ETSI TS 132 156 V11.0.2 (2013-04)

Technical Specification

Universal Mobile Telecommunications System (UMTS); Telecommunication management; Fixed Mobile Convergence (FMC) model repertoire (3GPP TS 32.156 version 11.0.2 Release 11)

A GLOBAL INITIATIVE

Reference

RTS/TSGS-0532156vb02

Keywords

UMTS

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:
http://portal.etsi.org/chaircor/ETSI_support.asp

Copyright Notification

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://ipr.etsi.org).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities, UMTS identities or GSM identities. These should be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under http://webapp.etsi.org/key/queryform.asp.

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

The JWG on Model Alignment work has chosen UML to capture behaviour of systems/entities under management.

UML provides a rich set of concepts, notations and model elements to model distributive systems. Usage of all UML notations and model elements is not necessary for the purpose of JWG Model Alignment work. This paper documents the necessary and sufficient set of UML notations and model elements, including the ones built by the UML extension mechanism <<stereotype>>, for use by JWG Model Alignment work. Collectively, this set of notations and model elements is called the FMC (developed by the Converged Management of Fixed/Mobile Networks project) Model Repertoire.

JWG Model Alignment specifications shall employ the UML notation and model elements of this repertoire. In the course of the JWG Model Alignment work, JWG Model Alignment group may modify (add, delete, modify) UML notation and model elements of this repertoire when necessary.

2 References

[1] OMG "Unified Modelling Language (OMG UML), Infrastructure", Version 2.3.

[2] OMG "Unified Modelling Language (OMG UML), Superstructure", Version 2.3.

[3] 3GPP TS 32.300: "3rd Generation Partnership Projects; Technical Specification Group Services and System Aspects; Telecommunication management; Configuration Management (CM); Name convention for Managed Objects".

[4] 3GPP TS 23.002: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Network architecture".

[5] 3GPP TS 32.107: "Fixed Mobile Convergence (FMC) Federated Network Information Model (FNIM)".

[6] 3GPP TS 28.620: "Fixed Mobile Convergence (FMC) Federated Network Information Model (FNIM) Umbrella Information Model (UIM)".

[7] ITU-T X.680,"OSI networking and system aspects – Abstract Syntax Notation One (ASN.1)".

[8] ITU-T X.501,"Information technology – Open Systems Interconnection – The Directory: Models".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of this document, the following definitions and abbreviations apply. For definitions and abbreviations not found here, see also Fixed Mobile Convergence (FMC) Federated Network Information Model (FNIM) 0, Fixed Mobile Convergence (FMC) Federated Network Information Model (FNIM) Umbrella Information Model 0.

Distinguished Name: See 3GPP TS 32.300 [3].

Naming attribute: It is a class attribute that holds the class instance identifier. See attribute `id` of *Top_* [6]. See examples of naming attribute in 3GPP TS 32.300 [3].

Lower Camel Case: It is the practice of writing compound words in which the words are joined without spaces. Initial letter of all except the first word shall be capitalized. Examples: ‘managedNodeIdentity’ and ‘minorDetails’ are the LCC for “managed node identity” and “minor details” respectively.

Upper Camel Case: It is the Lower Camel Case except that the first letter is capitalised. Examples: ‘ManagedNodeIdentity’ and ‘MinorDetails’ are the UCC for “managed node identity” and “minor details” respectively.

Well Known Abbreviation: An abbreviation can be used as the modelled element name or as a component of a modelled element name. The abbreviation, when used in such manner, must be documented in the same document where the modelled element is defined.

3.2 Abbreviations

CM	Conditional Mandatory
CO	Conditional Optional
DN	Distinguished Name
FMC	Fixed Mobile Convergence
FNIM	Federated Network Information Model
IOC	Information Object Class
IRP	Integration Reference Point
JWG	(3GPP/TM Forum) Joint Working Group
LCC	Lower Camel Case
M	Mandatory
NA	Not Applicable
NRM	Network Resource Model
O	Optional
OMG	Object Management Group
UCC	Upper Camel Case
UIM	Umbrella Information Model
UML	Unified Modelling Language (OMG)
WKA	Well Known Abbreviation

4 Requirements

The UML notations and model elements captured in this repertoire shall be used to model behaviours of the systems/entities specified by the JWG Resource Model Alignment work such as the Umbrella Information Model (UIM) 0 of the FNIM discussed in Converged Management of Fixed/Mobile Network project.

5 Model Elements and Notations

5.1 General

Note that the graphical notation in this document is only used to represent particular model elements. Although the graphical notation is a correct representation of the model element, it may not be a valid representation of a UML class diagram.

The examples used in this document are for illustration purposes only and may or may not exist in specifications.

UML properties not described in this document shall not be used in specifications based on this repertoire.

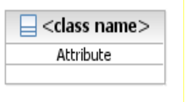
5.2 Basic model elements

UML has defined a number of basic model elements. This sub-clause lists the subset selected for use in specifications based on this repertoire. The semantics of these selected basic model elements are defined in [1].

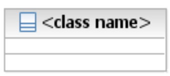
For each basic model element listed, there are three parts. The first part contains its description. The second part contains its graphical notation examples and the third part contains the rule, if any, recommended for labelling or naming it.

The graphical notation has the following characteristics:

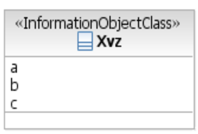
→ Section 7.2.7 of [2] specifies "A class is often shown with three compartments. The middle compartment holds a list of attributes while the bottom compartment holds a list of operations" and "Additional compartments may be supplied to show other details". This repertoire only allows the use of the name (top) compartment and attribute (middle) compartment. The operation (bottom) compartment may be present but is always empty, as shown in the figure below.



→ Classes may or may not have attributes. The graphical notation of a class may show an empty attribute (middle) compartment even if the class has attributes, as shown in figure below.



→ The visibility symbol shall not appear along with the class attribute, as shown below.



→ The use of the decoration, i.e. the symbol in the name (top) compartment, is optional.

5.2.1 Attribute

5.2.1.1 Description

It is a typed element representing a property of a class. See 10.2.5 Property of [1].

An element that is typed implies that the element can only refer to a constrained set of values.

See 10.1.4 Type of [1] for more information on type.

See 5.3.4 and 5.4.3 for predefined data types and user-defined data types that can apply type information to an element.

The following table captures the properties of this modelled element.

