CimConteXtor User Guide V5_DRAFT

CimConteXtor is an Enterprise Architect Add-In. The main purpose of *CimConteXtor* is to create a UML profile in EA, profile that is a subset of an existing UML model. This sub-setting can be done only according to some rules. CimConteXtor has implemented some rules based on those proposed by UN/Cefact[1] Core Component Technical Specification and some of those adopted by IEC[2] TC 57 (like IEC62325-450) for electricity market.

Originally, CimConteXtor has been developed to build CIM (Common Information Model from IEC) profiles, but it could be used with any Information Model.

Use of CimConteXtor implies that there is an existing UML Information Model.

Note : to use CimConteXtor, one should be familiar with :

- UML class diagram modelling, on the one hand,
- the Enterprise Architect UML tool, on the other hand,
- and with rules for creating a profile based on a UML Information Model (the best is to be aware of UN/Cefact Core Component Technical Specification – CCTS or IEC/TC57 62325-450).

CimConteXtor is an open-source tool under governance of ENTSO-E, and was originally developed at **Zamiren** by Sébastien Maligue-Clausse, André Maizener and Jean-Luc Sanson.

Enterprise Architect is a tool from **Sparx Systems**.

[1] UN/Cefact : United Nation - Center for Trade Facilitation and e-Business

[2] IEC : International Electro-technical Commission – Standard body for electro-technical domain

Changes from previous version:

This CimConteXtor version V5 corrects some bugs and introduces some new features:

- a new drag and drop function that drags and drops a class with its super classes hierarchy (feature targeting WG13 and CGMES needs): section 29 Annex H,
- a new CreatePackageProfile menu
- A new Utilities Menu offering following features:
    - change some CGMES old profile design, especially in the use of datatypes: LocalizeDatatype (see section 19.7) and ChangeSimpleFloatToFloat (see section 19.6)
    - create a profile for an all Information Model package or diagram: CreateGlobalProfile (see section 19.8 and 27 Annex F),
    - to recopy all information model notes in a profile: CopyAllNotes (see section 19.9)
    - to make all attribute types of a profile to be the ones of a domain profile: RecoverDatatype (see section 19.10)
    - to make all attributes of profile classes mandatory: MakeMembersMandatory (see section 19.11),
    - the Menu includes now IntegrityCheck features that were previously in an Integrity Menu.

This version has many features that are related to the profile modelling style used. Here is a tentative to classify the use of the features according to WG practises:

- For IEC WG 13 and EntsoE CGMES profiling style:
    - CreatePackageProfile
    - Enhanced drag & drop
    - EditIsBasedOn
    - Edit Connectors for WG13(CGMES)
    - Utilities:
        - IntegrityCheck
        - CGMESIntegrityCheck
        - ReplaceSimpleFloatByFloat
        - LocalizeDataTypes
        - CreateGlobalProfile
        - CopyAllNotes
        - RecoverProfileDatatypes

        - MakeMembersMandatory
- For IEC WG14 profiling style:
    - CreatePackageProfile
    - Basic drag & drop
    - EditIsBasedOn
    - Edit hierarchical connectors
    - Utilities:
        - IntegrityCheck

- For IEC WG16 ESMP profiling style:
    - CreatePackageProfile
    - Basic drag & drop
    - EditIsBasedOn
    - Edit hierarchical connectors
    - AttributeOrder
    - PropertyGrouping
    - Utilities:
        - ESMPIntegrityCheck

Table Of Content:

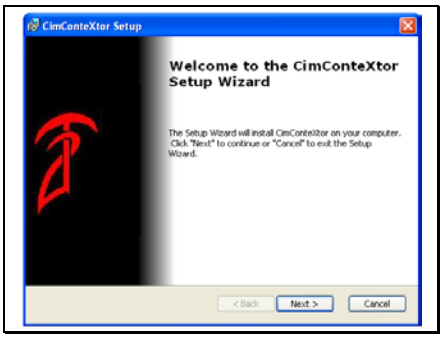
CimConteXtor Installation and Overview

1.1. Before installing "*CimConteXtor*"

Check on *Entso-E* site (https://www.entsoe.eu/digital/cim/#cim-tools) the version you are using and the requirement for it in terms of Operating Systems and Enterprise Architect version compatibility.

1.2. Installation

To install *CimConteXtor*, double-click on the delivered "*CimConteXtorSetup.msi*" file and follow the wizard instructions (Do remember to uninstall a previous installed version through the Windows "*Add or suppress programs*").



1.3. Overview

"*CimConteXtor*" is an "*Enterprise Architect*" Add-In. "*Enterprise Architect*" (EA) is primarily a UML modelling tool.

The main purpose of *CimConteXtor* is to create a UML profile in *EA*, profile that is a subset of an existing UML model. This sub-setting is done according to a methodology ("*IsBasedOn*") based on UN/Cefact Core Component Technical Specification and adapted by IEC/TC57 for its own purposes (see annexes of this document to see the methodology rules).

CimConteXtor is an EA Add-in: the UML version it uses is the *EA* one (right now UML 2.1. The produced file are "*eap*" files and could be exported with EA export feature as a "*xmi*" file.

To use *CimConteXtor*, one should be familiar with:

- UML class diagram modelling, on the one hand,
- Enterprise Architect UML tool, on the other hand,
- and with rules for creating a profile based on a UML Information Model (the best is to be aware of UN/Cefact Core Component Technical Specification – CCTS and IEC/TC57 62325-450) and look at the annexes of this document.

2. Before using CimConteXtor

2.1. Get some ideas about UML modelling and profiling methodology

In order to understand how to use *CimConteXtor*, read first Annexes A, B, C and D of this document. These annexes give you some principles on UML modelling and how it is used when doing profile according to a methodology based on UN/Cefact Core Component Technical Specification.

2.2. Be aware of naming rules

Component naming is something very important when doing modelling in general and UML modelling in particular. CCTS gives a lot of naming rules based on ISO/IEC 11179 standard. IEC TC 57 has borrowed some of these rules for its UML information model element naming. So before using *CimConteXtor* (see section 23.4 naming rules), be aware:

- of the use of naming rules for Class, Attribute, Association end role, Primitive, Enumeration, Datatype and Compound names,
- that, at Information Model level, class names MUST be unique,
- that, at Profile level, class names MUST be unique (but they could have the same name as one at Information Model level). If various classes are "*IsBasedOn*" the same Information Model Classes, use an appropriate qualifier to distinguish them,
- that Primitive, Datatype, enumeration and Compound names SHOULD be unique in all the EA project name space if they are used across all the model levels (Information Model and Profiles). In order to have uniqueness, use of qualifiers is required when appropriate to distinguish them (See section "*Define datatype template and dependencies*").
- that association end role names between two given classes MUST be unique (use qualifiers as appropriate).

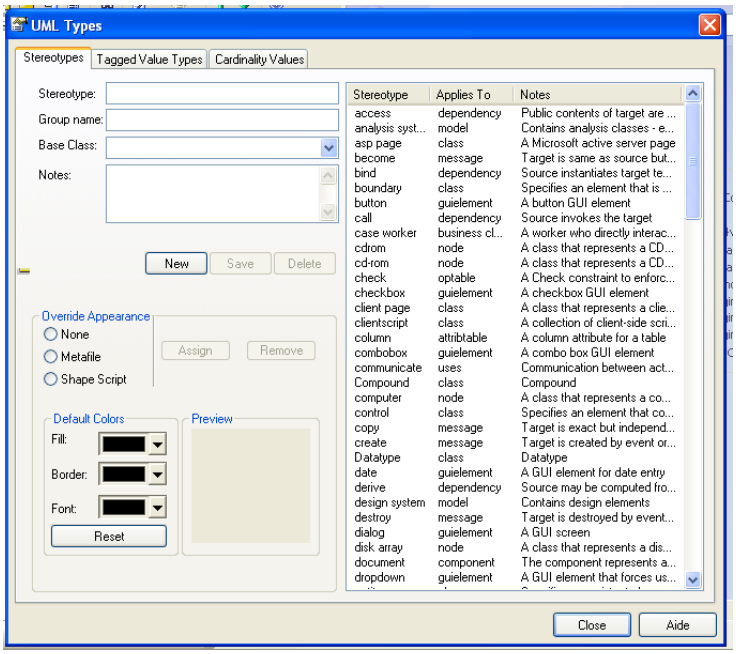
2.3. Define your Project Environment specific elements

Define all your specific UML project environment elements in *Enterprise Architect* before using *CimConteXtor*:

- Stereotype for each component category : class, attribute, association,
- Stereotype for datatyping (especially if you are starting your own information model): Datatype, Primitive, Enumeration, Compound,
- Tagged Values names,
- Cardinality values.

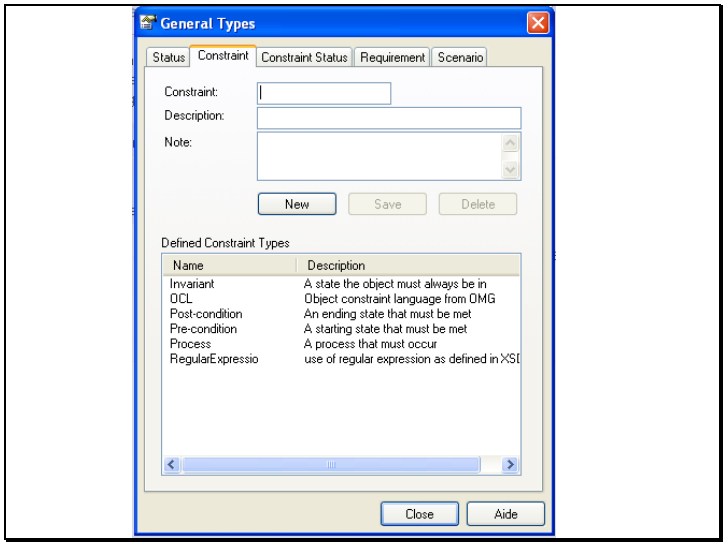
Use for that the usual EA interface:

Settings --> UML...



Define constraint types if needed. Use for that the usual EA interface:

Settings --> General Types



2.4. Define package template

Define your package template, this is very important for syntactic model generation (see CimSyntaxGen user manual), especially for assembly and syntactic level generation:

- Put Information Model and Profiles in different packages,
- At each level, put datatypes in a "*Domain*" Package,
- Define "*IsBasedOn*" package dependencies (see "*Packages Template and dependencies*" section below.

2.5. Define datatypes package template and dependencies

There are different ways to manage and use in profile primitives, enumerations, datatypes and compounds. This has to deal primarily with profile package standalone capacity and with datatype reusability.

1° If you want profiles standalone package, then all the primitives, enumerations, datatypes and compounds that are used in the profile package must be defined in this profile package or in a profile sub-package best named "*Profil Domain*".

In this case:

- primitives, enumerations, datatypes and compound names should be unique in the profile name space,
- but it means also that, if you use primitives, enumerations, datatypes or compounds already defined at the Information Model level, you should recreate them (with "*IsBasedOn*" function) at the profile level. When doing that, profile elements could have the same names than the Information model level ones.

2° If you do not want profile standalone package, then primitives, enumerations, datatypes and compounds that are used in the profile package could be defined at any modelling level. This means:

- that primitive, enumeration, datatype and compound names must be unique on the project name space (use of qualifier is required when doing an "*IsBasedOn*").

3° If you want datatype reusability in multiple profiles, the best is to put all primitives, enumerations, datatypes and compounds in a shared "*DomainProfile*" package. This implies too that all names are unique on the project name space (use of qualifier is required when doing an "*IsBasedOn*").

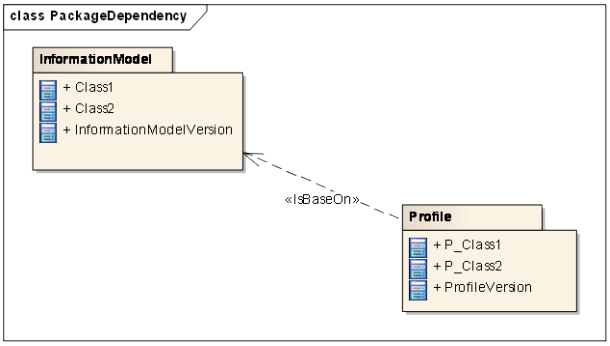
2.6. Define "IsBasedOn" Package Dependencies

"*IsBasedOn*" Package dependencies specify a hierarchy and must be defined at the time you work out Package Template definition (see use of "*CreateProfile*" Menu).

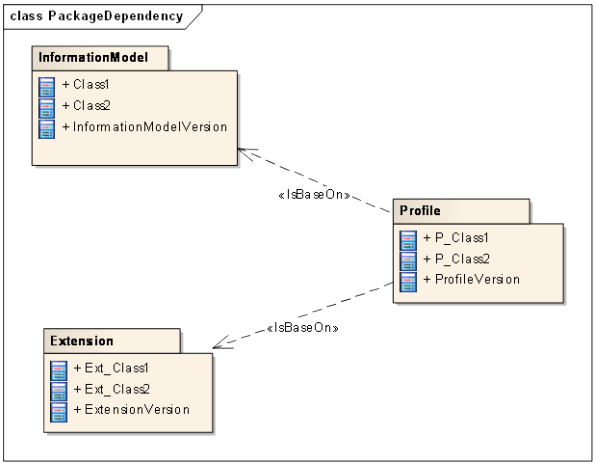
The "*IsBasedOn*" hierarchy could be of different kinds :

1° Profile Package is "*IsBasedOn*" an Information Model Package :

==> Profile classes are « *IsBasedOn* » on Information Model Package classes. Information Model Package can have sub-packages or not. But in the profile package there are no sub-packages.



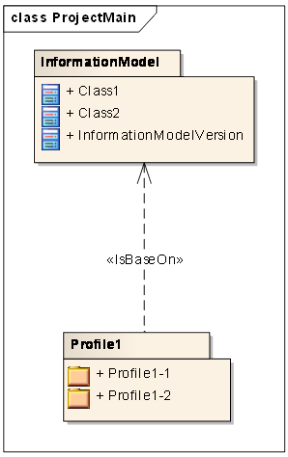
A profile could be based on several packages (example Information Model and Extensions):



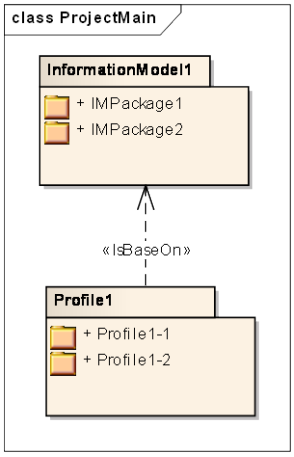
2° A Profile Package has Sub-Packages and is based on an Information Model Package Dependency

Profiles rules :

- All profile classes are in sub-packages,
- all profile classes names are unique in the context of the profile package (a profile sub-package class could not have the same name as another class of the sub-package, but also as another class in another different profile sub-package,
- associations between sub-package classes are allowed.



or

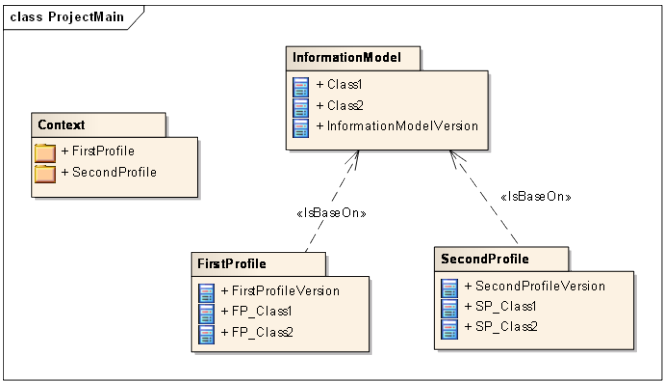


3° Profile packages are « *IsBasedOn* » an Information Model Package and are grouped in a common Package that is not itself « *IsBasedOn* » the Information Model

This is the case when different profiles are made in a given Context:

- All profile class names are unique in the context of a sub-package (classes in different profile sub-packages could have the same names).

- Associations between classes of different sub-packages are not allowed.



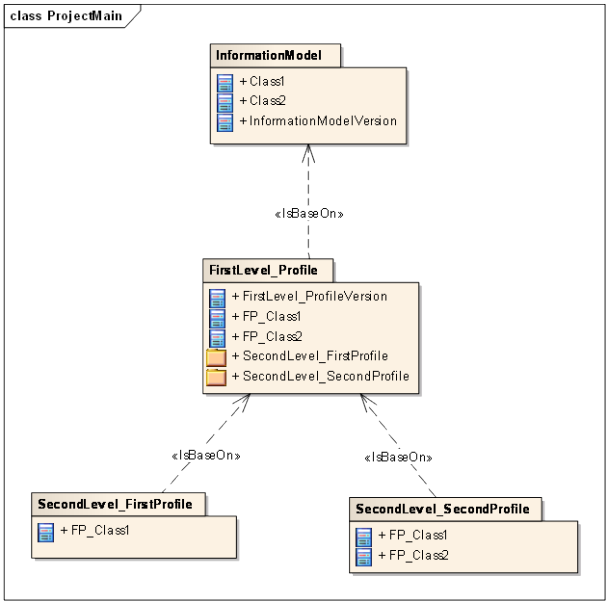
4° Sub-packages classes are based on the package classes

This case will happen when doing sub-profiles which are based on a given profile.

Profile Package is "*IsBasedOn*" an Information Model Package. Profile classes are contained in the Profile Package, and are not organised in sub-packages.

Sub-Profile packages are contained by the profile package.

This case is similar to the following one.



2.7. Do not forget

Do not forget:

- That the Information model must exist before using *CimConteXtor*, and that this Information Model must have been built according to some UML

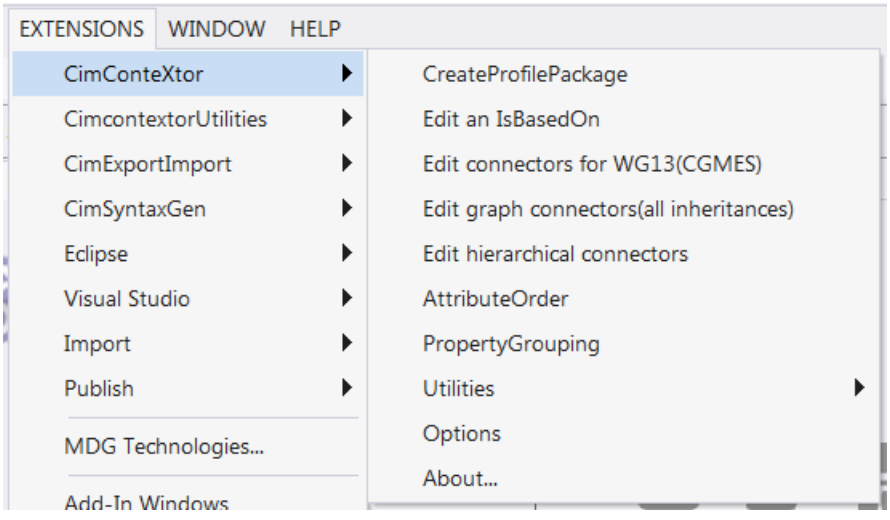
modelling rules (see annexes A, B, C and D),

- What will be the target for the syntactic model generation: XSD, RDFS, others...
- What are the package dependencies,
- Where you define your datatypes,
- That, at Profile level, all super classes MUST be "*Abstract*",
- That creating a profile is a process with several steps :
  1. Define project environment : Stereotypes, TaggedValues, Constraints types,
  2. Define package Template (Information Model level, Profile, Datatypes packages),
  3. Specify "*IsBasedOn*" dependency between Information packages and Profile packages (use the "*CreateProfile*" or "*CreateGlobalProfile*" function),
  4. Create profile (or "*IsBasedOn*") datatypes,
  5. Create profile (or "*IsBasedOn*") classes,
  6. Use profile to generate assembly model if required (Example: IEC 62325-450),
  7. Use profile or assembly model to generate syntax models (see "*CimSyntaxGen User Guide*").

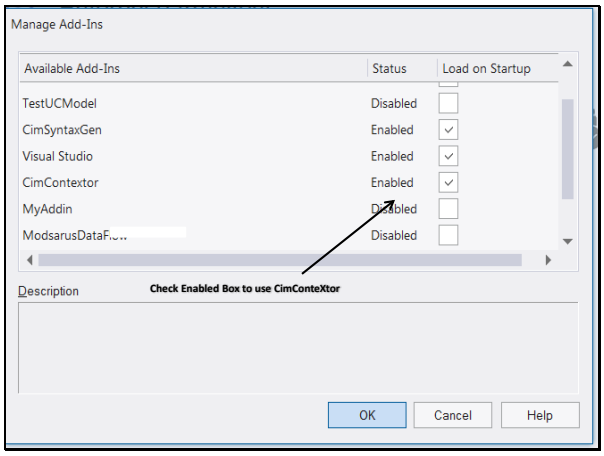
3. CimConteXtor Presentation

3.1. General

CimConteXtor is managed as an EA Add-In. It is referenced in the EA tool bar:



In order to use it you must first enabled it. To do that, click on the "*Manage Add-Ins*", this will display the "*Manage Add-Ins*" window where you check *CimConteXtor* enabled box:

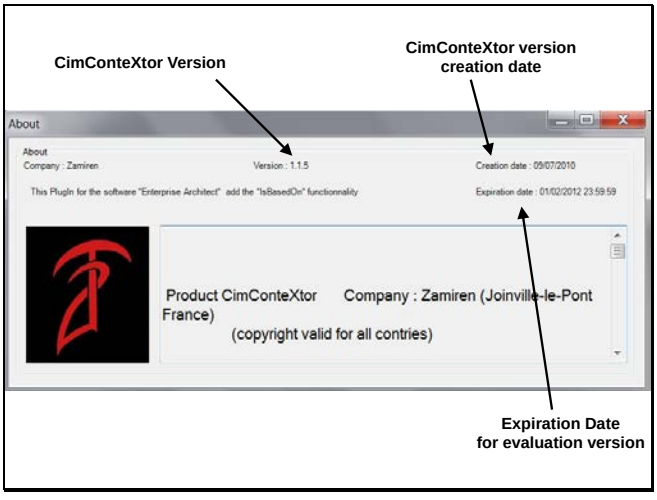


CimConteXtor has a base functionality of making an "*IsBasedOn"* Class by "dragging and dropping" an Information Model Class. But it has some other functionalities that are given by the *CimConteXtor* Menu items:

- CreateProfilePackage
- Edit an IsBasedOn
- Edit Connectors for WG13 (CGMES)
- Edit Graph Connectors (All inheritance)
- Edit Hierarchical Connectors
- PropertyGrouping
- Attribute Order
- Utilities
- Options
- About

3.2. "About" Menu

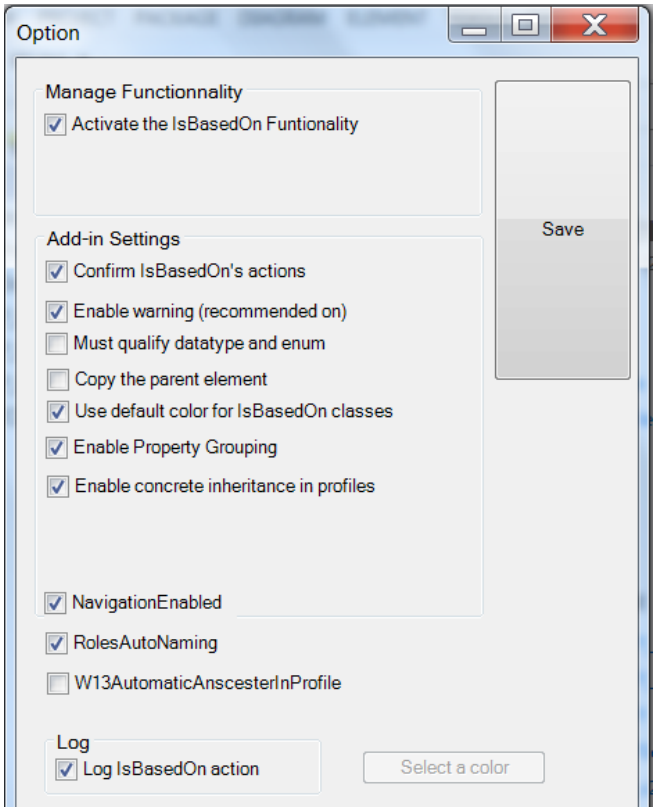
The "About" menu gives information on "CimConteXtor" version and validity:



3.3. "Options" Menu :

The "Options" menu lets you:

- Enable or Disable *CimConteXtor* Add-In functionality.
- Confirm IsBasedOn Actions. When unchecked, each drag and drop action will only enable "*IsBasedOn*" function, no class copy is allowed in a diagram. Unchecking is interesting when starting a profile.
- Enable or Disable warnings.
- Force the use of qualifiers for primitives, datatypes, enumerations and compounds when an "*IsBasedOn*" is made on one of this element. When checked this option will force the use of a qualifier: this is useful when you want reusability of all datatypes across profiles. Uncheck this option when you target a profile standalone package that includes the primitives, datatypes, enumerations and compounds it is using. But in this case be sure that attribute typing does not use primitives, datatypes, enumerations or compounds that belong to another package.
- Copy the parent element. This option fix what is the default for copying parent element when doing an "*IsBasedOn*" action. If unchecked, by default the parent class will not be copied in the current diagram.
- Fix the class box background color.
- Enable property Grouping. When checked this options allows "PropertyGrouping" in Assembly Model.
- Enable concrete inheritance in Profiles. When Checked, this option allows superclass in a profile hierarchy to be concrete.
- Navigation Enabled. This option let connectors in a hierarchy to be drawn as an arrow instead of an aggregation.
- RolesAutoNaming. This option allows change of end role names when an association of a super class in the information model is used as a native association for a profile class that is basedOn a subclass of the superclass.
- WG13AutomaticAnscesterInProfile. This option allows drag and droping of a class with all its ancestors in the diagram (option for WG13 and CGMES profiling style).
- Trace all actions in a log.



3.4. "CreateProfilePackage" Menu

The "*CreateProfilePackage*" is used to create a Profile Package and associated diagram with the right dependencies.

3.5. "Edit IsBasedOn" Menu:

The "*Edit IsBasedOn*" is used when you need to modify an "*IsBasedOn*" class that has been created with the "*Drag and Drop*" feature of CimConteXtor (See "*Edit IsBasedOn*" sections).

3.6. "Edit Connectors" Menu

The "*Edit Connectors*" is used whenever you want to specify or modify the associations of an "*IsBasedOn*" class (See section 12 "*Edit Connectors*" function).

There are three different kinds of Edit Connectors:

1. Edit Connectors for WG13. This menu optimizes the presentation of connections between classes for WG13 profile design (The profile keep the inheritance path of the information model, and at a class level there could be only the native associations).
2. Edit Graph Connectors. This menu is used in the design of complex graph profiles (any kind of inheritance could use at profile level).

3. Edit Hierarchy Connectors. This menu is used for the design of hierarchical profiles (like WG14 or WG16 ones).

3.7. "AttributeOrder" Menu :

The "*AttributeOrder*" is used at two levels: at Profile level to define the order of properties used for property grouping and at Assembly level to define the order of elements output in an XML schema (See section 18.2 "*AttributeOrder*" sections).

3.8. "PropertyGrouping" Menu :

The "*PropertyGrouping*" is used when you need to generate an assembly model with grouping of properties (See section 18.3 "*PropertyGrouping*" sections).

3.9. "Utilities" Menu :

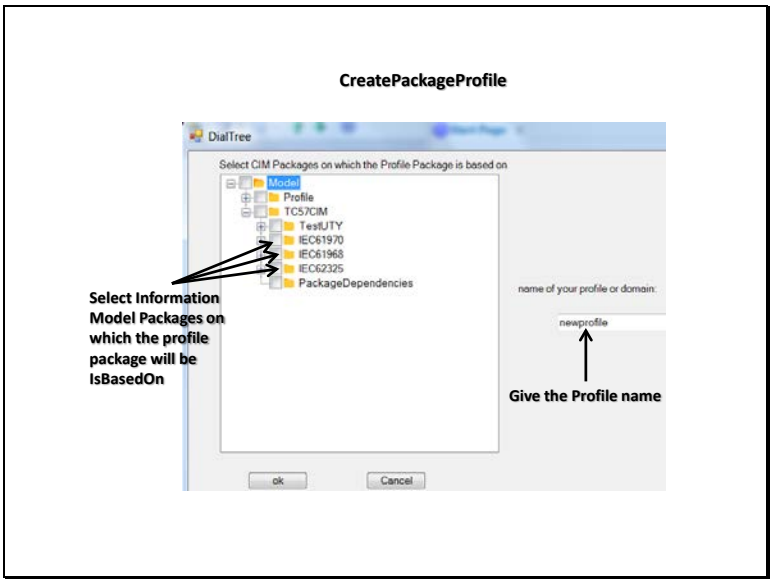
The "*Utilities*" is used:

- to change some CGMES old profile design, especially in the use of datatypes,
- to create a profile for an all Information Model package or diagram,
- to recopy all information model notes in a profile,
- to make all attribute types of a profile to be the ones of a domain profile,
- to make all attributes of profile classes mandatory,
- to check if a profile is conforming to the profiling rules. Currently, depending on the kind of profiles (IEC, CGMES and European Market style, there are different checking (See "*IntegrityCheck*" section 19.2).

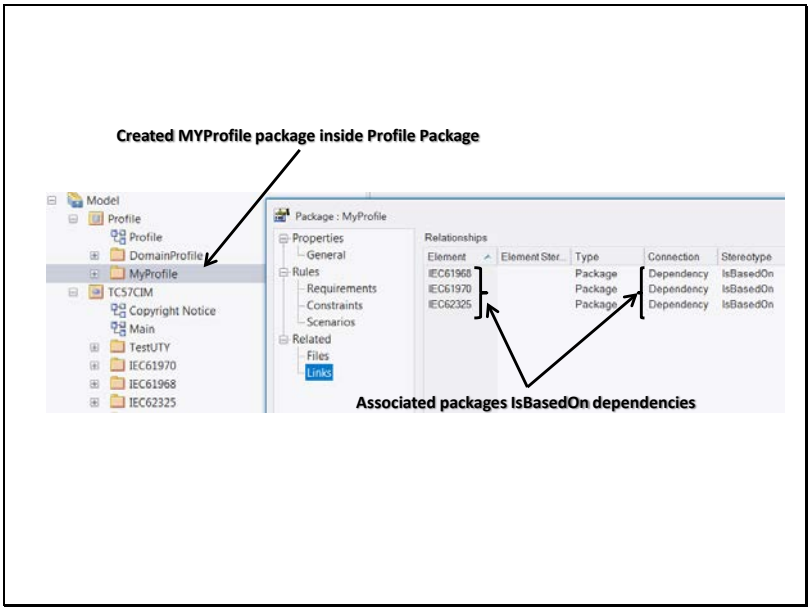
4. "Create ProfilePackage" with CimConteXtor

The "*CreateProfilePackage*" Menu is used to create a Profile package with its "*IsBasedOn*" package dependencies (usually Information Model packages).

To create a package profile inside another package, select the containing package and launch the CimConteXtor "*CreateProfilePackage*" Menu. This will open a window:



Enter the name of the profile and select the packages of the information model that will be used as the basis for the profile. Saying "*OK*" will create a profile package basedOn the selected packages:



5. Create "IsBasedOn Elements" with CimConteXtor

5.1. IsBasedOn Drag and Drop function:

The basic function of *CimConteXtor* is to create "*IsBasedOn*" elements. This is done by dragging and dropping an element (primitive, enumeration, datatype, compound and class) in a profile diagram. There are two ways of using this function:

- either directly, EA drag and drop function are bypassed and copying a class in another diagram is not possible, the "*IsBasedOn*" function is always active. This behavior is selected when the "*Confirm IsBasedOn's*

action" box is checked in the *CimConteXtor* "Options" Menu.

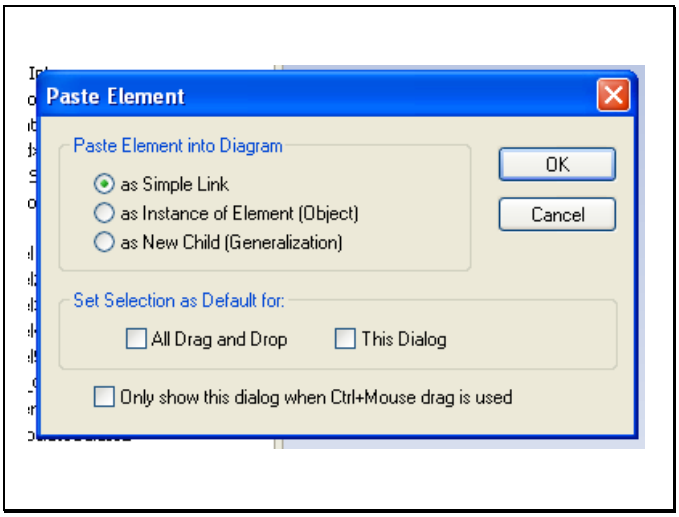
- Or indirectly, *CimConteXtor* asks if you want to bypass EA drag and drop function or if you want to use it. This will let you copy a class in a diagram or do an "*IsBasedOn*". This behavior is selected when the "*Confirm IsBasedOn's action*" box is unchecked in the *CimConteXtor* Options Menu.

When you drag and drop an Information Model element,

- if the direct mode is selected ("*Confirm IsBasedOn's action*" box is checked), the "*IsBasedOn*" window will appear directly (step 3),
- if the indirect mode is selected ("*Confirm IsBasedOn's action*" box is unchecked), three windows will be displayed successively (step 1, 2 and 3) :

1° a first prompt window is displayed :

This is the usual EA prompt window to ask if you want to show the existing element in a new diagram, or make an instance or make a child element. If you want to create a profile element based on the information class you drag and drop, click on "*OK*"



2° a second prompt window appears :

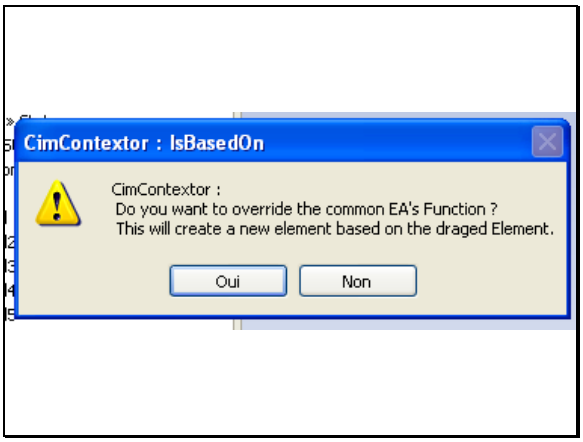
Note : if the window does not appear, check if CimConteXtor Add-In has been enabled (see Manage Add-Ins function In EA) and that the "Activate the IsBasedOn functionality" box is checked in the CimConteXtor "Options" window.

This pop up window asks you if you want to make:

- a drag and drop of the element in the diagram (same as EA function),
- or to make an "*IsBasedOn*" element based on the one you are dragging and dropping.

If you want:

- to drag and drop the element, select "*NO*" (Non),
- to make an "*IsBasedOn*" element, select "*OK*" (Oui).

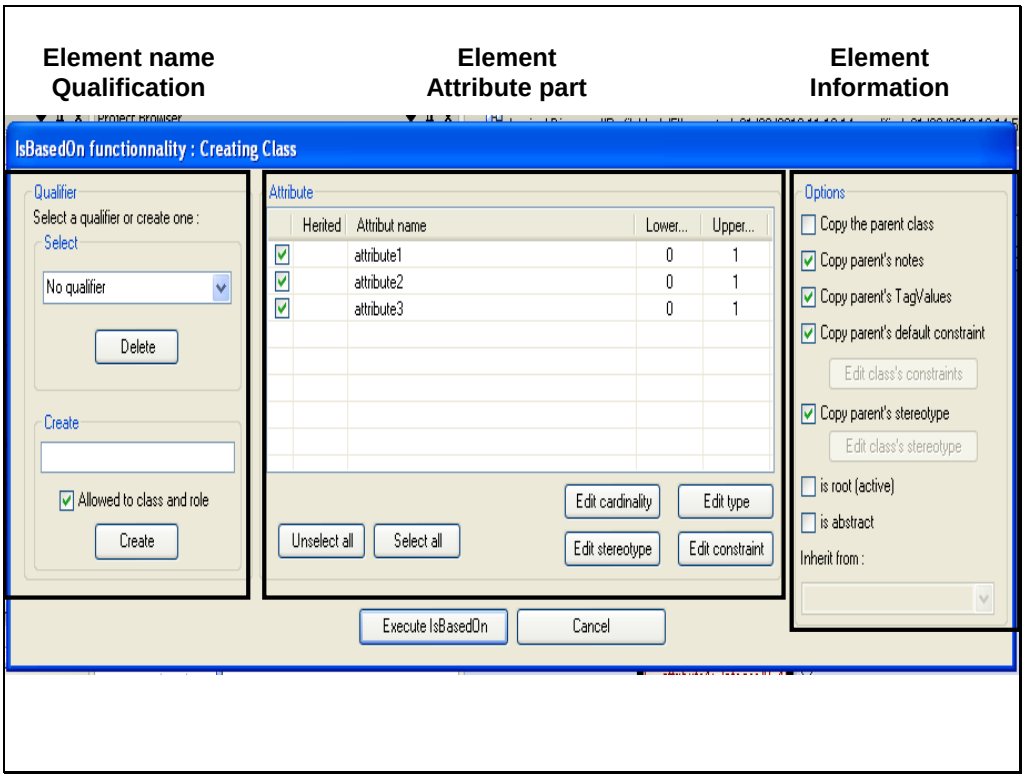


3° the "*IsBasedOn*" window appears:

But depending on the element (primitive, enumeration, datatype, compound, class) you have dragged and dropped, the "*IsBasedOn*" window will appear and have some differences according to the element that it is "*IsBasedOn*".

This window lets you qualify the "*IsBasedOn*" (or profile) element and refine its information and attributes. Basically, this window has three parts:

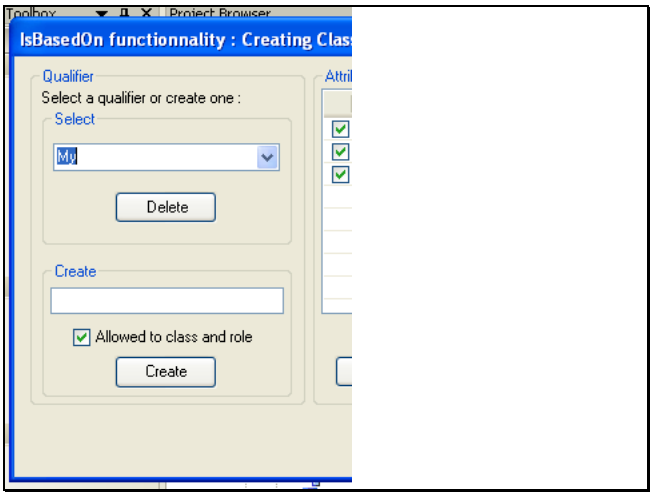
- one for giving a qualifier, on the left side,
- one for profile element information, on the right side,
- one for attribute selection and restriction, in the middle.



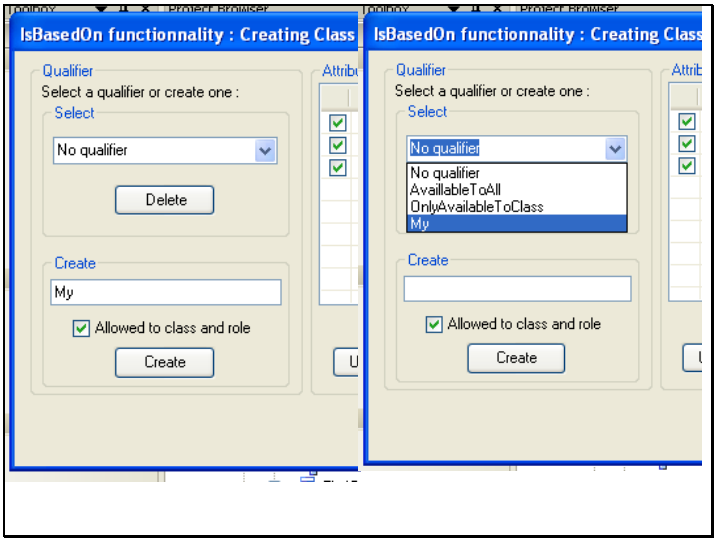
5.2. Qualify an "*IsBasedOn*" element Name:

Two ways:

- either you want to give a qualifier straightaway : write the name in the qualifier field directly,



- or you want to use an already defined qualifier that you could find and or insert into a list. Select the name in the drop-down menu.



Note : you could manage your list by entering new qualifiers in the create field and click create. You could also say that this qualifier will be entered in the qualifier class and role list, if the "*Allowed to class and role*" box is checked, otherwise it will belong only to the list of the element you are working on.

Note: you could specify a predefined list of qualifiers by using the "Configuration.xml" file of CimConteXtor (see configuration file section).

Now depending on the element that is the parent element of the IsBasedOn one, the window parts for information and attributes will have some differences.

5.3. "*IsBasedOn*" relation in EA

The "*IsBasedOn*" relation between two elements (Classes, Primitives, Datatypes, Compounds) could always be checked in EA:

- Open the "*Properties*" windows of the element
- Go to the "*link*" tab
  - If there is an "*IsBasedOn*" relation, it will be marked with a "dependency" connection and a "*IsBasedOn*" stereotype.
- Go to the "*General*" tab, and select "*Tags*"
  - If there is an "*IsBasedOn*" relation, there will be a "GUIDBasedOn" TaggedValue that gives the value of the GUID of the information model element.

6. Create "IsBasedOn" Primitives with CimConteXtor

6.1. Overview

There are two cases for using the "*IsBasedOn*" feature on a Primitive:

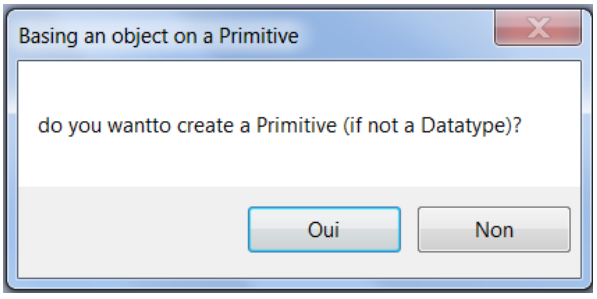
- Create a Profile Primitive,
- or create a CIMDatatype whose value space is a restriction of the primitive value space.

Example of creation:

1. a String profile primitive basedOn the String primitive of the information model.
2. a String value space restricted to 35 characters.

6.2. Drag and drop a Primitive, create a Primitive

So when the drag and drop element is a <<Primitive>>, a window open to ask if you want to create a profile Primitive (Yes) or if you want to create a Datatype (No) that is based on the Primitive but defines a restricted value space.



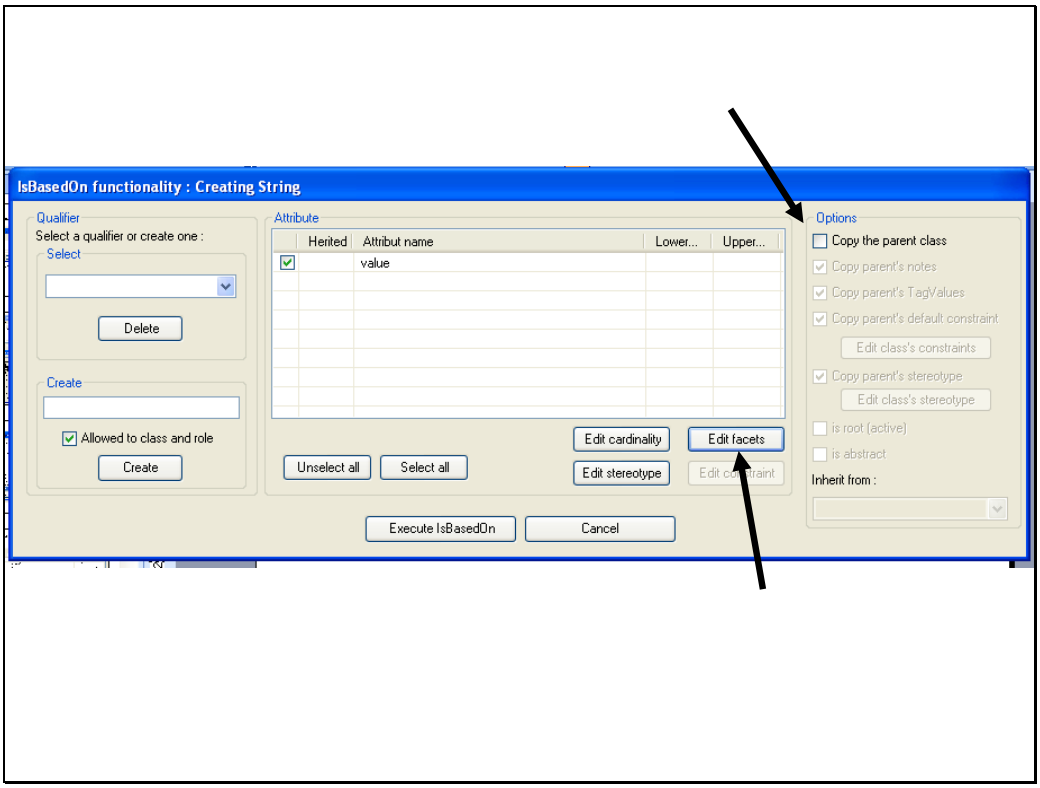
If you say "Yes", a profile Primitive will be created.

6.3. Drag and Drop a Primitive, create a CIMDatatype

If you say "No", it means that you want to define a value space that is a restriction of the value space of the Primitive. In this case a <<CIMDatatype>> with an attribute "*value*" is created as the "*IsBasedOn*" element:

- The type of the attribute "value" is identical to the <<Primitive>> it is based on : if the <<Primitive>> is String, the type will be String and so on,
- The value space of the attribute "*value*" is a subset of the primitive value space. This subset is specified by defining restrictions on the primitive value space. These restrictions are standardized as "*facets*", and these facets depend on the primitive.

The "*IsBasedOn*" window for a CIMDatatype basedOn a Primitive will appear like this:



6.4. Qualify the IsBasedOn CIMDatatype

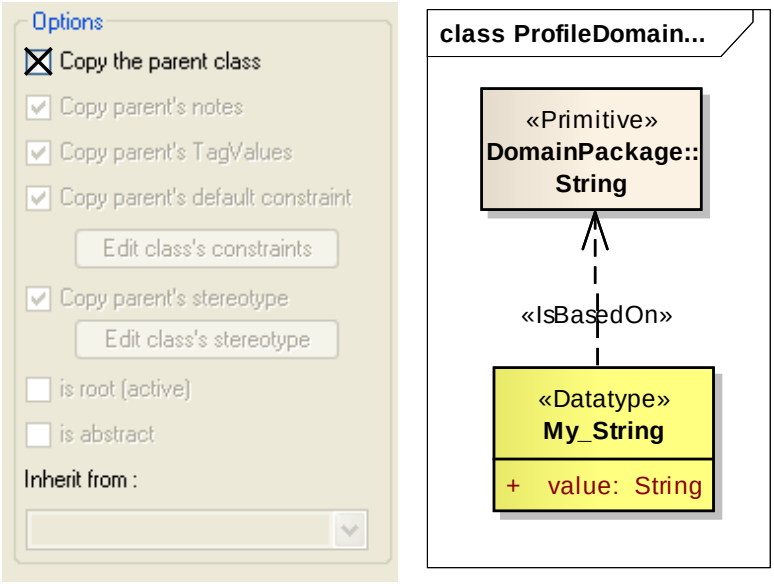
See section 5.2 for use and management of qualifiers.

- If the "*Must qualify datatype and enum*" box of the "*Options*" Menu has been checked, the "*IsBasedOn*" Datatype name must have the Primitive name prefixed by a Qualifier. *CimConteXtor* will force you to enter a qualifier for this Datatype.
- If the box is unchecked, the datatype could have the same name as the primitive, but this is not a good practice at all.

Remember that if you want to be able to reuse any primitive, datatype, enumeration or compound in any profile, their names must be unique in the Information Model and Profiles name space.

