~MPLS-TP in G.8152~~, etc.) and application (such as ASON control plane management in G.7718.1). This modelling process is intended to allow these extensions to be developed with as much independence as possible.

– Generic Model

The artefacts in the generic model (G.7711) will be used by the technology/application specific models either directly or with extension. The generic model will be constructed as a set of modules each addressing a specific topic to allow for easier navigation. This Recommendation is responsible for specifying and maintaining the generic model.

As a result of advancements in the industry it may be recognised that some parts of the generic model may need to be augmented or changed. This Recommendation will ensure that any such areas are clearly identified using lifecycle stereotypes. The older model forms will be maintained to ensure ongoing compatibility and to ease migration.

– Specific Models

It is expected that the transport forwarding technology or application specific domains will develop the appropriate models which contain objects, attributes and associations that relate solely to that respective domains. In some cases an application or forwarding technology addition will also require enhancement of the generic model.

In some cases an artefact in a specific model initially considered to be purely for a single forwarding technology or application may be subsequently recognised as common across several specific models and hence there will be a need to migrate this artefact to the generic model.

To assure coherency, any artefacts, attributes or associations that might be identified during the development of forwarding technology or application specific views should be included in the appropriate fragment of the common information model. Only those properties that relate to the specific encoding or style of interaction of an interface may be added outside the common information model.

1.1.2 Purpose Specific IM View

A purpose specific information model view is a subset of the Common IM and should be expressed in UML. A purpose specific information model view will typically be much smaller than the entire Common IM. If additional artefacts (objects, packages, attributes or associations) are identified while establishing a specific view, these should be added to the appropriate fragment of the Common IM so that they are available for future use.

To provide maximum reuse, a purpose specific view should be developed in two steps:

- Prune and refactor the artefacts of the Common IM to provide a model of the network to be managed. Only those artefacts that represent the capabilities that are both in scope and supported are include in the purpose specific IM.
- Define the access rights for the various groups of users that will manage that network

Pruning and refactoring provides a purpose specific IM that represents the capabilities of the network of interest. The definition of access rights provides the ability to limit the actions that can be taken by the various user groups that will use that IM. For example a user group responsible for network configuration could be provided with full read/write access and the ability to create or delete object instances; while a user group responsible for inventory may only be allowed read access (i.e. can see the network but cannot make changes).

- Pruning, i.e. remove the objects/packages/attributes that are not require:
 - Select the required object classes from the common IM
 - All mandatory (non-optional) attributes and packages must be included
 - Select the required conditional packages and optional attributes
 - Where appropriate conditional packages and optional attributes may be declared mandatory
 - Remove any optional associations that are not required
- Refactoring, i.e. reduce association flexibility:
 - Reducing multiplicity (for example from [1..*] to [1])
 - When this results in a composition association of multiplicity [1] between a subordinate and a superior object class, they can be combined into a single object class by pulling the attributes of the subordinate class into the superior class

- Where possible reducing the depth of the inheritance (i.e. by combining object classes by moving the attributes of the super class into the subclass)
- Add reverse navigation (if useful for the client).
 - The common IM only supports navigation from a subordinate object class to a superior object class. This allows new subordinate object classes to be added without any impact on the superior object class. In a purpose specific implementation it is frequently useful to be able to navigate the relationship between superior and subordinate object classes in both directions.
- Constraining attribute definitions
 - Reducing legal value ranges
 - Defining which (if any) attributes should be read only (for all users)
 - Defining constraints between attributes
- Definition of access rights:

If only one group will use the network specific IM then this step is not required. If more than one group will use the network specific IM this optional step provides a profile for each user group to:

 - Convert some attributes defined as read/write in the network specific IM to read only
 - Remove the rights to create/delete some or all object instances

1.1.3 Data Schema

A Data Schema (DS) is developed in the context of either (1) a specific protocol that is used to implement a purpose specific interface or; (2) a programming language that is used to invoke a purpose specific API. Note that it is possible to map directly from the purpose specific information model to interface encoding. The DS is constructed by mapping of the purpose specific information model into the DS together with the operations patterns from the Common IM to provide the interface protocol specific operations and notifications. The operations should include data structures taken directly from the purpose specific information model view with no further adjustment.

The development of the DS should consider the following:

- The operations should act on the information in a way consistent with the modelled object lifecycle interdependency rules. (Note: these need to be added to the core model)
 - Lifecycle dependencies to ensure sensible interface operation structuring and interface flow rule
 - Use transaction approach style of interface to account for lifecycle dependencies of the model
- The operations should abide by the attribute properties
 - Read only attributes (except those which are defined as setByCreate) should not be included in data related to creation of an object (e.g. not in createData) or in a specification of a desired object ~~structure~~-outcome
- Use of attribute value ranges, etc. to allow “effort” statement, optionality and negotiation to be supported by the interface

1.1.4 Interface Encoding

This step encodes either the purpose specific data schema or a purpose specific information model into either a specific protocol that is used to implement a purpose specific interface; or a programming language that is used to invoke a purpose specific API. If the interface is encoded directly from the purpose specific information model then the interface operations must be added as described [above in clause 1.1.3](#).

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

- [ISO/IEC 19505-1] ISO/IEC 19505-1:2012 - Information technology - Object Management Group Unified Modelling Language (OMG UML) - Part 1: Infrastructure. Iso.org.2012-04-20. Retrieved 2014-04-10, 2012
- [\[ITU-T G.800\]](#) [Recommendation ITU-T G.800 \(2012\), *Unified functional architecture of transport networks*](#)
- [ITU-T G.805] Recommendation ITU-T G.805 (2000), *Generic functional architecture of transport networks*
- [ITU-T G.808.1] Recommendation ITU-T G.808.1 (2014), *Generic protection switching - Linear trail and subnetwork protection*
- [ITU-T G.7710] Recommendation ITU-T G.7710/Y.1701 (02/2012), *Common equipment management function requirements*
- [ITU-T G.8080] Recommendation ITU-T G.8080/Y.1304 (02/2012), *Architecture for the automatically switched optical network*
- [ITU-T G.852.2] Recommendation ITU-T G.852.2 (3/1999), *Enterprise viewpoint description of transport network resource model*
- [ITU-T G.874.1] Recommendation ITU-T G.874.1 (10/2012), *Optical transport network: Protocol-neutral management information model for the network element view*, plus Amendment 1 (08/2013)
- [ITU-T G.8052] Recommendation ITU-T G.8052/Y.1346 (08/2013), *Protocol-neutral management information model for the Ethernet Transport capable network element*
- ~~[ITU-T G.8152] Recommendation ITU-T G.8152/Y.1375 (draft), *Protocol-neutral management information model for the MPLS-TP network element*~~
- [ITU-T G.7718.1] Recommendation ITU-T G.7718.1/Y.1709.1 (12/2006), *Protocol-neutral management information model for the control plane view*
- [\[ITU-T M.3100\]](#) [ITU-T Recommendation M.3100 \(04/2005\), *Generic network information model*](#)

- [ITU-T M.3160] ITU-T Recommendation M.3160 (11/2008), *Generic Management Information Model – Protocol Neutral*
- [ITU-T X.1036] ITU-T Recommendation X.1036 (11/2007), *Framework for creation, storage, distribution and enforcement of policies for network security*
- ~~[TMF TR225] TM Forum TR225, *Logical Resource: Network Function Model*~~

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

~~3.1.1 <Term 1> [Reference]: <optional quoted definition>~~

~~3.1.2 <Term 2> [Reference]: <optional quoted definition>~~

3.1.1 access point [ITU-T G.805]

3.1.2 connection termination point [ITU-T M.3100]

3.1.3 connection point [ITU-T G.805]

3.1.4 information model [ITU-T X.1036]

3.1.5 link [ITU-T G.805]

3.1.6 link connection [ITU-T G.805]

3.1.7 matrix [ITU-T G.805]

3.1.8 port [ITU-T G.805]

3.1.9 subnetwork [ITU-T G.805]

3.1.10 subnetwork connection [ITU-T G.805]

3.1.11 trail termination [ITU-T G.805]

3.1.12 termination connection point [ITU-T M.3100]

3.1.13 termination point [ITU-T M.3100]

3.1.14 trail termination [ITU-T G.805]

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

~~3.2.1 **Information Model:** An information model describes the things in a domain in terms of objects, their properties (represented as attributes), and their relationships.~~

~~3.2.2 **Data Model:** TBD~~

~~3.2.3 **Data Schema:** TBD~~

None

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

AP	Access Point
API	Application Programming Interface
ASON	Automatically Switched Optical Network
CFM	Core Foundation Module
CNM	Core Network Module
CP	Connection Point
CTP	Connection Termination Point
EMS	Element Management System
EP	End Point (of FC)
ETH	Ethernet MAC Layer
ETY	Ethernet Physical Layer
FC	Forwarding Construct
FD	Forwarding Domain
FDFr	Forwarding Domain Fragment
FTP	Floating Termination Point
GIM	Generic Information Model
GUID	Globally Unique Identifier
IM	Information Model
IMP	Inverse Multiplexing
LE	Link End
LP	Layer Protocol
LTP	Logical Termination Point
MAC	Media Access Control
MEP	Maintenance Entity Group End Point
<u>MSPRING</u>	<u>Multiplex Section Share Protection Ring</u>
NCD	Network Control Domain
NE	Network Element
OAM	Operation, Administration, and Maintenance
OAM&P	Operation, Administration, Maintenance, and Provisioning
OSS	Operation Support System
OTN	Optical Transport Network
PON	Passive Optical Network
SDN	Software Defined Networking
TCP	Termination Connection Point
TP	Termination Point
TTP	Trail Termination Point
UML	Unified Modelling Language
VNE	Virtual Network Element
XC	Cross Connection

The following abbreviations are for the name of the model constructs, such as object class. These names follow the UML naming convention, i.e., UpperCamelCase.

<u>EP</u>	<u>EndPoint (of FC)</u>
<u>FC</u>	<u>ForwardingConstruct</u>
<u>FD</u>	<u>ForwardingDomain</u>
<u>LE</u>	<u>LinkEnd</u>
<u>LP</u>	<u>LayerProtocol</u>
<u>LTP</u>	<u>LogicalTerminationPoint</u>
<u>NCD</u>	<u>NetworkControlDomain</u>
<u>NE</u>	<u>NetworkElement</u>

5 Conventions

5.1 UML modelling conventions

The information model defined in this Recommendation is expressed in a formal language called UML (Unified Modelling Language), which was developed by the Object Management Group (OMG). It is a general-purpose modelling language in the field of software engineering. In 2000 the Unified Modelling Language was also accepted by the International Organization for Standardization (ISO) as an approved ISO standard [ISO/IEC 19505-1].

UML defines a number of basic model elements, called UML artefacts. In order to assure consistent modelling, only a selected subset of these artefacts is used in the development of the G.7711 information model. The selected subset of UML artefacts is documented in Annex A “UML Model Guidelines” of this Recommendation.

5.2 Model Artefact Lifecycle Stereotypes conventions

Stereotypes are applied to entities in the model to indicate the degree of maturity. These are made visible in many of the figures.

The following stereotypes appear in this document:

- «experimental»

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.

- «preliminary»

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.

If no stereotype is shown the entity is mature. There are other lifecycle stereotypes that are not relevant in this document. See Clause A.5.2 for more details.

5.3 Forwarding entity terminology conventions

In this Recommendation, the terms Forwarding Domain (FD) and Forwarding Construct (FC) have been used in place of the traditional terms SubNetwork (SN) and SubNetwork-Connection (SNC) respectively.

5.4 Conditional package conventions

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. In this Recommendation, package names follow the same rules as defined for object classes. The name ends with the suffix "_Pac".

5.5 Pictorial diagram conventions

The symbols highlighted below are used in this document in pictorial representation of the model.

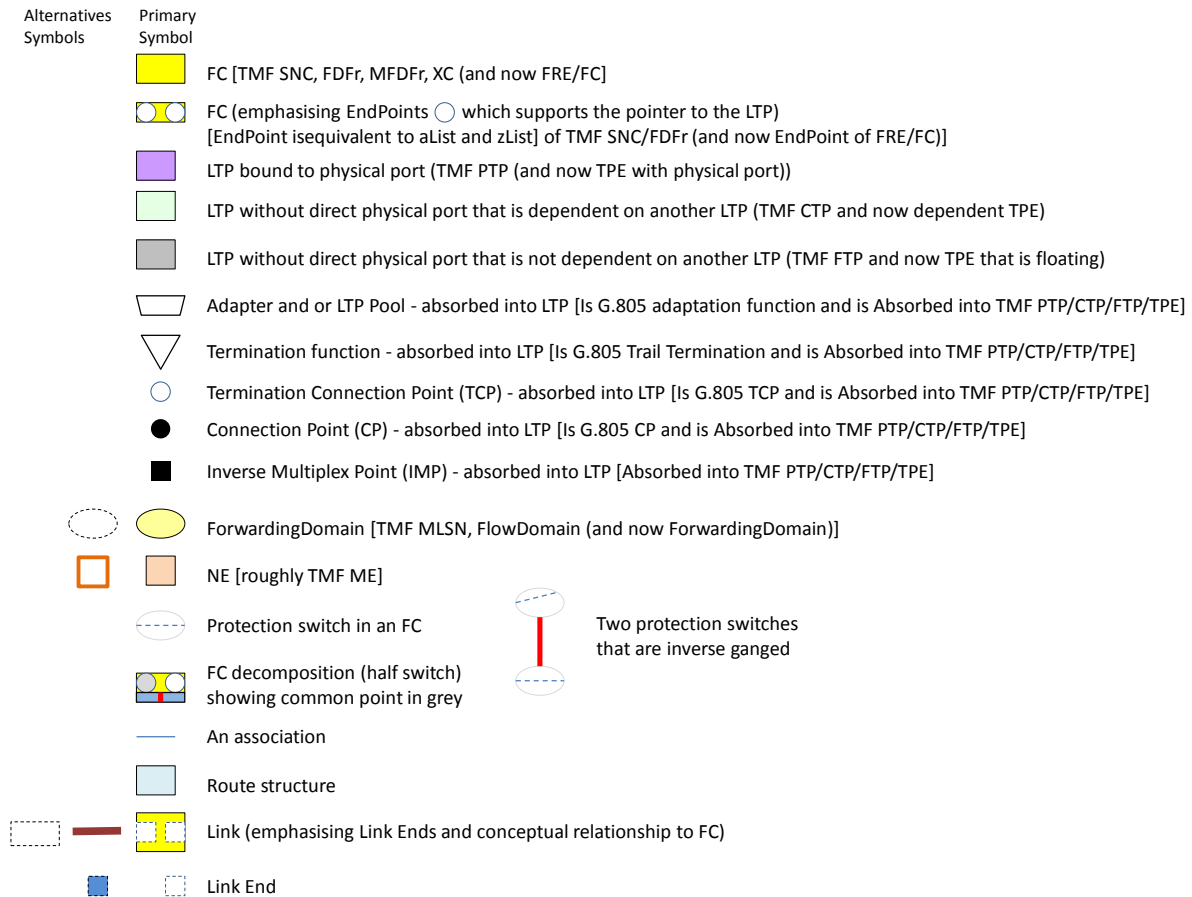


Figure 5-1 Illustrative Keys

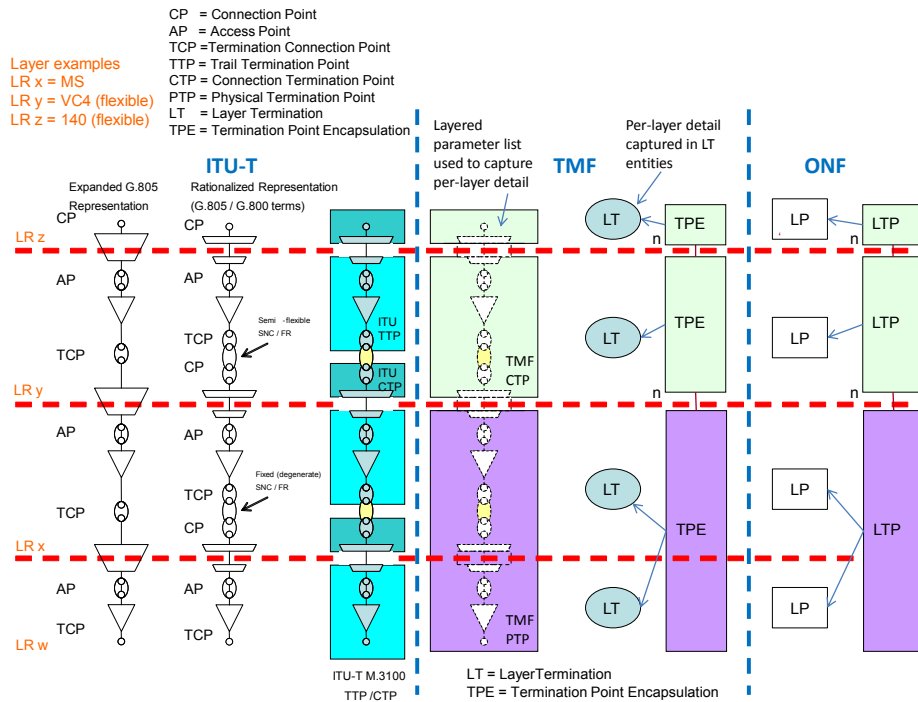


Figure 5-2 Mapping from ITU-T and TM Forum Termination models to the Generic Information Model²

6 Overview of the Generic Information Model (GIM)

The focus of this Recommendation is the generic information model (GIM). Application- or forwarding technology-specific information models are out of scope of this Recommendation. The GIM is divided into several packages (including modules and subsets). In the following sub-clauses GIM is discussed from several different perspectives:

- CoreFoundationModule (CFM):

Covers aspects common to all classes such as identifiers and naming.

- CoreNetworkModule (CNM):

Covers the essentials for modelling of the Network providing an overview of the key classes.

- Termination Subset of the CNM:

Covers the modelling of the processing of transport characteristic information, such as termination, adaptation, OAM, etc.

Note that technology specific details are covered in the forwarding technology model (this aspect is not in the scope of this document)

- Forwarding ~~Subset~~ Subset of the CNM:

² It should be noted that in the Generic Information Model the terms ForwardingDomain (FD) and ForwardingConstruct (FC) are used in place of SubNetworkConnection (SNC) and SubNetwork (SN) (respectively used in the technology-specific information models.)

Covers the details of forwarding entities, including:

- The Basic Forwarding
- The ForwardingConstructSpec
- Topology Subset of the CNM:

Covers the modelling of network topology information in detail and describes the attributes relevant when working with network topology. ³

This clause provides visual views, using the UML class diagrams for the object classes of the information model. These diagrams depict a subset of the relationships among the object classes, such as inheritance (i.e. specialization), association relationships (such as aggregation and composition), and conditional features or capabilities. Some diagrams also show further details of the individual object classes, such as the attributes of the object classes, and the data types used by the attributes. In addition to the class diagrams there are also pictorial representation of stylized network fragments to assist in the understanding of the model.

6.1 Overview of the Core Network Module (CNM)

This clause provides a high-level overview of the generic information model. Figure 6-1 below is a skeleton class diagram illustrating the key object classes defined in the CoreNetworkModule of the CoreModel.

³ The information described in this subset can be used for example for path computation and to provide views of network capacity/capability with information maintained in a topology database.