Table 1: Attribute properties

Property name	Description	Legal values
documentation	Contains a textual description of the attribute. Should refer (to enable traceability) to the specific requirement.	Any
isOrdered	For a multi-valued multiplicity; this specifies if the values of this attribute instance are sequentially ordered. See section 7.3.44 and its Table 7.1 of [2].	True, False (default)
isUnique	For a multi-valued multiplicity, this specifies if the values of this attribute instance are unique (i.e., no duplicate attribute values). See section 7.3.44 and its Table 7.1 of [2].	True (default), False
isReadable	Specifies that this attribute can be read by the manager.	True (default), False
isWritable	Specifies that this attribute can be written by the manager under the conditions specified in Annex B.	True, False (default)
type	Refers to a predefined (see section 5.4.3) or user defined data type (see section 5.3.4. See also section 7.3.44 of 0, inherited from StructuralFeature.	NA
isInvariant	Attribute value is set at object creation time and cannot be changed under the conditions specified in Annex B.	True, False (default)
allowedValues	Identifies the values the attribute can have.	Dependent on type
isNotifyable	Identifies if a notification shall be sent in case of a value change.	True (default), False
defaultValue	Identifies a value at specification time that is used at object creation time under conditions defined in Annex B.	Dependent on type
multiplicity	Defines the number of values the attribute can simultaneously have. See section 7.3.44 of 0; inherited from StructuralFeature.	See 5.2.8 Default is 1
isNullable	Identifies if an attribute can carry no information. The implied meaning of carrying "no information" is context sensitive and is not defined in this Model Repertoire.	True, False (default)
supportQualifier	Identifies the required support of the attribute. See also section 6.	M, O (default), CM, CO, C

5.2.1.2 Example

This example shows three attributes, i.e., `a`, `b` and `c`, listed in the attribute (the second) compartment of the class `Xyz`.

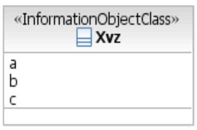


Figure 1: Attribute notation

5.2.1.3 Name style

An attribute name shall use the LCC style.

Well Known Abbreviation (WKA) is treated as a word if used in a name. However, WKA shall be used as is (its letter case cannot be changed) except when it is the first word of a name; and if so, its first letter must be in lower case.

5.2.2 Association relationship

5.2.2.1 Description

It shows a relationship between two classes and describes the reasons for the relationship and the rules that might govern that relationship.

It has ends. Its end, the association end(s), specifies the role that the object at one end of a relationship performs. Each end of a relationship has properties that specify the role (see 5.2.9), multiplicity (see 5.2.8), visibility and navigability (see the arrow symbol used in Figure 3: Unidirectional association relationship notation) and may have constraints. Note that visibility shall not be used in models based on this Repertoire (see bullet 3 of 5.1).

See 7.3.3 Association of [2].

Three examples below show a binary association between two model elements. The association can include the possibility of relating a model element to itself.

The first example (Figure 2) shows a bi-directional navigable association in that each model element has a pointer to the other. The second example (Figure 3) shows a unidirectional association (shown with an open arrow at the target model element end) in that only the source model element has a pointer to the target model element and not vice-versa. The third example (Figure 4) shows a bi-directional non-navigable association in that each model element does not have a pointer to the other; i.e., such associations are just for illustration purposes.

5.2.2.2 Example

An association shall have an indication of cardinality (see 5.2.8).

It shall, except the case of non-navigable association, have an indication of the role name (see 5.2.9). The model element involved in an association is said to be "playing a role" in that association. The role has a name such as `+aClass` in the first example below. Note that the "+" character in front of the role name, indicating the visibility, is ignored.

Figure 2: Bidirectional association relationship notation

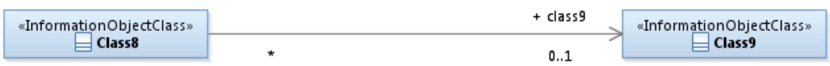


Figure 3: Unidirectional association relationship notation

Figure 4: Non-navigable association relationship notation

Note that some tools do not use arrows in the UML graphical representation for bidirectional associations. Therefore, absence of arrows is not, but absence of role names is, an indication of a non-navigable association.

5.2.2.3 Name style

An Association can have a name. Use of Association name is optional. Its name style is LCC style.

A role name shall use the LCC style.

5.2.3 Aggregation association relationship

5.2.3.1 Description

It shows a class as a part of or subordinate to another class.

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.

See 7.3.2 AggregationKind (from Kernel) of [2].

5.2.3.2 Example

A hollow diamond attached to the end of a relationship is used to indicate an aggregation. The diamond is attached to the class that is the aggregate. The aggregation association shall have an indication of cardinality at each end of the relationship (see 5.2.8).

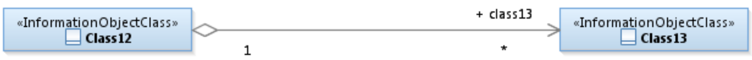


Figure 5: Aggregation association relationship notation

5.2.3.3 Name style

An Association can have a name. Use of Association name is optional. Its name style is LCC.

5.2.4 Composite aggregation association relationship

5.2.4.1 Description

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.

A composite aggregation shall contain a description of its use.

See 7.3.3 Association (from Kernel) of [2].

5.2.4.2 Example

A filled diamond attached to the end of a relationship is used to indicate a composite aggregation. The diamond is attached to the class that is the composite. The composition association shall have an indication of cardinality at each end of the relationship (see 5.2.8).

Figure 6: Composite aggregation association relationship notation

5.2.4.3 Name style

An Association can have a name. Use of Association name is optional. Its name style is LCC.

5.2.5 Generalization relationship

5.2.5.1 Description

It indicates a relationship in which one class (the child) inherits from another class (the parent).

See 7.3.20 Generalization of [2].

5.2.5.2 Example

This example shows a generalization relationship between a more general model element (the `IRPAgent`) and a more specific model element (the `IRPAgentVendorA`) that is fully consistent with the first element and that adds additional information.

Figure 7: Generalization relationship notation

5.2.5.3 Name style

It has no name so there is no name style.

5.2.6 Dependency relationship

5.2.6.1 Description

"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 7.3.12 Dependency of [2].

5.2.6.2 Example

This example shows that the `BClass` instances have a semantic relationship with the `AClass` instances. It indicates a situation in which a change to the target element (the `AClass` in the example) will require a change to the source element (the `BClass` in the example) in the dependency.

Figure 8: Dependency relationship notation

5.2.6.3 Name style

An Association can have a name. Use of Association name is optional. Its name style is LCC.

5.2.7 Comment

5.2.7.1 Description

A comment is a textual annotation that can be attached to a set of elements.

See 7.3.9 Comment (from Kernel) from [2].

5.2.7.2 Example

This example shows a comment, as a rectangle with a "bent corner" in the upper right corner. It contains text. It appears on a particular diagram and may be attached to zero or more modelling elements by dashed lines.

Figure 9: Comment notation

5.2.7.3 Name style

It has no name so there is no name style.