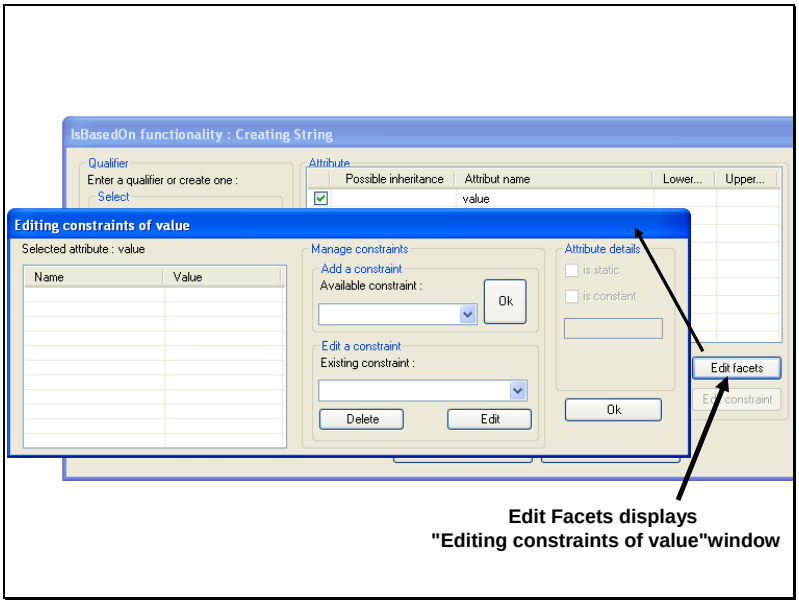
6.5. Copy Parent Element

Selecting "*Copy parent's class*", allows the inclusion of the Information Model Primitive along with the corresponding "*IsBasedOn*" datatype, and shows the "*IsBasedOn*" dependency link between the two elements :

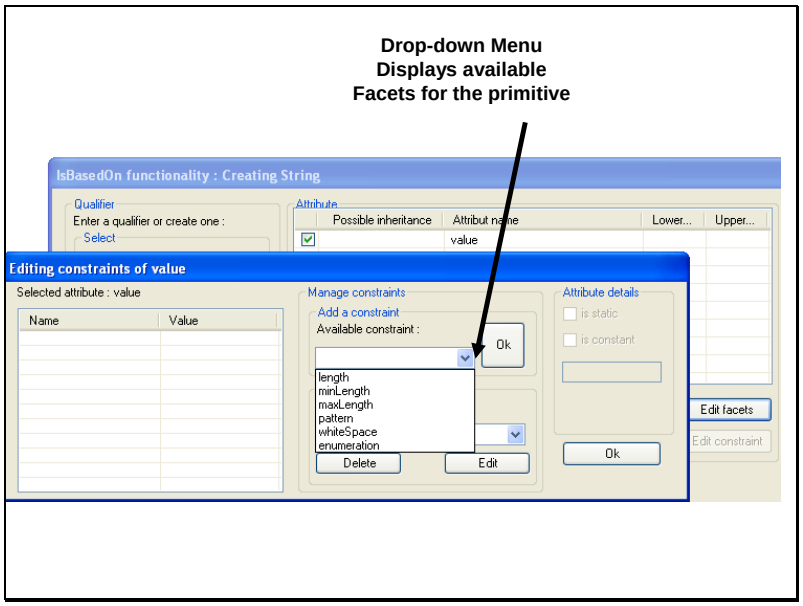


6.6. Define value space restrictions

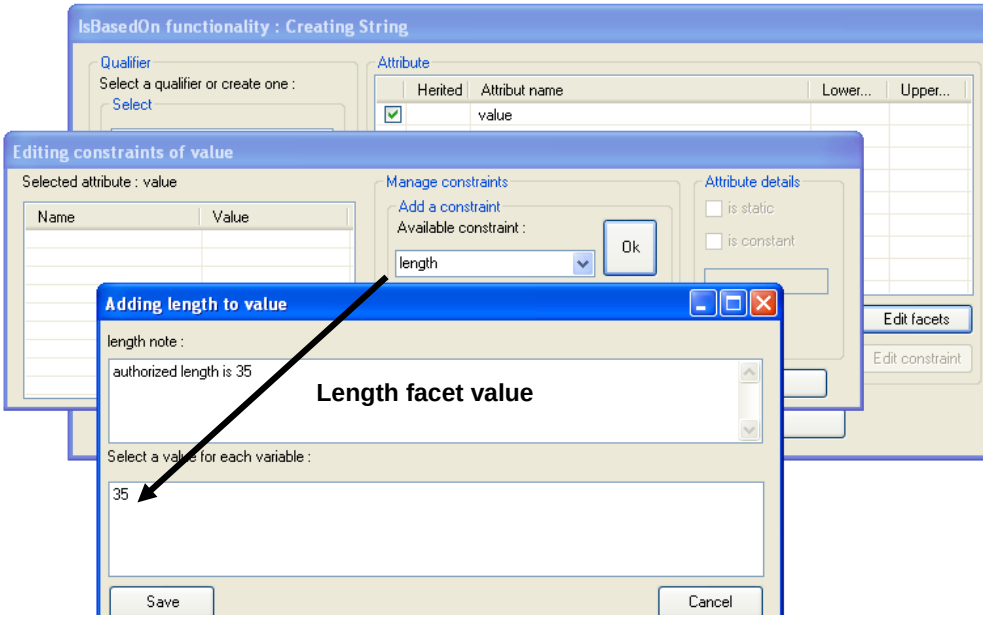
Selecting "*Edit facets*" lets you define all restrictions (or facets or constraints) on the value space of the primitive. A new window appears to express constraints:



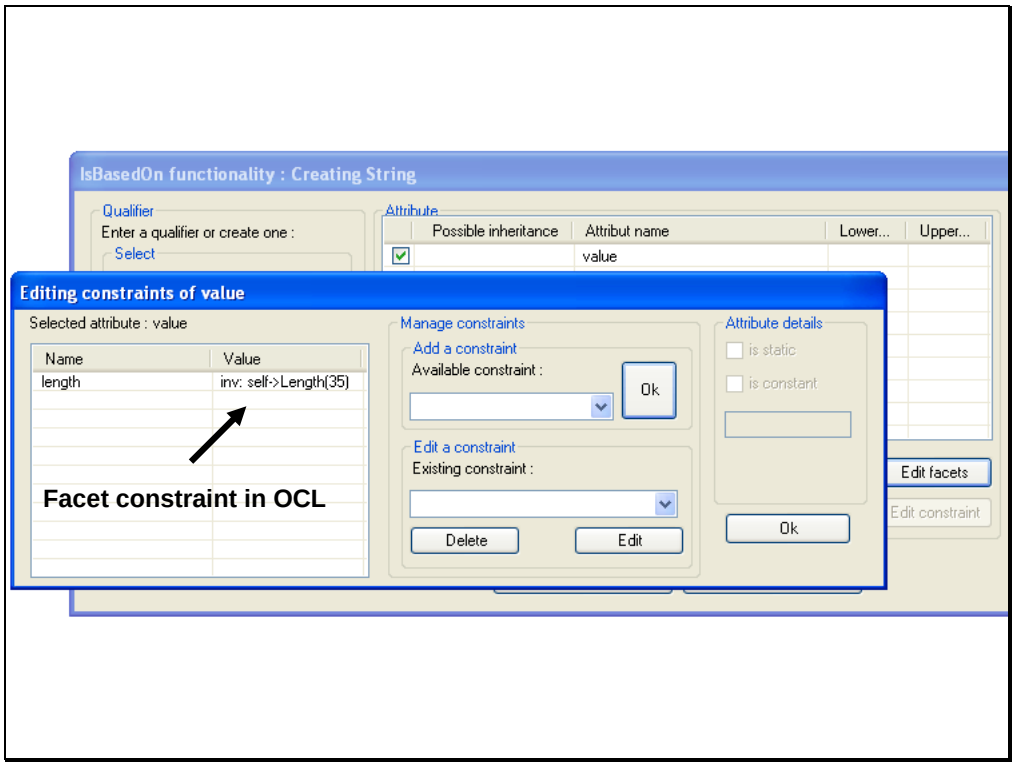
To add a facet, click on the arrow of the drop-down menu of "*Available constraint*":



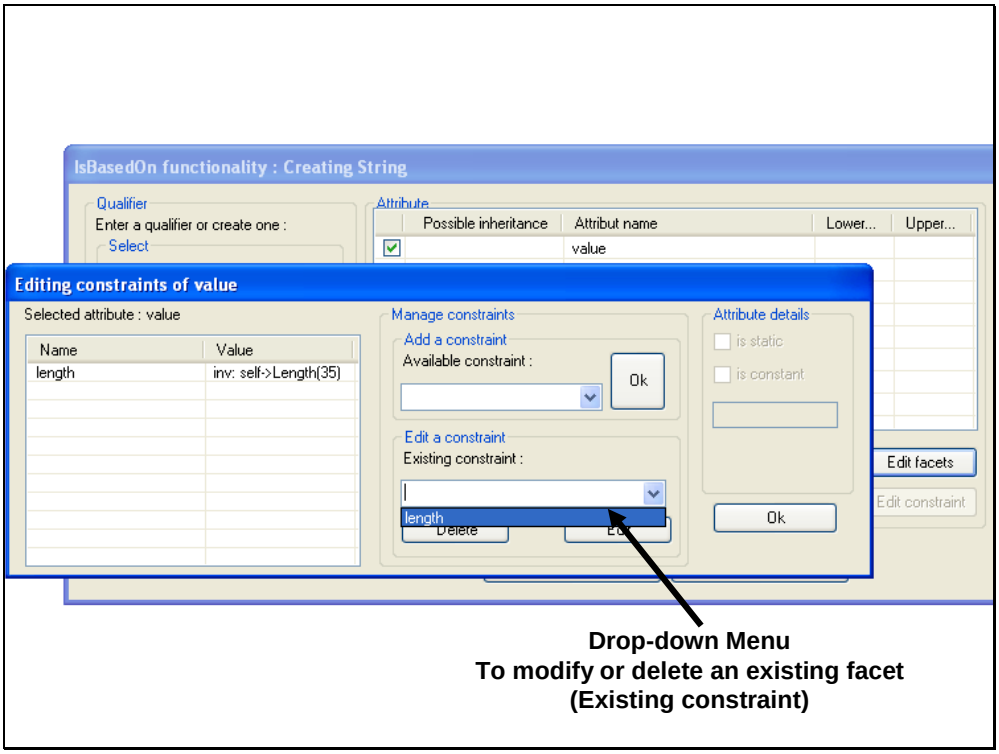
This will show all the available facets (or constraints) for the primitive (here a String). Select a facet and click on "*OK*" button, this will display the facet window:



Enter the value for the given facets and optionally enter facet notes. Then "*Save*". The Constraint window will appear, and will show the facet with its value (note that the facet value is entered as an OCL constraint, but this is done automatically by *CimConteXtor*):



The existing facet is now also part of the drop-down menu of the existing constraint: this will enable you to modify or delete this existing constraint:



7. Create “IsBasedOn Enumeration” with CimConteXtor

7.1. Reminder:

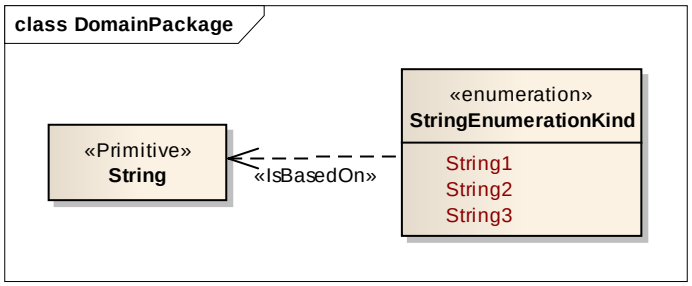
- All <<enumeration>> names follow the UML naming rule that specifies that the enumeration name is ended by the term “*Kind*”.
- All <<enumerations>> are “*IsBasedOn*” either a <<Primitive>>, or a <<CIMDatatype>> “*IsBasedOn”* a <<Primitive>> or another <<enumeration>>. This makes a hierarchy of elements.

7.2. Create an <<enumeration>>:

This is done with usual EA tools, by selecting the enumeration box in the EA “*Class Menu*” of the “*Tool Box*” and by dragging and dropping it in a diagram.

Depending on the kind of <<enumeration> you are doing, you must make a dependency link stereotyped with “*IsBasedOn*” between this <<enumeration>> and the <<Primitive>> (or <<CIMDatatype>>) it is based on.

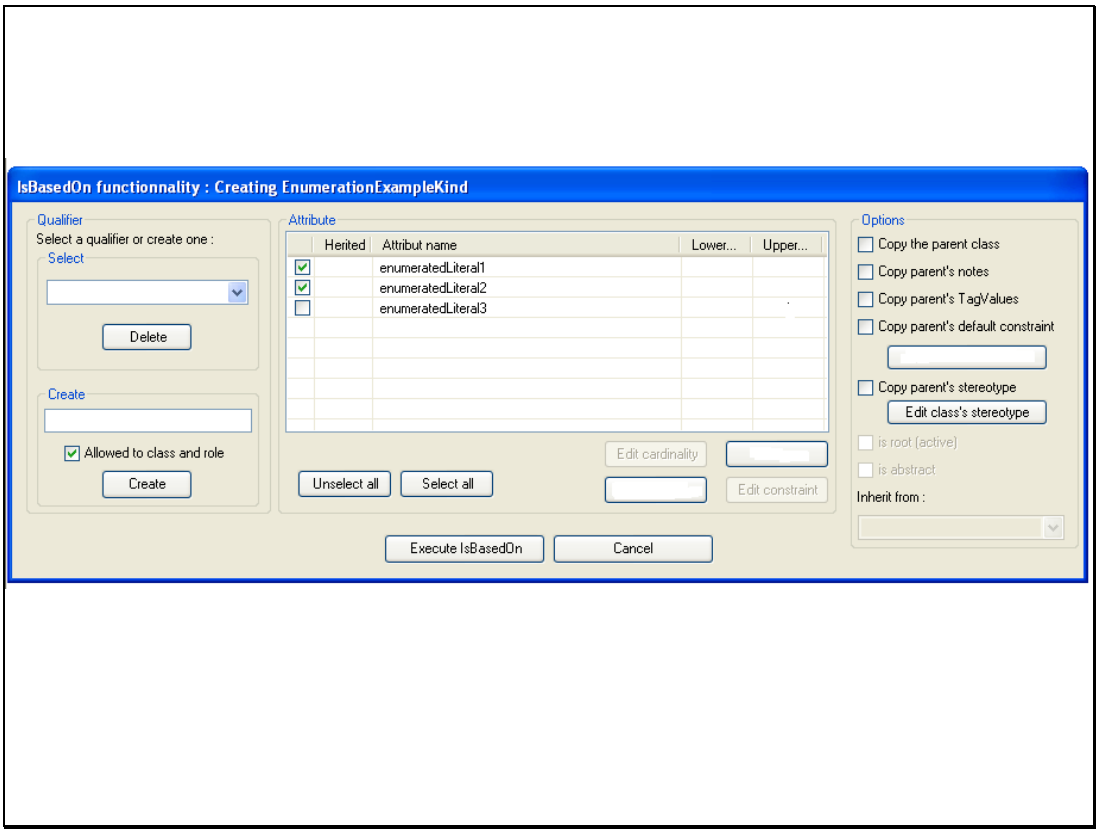
This is done with usual EA functions not with CimConteXtor



7.3. Create an “IsBasedOn” Enumeration with *CimConteXtor*:

This function is only to create an “*IsBasedOn*” <<enumeration>> on an existing <<enumeration>>.

When the drag and drop element is an <<enumeration>>, it means that you could restrict the list of “*enumeratedLiterals*” of the <<enumeration>>. The “*IsBasedOn*” window for an enumeration will appear like this:



- Qualify the IsBasedOn enumeration (see section 5.2 for use and management of qualifiers):
    - If the "*Must qualify datatype and enum*" box of the "*Options*" Menu has been checked, the "*IsBasedOn*" enumeration name will have the name of the enumeration it is based on prefixed by a Qualifier. *CimConteXtor* will force you to enter a qualifier for this enumeration.
    - If the box is unchecked, the enumeration could have the same name as the enumeration it is "*IsBasedOn*", but this is not a good practice at all.

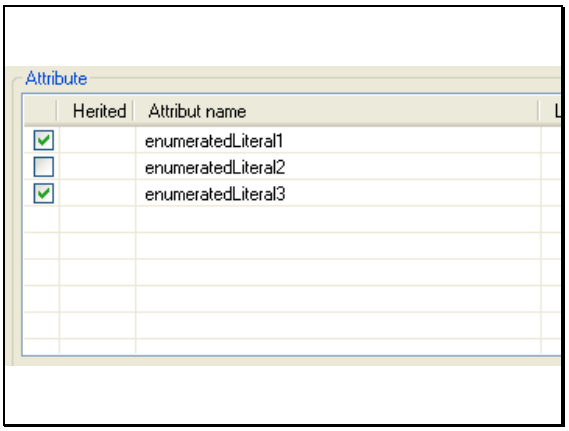
Remember that if you want to be able to reuse any primitive, datatype, enumeration or compound in any profile, their names must be unique in the Information Model and Profiles name space.

- *Copy Parent Element*

Selecting "*Copy parent's class*", allows the inclusion of the Information Model enumeration along with the corresponding "*IsBasedOn*" enumeration, and shows the "*IsBasedOn*" dependency link between the two elements :

- *Select "EnumeratedLiterals"*

Select the "*enumeratedLiterals*" for the "*IsBasedOn*" enumeration



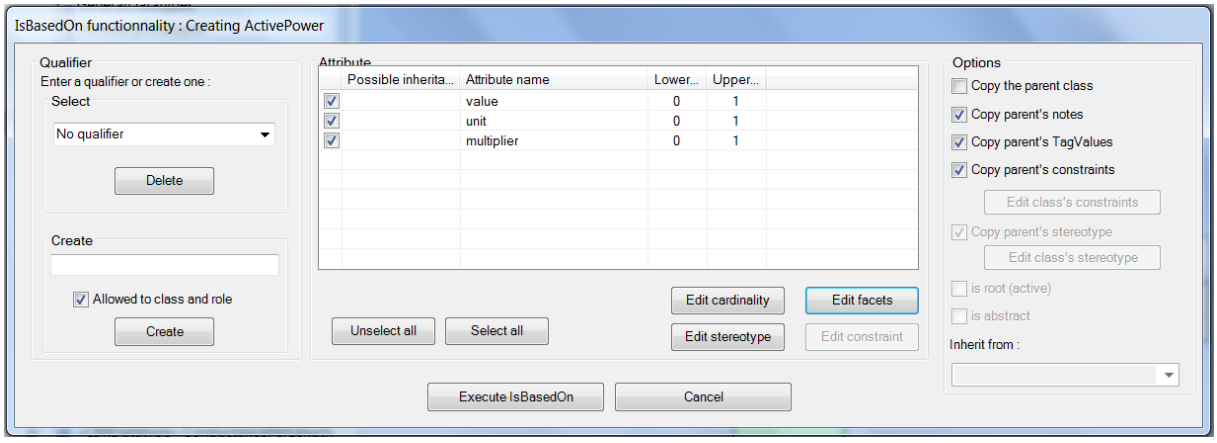
7.4. Edit an IsBasedOn enumeration:

If you need to change some choices on an IsBasedOn enumeration, or update the profile enumeration according to Information Model new version, you could use the Edit an IsBasedOn.

Note: in this version do not use the EditIsBasedOn on profile UnitSymbol enumeration neither on Currency enumeration. Delete the profile enumeration and redo a drag and drop of the CIM enumeration.

8. Create "IsBasedOn Datatypes" with CimConteXtor

When the drag and drop element is a <<CIMDatatype>>, it means that you want to restrict the value space, the value domain or both of the <<CIMDatatype>>. The "*IsBasedOn*" window for a <<CIMDatatype>> will be displayed as follows:



8.1. Qualify the IsBasedOn Datatype

To qualify the name of a datatype see section 5.2 (use and management of qualifiers):

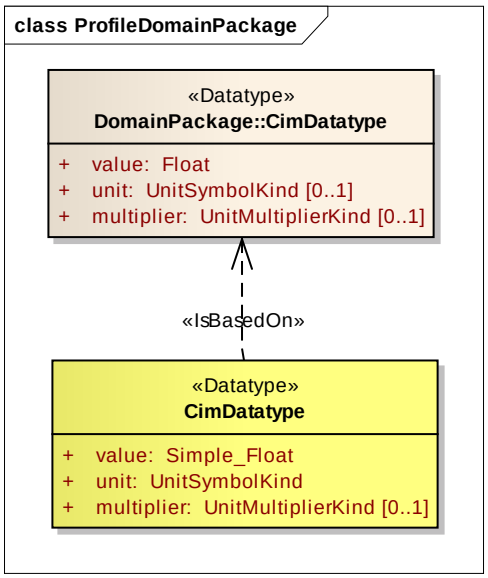
- If the "*Must qualify datatype and enum*" box of the "*Options*" Menu has been checked, the "*IsBasedOn*" datatype name will have the name of the datatype it is based on prefixed by a Qualifier. *CimConteXtor* will force you to enter a qualifier for this datatype.

- If the box is unchecked, the datatype could have the same name as the datatype it is "*IsBasedOn*", but this is not a good practice at all.

Remember that if you want to be able to reuse any primitive, datatype, enumeration or compound in any profile, their names must be unique in the Information Model and Profiles name space.

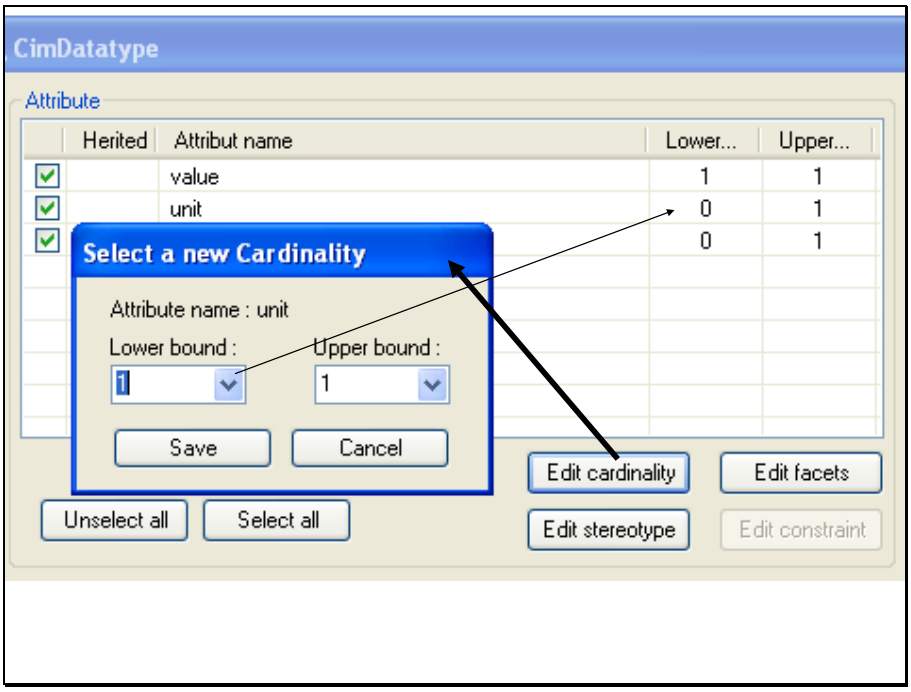
8.2. Copy Parent Element

Selecting "*Copy parent's class*", allows the inclusion of the Information Model enumeration along with the corresponding "*IsBasedOn*" enumeration, and shows the "*IsBasedOn*" dependency link between the two elements :



8.3. Define attribute cardinality

Select an attribute and click on the "*Edit cardinality*" button. This will display the "*cardinality*" window. Select appropriate cardinality and save :



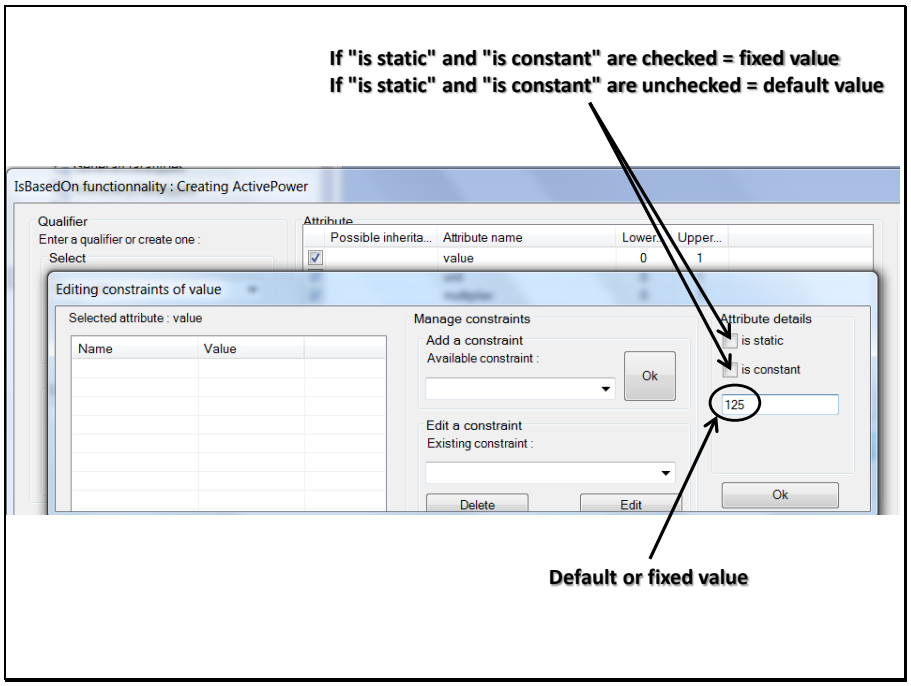
8.4. Define CIMDatatype value space

The value space could be defined as described in section 6.6 (define value space restriction).

8.5. Define default or fixed value

The type window could also be used to set up:

- a default value if the "*default value field*" is filled,
- or a fixed value if there is a value in the "*default value field*" and the "*is constant*" and "*is static*" boxes are selected.



8.6. Define Datatype value domain

Select one of the other datatype attributes (other than attribute "*value*"), and refine if necessary its cardinality and type, as done for the attribute "*value*".

9. Create "isBasedOn Compound" with CimConteXtor

When the drag and drop element is a <<Compound>>, it means that an "*IsBasedOn*" <<compound>> class should be created. Apart from the fact that a <<compound>> class has no associations, the basic "*IsBasedOn*" process is nearly the same as the one for a class : see next section on "*Create IsBasedOn Class with CimContextor*".

The only difference with class "*IsBasedOn*" process is the use of qualifiers:

- If the "*Must qualify datatype and enum*" box of the "*Options*" Menu has been checked, the "*IsBasedOn*" compound name will have the name of the compound it is based on prefixed by a Qualifier. *CimConteXtor* will force you to enter a qualifier for this compound.
- If the box is unchecked, the datatype could have the same name than the datatype it is "*IsBasedOn*", but this is not a good practice at all.

Remember that if you want to be able to reuse any primitive, datatype, enumeration or compound in any profile, their names must be unique in the Information Model and Profiles name space.

10. Create “IsBasedOn Class” with CimConteXtor

10.1. Overview

When the drag and drop element is a class, it means that an “*IsBasedOn*” class should be created, and that the properties of this class are going to be restricted.

There are two behaviours for this IsBasedOn drag and drop feature:

- *The **basic one (described here in section 10),** where the result of the drag and drop is a profile class. This is the default feature.*
- *The **enhanced one (described in annex H),** where the result of the drag and drop is a profile class with all its super classes.* This is used for example in WG13 or CGMES profiling style. This feature is activated by the checking of the " WG13AutomaticAnscesterInProfile"in the configuration file.

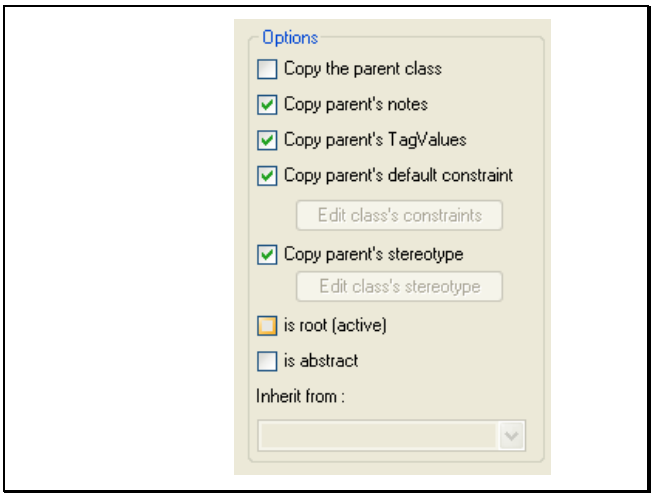
Reminder : classes have two kind of properties :

- the attributes (simple properties),
- the associations (complex properties).

The drag and drop function will let you qualify the class name, define its information and define its attributes based on the parent class. It will not perform class association definition: for this definition the “*Edit connectors*” function will be used (see “*Edit Connectors*” section).

10.2. Select Information for the IsBasedOn class (Options Menu)

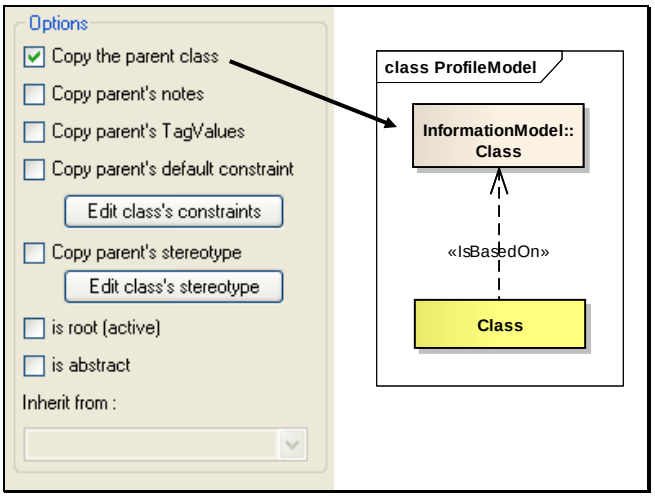
The “*Options*” Menu allows you to define all “*IsBasedOn*” class information : Notes, TaggedValues, Constraints, Stereotype, isRoot Class, isAbstract Class.



10.3. Copy parent class

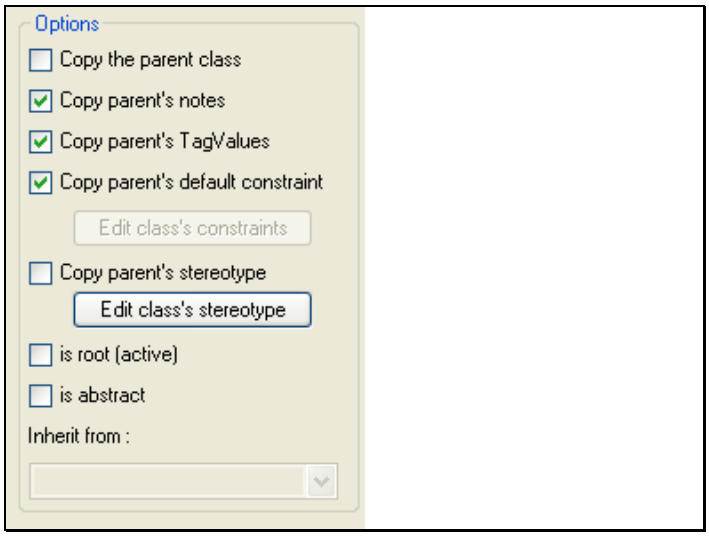
Copy the information Model class in the profile diagram : this option allows the inclusion in the profile diagram of the Information Model class along with the

corresponding profile class, and shows the "*IsBasedOn*" dependency link between the two classes :

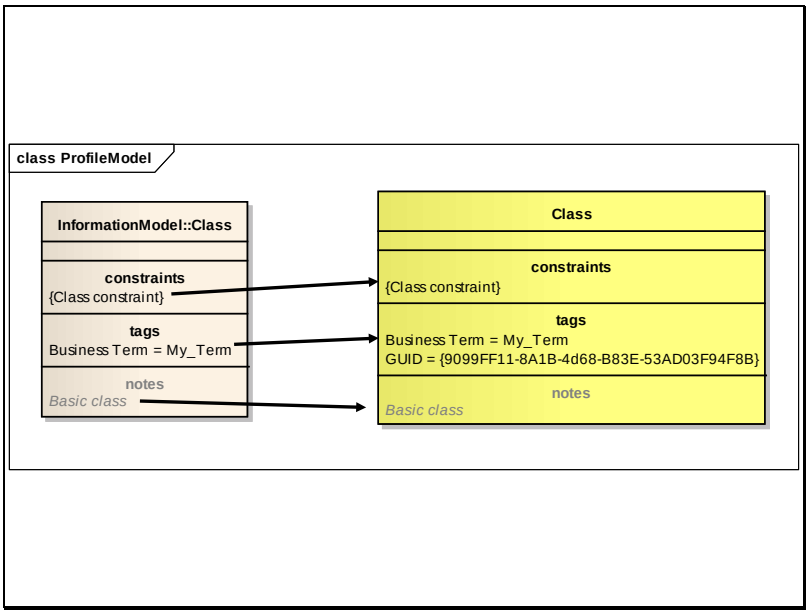


10.4. Copy profile class Notes, TaggedValues, and Constraints

Copy data from the Information Model Class. This option (checking appropriate boxes) allows the copying of all information attached to the Information Class in the Profile Class :



The copied information will be displayed in EA diagram if appropriate EA feature visibility is selected:



Note : the GUID (internal unique identifier attributed by EA for a Model element) TaggedValue is always added to a profile element

10.5. Edit profile class Notes and TaggedValues of the profile (IsBasedOn) class.

To Edit new Notes and select other TaggedValues for the IsBasedOn Class: just use the usual EA class property window.

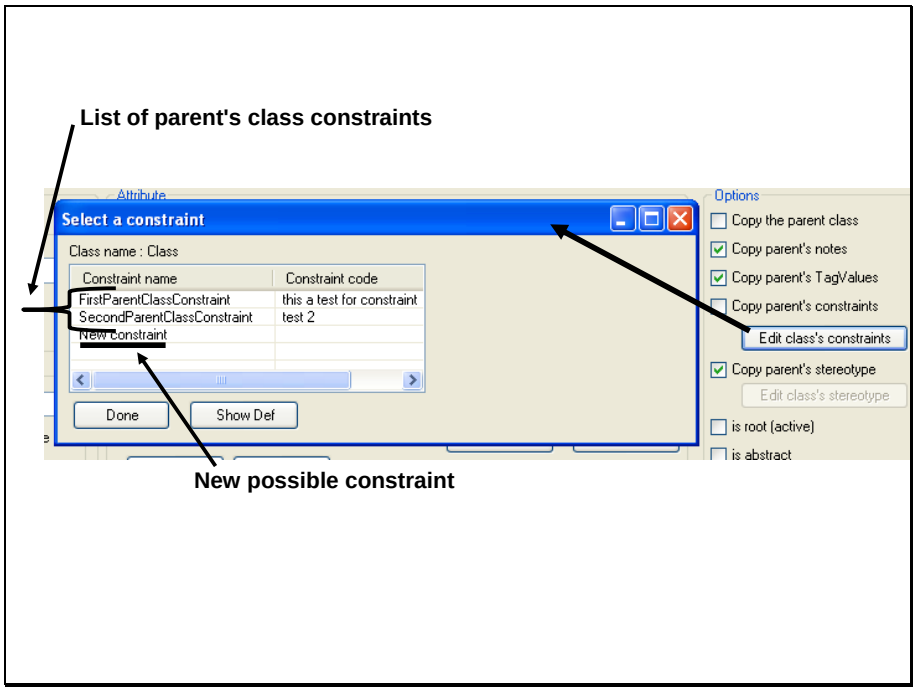
10.6. Edit profile class Constraints

10.6.1. Overview

A profile class might have for constraints:

- a copy of parents class constraints, a selection of parents class constraints, and use of any of them with or without modification,
- new constraints.

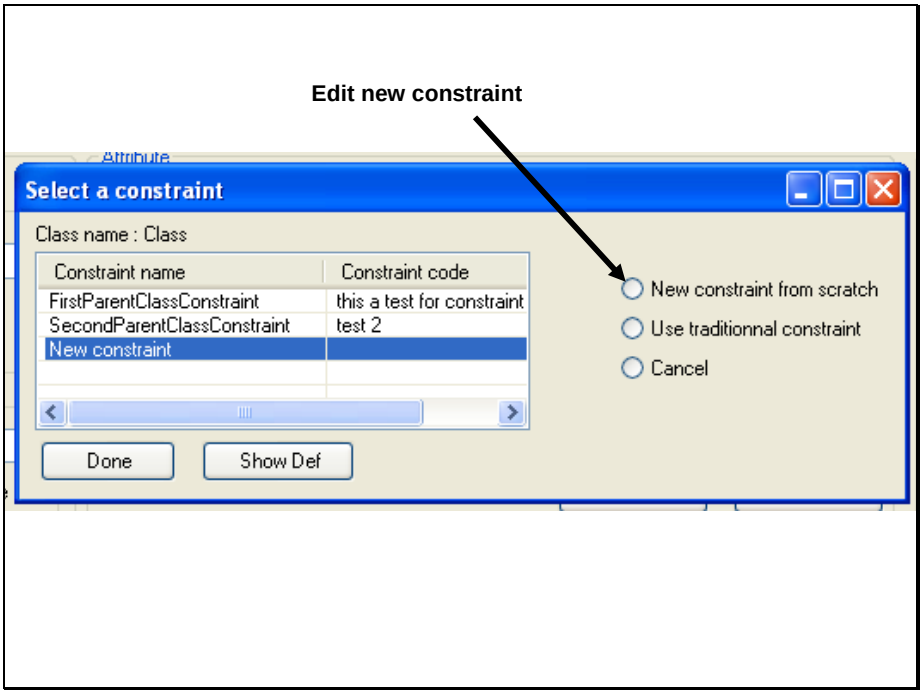
To edit constraints for the Profile Class: click on the "*Edit Class's constraints*" button, a new pop-up window appears and displays a list that includes copy of parents' class constraints (if they are available) and potential new constraint:



To Edit a new constraint for the profile class, select "*New constraint*" (it will be highlighted in blue), the window changed and offers you a choice for constraint editing: edit a new constraint from scratch or use a predefined constraint.

10.6.2. Edit a new constraint from scratch :

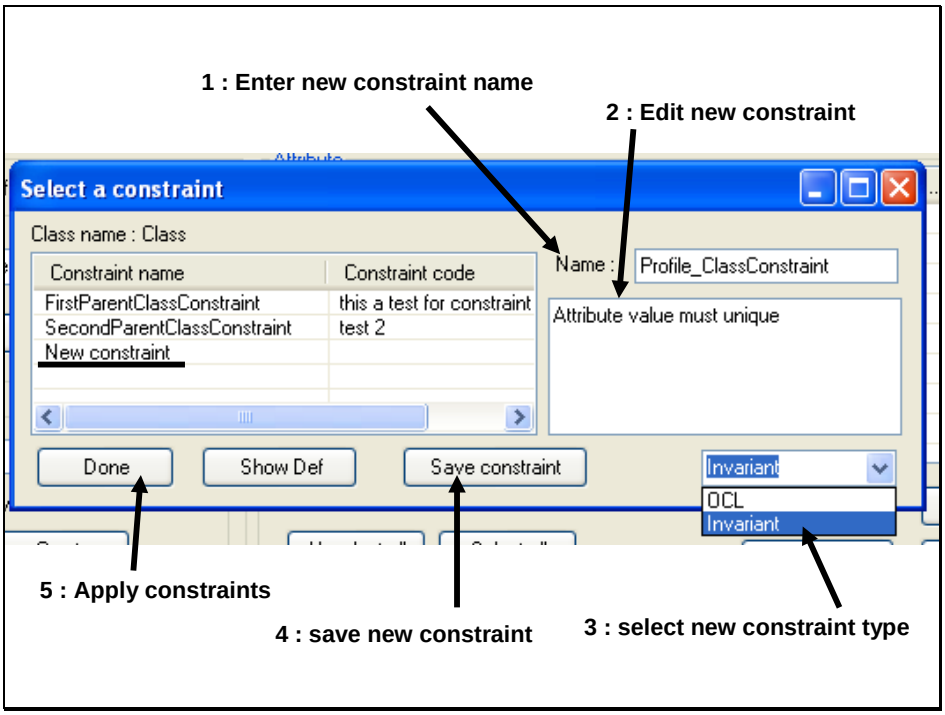
Select "*New constraint from scratch*"; this changes the window as follows:



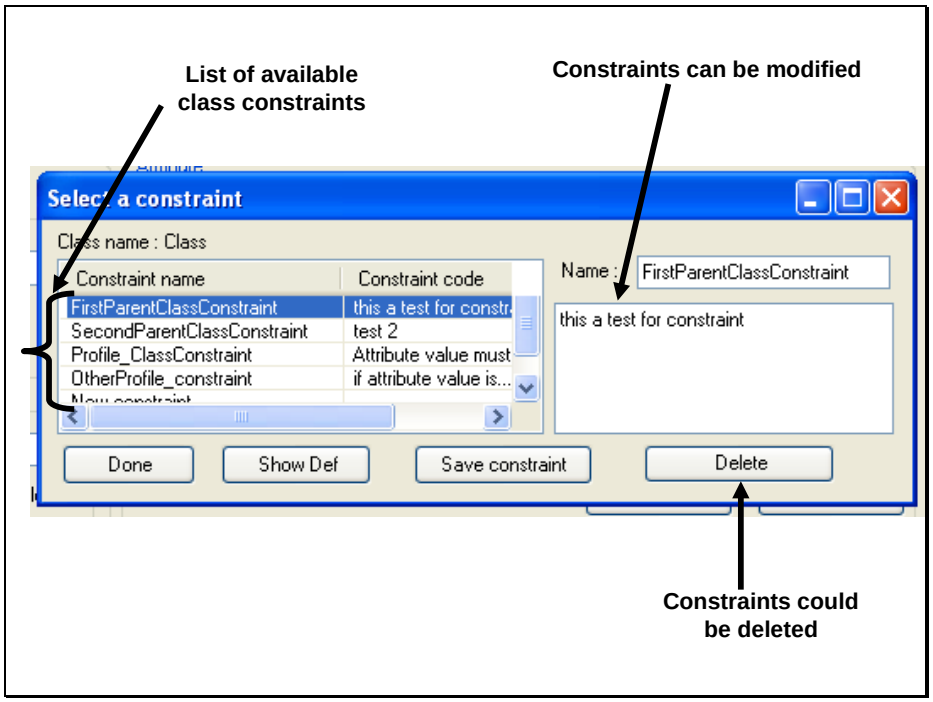
To edit the new constraint:

1. First enter the name of the constraint,
2. Edit the constraint ,

3. Select the constraint type (the list of constraint type is managed by EA Menu and is configurable, by default it is OCL or INVARIANT). Usually, the type will be the language in which the constraint is expressed,
4. Click on "*Save constraint*" button,



5. Then you can add other constraints and manage the list of constraints.

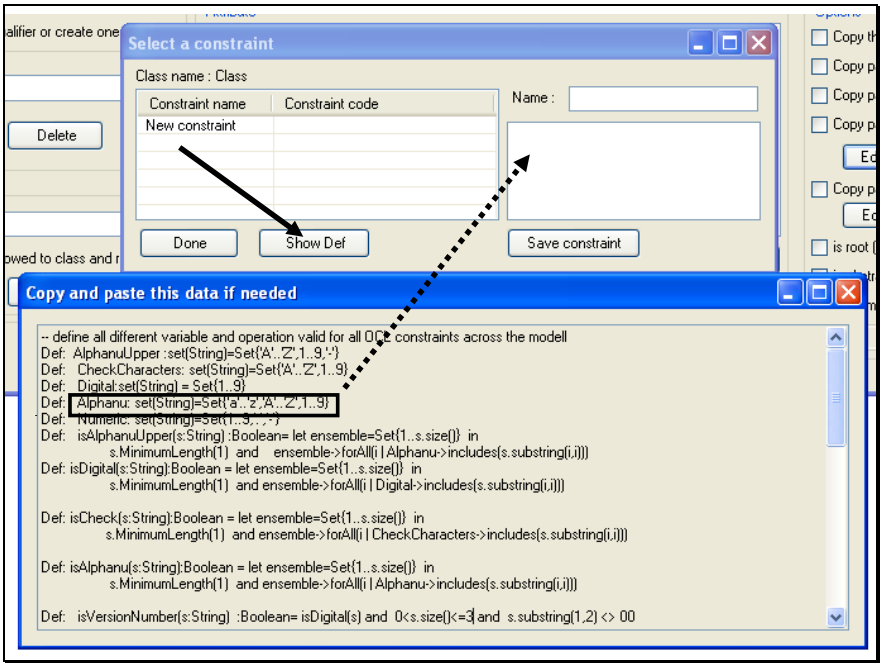


Constraints could be modified and deleted.

Finally, apply the constraints to the profile class by clicking on "*Done*".

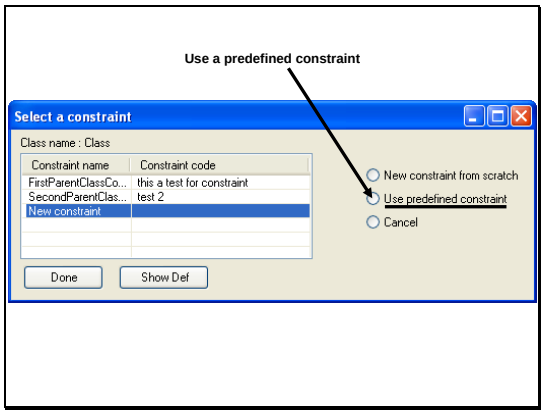
10.6.3. Use of constraint template

Constraint templates may be edited in the configuration file of *CimConteXtor*. Then this template could be used to edit new constraints. These templates could be accessed through the "*Show Def*" button, which will display in a "*Copy and paste this data if needed*" window, where the constraint template could be copied and pasted:

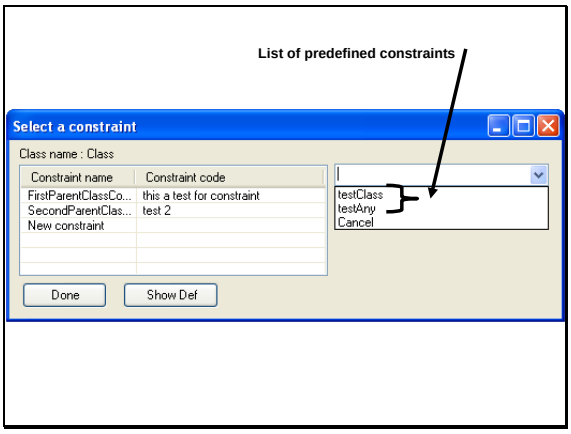


10.6.4. Use of predefined constraints:

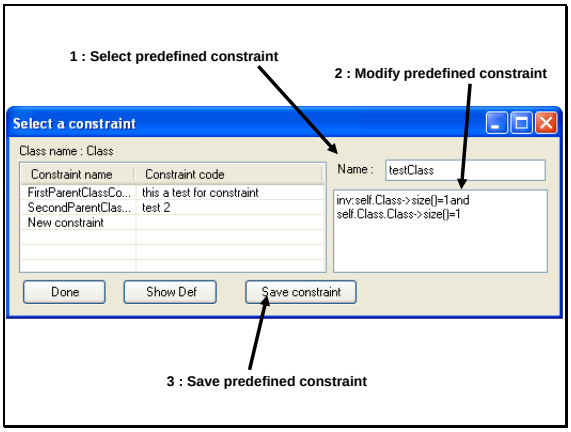
Predefined constraints are edited in the configuration file of *CimConteXtor*. To use them, select "*Use predefined constraint*":



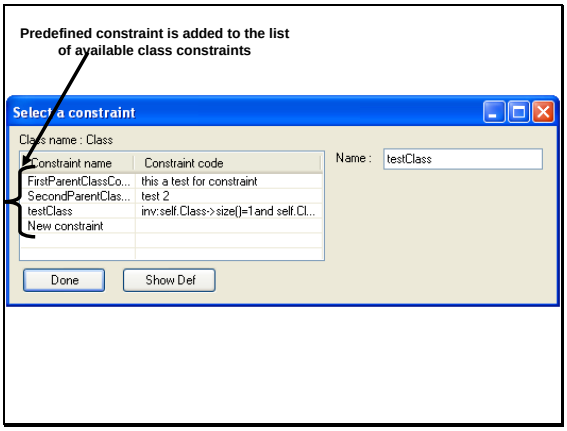
The following window is displayed, and a drop-down menu gives the list of predefined constraints:



Select a predefined constraint, modify it if appropriate and save:

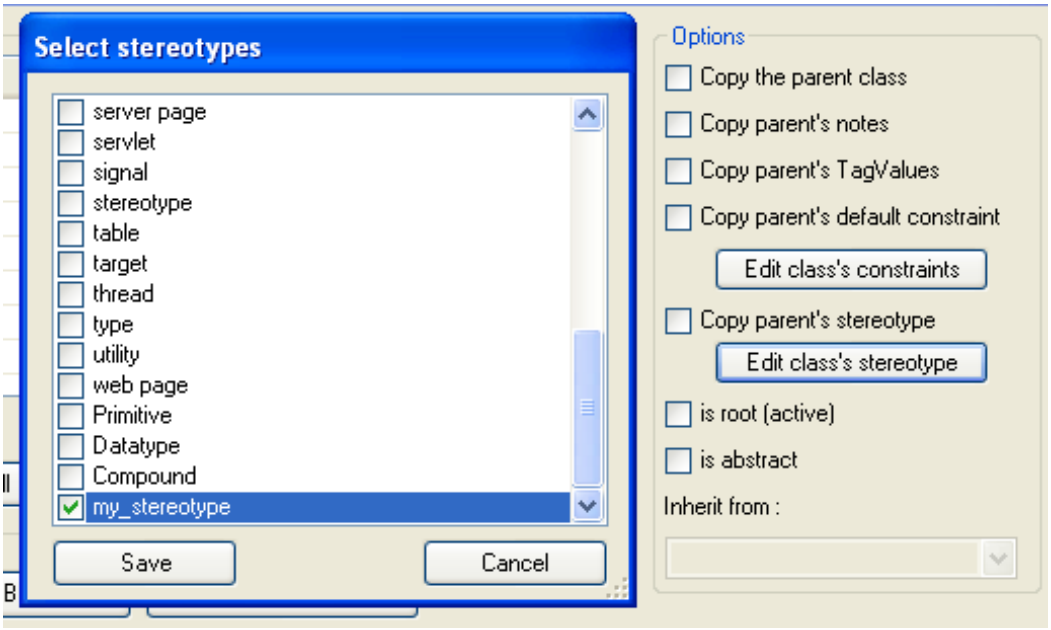


The predefined constraint is added to the list of class constraints:



10.7. Edit profile class Stereotype

To edit a new stereotype for the Profile Class: do not select "*Copy parent's stereotype*" in the "*Options Menu*". By doing that, the "*Edit Class's stereotype*" button is made available. By clicking on it, a new pop-up window appears that displays all possible class stereotypes:



Select a stereotype

10.8. Define profile class as a Root class

Select the “is root (active)” box, if the class is a root class for an XSD implementation. (note, the CimConteXtor “is root (active)” is displayed in the EA class property window as an “Is Active” property and not as an “Is Root” one).