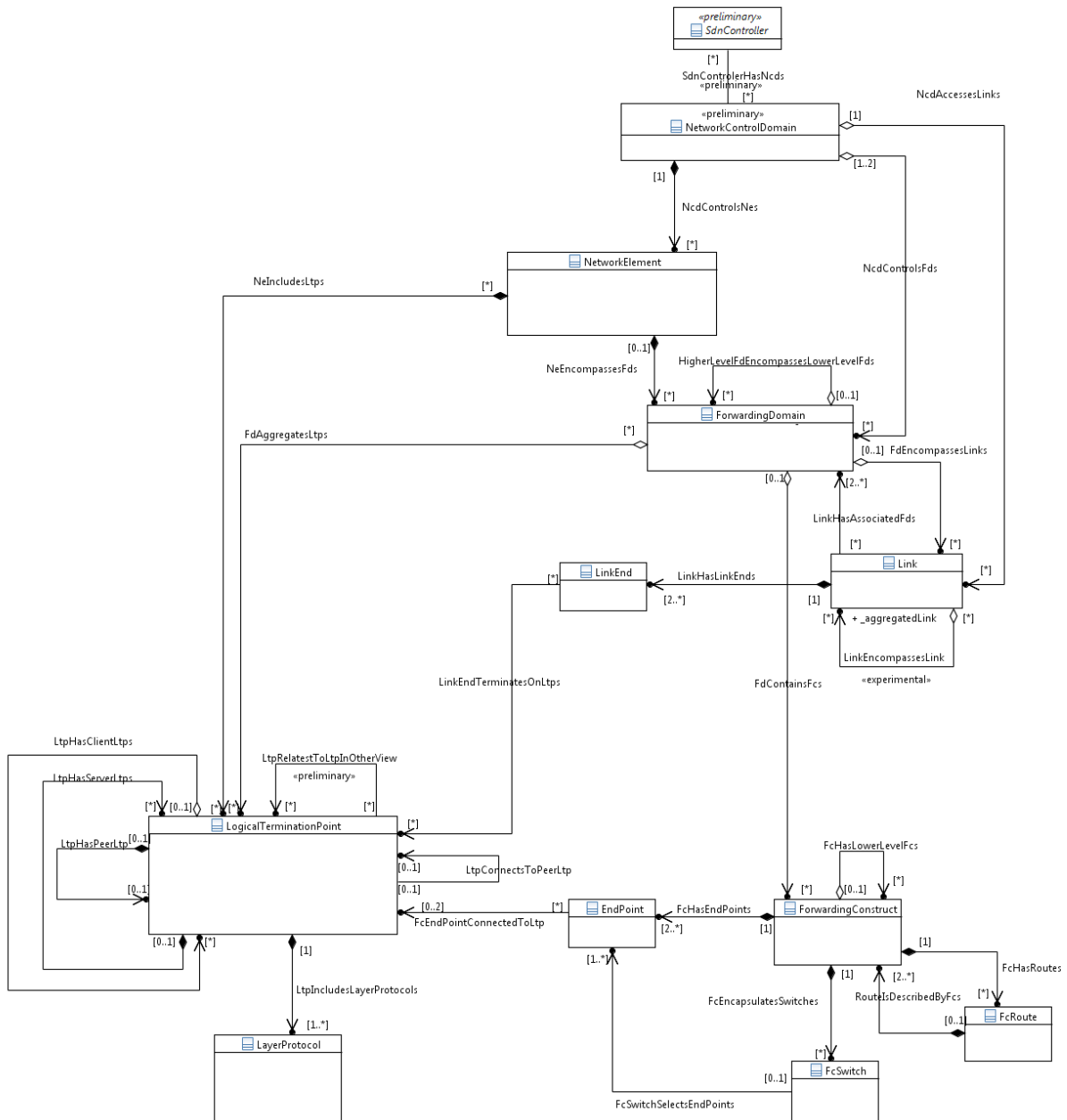


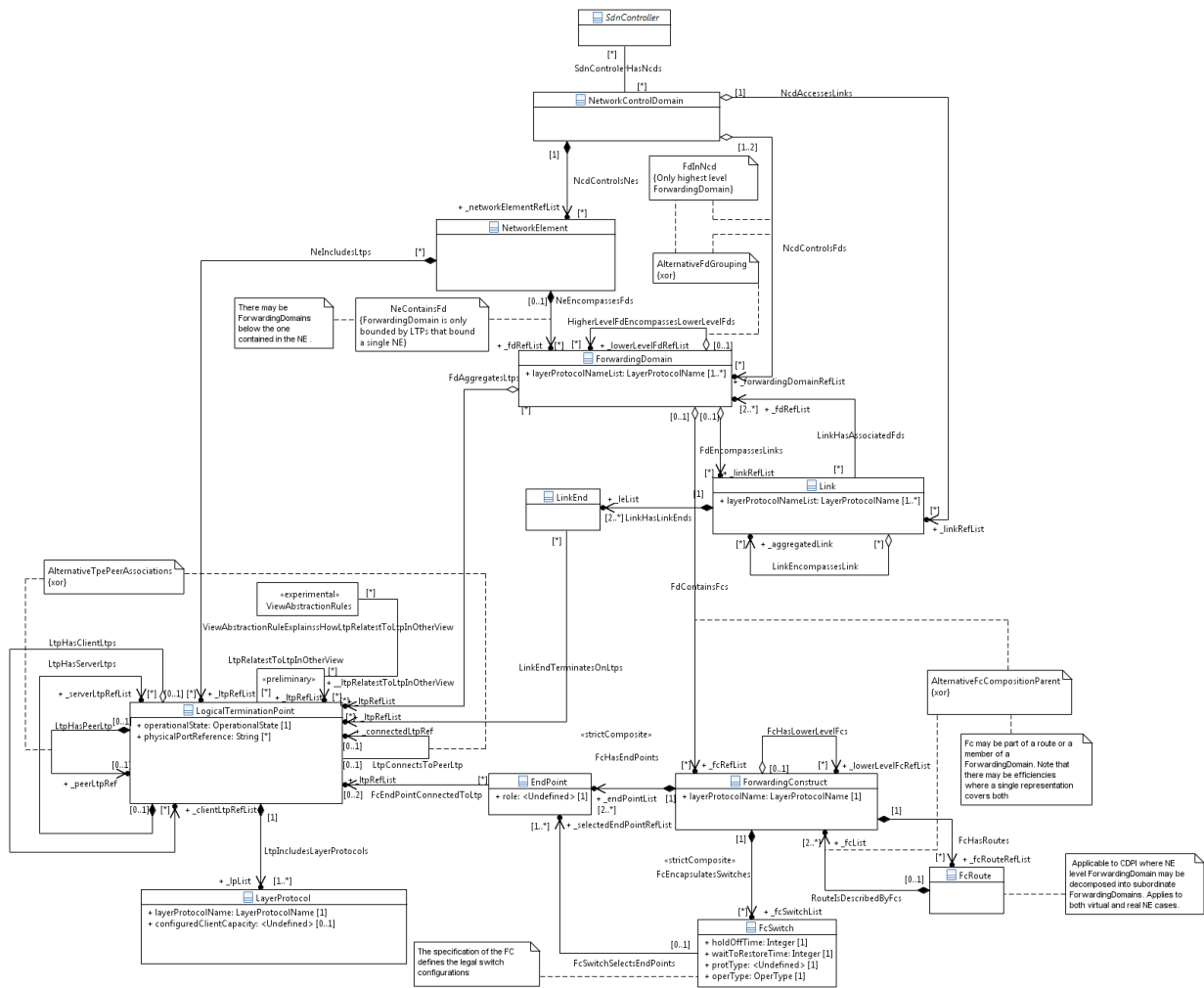
Figure 6-1 Skeleton Class Diagram of key object classes

Note: A more convenient way to view the details of the diagram is by using the companion PNG file, which can be independently zoomed in and out without impacting the viewing of the main document.

When applying the information model to a specific-purpose interface, not all of the information model artefacts need to be considered. That is, only a subset of the overall information model may be needed. Depending on the scope of the interface, pruning of the information model may be necessary, such as excluding a whole object class or part of an object class. For example, in the

interface from an SDN controller directly to an NE in the infrastructure layer, the NetworkControlDomain object class is beyond the scope of the NE⁴ interface and thus not needed. In addition, re-factoring of the selected model artefacts may be necessary to meet the specific-purpose needs. However, re-factoring of the model artefacts should not add semantics beyond those defined in the information model.

Figure 6-2 below is a more comprehensive class diagram than the previous one showing some attributes and constraints of the associations in the mode.




 HighLevelDetail.PNG

Figure 6-2 Class Diagram of all key object classes showing attributes and constraints

⁴ The NE scope of the direct interface from a SDN controller to an NE in the infrastructure layer is similar to the EMS-to-NE management interface defined in the ITU-T Q14/15 information models [ITU-T G.874.1] (OTN), [ITU-T G.8052] (Ethernet), and draft [ITU-T G.8152] (MPLS-TP). On the other hand, the network scope of the interface between two SDN controllers is similar to the OS-to-OS management interface defined in the TMF 612 information model.

6.1.1 LogicalTerminationPoint (LTP) and LayerProtocol (LP)

The LogicalTerminationPoint (LTP) object class encapsulates the termination, adaptation and OAM functions of one or more transport layers. The structure of LTP supports all transport protocols including circuit and packet forms. Each transport layer is represented by a LayerProtocol (LP) instance. The LayerProtocol instances of the LTP can be used for controlling termination and OAM functionality of that layer. It can also be used for controlling the adaptation (i.e. encapsulation and/or multiplexing of client signal). Where the client – server relationship is fixed 1:1 and immutable, the different layers can be encapsulated in a single LTP instance. Where there is a n:1 relationship between client and server, the layers must be split over separate instances of LTP. Rules for forming LTP instances are provided in clause 6.3 Termination Subset of this document.

The LP object class is defined with generic attributes “layerProtocolName” for indicating the supported transport layer protocol.

Transport layer specific properties (such as layer-specific termination and adaptation properties) are modeled as attributes of conditional packages (called “Pacs” in the UML notation of the information model) associated with the LP object class⁵. Transport layers supported by the information model include:

- Layer 0 (L0): OPS, OTS, OMS, OCh
- Layer 1 (L1): OTU, ODU
- Layer 2 (L2): Carrier Grade Ethernet (ETY, ETH), MPLS-TP (MT)

In the OT information model, the conditional packages for supporting layer-specific termination functions and adaptation functions have been derived from the various ConnectionTerminationPoint (CTP) and TrailTerminationPoint (TTP) object classes defined in the NE-view information models, including the ODUk TTP and ODUk CTP object classes in G.874.1 and the ETH TTP and ETH CTP object classes in G.8052.

Functions that can be associated/disassociated to/from an LTP instance (and ForwardingConstruct), such as OAM, protection switching, and performance monitoring are modelled as secondary object classes.

6.1.2 ForwardingDomain (FD)

The ForwardingDomain (FD) object class models the topological component which represents the opportunity to enable forwarding between points represented by the LTP in the model. The LTPs available are those defined at the boundary of the FD (see “subnetwork” topological components in G.852.2 and TMF 612). The FD object can hold zero or more instances of ForwardingConstruct (FC) of one or more layer networks; e.g., OCh, ODU, ETH, and MPLS. The FD object provides the context for instructing the formation, adjustment and removal of FCs.

The FD object class supports a recursive aggregation relationship (HigherLevelFdEncompassesLowerLevelFds) such that the internal construction of an FD can be

⁵ Note that some implementation languages may allow the addition and removal of _Pacs and attributes from instances of a running system and others may be restricted such that _Pacs/attributes cannot be added/removed once an instance has been created. The model supports both modes of operation.

exposed as multiple lower level FDs and associated Links (partitioning)⁶ (See Figure 6-14 in clause 6.5.1). An FD within a NetworkElement represents a switch matrix/ fabric)⁷. Note that a NetworkElement can encompass multiple matrixes (represented by FDs): see Figure 6-15 in clause 6.5.1. An instance of FD is associated with zero or more LTP objects via the FdAggregatesLtps aggregation.

6.1.3 ForwardingConstruct (FC)

The ForwardingConstruct (FC) object class models enabled potential for forwarding between two or more LTPs, and like the LTP, supports any transport protocol including all circuit and packet forms. The association of the FC to LTPs is made via EndPoints (essentially the ports of the FC) where each EndPoint (EP) of the FC has a role in the context of the FC. The traffic forwarding between the associated EPs of the FC depends upon the type of FC and may be associated with FcSwitch object instances.

An FC can be in only one FD. An FC object supports a recursive aggregation relationship such that the internal construction of an FC can be exposed as multiple lower level FC objects (partitioning)⁸. An FC object can have zero or more routes, each of which is defined as a list of lower level FC objects (with the implicit understanding that link connections are interleaved between the lower level FCs)⁹. At the lowest level of recursion, a FC represents a cross-connection within a NetworkElement¹⁰. Thus the route of an FC may represent the cross-connections in a NetworkElement.

If an FC provides protection, the FC will have one or more associated FcSwitch objects as described in clause 6.1.6 FcSwitch that have protection configuration related attributes.

The FC object can be used to represent many different structures including point-to-point (P2P), point-to-multipoint (P2MP), rooted-multipoint (RMP) and multipoint-to-multipoint (MP2MP) bridge and selector structure for linear, ring or mesh protection schemes.

6.1.4 FcRoute

The FcRoute object class models the individual routes of an FC. The route of an FC object is represented by a list of FCs at a lower level. Note that depending on the service supported by an FC, the FC can have multiple routes.

⁶ The model actually represents aggregation of lower level FDs into higher level FDs as views rather than FD partition, and supports multiple views. This allow reallocation of capacity from lower level FDs to different higher level FDs as if the network is reorganized (as the association is aggregation not composition).

⁷ There are cases where the matrix is itself decomposed so the FD may be smaller than the scope of the matrix.

⁸ The LinkConnections associated with the Links that are exposed as part of the internal structure of the FD are not modelled at this point.

⁹ The route is an alternative view of the internal structure of the FC. There are cases where a route is the most appropriate representation and cases where the aggregation is the most appropriate form.

¹⁰ Recognizing that as for it may be necessary to decompose the matrix into smaller FDs so it may be necessary to decompose the XC into smaller FCs (this is true at a control domain boundary for example).

6.1.5 EndPoint (EP)

The EndPoint (EP) object class models the access to the FC function. Each EP instance has a role (e.g., working, protection, protected, hub, spoke, leaf, root, etc.) with respect to the FC function.

The EP replaces the Protection Unit of a traditional protection model. It represents a protected (resilient/reliable) point or a protecting (unreliable working or protection) point.

An EP may be associated with a LTP.

6.1.6 FcSwitch

The FcSwitch object class models the switched forwarding of traffic (traffic flow) between EPs and is present where there is protection functionality in the FC. It plays the role of an aspect of the Protection Group of the traditional information model. When supporting the protection function, an FcSwitch object instance associates two or more EPs, each of which is playing the role of a Protection Unit.

It is possible for one or more protection EPs (standby/backup) to provide protection for one or more working (i.e., regular/main/preferred) EPs where either the protection or working EPs can feed one or more protected EPs. The protection system may operate in revertive or non-revertive (symmetric) mode. The waitToRestore attribute defines the revertive timer for the revertive mode. The protection function of the FcSwitch may operate in one of several modes including source switched, destination switched, source and destination switched, etc. (covering cases such as 1+1 and 1:1). It may be lockout (prevented from switching), force switched or manual switched. It will indicate switch state and notify change of state.

A specific instance of FC may not contain any FcSwitch instances when there is no protection capability, and may contain many FcSwitch instances in complex cases. The arrangement of switches for a particular instance is described by a referenced FcSpec¹¹ (see [Error! Reference source not found.6.4.2 Error! Reference source not found.Forwarding Construct Specification and other details of Forwarding](#)). The approach supports all forms of protection described in [ITU-T 808.1].

6.1.7 Link and LinkEnd

The Link object class models effective adjacency between two or more¹² ForwardingDomains (FD).¹³ In its basic form (i.e., point-to-point Link) it associates a set of LTP clients on one FD with an equivalent set of LTP clients on another FD. Like the FC, the Link has endpoints (LinkEnd) which take roles relevant to the constraints on flows offered by the Link (e.g., Root role or leaf role for a Link that has a constrained Tree configuration). A Link may offer parameters such as capacity and delay (see clause [Error! Reference source not found.6.5.3 Error! Reference source not found.Detailed properties of Topology](#)).- These parameters depend on the type of technology that supports the link. An FD may aggregate Links: see Figure 6-14 in clause 6.5.1. The

¹¹ Many instances of FC may reference the same FcSpec.

¹² At this point the model supports point to point links fully. The model allows multi-point but anything above 2 ~~is essentially~~ (i.e., 3..*) is preliminary

¹³ The model supports an experimental attribute, offNetworkAddress, in the LinkEnd to cover cases where the FD that the Link ends on is outside the visibility (and hence off network).

FdEncompassesLinks association is modeled to collect links that are wholly within the bounds of the FD.^{14 15}

The Link can support multiple transport layers via the associated LTP object. Instance of Link can be formed with the necessary properties according to the degree of virtualization. For implementation optimization, where appropriate, multiple layer-specific links can be merged and represented as a single Link instance as the Link can represent a list of layer protocols (identified via the layerProtocolNameList attribute).

6.1.8 NetworkElement

The NetworkElement object class represents a Network Element in the data plane or a virtual network element visible in the interface where virtualization is needed.

In the direct interface from an SDN controller to a Network Element in the data plane, the NetworkElement object defines the scope of control for the resources within the network element, e.g., internal transfer of user information between the external terminations (ports), encapsulation, multiplexing/demultiplexing, and OAM functions, etc. The NetworkElement provides the scope of the naming space for identifying objects representing the resources within the Network Element.

The NeEncompassesFd association occurs for FDs that are within the bounds of the NetworkElement definition such that the FD is bounded by LTPs, all of which are on the boundary of the NetworkElement or are within the NetworkElement¹⁶.

Where virtualization is employed, the NetworkElement object represents a Virtual Network Element (VNE). The mapping of the VNE to the Network Elements is the internal matter of the SDN controller that offers the view of the VNE. Via the CPI interface between hierarchical SDN controllers, NetworkElement instances can be created (or deleted) for providing (or removing) virtual views of the combination of slices of network elements in the data plane.

6.1.9 NetworkControlDomain (NCD)

The NetworkControlDomain (NCD) object class represents the scope of control that a particular SDN controller has with respect to a particular network, i.e., encompassing a designated set of interconnected (virtual) network elements.

In the interfaces between SDN controllers where virtualization is necessary, e.g., in client/server SDN controller relationship, the NCD object defines the scope of control of the client SDN controller on the virtual network that has been provided by the server SDN controller (i.e., the scope of control relates to the partitioned provider resources allocated to that particular client). The NCD provides the scope of naming space for identifying objects representing the virtual resources within the virtual network.

¹⁴ This association can also be inferred from the higherLevelFdEncompassesLowerLevelFd association together with the linkHasAssociatedFds association. Note that Link decomposition can also be represented using the LinkEncompassesLink association (similar to the FdEncompassesFd usage for the ForwardingDomain (this association is experimental).

¹⁵ A Link with an Off-network end cannot be encompassed by an FD.

¹⁶ Where an FD is referenced by the NeEncompassesFd association, any FDs that it encompasses (i.e., that are associated with it by HigherLevelFdEncompassesLowerLevelFds), must also be encompassed by the NE and hence must have the NeEncompassesFd association.