5.2.8 Multiplicity, a.k.a. cardinality in relationships

5.2.8.1 Description

"A multiplicity is a definition of an inclusive interval of non-negative integers beginning with a lower bound and ending with a (possibly infinite) upper bound. A multiplicity element embeds this information to specify the allowable cardinalities for an instantiation of this element…", an extract from 7.3.32 MultiplicityElement of [2].

Table 2: Multiplicity-string definitions

Multiplicity	Explanation
1	Attribute has one attribute value.
m	Attribute has *m* attribute values.
0..1	Attribute has zero or one attribute value.
0..*	Attribute has zero or more attribute values.
*	Attribute has zero or more attribute values.
1..*	Attribute has at least one attribute value.
m..n	Attribute has at least *m* but no more than *n* attribute values.

The use of "0..n" is not recommended although it has the same meaning as "0..*" and "*".

The use of a standalone symbol zero (0) is not allowed.

5.2.8.2 Example

This example shows a multiplicity attached to the end of an association path. The meaning of this multiplicity is one to many. One `Network` instance is associated with zero, one or more `SubNetwork` instances. Other valid examples can show the "many to many" relationship.

Figure 10: Cardinality notation

The cardinality zero is not used to indicate the IOC's so-called "transient state" characteristic. For example, it is not used to indicate that the instance is not yet created but it is in the process of being created. The cardinality zero will not be used to indicate this characteristic since such characteristic is considered inherent in all IOCs. All IOCs defined are considered to have such inherent "transient state" characteristics.

Note that the use of "0..*", "0..n" or '*' means "zero to many". The use of "0..*" is recommended. The following table shows some valid examples of multiplicity.

Table 3: Multiplicity-string examples

Multiplicity	Explanation
1	Attribute has exactly one attribute value.
5	Attribute has exactly 5 attribute values.
0..1	Attribute has zero or one attribute value.
0..*	Attribute has zero or more attribute values.
1..*	Attribute has at least one attribute value.
4..12	Attribute has at least 4 but no more than 12 attribute values.

5.2.8.3 Name style

It has no name so there is no name style.

5.2.9 Role

5.2.9.1 Description

It indicates navigation, from one class to another class, involved in an association relationship. A role is named. The direction of navigation is to the class attached to the end of the association relationship with (or near) the role name.

The use of role name in the graphical representation is mandatory for bidirectional and unidirectional association relationship notations (see Figure 2: Bidirectional association relationship notation and Figure 3: Unidirectional association relationship notation). Role name shall not be used in non-navigable association relationship notation (see Figure 4: Non-navigable association relationship notation).

A role at the navigable end of a relationship becomes (or is mapped into) an attribute (called role-attribute) in the source class of the relationship. Therefore roles have the same behaviour (or properties) as attributes. See *Table 1: Attribute properties*.

The role-attribute shall have all properties defined for attributes in section 5.2.1 Attribute and in addition the following properties:

→ Passed by id;
Identifies if the role (navigable association end that points to an object) contains just a pointer to the object (passed by id = true) or contains the whole object instance itself (passed by id = false).
Legal values: true, false; default value = "false".

5.2.9.2 Example

This example shows that a `Person` (say instance John) is associated with a `Company` (say whose DN is "Company=XYZ"). We navigate the association by using the opposite association-end such that John's `Person.theCompany` would hold the DN, i.e. "Company=XYZ".

Figure 11: Role notation

5.2.9.3 Name style

A role has a name. Use noun for the name. The name style follows the attribute name style; see section 5.2.1.3.

5.2.10 Xor constraint

5.2.10.1 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 7.3.10 Constraint (from Kernel) of [2].

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.

5.2.10.2 Example

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

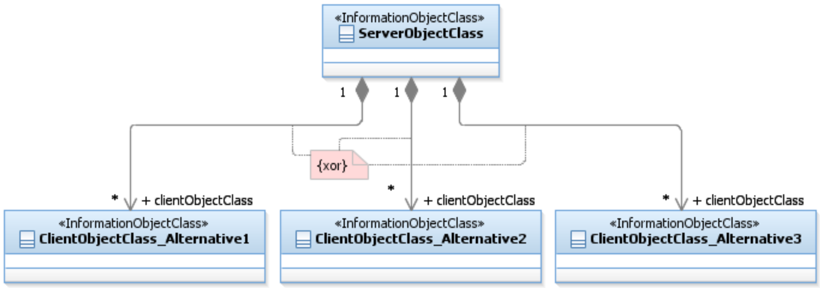


Figure 12: {xor} notation

5.2.10.3 Name style

It has no name so there is no name style.

5.3 Stereotype

Sub-clause 5.1 listed the UML defined basic model elements. UML defined a stereotype concept allowing the specification of simple or complex user-defined model elements.

This sub-clause lists all allowable stereotypes for this repertoire.

For each stereotype model element listed, there are three parts. The first part contains its description. The second part contains its graphical notation examples and the third part contains the rule, if any, recommended for labelling or naming it.

5.3.1 <<ProxyClass>>

5.3.1.1 Description

It is a form or template representing a number of <<InformationObjectClass>>. It encapsulates attributes, links, methods (or operations), and interactions that are present in the represented <<InformationObjectClass>>.

The semantics of a <<ProxyClass>> is that all behaviour of the <<ProxyClass>> is present in the represented <<InformationObjectClass>>. Since this class is simply a representation of other classes, this class cannot define its own behaviour other than those already defined by the represented <<InformationObjectClass>>.

A particular <<InformationObjectClass>> can be represented by zero, one or more <<ProxyClass>>. For example, the ManagedElement <<InformationObjectClass>> can have MonitoredEntity <<ProxyClass>> and ManagedEntity <<ProxyClass>>.

The attributes of the <<ProxyClass>> are accessible by the source entity that has an association with the <<ProxyClass>>.

5.3.1.2 Example

This shows a <<ProxyClass>> named `MonitoredEntity`. It represents (or its constraints is that it represents) all NRM <<InformationObjectClass>> (e.g. `GgsnFunction` <<InformationObjectClass>>) whose instances are being monitored for alarm conditions. It is mandatory to use a Note to capture the constraint.

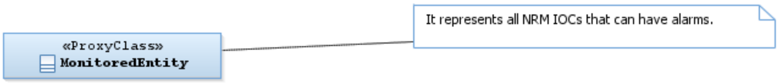


Figure 13: <<ProxyClass>> notation

See Annex A for more examples that use <<ProxyClass>>.

5.3.1.3 Name style

For <<ProxyClass>> name, use the same style as <<InformationObjectClass>> (see 5.3.2).

5.3.2 <<InformationObjectClass>>

5.3.2.1 Description

The <<InformationObjectClass>> is identical to UML *class* except that it does not include/define methods or operations.

A UML *class* represents a capability or concept within the system being modelled. Classes have data structure and behaviour and relationships to other elements.

This class can inherit from zero, one or multiple classes (multiple inheritances).