10.9. Define profile class as an Abstract Class

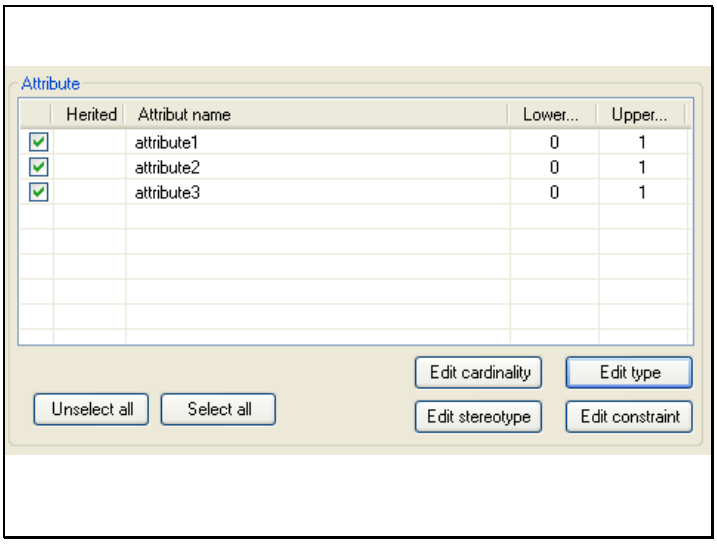
Select “*is abstract*” box, if the profile class is abstract (see “*Use of inheritance for profile class*” section).

10.10. Inherits

This option is explained in the “*Use of inheritance for profile class*” section.

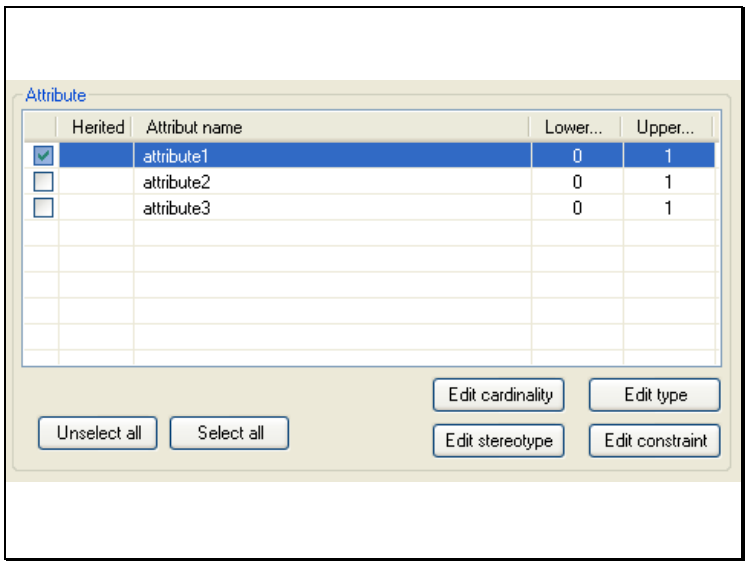
10.11. Define Attributes for Profile Class (Attribute Menu)

All the attribute definitions are in the attribute part of the “*IsBasedOn*” window :



10.12. Select attributes

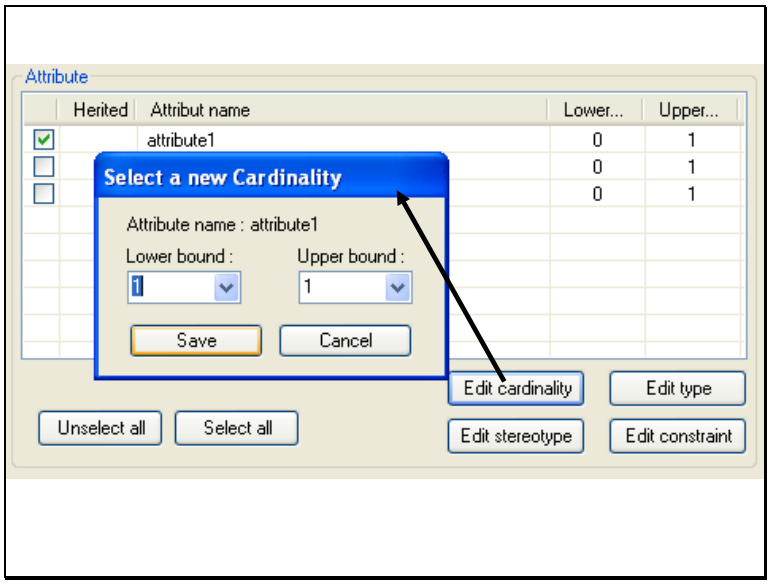
To select attributes that will become properties of the profile class, check appropriate attribute's box:



Note : attributes that are mandatory cannot be deselected.

10.13. Define attribute's cardinality (optional or mandatory)

To refine an attribute's cardinality, click on the attribute whose cardinality you want to refine (it will be highlighted in blue), then click on "*Edit cardinality*" button, the "*Edit cardinality*" window will be displayed :

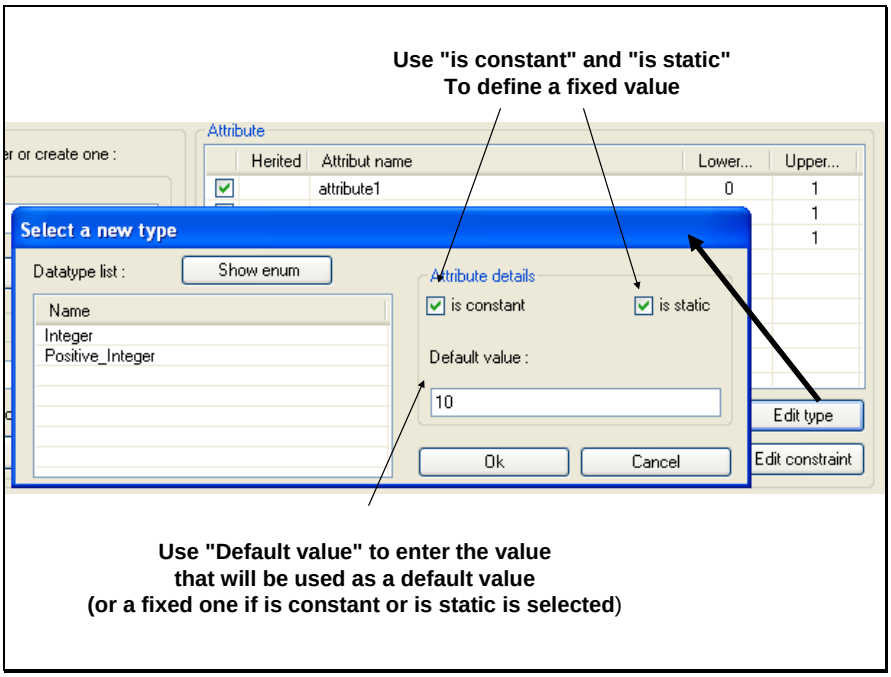


For attributes, the only allowed cardinalities are optional and mandatory : mandatory being a more restricted cardinality than the optional one.

10.14. Define attribute's default or fixed value

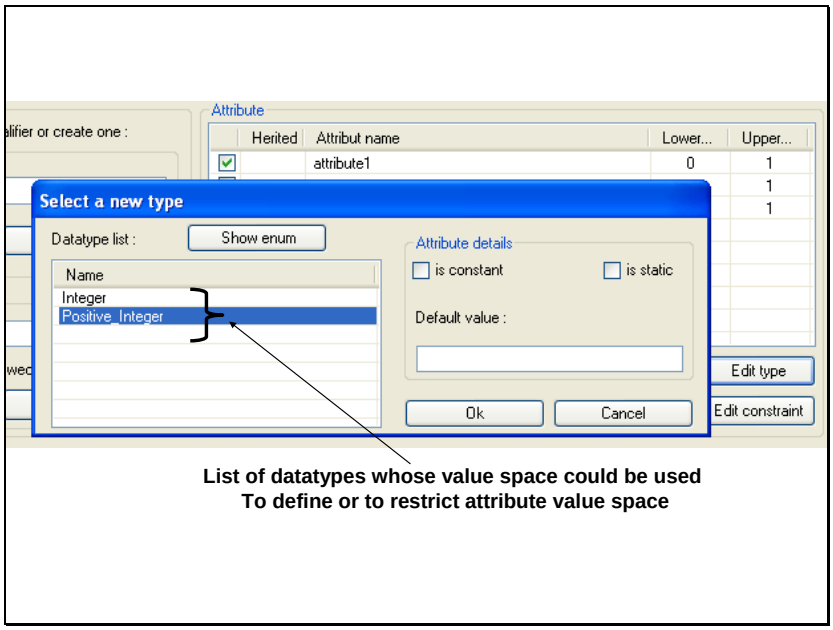
To define default or fixed value for an attribute, click on the "*Edit type*" button, then the "*Edit type*" window will be displayed. Then you can set up :

- a default value by entering a value in the "*default value"* field,
- **or** a fixed value if there is a value in the "*default value field*" and the "*is constant*" and "*is static*" boxes are selected.

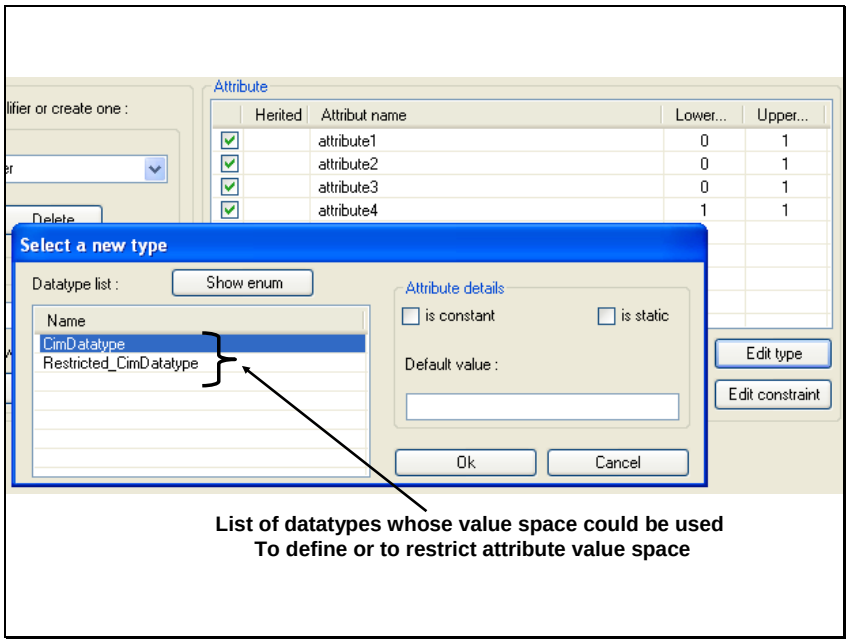


10.15. Edit attribute type:

To define the value space (and value domain if appropriate) of an attribute, select the attribute (it will be highlighted), then click on the "*Edit type*" button, the "*Edit Type*" window will be displayed:

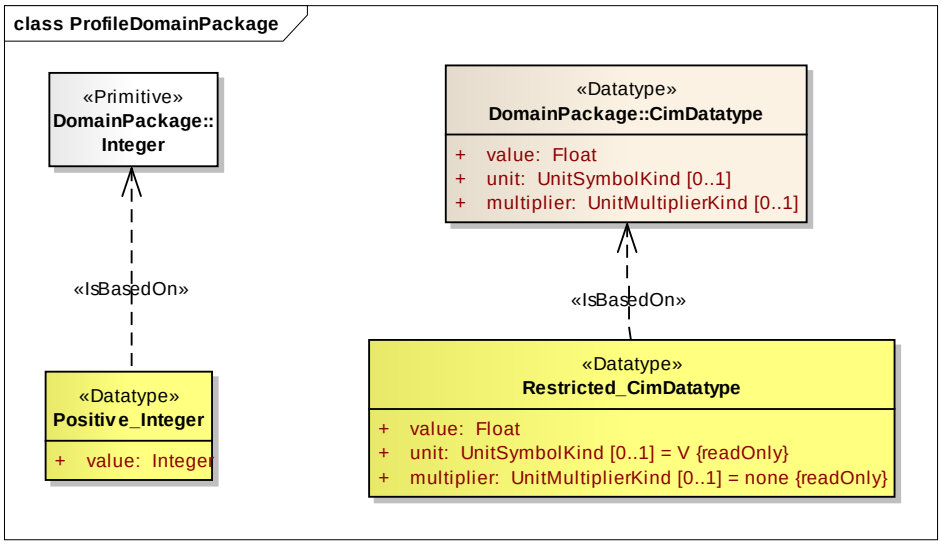


or



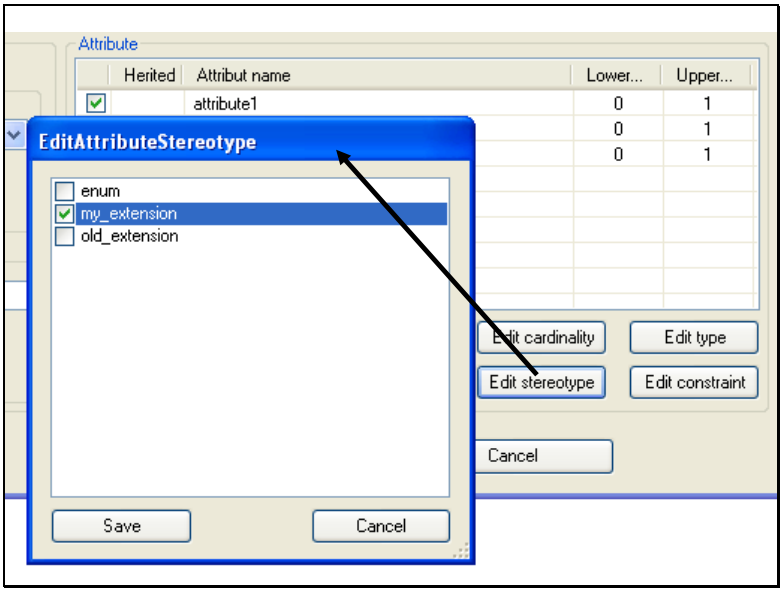
A datatype list is displayed and shows which **already existing** datatypes could be used to restrict the attribute's value space, and/or its value domain. Select the appropriate datatype.

Remember: datatypes are organized in a hierarchy of "*IsBasedOn*" datatypes, and the list will give an idea of the hierarchy, and thence all the possible restrictions. *The first datatype is always the least restricted one.*



10.16. Edit attribute stereotype

To edit attribute's stereotype, select the attribute, then click on "*Edit stereotype*" button, the "*EditAttributeStereotype*" window will be displayed. The list of all available attribute's stereotypes will be shown, check the appropriate stereotype and "*Save*":

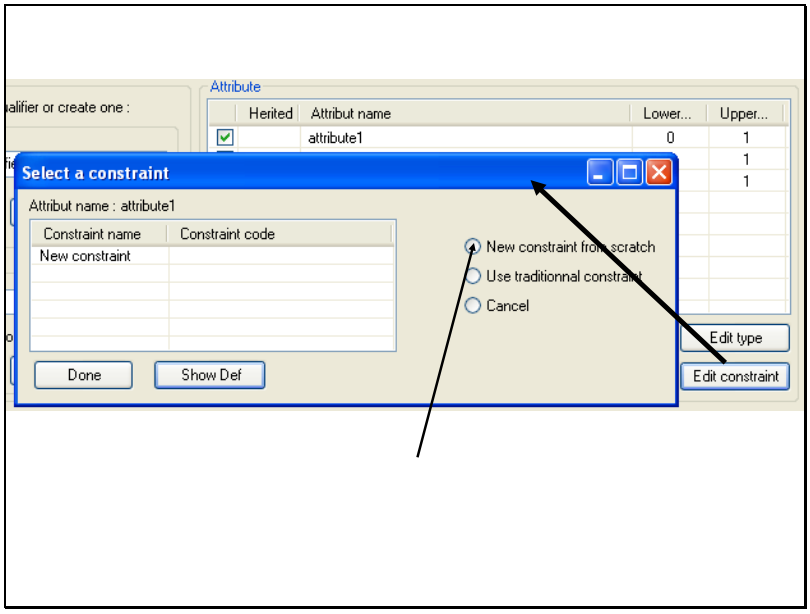


- **Edit attribute constraint**

Remember, make difference between :

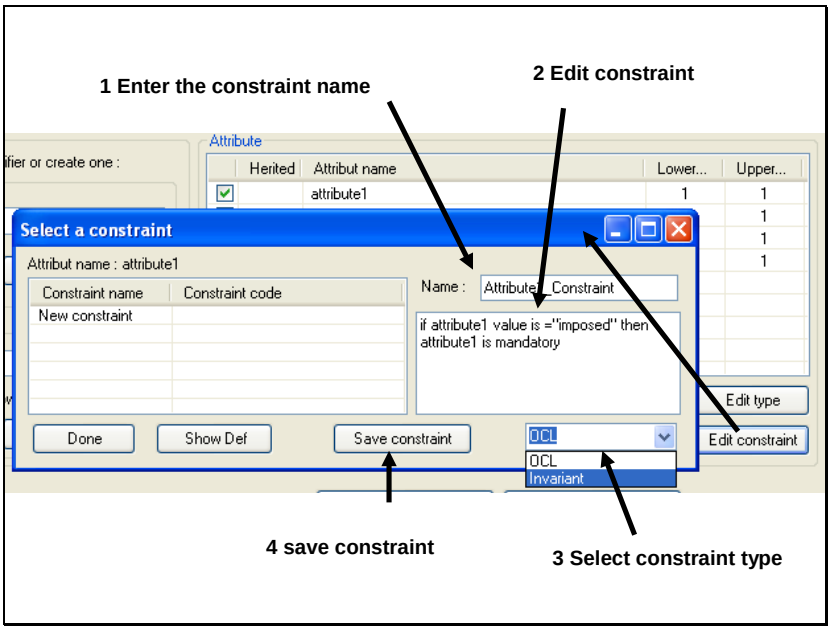
- the "*Edit facets*" that is used to restrict the value space of a datatype,
- the "*Edit type*" that is used to select the type of an attribute,
- and "*Edit constraint*" that is used to express attribute's constraint (example the value of the attribute should be unique).

To edit an attribute constraint, select the attribute (it will be highlighted), then click on the "*Edit constraint*" button, the "*Edit Constraint*" window will be displayed. To define a new constraint, click on the "*New constraint from scratch*" :



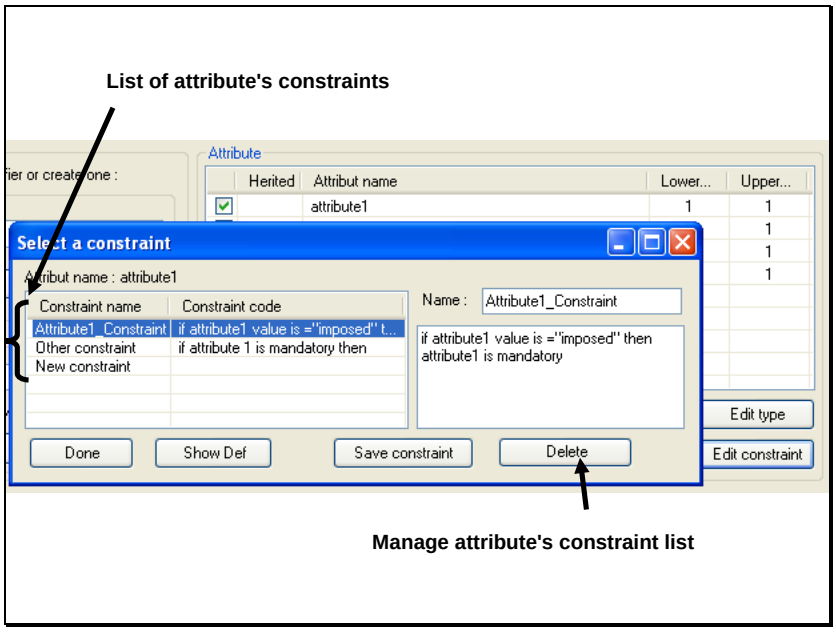
Define attribute new constraint:

1. Enter a name for the constraint
2. Edit constraint
3. Select constraint type



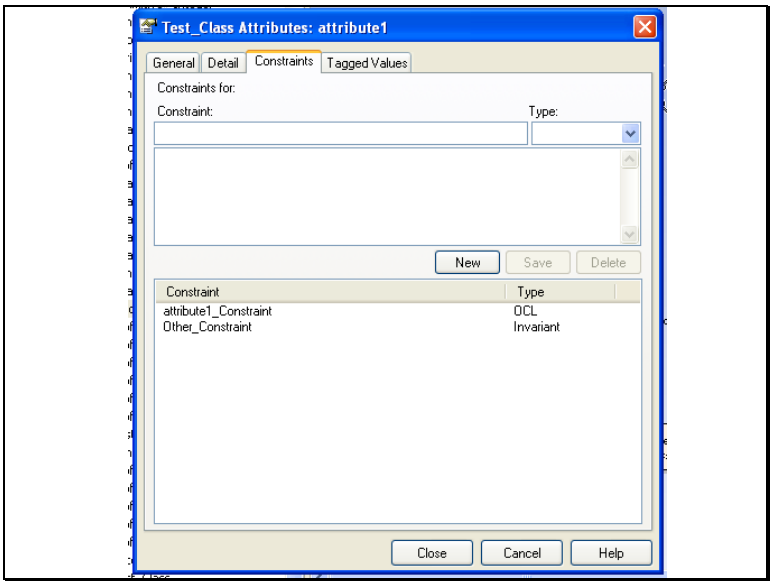
4. Then "*Save*".

The constraint has been added to the list of the possible constraints for the attribute:



Attribute constraints could also be done using constraint template and predefined constraints. These functions are the same as those used for class constraint editing.

The attribute constraints could be seen in the corresponding EA attribute property window :



11. Modify an IsBasedOn Element

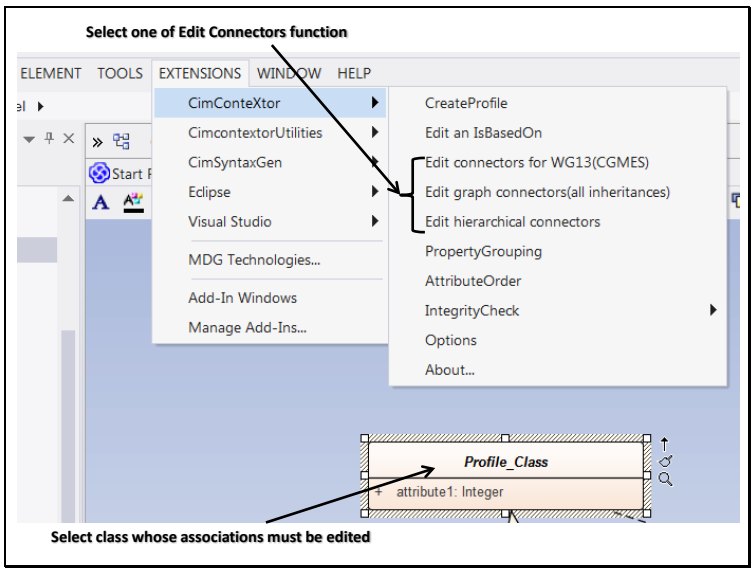
To modify an "*IsBasedOn*" (or profile) element (enumeration, datatype, compound or class) uses the "*Edit an IsBasedOn*" function.

12. Create Associations for profile classes: use of “Edit Connectors”

12.1. Overview

Once profile classes have been edited, then associations between them can be specified. Here we describe associations editing between profile classes, in the case of profile classes that stand by themselves: this means that the inheritance functionality is not used and so that there are no profile super classes. (See Inheritance section if you want to use inheritance in your profile and have profile super classes).

To specify associations for a profile class, click on the class and then go to the “*CimConteXtor Menu*” and select one of the “*Edit connectors*” functions:

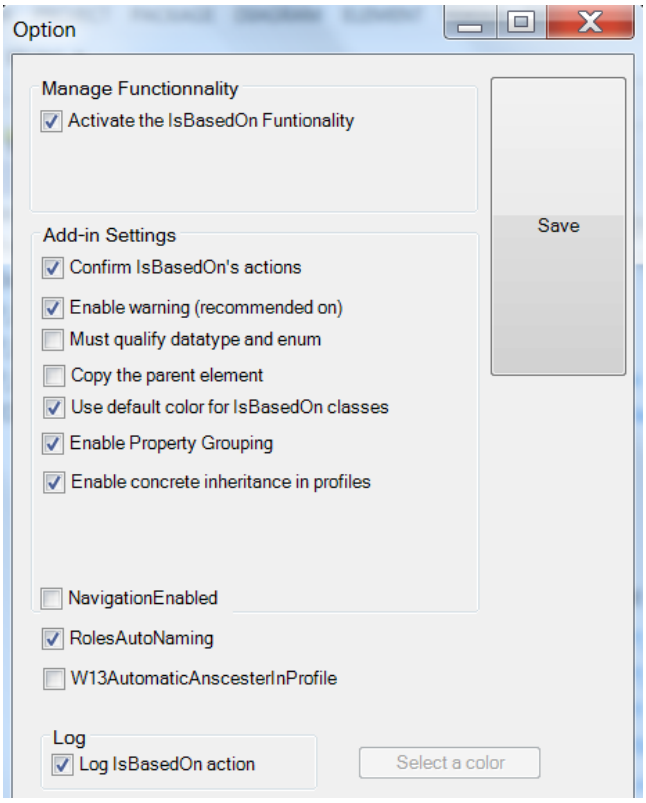


12.2. Edit connectors functions:

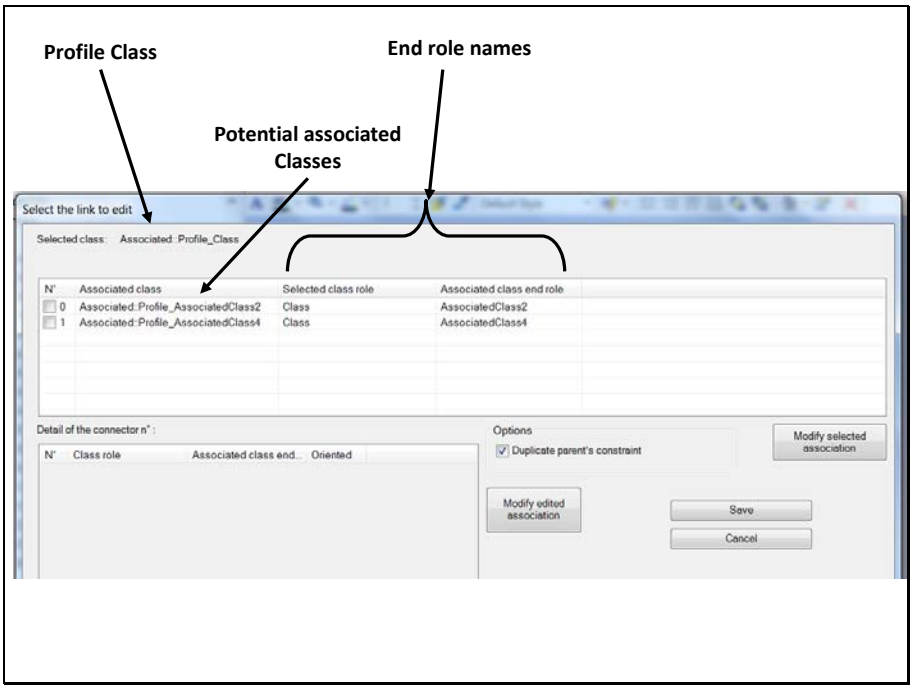
There are three different "*Edit Connectors*" functions for three different profile targets:

- two (Edit Connectors for WG13 and Edit Graph Connectors) targets graph profiles (no hierarchy):
  - simple graph according to IEC 61970-452 (IEC/TC57/WG13) or CGMES (Entso-E),
  - complex graph.
- One targets hierarchical profiles (according for examples to IEC/TC57/WG14 profiles or IEC/TC57/WG16 European style Market Profiles whose syntactic output is XSD).

Warning: if you want to target hierarchical profiles, first go to the CimConteXtor Options Menu and if "*NavigationEnabled*" box is checked, unchecked it, and make a Save:

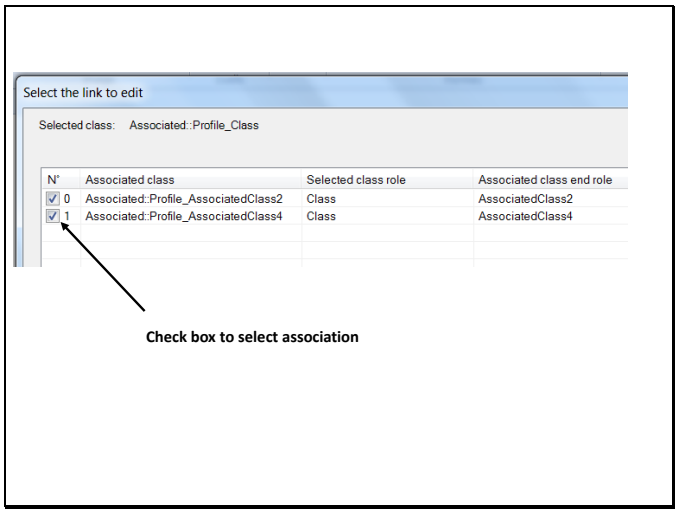


When you select one edit connector function, then a general "*Edit connectors*" window is displayed, whatever function (WG13, Graph or Hierarchical Edit Connectors) is selected :



12.3. Select profile class associations:

This is done by checking the association or connector box.

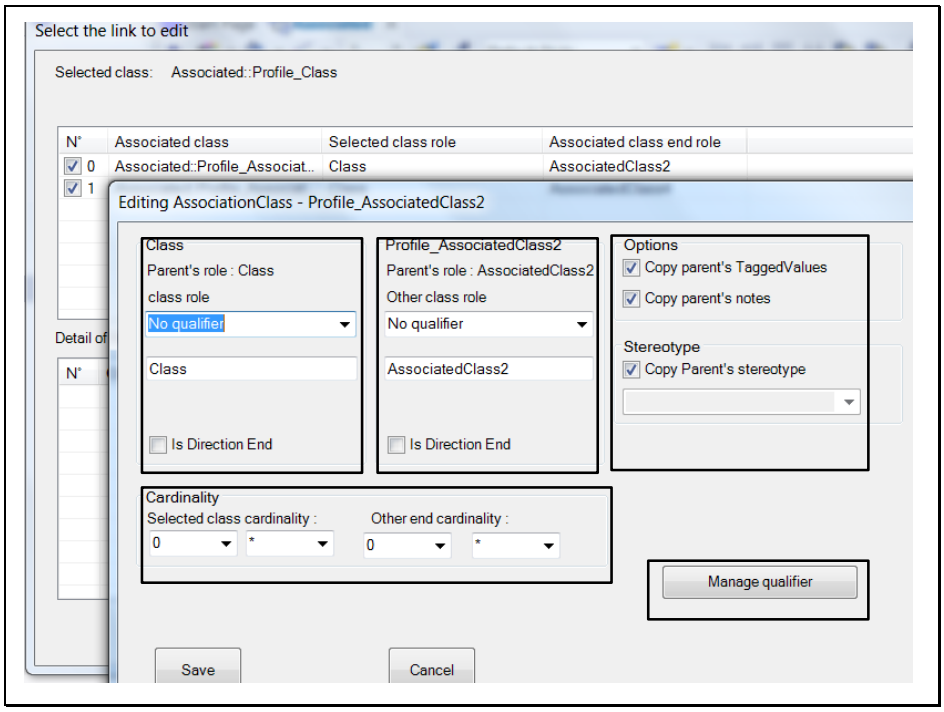


If you do not need to refine the association, then click on "*Save*" button: this will draw the lines that show associations between the selected profile class and the associated classes (and create the association in the profile model).

13. Refine profile class associations for WG13 or Graph Profile style

13.1. Overview

When you have used the WG13 or graph Edit Connectors function, and you have selected an association, then if you click on "*Modify Selected Association*" button, an "*EditingAssociation*" window pops up and allows the refinement of the "*IsBasedOn*" association as follows:



There are several parts in this "*Modify Selected Association*" window:

- Selected class role
- Other end role
- Options
- Stereotype
- Cardinality
- Manage qualifier

13.2. Copy TaggedValues or Notes :

If boxes in the "*Options*" part of the window are selected, the parent association TaggedValues and Notes will be copied in the "*IsBasedOn*" association.

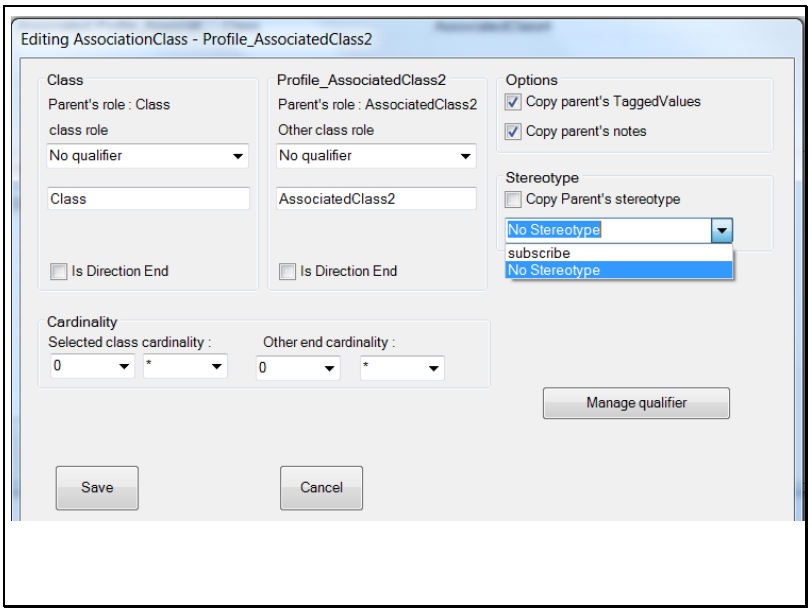
13.3. Edit Profile Association TaggedValues or Notes

To edit new TaggedValues or new Notes, use the usual EA association Properties window.

13.4. Association Stereotype

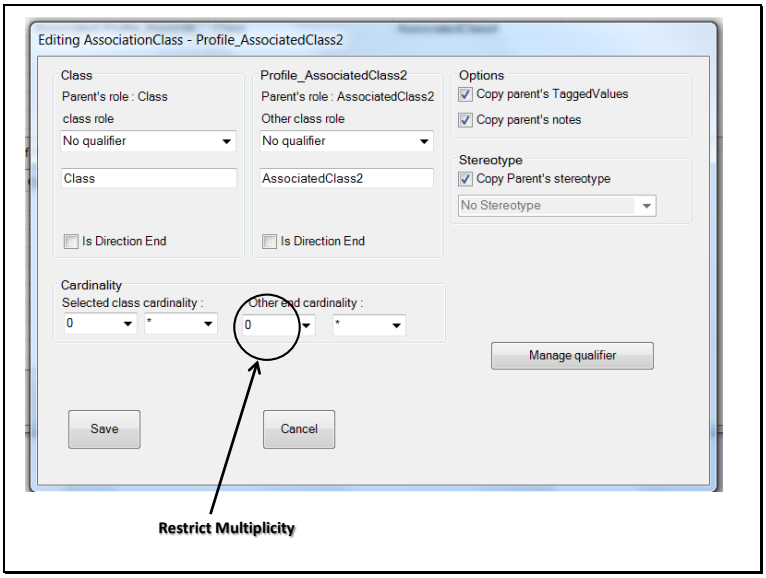
To copy the parent association stereotype check "*Copy Parent's stereotype*" box.

To edit a new stereotype, uncheck the "*Copy Parent's stereotype*" box and pick up a stereotype in the drop-down menu.



13.5. Restrict Association cardinality

To restrict association ends cardinalities, enter or select proper values in each field of either of the association end sides.

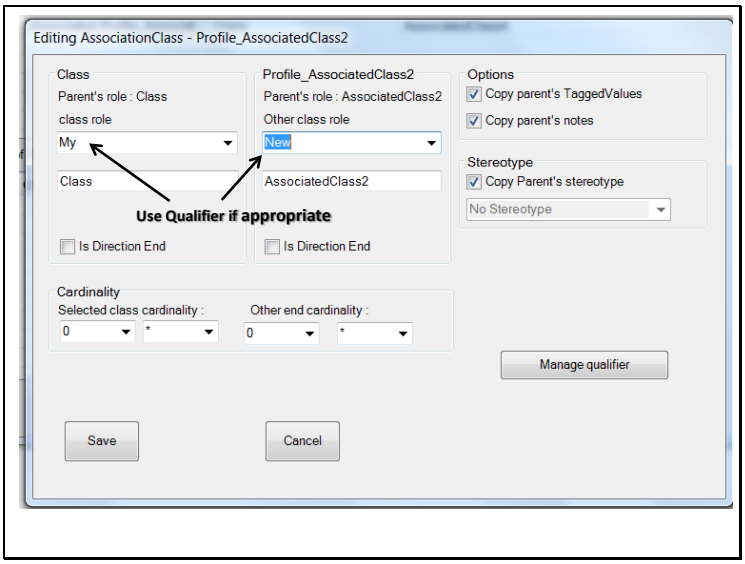


13.6. Qualify association end role name

Association end role names could be qualified:

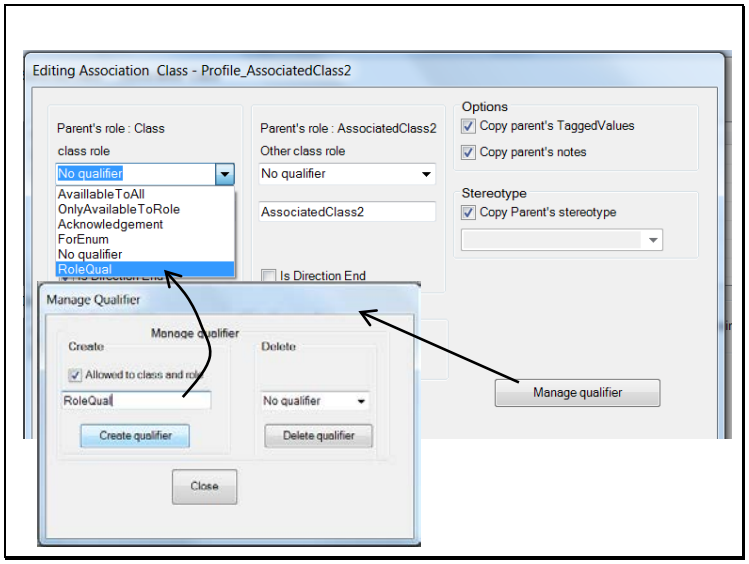
- either to express a more refined semantics of the end role name,
- or to distinguish end role name when there are multiple associations between two classes : in this case qualifiers are mandatory.

Enter qualifiers or select qualifiers in the drop-down menus for each end role name:



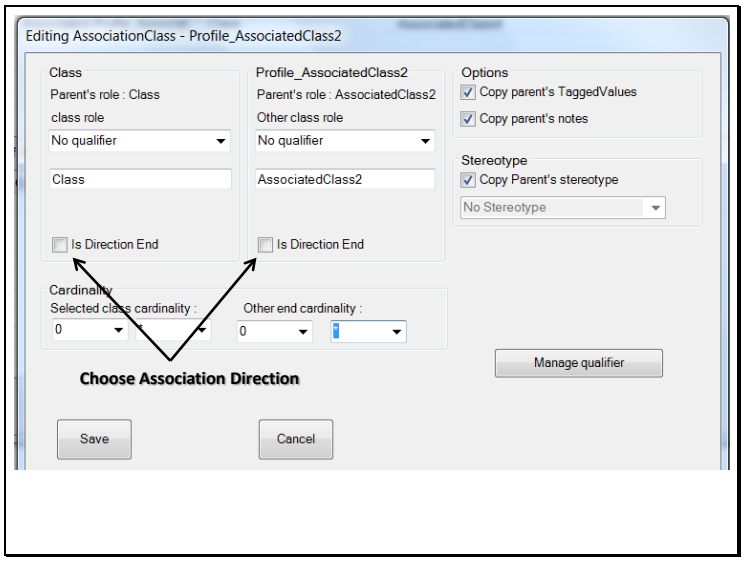
13.7. Managing end role name qualifier

Use the "*ManageQualifier*" button to enter new qualifiers (See also section 5.2.).



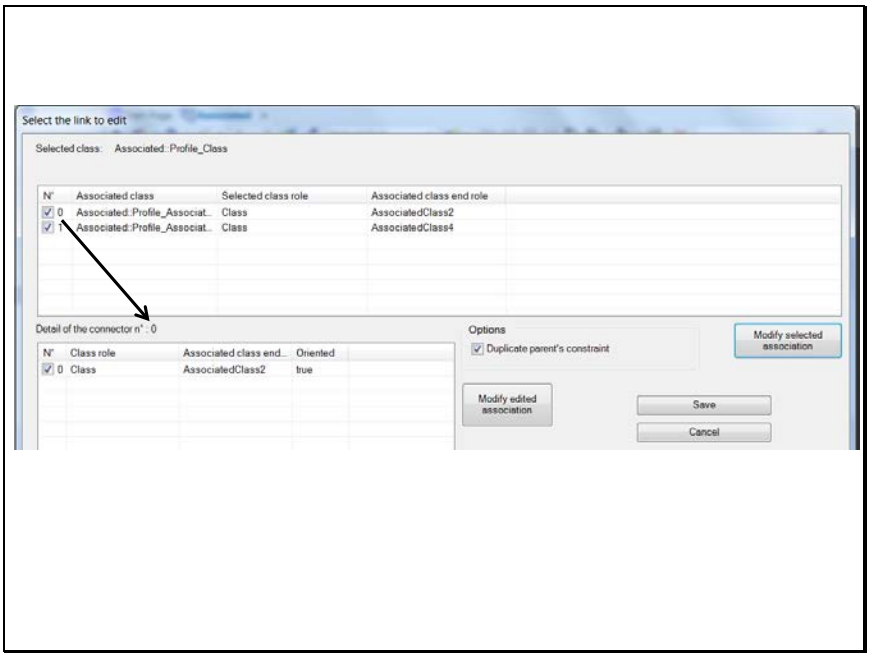
13.8. Select association direction

The association must be oriented for WG13 style design (The association will be drawn in UML diagram as an arrow). To do that, select the "*IsDirectionEnd*" box, on the side you want to have the arrow:



13.9. Save refined association

When you click on the "*Save*" button of the "*Modify Selected Association*" window, the edited association appears in the "*Detail of connector*" window part, where it shows that the association has been refined.

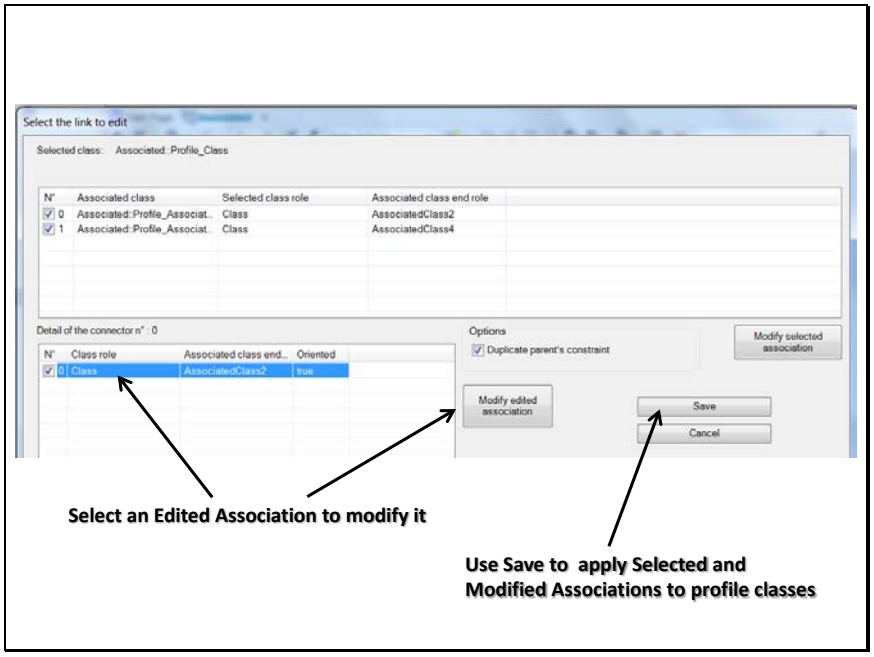


13.10. Edit a modified association

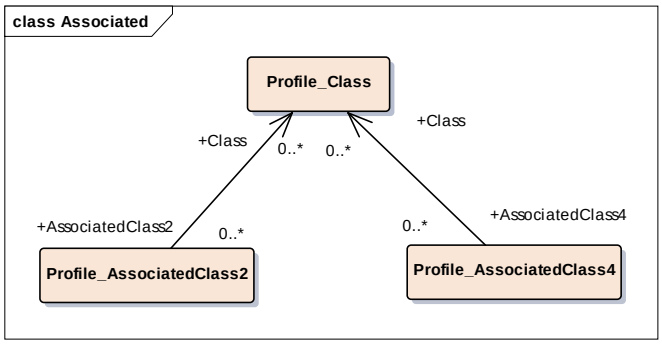
If you want to reedit a modify association, select the modify association in the "*Detail connector*" part and click on the "*Modify Edited Association*", this will display again the "*Editing Association*" window that will let you modify the modified association characteristics.