6.2 Core Foundation Module (CFM)

To communicate about a thing it is important to have some way of referring to that thing, i.e., to have some reference. Terms such as name and identifier are often used when describing the reference. Unfortunately these terms in general usage have ambiguity in their definition that leads to erroneous system behaviour. With the aim to ensure that the controller system behaviour is not erroneous, the model will adopt the following (hopefully suitably rigorous) principal definitions:-

- Entity: A thing with an identity, defined boundary, properties, functionality and life
 - Examples: A circuit pack, an LTP
- Feature of an Entity: A thing that is an inseparable part of an entity but is a distinct surface characteristic of the entity.
 - Examples: A pin on an integrated circuit, the endpoint on a FC, a face of a cube, the handle of a cup.
 - Note that this is important from a modeling perspective as the representation appears similar to that of an Entity
- Object Class: The representation of a thing that may be an entity or an inseparable “Feature of an Entity”.
- Role: A specific structure of responsibilities, knowledge, skills, and attitudes in the context of some activity or greater structure. The role has an identity and an identifier
- Identifier: A property of an entity/role with a value that is unique within an identifier space, where the identifier space is itself globally unique, and immutable. An identifier carries no semantics with respect to the purpose of the entity.
- Globally Unique Identifier (GUID): An identifier that is globally unique.
- Local ID: An identifier that is unique in the context of some scope that is less than the global scope.
- Name: A property of an entity with a value that is unique in some namespace but may change during the life of the entity. A name carries no semantics with respect to the purpose of the entity.
- Label: A property of an entity with a value that is not expected to be unique and is allowed to change. A label carries no semantics with respect to the purpose of the entity and has no effect on the entity behaviour or state.
 - A label can be used to carry a freeform text string for any operator purpose. The contents of a label in one view may happen to be the value of a name or identifier in another view. From the perspective of the view with the label there is no expectation other than the value is a string.
- Address: A structure of named values¹⁷ in some address space that defines a location (a volume in that address space) where the structure is a nested hierarchy
 - A named value may be a name or identifier the name of the value may be a name or identifier.
- Route: the way (via specified intermediate locations and paths) to get to one location from another
- Property: A quality associated with a thing, structure or location.
- Semantics: Meaning.

¹⁷ A named value is simply a tuple with two terms, one being a value and the other being the name of that value. For example in a street address a value may be “London” and the name of that value would be “City”.

- Reference: Data in a communication between two applications that allows a shared understanding of the individual things.
 - This could be an identifier (including a GUID), a name, an address, or a route, depending upon the needs.

Note:

- An entity may be known to be at a place in some functional or physical structure.
- A role may be known to be at a place in some process or behavioural structure.

Figure 6-3 below illustrates the naming/identifier-related attributes defined in the Generic information model. They are Global Unique ID (GUID), Local ID, Name, and Label.

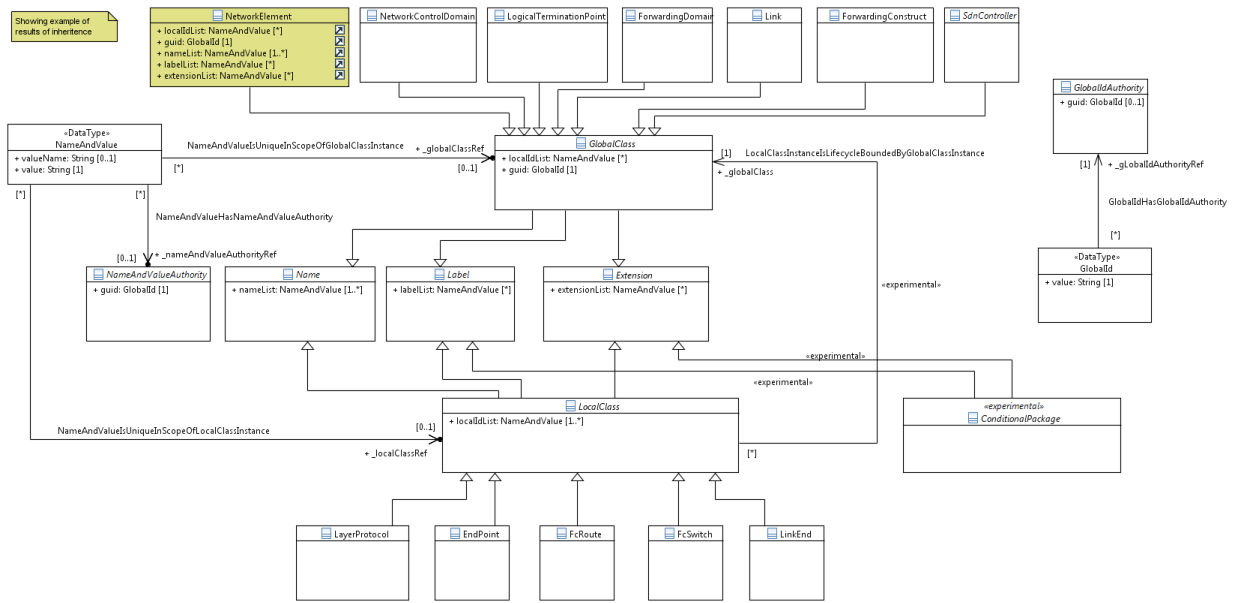
The model includes two abstract classes that provide names and identifiers, the GlobalClass and the LocalClass.¹⁸ A GlobalClass represents a type of thing that has instances which can exist in their own right (independently of any others). A LocalClass represents a type of thing that is inseparable from a GlobalClass, but that is a distinct feature of that GlobalClass such that the instances of LocalClass are able to have associations with other instances. The mandatory LocalId of the LocalClass instance is unique in the context of the GlobalClass instance, from which it is inseparable.

The model also includes Extension which is not related to naming/identification. Extension provides an opportunity to define properties not declared in the class that extend the class enabling a realization with simple ad-hoc extension of standard classes to be conformant.

Note that the GUID is applicable only to global type object classes (i.e. subclass of GlobalClass) that their instances can exist on their own right, i.e., NCD, NetworkElement, LTP, FD, Link, FC, and SdnController. The other naming/identifier-related attributes are applicable to both global type object classes and local type object classes (i.e., subclass of LocalClass).¹⁹

¹⁸ The model also provides ConditionalPackage to supply names and identifiers to _Pac classes but this is currently experimental.

¹⁹ The intention is that only classes from the Core Model are shown in the figure. The classes shown are essentially illustrative. There is another figure in the model that captures Core Model inheritance in detail. All classes from all fragments should inherit from GlobalClass, LocalClass or ConditionalPackage. There is no issue with model dependency as the inheritance association is maintained with the class that is inheriting properties. Although not mandatory, it would seem advisable to maintain a figure per fragment that shows all classes from that fragment and their inheritance.



CommonPackages-N
oNotes.PNG

Figure 6-3 Class Diagram for Naming and Identifier of Objects

The Core Foundation module also defines a State_Pac artifact, which is a package of state attributes. The work on states is experimental at this stage (it is derived from ITU-T X.731). The State_Pac is inherited by GlobalClass and LocalClass object classes. The State_Pac consists of the following state-related attributes:

- Operational State:
 - o Indicates the operability of the entity.
 - o Read-only with values:
 - DISABLED: The entity is totally inoperable and unable to provide service to its users(s).
 - ENABLED: The entity is partially or fully operable and available for use.
- AdministrativeControl (not derived from X.731):
 - o Reflects the current control action when the entity is not in the desired state.
 - o Read/Write with values:
 - NO_CONTROL: There is no current control action active as the entity is in the desired state.
 - UNLOCK: The intention is for the entity to become unlocked and the entity is not UNLOCKED.

- LOCK_PASSIVE: The intention is for the entity to become locked but no effort is expected to move to the Locked state (the state will be achieved once all users stop using the resource). The entity is not LOCKED.
 - LOCK_ACTIVE: The intention is for the entity to become locked and it is expected that an effort will be made to move to the Locked state (users will be actively removed). The entity is not LOCKED.
- Administrative State (derived from X.731 and extended):
- Indicates the degree to which the capabilities of the entity are allowed for use.
 - Read-only with values:
 - LOCKED-: The entity is administratively prohibited from performing services for its users.
 - UNLOCKED: The entity is administratively permitted to perform services for its users. This is independent of its inherent operability.
 - SHUTTING_DOWN_PASSIVE: The entity is administratively restricted to existing instances of use only. There may be no new instances of use enabled. This corresponds to a control of LOCK_PASSIVE.
 - SHUTTING_DOWN_ACTIVE: The entity is administratively restricted to existing instances of use only. There are specific actions to remove existing uses. There may be no new instances of use enabled. This corresponds to a control of LOCK_ACTIVE.
- Usage State:
- Indicates the degree to which the entity is used.
 - Read-only with values:
 - IDLE: The entity is not currently in use.
 - ACTIVE: The entity is in use and has sufficient spare operating capacity to provide for additional simultaneous uses.
 - BUSY: The entity is in use but has no spare operating capacity to provide for any further simultaneous uses.

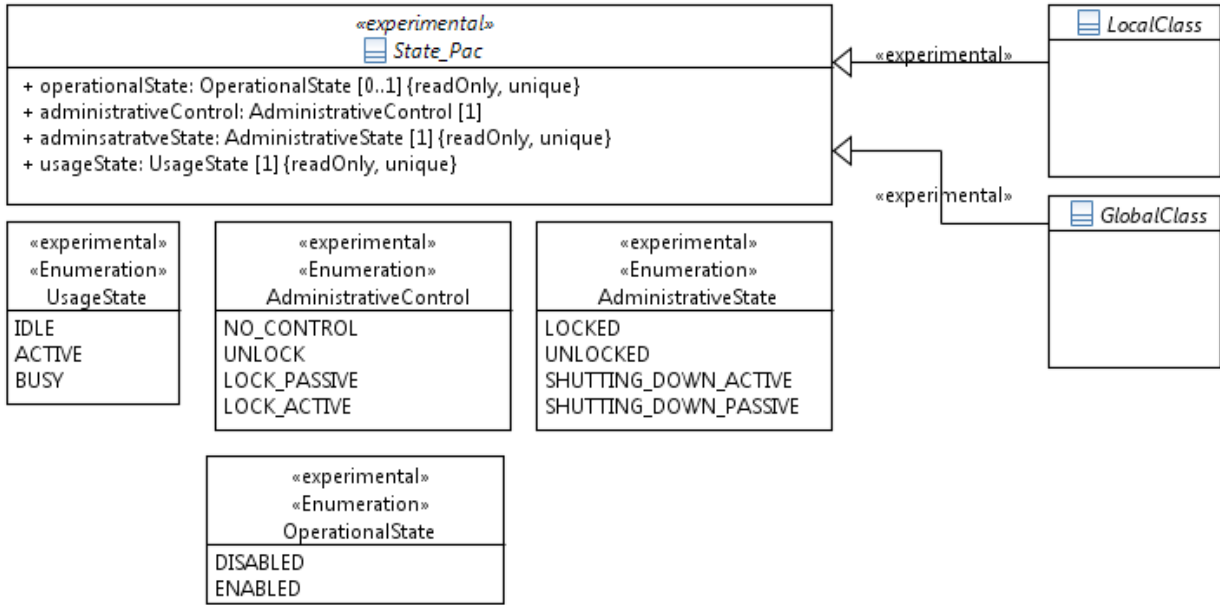
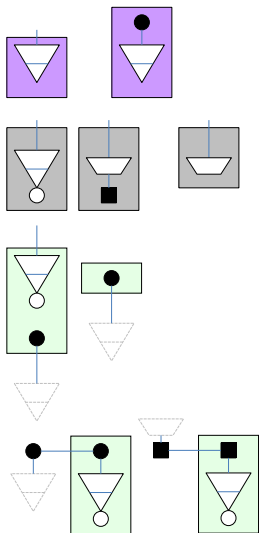


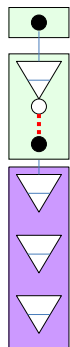
Figure 6-4 States for all Objects

6.3 Termination Subset of CNM

Examples of LTPs (using figures consistent with those used in TM Forum for PTP, CTP and FTP)



Port with various layers and flexibilities modeled as LTPs (PTPs and CTPs)



More precise view of port

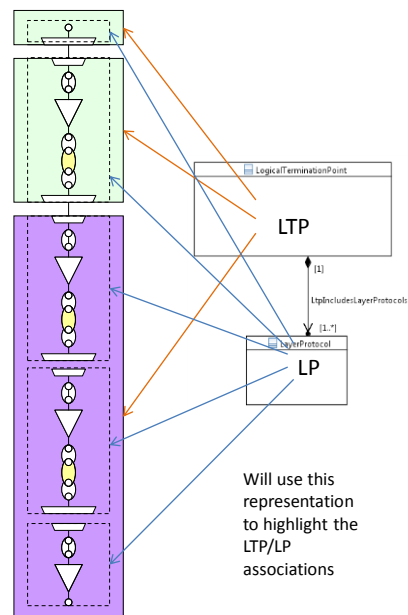


Figure 6-5 Representations of LTPs

In Figure 6-5 above the pictorial form shows a number of representations of LTPs (purple, grey and green) representing the layering associated with physical ports (purple), their connectable clients (green) and floating LTPs (grey). The right most pictorial form shows the relationship between the LTP and the LP in terms of a detailed symbol derived from work by TM Forum and ITU-T²⁰. An LP instance represents all aspects of termination of a single LayerProtocol. An LTP is composed on 1 or more LPs where the LPs represent the stack of terminations relevant to the LTP as depicted in the pictorial view. A termination stack may spread across several LTPs. The reason for this split includes multiplicity and connection flexibility transitions (see also Figure 5-2 in clause [Error! Reference source not found.](#)^{5.3} [Error! Reference source not found.](#) Pictorial diagram Key).

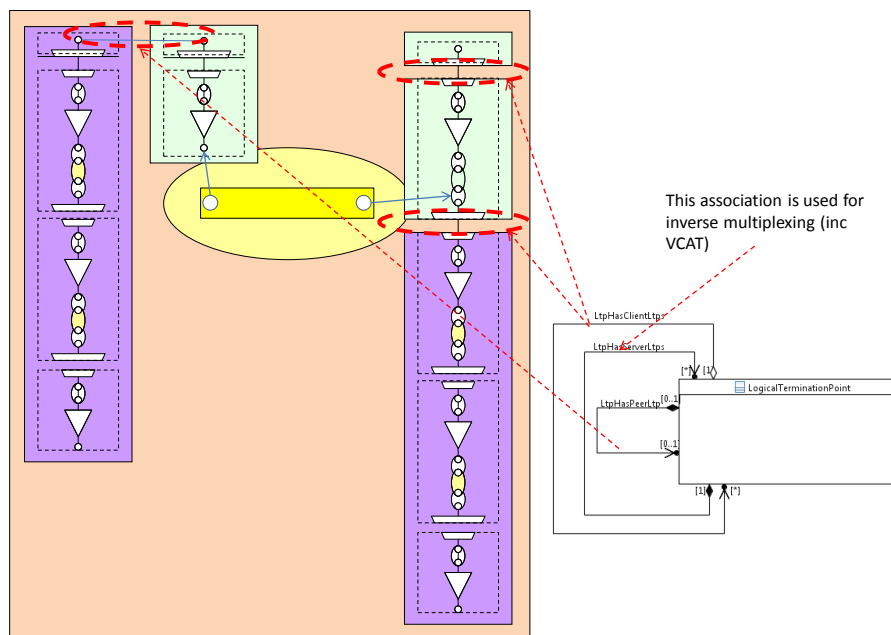


Figure 6-6 LTP relationships illustrated in a simple Network Element context

In Figure 6-6 above the pictorial form shows a number of LTPs (purple and green) representing the layering associated with physical ports (purple) and their connectable clients (green) as described in the previous section. This figure shows in more detail the partitioning of the layer stack between LTPs. Several different relationships are used at the split, depending upon the orientation of traffic flow.

Considering the left most LTP pair in the pictorial form and a signal entering the bottom of the purple LTP (at a physical port), the signal would be de-multiplexed up to the top of the purple LTP and then re-multiplexed as it travels down the green LTP. The association between the two is essentially a degenerate 1:1 FC. The LTPs are split because of the change in flow multiplexing orientation. The association supporting this relationship is shown in the UML fragment.

Considering the right most LTPs in the pictorial form and a signal entering the bottom of the purple LTP (at a physical port), the signal would be de-multiplexed up to the top of the purple LTP and then further de-multiplexed in the client LTPs. The LTPs are split because of a change in multiplicity or the opportunity to connect with an FC. The association supporting this relationship is shown in the UML fragment.

²⁰ The work has been liaised by TM Forum and related to Recommendation ITU-T G.805.

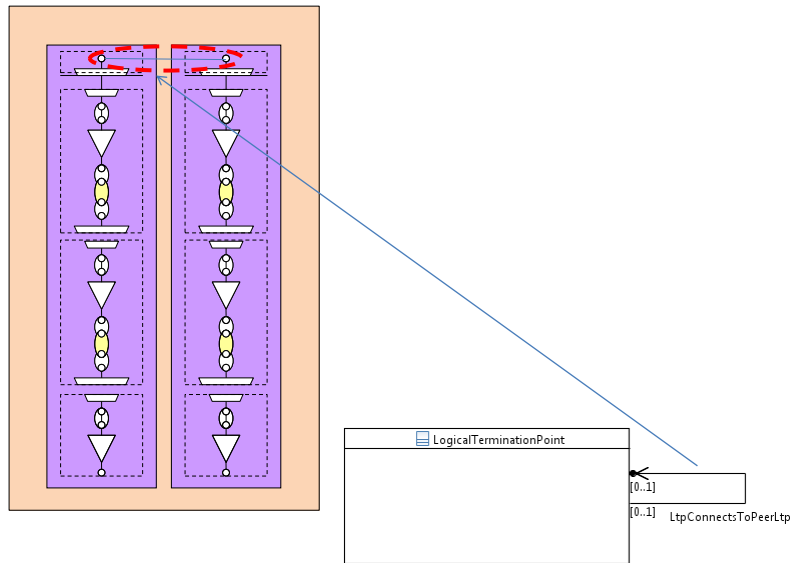


Figure 6-7 LtpConnectsToPeerLtp illustrated in an Amplifier/Regenerator context

In the simple Figure 6-7 above the final LTP to LTP association is highlighted. This allows two LTPs that are associated with physical ports without the need for an FC. This is only allowed in a case when the relationship between the LTPs is such that the whole signal from one LTP must flow to the other with no flexibility. The association effectively represents a degenerate FC.

6.4 Forwarding Subset of CNM

6.4.1 Basic Forwarding

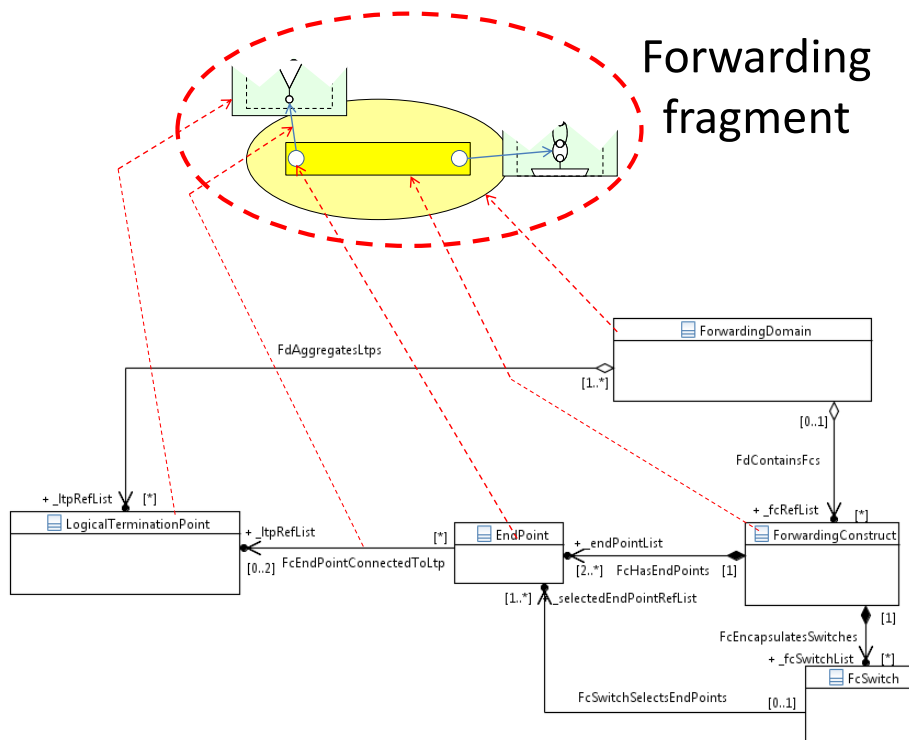


Figure 6-8 Forwarding fragment

The pictorial form in Figure 6-8 above shows the ForwardingConstruct (FC) in the context of two LTPs (a fragment of an earlier figure). The EndPoint (EP) of the FC is depicted as within the FC emphasizing the strict part-whole relationship and lifecycle dependency of the EP on the FC. The EPs are effectively ports on the FC component. The FC shown has two EPs but the model allows for two or more EPs [2..*] where in some cases the EP could be selected as a source or destination for switching. The protection switching capability is explained elsewhere in this document.