See more on UML *class* in 10.2.1 of [1].

5.3.2.2 Example

This example shows an `AbcFunction` <<InformationObjectClass>>.

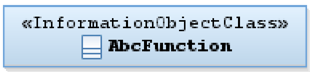


Figure 14: <<InformationObjectClass>> notation

The following table captures the properties of this modelled element.

Table 4: <<InformationObjectClass>> properties

Property name	Description	Legal values
documentation	Contains a textual description of this modelled element. Should refer (to enable traceability) to a specific requirement.	Any
isAbstract	Indicates if the class can be instantiated or is just used for inheritance.	True, False (default)
isNotifyable	Identifies the list of the supported notifications.	List of names of notification
supportQualifier	Identifies the required support of the object class. See also section 6.	M, O (default), CM, CO, C

5.3.2.3 Name style

The name shall use UCC style. The name shall end with an underscore if it is an abstract class in the UIM. The name must not end with an underscore if it is a concrete class.

WKA is treated as a word if used in a name. However, WKA shall be used as is (its letter case cannot be changed) except when it is the first word of the name; and if so, its first letter must be in upper case.

Embedded underscore is not allowed except the name is for an Association class (see 5.4.1.)

5.3.3 <<names>>

5.3.3.1 Description

The <<names>> is modelled by a composition association where both ends are non-navigable. The source object class is the composition and the target object class is the component. The target instance is uniquely identifiable, within the namespace of the source entity, among all other targeted instances of the same target class and among other targeted instances of other classes that have the same <<names>> composition with the source.

The source class and target class shall each has its own naming attribute.

The composition aggregation association relationship is used as the act of name containment providing a semantic of a whole-part relationship between the domain and the named elements that are contained, even if only by name. From the management perspective access to the part is through the whole. Multiplicity shall be indicated at both ends of the relationship.

A target instance can not have multiple <<names>> with multiple sources, i.e. a target instance can not participate in or belong to multiple namespaces.

5.3.3.2 Example

This shows that all instances of `Class4` are uniquely identifiable within a `Class3` instance's namespace.

Figure 15: <<names>> notation

5.3.3.3 Name style

It has no name so there is no name style.

5.3.4 <<dataType>>

5.3.4.1 Description

It represents the general notion of being a data type (i.e. a type whose instances are identified only by their values) whose definition is defined by user (e.g. specification authors).

This repertoire uses two kinds of data types: predefined data types and user-defined data types. The former is defined in sub-clause 5.4.3. The latter is defined by the specifications authors using this <<dataType>> model element.

The user-defined data types support the modelling of structured data types (see <<dataType>> notations in 5.3.4.2). When user-defined or predefined data type is used to apply type information to a class attribute (see 5.2.1), the data type name is shown along with the class attribute. See user example of <<dataType>> in 5.3.4.2.

5.3.4.2 Example

The following examples are two user-defined data types. The left-most is named `PlmnId` that consists of Mobile Country Code (MCC) and Mobile Network Code (MNC), whose types are the predefined data types in 5.4.3. The right-most is named `Xyz` that consists of two predefined data types (i.e., `String`, `Integer` and one user-defined data type `PlmnId`.

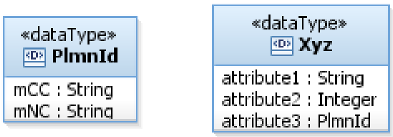


Figure 16: <<dataType>> notations

The following example shows a `ZClass` using two user-defined data types and two predefined data types.

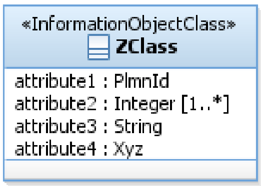


Figure 17: Usage example of <<dataType>>

5.3.4.3 Name style

For <<dataType>> name, use the same style as <<InformationObjectClass>> (see 5.3.2).

For <<dataType>> attribute, use the same style as Attribute (see 5.2.1).

5.3.5 <<enumeration>>

5.3.5.1 Description

An enumeration is a data type. It contains sets of named literals that represent the values of the enumeration. An enumeration has a name.

See 10.3.2 Enumeration of [1].

5.3.5.2 Example

This example shows an enumeration model element whose name is `Account` and it has four enumeration literals. The upper compartment contains the keyword <<enumeration>> and the name of the enumeration. The lower compartment contains a list of enumeration literals.

Note that the symbol to the right of <<enumeration>> Account in the figure below is a feature specific to a particular modelling tool. It is recommended that modelling tool features should be used when appropriate.

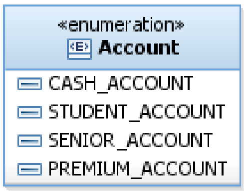


Figure 18: <<enumeration>> notation

5.3.5.3 Name style

For <<enumeration>> name, use the same style as <<InformationObjectClass>> (see 5.3.2).

For <<enumeration>> attribute (the enumeration literal), use the following rules:

- Enumeration literal is composed of one or more words of upper case characters. Words are separated by the underscore character.

5.4 Others

5.4.1 Association class

5.4.1.1 Description

An association class is an association that also has class properties (or a class that has association properties). Even though it is drawn as an association and a class, it is really just a single model element.

See 7.3.4 AssociationClass of [2].

Association classes are appropriate for use when an «InformationObjectClass» needs to maintain associations to several other instances of «InformationObjectClass» and there are relationships between the members of the associations within the scope of the "containing" «InformationObjectClass». For example, a namespace maintains a set of bindings, a binding ties a name to an identifier. A NameBinding «InformationObjectClass» can be modelled as an Association Class that provides the binding semantics to the relationship between an identifier and some other «InformationObjectClass» such as Object in the figure. This is depicted in the following figure.

5.4.1.2 Example

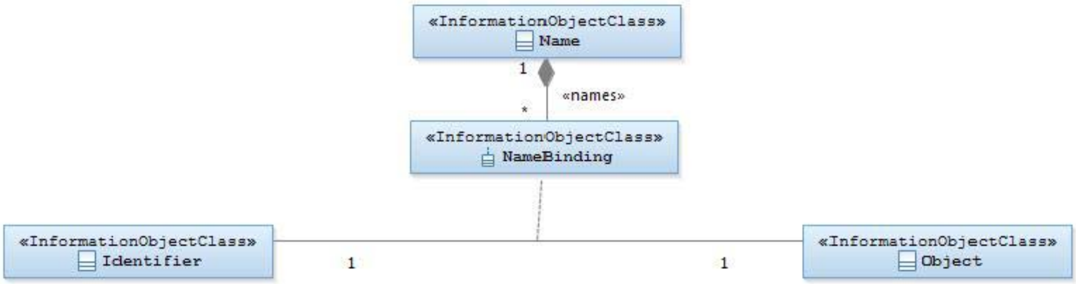


Figure 19: Association class notation

5.4.1.3 Name style

The name shall use the same style as in <<InformationObjectClass>> (see 5.3.2.3).