13.11. Create associations

To create the associations between a class and other ones, click on the "*Save*" button, this will draw associations in the diagram:



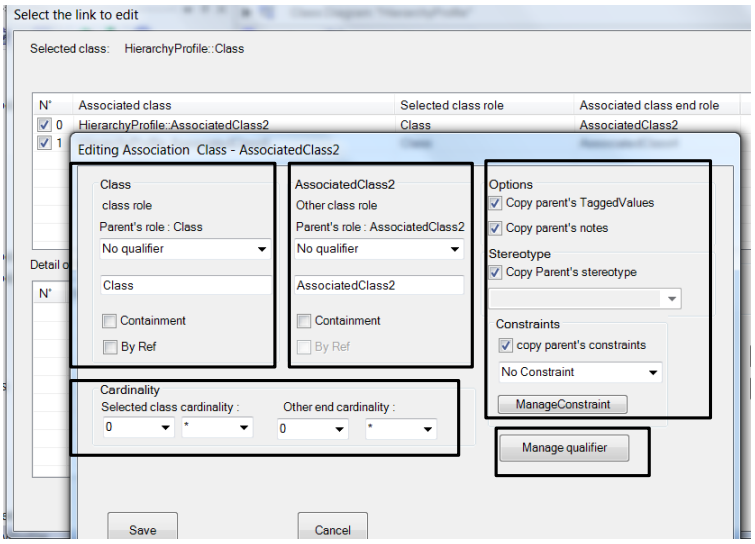
The result will be:



14. Refine profile class associations for Hierarchical Profile style

14.1. Overview

When you have used the Hierarchical Edit Connectors function, and you have selected an association, then if you click on "*Modify Selected Association*" button, an "*EditingAssociation*" window pops up and allows the refinement of the "*IsBasedOn*" association as follows:



There are several parts in this "*Modify Selected Association*" window:

- Selected class role
- Other end role
- Options
- Stereotype
- Cardinality
- Manage qualifier

14.2. Copy TaggedValues or Notes :

If boxes in the "*Options*" part of the window are selected, the parent association TaggedValues and Notes will be copied in the "*IsBasedOn*" association.

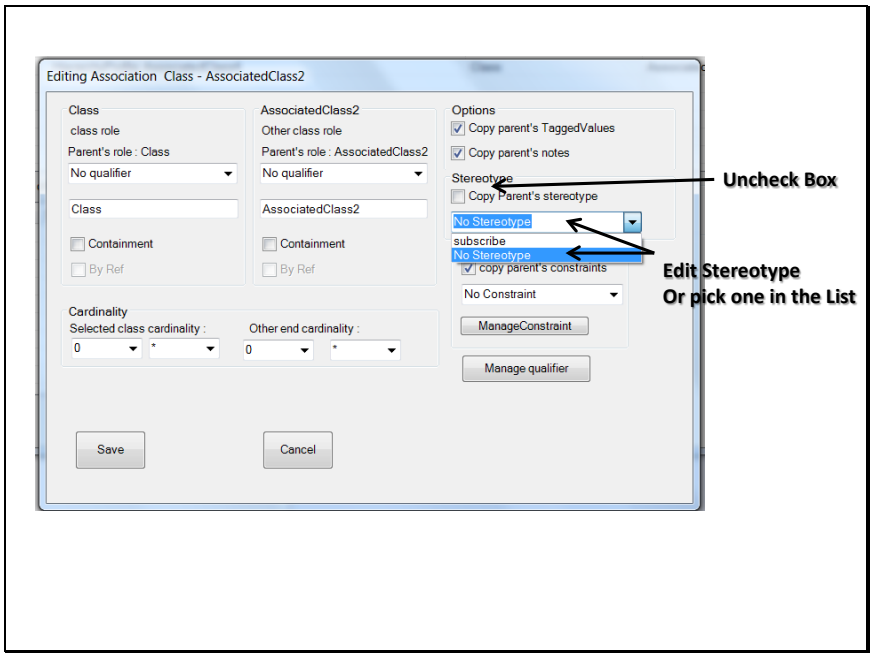
14.3. Edit Profile Association TaggedValues or Notes

To edit new TaggedValues or new Notes, use the usual EA association Properties window.

14.4. Association Stereotype :

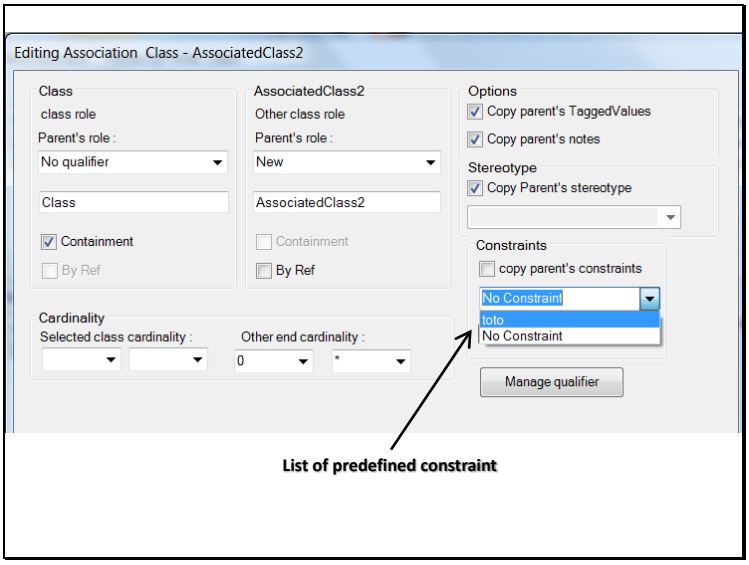
To copy the parent association stereotype check "*Copy Parent's stereotype*" box.

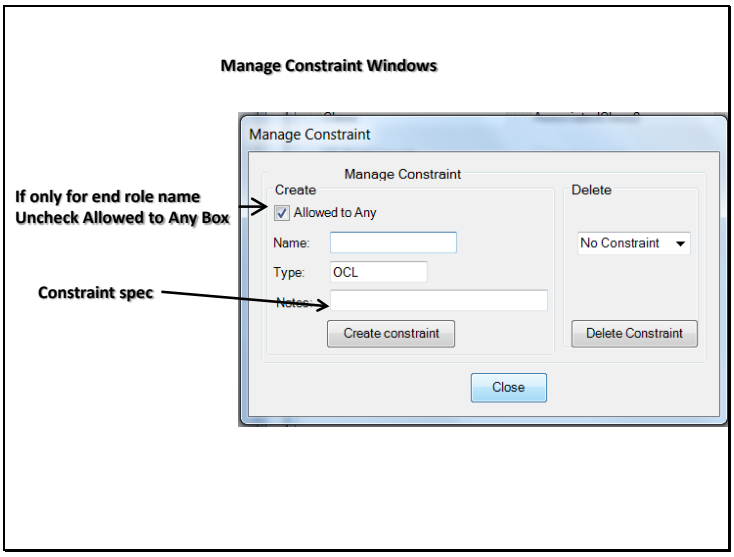
To edit a new stereotype, uncheck the "*Copy Parent's stereotype*" box and pick up a stereotype in the drop-down menu.



14.5. Association Constraint

See section 10.6 for using constraints already defined by the config file or by the "ManageConstraint" button. To attach a constraint to the association, get the list of constraints and select one. If you need to define a new (not existing in the list) use the "*ManageConstraint*" button to edit a constraint that will be added to the list.





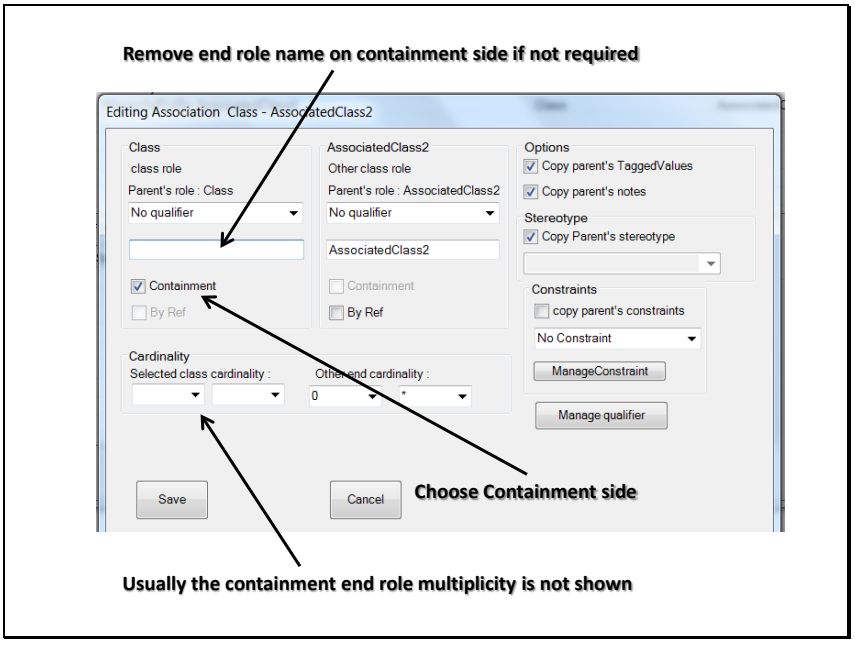
14.6. Manage Qualifier

See section 5.2

14.7. Select association type

The association must be restricted to a containment (usually aggregation is used). To do that, select the containment box:

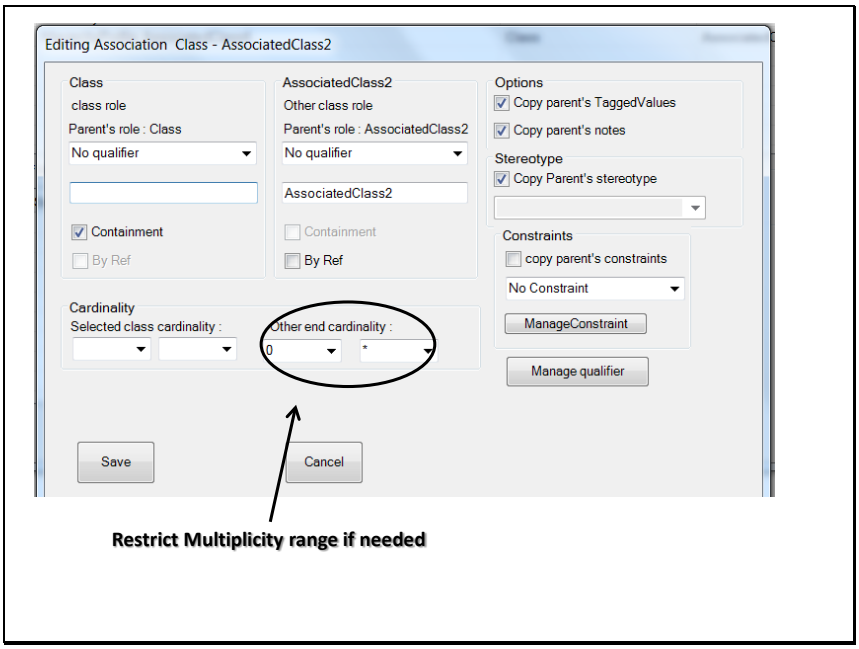
- This will turn down the containment end side cardinality, because usually the cardinality on this end is 1 (but it could be set up if needed),
- And usually, the containment end side role name is blanked, but it could be kept if needed; so if you do not need it, remove the end role name.



Note: in case the association is based on an information model super class, but is used with a subclass, then changing end role name to subclass name is possible if RolesAutoNaming option is checked (See inheritance section).

14.8. Restrict Association cardinality

To restrict association ends cardinalities, enter or select proper values in each field of either of the association end sides.

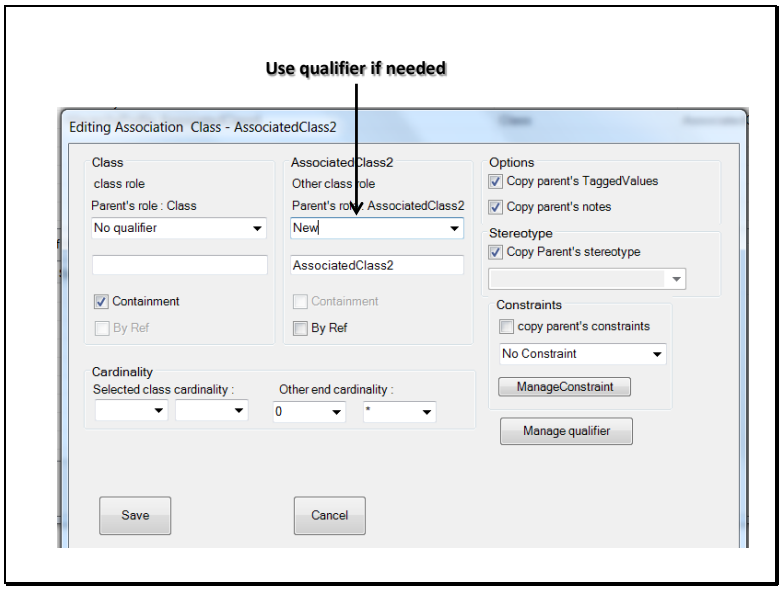


14.9. Qualify association end role name

Association end role names could be qualified:

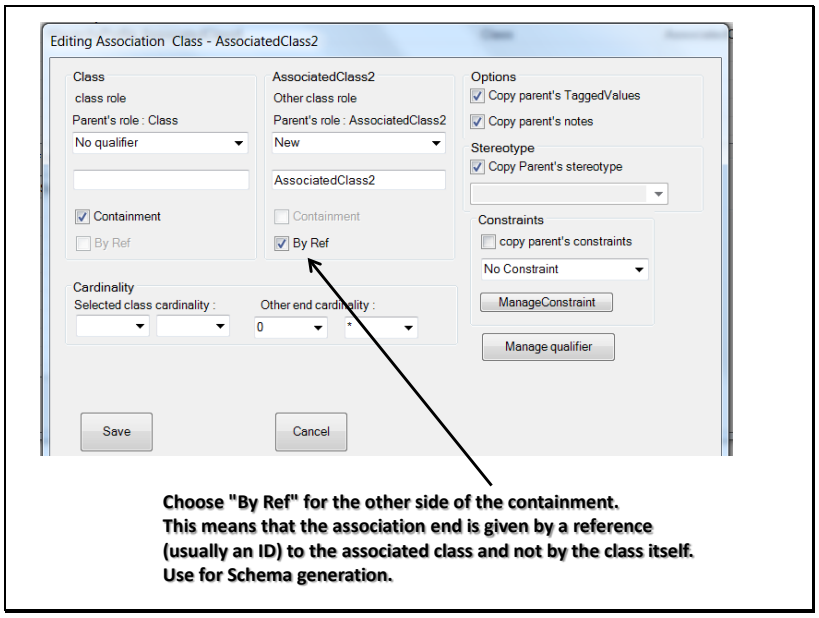
- either to express a more refined semantics of the end role name,
- or to distinguish end role name when there are multiple associations between two classes : in this case qualifiers are mandatory.

Enter qualifiers or select qualifiers in the drop-down menus for each end role name :



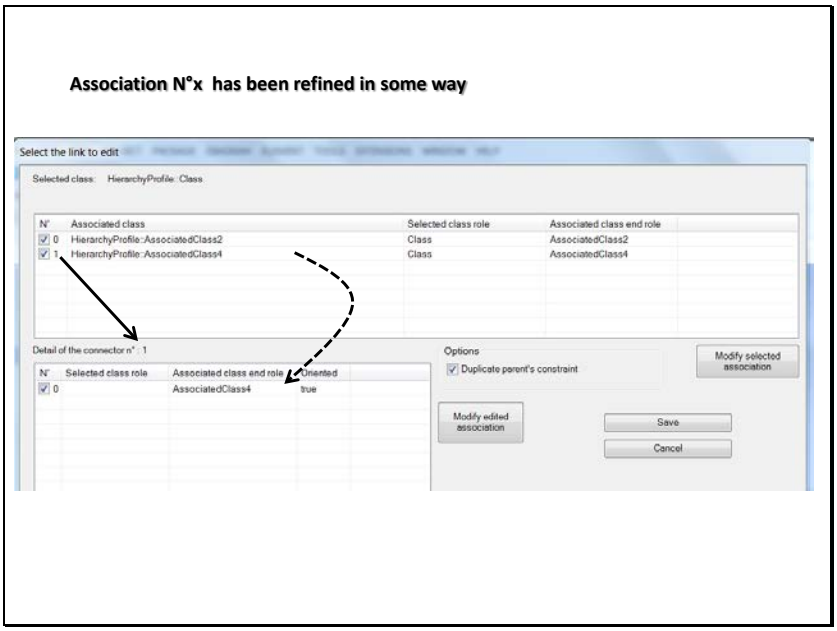
14.10. By Ref association

Sometime, especially with XML Schema, when a profile class is aggregated to several other classes, it might be interesting to use a pointer to it instead of having a duplication of its instances. To enable that, check the "*By Ref*" box :



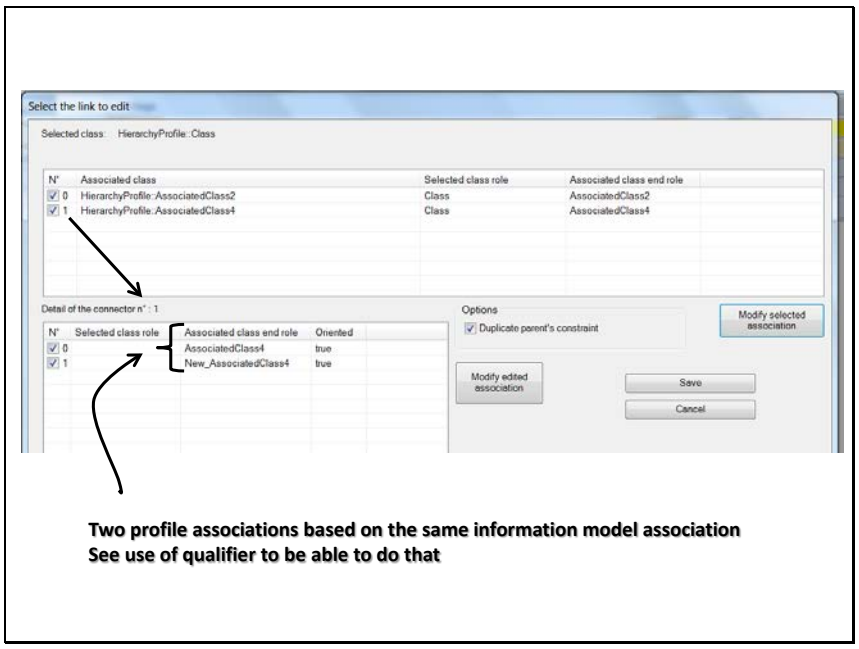
14.11. Save refined association

When you click on the "*Save*" button of the "*Modify Selected Association*" window, the edited association appears in the "*Detail of connector*" window part, where it shows that the association has been refined.

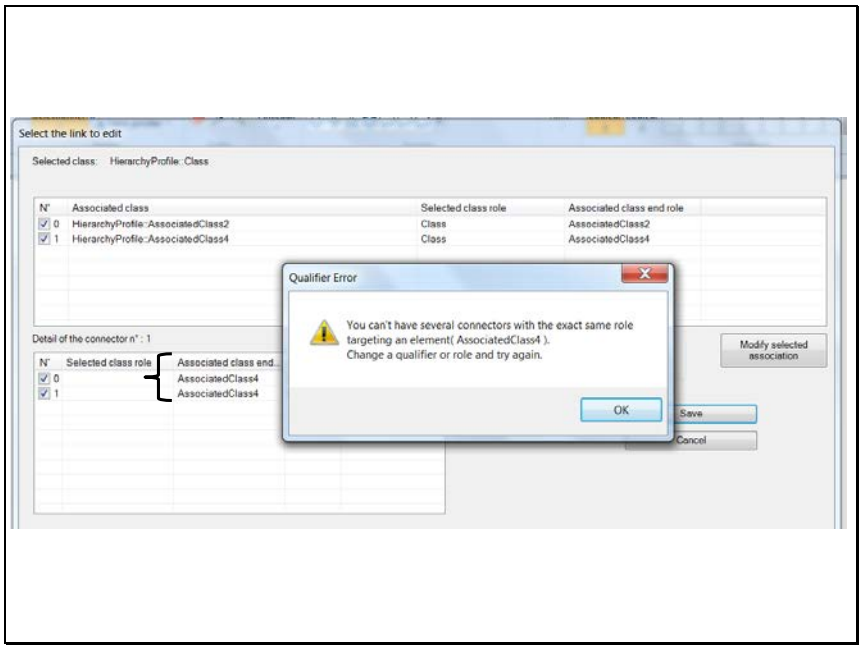


14.12. Make multiple profile associations basedOn the same Information Model association

If you need to have, between two classes, several associations that are based on the same parent association, redo all the steps of the "*Modify Selected Association*" function. When you save this new association, it will appear in the "*Detail of the connector N°*" window part alongside with the other modify associations based on the same one:



But when you do several associations based on the same one, you need to qualify the end role names in order to distinguish the associations. If you do not do that, *CimConteXtor* will remind you to do it when you try to create the associations in the profile by clicking on the "*Save*" button :

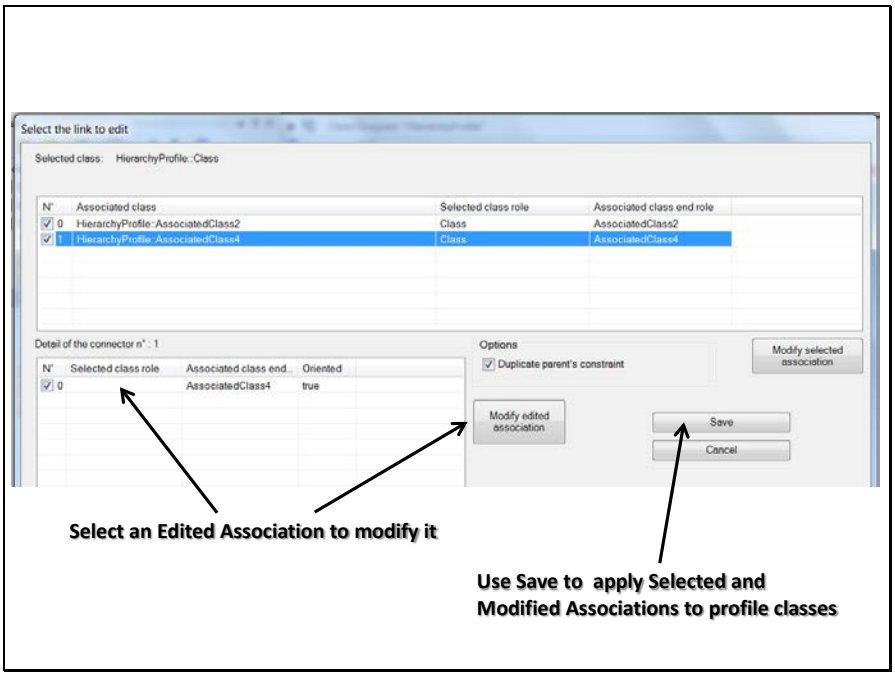


14.13. Edit a modified association

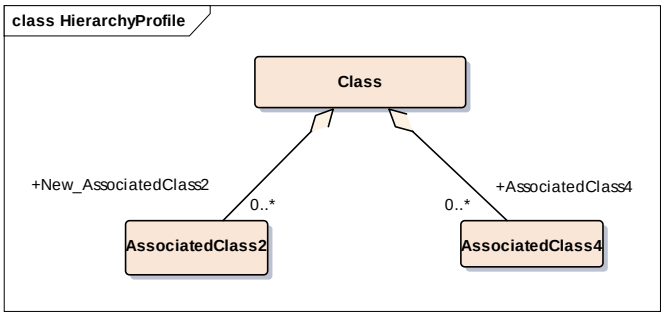
If you want to reedit a modify association, select the modify association in the "*Detail connector*" part and click on the "*Modify Edited Association*", this will display again the "*Editing Association*" window that will let you modify the modified association characteristics.

14.14. Create associations

To create the associations between a class and other ones, click on the "*Save*" button, this will draw associations in the diagram:



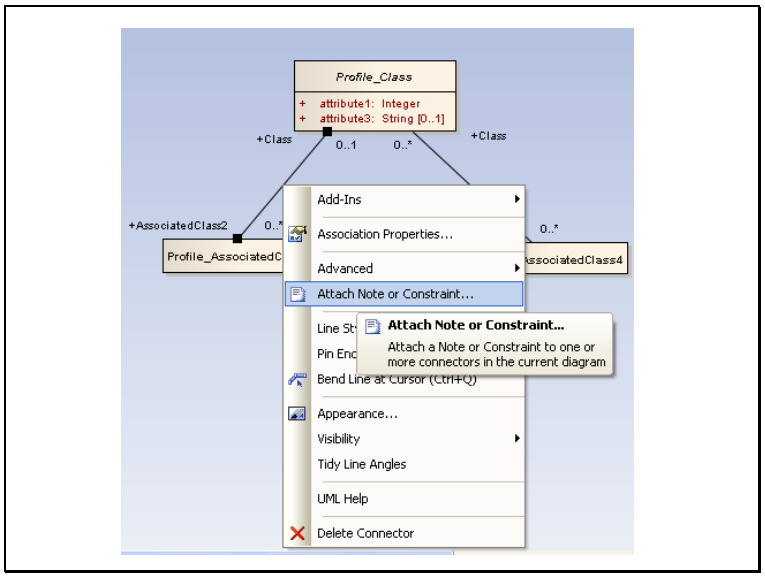
Save will display associations in the diagram:



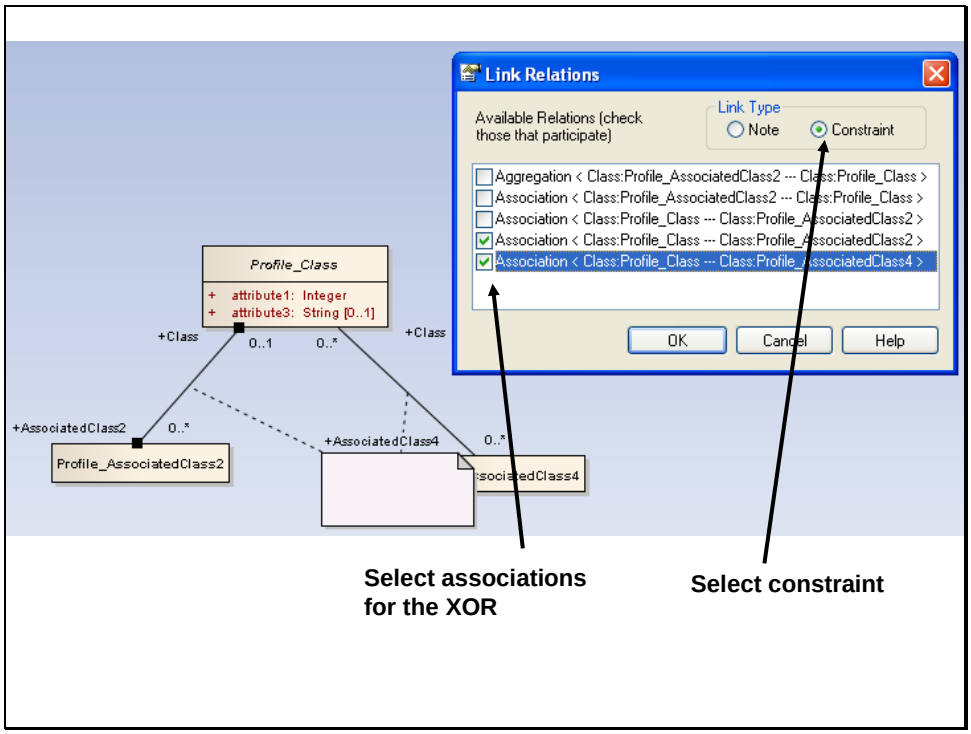
14.15. XOR constraint on Associations

"*XOR*" is a constraint that says that only one of the associations should be used at a time in an instance.

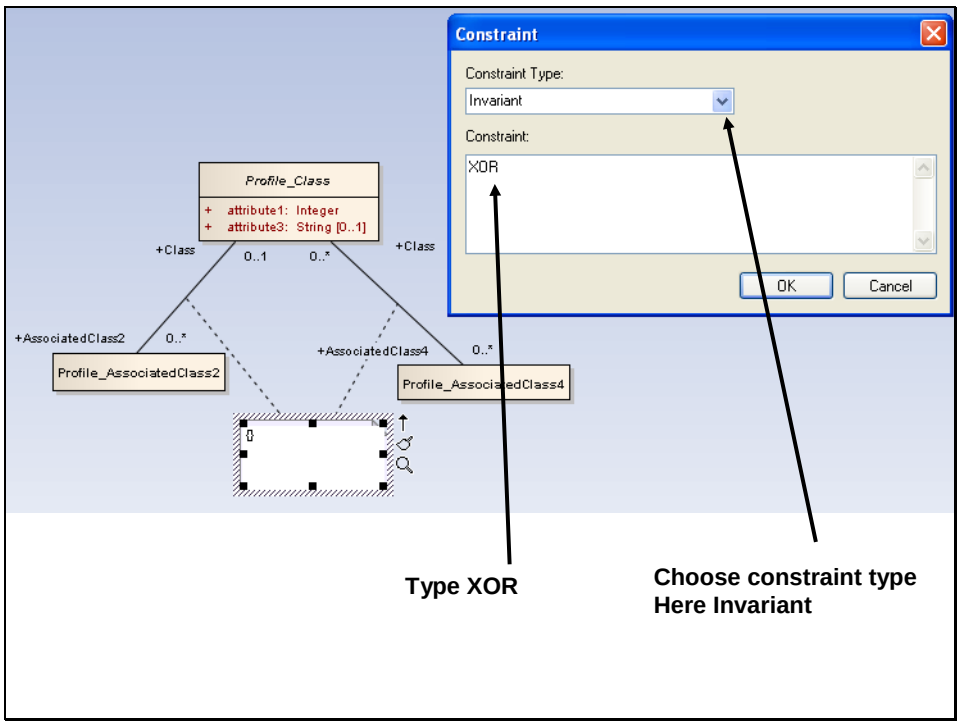
To specify make a "*XOR*" constraint between associations, use the usual EA function. Select an association, then right click on it to have the Menu, and select "*Attach Note or Constraint*":



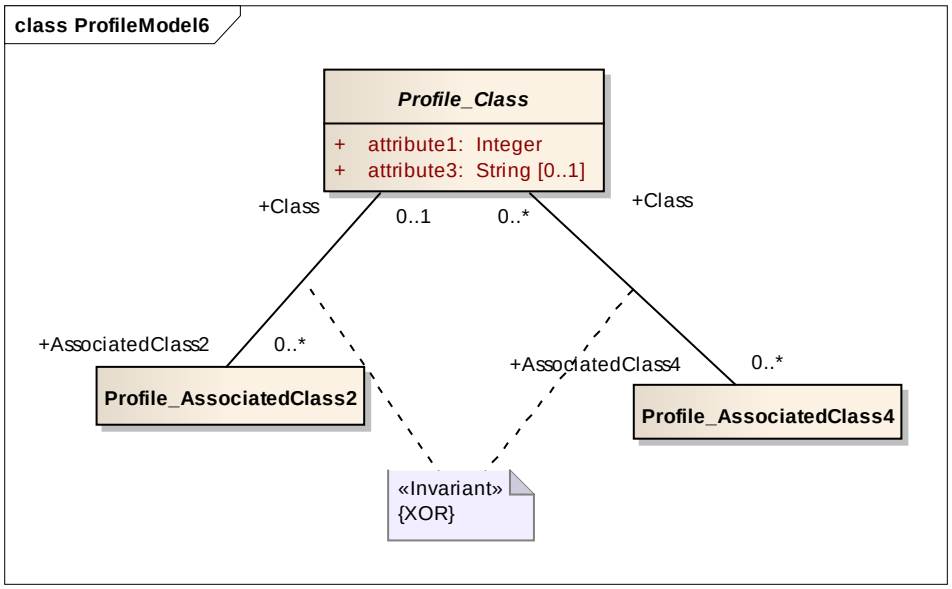
This will display the "*Attach Note or Constraint*" window. Select constraint, and select associations involved in the common XOR constraint:



Click "*OK*", this will display an "*Edit constraint*" window. Select '*Invariant*" for type, and enter "*XOR*" in the constraint field (XOR term is mandatory, enter it exactly like this) :



By clicking on "*OK*" the attach constraint will be displayed in the diagram :



WARNING : the attached XOR constraint note must be recreated each time the "*Edit connectors*" is reused.

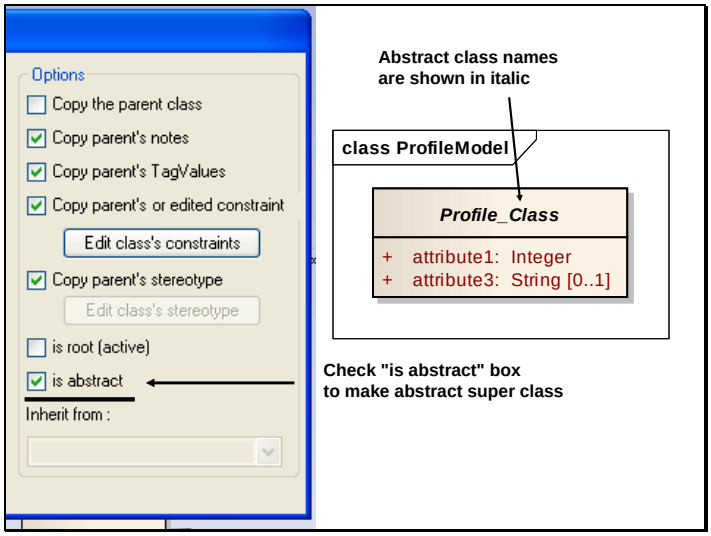
15. Modify profile class associations

Select profile class and use the appropriate "*Edit Connectors*" function.

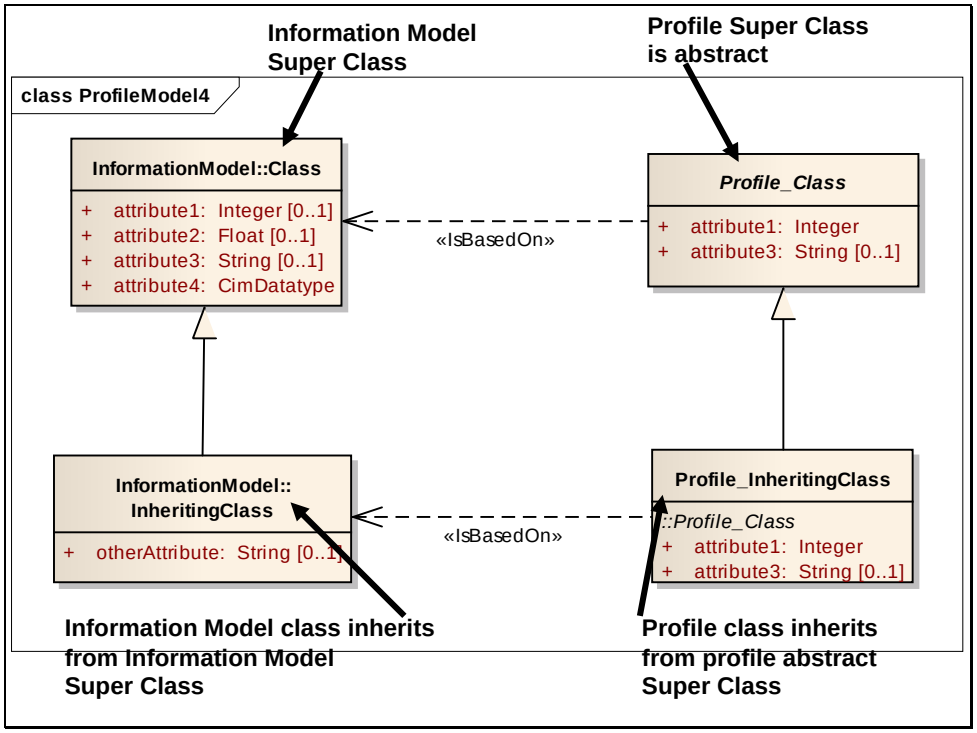
16. Use Inheritance in profile for IsBasedOn classes

16.1. Remember :

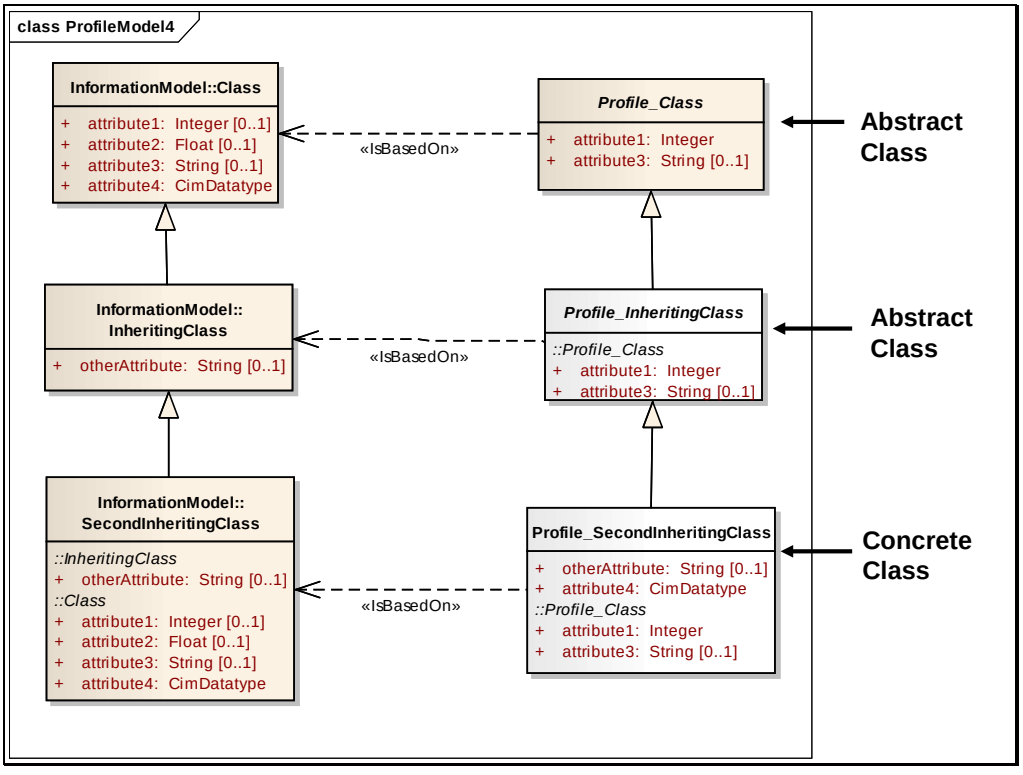
1. *In a profile, inheritance can be used, but the good practice is to make all the profile super classes ABSTRACT.*



2. *In a profile, inheritance can be used between profile classes ONLY if these classes are "IsBasedOn" Information Model classes that have themselves an inheritance relationship :*



3. *In a profile, inheritance could be recursive, in this case all super classes are abstract and only the last class that inherits is concrete*



16.2. CimContextor inheritance possibility

By default, a profile class can only inherits from a profile superclass if their inheritance relationship reflects a direct inheritance relationship in the corresponding information model. But this could be removed, by unchecking the "*EnabledIntermediaryInheritance*" element in the configuration file.

16.3. Good practice:

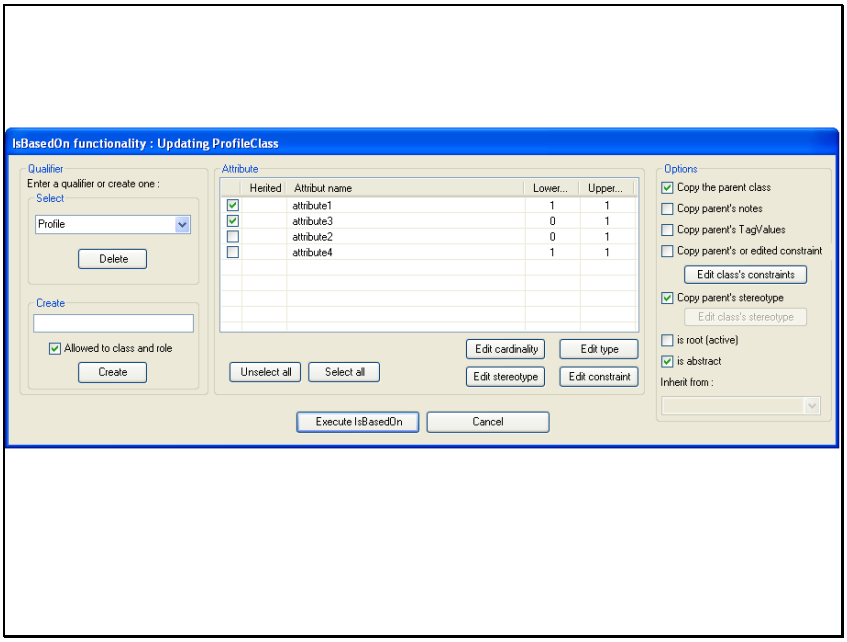
A good practice for doing a profile with super classes is to create first all the profile super classes by using the *CimConteXtor* "*drag and drop function*" as described in section 10 (create "*IsBasedOn classes with CimConteXtor"*).

But do not forget to check the "is abstract" box for the class that is "IsBasedOn". In UML, when the class is abstract, its name appears in Italic in the diagram.

16.4. Create an "*IsBasedOn*" class that inherits from another "*IsBasedOn*" class

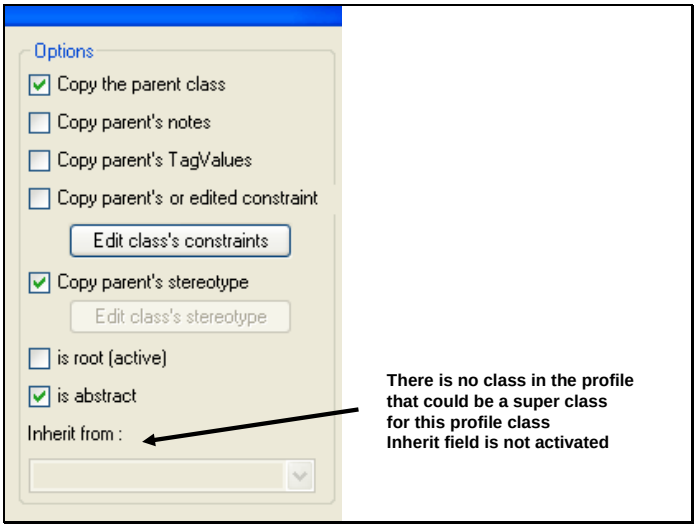
To create the "*IsBasedOn*" class use the drag and drop function, several successive windows will be displayed:

- the usual EA window for making a link, or an instance or a child (select a link),
- then the *CimConteXtor* window that asks if you want to override EA function (say yes),
- and at last the *CimConteXtor* "*Edit IsBasedOn*" window.

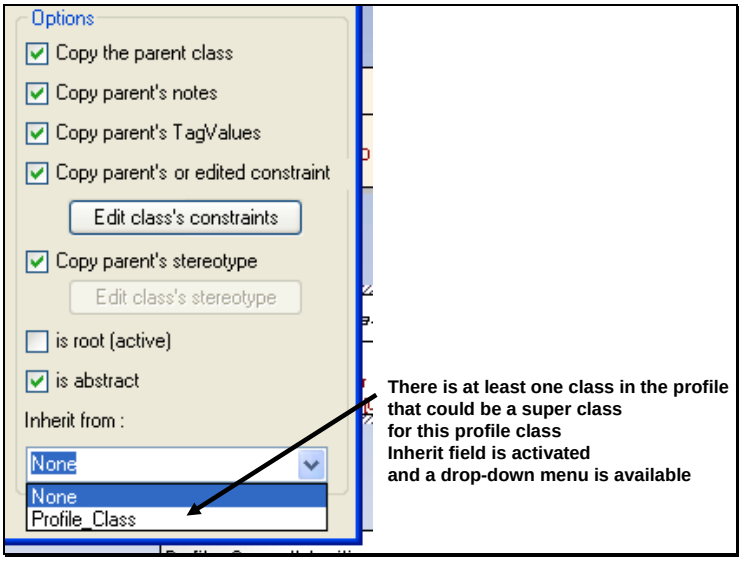


CimConteXtor is looking if there is a possible profile superclass for this "*IsBasedOn*" profile class :

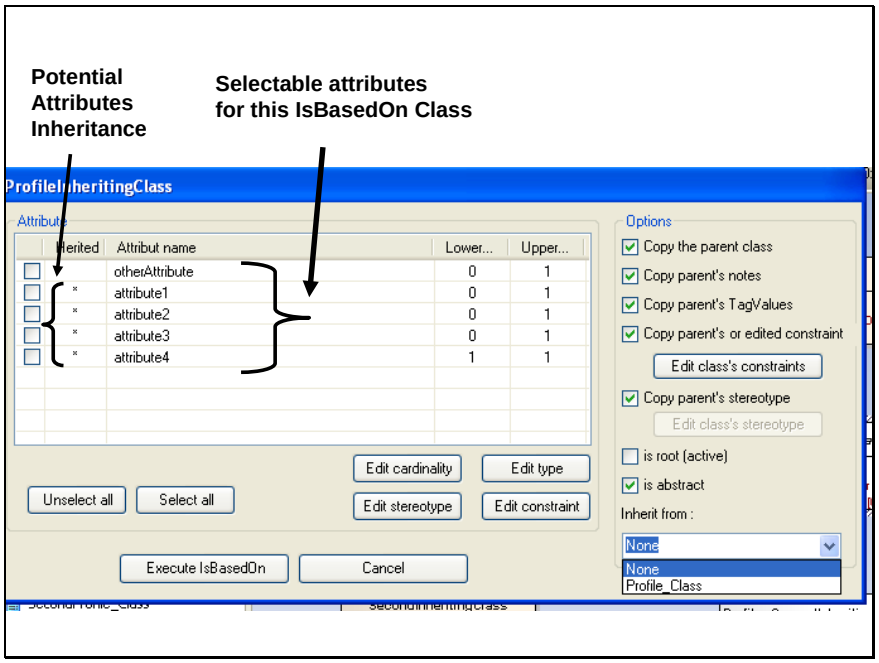
- If there is no possible super class, the "*Inherit from*" field is not shown :



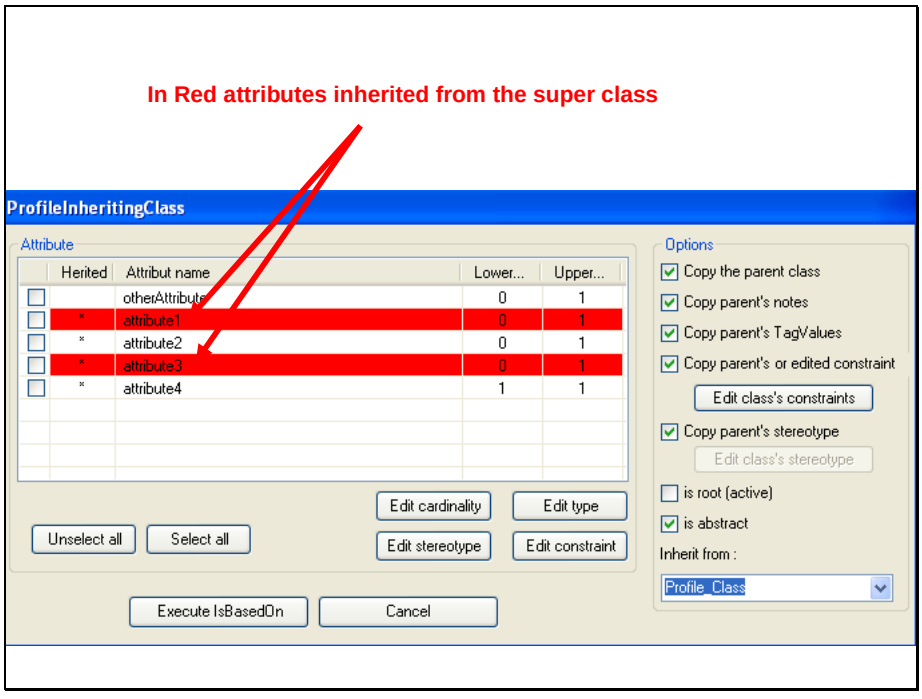
- If there is a possible super class, the "*Inherit from*" field is shown and a drop-down menu shows which classes it could inherit from :



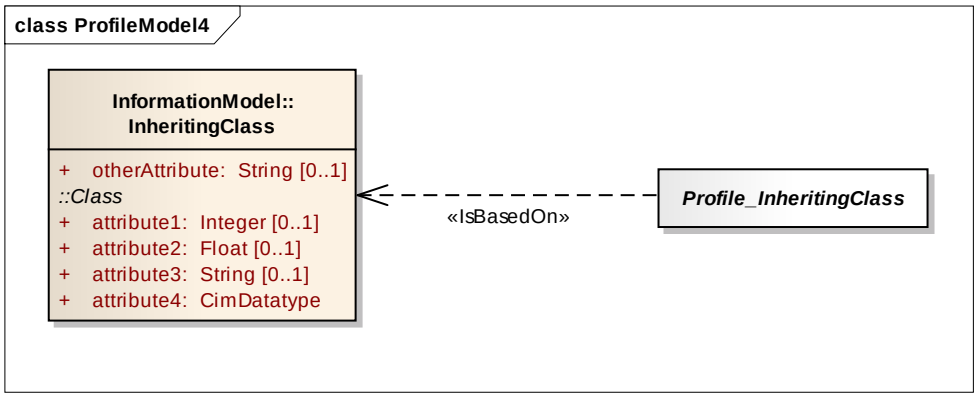
The attribute part of the "*IsBasedOn*" window will show all possible attributes for this "*IsBasedOn*" class and all possible inheritance for attributes.