The [0..2] multiplicity of _ltpRefList allows for a bidirectional FC end to associate with two unidirectional LTPs.

6.4.2 Forwarding Construct Specification and other details of Forwarding

Prior to embarking on a brief description of the FC specification and associated classes it is important to explain the concept of specification classes in general. In this model the specification classes provide a mechanism to express the restrictions of a particular case of application of a specific class or set of classes. For example an FC may in general have [2..*] endpoints while a specific case of FC may have exactly 4. This case may also be such that it has 2 switches and such that these switches affect specific flows in the FC. The FcSpec is designed to allow the expression of cases of this sort.

At this point only limited work has been done on specification in general with a focus on the FcSpec and associated classes. It is anticipated that in general specification classes would be developed for all entities in the model.

In the diagram below the FcSpec and supporting EndpointSetSpec describe the capabilities of the FC in terms of MultiSwitchedUniFlows each of which has [1..*] IngressEndpointSets and [1..*] EgressEndpointSets. Each MultiSwitchedUniFlow may have [0..1] ingress switches and [0..1] egress switches where the ingress switch may select only one set member from one set and the egress switch may select [1..*] set members from the egress set. The ingress and egress switch selections are controlled by the ConfigurationAndSwitchControl element that may be:

- embedded in the switch when there is no coordination of switches required
- embedded in the FC when there is coordination of switches in the scope of the FC but no wilder
- independent of the FC and described by the ConfigurationGroupSpec where there is multi-FC coordination required

The behaviour of the ConfigurationAndSwitchControl element is described by ControlRules.

The model has been exercised for a number of different cases (not detailed here). Figure 6-9 below provides the class diagram of the FC specification fragment.

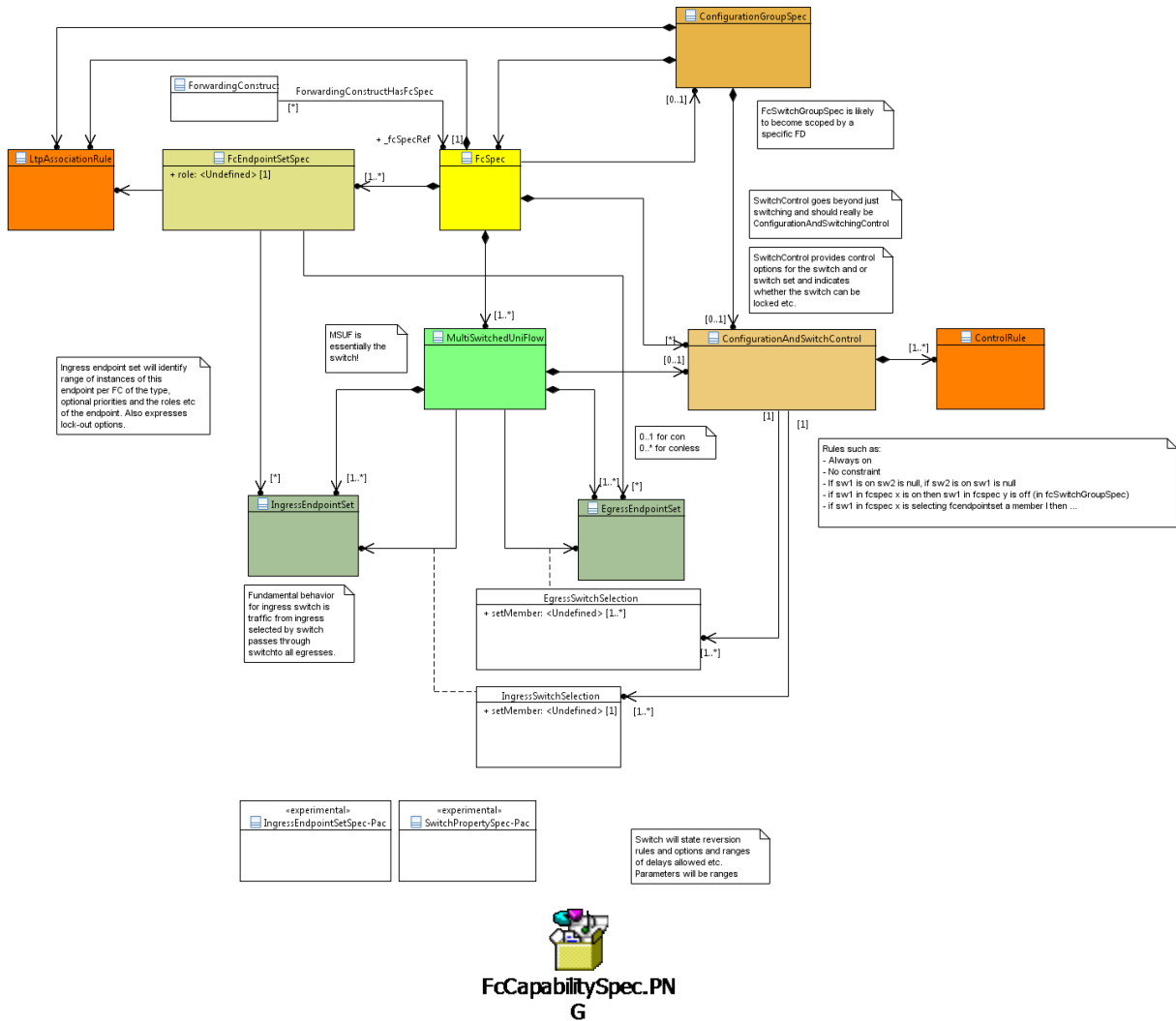


Figure 6-9 Class Diagram of the Spec Model of Connection Control

The diagrams below show a pictorial view of some of the classes above (the colors used in the figure are consistent with those used in the model above).

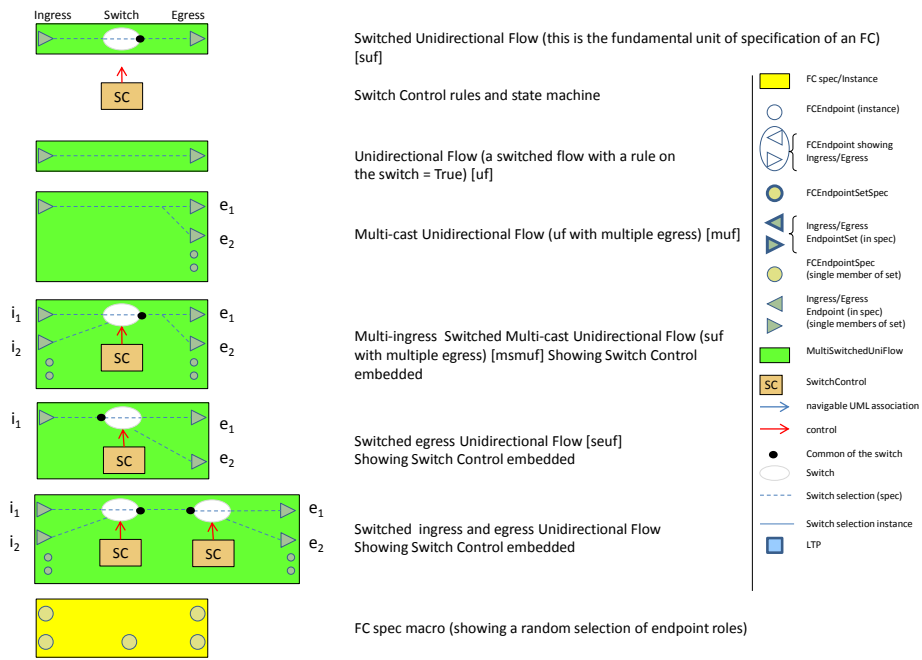


Figure 6-10 Pictorial view of the Spec Model of Configuration Control

The diagrams below show a pictorial view of a case of FcSpec. The lower element of the diagram shows specification class instances and the upper element shows an instance of FC abiding by the spec.

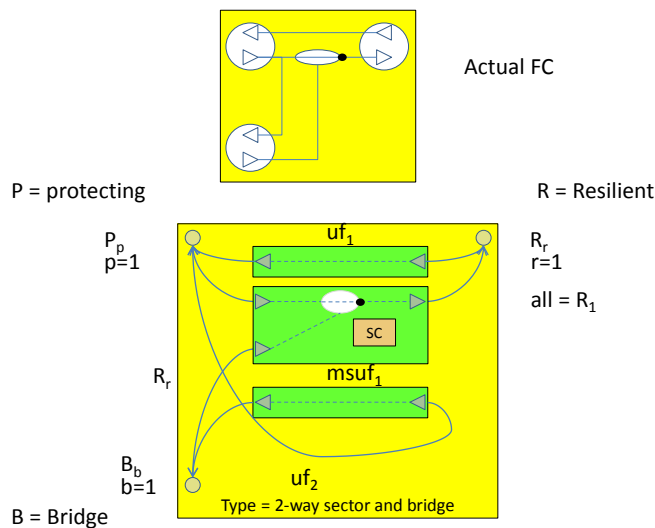
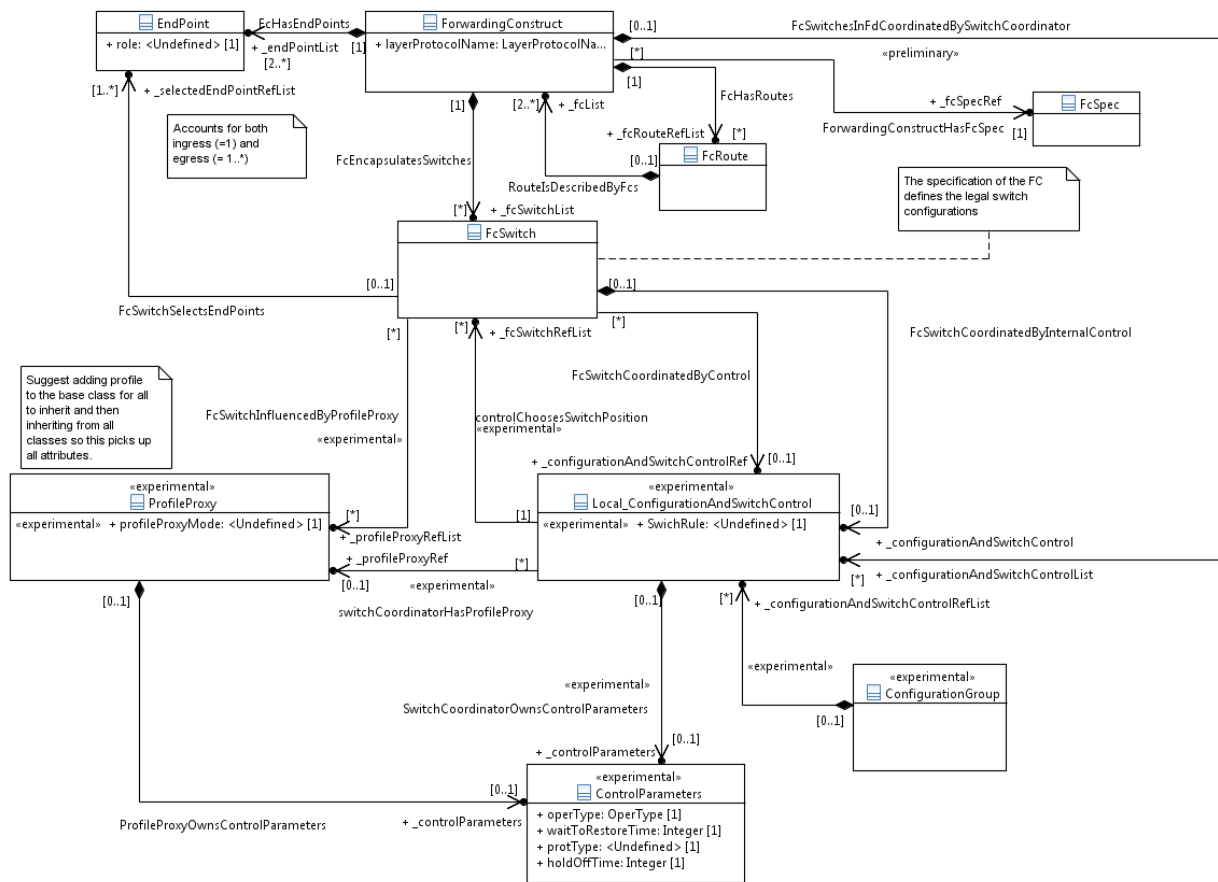


Figure 6-11 –Pictorial view of spec model and resulting FC instance

Figure 6-12 below provides the class diagram of further detailed FC and protection switching related object classes. The figure shows development of the controller of the FcSwitch. This area of model is experimental work in progress as highlighted by the «experimental» stereotypes.

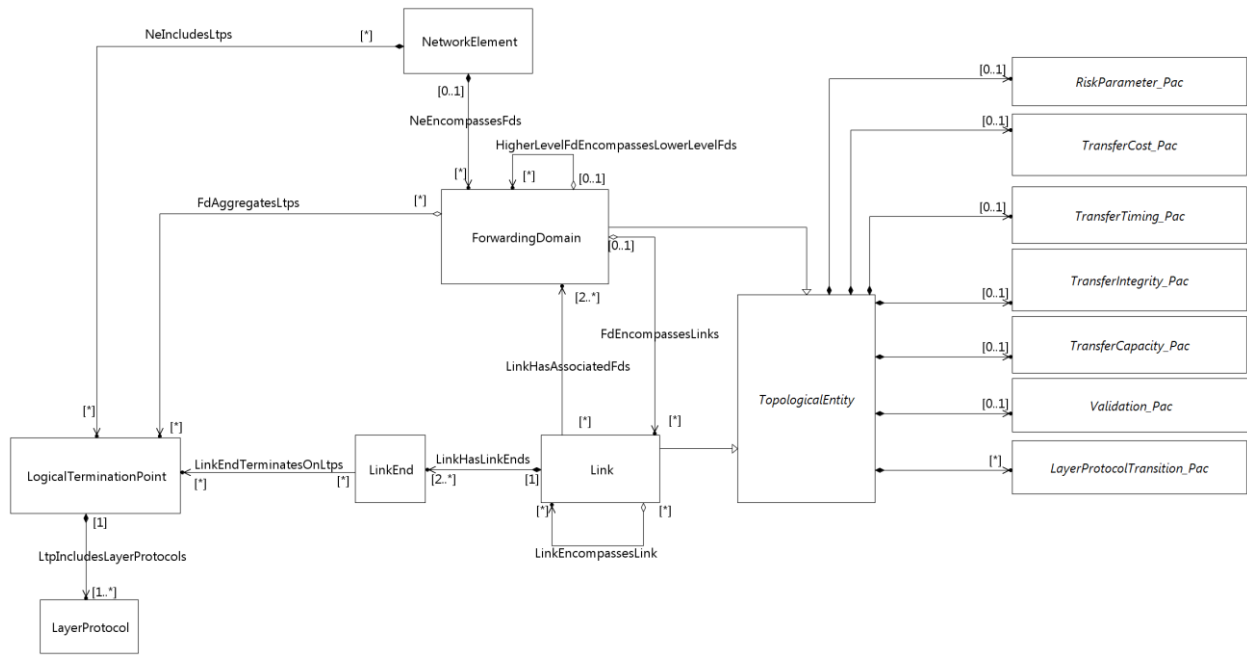



 SketchOfSwitchCoor
 dinatorAndProfile.PNI

Figure 6-12 Class Diagram of Connection related Object Classes

6.5 Topology Subset of CNM

The topology subset is summarized in the following figure.



Topology-HighLevelIO
 viewOfStructure-I

Figure 6-13 Classes of the Topology Subset

The figure above shows a lightweight view of the model omitting the attributes (where appropriate these will be described later in this section). The figure focuses on interrelationships and shows that:

- An FD may be a subordinate part of a NetworkElement, may coincide with an NetworkElement or may be larger than, and independent of, any NetworkElement (see for example FDs A.1 and A.3 in Figure 6-15).
- An FD may encompass lower level FDs. This may be such that:
 - o An FD directly contained in a NetworkElement is divided into smaller parts
 - o An FD not encompassed by a NetworkElement is divided into smaller parts some of which may be encompassed by NetworkElements
 - o The FD represents the whole network

Note that an FD at the lowest level of abstraction (i.e., a fabric) does not encompass FDs while an FD at the highest level of abstraction (i.e., the FD representing the whole network) is not encompassed by any higher level FDs.

- An FD encompasses Links that interconnect any FDs encompassed by the FD

Note that Offnet Links are not encompassed by any FD. All other Links are always encompassed by one FD which may be the FD representing the whole network. As a consequence, the FD representing the whole network shall always be instantiated.

- A Link may aggregate Links in several ways:

- In parallel where several links are considered as one
- In series where Links chain to form a Link of a greater span
 - Note that this case requires further development in the model
- A Link has associated FDs that it interconnects
 - A Link may interconnect 2 or more FDs²¹
 - Note that it is usual for a Link to interconnect 2 FDs but there are cases where many may be interconnected by a Link
- A Link has LinkEnds (LE) that represent the ports of the Link itself
 - LEs are especially relevant for multi-ended asymmetric Link
- An FD aggregates LogicalTerminationPoints (LTPs) that bound it. The LTP represents a stack of LayerProtocol terminations where the details of each is held in the LayerProtocol (LP). The LTP may be:
 - Part of a NetworkElement
 - Conceptually independent from any NetworkElement
- An LE references LTPs on which the Link associated to the LE terminates

Both the Link and FD are TopologicalEntities (an abstract class, i.e. a class that will never instantiate) and hence they can acquire contents from the conditional packages (_Pacs). The conditional packages provide all key topology properties.

6.5.1 Basic Topology

The first two figures focus on the ForwardingDomain class and the recursive aggregation relationship as well as the relationship between the ForwardingDomain, Link and the NetworkElement.

²¹ An off-network link with two ends does not interconnect any FDs in the view.