5.4.2 Abstract class

5.4.2.1 Description

It specifies a special kind of <<InformationObjectClass>> as the general model element involved in a generalization relationship (see 5.2.5). An abstract class cannot be instantiated.

This modelled element has the same properties as class. See 5.3.2.

5.4.2.2 Example

This shows that `Class5_` is an abstract class. It is the base class for `SpecializedClass5`.

Figure 20: Abstract class notation

5.4.2.3 Name style

For abstract class name, use the same style as <<InformationObjectClass>> (see 5.3.2) and its last character shall be an underscore. Furthermore, the name shall be in italics.

5.4.3 Predefined data types

5.4.3.1 Description

It represents the general notion of being a data type (i.e. a type whose instances are identified only by their values) whose definition is defined by this specification and not by the user (e.g. specification authors).

This repertoire uses two kinds of data types: predefined data types and user-defined data types. The latter are defined in 5.3.4 <<dataType>> and 5.3.5 <<enumeration>>.

The following table lists the UML data types selected for use as predefined data type.

Table 5: UML defined data types

Name	Description and reference
Boolean	See Boolean type of [7].
Integer	See Integer type of [7].
String	See PrintableString type of [7].

The following table lists data types that are defined by this repertoire.

Table 6: Non-UML defined data types

Name	Description and reference
DateTime	See GeneralizedTime of 0.
DN	The DN (see Distinguished Name of 0) of an object contains a sequence of one or more name components. Each initial sub-sequence (note 1) of the object name is also the name of an object. The sequence of objects so identified, starting with the one identified by only the first name component and ending with the object being named, is such that each is the immediate superior (note 2) of that which follows it in the sequence. Note 1: Suppose an object's DN is composed of a sequence of 4 name components, i.e., 1st, 2nd, 3rd and 4th components. The "initial sub-sequence" is composed of the 1st, 2nd and 3rd components. Note 2: Suppose object A is name-contained (see 5.3.3) by object B, object B is said to be the immediate superior of object A.
Real	See Real type of [7]
BitString	See Bit string of clause 3 and clause G.2.5 of [7].

5.4.3.2 Example

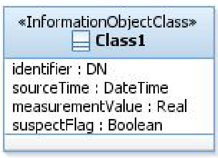


Figure 21: Predefined data types usage

Note: Use of this is optional. Uses of other means, to specify Predefined data types, are allowed.

5.4.3.3 Name style

It shall use the UCC style.

6 Qualifiers

This clause defines the qualifiers applicable for model elements specified in this document, e.g., the IOC (see 5.3.2), the Attribute (see 5.2.1). The possible qualifications are M, O, CM, CO and C. Their meanings are specified in this section. This type of qualifier is called Support Qualifier (see supportQualifier of IOC in Table 3 and supportQualifier of attribute in Table 1 of 5.2.1).

This clause also defines the qualifiers applicable to various properties of a model element, e.g. see the IOC properties excepting 'supportQualifier' in Table 3 and attributes properties excepting supportQualifier in Table 1 of 5.2.1. The possible qualifications are M, O, CM, CO and '-'. Their meanings are specified in this section. This type of qualifier is simply called Qualifier.

Definition of qualifier M (Mandatory):

→ The capability (e.g. the Attribute named `abc` of an IOC named `Xyz`; the write property of Attribute named `abc` of an IOC named `Xyz`; the IOC named `Xyz`) shall be supported. The property value is True.

Definition of qualifier O (Optional):

→The capability may or may not be supported. When supported, the property value is True.

Definition of qualifier CM (Conditional-Mandatory):

→The capability shall be supported under certain conditions, specifically:

→The class attribute qualified as CM shall have a corresponding constraint defined in the specification. If the specified constraint is met then the capability shall be supported.

Definition of qualifier CO (Conditional-Optional):

→ The capability may be may be supported under certain conditions, specifically:

→The class attribute qualified as CO shall have a corresponding constraint defined in the specification. If the specified constraint is met then the capability may be supported.

Definition of qualifier C (Conditional):

→Used for items that have multiple constraints that can not be expressed with one of the M, O, CM, CO or "-" (no support) qualifier alone. Each constraint is worded as a condition for mandatory support, optional support or "no support". The constraints must be mutually exclusive in that evaluation of all the constraints can only result in either mandatory support, optional support or "no support" for the item. Specifically:

→Each item having the support qualifier C shall have the corresponding multiple constraints defined in the IS specification. If the specified constraint is met and is related to mandatory, then the item shall be supported. If the specified constraint is met and is related to optional, then the item may be supported. If the specified constraint is met and is 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 qualifier SS (SS Conditional):

→The capability shall be supported by at least one but not all solutions.

Definition of qualifier '-' (no support):

→The capability shall not be supported. The property value is False.

7 UML Diagram Requirements

Object classes and their relationships shall be presented in class diagrams.

It is recommended to create:

→An overview class diagram containing all object classes related to a specific management area (Class Diagram).

→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 7.3.44 of 0 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.

Annex A (informative): Examples of using <<ProxyClass>>

A.1 First Example

This shows a <<ProxyClass>> named `YyyFunction`. It represents all IOCs listed in the Note under the UML diagram. All the listed IOCs, in the context of this example, inherit from `ManagedFunction` IOC.

The use of <<ProxyClass>> eliminates the need to draw multiple UML <<InformationObjectClass>> boxes, i.e. those whose names are listed in the Note, in the UML diagram.

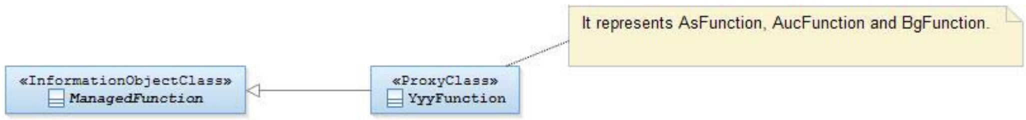


Figure 22: <<ProxyClass>> Notation Example A.1

A.2 Second Example

This shows a <<ProxyClass>> named `YyyFunction`. It represents all IOCs listed in the attached (or associated) Note. All the listed IOCs, in the context of this example, have link (internal and external) relations.

This shows a <<ProxyClass>> `InternalYyyFunction`. It represents all IOCs listed in the attached (or associated) Note.

This shows a <<ProxyClass>> `Link_a_z` and `ExternalLink_a_z`. They represent all IOCs listed in the attached (or associated) Note.