Here, because you want to create an "*IsBasedOn*" class that inherits, select the profile class that will be the superclass for your "*IsBasedOn*" class. By doing this the attribute part of the "*IsBasedOn*" window will highlight in red all the attributes that are inherited from the superclass and that you cannot uncheck now when you do your attribute selection.



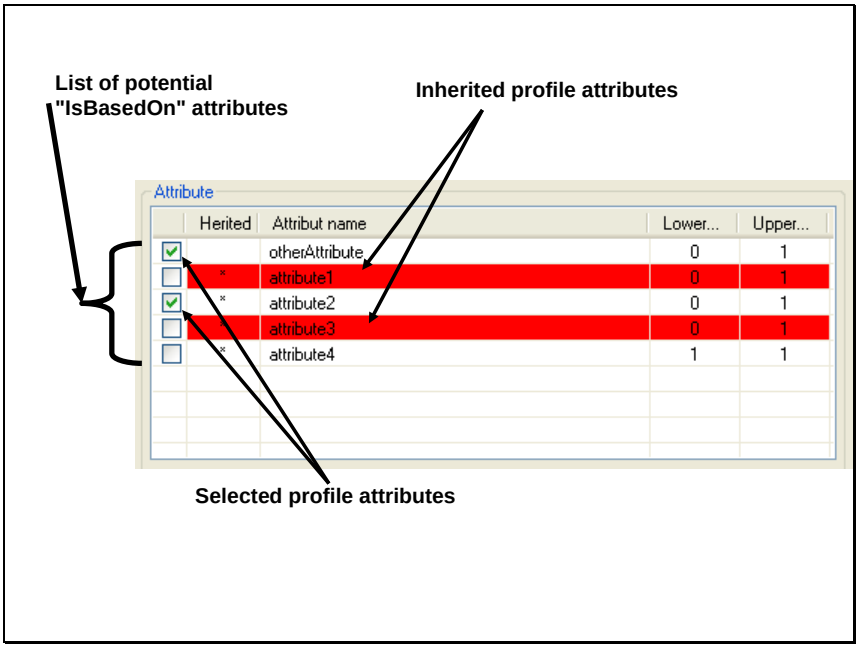
Some explanations in the example shown:



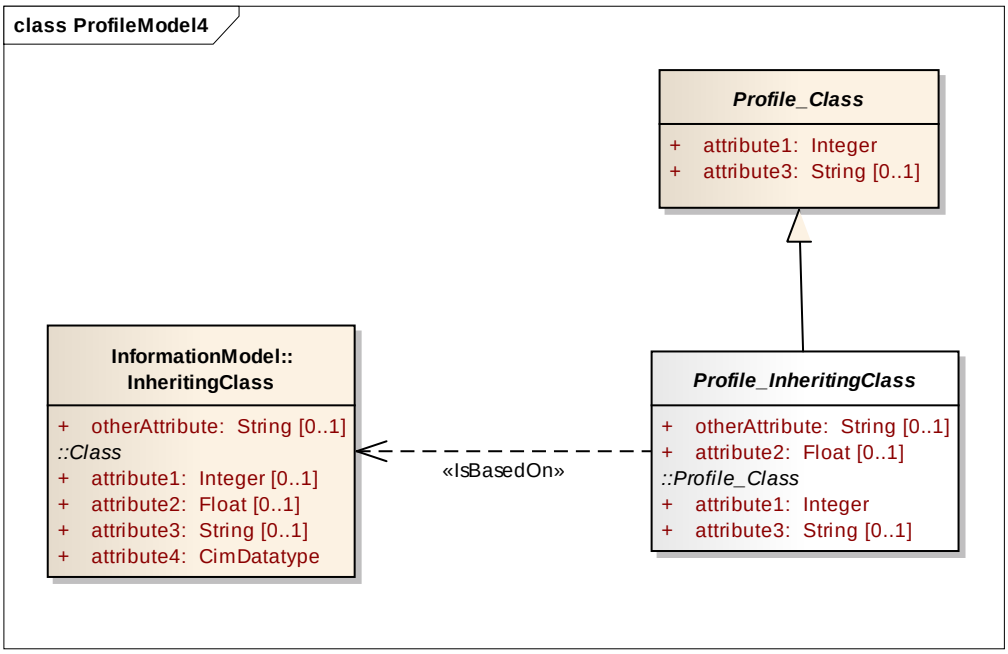
"*Profile_InheritingClass*" is "*IsBasedOn*" the Information Model "*InheritingClass*". So, potentially it could have all the attributes of "*InheritingClass* :

- "*otherAttribute*" (intrinsic "*InheritingClass*" attribute),
- "*attribute1*" ("*InheritingClass*" attribute that was inherited from Information Model "Class"),
- "*attribute2*" (*InheritingClass* attribute that was inherited from Information Model "Class"),
- "*attribute3*" (*InheritingClass* attribute that was inherited from Information Model "Class"),
- "*attribute4*" (*InheritingClass* attribute that was inherited from Information Model "Class"),

If "*Profile_InheritingClass*" inherits from "*Profile_Class*" that has two attributes "*attribute1*" and "*attribute3*", it could be able to select the remaining attributes "*otherAttribute*", "*attribute2*" and "*attribute4*". Suppose we select "*otherAttribute*" and "*attribute2*", then "*Profile_InheritingClass*" will have four attributes : two coming through inheritance and two selected from the "*IsbasedOn*" :



Then when the "*Execute IsBasedOn*" will be processed, the EA diagram will appear like this:



17. Use Inheritance in profile for Associations

17.1. Reminder

To use inheritance for association editing in a profile, all involved profile classes, concrete and super classes must have been created first in the profile: only associations between profile classes could be edited. The rules expressed in the above section ("*Use Inheritance in profile for IsBasedOn classes*") are a prerequisite:

- In a profile, inheritance can be used, but in this case **all super classes MUST be ABSTRACT**,
- In a profile, inheritance can be used between profile classes ONLY if these classes are "*IsBasedOn*" Information Model classes that have themselves an inheritance relationship,
- In a profile, inheritance could be recursive, in this case all super classes are abstract and only the last class that inherits is concrete.

17.2. Create "Associations" for an "*IsBasedOn*" class that inherits from another "*IsBasedOn*" class

The use of inheritance between two profile classes implies good understanding of what it means for profile association's creation, as explained in annex D "*Profiles rules for inheritance*" and especially in its part "*Profile level association inheritance rules*". At a profile class level we have two kind of associations:

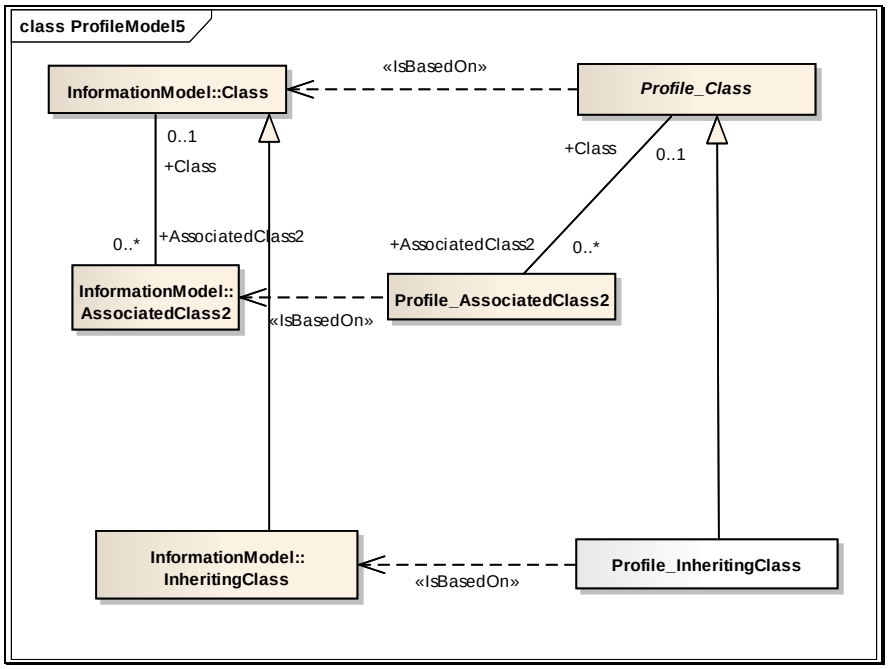
- the associations that are inherited from an abstract profile class,
- the associations that are "*IsBasedOn*".

Let's take an example, where we have:

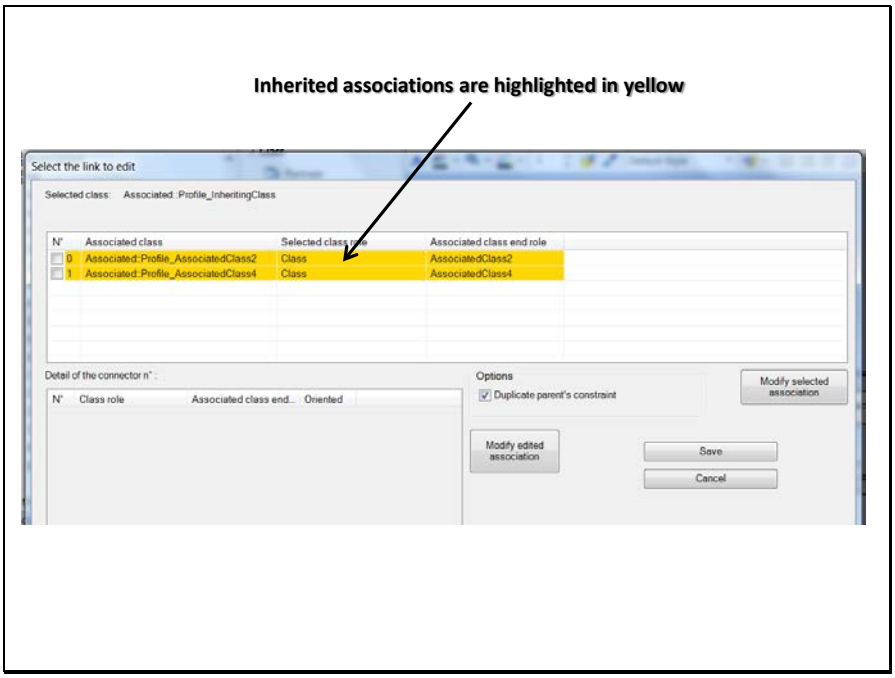
- An abstract profile class named "*Profile_Class*" that have an association with another profile class named "*Profile_AssociatedClass2*"
- A profile class named "*Profile_InheritingClass*" that inherits from "*Profile_Class*".

"Profile_InheritingClass"

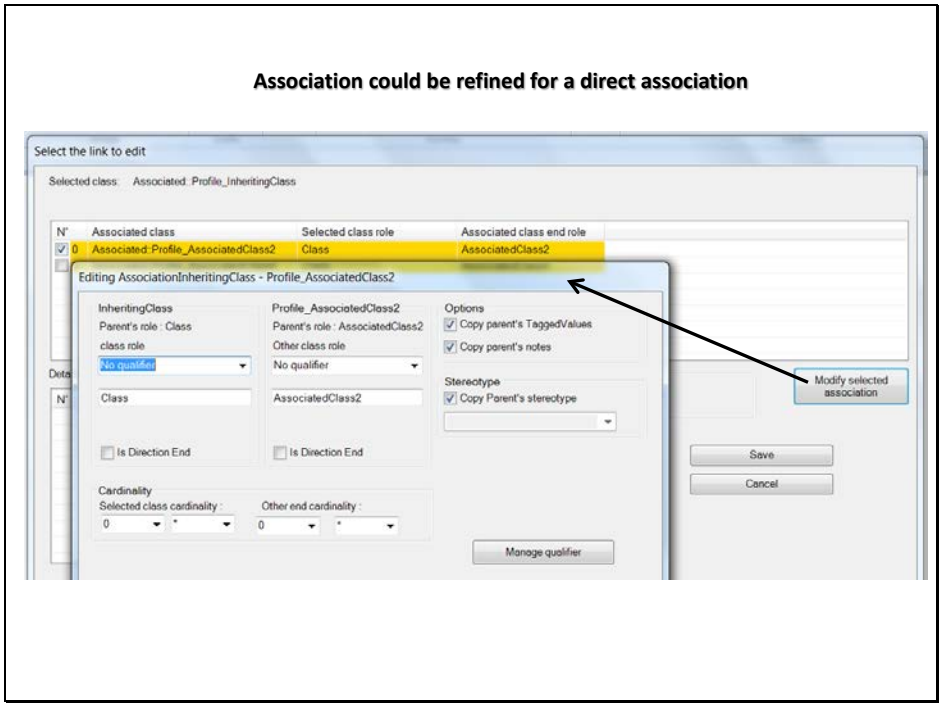
- inherits the association with "*Profile_AssociatedClass2*", this association has two end role names "*Class*" and "*AssociatedClass2*",
- but potentially could also have direct associations with "*Profile_AssociatedClass2*". But in this case, the end role names MUST be qualified.



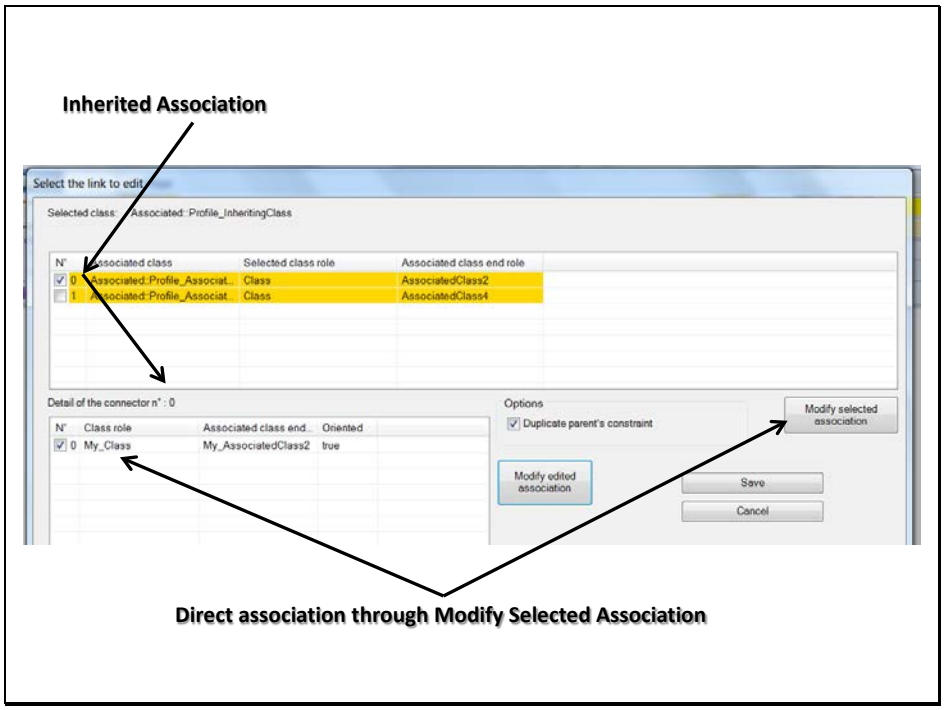
Selecting "*Profile_InheritingClass*" and selecting "*Edit connectors*" will display the "*Edit connectors*" window. But this time, a new information will be shown: inherited associations that are highlighted in yellow :



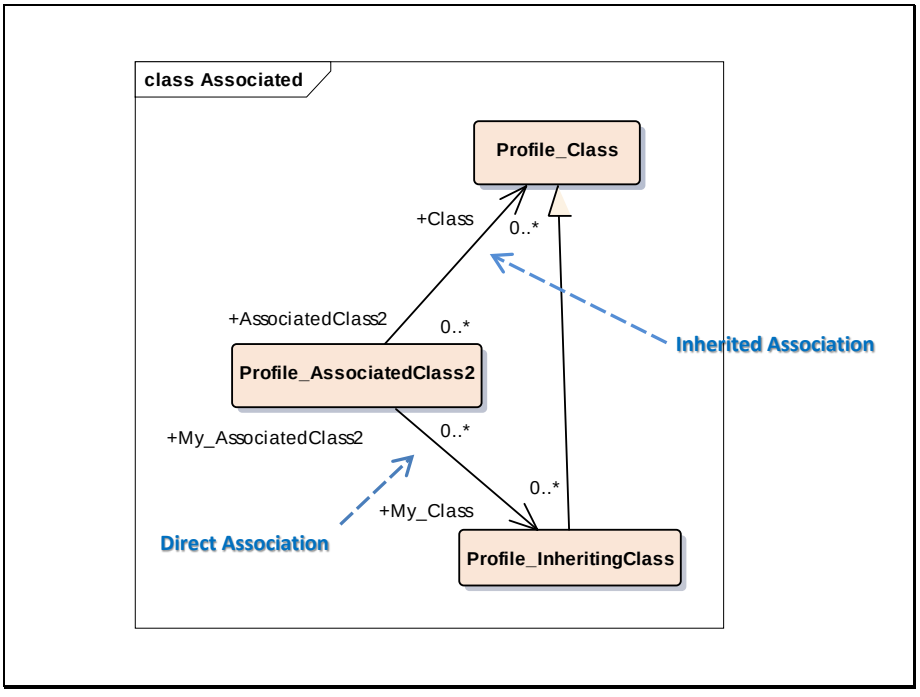
If we want to have also a direct association between "*Profile_InheritingClass*" and "*Profile_AssociatedClass2*", we must use the "*Modify Selected Association*" feature because we need to qualify end role names.



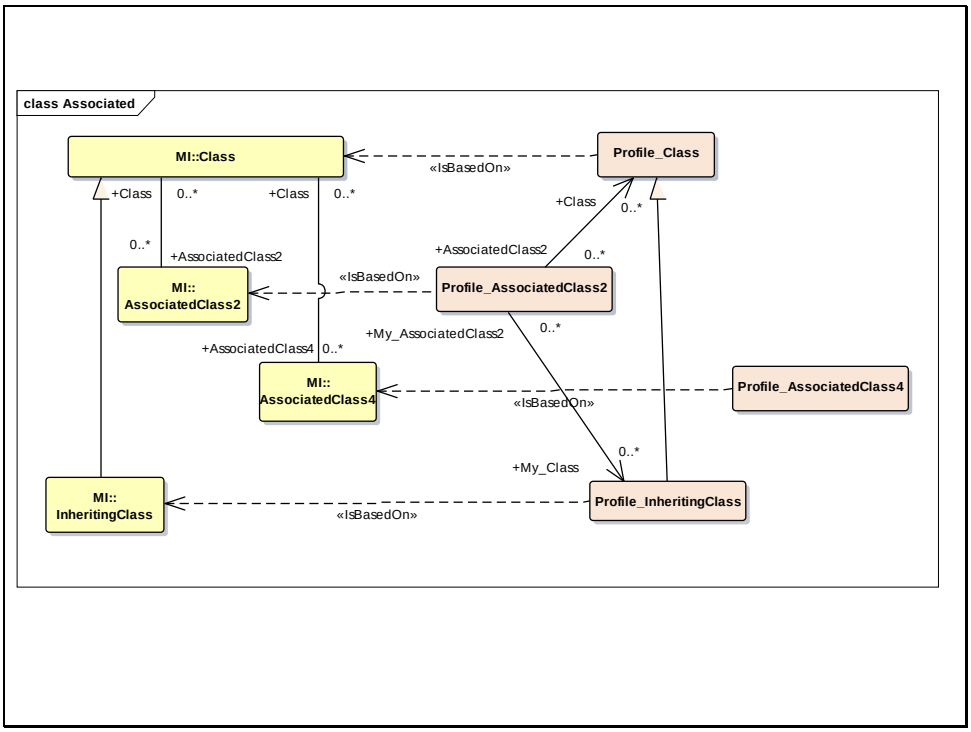
Then save will display the modified association in the "*Detail Of Connector*" of the "*Edit connectors*" window:



Click on "*Save*" will show association in the diagram :

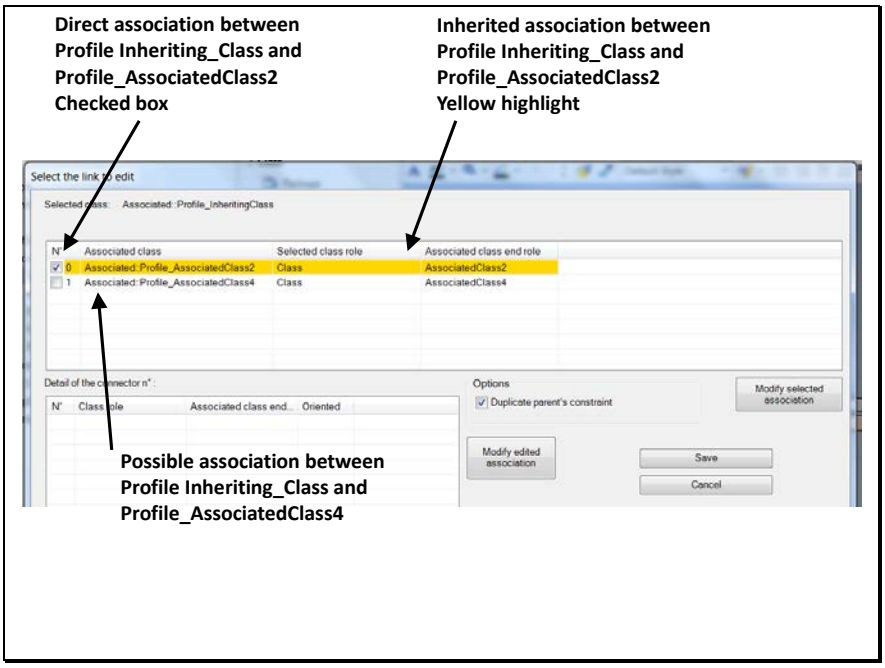


Let's continue with another example:

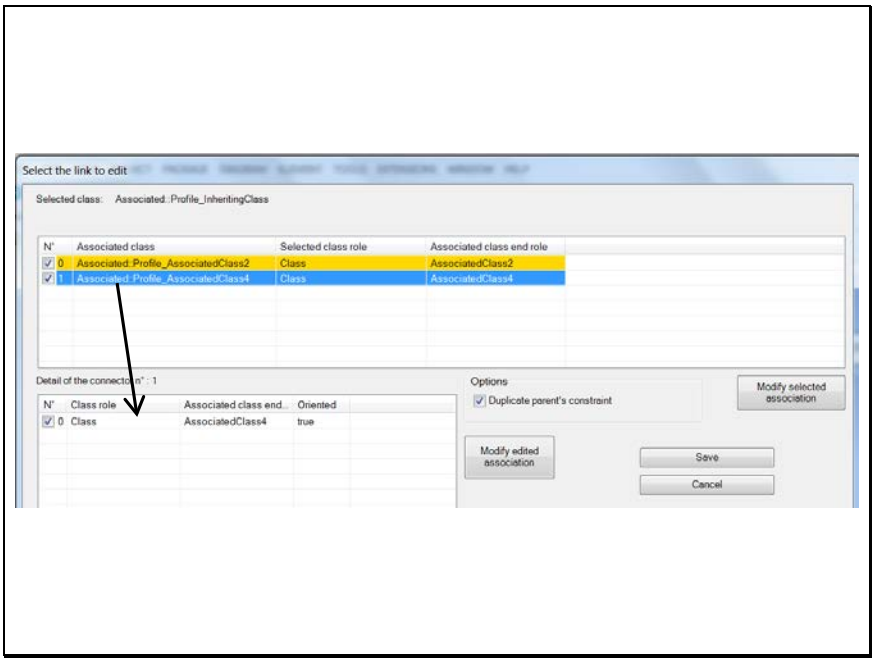


Here, there is a new profile class named "*Profile_AssociatedClass4*". There could be an association with "*Profile_InheritingClass*". How it will be shown?

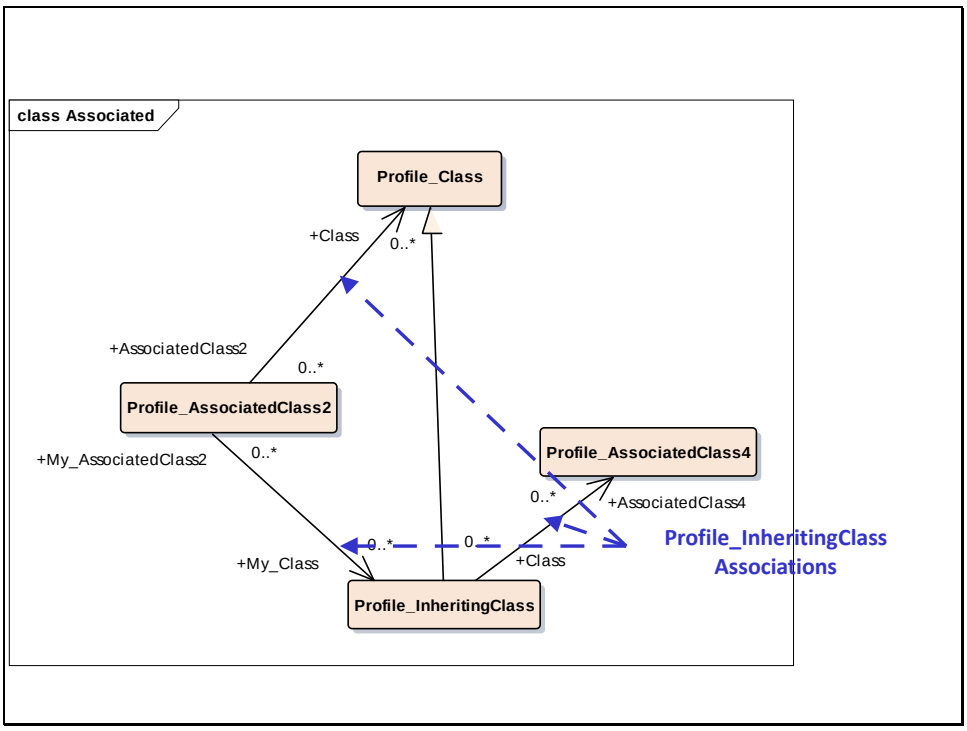
The association between "*Profile_InheritingClass*" and "*Profile_AssociatedClass2*" is inherited so it is highlighted in yellow and there is also a potential direct "*IsBasedOn*" association between "*Profile_InheritingClass*" and "*Profile_AssociatedClass4*"



This association could be duplicated directly or it could be modified if needed:

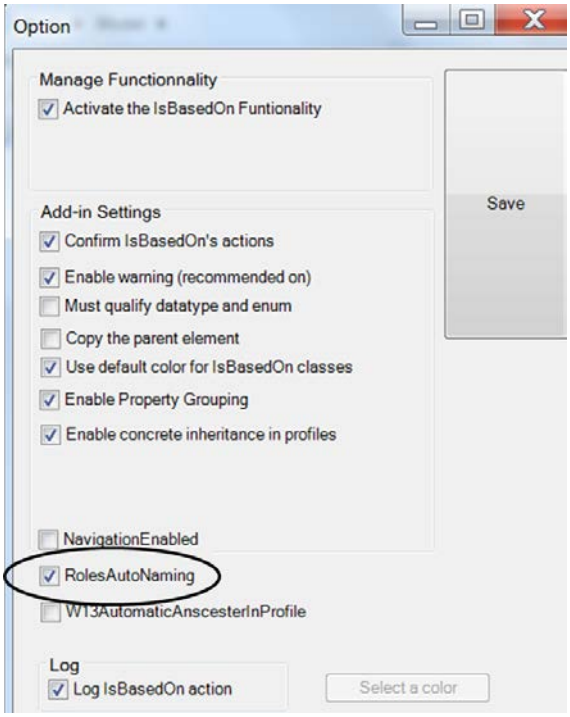


Click on "*Save*" will show associations in the diagram:



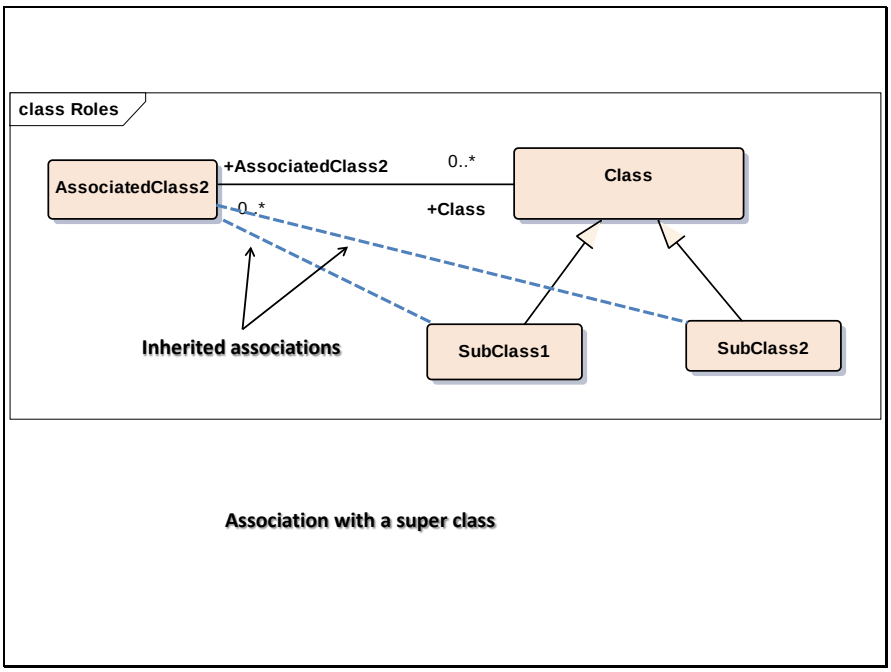
17.3. Roles AutoNaming

This option is used with the "*EditHierarchicalConnectors*" Menu. To allow this option, the "*RolesAutoNaming*" box of the CimConteXtor Option Menu must be checked.



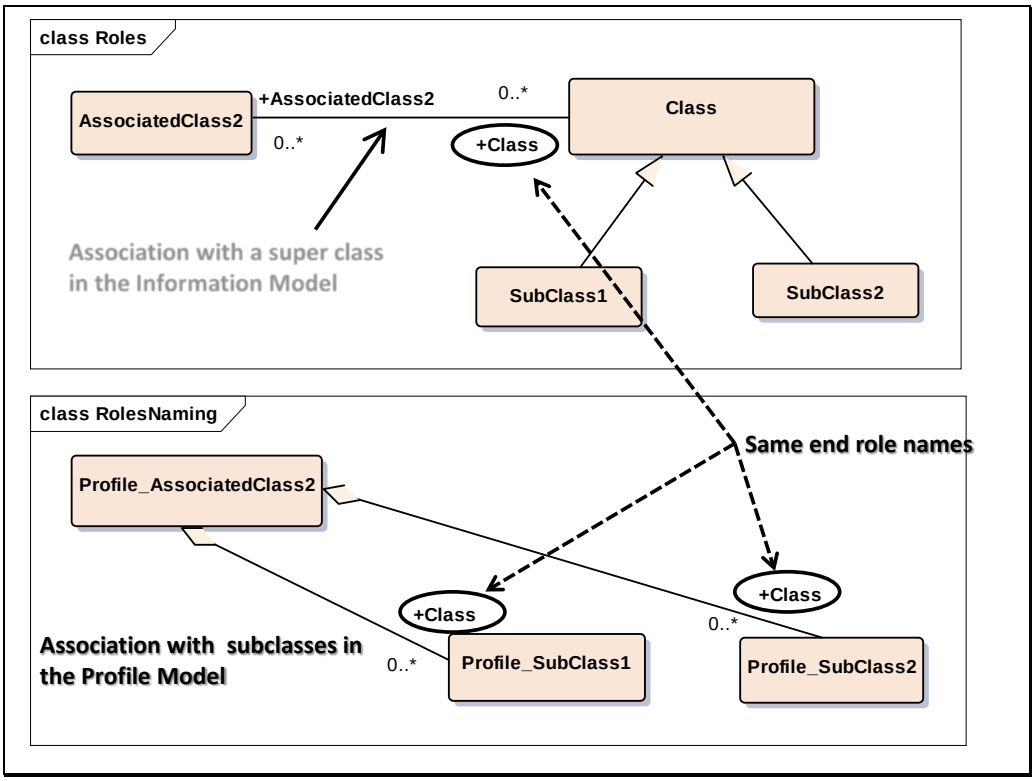
When this option is useful? Let's take an example.

If we have a class (*AssociatedClass2*) that has an association with another class (*Class*) that acts as a superclass for classes (*SubClass 1 and 2*), this means that because of the inheritance, *AssociatedClass2* is also associated with *SubClass1* and *SubClass2*.



When doing a profile, in a hierarchy mode, the associations selected could be between the associated class and the subclasses as in the following diagram. Note

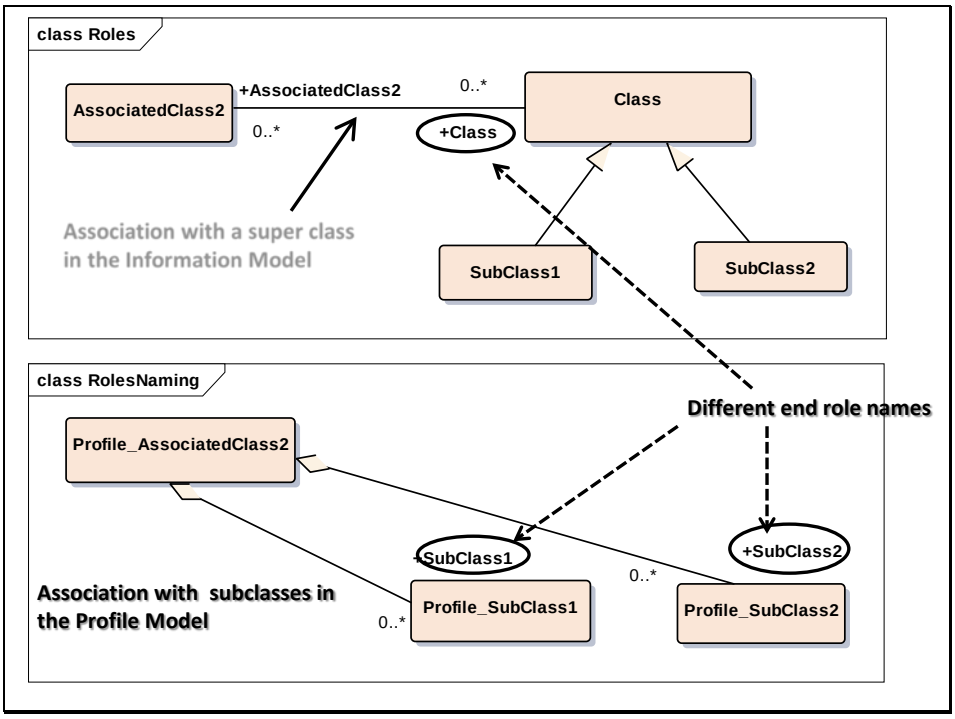
that in this case, in the profile, the end role name on the side of the subclasses will be "*class*" as specified in the information model.



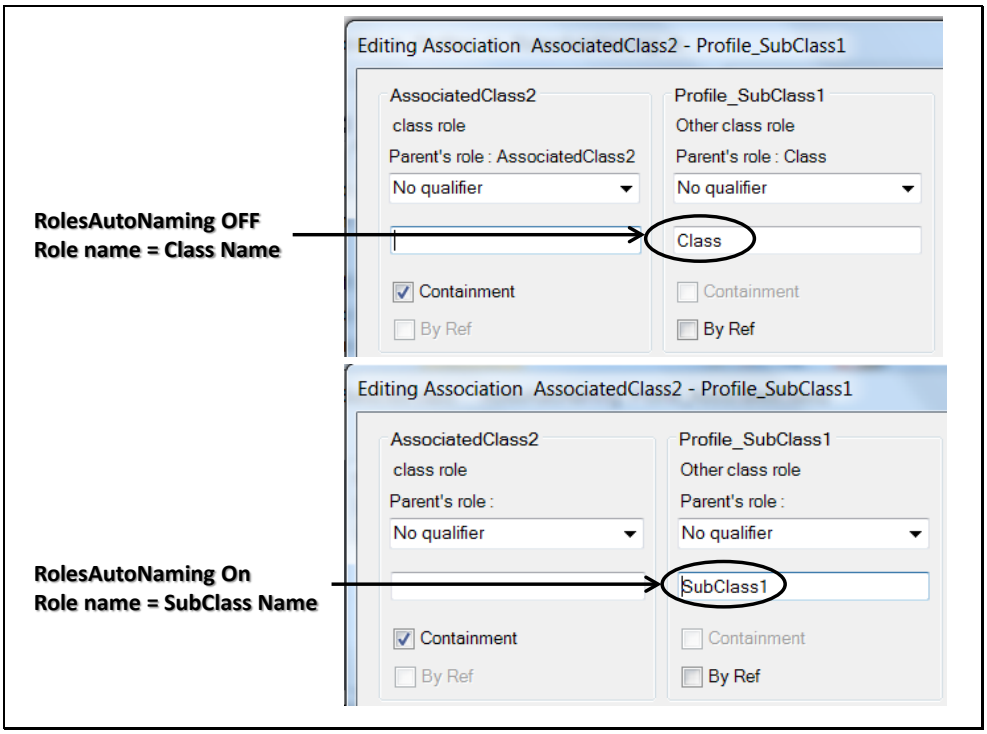
There are two drawbacks in this design:

1. When using a superclass, it is to mutualize properties. But in fact, do we want to say that *AssociatedClass* has a "class" property (end role name) with *Class* and *SubClass*, or do we want to say that *AssociatedClass* has a "class" property with *Class* and a "subclass" property with *SubClass*? So depending on the interpretation, we could take one or the other.
2. When targeting a hierarchy profile, where the intended mapping is XSD, having same end role name with two different classes (SubClass1 and SubClass2 will lead to a wrong XSD. That is why in IEC 62321-100, which specifies how the mapping is done, there is a changing name rule. But making the change at mapping level results in having different names in UML and XSD.

So the RolesAutoNaming provide a way, if needed, to be able to change end role names at UML profile level.

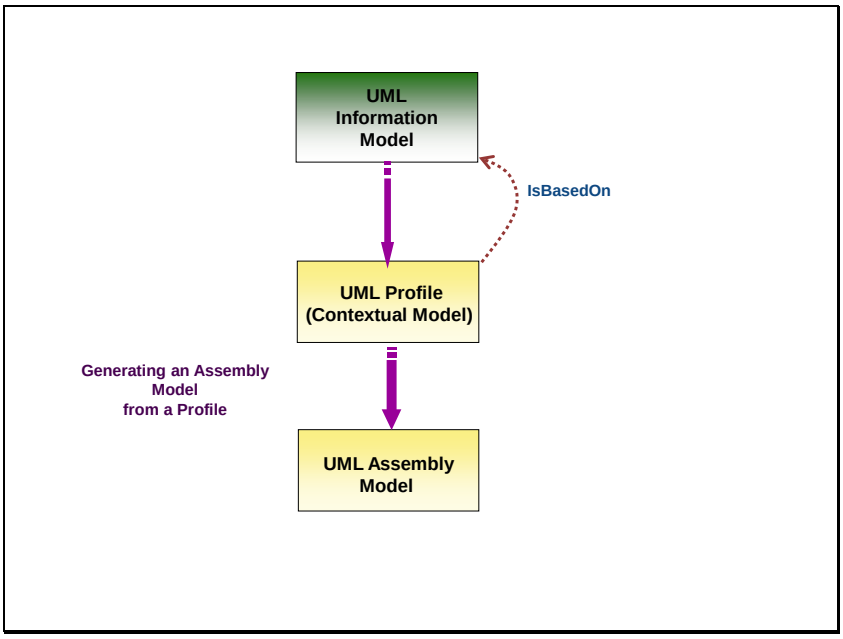


When using "*RolesAutoNaming*", changed name will appear in the EditingAssociation windows:



18. Assembly Model

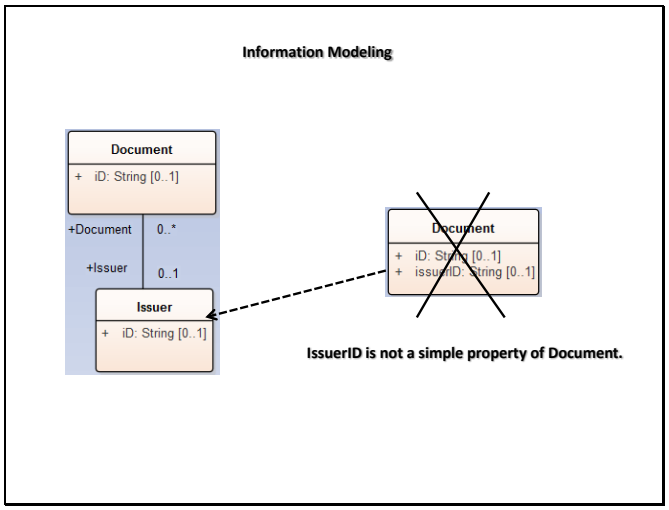
18.1. Overview



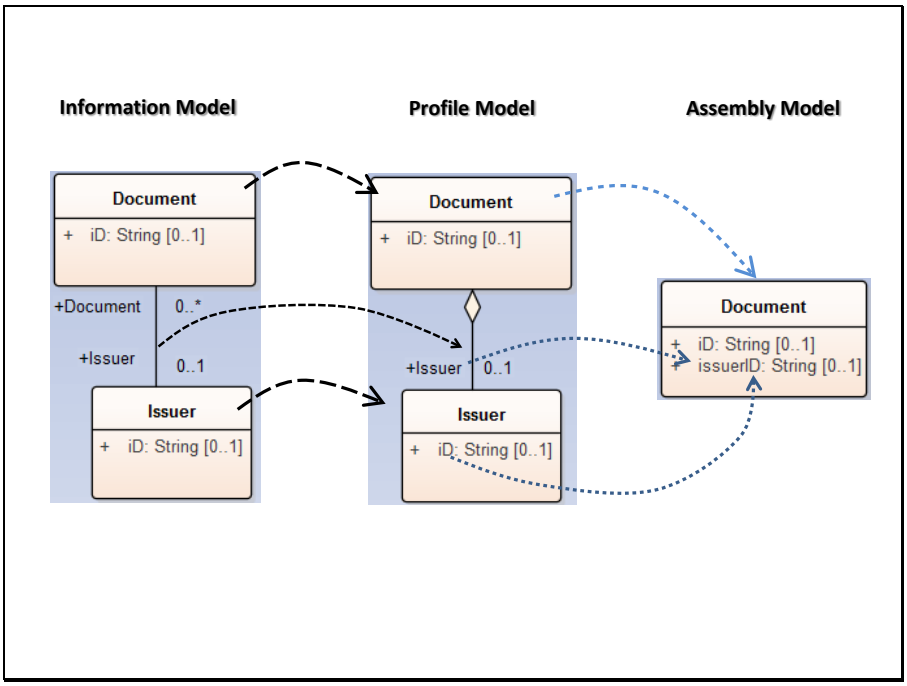
An Assembly model has several purposes:

- To add to the profile model a header,
- To realize "*Property Grouping*".

Here we just define how to generate an Assembly model that is used to specify the "*Property Grouping*". In the design of information model, the rule is to have in a given class, only the properties that belong to this class. Example:



But for different reasons, at profile level, requirements have been made to regroup properties so that a class could have, as simple properties, properties that are coming from other classes. Here is an example:

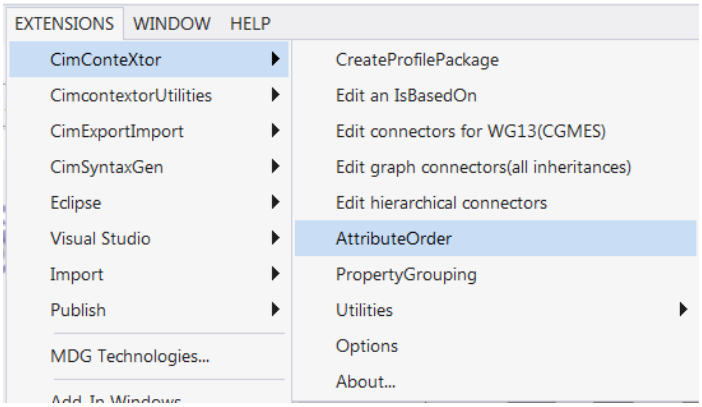


In this example, at the information model level we have two classes (Document and Issuer). At hierarchical profile level, Document is the upper hierarchy class and Issuer a leaf class. The assembly level is going to group properties in one class. There are rules for enabling property grouping between classes:

- Properties of a leaf class could be aggregated in an upper class only if the association multiplicity is 0..1 or 1 (no property grouping if multiplicity 0 or 1..n.
- The name of the aggregated property is "*Issuer.iD*", concatenation of:
  - Name of the end role name (here Issuer),
  - A dot,
  - Name of the attribute (here iD).
- Property grouping could be iterating.

18.2. Attribute Order for Property Grouping

Before doing the property grouping itself, one need to define in which order the items (attribute and associated class attribute) part of the grouping will be grouped. This is done by using "AttributeOrder" of the CimConteXtor Menu:



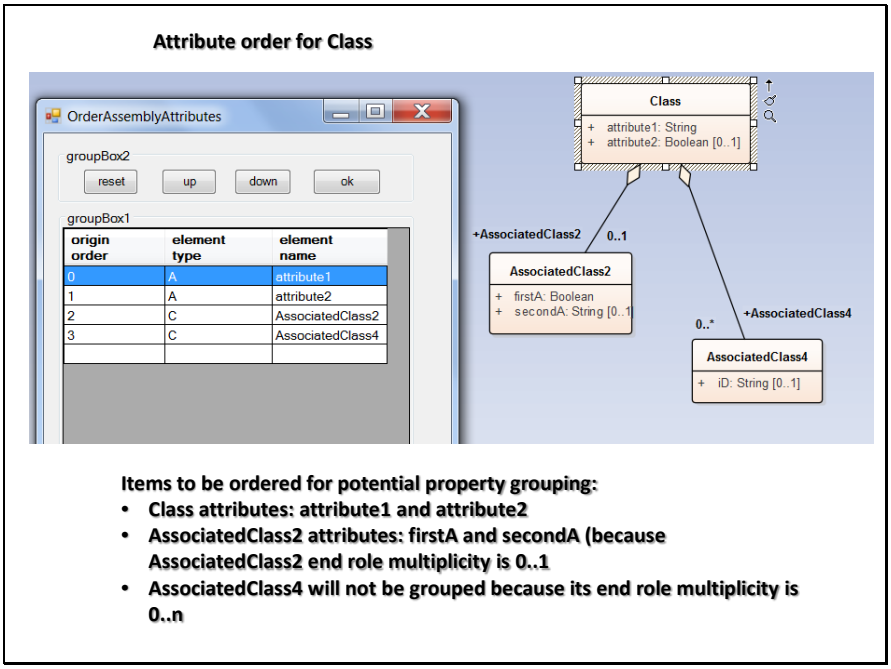
The attribute order feature is to be applied to each class of the profile. The result will generate a tagged value called "*ESMPRG*" for each attribute and association.