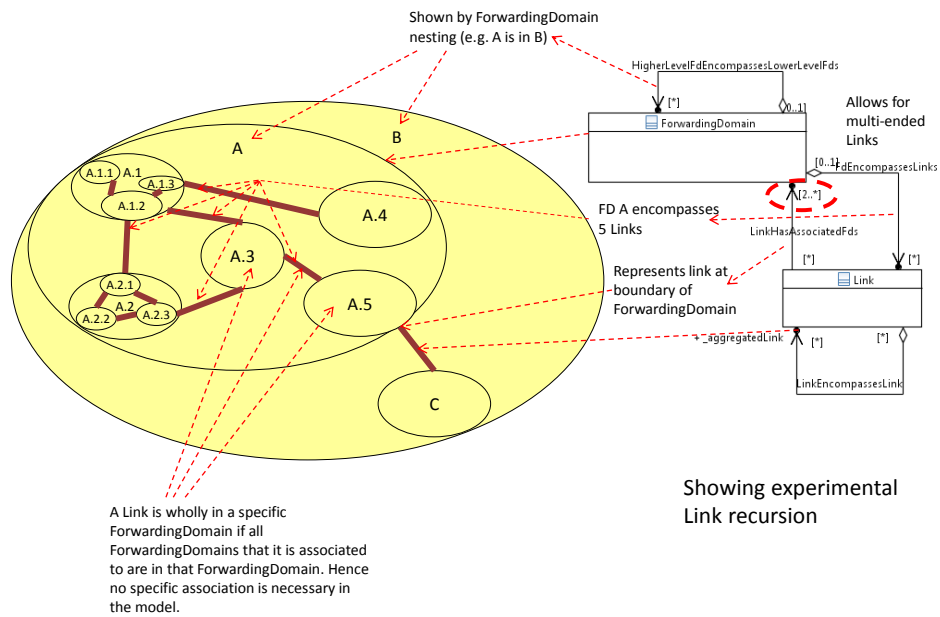


Figure 6-14 ForwardingDomain recursion with Link

Figure 6-14 shows a UML fragment including the Link and ForwardingDomain (FD). For simplicity it is assumed here that the Links and FDs are for a single LayerProtocol (LP) although an FD can support a list of LPs.

The pictorial form shows a number of instances of FD interconnected by Links and shows nesting of FDs. The recursive aggregation HigherLevelFdEncompassesLowerLevelFds relationship (aggregation is represented by an open diamond) supports the ForwardingDomain nesting but it should be noted that this is intentionally showing no lifecycle dependency between the lower ForwardingDomains and the higher ones that nest them (to do this composition, a black diamond would have been used instead of an open diamond). This is to allow for rearrangements of the ForwardingDomain hierarchy (e.g., when regions of a network are split or merged) and to emphasize that the nesting is an abstraction rather than decomposition. The underlying network still operates regardless of how it is perceived in terms of aggregating ForwardingDomains. The model allows for only one hierarchy.

In the example of Figure 6-14, there are fourteen FD instances with the following instances of the “HigherLevelFdEncompassesLowerLevelFds” relationships:

- B encompasses two FDs: A and C
- A encompasses five FDs: A.1, A.2, A.3, A.4 and A.5
- A.1 encompasses three FDs: A.1.1, A.1.2 and A.1.3
- A.2 encompasses three FDs: A.2.1, A.2.2 and A.2.3

When one FD is removed, the “HigherLevelFdEncompassesLowerLevelFds” relationships are modified. For example, if FD A.1 in Figure 6-14 is removed, the instances of the “HigherLevelFdEncompassesLowerLevelFds” relationships will be modified as follows:

- B encompasses two FDs: A and C

- A encompasses seven FDs: A.1.1, A.1.2, A.1.3, A.2, A.3, A.4 and A.5²²
- A.2 encompasses three FDs: A.2.1, A.2.2 and A.2.3

An FD can also be added. Initially it will have no associated lower level FDs. Existing FDs can be moved as appropriate to form the new hierarchy.

The association between Link and FD allows a Link to be terminated on two or more FDs (see clause [Error! Reference source not found.](#) [6.1.7 Error! Reference source not found.](#) [Link and LinkEnd](#)). Through this the model supports point to point Links as well as cases where the server ForwardingConstruct is multi-point terminated giving rise to a multi-pointed Link. Multi-pointed links occur in for PON and Layer 2 MAC in MAC configurations²³.

It should be noted that the model includes LinkEnd which further details the relationship between FD and Link. This is explained below.

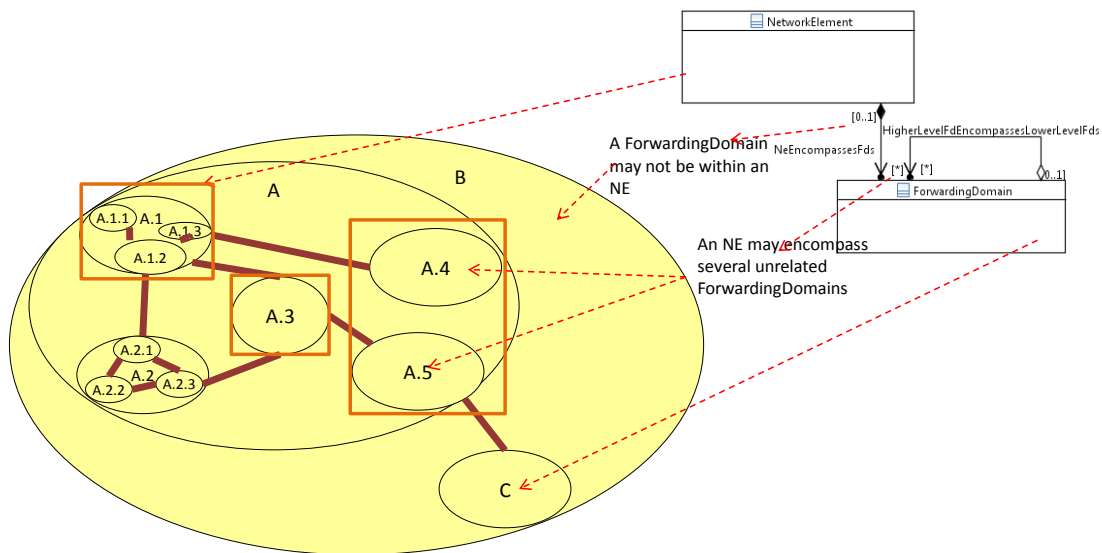


Figure 6-15 ForwardingDomain recursion with link and NetworkElement

In Figure 6-15 above the pictorial form shows an overlay of NetworkElement on the ForwardingDomains and a corresponding fragment of UML showing only the ForwardingDomain and NetworkElement classes.

The figure emphasizes that at and below one particular level of abstraction of ForwardingDomain, the ForwardingDomains are all bounded by a specific NetworkElement. This is represented in the UML fragment by the composition association (black diamond) that explains that there is a lifecycle dependency in that the ForwardingDomain at this level cannot exist without the NetworkElement. The figure also shows that a ForwardingDomain needs not be bounded by a NetworkElement (as explained in the UML fragment by the 0..1 composition), and that a ForwardingDomain may have a smaller scope than the whole NetworkElement (even when considering only a single LayerProtocol as noted earlier). In one case depicted (e.g., the right hand side NetworkElement encompassing two FDs), the two ForwardingDomains in the NetworkElement are completely independent. In the other

²² Clearly the FD naming in the figure is for ease of reading the diagram and does not represent hierarchy.

²³ Work supporting this was liaised from TM Forum.

cases depicted (e.g., the left hand side NetworkElement encompassing three FDs), the subordinate ForwardingDomains are themselves joined by Links emphasizing that the NetworkElement does not necessarily represent the lowest level of relevant network decomposition.

The figure also emphasizes that just because one ForwardingDomain at a particular level of decomposition of the network happens to be the one bounded by a NetworkElement does not mean that all ForwardingDomains at that level are also bounded by NetworkElements.²⁴

6.5.2 Advanced Topology

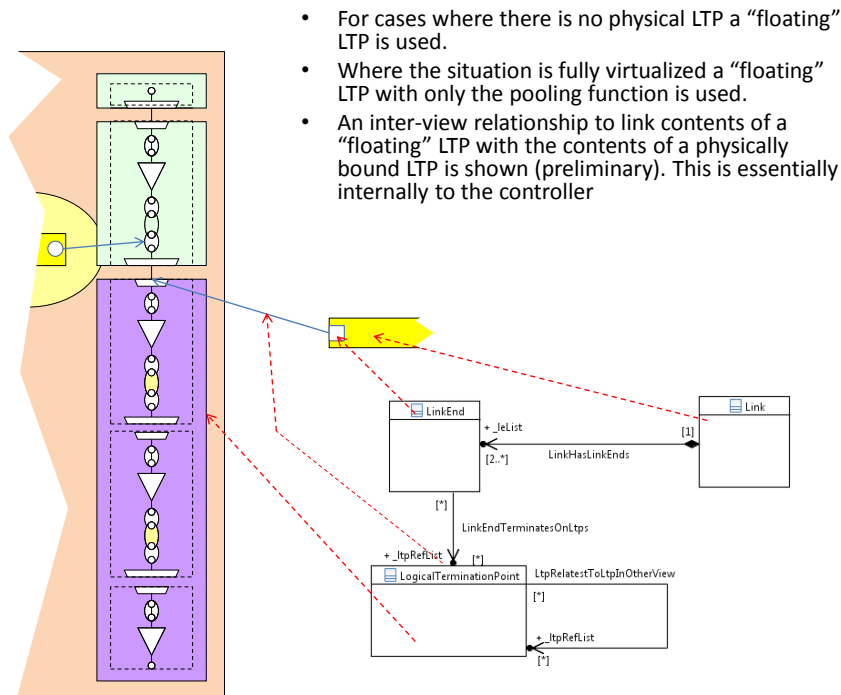


Figure 6-16 LTP “pooling” client LTPs

Figure 6-16 above shows how the Link terminates on the LTP via the LinkEnd (LE).

²⁴ It should be noted that a NetworkElement is never within the bounds of an FD. The NetworkElement is associated with levels in the FD hierarchy.

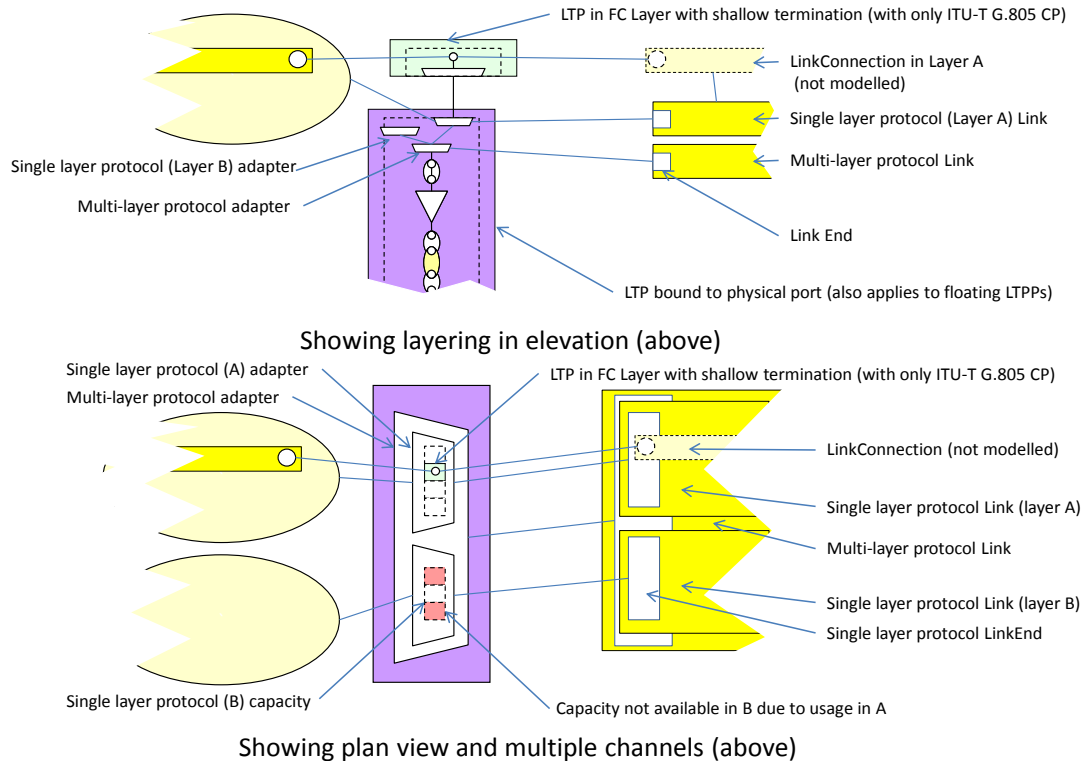


Figure 6-17 Views of Link, LinkEnd and LTP showing LTP pooling

The LTP may have the capability²⁵ to map to multiple client layer protocols where there is an interaction between the client mappings (e.g., if capacity/channel x of client layer protocol A is used then capacity/channel set y of client layer protocol B is no longer available). The capacity of the Link is determined by evaluating the “intersection” of capabilities of the LTPs at the ends (which is complex in a multi-ended case).

The used capacity is determined by considering which client LTPs exist as a result of their being FCs.

A Link may be multi-layered and hence may represent the whole client capacity of an LTP or it may be single layered.

²⁵ This capability of the LTP is not currently modeled but work is under way to construct an LTP specification model

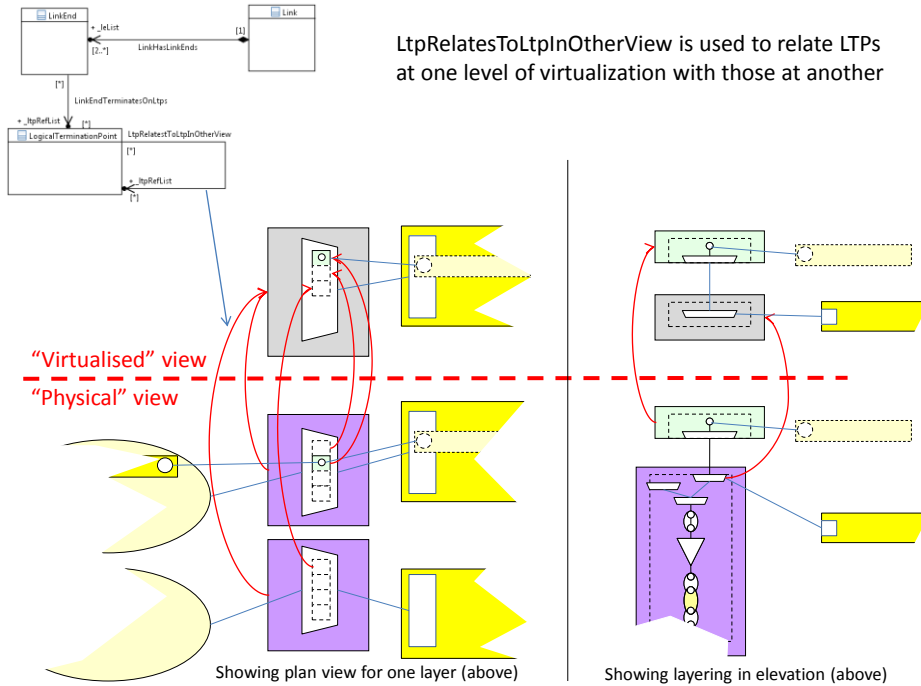


Figure 6-18 Views of “virtualization” of LTP

Some capacity may be taken from each of a number of Links supporting a particular layer protocol and offered in a “virtualized” view perhaps for use in a particular application etc. The “virtualized” view will normally be referenced in a different name space. The rules for grouping capacity into Links in the “virtualized” view have not yet been documented. The same model is used for Links and LTPs in the “virtualized” view as is used in the “physical” view.

6.5.3 Detailed properties of Topology

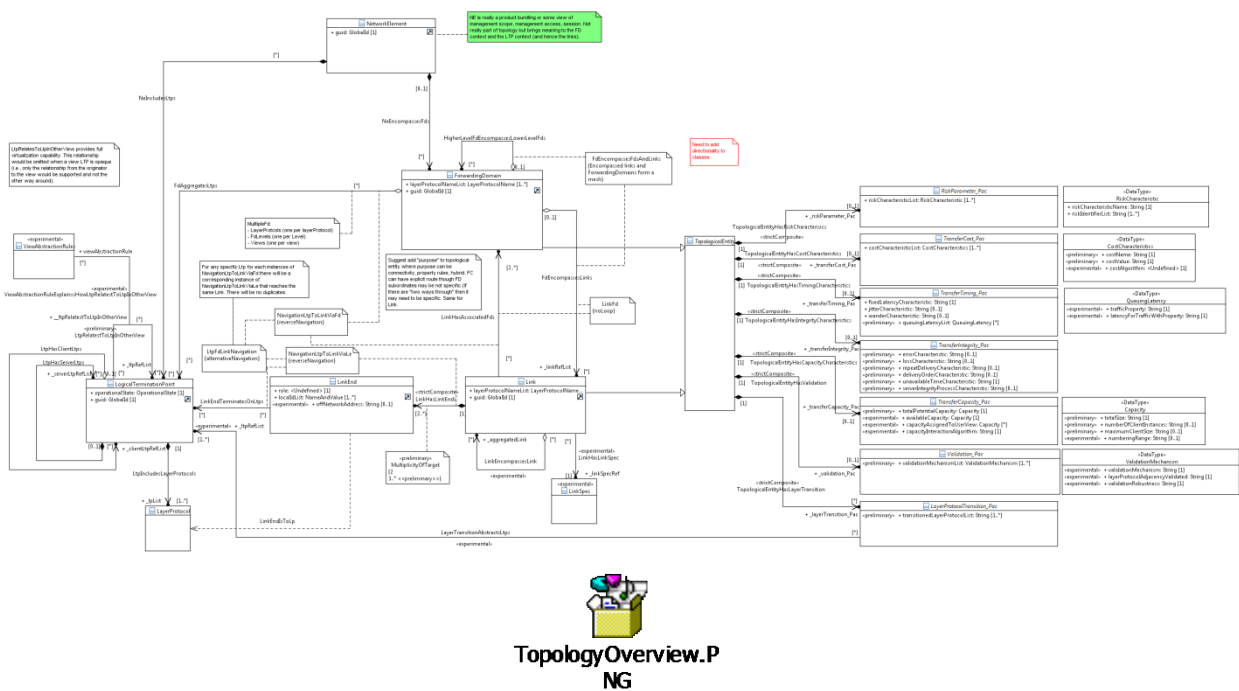


Figure 6-19 Topology detail

The figure above shows finalized, preliminary and experimental extensions of the Topology model. The model recognizes that both ForwardingDomain and Link share topological properties (the TopologicalEntity, which is abstract and hence not intended to be instantiated, provides the linkage²⁶). The classes related to TopologicalEntity, the _Pacs, are « strictComposition » and hence are essentially part of the ForwardingDomain and of the Link. The _Pacs are optional as in some cases of Link/ForwardingDomain they are essentially not relevant.

The figure below shows the _Pacs in more detail.