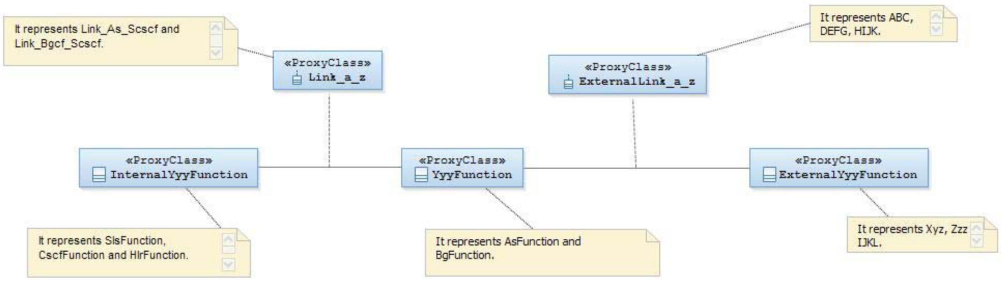


Figure 23: <<ProxyClass>> Notation Example A.2

Annex B (normative): Attribute properties

isInvariant	write	defaultValue	manager must provide a value when manager requests object creation	Meaning
☑	☑	☑	☑	Not valid.
☑	☑	☑		May be set by the manager only during object creation time; if no value is provided by the manager, the default value is used.
☑	☑		☑	Must be set by the manager during object creation time.
☑	☑			May be set by the manager only during object creation time; if no value is provided by the manager, the agent must provide a value.
☑		☑	☑	Not valid.
☑		☑		Valid but not useful.
☑			☑	Not valid.
☑				Must be set by the agent during object creation time.
	☑	☑	☑	Not valid.
	☑	☑		May be set by the manager anytime; if no value is provided by the manager at object creation time, it is set to the default value.
	☑		☑	Must be set by the manager at object creation time and may be changed anytime.
	☑			May be set by the manager at object creation time and may be changed anytime.
		☑	☑	Not valid.
		☑		Must be set by the agent to the default value at object creation time; may be changed by the agent anytime.
			☑	Not valid.
				May be set by the agent at object creation time and may be changed by the agent anytime.

Annex C (normative): Design patterns

C.1 Intervening Class and Association Class

C.1.1 Concept and Definition

Classes may be related via simple direct associations or via associations with related association classes.

However, in situations where the relationships between a number of classes is complex and especially where the relationships between instances of those classes are themselves interrelated there may be a need to encapsulate the complexity of the relationships within a class that sits between the classes that are to be related. The term “intervening class” is used here to name the pattern that describes this approach. The name “intervening class” is used as the additional class “intervenes” in the relationships between other classes.

The “intervening class” differs from the association class as the intervening class does break the association between the classes where as the association class does not but instead sits to one side. This can be seen in the following figure. A direct association between class A and C appears the same at A and C regardless of the presence or absence of an association class where as in the case of the “intervening class” there are associations between A and the “intervening class” B and C and the “intervening class” B.

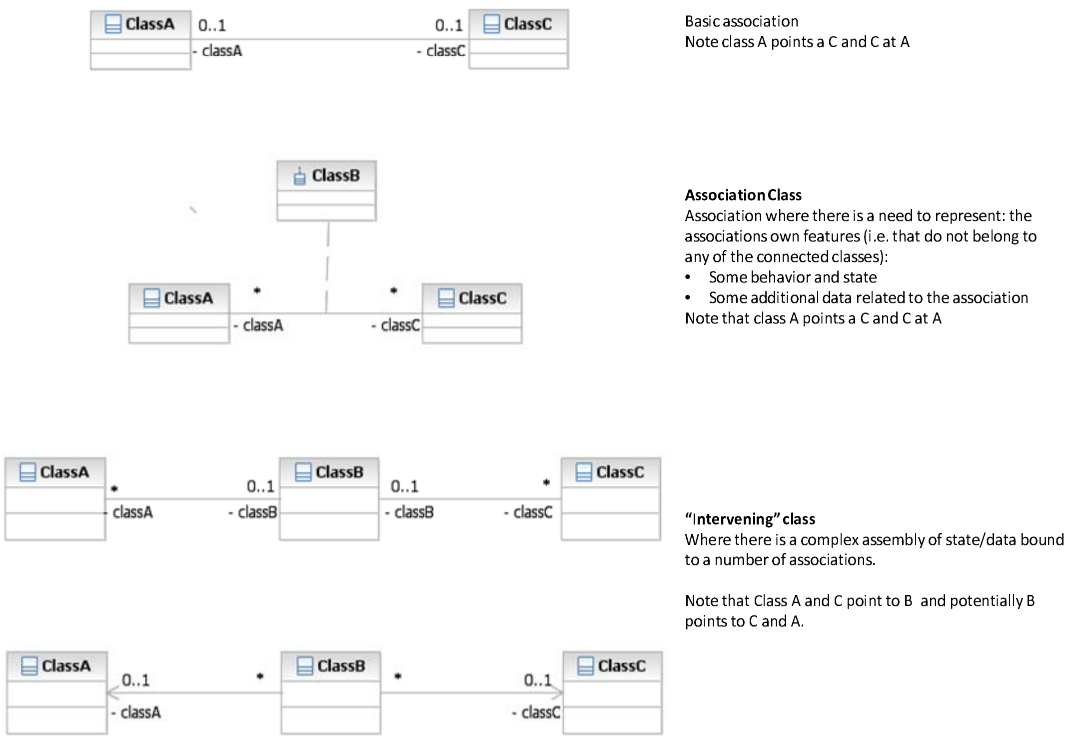


Figure 24: Various association forms

The “intervening class” is essentially no different to any other class in that it may encapsulate attributes, complex behaviour etc.

The following figure shows an instance view of both an association class form and an “intervening class” form for a complex interrelationship