To order the attributes and the associations in the UML profile class diagram, the following steps are to be made:

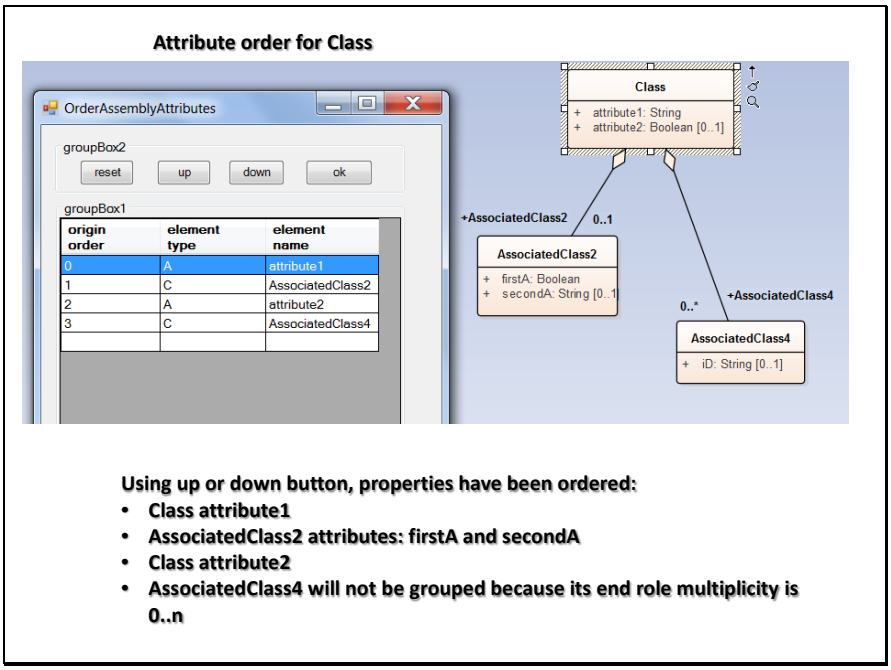
1. Select one of the class to be ordered.

Note: it is highly recommended to start from the "leaf" classes up to the root class.

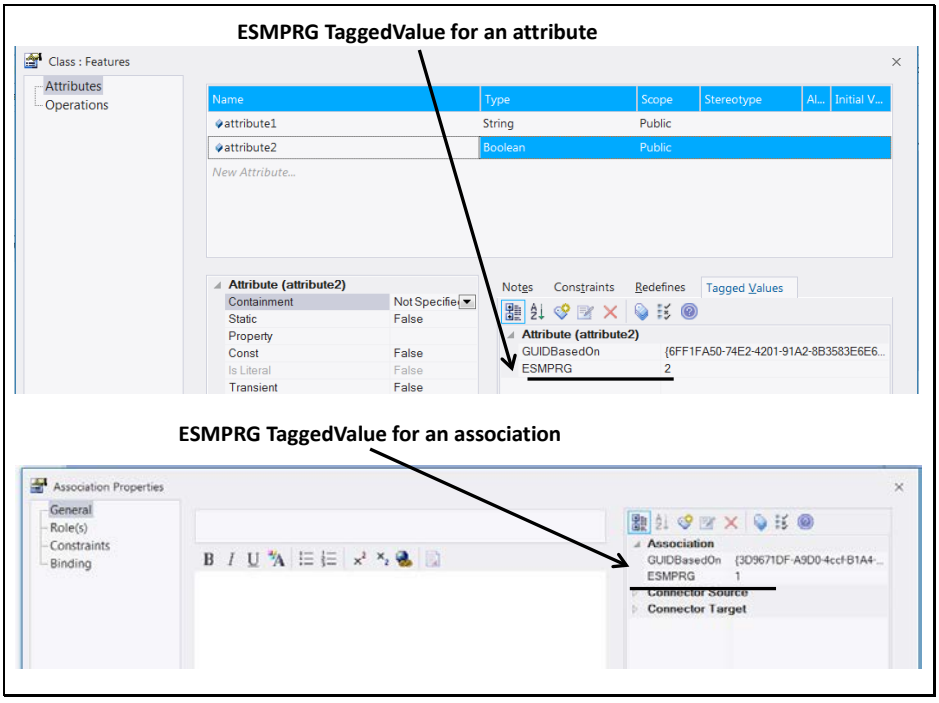
2. Right click, select "Extension", then "CimContextor" and "AttributeOrder".
3. A dialog box will appear:



4. Use the "up" or "down" button to order each attribute of the class (element type "A") or the association end role name for the associated classes (element type "C"). It is recommended that the associations of multiplicity 0..* be put at the end of the list of attributes, because they will not be grouped.
5. When the order is fine, click on "ok" to end the dialog and save the configuration.



With the change made, the attributes in a class will respect the ESMPRG tagged value.



In case the attributes have not been ordered, there will be an error message at the following step, i.e. PropertyGrouping.

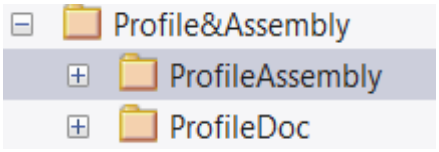
18.3. Property Grouping

The "*PropertyGrouping*" option is to be used to generate from a property model the associated assembly model.

All the classes associated with a multiplicity of 0..1 or 1..1 to a class are inserted within this latter class, and the order is provided by the "*AttributeOrder*" operation.

To generate the assembly model, the following steps are to be made:

1. Define an Assembly model package in the browser. This package must be inside a package where the Profile (here ProfileDoc) Model is defined:



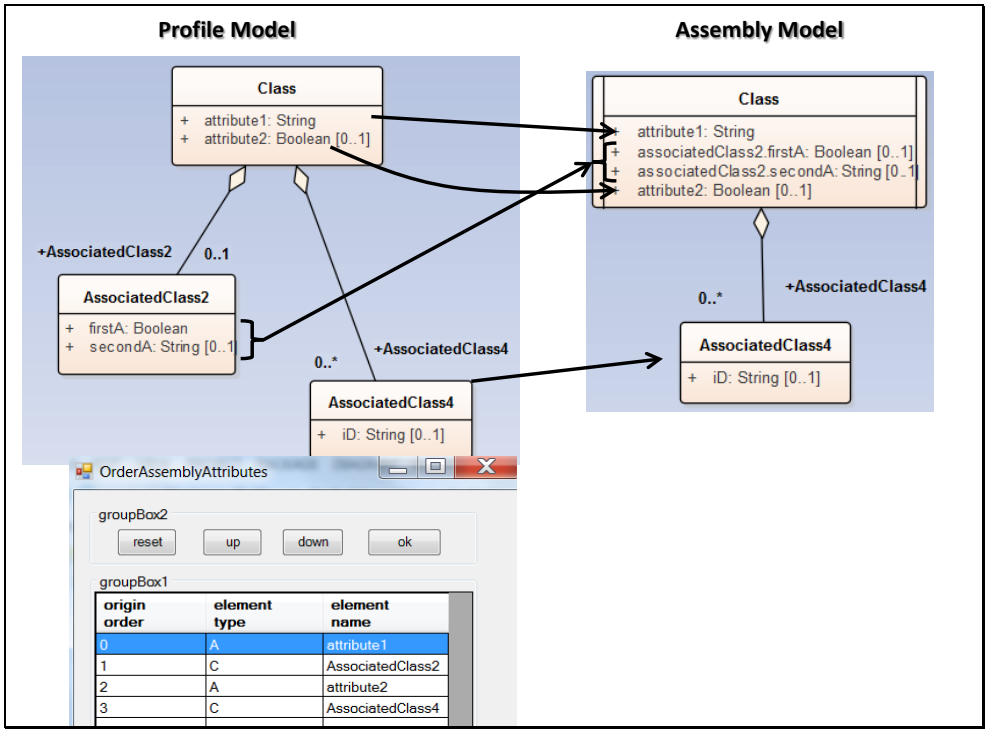
2. In the package browser, select the assembly package to be generated.

   Note: if the assembly package is not empty, all its content will be deleted.

3. Right click, select “Extension”, then “CimContextor” and “PropertyGrouping”.

   The generation process starts, and a confirmation of the package to be grouped will be asked; click “Ok” when requested.

At the end of the process, the assembly diagram is displayed.



It is recommended to “arrange” the display (position of the classes, position of the association, etc.) to enhance the visual aspect. It is also recommended to check that all the attributes are in the appropriate order; otherwise the steps described are to be resumed.

Note: the process will always put the assembly package before the contextual package. You will have to change this order before generating documentation.

18.4. Attribute order for syntax generation

At this stage an "*OrderAttribute*" must be now done on the Assembly Model Classes in order to define the XSD mapping.

19. Utilities Menu

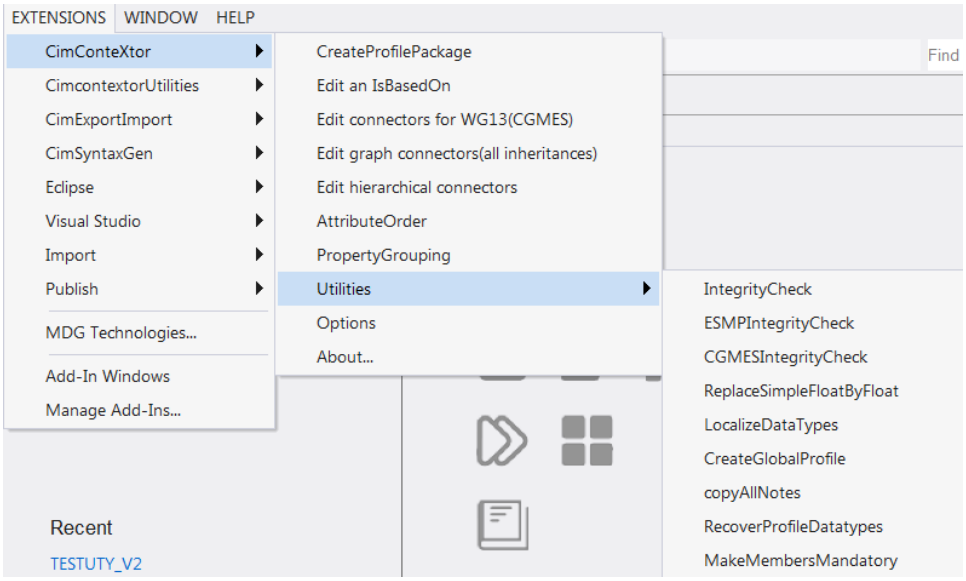
"*Utilities*" Menu offers several features to check if profiles are correct according to profiling rules and to help profile building.

19.1. Check profiling

Check features of the "*Utilities*" menu are used to verify if a UML model follows rules for UML modelling. Currently, depending on the modelling context, there are three different checks:

- General integrity check,
- Integrity check for European Style Market Profile,
- Integrity check for Entso-E CGMEs profiles.

Result of Integrity check can be found in the log file.



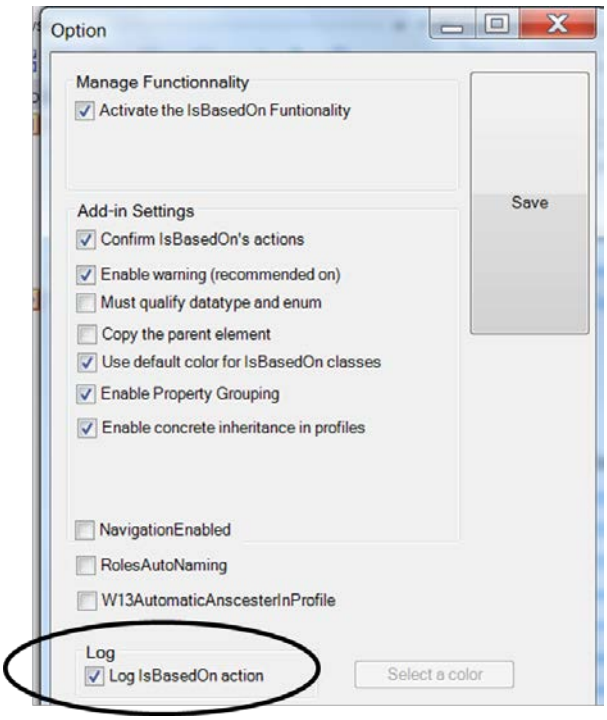
19.2. IntegrityCheck

IntegrityCheck checks the possible errors in editing a new profile. It checks the basic errors, without specific rules used by different profiling styles (CGMES, ESMP…).

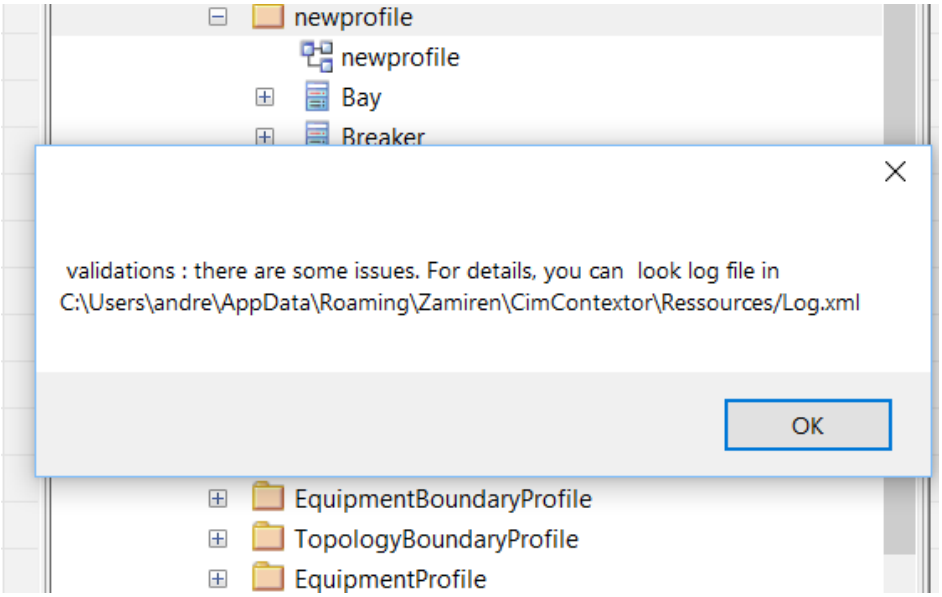
The errors are stored in the log file which is:

- either in "*User/AppData/Roaming/Zamiren/CimContextor/Ressources/Log.xml*"
- or in the same folder as the EA project, if there is a local config file in that folder.

Do not forget to select the Log option in the menu options:



The location of the log file is displayed when the checking is achieved in case there are errors for instance;



Here is an example of an excerpt of the results:

<add-inLogs>

<logDate date="11-21-2018">

<log time="23:37:56" username="ANDREDELL2\andre" ConfigSaved="Element:EnableESMPHierarchy Value:Unchecked" />

<log time="23:37:56" username="ANDREDELL2\andre" IntegrityCheck=" info: ++++++++ the package newprofile is beeing checked for consistency ++++++++++++" />

<log time="23:37:56" username="ANDREDELL2\andre" IntegrityCheck=" ********** entree dans verif_element Bay**********" />

<log time="23:38:06" username="ANDREDELL2\andre" IntegrityCheck=" issue: get IBO element for Bay La reference d'objet n'est pas definie Ã une instance d'un objet." />

<log time="23:38:06" username="ANDREDELL2\andre" IntegrityCheck=" ********** entree dans verif_element Breaker**********" />

<log time="23:38:06" username="ANDREDELL2\andre" IntegrityCheck="info: the type of attribute Breaker.normalOpen is Boolean" />

<log time="23:38:07" username="ANDREDELL2\andre" IntegrityCheck="info: the type of attribute Breaker.open is Boolean" />

<log time="23:38:07" username="ANDREDELL2\andre" IntegrityCheck=" ********** entree dans verif_element ConductingEquipment**********" />

<log time="23:38:07" username="ANDREDELL2\andre" IntegrityCheck="info: the type of attribute ConductingEquipment.aggregate is Boolean" />

<log time="23:38:07" username="ANDREDELL2\andre" IntegrityCheck="info: the type of attribute ConductingEquipment.mRID is String" />

<log time="23:38:07" username="ANDREDELL2\andre" IntegrityCheck=" ********** entree dans verif_element EquipmentContainer**********" />

The syntax of a log line is:

<log time=xxx username=yyy IntegrityCheck= "a line of information"

A line of information begins with either:

- "Info:", if it is an info
- "issue", if it is an issue

Each checking of an element begins with:

*<log time=xxx username=”yyy" IntegrityCheck=" ********* entree dans verif_element zzz*****" />*

The text after “info:” or “issue:” is self-explanatory. For instance in the following example:

IntegrityCheck=issue: get IBO element for Bay La reference d'objet n'est pas definie Ã une instance d'un objet."

The text which comes from a code exception in local language means that the object is not based on an Information Model object.

It is impossible to tell if the type of an object is correct, so this type is printed as info line and may be checked visually:

<log time="23:38:07" username="ANDREDELL2\andre" IntegrityCheck="info: the type of attribute ConductingEquipment.mRID is String" />

19.3. ESMPIntegrityCheck

This program does all the checks of “IntegrityCheck” while adding some additional checking adapted to the rules followed by European Style Market profiles (see IEC62325-450).

Some of these rules are:

- The existence of a regional contextual profile (IEC62323-351) from which all the other profiles are derived,

- All association must have an end role of type “shared”,
- All classes have an additional stereotype “ABIE”,

  All identifiers of the type of an attribute must be defined in a special datatype domain shared by all profiles and different from the CIM,

The location and the syntax of the log file are identical as the “IntegrityCheck”'s:

- either in "*User/AppData/Roaming/Zamiren/CimContextor/Ressources/Log.xml*"
- or in the same folder as the EA project, if there is a local config file in that folder.

19.4. CGMESIntegrityCheck

This program checks the profile (graph) according to the practice of Entsoe CGMES. In the log file, programming or system issues (like access to fines) and not profiling issues may be specified.

The results of checking the profile can be found at the same location as the log file under the name "*CheckProfileErrorFiles.csv*". This is a spreadsheet with the following columns:

Message: a self-explanatory message concerning a found issue,

Package: the checked package,

Class: the checked class,

Attribute: the checked attribute,

Association: the checked association,

CIMText: content on the CIM side,

ProfileText: content on the profile side.

Here is an example of the resulting file:

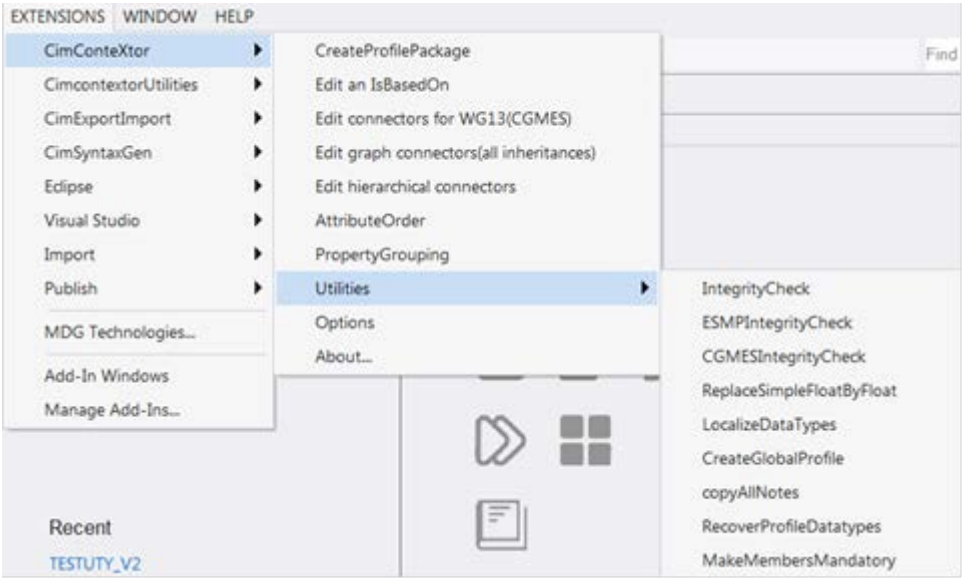
Message	Package	Class	Attribute	Association	CIMText	ProfileText
Info	Check the Profile Graph2Profile	CIMversion=62325CIM03v01	date=2018-10-10	Profile Version=Euml	from CIMVersion=Buml	
HasNote	Graph2Profile					Beginning of the note ="mon profil
NotSameNote	Graph2Profile	A1			79	110
HasAttachedNote	Graph2Profile	A1				Beginning of the note ="note de classe
NotSameCardinality	Graph2Profile	A1	at1		for CIM attribute LowerBound=0	LowerBound=1
NotNaturalDirection	Graph2Profile	A1		b2 -> a3	b2[0..*]	a3[0..*]
NotSameNote	Graph2Profile	A3			77	137
NotSameCardinality	Graph2Profile	A3		a3 -> b2	b2[0..*]	a3[0..*]
NotNaturalDirection	Graph2Profile	A3		a3 -> b2	b2[0..*]	a3[0..*]
NotNaturalDirection	Graph2Profile	A3		b2 -> a3	b2[0..*]	a3[0..*]
NotConsistantEnumerationValues	Graph2Profile	ActivePower	multiplier		for CIM initial value=	for profilemultiplier initial value=22
NotSameNote	Graph2Profile	B1			83	111
NotSameNote	Graph2Profile	B2			82	109
NotSameAbstract	Graph2Profile	B2			Abstract=0	Abstract=1
HasNonAttachedNote	Graph2Profile					Beginning of the note ="note dissociÃ©e

19.5. Profile building helps

Utilities offers many functions to help profiles building:

- "*ReplaceSimpleFloatByFloat*" enables the change of CIMDatatype called "*Simple_Float*" for a "Float" Primitive in CGMES older profiles,
- "*LocalizeDataTypes*" allows older version of CGMES profiles to have their own specific domain package, instead of sharing a common one,

- "*CreateGlobalProfile*" enables the creation of a profile from an information model package or a diagram, (this is different from "*CreateProfile*" feature described in section),
- "*CopyAllNotes*" is a function that replaces all profile element description (notes) by the Information Model element description (useful in case of a new CIM version),
- "*RecoverProfileDatatypes*" checks if datatypes used as profile attribute type are profile datatypes or not. When it is not the case, it corrects the profile and if needed create the profile datatypes needed ,
- "*MakeMembersMandatory*" makes every attribute in a profile class mandatory.



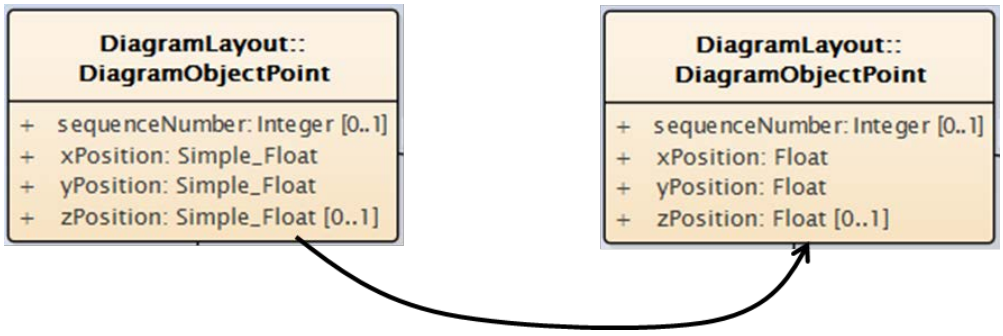
19.6. Use of "ReplaceSimpleFloatByFloat"

CIMDatatype "Simple_Float" was used to specify the value space of a single precision floating point number, while Primitive "Float" in CIM is specifying the value space of a floating point number (simple or double precision…).

CIM wants to change the definition of Float to be a simple precision value space.

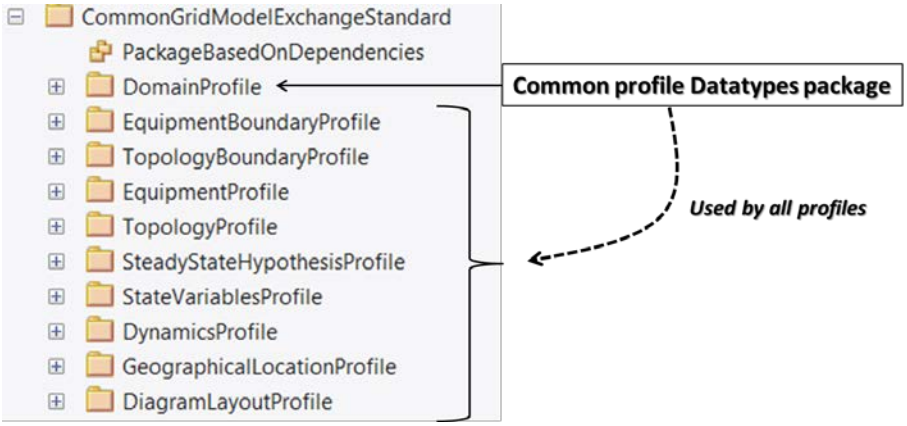
In order to keep existing profiles with this new definition, this utility feature replace Simple_Float datatype by Float in a given profile package.

Select a profile package, launch the ReplaceSimpleFloatByFloat, the attribute datatype will be changed in every class of a package.

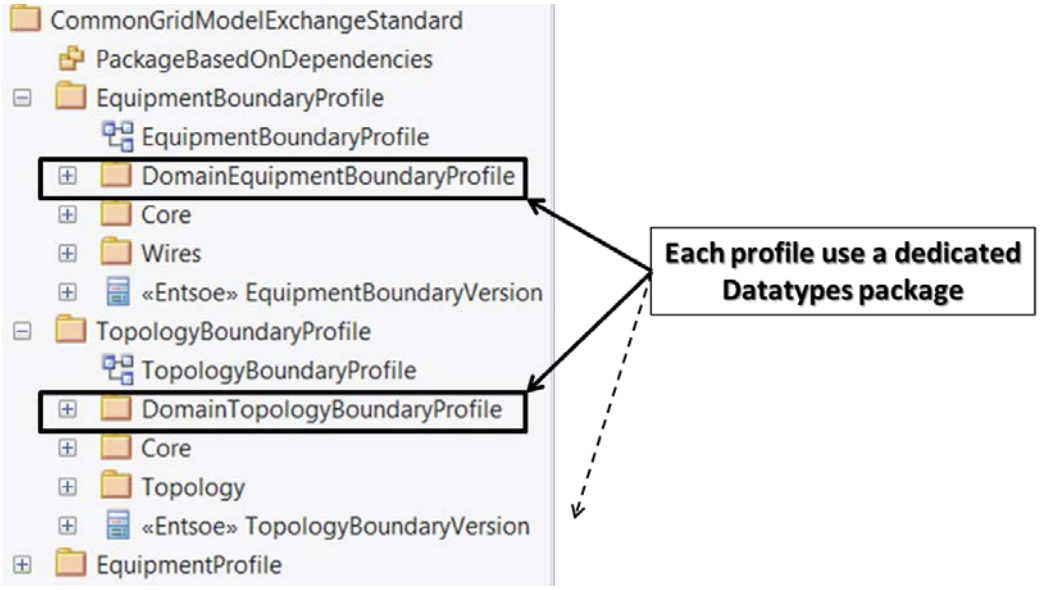


19.7. Use of LocalizeDataTypes

As stated in section 2, there are many ways to define the profile package structure. For example, all the profiles could share a common Domain package that holds all the profiles datatypes, this was the case for CGMES:



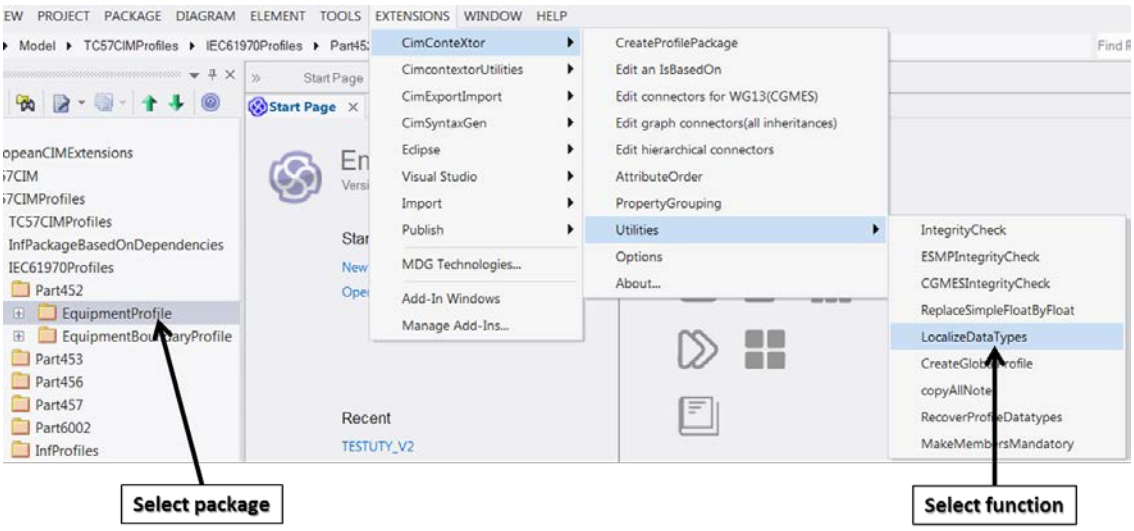
But another design could have been to have one domain package for each profile, like this:



The LocalizeDataTypes Utility feature is a mean to move from a one shared domain package to one domain package per profile. The name for each profile domain

package is the name of the profile (example EquipmentBoundaryProfile) prefixed by Domain.

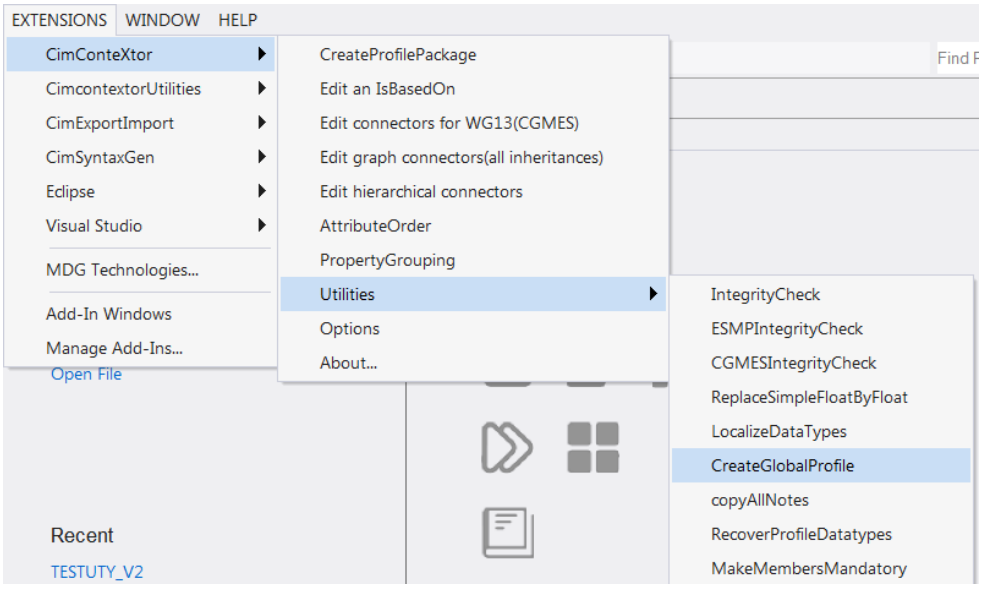
The way to change the structure is to select a profile package, then select the "*LocalizeDataTypes*" feature. CimConteXtor will create the dedicated domain profile package and change the type of each profile class attributes to match the profile domain datatypes.



19.8. Use of "CreateGlobalProfile"

"Utilities/CreateGlobalProfile" is different from the "CreateProfilePackage" (see section 4). The second one is used to create the template for the package profile, when the first one is used to create also profile classes.

The starting point for "CreateGlobalProfile" is either a package or a diagram. So select either a package or a diagram and then launch the Utilities/CreateGlobalProfile:



19.8.1. CreateGlobalProfile for a package

If you have selected a package and launched the command CreateGlobalProfile, see section 28.1 AnnexF.

19.8.2. CreateGlobalProfile for a diagram

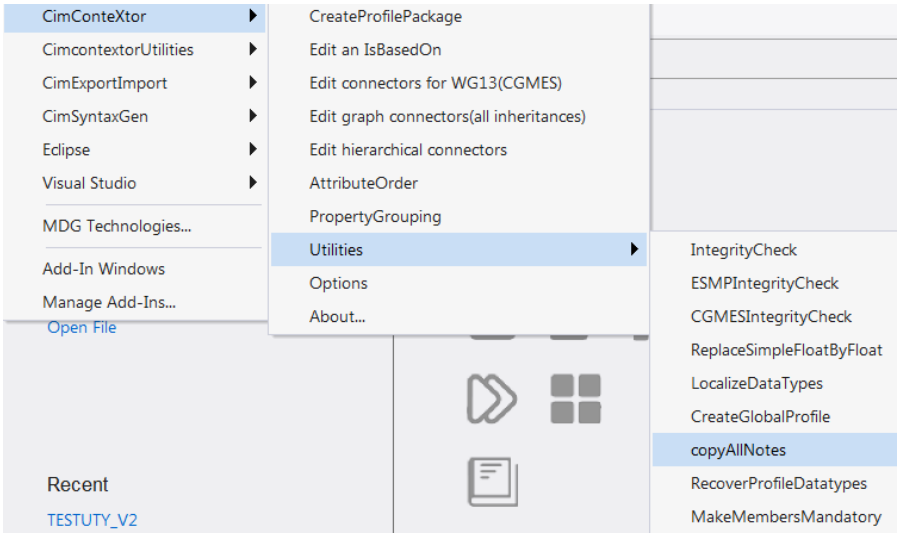
If you have selected a diagram and launched the command CreateGlobalProfile, see section 28.2 AnnexF.

19.9. Use of "CopyAllNotes"

CopyAllNotes is used to update the description (notes) of profile elements (class, attribute, end role name). This happens when a new version of the information model is delivered, and where some descriptions have been changed: so it is a way to keep profile description with information model ones.

Note: this supposed that the description in the profile must be the same as the one in the information model.

Select a profile package and launch the *"Utilities/CopyAllNotes*":

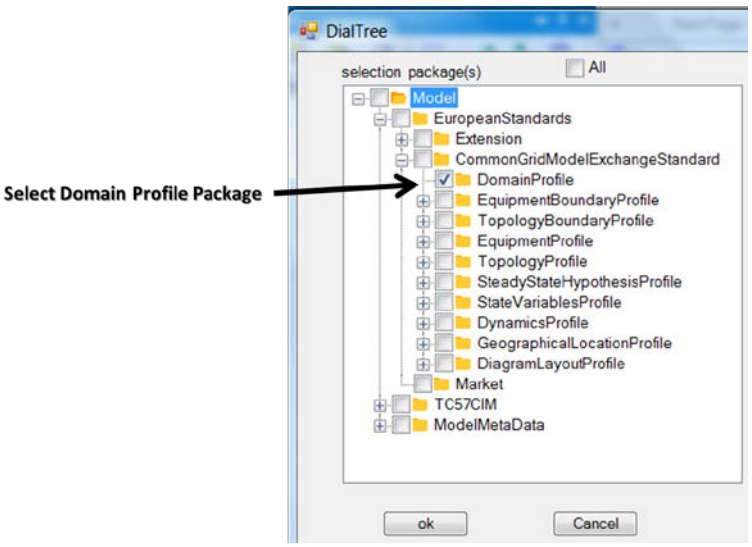


After the end of the process, the profile element descriptions have been updated.

19.10. Use of "RecoverProfileDatatypes"

"RecoverProfileDatatypes" is used to check if the datatype used as the type of a profile attribute is belonging to the right domain package: usually it should belong to a domain profile and not to the Information Model one. If the datatype is not pointing to the right profile package, it makes the change if the datatype exists, or it creates the datatype in the domain profile package and then it makes the change.

Select a package where you want to verify datatypes, you will be prompted by a dialog box, asking you to select the domain profile package that holds the right datatypes for the given profile:

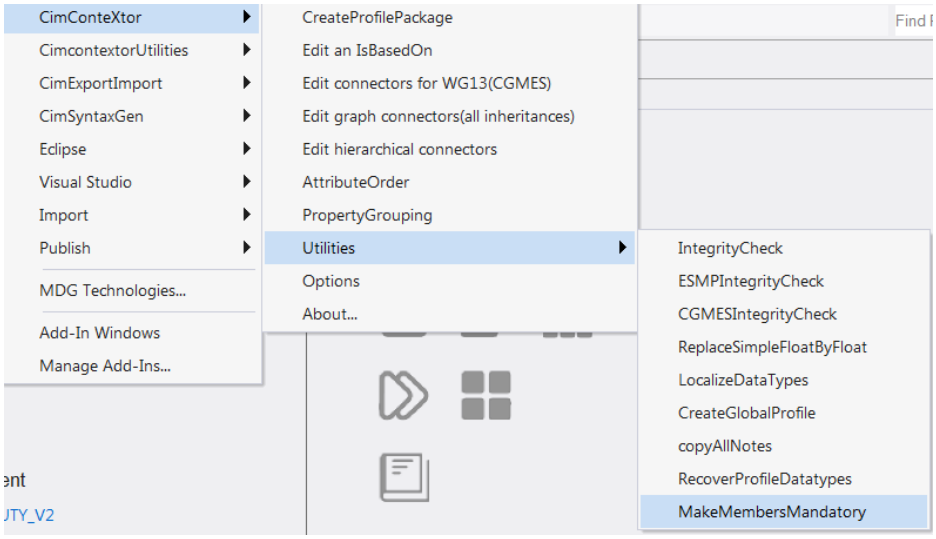


Click "OK" and the process will end: wrong datatypes pointing has been changed, and missing profile datatypes have been created.

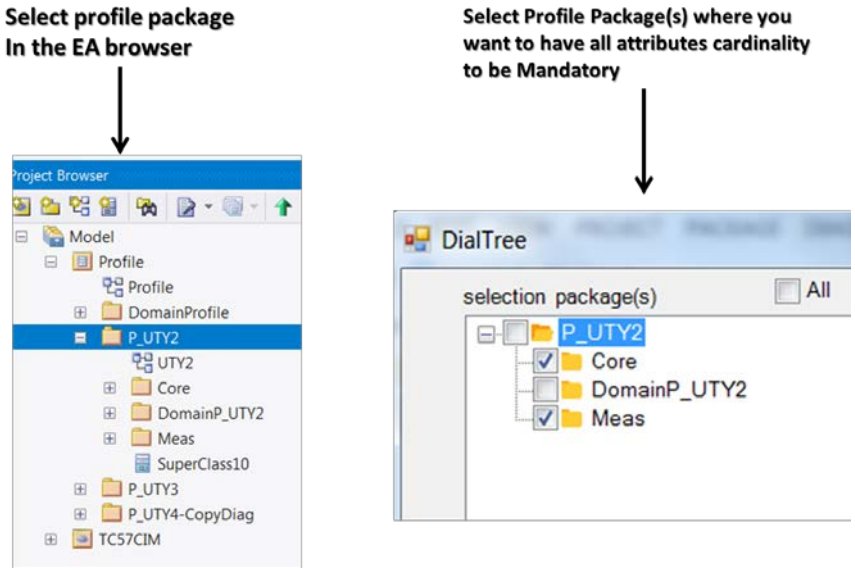
19.11. Use of "MakeMembersMandatory"

MakeMembersMandatory is a feature that enables the profile class attribute cardinality to be changed from optional to mandatory. This is useful, in case most of the attributes of the profile are "mandatory", adjustment could be done after by using the EditIsBasedOn feature.

Select a profile package and launch the Utilities/MakeMembersMandatory:



You will be prompted by a dialog box, asking you on which package or sub package(s) you want to run the feature:



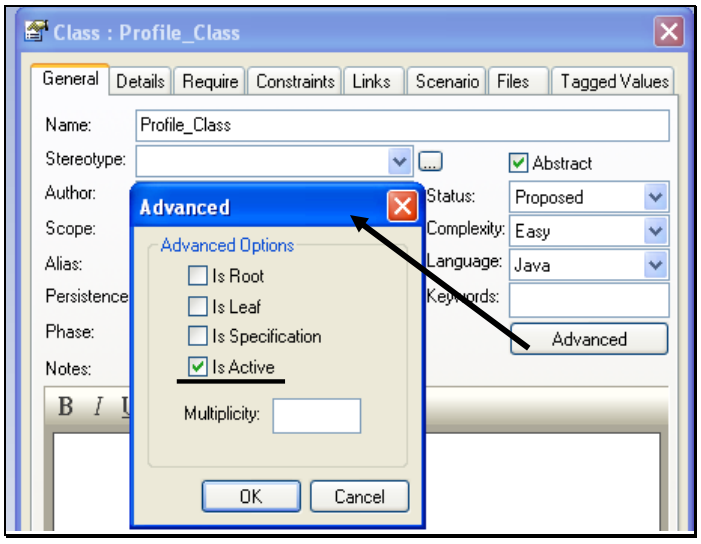
Then click on "OK", and at the end of the process, every attribute's cardinality is mandatory in the selected package(s).

20. XSD Syntactic Generation

CimConteXtor does not do syntactic generation: to do that, you could use another Add-in "*CimSyntaxGen*", which performs, in its first version, RDF Schemas and XSD schema generation (see *CimSyntaxGen* user's manual). But for syntactic generation, there is some information that must be given, and UML could be used to do that : here are some examples.

20.1. Root Classes

When the syntactic target generation is XML Schema (see *CimSyntaxGen* user manual), you must define before which classes will be taken as « *Root Classes* », and therefore will be the starting point of the hierarchies for the XML Schema. In order to do that you must check the "*Is Active*" box in the "*Advanced*" window of the class properties.



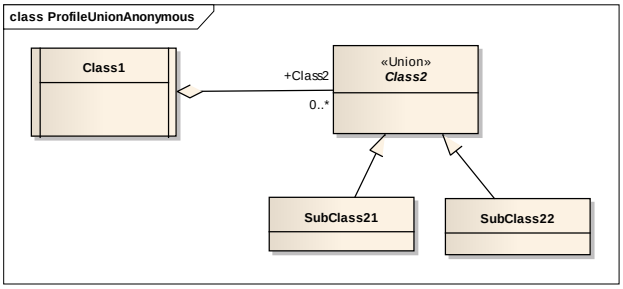
Note : there could be several "*Root*" classes in the profile, but each hierarchy under these "*Root*" classes must be independent.

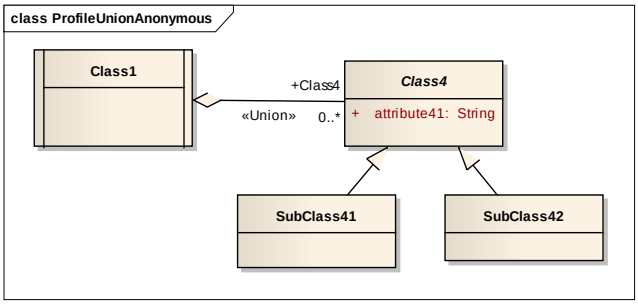
20.2. Use of "Union" and inheritance

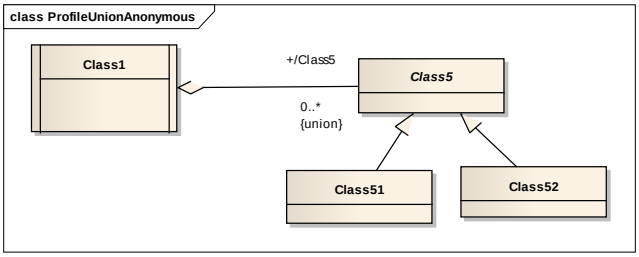
Union could be used to express, in a short cut, the fact that an aggregation between a class and an abstract superclass implies that all aggregation between the class and the subclasses are taken into account in the XSD.

Union could be seen several ways and may be defined at several levels :

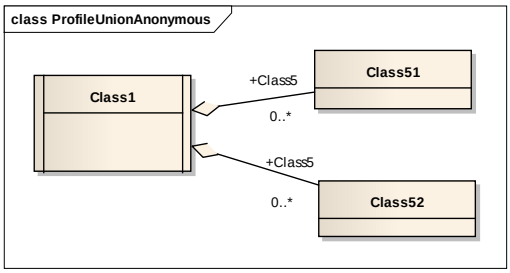
- The superclass could be marked with a Union stereotype,
- The aggregation could be marked with a Union stereotype,
- The aggregated end role name could be marked using a {union} constraint.







Note: the mapping to XSD is different than the following representation (see CimSyntaxGen).



20.3. Element Order (Attribute Order)

The order of the XML elements is driven by the attribute order specification.

21. Configuration File

21.1. Managing configuration file

CimConteXtor has an xml configuration file (*Config.xml*) that is created at the add-In installation. It is created in the following directory:

- "*User/AppData/Roaming/Zamiren/CimContextor/Ressources/*"

This file could be updated by the Options Menu or manually.

This file can be copied in a EA project folder, usually to be able to have a config file tailored for a project. This has a consequence: when an EA project is launched, there are two cases:

- If there is no configuration file in the same folder as the EA project, the used configuration file will be the one in the User/AppData place,

- If there is a specific configuration file in the same folder as the EA project, it is this config file that will be used,
- But there is also another side effect: the log files for all features and results files for Integrity Checking will be put in the folder of the active config file for the project.

If you want to have a specific configuration, duplicate the configuration file in the target folder of the project, with your own setting of parameters. This means:

- You have to manage your configuration file,
- And if the configuration file has new features, you will have to update your own configuration file.

21.2. Overview

The goal of this file is to provide parameters for CimConteXtor that could be tailored by the user to fulfil its requirements. It means that the user could change some parameters and add new ones.

The file is coming with five different kinds of parameters:

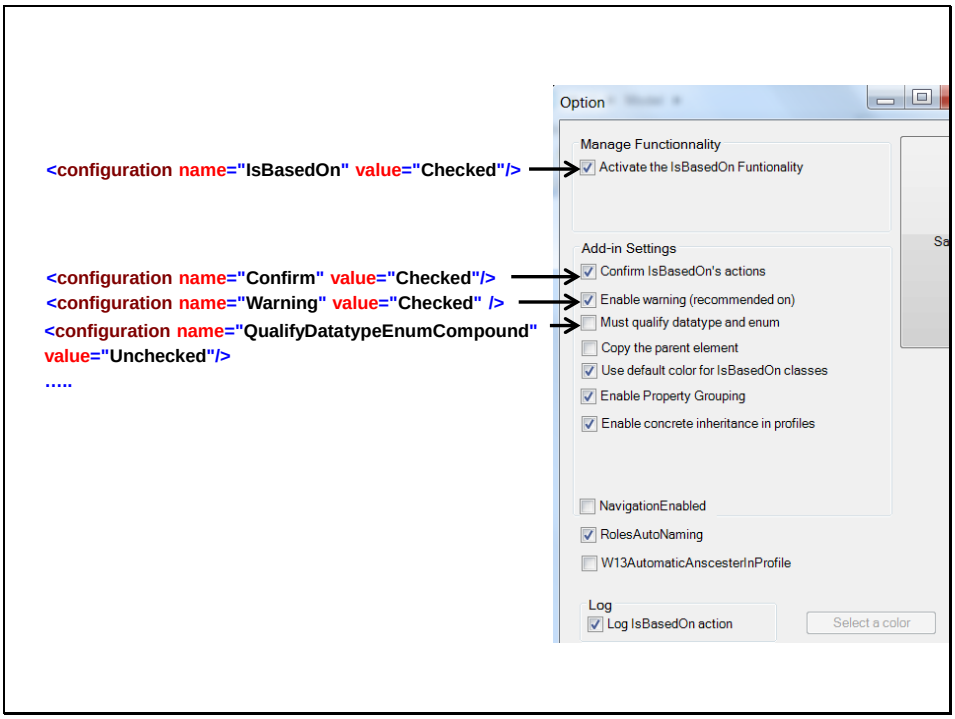
- appSettings: this gives the state of elements in the Option Menu,
- dataProfiles: this gives a list of stereotypes used in a profile, or what is the profile domain package name,
- dataQualifier: this gives a list of predefined qualifiers that could be used by some given UML element,
- dataConstraint: this gives a list of constraints that could be used by some given UML element,
- dataClassifierConstraint: this gives a list of constraints that are used to restrict Primitive value space.