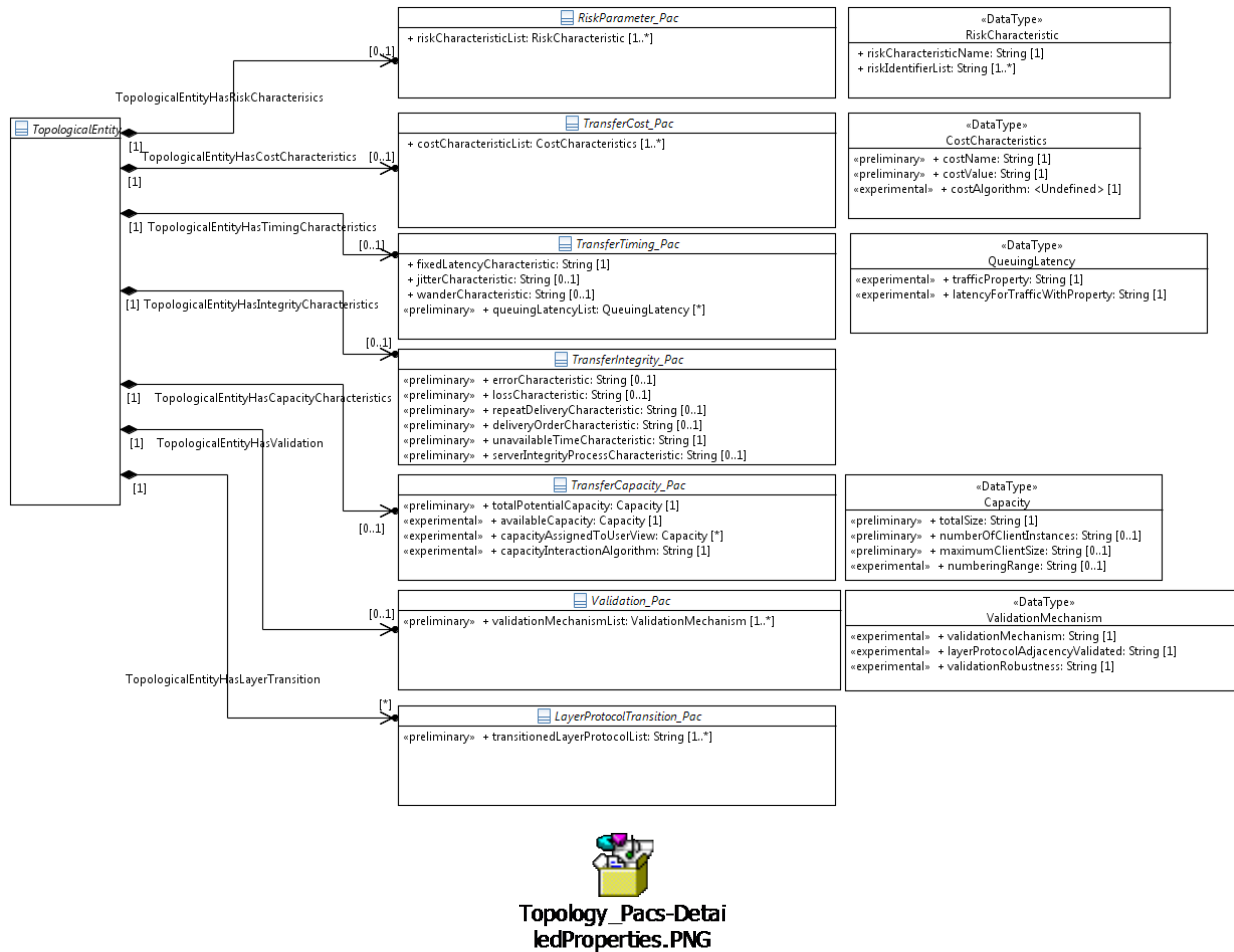


Figure 6-20 Topology _Pac detail

As shown in the figure an object class “TopologicalEntity” has been defined to collect topology-related properties (characteristics etc.) that are common for FD and Link.

A TopologicalEntity is an abstract representation of the emergent effect of the combined functioning of an arrangement of components (running hardware, software running on hardware, etc). The effect can be considered as the realization of the potential for apparent communication adjacency for entities that are bound to the terminations at the boundary of the TopologicalEntity.

²⁶ TopologicalEntity has no direct attributes and only relationships that the ForwardingDomain and Link inherit.

The TopologicalEntity enables the creation of constrained forwarding to achieve the apparent adjacency. The apparent adjacency has intended performance degraded from perfect adjacency and a statement of that degradation is conveyed via the attributes of the packages associated with this class. In the model both ForwardingDomain and Link are TopologicalEntities.

This abstract class is used as a modeling approach to apply packages of attributes to both Link and ForwardingDomain. Link and ForwardingDomain are the key TopologicalEntities.

The _Pacs are detailed in the following sections.

6.5.3.1 RiskParameter_Pac

The risk characteristics of a TopologicalEntity come directly from the underlying physical realization.

The risk characteristics propagate from the physical realization to the client and from the server layer to the client layer, this propagation may be modified by protection.

A TopologicalEntity may suffer degradation or failure as a result of a problem in a part of the underlying realization.

The realization can be partitioned into segments which have some relevant common failure modes.

There is a risk of failure/degradation of each segment of the underlying realization.

Each segment is a part of a larger physical/geographical unit that behaves as one with respect to failure (i.e. a failure will have a high probability of impacting the whole unit (e.g. all cables in the same duct).

Disruptions to that larger physical/geographical unit will impact (cause failure/errors to) all TopologicalEntities that use any part of that larger physical/geographical entity.

Any TopologicalEntity that uses any part of that larger physical/geographical unit will suffer impact and hence each TopologicalEntity shares risk.

The identifier of each physical/geographical unit that is involved in the realization of each segment of a Topological entity can be listed in the RiskParameter_Pac of that TopologicalEntity.

A segment has one or more risk characteristics.

Shared risk between two TopologicalEntities compromises the integrity of any solution that uses one of those TopologicalEntities as a backup for the other.

Where two TopologicalEntities have a common risk characteristic they have an elevated probability of failing simultaneously compared to two TopologicalEntities that do not share risk characteristics.

- riskCharacteristicList: A list of risk characteristics (RiskCharacteristic) for consideration in an analysis of shared risk. Each element of the list represents a specific risk consideration.
- RiskCharacteristic: The information for a particular risk characteristic where there is a list of risk identifiers related to that characteristic. It includes:
 - riskCharacteristicName: The name of the risk characteristic. The characteristic may be related to a specific degree of closeness. For example a particular characteristic may apply to failures that are localized (e.g., to one side of a road) where as another characteristic may relate to failures that have a broader impact (e.g., both sides of a road that crosses a bridge). Depending upon the importance of the traffic being routed different risk characteristics will be evaluated.

- riskIdentifierList: A list of the identifiers of each physical/geographic unit (with the specific risk characteristic) that is related to a segment of the TopologicalEntity.

6.5.3.2 TransferCost_Pac

The cost characteristics of a TopologicalEntity are not necessarily correlated to the cost of the underlying physical realization.

They may be quite specific to the individual TopologicalEntity, e.g., opportunity cost that relates to layer capacity.

There may be many perspectives from which cost may be considered for a particular TopologicalEntity and hence many specific costs and potentially cost algorithms.

Using an entity will incur a cost.

- costCharacteristicList: The list of costs (CostCharacteristic) where each cost relates to some aspect of the TopologicalEntity.
 - CostCharacteristic: The information for a particular cost characteristic
 - costName: The cost characteristic will related to some aspect of the TopologicalEntity (e.g., \$ cost, routing weight). This aspect will be conveyed by the costName_.
 - costValue: The specific cost.
 - costAlgorithm: The cost may vary based upon some properties of the TopologicalEntity. The rules for the variation are conveyed by the costAlgorithm.

6.5.3.3 TransferTiming_Pac

A link will suffer effects from the underlying physical realization related to the timing of the information passed by the link.

- fixedLatencyCharacteristic: A TopologicalEntity suffers delay caused by the realization of the servers (e.g., distance related; FEC encoding etc.), along with some client specific processing. This is the total average latency effect of the TopologicalEntity.
- jitterCharacteristic: High frequency deviation from true periodicity of a signal and therefore a small high rate of change of transfer latency. Applies to TDM systems (and not packet).
- wanderCharacteristics: Low frequency deviation from true periodicity of a signal and therefore a small low rate of change of transfer latency. Applies to TDM systems (and not packet).
- queuingLatencyList: The effect on the latency of a queuing process. This only has significant effect for packet based systems and has a complex characteristic (QueuingLatency).
 - QueuingLatency: Provides information on latency characteristic for a particular stated trafficProperty.

6.5.3.4 TransferIntegrity_Pac

Transfer integrity characteristic covers expected (specified) error, loss and-or duplication of signal content as well as any damage of any form to total link and to the client signals.

- errorCharacteristic: describes the degree to which the signal propagated can be errored. Applies to TDM systems as the errored signal will be propagated, but does not apply to packet systems, as errored packets will be discarded.
- lossCharacteristic: Describes the acceptable characteristic of lost packets where loss may result from discard due to errors or overflow. Applies to packet systems and not TDM (~~as for TDM errored signals are propagated unless grossly errored and overflow/underflow turns into timing slips~~).
- repeatDeliveryCharacteristic: Primarily applies to packet systems where a packet may be delivered more than once (in fault recovery for example). It can also apply to TDM where several frames may be received twice due to switching in a system with a large differential propagation delay.
- deliveryOrderCharacteristic: Describes the degree to which packets will be delivered out of sequence. Does not apply to TDM as the TDM protocols maintain strict order.
- unavailableTimeCharacteristic: Describes the duration for which there may be no valid signal propagated.
- serverIntegrityProcessCharacteristic: Describes the effect of any server integrity enhancement process on the characteristics of the TopologicalEntity.

6.5.3.5 TransferCapacity_Pac

The TopologicalEntity derives capacity from the underlying realization.

A TopologicalEntity may be an abstraction and virtualization of a subset of the underlying capability offered in a view or may be directly reflecting the underlying realization.

A TopologicalEntity may be directly used in the view or may be assigned to another view for use.

The clients supported by a multi-layer TopologicalEntity may interact such that the resources used by one client may impact those available to another. This is derived from the LTP spec details.

A TopologicalEntity represents the capacity available to user (client) along with client interaction and usage.

A TopologicalEntity may reflect one or more client protocols and one or more members for each profile.

- totalPotentialCapacity: An optimistic view of the capacity of the TopologicalEntity assuming that any shared capacity is available to be taken.

Note that this area is still under development to cover concepts such as:

- exclusiveCapacityList: The capacity allocated to this TopologicalEntity for its exclusive use.
- sharedCapacityList: The capacity allocated to this TopologicalEntity that is not exclusively available as it is shared with others.
- assignedAsExclusiveCapacityList: The capacity assigned from this TopologicalEntity to another TopologicalEntity for its exclusive use.
- assignedAsSharedCapacityList: The capacity assigned to one or more other TopologicalEntities for shared use where the interaction follows some stated algorithm.
- Capacity which includes:
 - totalSize

- numberOfUsageInstances
- maximumUsageSize
- numberingRange

6.5.3.6 Validation_Pac

Validation covers the various adjacent discovery and reachability verification protocols. Also may cover Information source and degree of integrity.

- validationMechanismList: Provides details of the specific validation mechanism(s) used to confirm the presence of an intended topologicalEntity.

6.5.3.7 LayerProtocolTransition_Pac

Relevant for a Link that is formed by abstracting one or more LTPs (in a stack) to focus on the flow and deemphasize the protocol transformation.

This abstraction is relevant when considering multi-layer routing.

The layer protocols of the LTP and the order of their application to the signal is still relevant and need to be accounted for. This is derived from the LTP spec details.

This Pac provides the relevant abstractions of the LTPs and provides the necessary association to the LTPs involved.

Links that included details in this Pac are often referred to as Transitional Links.

- transitionedLayerProtocolList: Provides the ordered structure of layer protocol transitions encapsulated in the TopologicalEntity. The ordering relates to the LinkEnd role.

7 Future Area of the GIM

The clause contains place holders of areas that are being or will be modelled.

- 7.1 Synchronization (frequency and time/phase) module
- 7.2 Scheduling module
- 7.3 Logging module
- 7.4 Notification module
- 7.5 Performance management module
- 7.6 Fault management module
- 7.7 ECC management module
- 7.8 Policy management module
- 7.9 Physical Equipment management module
- 7.10 Generalized OAM/MEP functions

8 UML model files

8.1 Papyrus File

This section contains the information model files and the companion profile file specified using the “Papyrus” modelling tool.



G.7711_v1.0_PAP.zip

- Model: consists of four files:
 - .project,
 - CoreModel.di,
 - CoreModel.notation
 - CoreModel.uml
- Profiles: consists of four files:
 - .project,
 - OpenModel_Profile.profile.di
 - OpenModel_Profile.profile.notation
 - OpenModel_Profile.profile.uml

In order to view and further extend or modify the information model, one will need to install the open source Eclipse software and the Papyrus tool. The installation guide for Eclipse and Papyrus can be found at <https://www.eclipse.org/papyrus/updates/index.php>.

8.2 Data Dictionary File

~~Attached below is the~~A data dictionary ~~format~~ of the G.7711 v1.0 information model in MS WORD (.docx) ~~document format will be generated~~. The data dictionary includes the description ~~and properties~~ of the object classes and their attributes, including the ~~and~~ association attributes ~~etc~~.

- Core ~~Network Module~~Model data dictionary



CoreNetworkModule
Object Classes-v0.02_



G.7710_v1.0_DD.do
cx

~~Foundation Module data dictionary~~



FoundationModule
Object Classes-v0.02

NOTE: The data dictionary is generated by the Gendoc
(<http://projects.eclipse.org/projects/modeling.gendoc>) tool using the following script:



G.7710_v1.0_DD_Sc
ript_for_Gendoc.docx

Annex A

UML Modelling Guideline

(This annex forms an integral part of this Recommendation.)

A.1 Introduction

This Annex defines the guidelines that have to be followed during the creation of a protocol-neutral UML (Unified Modelling Language) information model. These UML Modelling Guidelines are based on the UML guidelines defined in Recommendation ITU-T G.8052 and have been harmonized with the UML modelling guidelines used in the industry, such as TM Forum [b-3GPP/TMF-JWG] and ONF [b-ONF-UML-Guide].

UML defines a number of basic model elements, called UML artefacts. In order to assure consistent modelling, only a selected subset of these artefacts is used in the UML model guidelines in this Recommendation. The semantic of the selected artefacts is defined in [b-UML].

The description of each basic model artefact 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 [b-Papyrus].

A.2 Source References

- Papyrus Eclipse UML Modeling Tool (<https://www.eclipse.org/papyrus/>)
- Unified Modeling Language™ (UML®) (<http://www.uml.org/>)
- OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4
- 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)

A.3 Overview

A.3.1 Modelling 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.

A.3.2 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.

A.4 UML Artifact Descriptions

A.4.1 Object Classes

Object classes are used to convey a static²⁷ representation of an entity, including properties and attributes; i.e., data model, the static part of the model.

A.4.1.1 Object Class Notation

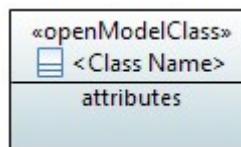


Figure A.0.1: Graphical Notation for Object Classes

As highlighted in [Figure A.0.1](#)~~Figure A.4.1~~, an object class is represented with a name compartment and an attributes compartment. The attributes compartment can be set in a diagram to not expose the attributes or to expose some or all of the attributes.

In some diagrams the attributes are not exposed so as to reduce clutter, in others only a subset of the attributes is exposed so as to focus attention. It is also possible to hide the attribute compartment of a class in the class diagrams where a large number of classes need to be shown, as depicted in [Figure A.0.2](#)~~Figure A.4.2~~.

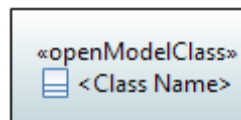


Figure A.0.2: Graphical Notation for Object Classes without attributes compartment

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 A.0.3](#)~~Figure 4.3~~, but as in the model the static part and the dynamic part of the model are decoupled, the operation compartment, is not used and always hidden.

²⁷ Not about operations acting on the entity.

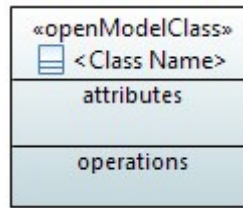




Figure A.0.3: Graphical Notation for Object Classes with attributes and deprecated operations compartment

A.4.1.2 Object Class Properties

An object class  has the following properties:

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

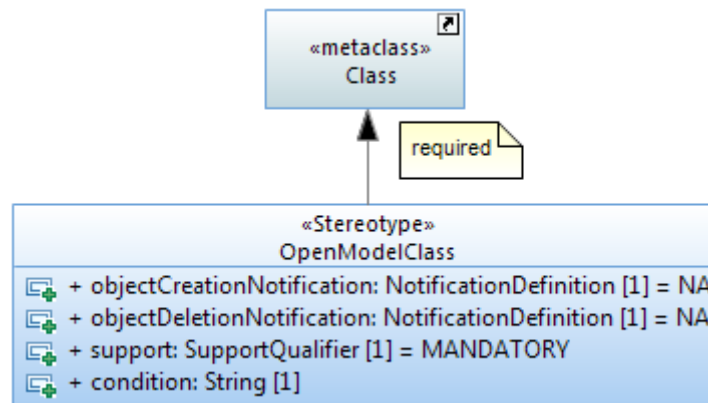


Figure A.0.4: «OpenModelClass» Stereotype

- objectCreationNotification (only relevant in the purpose-specific modules of the Information Model; see [Error! Reference source not found, Figure A.3.1](#))
Defines whether an object creation notification has to be send when the object instance is created.
- objectDeletionNotification (only relevant in the purpose-specific modules of the Information Model; see [Error! Reference source not found, Figure A.3.1](#))
Defines whether an object deletion notification has to be send when the object instance is deleted.
- support
This property qualifies the support of the object class at the management interface. See definition in section A.04.9.

- 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.

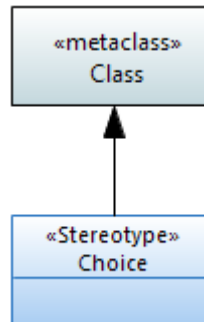


Figure A.0.5: Potential Choice annotation for Object Classes

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

A.4.2 Attributes in Object Classes

Attributes contain the properties²⁸ of an object class. Note that the roles of navigable association ends become an attribute in the associated object class.

A.4.2.1 Attribute Notation

The notation is:

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

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

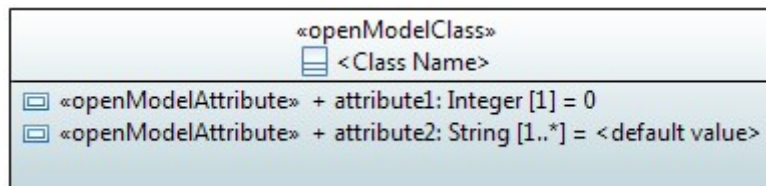


Figure A.0.6: Graphical Notation for Object Classes with Attributes

A.4.2.2 Attribute Properties

An attribute has the following properties:

- Name
Follows Lower Camel Case (LCC) and is unique across all attribute names in the model. An example of Lower Camel Case: subNetworkConnectionIdentifier. The plural form is "<attribute name>List".
Boolean typed attribute names always start with a verb like 'is', 'must', etc. (e.g., 'isAbstract') and the whole attribute name must be composed in a way that it is possible to answer it by "true" or "false".


²⁸ In Papyrus an attribute is a property.

- **Documentation**
Contains a short summary of the usage. The documentation is carried in the “Comments” field in Papyrus.
- **Ordered**
For a multi-valued multiplicity; this specifies whether the values in an instantiation of this attribute are sequentially ordered; default is false.
- **Unique**
For a multi-valued multiplicity, this specifies if the values of this attribute instance are unique (i.e., no duplicate attribute values); default is true.