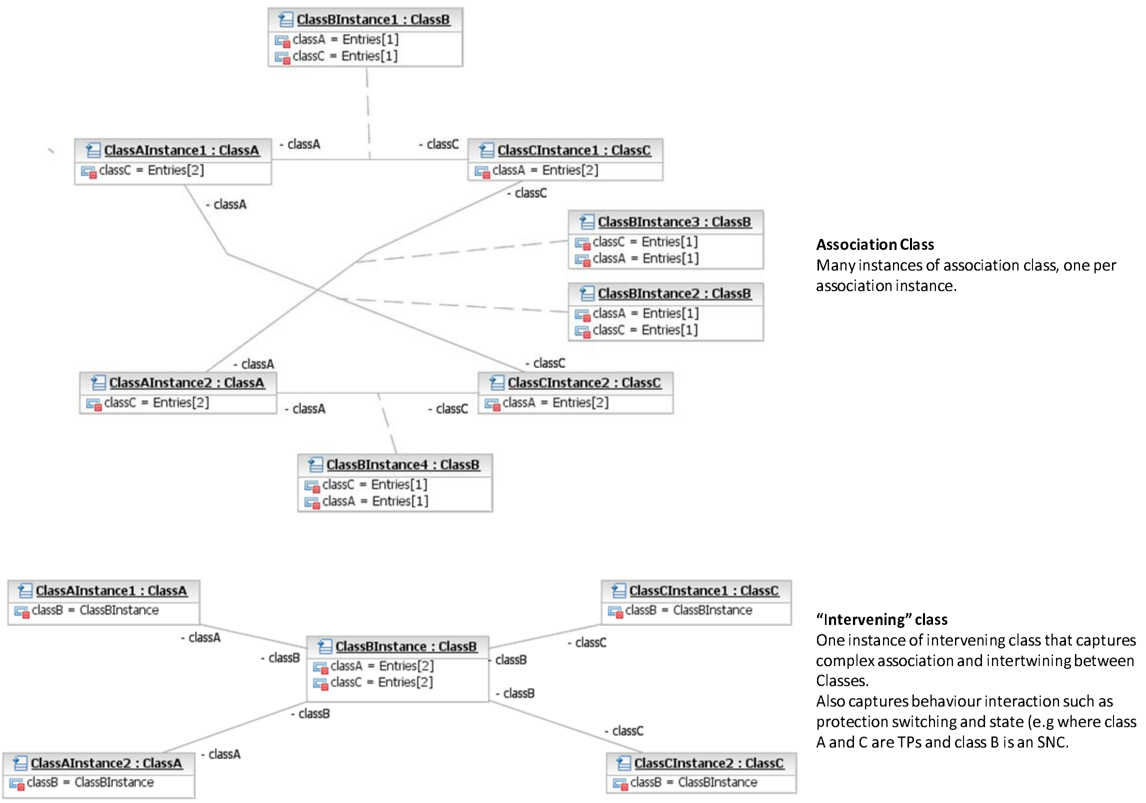


Figure 25: Instance view of "intervening class"

The case depicted above does not show interrelationships between the relationships. A practical case from modeling of the relationships between Termination Points in a fixed network does show this relationship interrelationship challenge. In this case the complexity of relationship is between instances of the same class, the Termination Point (TP). The complexity is encapsulated in a SubNetworkConnection (SNC) class.

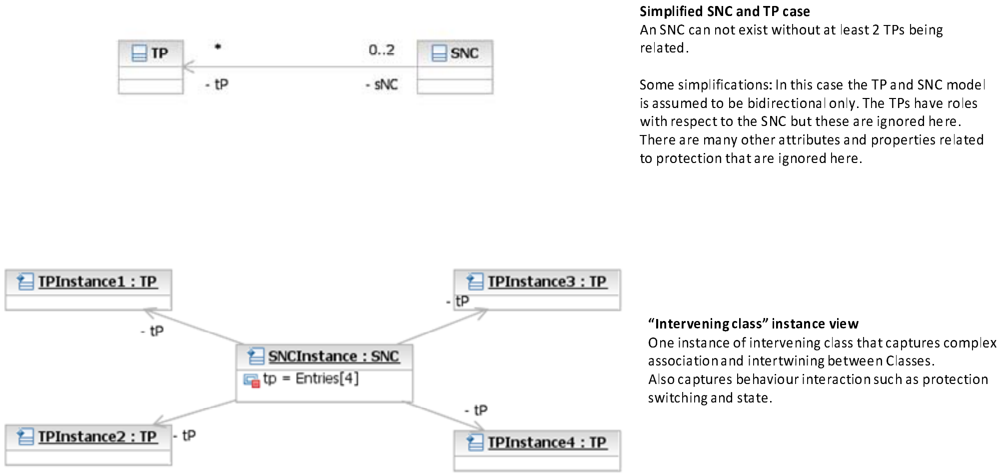


Figure 26: SNC intervening in TP-TP relationship

The SNC also encapsulates the complex behaviour of switching and path selection as depicted below.

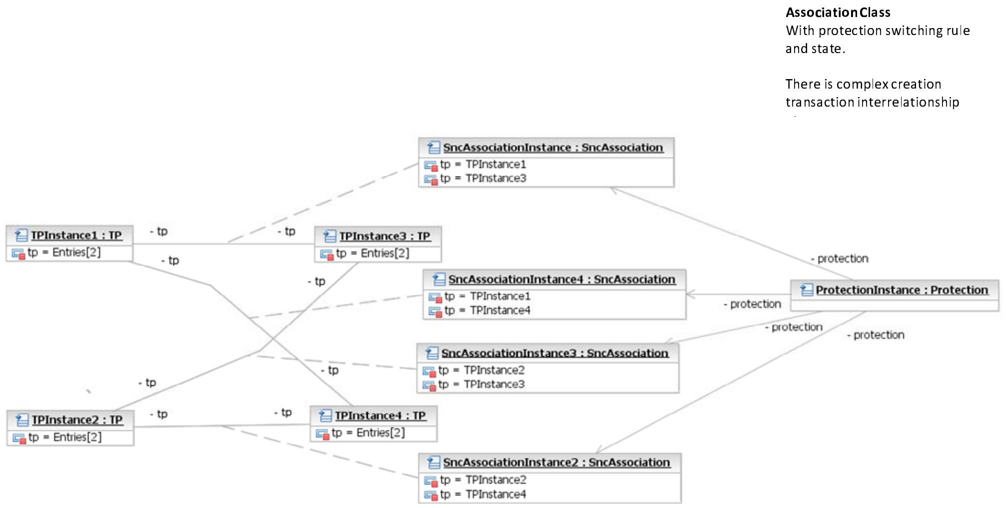


Figure 27: Complex relationship interrelationships

C.1.2 Usage in the non-transport domain

The choice of association class pattern or intervening class pattern is on a case-by-case basis.

The transport domain boundary is highlighted in the following figure.