21.3. AppSettings parameters

AppSettings parameters are defined as xml element called "*configuration*" with a name and a value. Example:

```
<configuration name="IsBasedOn" value="Checked"/>
```

Most of those parameters are used for defining the Option Menu where the user can check or uncheck parameter box. The AppSettings give the default for the option parameters: checked or unchecked boxes:



These parameters are (see section 3.3 for the meanings):

- IsBasedOn
- Confirm
- Warning
- QualifyDatatypeEnumCompound
- CopyParentElement
- ConfigColor
- EnablePropertyGrouping
- EnableConcreteInheritanceInProfiles
- NavigationEnabled
- AutomaticChangeOfRoleName
- WG13AutomaticAnscesterInProfile
- Log

These parameters values are updated when doing a Save in the Options Window.

There are some more parameters that are not related to the Option Menu; they are not shown, but they drive CimConteXtor behavior:

- *EnabledIntermediaryInheritance*: this will enable the use of a partial hierarchy path. It means that a class could inherit from any of the inheritance hierarchy class: this will be shown in the inherit box of the EditIsBasedOn windows,
- *EnableESMPHierarchy*: this a parameter for the ESMPIntegrityCheck menu, that check if the hierarchy of packages and dependencies is good,

- *EnableCheckAttributeIdentifier*: for IntegrityCheck also. When this parameter is set, Integrity check verify that the attribute type is conforming to the type of information model one,
- *SimpleProfiling*: for future use. If set, it prohibits the use of qualifiers.
- *TimeAnalysis*: for development only. If set, it allows performance analysis.

21.4. DataProfiles parameters

These parameters are specific for profile management, they are called "profdata". They are currently two of them:

ListStereoNamespace: used to handle list of stereotypes used by Entso-E CGMES, list for IntegrityCheck. Current values: " **Entsoe|ShortCircuit|Operation|Abstract",**

EntsoeDataTypesDomain: gives the name of profile domain package for Entso-E CGMES profiles.

ProfilesPackage: gives the name of the package that contains profile packages and Domain package.

```
<profdata name="ListStereoNamespace" value="Entsoe|ShortCircuit|Operation|Abstract"/>
<profdata name="EntsoeDataTypesDomain" value="DomainProfile"/>.
<profdata name="ProfilesPackage" value="Profile" />
```

21.5. DataQualifier parameters

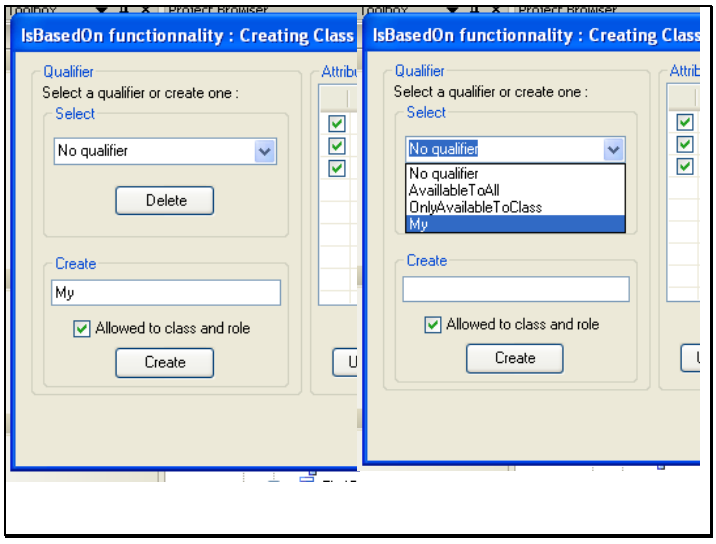
These parameters give the list of predefined "qualifiers" for given UML elements, they are called "qualifier", have a name and have an assignment to UML elements given by the value of "AllowedTo". Example:

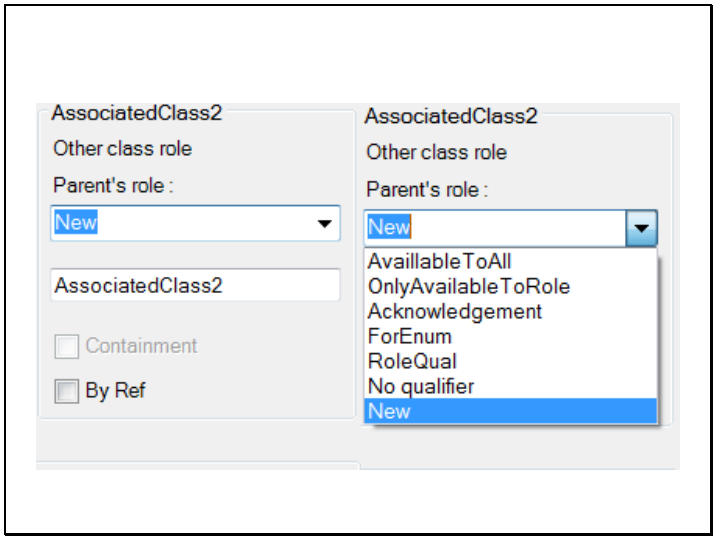
```
<qualifier name="Availlable" allowedTo="any"/>
```

The list of "*allowedTo*" value:

- Any: qualifier could be used by class, datatype and end role name,
- Class: qualifier could be used only by class,
- Datatype: qualifier could be used only for datatypes,
- Role: qualifier is only for end role name.

The list of qualifiers is used by the drop-down menu of the qualifier box of either the "*EditIsBasedOn*" class (class and datatype) windows or the "*ModifySelectedAssociation*" windows:





The list is dynamic, because when you use the manage qualifier feature for associations or create qualifier feature for classes, when save, the created qualifier is added to the configuration file.

Note: check the config file and make your own qualifiers if needed.

21.6. DataConstraint parameters

These parameters called "*constraint*" give predefined constraint for class (including datatype) and association. These constraints will appear in the drop-down menu.

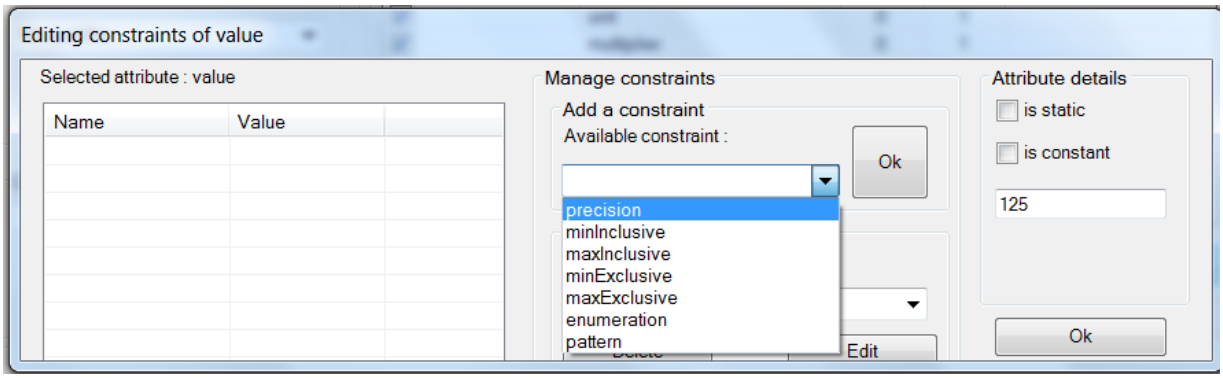
<constraint name="testDatatype" type="OCL" allowedTo="datatype" notes="OCL constraint">.

- name: name of the constraint
- type: OCL or INV
- allowedTo: any, class, datatype, attribute and role
- notes: constraint expression

21.7. DataClassifierConstraint parameters

21.7.1. Overview

These parameters called "*classifierconstraint*" define a value space constraint for a given <<Primitive>>. They can only be defined in the configuration file. They will appear in the drop-down menu of an "*EditFacet*" window of a CIMDatatype "value" attribute.



There are different classifier constraints:

1. Constraints that are XML datatype facets,
2. Constraints that are predefined XML value space (example token…).

A data classifier is defined by the following element:

- name: name of the facet, or name of the value space,
- type: type of constraint defined in the value element (OCL or INV),
- allowedTo: primitive on which the constraint applied,
- variable type: type of the variable that will be used in the constraint expression (any, Numeric, an xml primitive, or DEFINED)
- variable list: if the variable type is "DEFINED", this is the list of possible values
- comment: a comment on the constraint
- constraint expression: usually an OCL constraint that takes a parameter (N, P, I or C) to output the right expression with the parameter entered by the user.

DataClassifier example:

```
<classifierconstraint
        name="minInclusive" ➔ xml minInclusive facet
        type="OCL" ➔ type of output constraint
        allowedTo="DateTime" ➔datatype it is applied to
        variableType="Integer" ➔ the value that will be entered will be an integer
        variableList="" ➔ reserved for DEFINED variable Type
        comment=""> ➔any comment for this constraint
                inv: self-&gt;minInclusive($N$) ➔ template for the constraint expression
</classifierconstraint>
```

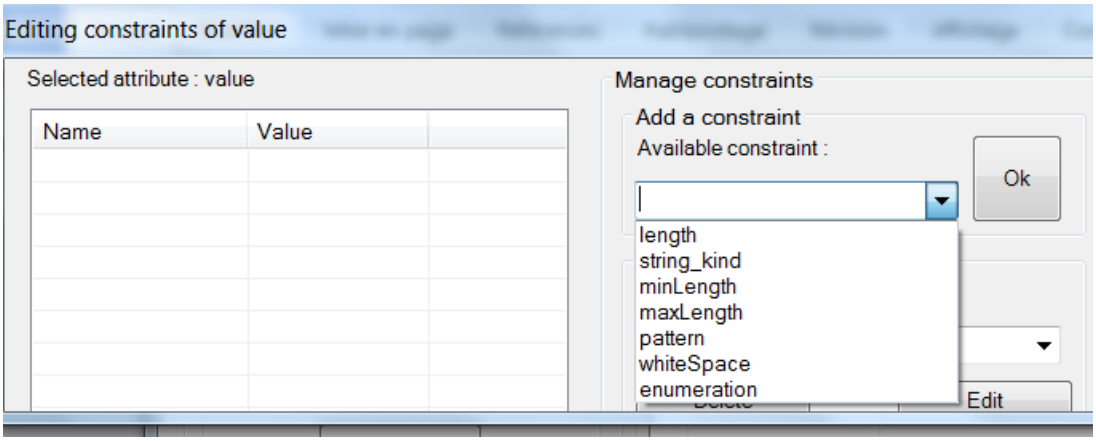
21.7.2. Classifier constraint as XML facets

These constraints give the parameters to define a restriction on the value space of a given primitive, to allow the user to define the value of the restriction and to allow CimConteXtor to generate the corresponding expression in OCL. How does it work?

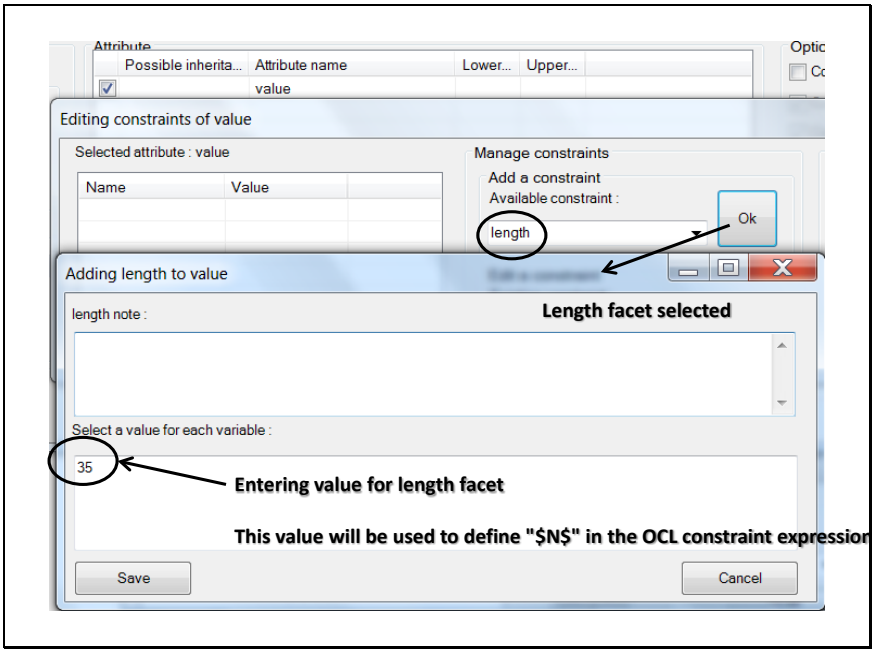
Example of a "*length*" restriction for String. The following classifier constraint is in the configuration file:

```
<classifierconstraint name="length" type="OCL" allowedTo="String"
variableType="Numeric" variableList="" comment="">
inv: self-&gt;Length($N$)
</classifierconstraint>
```

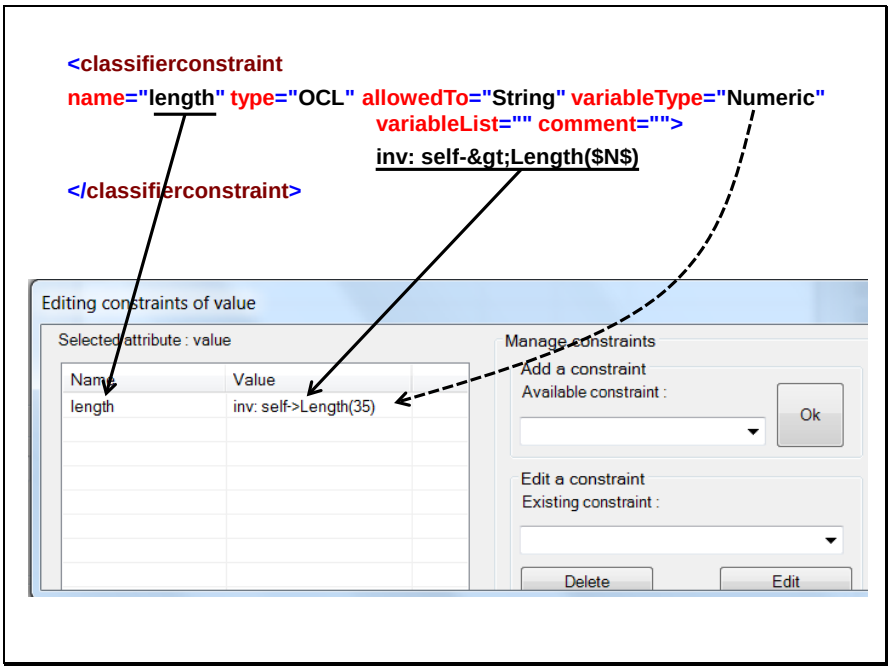
When the "*EditFacet*" of a value attribute is activated, the following window appears, and in the drop-down menu of the available constraint field there is the "*length*" facet:



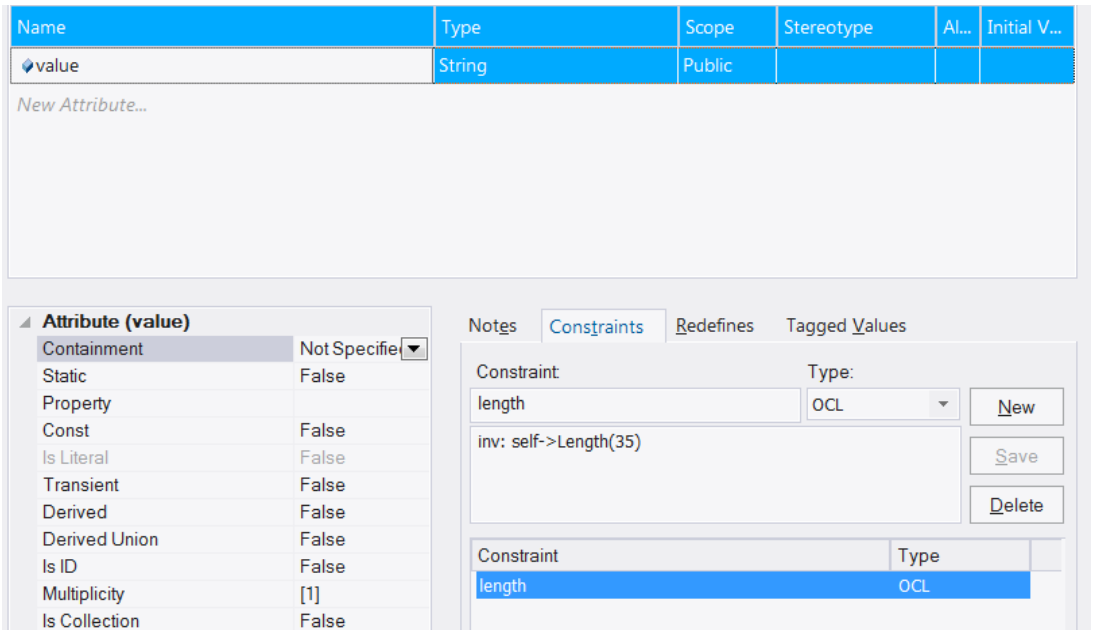
Selecting "length" and make "Ok" will open a new window:



By entering a value (here a numeric value), the restriction is saying that the length of the string will be this value (35 in the example). By making a "Save", this will enter the restriction and generate the OCL constraint (according to the template define in the constraint classifier: " *inv: self->Length(N)" = "inv:self->Length(35*)".



This constraint is now visible in EA datatype properties window:



The list of constraint classifier is that of xml datatypes.

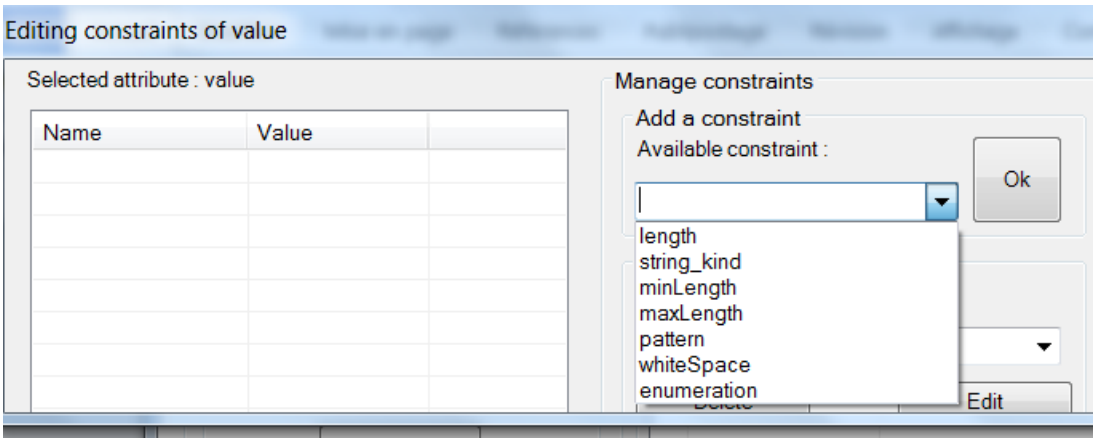
21.7.3. Classifier constraint as XML predefined datatypes

These constraints give the parameters to select a predefined named xml value space that restricts already the value space of a given primitive. This will allow the user to use predefined value space, which may also be restricted. CimConteXtor will generate the corresponding value space as a name. How does it work?

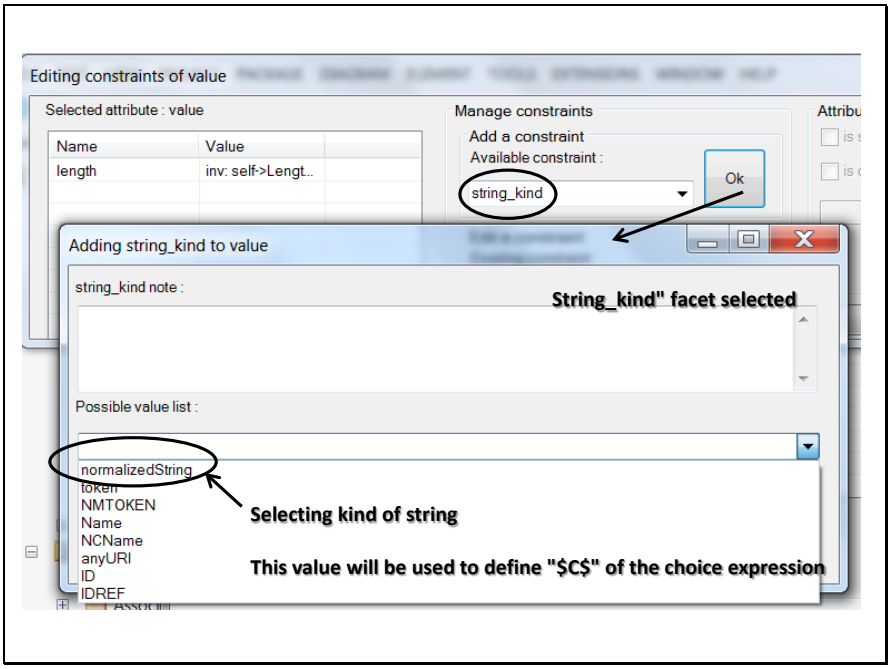
Example of a "*string_kind*" restriction for String. The following classifier constraint is in the configuration file:

```
<classifierconstraint name="string_kind" type="INV" allowedTo="String"
variableType="DEFINED"
variableList="normalizedString,token,NMTOKEN,Name,NCName,anyURI,ID,IDREF"
comment="">
choice=$C$
</classifierconstraint>
```

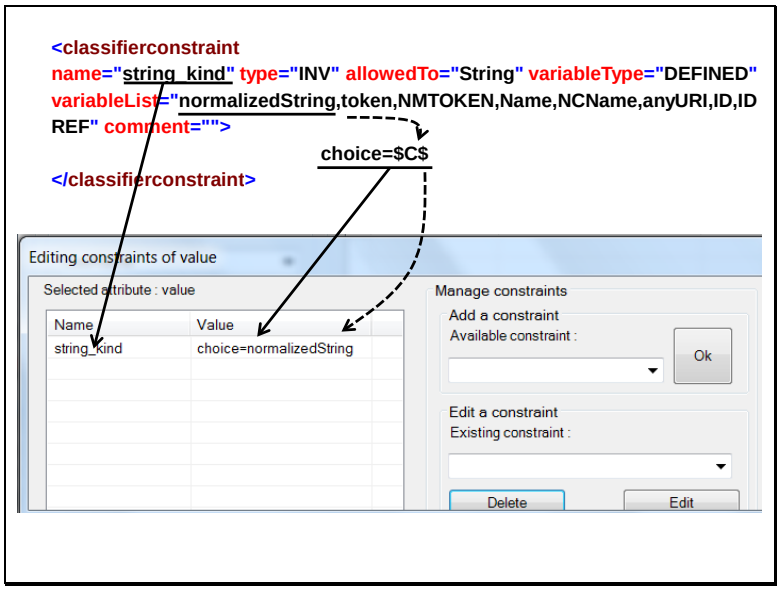
When the "*EditFacet*" of a value attribute is activated, the following window appears, and in the drop-down menu of the available constraint field there is the "*string_kind*"" facet:



Selecting "string_kind" and make "Ok" will open a new window:



By selecting one of the values (here a normalizedString), the restriction is saying that the value space is that of an XML datatype (normalizedString in the example). By making a "Save", this will enter the selected value and generate a choice constraint (according to the template define in the constraint classifier: "*choice=C = "choice=normalizedString*".



Predefined XML value spaces exist for:

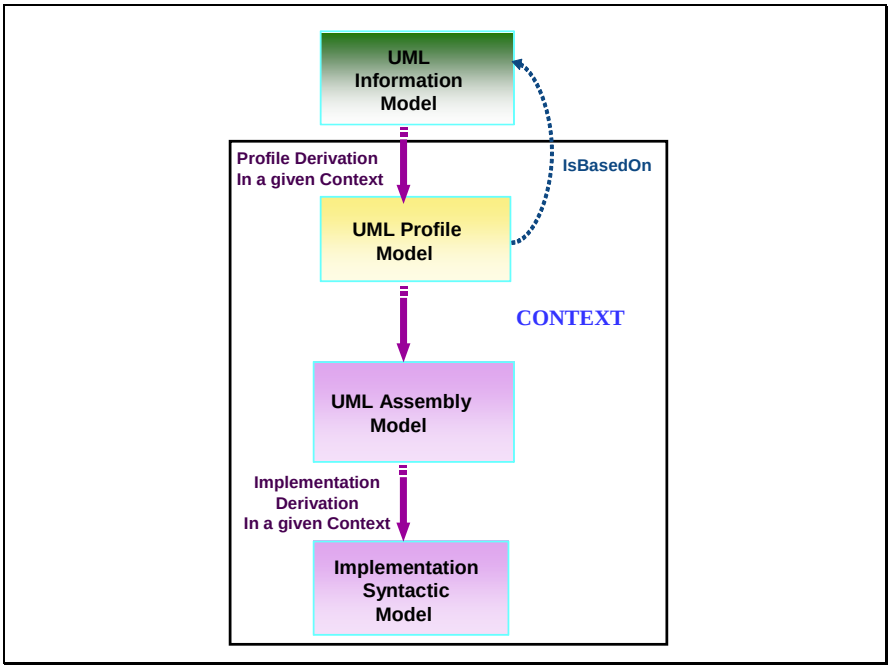
- String: normalizedString, token, NMTOKEN, Name, NCName, anyURI, ID, IDREF
- Integer: long, int, short, byte, unsignedLong, unsignedInt, unsignedShort, unsignedByte.

Annexes

22. Annex A: Modelling methodology summary

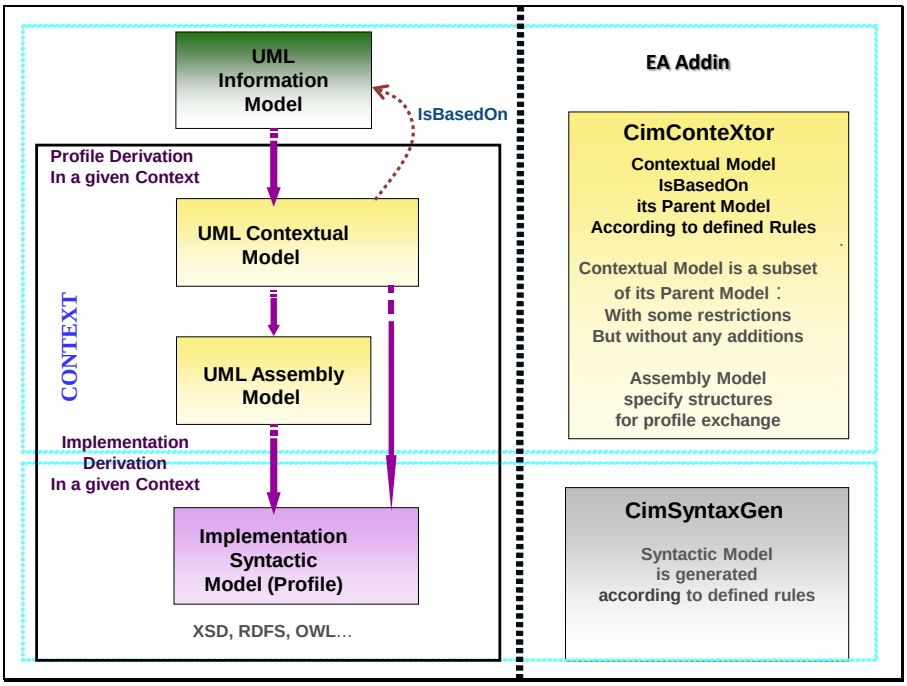
The modelling methodology that is behind *CimConteXtor* defines four modelling levels :

1. the Information Model level that defines a domain model (for example the CIM for Electro technical Domain),
2. the Profile Model Level, that defines how this information model is used in a context, or which subset of this information model is used in some given context (example CPSM -- Common Power System Modelling -- network exchange) : this is the "*IsBasedOn*" concept,
3. the Assembly level, that defines how the profile artefacts are assembled for exchange,
4. the syntax level, that defines in which syntax the exchange will take place, and how the Assembly level will be mapped to the chosen syntax.



The first three levels are modelled in UML.

CimConteXtor is a tool that builds an UML Profile and possibly an Assembly Model based on an UML Information Model. *CimConteXtor* is used with companion tools that generate syntactic schemas (XSD or RDFS) according to given Assembly rules and Syntax Naming and Design Rules (See *CimSyntaxGen* Add-In Tool user Guide for XSD and RDFS generation).



23. Annex B : Information Model UML modelling

23.1. UML Class Modelling

UML models artefacts (or objects) are classes that have attributes (their simple properties) and that have associations (their complex properties) between them. UML uses a graphical notation to represent all these elements (Class, attribute, association).

Attributes :

- have cardinality (optional or mandatory).
- have a type : either a Primitive, an enumeration, a Datatype, or a Compound,
- can be grouped (in CIM, grouping is defined by a named class stereotyped by <<Compound>>), in order to be able to reuse this group in different classes.

Associations :

- define relations between classes,
- could be of different types : bidirectional, aggregation or composition,
- have multiplicity and have end role names.

23.2. UML element attached information

In UML various information can be attached to classes, attributes, associations, datatypes and compounds. This information are : Stereotype, Note, TaggedValue, Constraint...

Stereotypes :

Stereotype is a general mechanism that is used to categorize the various elements : classes, attributes, associations. In UML, they are shown between double brackets <<...>>.

Notes :

Notes are used to carry all textual information about an element (Class, attribute, association, datatype). A good practice is to use it just for giving the definition of the element (to make a good definition see ISO/IEC 11179).

Most of the time, people use it also to give usage rules of the element, but they should use another item for that : *TaggedValue*.

TaggedValue :

TaggedValue specifies additional information about an element. TaggedValue has a name, content (the value) and notes. So a TaggedValue could be used to specify a category of additional information like “Usage Rule”, “Business Term”...

Constraints :

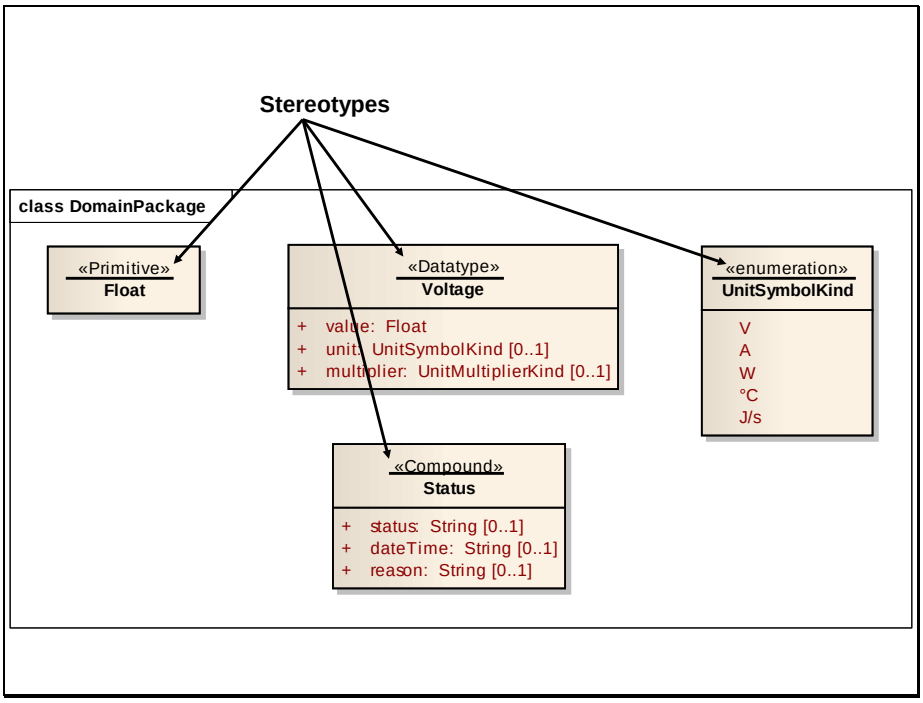
Constraints are conditions imposed on one or multiple elements.

- For a class it could be the conditions under which the element can exist or function,
- For an attribute, it could say something about the rules and conditions under which the attribute is used,
- for association, it could say something about the rules and conditions under which a relation operates.

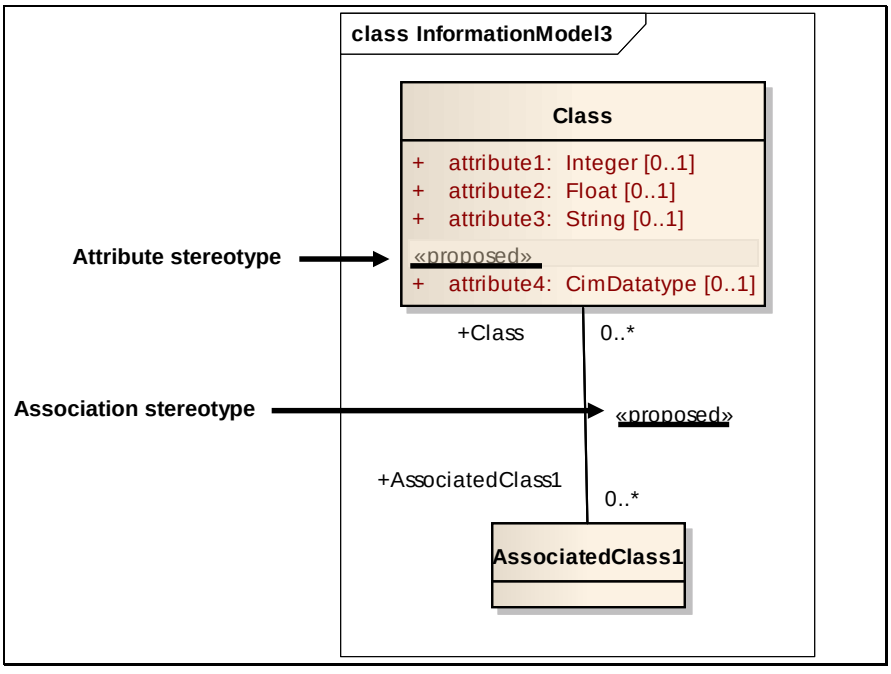
Constraints could be expressed in different formats or languages that could be specified too. UML proposes to use OCL (Object Constraint Language), but other languages could be used too. For example, *CimConteXtor* is using OCL to express facets for Datatypes.

23.3. UML stereotype examples

All elements can be stereotyped. A stereotype is information that is attached to an element in order to classify it, and to group elements that have common characteristics. For example, there are different kinds of attribute types: Primitive, enumeration, Datatype and Compound. Each of these types is stereotyped: in UML, stereotypes are represented by a name enclosed in double brackets <<Primitive>>, <<enumeration>>, <<Datatype>>, <<Compound>>...



Associations and attributes can also have stereotypes. Example: stereotype <<*proposed*>> can be used to express an Information Model proposed extension.



23.4. Information Model Naming Rules

At the information model level, all the naming rules should be in conformity with TC 57 naming rules (similar to UN/Cefact CCTS ones):

- Use of UpperCamelCase for Class and Association,
- Use of lowerCamelCase for Attributes,

- Names should be in their singular form, except if the concept is plural,
- Names can include several terms, but use of articles and prepositions should be prohibited,
- Use of Acronyms and Abbreviations should be explained in definition, and a good practice is to control the list of allowed acronyms and abbreviations.

Enumeration names always ended up with the "*Kind*" term : "*EnumerationKind*"

23.5. Information Model modelling rules

At the information model level:

- All attributes and associations are optional,
- Allowed attribute types are : Primitive, enumeration, Datatype and compound,
- Attributes types are the least restricted ones,
- All associations are bidirectional ones,
- All classes, attributes and association end role names follow naming rules.

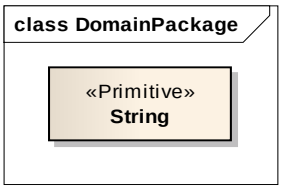
23.6. Datatypes Types and UML modelling rules

There are different types of datatypes : basic and domain datatypes.

Basic Datatypes :

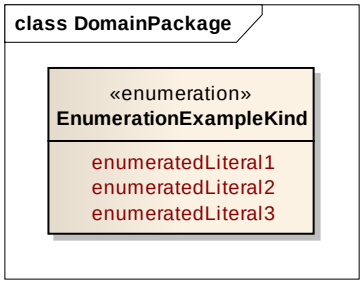
By definition, a datatype is a set of distinct values, characterized by properties of those values and by operations on those values (see ISO/IEC 11404 standard). According to this standard, there are three kinds of basic datatypes :

- Primitive : a primitive defines a value space axiomatically, without reference to any other value spaces. In UML, primitives are represented as a classifier with no attributes. Here, the stereotype used for a "*user primitive*" is "Primitive" with an upper case to distinguish it from UML own primitives (that are stereotyped with a lower case "primitive") or even from UML tool primitives :



- Enumeration : an enumeration is a subset of a primitive value space and defines a list of allowed values taken in the primitive value space. In UML, enumerations are represented as a classifier with "*enumeratedLiterals*" not with attributes. Stereotype used for

enumeration is “enumeration” :



- Simple Datatype : a simple datatype defines a subset of a primitive value space by some kind of restrictions. Simple Datatypes are represented as a classifier with one attribute named “value”, whose type is a primitive, and whose primitive value space restrictions are expressed as value attribute constraints. Here, stereotype used for simple datatype is “Datatype” to distinguish it from usual UML datatypes (that are stereotyped with “dataType”). There is a relationship between the primitive and the datatype, and this relation is an “*IsBasedOn*” one (see sections on profile and on Datatype IsBasedOn process)

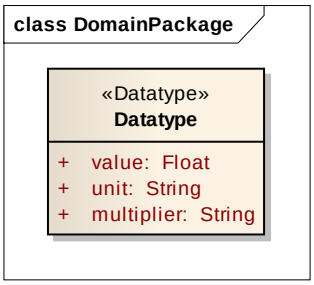
Domain Datatypes :

Domain datatypes are special datatypes that, alongside with the value space definition, gives the domain where the value space is meaningful, by adding some extra information to express this domain. CimDatatypes, Core and Business UN/Cefact datatypes are a good example of these datatypes.

Domain datatype defines:

- a primitive value space or a subset of it by some kind of restrictions, as does simple datatype,
- the value domain for this value space with some extra information expressed as supplementary attributes alongside with the “*value*” attribute (in case of CimDatatype) or “*Content*” attribute (in case of UN/Cefact datatypes).

Domain Datatypes are represented as a classifier with one attribute named “value”, and supplementary attributes. In CIM, those attributes are: unit, multiplier, denominatorUnit and denominatorMultiplier. Stereotype used for “*Domain datatypes*” is “*Datatype*” (or user defined with some prefix added to Datatype):



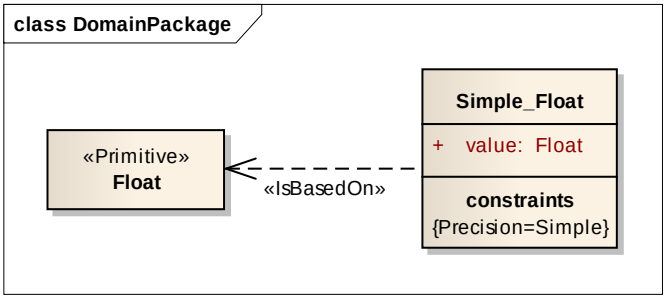
23.7. Primitive value space restrictions:

The value space of a <<Primitive>> can be restricted. Each <<Primitive>> value space has its own kind of allowed restrictions (or facets). The list of allowed facets has been defined : most of them are similar to the ones defined for XML datatypes, but there are some more.

Primitive Type	Constraint/facet
Boolean	no facet
String	String_kind (normalizedString, token, NMTOKEN, Name, NCName, anyURI, ID, IDREF)
	length
	minLength
	maxLength
	pattern
	whiteSpace
	enumeration
Integer	Integer_kind (long, int, short, byte, unsignedLong, unsignedInt, unsignedShort, unsignedByte)
	totalDigits
	minInclusive
	maxInclusive
	minExclusive
	maxExclusive
	enumeration
Float	precision (simple, double)
	minInclusive
	maxInclusive
	minExclusive
	maxExclusive
	enumeration
	pattern
Decimal	totalDigits
	fractionalDigits

	minInclusive
	maxInclusive
	minExclusive
	maxExclusive
	enumeration
TimePoint ***note this Primitive type is missing in CIM, here it is used for defining time-related facets***	timePoint (dateTime, date, time)
	minInclusive
	maxInclusive
	minExclusive
	maxExclusive
	enumeration
	pattern

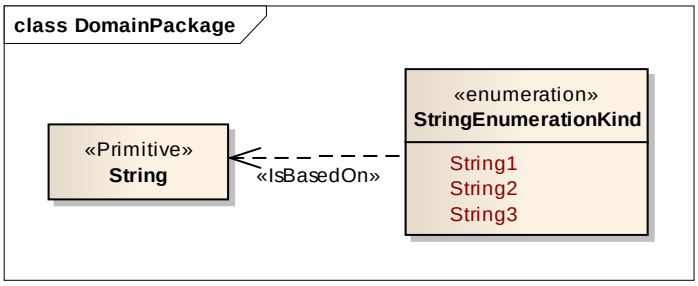
When the value space of a <<Primitive>> is restricted, it specifies a new value space that is represented by a <<Datatype>> that has just a value attribute. This <<Datatype>> has an "*IsBasedOn*" relationship with the <<Primitive>> (we say it is "*IsBasedOn*" the <<Primitive>>).



23.8. Enumeration value space :

The list of the enumerated literals of an <<enumeration>> defines the allowed values in a value space that is defined by another datatype (primitive, other enumeration or simple datatype) : so an <<enumeration>> has always an "*IsBasedOn*" relationship with a datatype (and at first with a primitive).

Example : an enumeration whose "*enumeratedLiterals*" are String :

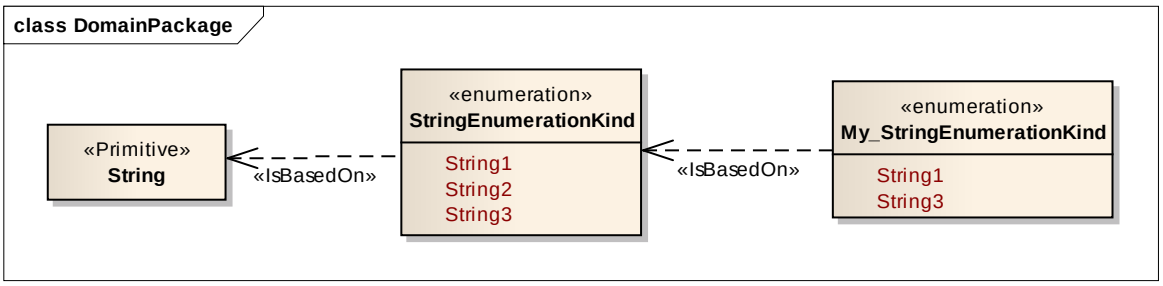


23.9. CIMDatatype "IsBasedOn" process :

As stated above, a simple <<CIMDatatype>>, whose value space is a subset of a <<Primitive>> has an "*IsBasedOn*" relationship with this primitive.

As stated above, all <<enumerations>> have an "*IsBasedOn*" relation with a <<Primitive>> (or a <<CIMDatatype>> or an enumeration that have themselves an "*IsBasedOn*" relationship to a <<Primitive>>).

An <<enumeration>> could be "*IsBasedOn*" on another one: in this case its enumerated literals are a subset of the upper level ones,



A <<CIMDatatype>> can be "*IsBasedOn*" on another one. All the rules that apply on the "*IsBasedOn*" class process are applying to "*IsBasedOn*" datatype process.

All the restrictions (or "*IsBasedOn*" process) could be done both at Information Model Level or Profile Level. Good practices are :

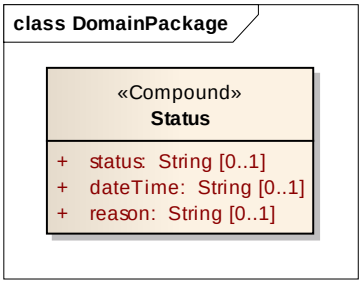
- To group all datatypes (Primitive, enumerations and CIMDatatypes) in a "Domain" Package both at Information Model level and Profile level,
- To put all <<Primitives>> at the Information Model level.

All this defines a hierarchy of "*IsBasedOn*" datatypes (Primitives, enumerations and CIMDatatypes).

Important : Primitive, enumeration, CIMDatatype (and Compound) names MUST be unique in the Information Model and Profile Model name space if there is a need to share all these elements.

23.10. Compound

Compounds are a special kind of attribute typing: they define a named group of attributes. They are stereotyped by the term "*Compound*". A Compound follows the same rules for naming as classes and follow the same rules as classes for the "*IsBasedOn*" process (see section 24 Annex C: profiling rules). The only difference with classes is that they cannot be instantiated as objects and that they do not have associations.



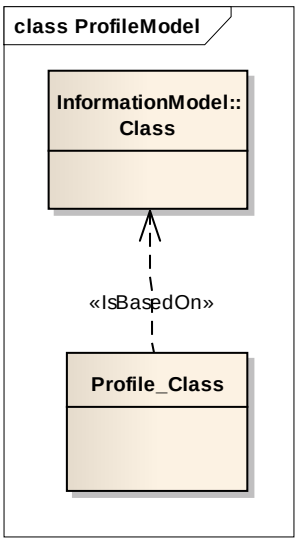
24. Annex C: Profile UML modelling rules

24.1. Profiling rules :

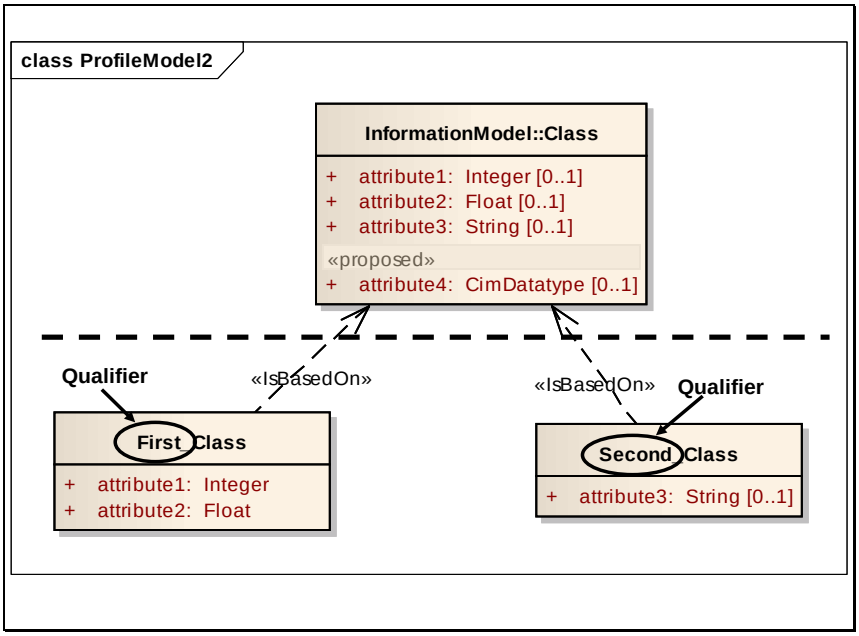
Profile is a subset of another UML Model. The upper model level is the Information Model. Nothing can be added at the profile level that is not already expressed at the upper level (usually the Information Model Level): no new classes, no new attributes and no new associations.

24.2. "IsBasedOn" for Profile Classes

A UML Profile Class is derived from the UML Information Model Class : the profile class is "***IsBasedOn***" ("IBO") on the upper level one. In *CimConteXtor*, "*IsBasedOn*" is marked as a "*Dependency*" association stereotyped with <<IsBasedOn>>.