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

Table A.0.1: 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. The state is dependent on the additional attribute property writeable. I.e., if the attribute cannot be set/changed by the client, it is read only.
- **Type**
Refers to a data type; see section A.[04.8](#).
- **Default Value**
Provides the value that the attribute has to start with in case the value is not provided during creation or already defined because of a system state.
In cases where a default value makes no sense (e.g., if representing the state of the system) it is undefined which is shown by “--“.
- **Multiplicity (*, 1, 1..*, 0..1, ...)**
Defines the number of values the attribute can simultaneously have.
 - * is a list attribute with 0, one or multiple values;
 - 1 attribute has always one value;
 - 1..* is a list attribute with at least one value;
 - 0..1 attribute may have no or at most one value;Default value is 1.
Other values are possible; e.g., “2..17”.
- **Additional properties are defined in the «OpenModelAttribute» stereotype which extends ( [Extension](#)) by default ({required}) the «metaclass» Property:**

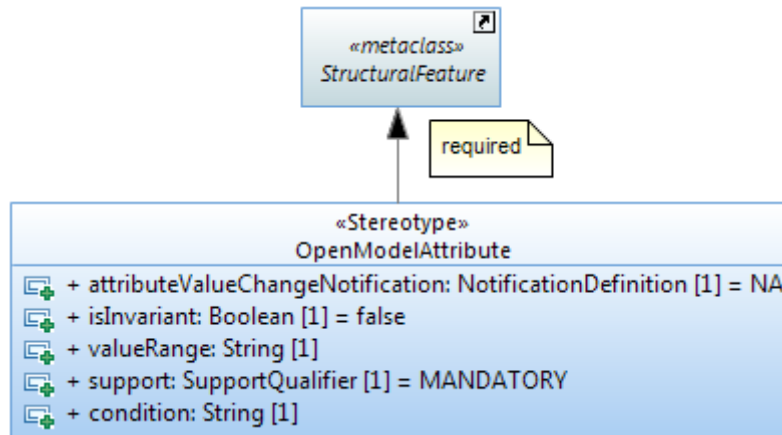


Figure A.0.7: «OpenModelAttribute» Stereotype

- attributeValueChangeNotification (only relevant in the purpose-specific modules of the Information Model; see [Error! Reference source not found, Figure 3.4](#))
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 A.[04.9](#).
- 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 if 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.

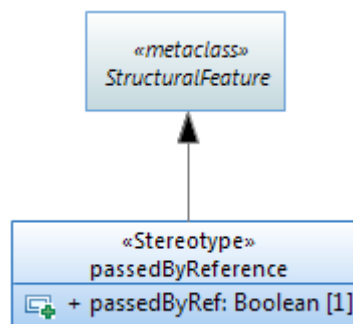


Figure A.0.8: «PassedByReference» Stereotype

- UML/Papyrus defined attribute properties that are not used in the model:
 - Is derived
 - Is derived union

- Is leaf
- Is static
- Is unique
- Visibility

A.4.3 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.

A.4.3.1 Association Notation

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

[Figure A.0.9](#)[Figure 4.9](#) shows a bi-directional navigable association where each object class has a pointer to the other. The role name becomes the name of the corresponding attribute. I.e., in the example: ClassA will have an attribute named “_classBRefList” pointing to ClassB and vice versa.

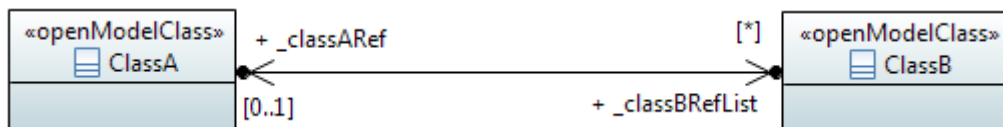


Figure A.0.9: Bidirectional Association Relationship Notation

[Figure A.0.10](#)[Figure A.4.10](#) 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.

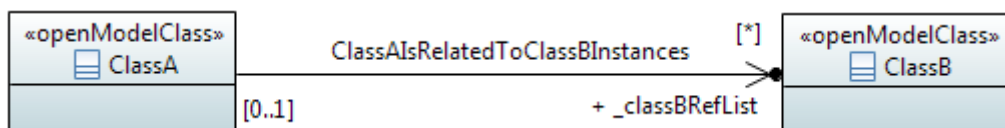


Figure A.0.10: Unidirectional Association Relationship Notation

[Figure A.0.11](#)[Figure A.4.11](#) 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.

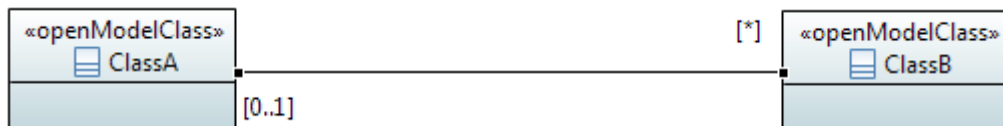


Figure A.0.11: – Non-navigable Association Relationship Notation

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.

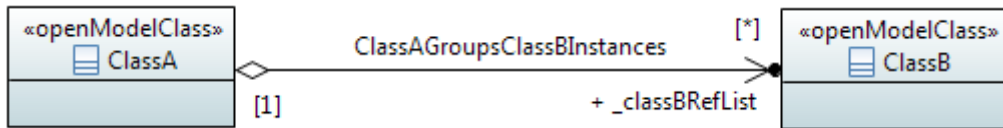


Figure A.0.12: Aggregation Association Relationship Notation

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.

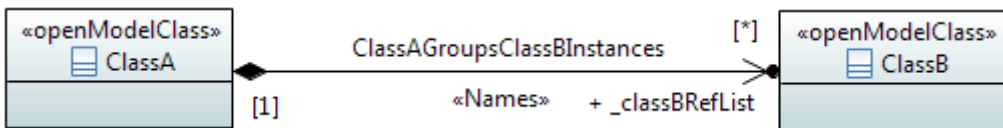


Figure A.0.13: Composite Aggregation Association Relationship Notation

Name	ClassAGroupsClassBInstances																					
Visibility	public																					
Member End	<table border="1"> <tr> <td>Name</td> <td>_classARef</td> </tr> <tr> <td>Owner</td> <td>Association</td> </tr> <tr> <td>Navigable</td> <td><input type="radio"/> true <input checked="" type="radio"/> false</td> </tr> <tr> <td>Aggregation</td> <td>none</td> </tr> <tr> <td>Multiplicity</td> <td>1</td> </tr> </table>	Name	_classARef	Owner	Association	Navigable	<input type="radio"/> true <input checked="" type="radio"/> false	Aggregation	none	Multiplicity	1	<table border="1"> <tr> <td>Name</td> <td>_classBRefList</td> </tr> <tr> <td>Owner</td> <td>Classifier</td> </tr> <tr> <td>Navigable</td> <td><input checked="" type="radio"/> true <input type="radio"/> false</td> </tr> <tr> <td>Aggregation</td> <td>composite</td> </tr> <tr> <td>Multiplicity</td> <td>0..*</td> </tr> </table>	Name	_classBRefList	Owner	Classifier	Navigable	<input checked="" type="radio"/> true <input type="radio"/> false	Aggregation	composite	Multiplicity	0..*
Name	_classARef																					
Owner	Association																					
Navigable	<input type="radio"/> true <input checked="" type="radio"/> false																					
Aggregation	none																					
Multiplicity	1																					
Name	_classBRefList																					
Owner	Classifier																					
Navigable	<input checked="" type="radio"/> true <input type="radio"/> false																					
Aggregation	composite																					
Multiplicity	0..*																					

Figure A.0.14: Papyrus Settings for Composite Aggregation

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

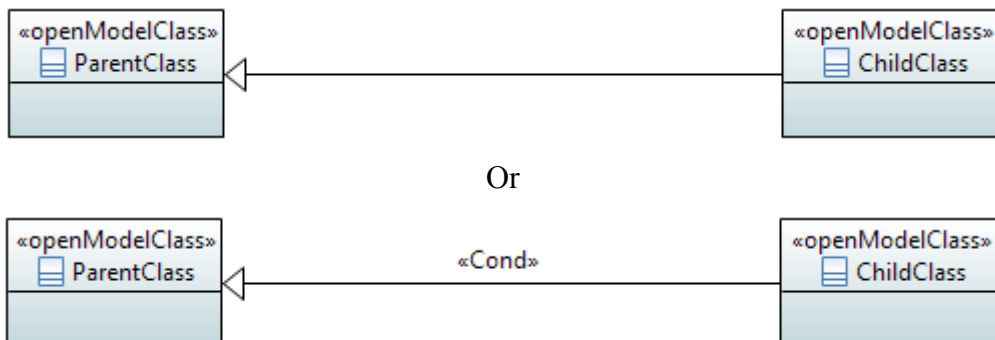


Figure A.0.15: Generalization Relationship Notation (normal and conditional)

“A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation. This means that the complete semantics of the depending elements is either semantically or structurally dependent on the definition of the

supplier element(s)...“, an extract from [\[2\]](#).

A dependency relationship may define naming identified by the «NamedBy» stereotype.

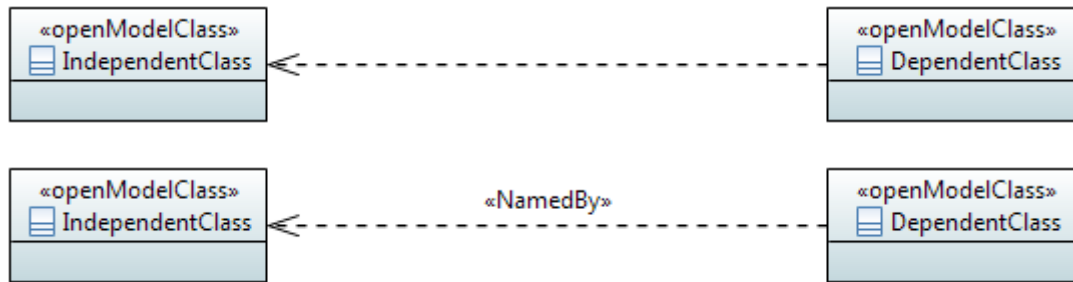


Figure 0.16: Dependency Relationship Notation (normal and naming)

A.4.3.2 Association properties

An association has the following properties:

- **Name**
Follows Upper Camel Case (UCC) and is unique across all association names defined in the whole model.
The format is "<Class1Name><VerbPhrase><Class2Name>" where the verb phrase creates a sequence that is readable and meaningful.
- **Documentation**
Contains a short summary of the usage. The documentation is carried in the “Comments” field in Papyrus.
- **Abstract**
Associations which are just for explanation to the reader of the model are defined as "abstract". Their ends are not navigable and have no role names. Abstract associations must not be taken into account in a protocol specific implementation.
- **Type**
The following types are used:
 - inheritance,
 - simple association,
 - composition,
 - aggregation.
- **Role Name**
Follows Lower Camel Case (LCC) with an underscore “_” prefix and identifies the role that the object plays at this end of the association. The plural form is “_<role name>List”. Only navigable association ends have role names and follow the definitions made for attributes in section [A.04.2](#).
- **Role Type**
The type of the role is fixed to the object class attached to the association end. Therefore it is important to define the type as passedByReference or passedByValue. The «PassedByReference» stereotype identifies that the role (becoming an attribute) that has the stereotype associated, contains only the reference (name, identifier, address) to the referred object instance(s) when being transferred across the interface. Otherwise the role (becoming an attribute) contains the complete information of the object instance(s) when being transferred across the interface.

- Role Multiplicity
Identifies the number of object instances that can participate in an instance of the association.
- Additional 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). Means that the content of the composed classes is part of the parent class and has no opportunity for independent lifecycle. The composed classes are essentially carrying attributes of the parent class where the composite is used to provide grouping of similar properties. The composed classes just provide groups of attributes for the parent class; i.e., they are abstract and cannot be instantiated.

Whereas in an association with a composite end that is not StrictComposite the composed class is a part that has a restricted independent lifecycle. In this case an instance of the composed class can be created and deleted in the context of the parent class and should be represented as a separate instance from the parent in an implementation. This is especially true where there is a recursive composition. It is possible that in some cases the composed instance could move from one parent to another so long as it exists with one parent only at all points of the transaction. This move is not meaningful for a class associated via a StrictComposite association.

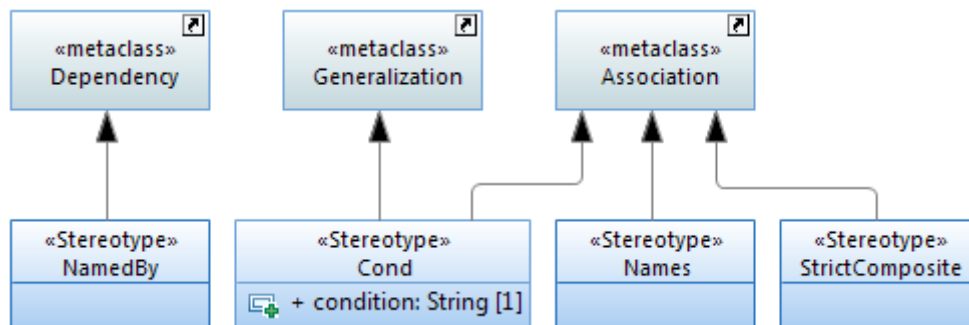


Figure A.0.17: Potential annotations for Associations

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

A.4.4 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 [Error! Reference source not found, Figure 1-2](#).

A.4.4.1 «Interface» Notation

Interfaces are identified by the stereotype «Interface».

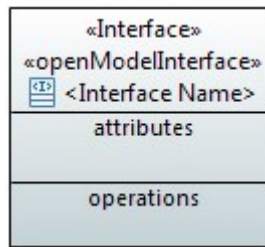


Figure A.0.18: Graphical Notation for «Interface» in the model

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

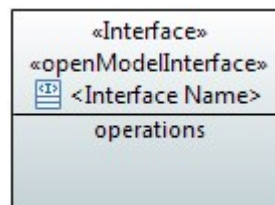




Figure A.0.19: Graphical Notation for «Interface» without Attributes Compartment

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

A.4.4.2 «Interface» Properties

An «Interface»  has the following properties:

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

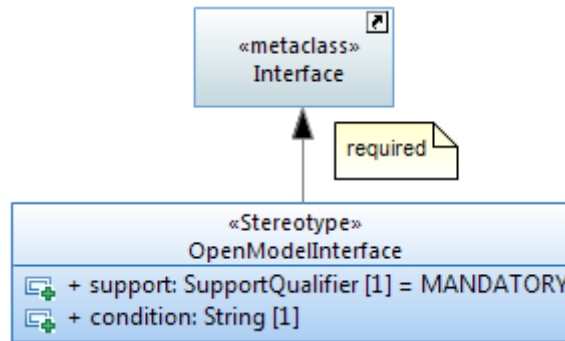



Figure 0.20: «OpenModelInterface» Stereotype

- support
This property qualifies the support of the «Interface» at the management interface. See definition in section [04.9](#).
- condition
This property contains the condition for the condition-related support qualifiers.
- UML/Papyrus defined interface properties that are not used in the model:
 - Is leaf
 - Visibility

A.4.5 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 [Error! Reference source not found. Figure 3-1](#).

A.4.5.1 Operation Notation

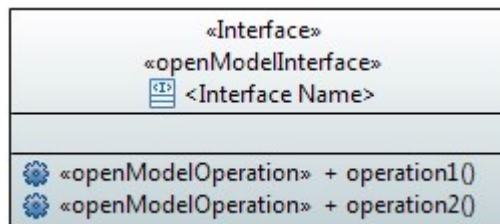


Figure A.0.21: Graphical Notation for «Interface» with Operations

A.4.5.2 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 “Comments” field in Papyrus.
- Pre-condition(s)
This property defines the conditions that have to be true before the operation can be started

(i.e., if not true, the operation will not be started at all and a general “precondition not met” error will be returned, i.e., exception is raised).


- Post-condition(s)
This property defines the state of the system after the operation has been executed (if successful, or if not successful, or if partially successful).
Note that partially successful post-condition(s) can only be defined in case of non-atomic operations.
Note that when an exception is raised, it should not be assumed that the post-condition(s) are satisfied.
- Parameter(s)
See section [04.6](#).
- 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 extents ( **Extension**) by default ({required}) the «metaclass» Operation:

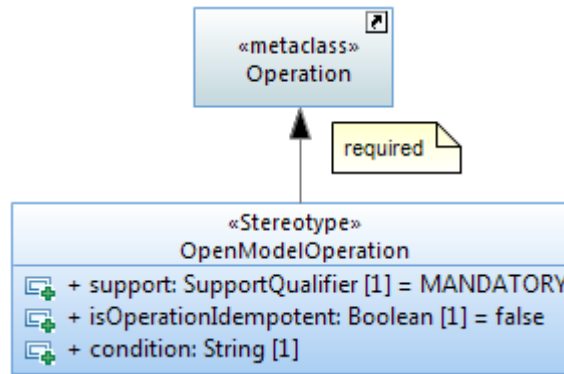


Figure A.0.22: «OpenModelOperation» Stereotype

- isOperationIdempotent (Boolean)
 Defines if the operation is idempotent (true) or not (false).
- support
 This property qualifies the support of the operation at the management interface. See definition in section [04.9](#).
- condition
 This property contains the condition for the condition-related support qualifiers.
- UML/Papyrus defined operation properties that are not used in the model:
 - Is leaf
 - Is query
 - Is static

A.4.6 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 [Error! Reference source not found.](#) [Figure A.3.1](#).

A.4.6.1 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

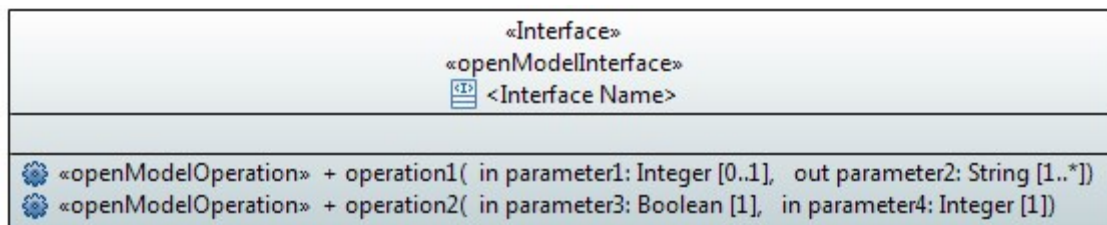



Figure A.0.23: Graphical Notation for «Interface» with Operations and Parameters

A.4.6.2 Parameter Properties

A parameter has the following properties:

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

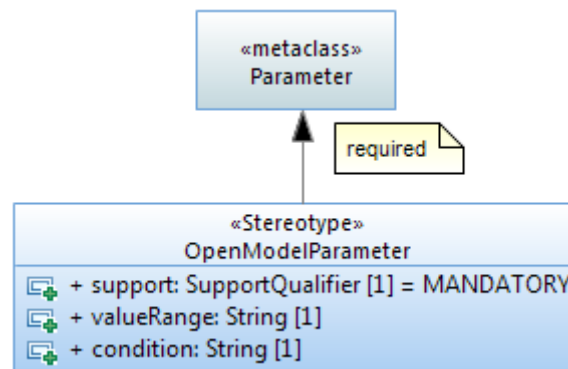


Figure A.0.24: «OpenModelProperty» Stereotype

- 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 A.[04.9](#).
- 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.