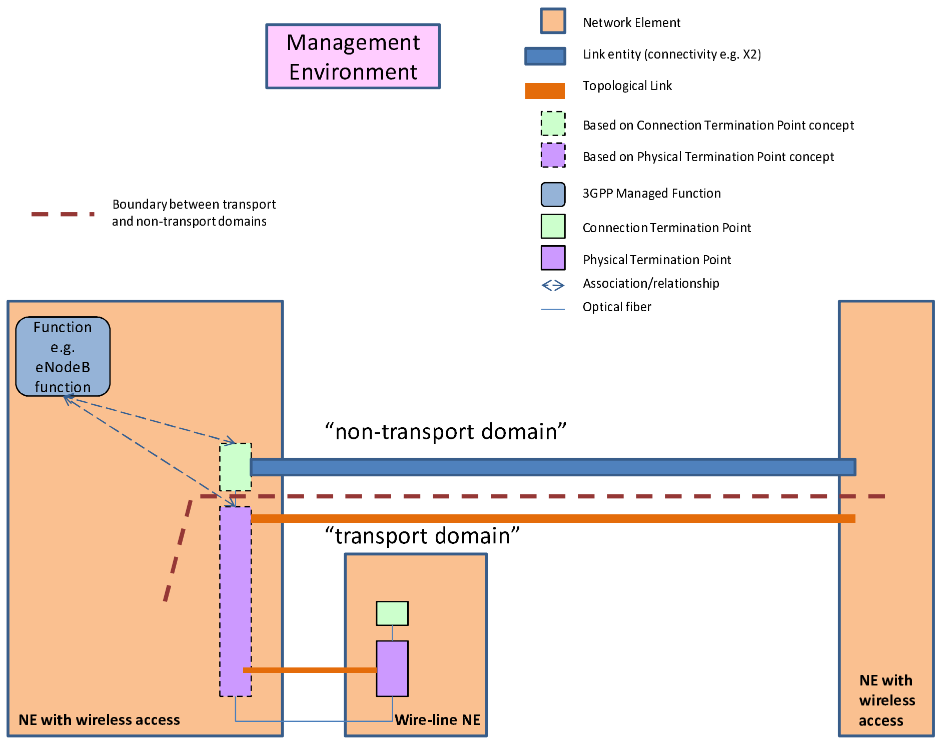


Figure 28: Highlighting the boundary between transport and non-transport domains

C.1.3 Usage in the transport domain

The following guidelines must be applied to the models of the "transport domain".

When considering interrelationships between classes the following guidelines should be applied:

- If considering all current and recognised potential future cases it is expected that the relationship between two specific classes will be 0..1:0..1 then a simple association should be used
  - This may benefit from an association class to convey rules and parameters about the association behaviour in complex cases.
- If there is recognised potential for cases currently or in future where there is a 0..*:0..* between two specific classes then intervening classes should be used to encapsulate the groupings etc. so as to convert it to 0..1:n..*.
  - Note that the 0..1:n..* association may benefit from an association class to convey rules and parameters about the association behaviour in complex cases but in the instance form this can probably be ignored or folded into the intervening class
- In general it seems appropriate to use an association class when the properties on the relationship instance cannot be obviously or reasonably folded into one of the classes at either end of the association and when there is no interdependency between association instances between a set of instances of the classes.

An example of usage of intervening class is the case of the TP-TP (TerminationPoint) relationship (0..*:0..*) where the SNC (SubNetworkConnection) is added as the intervening class between multiple TPs, i.e. TP-SNC. Note that TP-SNC actually becomes 0..2:n..* due to directionality encapsulation.

Considering the case of the adjacency relationship between PTPs it is known that although the current common cases are 1:1 there are some current and many potential future case of 0..*:0..* and hence a model that has an intervening class, i.e. the TopologicalLink, should be used.

For a degenerate instance cases of 0..*:0..* that happens to be 0..1:0..1 the intervening class pattern should still be used:

- Using the 0..1:0..1 direct association in this degenerate case brings unnecessary variety to the model and hence to the behaviour of the application (the 0..1:n..* model covers the 0..1:0..1 case with one single code form clearly)
- An instance of the 0..1:0..1 model may need to be migrated to 0..1:n..* as a result of some change in the network forcing an unnecessary administrative action to transition the model form where as in the 0..1:n..* form requires no essential change.

C.2 Use of "ExternalXyz" class

This section will be completed for the next release.

Annex D (informative): Stereotypes for naming purposes

The following diagram illustrates the various stereotypes for naming purposes.

The <<names>> with solid-diamond (see 5.3.3) identifies:

- The naming class (close to the solid diamond) and a named class;
- The naming scheme is DN;
- The container (close to the solid diamond) and the content.

The <<names>> with other types of associations (and excluding those labelled “Not Allowed”) identifies:

- The naming class (close to the hollow diamond or the source with regard to arrow direction) and a named class (the target);
- The naming scheme is DN.

The <<namedBy>> with dependency (dotted arrowed line) identifies:

- The naming class (target with regard to arrow direction) and a named class (the source);
- The naming scheme is DN.

Referring to the figure, RMA Phase 1 allows the form Class7<<names>>Class8.

The forms “in red” are not allowed.

The rest of the forms are “under investigation in Phase 2” since they all require an agreed standard mechanism on handling (named) instances whose related naming instance have been destroyed. They also lack use case support, thus far.

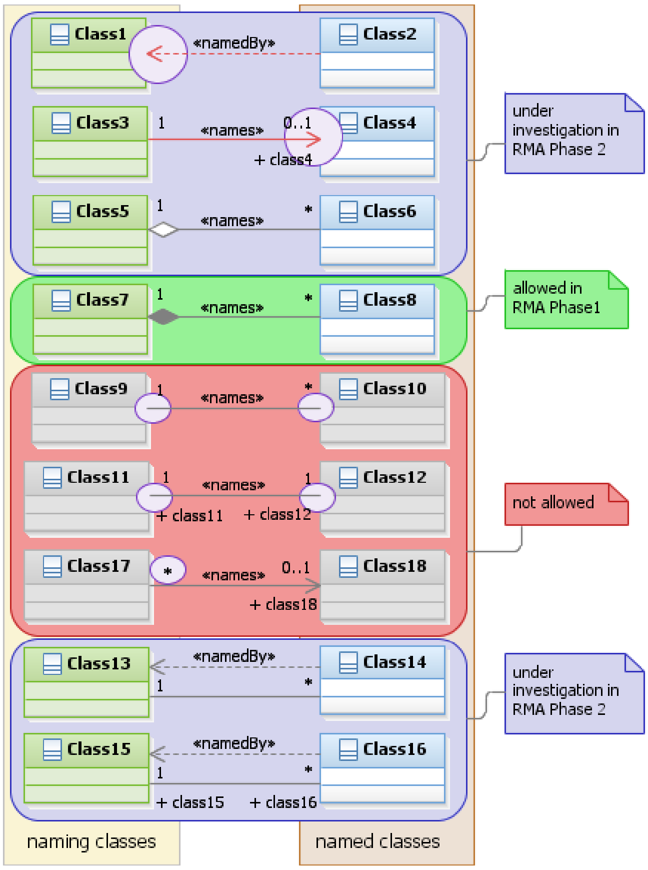


Figure 29: Various forms of naming stereotypes

Annex E (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
2012-08					First draft	--	0.1.0
2012-09	SA#57				Presented for information	0.1.0	1.0.0
2012-12	SA#58				Presented for approval	1.0.0	2.0.0
2012-12					New version after approval	2.0.0	11.0.0
2013-01					Fixed layout problems	11.0.0	11.0.1
2013-03					Fixed title of the spec by removing a semi colon	11.0.1	11.0.2

History

Document history		
V11.0.1	February 2013	Publication
V11.0.2	April 2013	Publication