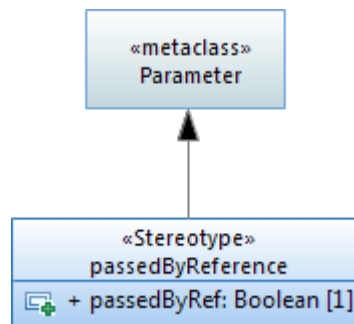


Figure A.0.25: «PassedByReference» Stereotype

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

A.4.7 Notifications

Note: Notifications may only be defined in the purpose-specific modules of the Information Model; see [Error! Reference source not found. Figure 3.1.](#)

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

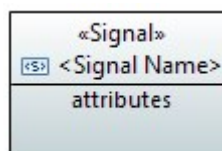


Figure A.0.26: Graphical Notation for «Signal»

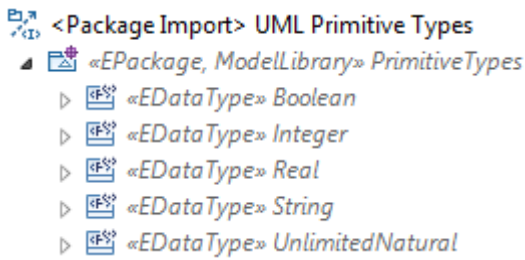
A.4.8 Types

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:



A.4.8.1 Type Notation

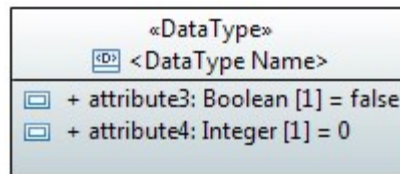


Figure A.0.27: Graphical Notation for «DataType»

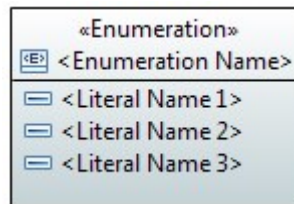


Figure A.0.28: Graphical Notation for «Enumeration»

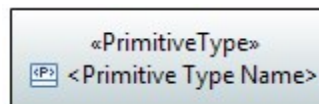


Figure A.0.29: Graphical Notation for «PrimitiveType»

A.4.8.2 Type Properties

A type has the following properties:

- Category
Three categories are used in the model:
 - dataType
 - enumeration
 - primitive.
- Name
Follows Upper Camel Case (UCC) and is unique across all data type names defined in the whole model.
- Documentation
Contains a short summary of the usage. The documentation is carried in the “Comments” field in Papyrus.

- Data type attributes (only in dataTypes)
Follow the definitions made for attributes in section [04.2](#) 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.
 - Exception
This stereotype defines a data type used for an operation exception.

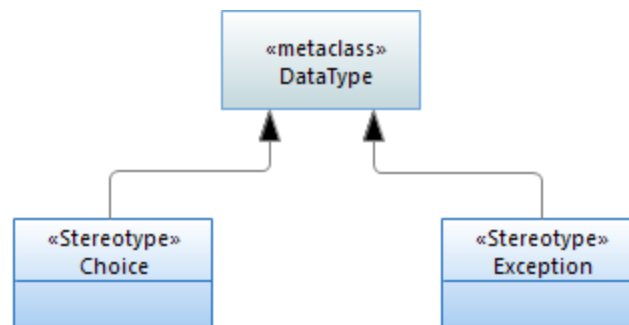


Figure A.4.30: Potential annotations for Data Types

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

A.4.9 Qualifiers

This clause defines the qualifiers applicable for model elements specified in this document, e.g. the «openModelClass» (see section [A.04.1.2](#)), and the «OpenModelAttribute» (see section [A.04.2.2](#)). The qualifications are M, O, CM, CO, C and 'SS'. Their meanings are specified in this section. This type of qualifier is called Support Qualifier.

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

item shall be supported. If all the specified constraints are met and are related to optional, then the item may be supported. If all the specified constraints are met and are related to "no support", then the item shall not be supported.

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

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

A.5 UML Profile Definitions

A.5.1 Additional Properties Definitions

Section 04 has already described the additional properties for each UML artifact. All defined stereotypes are shown as an overview in [Figure A.0.6](#) [Figure A.5.1](#) and [Table A.0.2](#) [Table A.5.1](#) below.

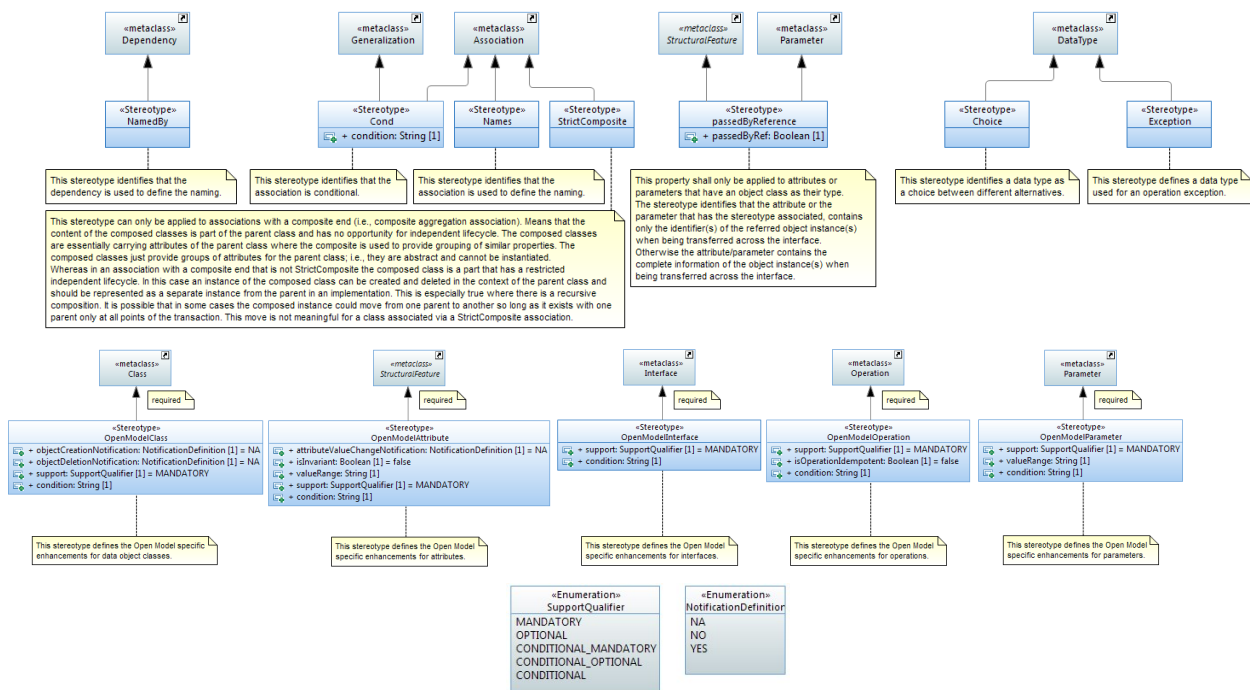


Figure A.0.64: UML Artifact «Stereotypes»

Table A.0.24: UML Artifact Properties defined in complex «Stereotypes»

Stereotype	Name of property	Type	Allowed values	Default value	Associated to metaclass
openModelClass	objectCreation Notification	enumeration	NO, YES, NA	NA	Class
	objectDeletion Notification	enumeration	NO, YES, NA	NA	
	support	enumeration	MANDATORY OPTIONAL CONDITIONAL_ MANDATORY_ CONDITIONAL_ OPTIONAL CONDITIONAL	MANDATORY	
	condition	string		none	
OpenModelAttribute	attributeValue Change Notification	enumeration	NO, YES, NA	NA	Property
	isInvariant	Boolean	true/false	false	
	valueRange	string		NA	
	support	enumeration	MANDATORY OPTIONAL CONDITIONAL_ MANDATORY_ CONDITIONAL_ OPTIONAL CONDITIONAL	MANDATORY	
OpenModelOperation	isOperationIdempotent	Boolean	true/false	false	Operation
	support	enumeration	MANDATORY OPTIONAL CONDITIONAL_ MANDATORY_ CONDITIONAL_ OPTIONAL CONDITIONAL	MANDATORY	
	condition	string		none	
OpenModelParameter	valueRange	string		NA	Parameter
	support	enumeration	MANDATORY OPTIONAL CONDITIONAL_ MANDATORY_ CONDITIONAL_ OPTIONAL CONDITIONAL	MANDATORY	
	condition	string		none	

A.5.2 Modeling Lifecycle Definitions

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

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

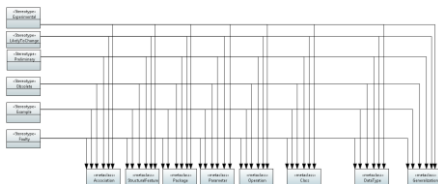


Figure A.0.72: UML Artefact Lifecycle «Stereotypes»

A.5.3 Profiles File



Date: Feb. 20th 2015

A.6 Recommended Modelling Patterns

A.6.1 File Naming Conventions

For Further Study

A.6.2 Model Structure

For Further Study

A.6.3 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 [06.3.1](#) below).

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

→ 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).

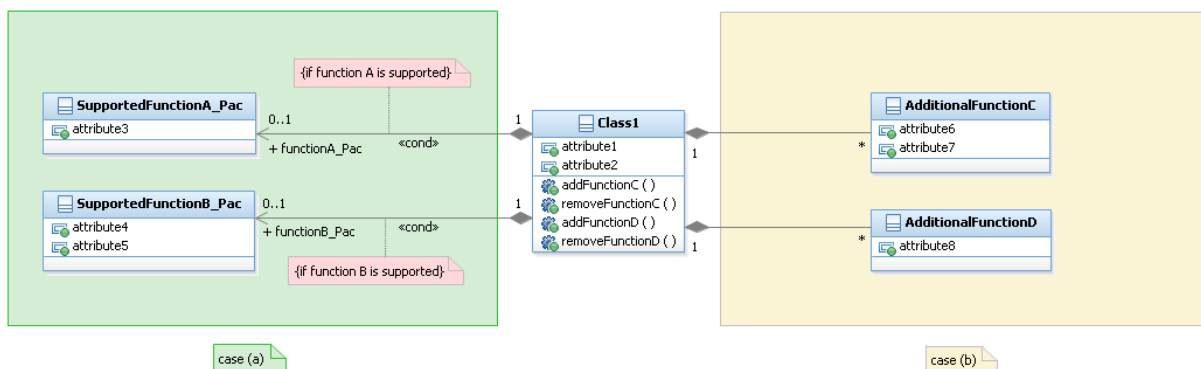


Figure A.0.1: Flexible Attribute Assignment to Object Classes

A.6.3.1 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.

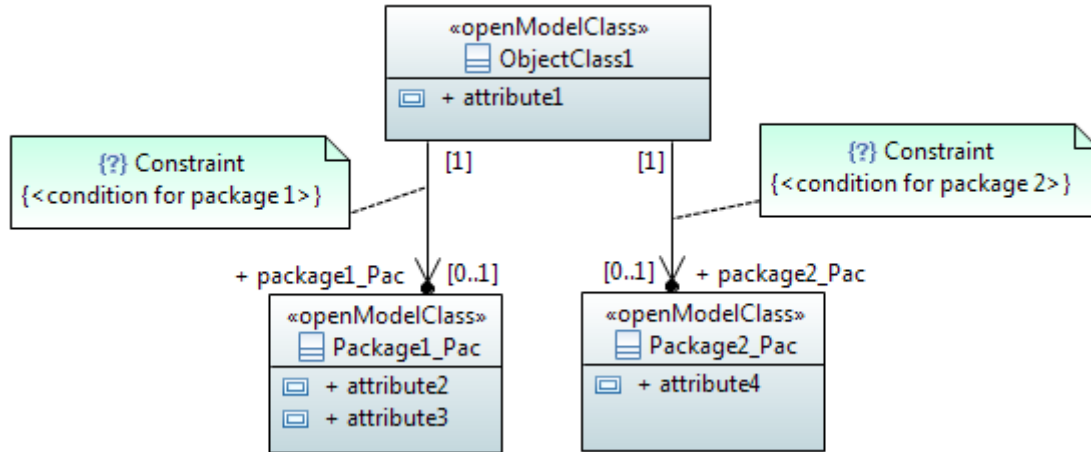


Figure A.0.2: Enhancing object classes using conditional packages

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 follow the same rules as defined for attributes; i.e., LCC. The name ends with the suffix "_Pac".

A.6.4 Use of XOR/Choice

A.6.4.1 Xor constraint

A.6.4.1.1 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.

A.6.4.1.2 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.

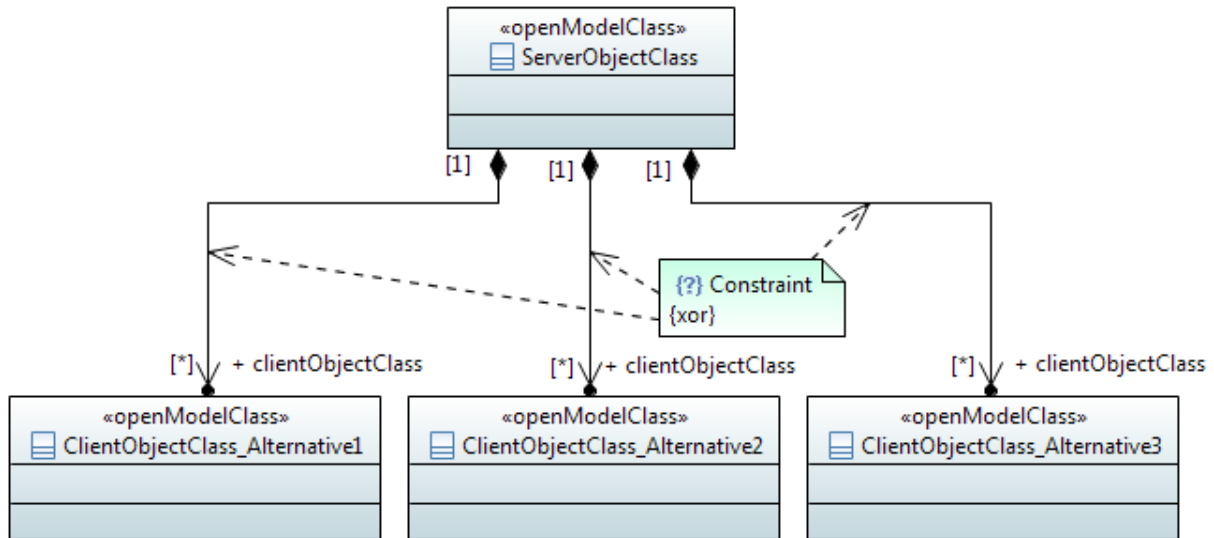


Figure A.0.3: {xor} notation

A.6.4.1.3 Name style

It has no name so there is no name style.

A.6.4.2 «Choice»

A.6.4.2.1 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 [06.4.1](#)).

A.6.4.2.2 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 realised.

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

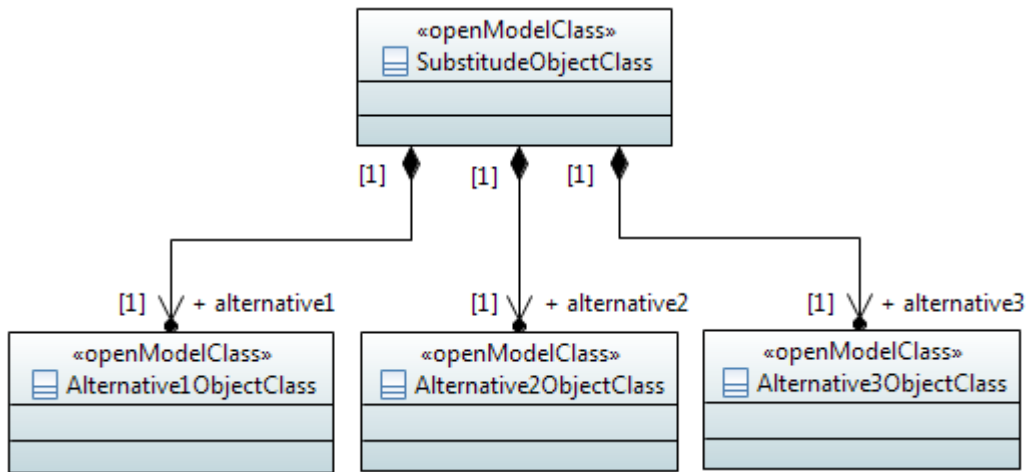


Figure A.0.4: Information model element example using «Choice» notation

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 realised.

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

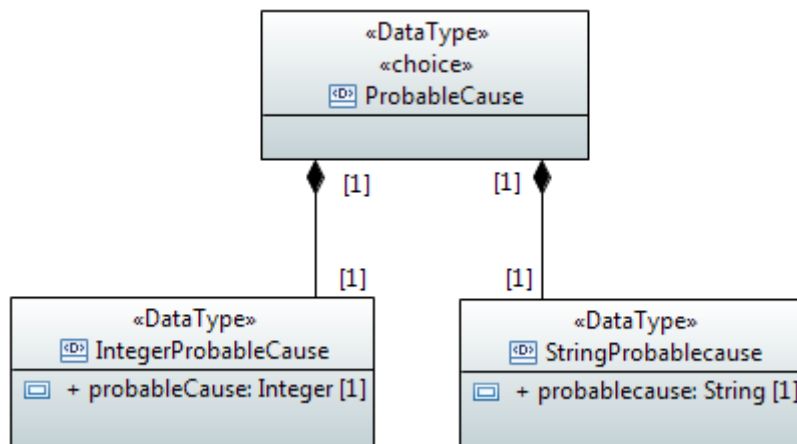


Figure A.0.5: Operations model element example using «Choice» notation

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

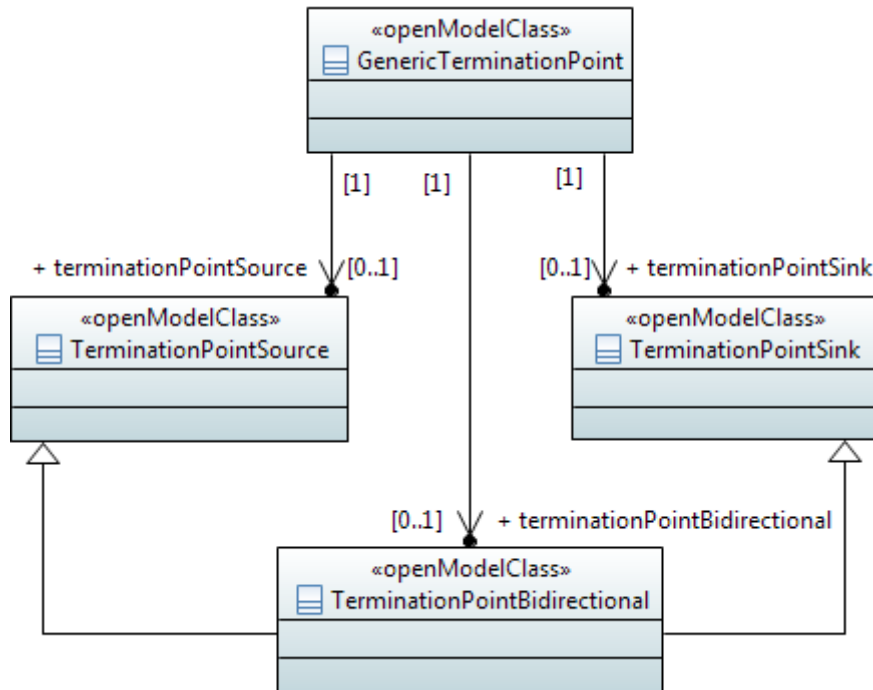


Figure A.0.6: Sink/source/bidirectional termination points example using «Choice» notation

A.6.4.2.3 Name style

For «Choice» name, use the same style as «openModelClass» (see [04.1.2](#)).

A.6.5 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 [f2](#) 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.

A.6.6 Style Guides

For Further Study

Bibliography

- [b-3GPP/TMF-JWG] 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)
- [b-ONF-CIM-Core-Model] *ONF TR-512 ONF CIM Core Information Model*
- [b-ONF-CIM-Overview] *ONF TR-513 ONF CIM Overview*
- [b-ONF-UML-Guide] ONF TR-514 *UML Modelling Guidelines*
- [b-ONF-Papyrus-Guide] *ONF TR-515 ONF Papyrus Guidelines*
- [b-UML] Unified Modeling Language™ (UML®) (<http://www.uml.org/>)
- [b-Papyrus] Papyrus Eclipse UML Modelling Tool (<https://www.eclipse.org/papyrus/>)
- [b-TMF TR225] *TM Forum TR225, Logical Resource: Network Function Model*
-