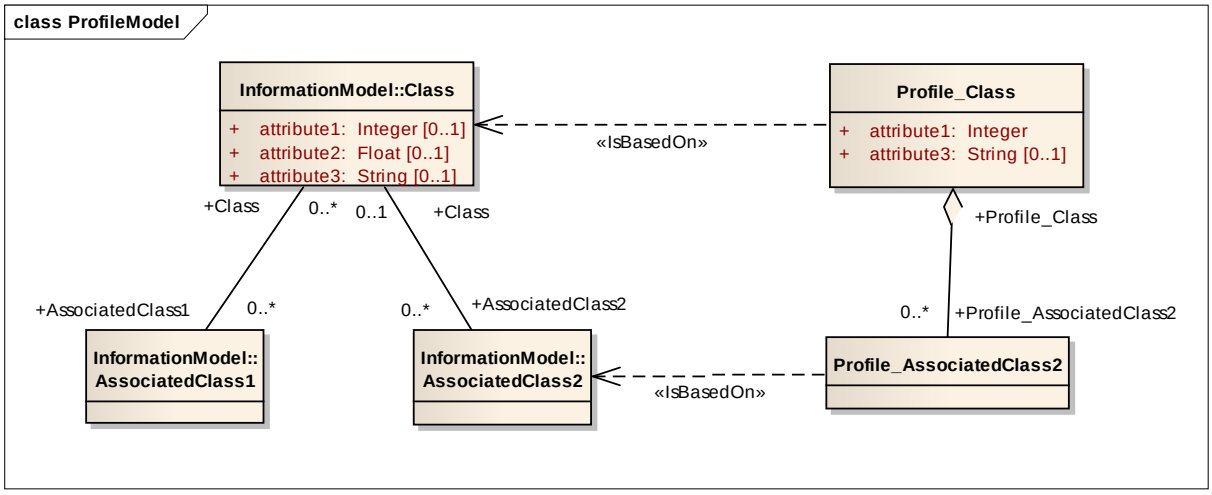


Several Profile classes can be based on the same information model class : in this case the names of these profile classes should be differentiated by the use of qualifiers (see naming rules) :



A Profile UML class ("*IsBasedOn*" class) can:

- have as attributes only the attributes that are defined in the class on which it is based,
- have as attributes all or a subset of the attributes that are defined in the class on which it is based,
- have as associations only the associations that are defined in the class on which it is based,
- have as associations all or a subset of the associations that are defined in the class on which it is based.



24.3. "IsBasedOn" for Profile class attributes :

A Profile class attribute can :

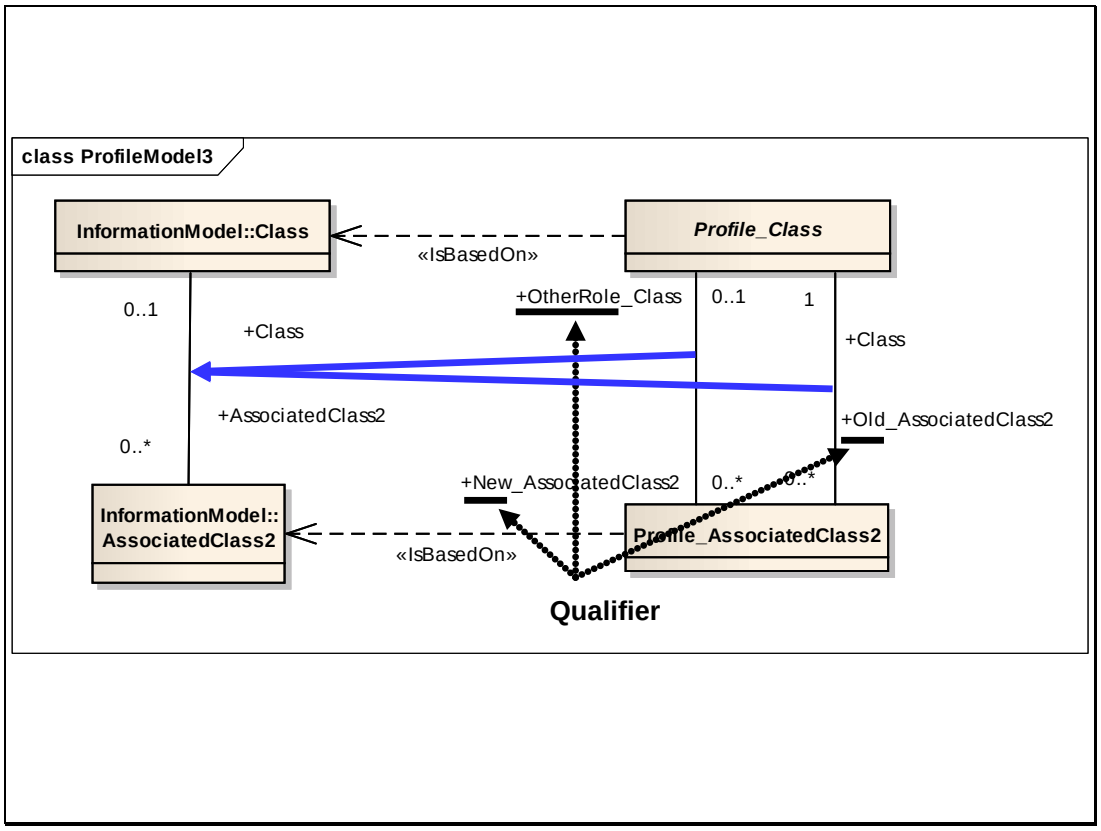
- have a cardinality that is the same or a more restricted one than that of the corresponding attribute in the information model class,
- have a type that is the same or a more restricted one than that of the corresponding attribute in the information model class (see Datatypes and Compounds section).

24.4. "IsBasedOn" for Profile Class associations :

A Profile class association could :

- have a cardinality that is the same or a more restricted one than that of the corresponding association in the information model class,
- have a type that is the same or a more restricted one than that of the corresponding association in the information model class, for example the association could become an aggregation,
- have role end names that are more refined or specialized than that of the corresponding association in the information model class,

Several Profile class associations can be based on the same information model class association : in this case the end role names of these profile associations should be differentiated by the use of qualifiers (see naming rules).



24.5. Profile naming rules

At the Profile level, the names of Classes, Attributes and Associations are the same as their corresponding counterpart in the information model level : but all classes and association names can be "*qualified*" in order to differentiate them when there are several classes or associations that are based on the same class or association of the information model.

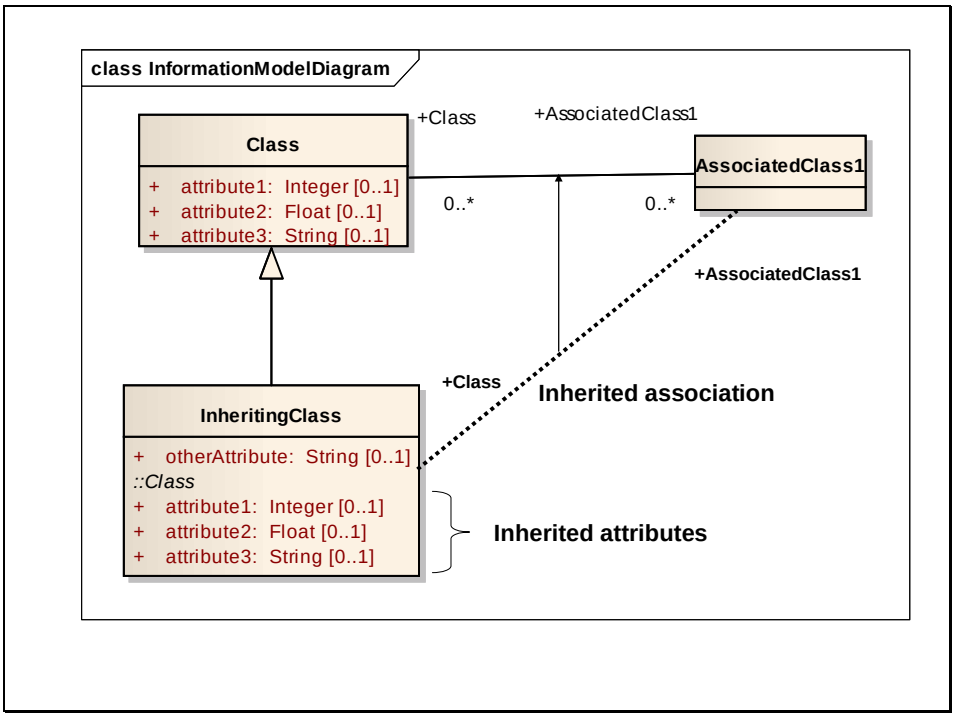
Qualifier:

- Qualifiers follow the same naming rules as class and association names,
- Qualifiers precede the class or association names and are separated from them by an "*underscore*" (_).

25. Annex D : Profile rules for inheritance

25.1. Inheritance principles:

A Class could inherit from another class (its "*super Class*"): it means that it inherits the super Class properties (attributes and associations). At the Information level all classes are concrete classes.



"*InheritingClass*"

- has its own attribute named "*otherAttribute*",
- inherits from "*Class*", so :
  - it inherits attributes named "*attribute1*", "*attribute2*" and "*attribute3*" from "*Class*",
  - and it inherits association with "*AssociatedClass1*", association whose end role names are "*Class*" and "*AssociatedClass1*".

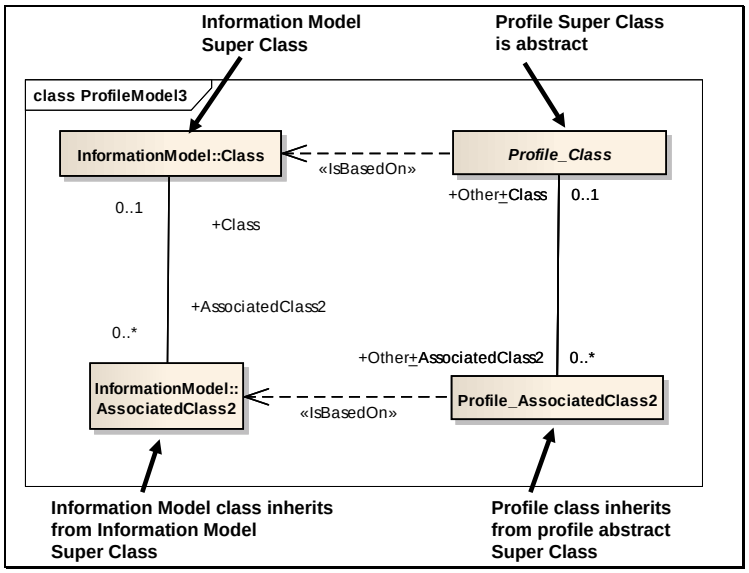
25.2. Profile level class inheritance rules:

At Profile level, a profile class can inherit from a profile superclass only if this profile superclass is ***abstract*** (in UML abstract classes have a name in italics).

Profile classes are still "*IsBasedOn*" on an Information Model level class, whether they are profile simple or super classes.

Profile classes can inherit from another Profile Class (its superclass) only if the two Information Level Classes that serve for the "*IsBasedOn*" process have themselves an inheritance relationship.

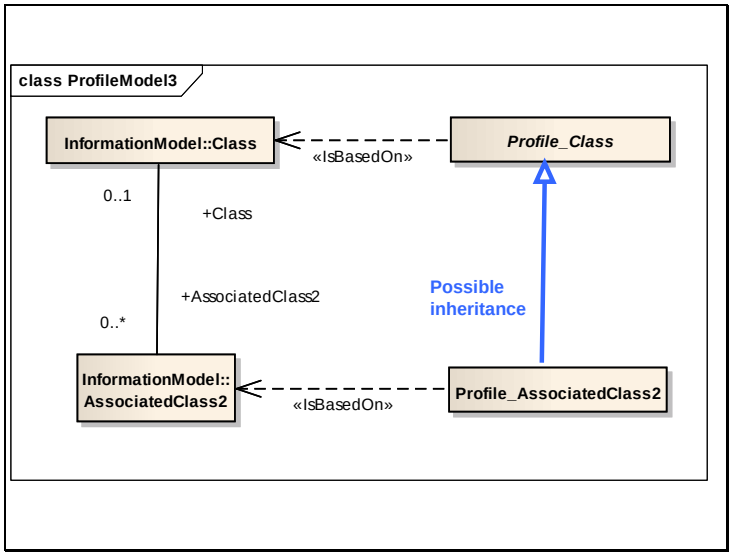
The inheritance rules between profile classes are the usual inheritance rules.



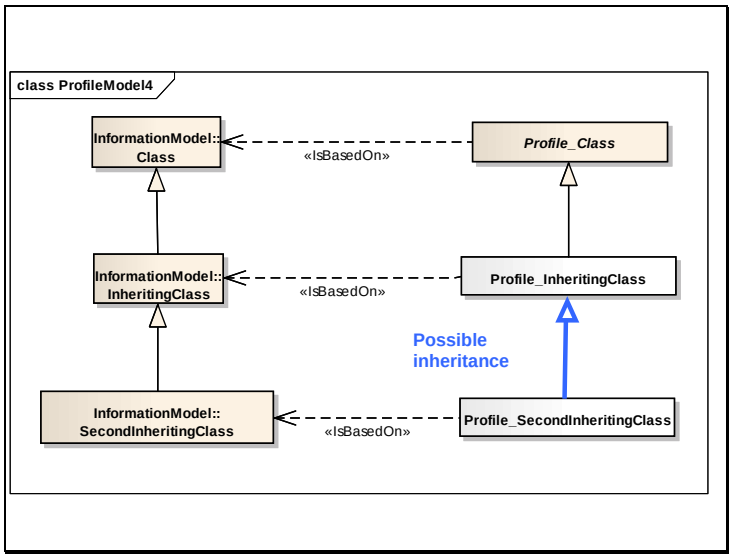
25.3. Profile level "IsBasedOn" class inheritance rules

At Profile level, there are several inheritance possibilities for a profile class. The profile class inherits from a profile superclass that:

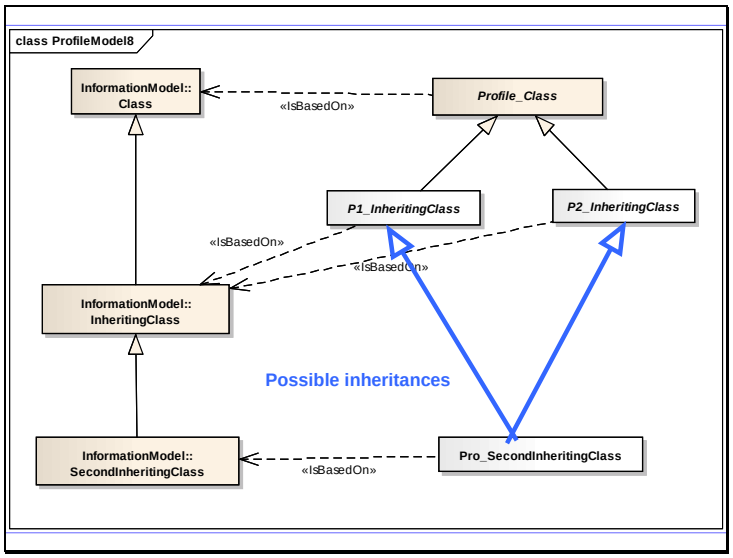
- stands by itself,



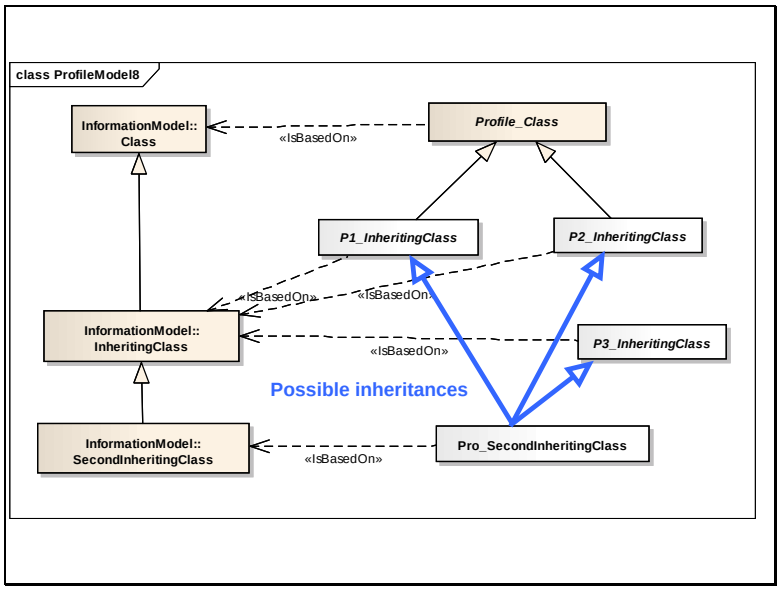
- is the last one of a profile inheritance hierarchy that mimics the information model level inheritance hierarchy,



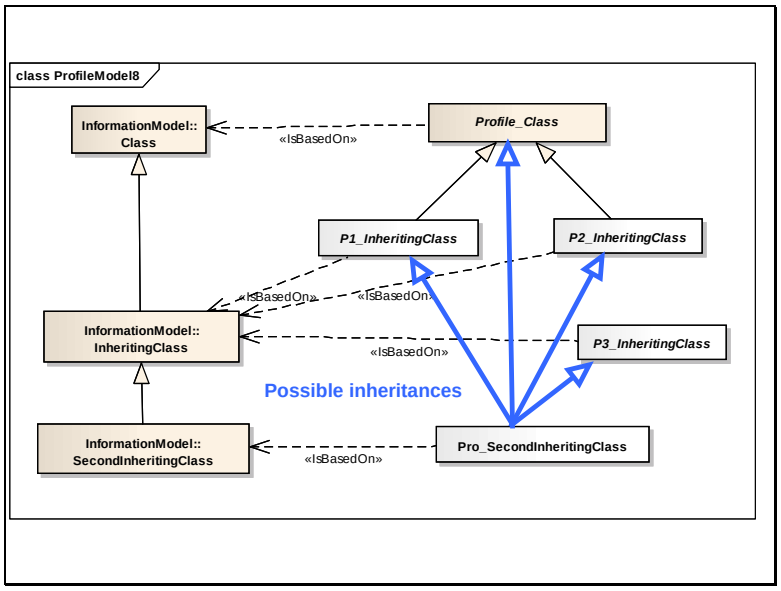
- is one of the profile super classes that are based on the same information model superclass,
    - first example



    - second example

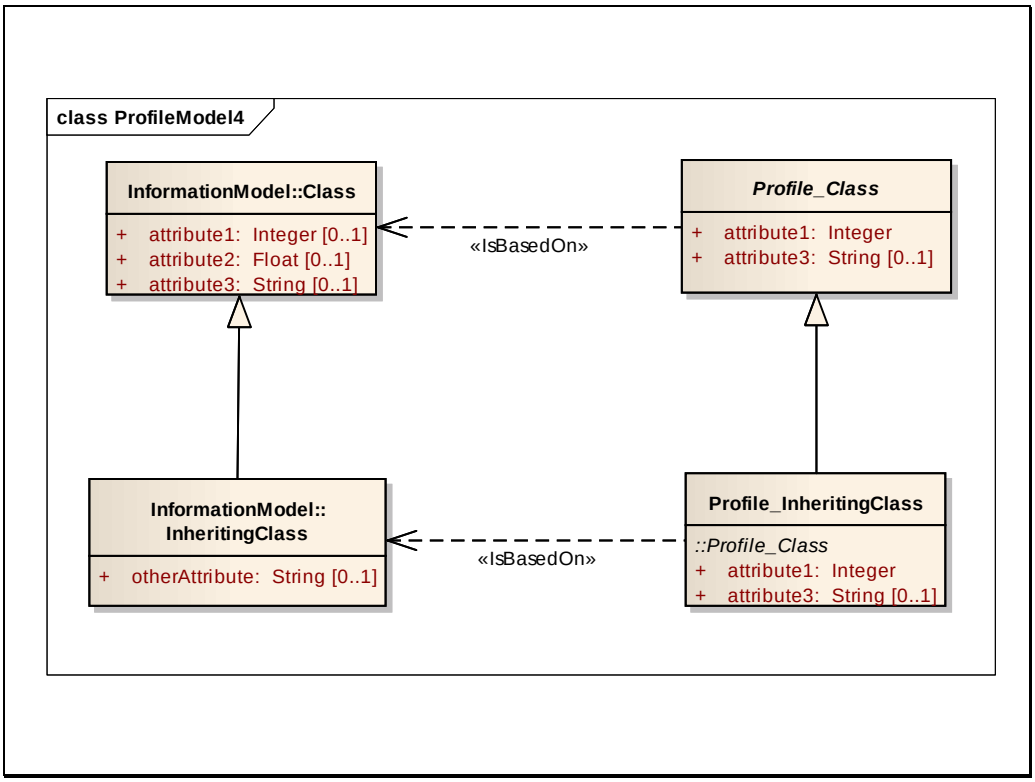


- Finally, it could be any of the profile super classes that are based on one of the information model level hierarchies :



25.4. Profile level attributes inheritance rules:

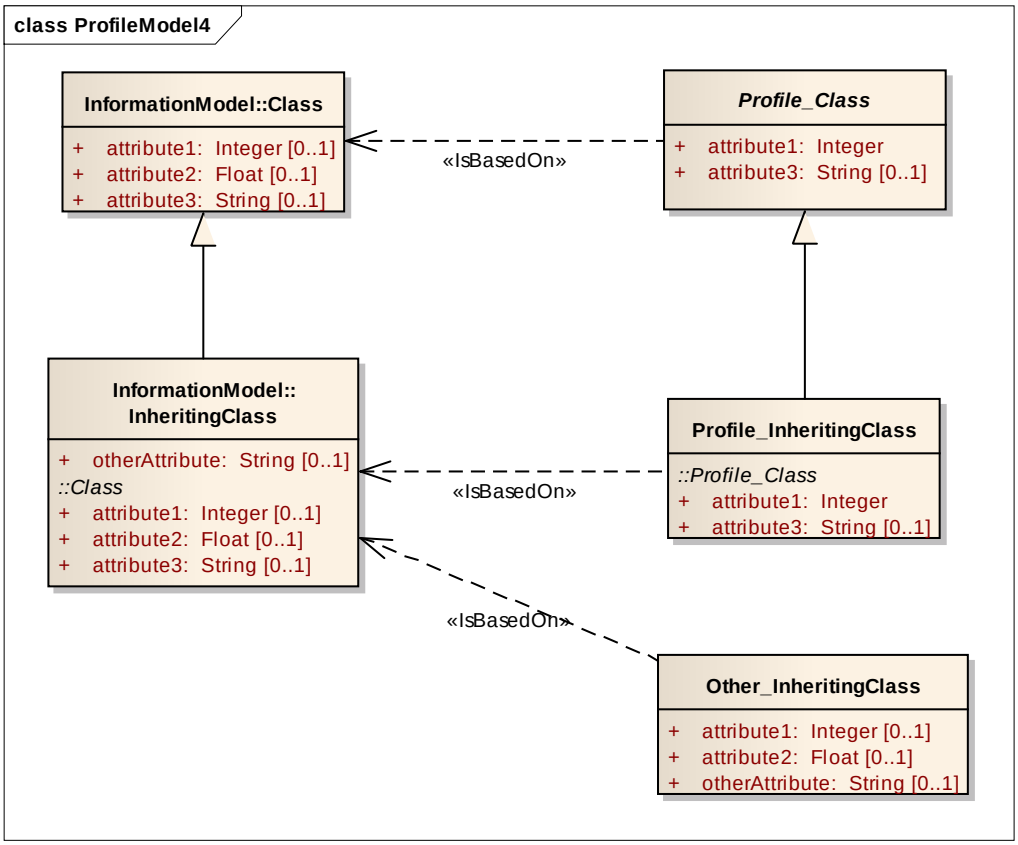
At profile level, the profile class attributes could be inherited from the profile superclass and could come from the Information Model Class on which it is "*IsBasedOn*".



"*ProfileClass*" is an Abstract Class : it is "*IsBasedOn*" on "*Class*" (from Information Model). "*Profile_InheritingClass*" is "*IsBasedOn*" on "*InheritingClass*" (from the Information Model) and inherits from "*Profile_Class*".

So at Profile level, there could be different cases of "*IsBasedOn*" classes, see the following example:

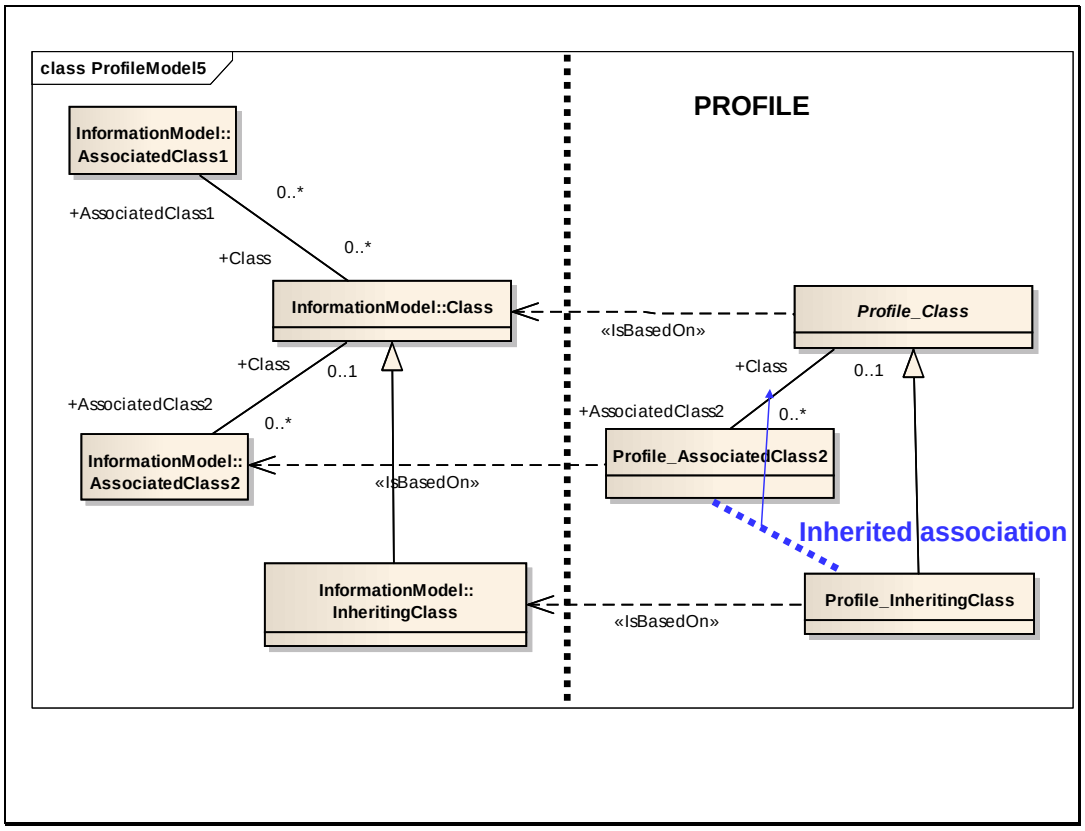
- a Profile Class named "*Profile_InheritingClass*" that is "*IsBasedOn*" the Information Model "*InheritingClass*" and inherits from the abstract Profile_Class named "*Profile_Class*",
- and a Profile Class named "*Other_InheritingClass*" that is "*IsBasedOn*" the same Information Model "*InheritingClass*".



25.5. Profile level association inheritance rules:

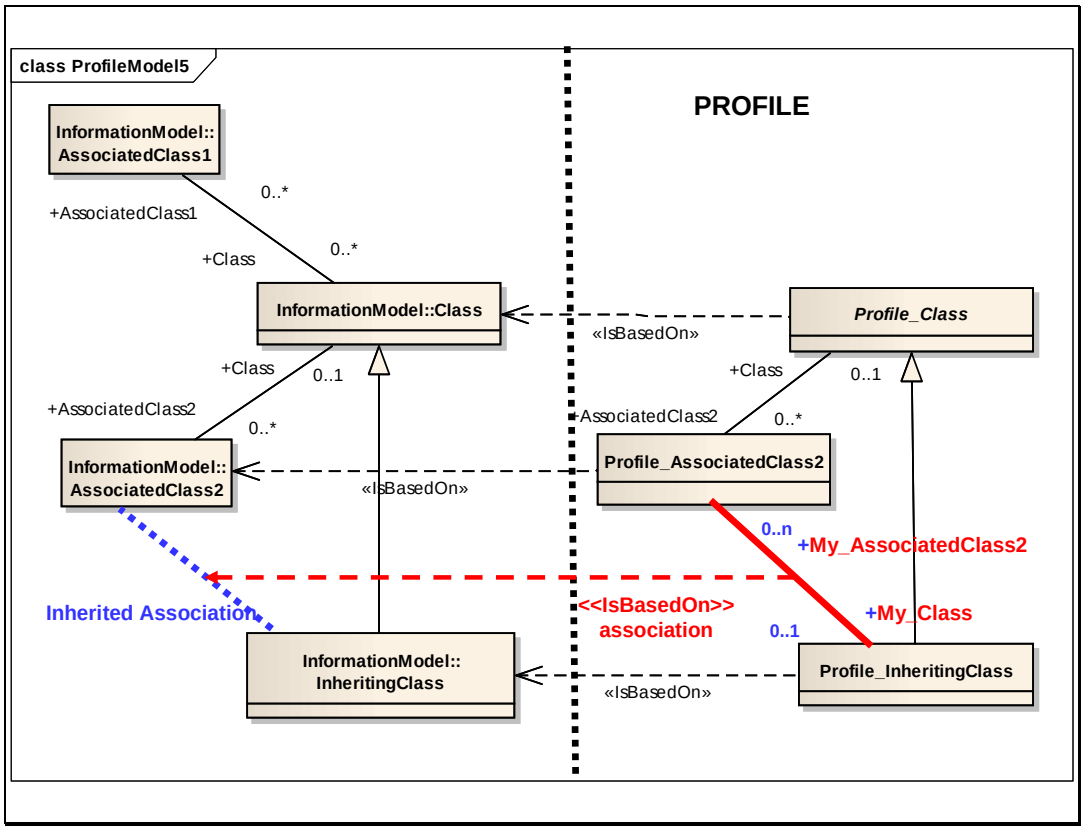
At profile level, the profile class associations could be inherited from the profile superclass and could come from the Information Model Class on which it is "*IsBasedOn*".

Example of inherited profile associations:



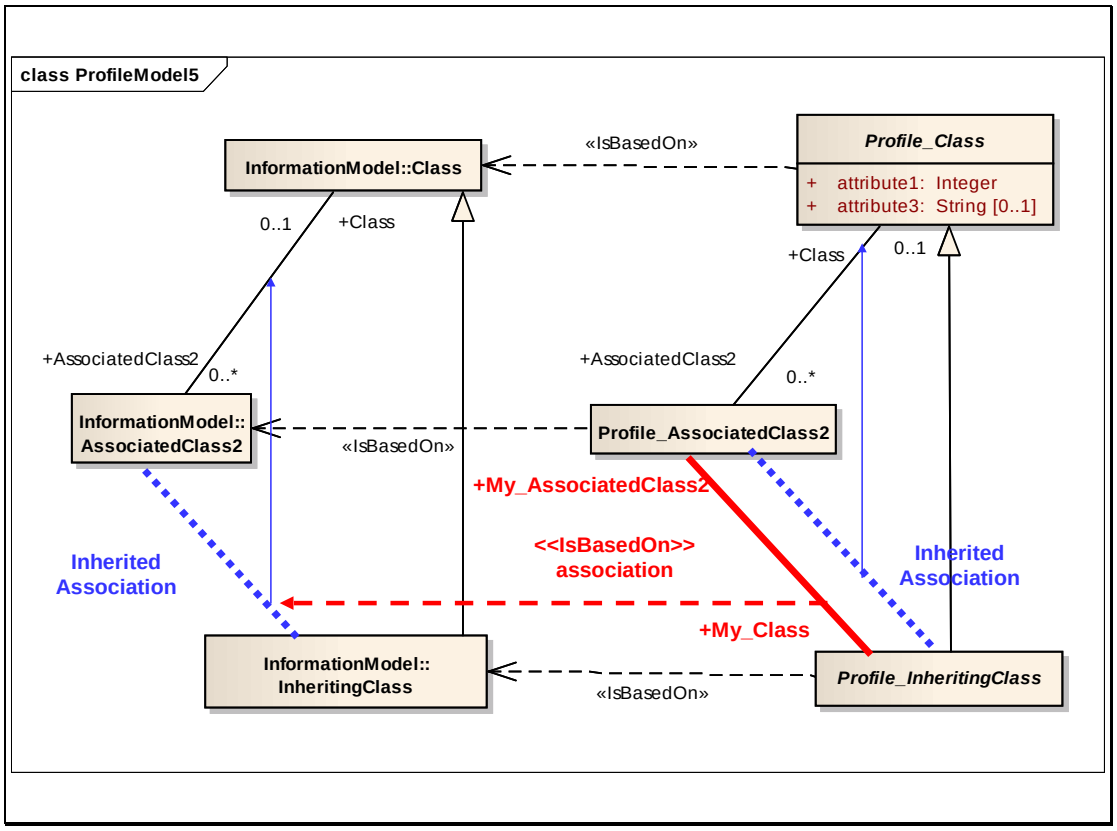
"*Profile_InheritingClass*" inherits from "*Profile_Class*" and thus inherits the association with "*Profile_AssociatedClass2*".

Example of an "*IsBasedOn*" association for the profile class that has a profile superclass :



"*Profile_InheritingClass*" is "*IsBasedOn*" on Information Model Class named "*InheritingClass*", thus it could have an association with "*Profile_AssociatedClass2*" because at Information Model level an association exists between "*InheritingClass*" and "*AssociatedClass2*".

So a profile class could have both "*Inherited*" and "*IsBasedOn*" associations. But one has to remember that all these associations should be differentiated by the use of appropriate "*Qualifiers*".



"*Profile_InheritingClass*"

- inherits an association with "*Profile_AssociatedClass2*". The end role names are "*Class*" and "*AssociatedClass*".
- has an association of its own with "*Profile_AssociatedClass2*". The end role names are "*My_Class*" and "*My_AssociatedClass*".

Note : the use of qualifiers like "My" to distinguish the associations by their end role names.

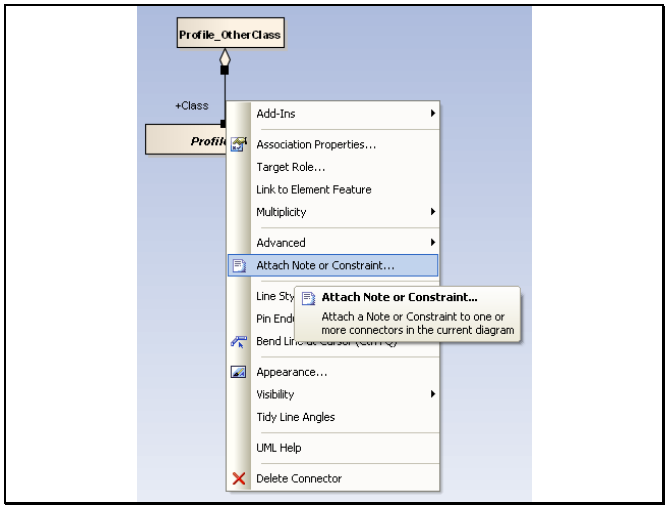
25.6. Use of Association CHOICE Constraint with inheritance rules

When a profile class has an association with a profile abstract class that has subclasses (inheriting classes): this means that this profile class has an association with each profile subclass. If we want to express that these associations are XORed or that at instance level a profile class can be associated with only one profile subclass, we must put a constraint somewhere.

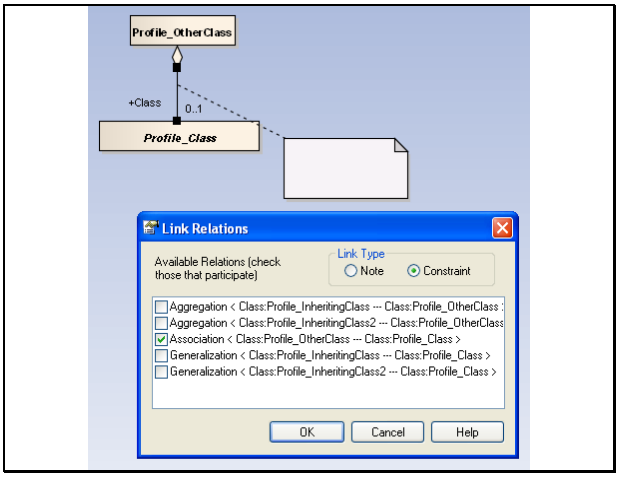
There are two ways to do that:

- Express the constraint on the association that will be inherited (i.e. between profile class and profile abstract class),
- Or express it on the inherited associations between profile class and profile subclasses.

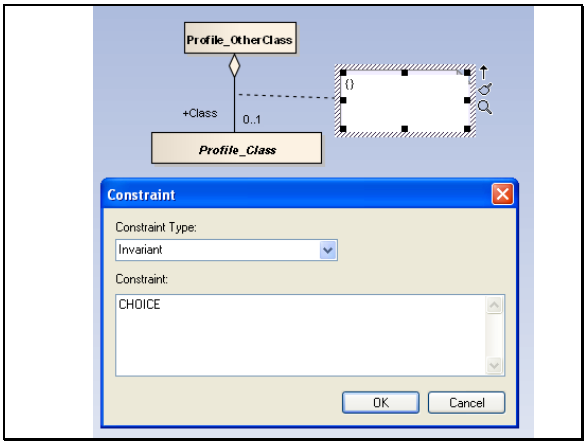
In the first case, a “CHOICE” constraint could be put on an association between the profile class and the profile abstract class, using the EA association “*Attach Note or Constraint*” Menu:



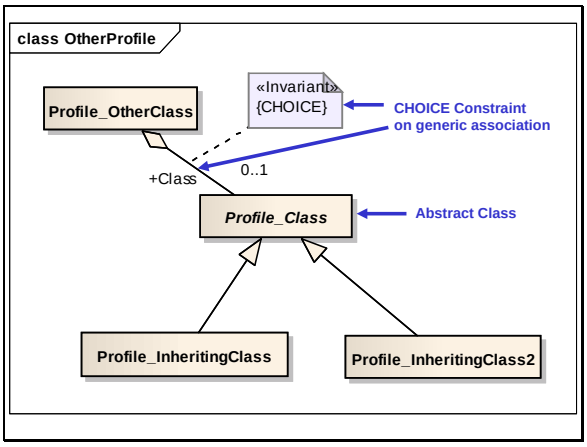
Select Constraint and association:



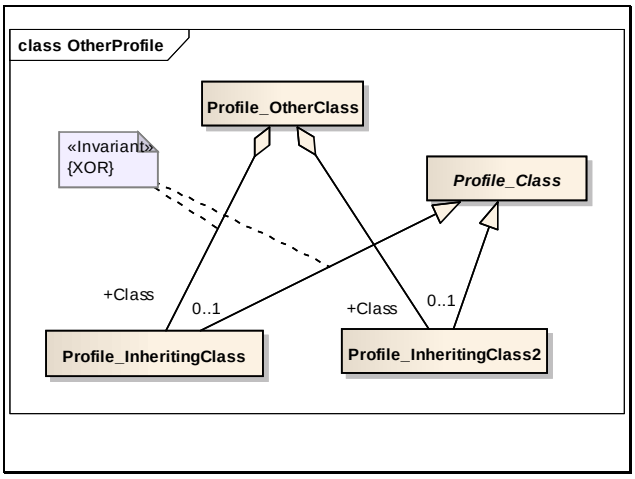
Enter “*invariant”* as constraint Type and “*CHOICE*” as constraint:



This will attach the “CHOICE” constraint to the association:



This is the same as putting a XOR constraint on the inherited association for the inheriting class:



WARNING : the attached XOR or CHOICE constraint note must be recreated each time the "*Edit connectors*" is reused.

26. Annex E: Information Model Extension

26.1. General

When extending Information Model with new packages:

- Use packages to create manageable and logical model pieces.
- Before creating a new package, do a search for the intended package name; if such a package already exists in the model, consider using another name or adding a prefix to the name of the new package.

When extending the information with new classes or associations:

- Avoid creating new primitive data types. Reason is that Information Model (CIM) is a semantic model that stays away from the extensive data typing available in many implementation technologies.

- Avoid creating synthetic domain objects that do not represent objects in the domain. Reason is that naming and identification of synthetic objects do not align with the problem domain and create extra administration.
- Before creating a new class, do a search for intended class name; if such a class already exists in the model, consider using another name.
- Create new domain object classes by sub classing an existing class, e.g. IdentifiedObject, Device, Schedule, etc. If it is not possible to subclass the object, an association between the two objects may be created. When adding an association, the information from the use case that motivated the addition is typically lost. A way to avoid this loss is to include the information in the documentation of the attribute or association end.
- When defining a new subclass, display on a diagram the superclass and its related classes with all their attributes to avoid inadvertent duplication: (a) either with the same attribute/association end names that get inherited, or (b) by duplicating an existing concept but with a different name.

26.2. Custom Information Model extensions

26.2.1. General

A custom extension is adding an element to the standard IM with the goal of keeping this element distinct from the IM elements, using another namespace.

The objective of these recommendations is to allow for clear modularization of extensions such that they can be readily identified as extensions and migrated more easily to updated versions of the standard IM, if needed,

Extensions to the IM model should be included in the extension package just as any other IM modelling except for the differences noted in this clause. The extension package should have a stereotype to identify all its contents as extensions to the IM information model. For example, the stereotype name could have an "Ext" suffix. The extension package should also introduce a new namespace and have a CIMVersion Class.

The sub-package structure of the extensions may pattern after that of standard IM packages, but the package structure should reflect the needs for individually managing and organizing parts of the extensions.

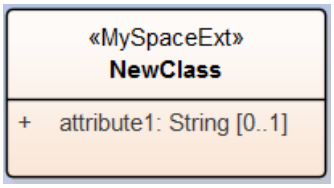
It is recommended to utilize existing data types and IM classes where possible. New classes can be created where new business objects concepts are introduced with their own lifecycle.

There are several cases for making an extension:

- Adding an extension class,
- Adding an extension attribute to an IM class,
- Adding an extension enumerated literal to an IM enumeration,
- Adding an extension association between an extension class and an IM class,
- Adding an extension association between two IM Classes.

26.2.2. Adding an extension class

Name the new extension class. Give the class the stereotype used for the specific extension (Example MySpaceExt). Add the class attributes.



Do not forget to give definition to the class and attributes.

26.2.3. Adding an extension attribute

In case of adding an extension attribute to an IM existing classes, a method of using multiple inheritance is allowed.

Name the new extension class with the same name as the standard IM class with the suffix "Ext". Stereotype it with the stereotype used for extension (Example MySpaceExt). Add the needed attribute with the stereotype used for extension (Example MySpaceExt).

Give the new class a generalization relationship from the standard IM class such that the standard IM class is a specialisation of the extension class. Generally, the extension class will have no base class and the standard IM class will now have multiple inheritance. The generalisation relationship should have the stereotype with the stereotype used for extension (Example MySpaceExt). Next figure shows an example of adding an extension attribute to an existing IM class:

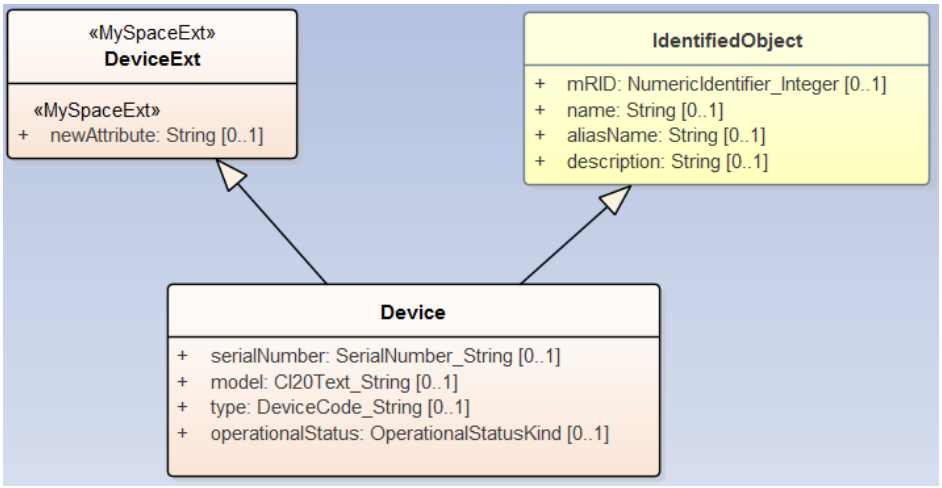


Figure 1: adding an extension attribute

26.2.4. Adding an extension enumerated literal

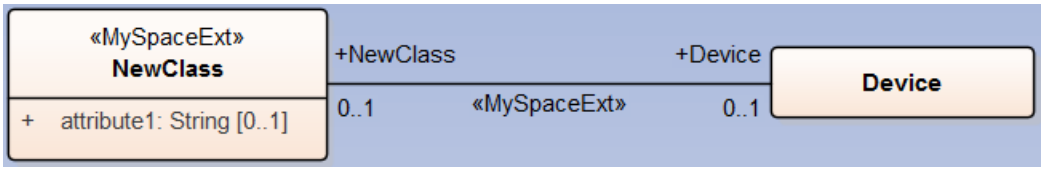
In case of adding an extension enumerated to an IM existing enumeration, the process is the same that one used for adding an attribute, except there will be no multiple inheritance because Enumerations do not inherit from another class.

Name the new extension enumeration with the same name as the standard IM enumeration with the suffix "Ext". Stereotype it with the stereotype used for extension (Example MySpaceExt). Add the needed enumerated literal with the stereotype used for extension (Example MySpaceExt).

Give the new enumeration a generalization relationship from the standard IM enumeration such that the standard IM enumeration is a specialisation of the extension enumeration. The generalisation relationship should have the stereotype used for extension (Example MySpaceExt).

26.2.5. Adding an extension association between extension and IM classes

The typical case of adding a new extension class and associating with existing IM classes can be modelled in a diagram within the extension package as shown in next figure. This situation does not require multiple inheritance.



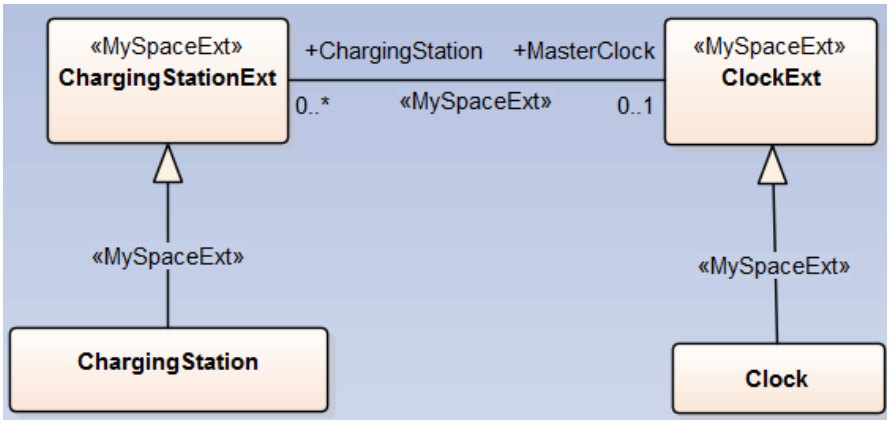
Note that the introduction of an association from a new extension class to an existing standard IM class requires nothing special, just make the association as it is implicitly owned by the depending extension class's package, but start to draw the association from the extension class. It is clearer to put both ends of the association into the extension package as the namespace for both association ends will follow IM package containment rules which rely upon package dependency.

26.2.6. Adding an extension association between two IM classes

In the case of a new association with both ends referencing existing classes, a method of using multiple inheritance is allowed.

Make two extension classes corresponding to the IM classes that should get a new association. Name the new extension classes with the name of the standard IM classes suffixed by Ext. Give the classes the stereotype used for the specific extension (Example MySpaceExt). Give the new classes a generalization relationship from the standard IM classes such that the standard IM classes are a specialisation of the extension classes. The generalisation relationships should have stereotype used for the specific extension (Example MySpaceExt).

The new association can be added to the new extension classes and they will be inherited by the standard M classes and in effect appearing as new properties of the standard classes. Next figure shows an example of adding an extension association between two IM classes:



An extension class should be used only once within a namespace. The extension class should have the extension generalization pointing to a standard IM class of the same name. A particular extension is an extension to the one and only standard IM class.

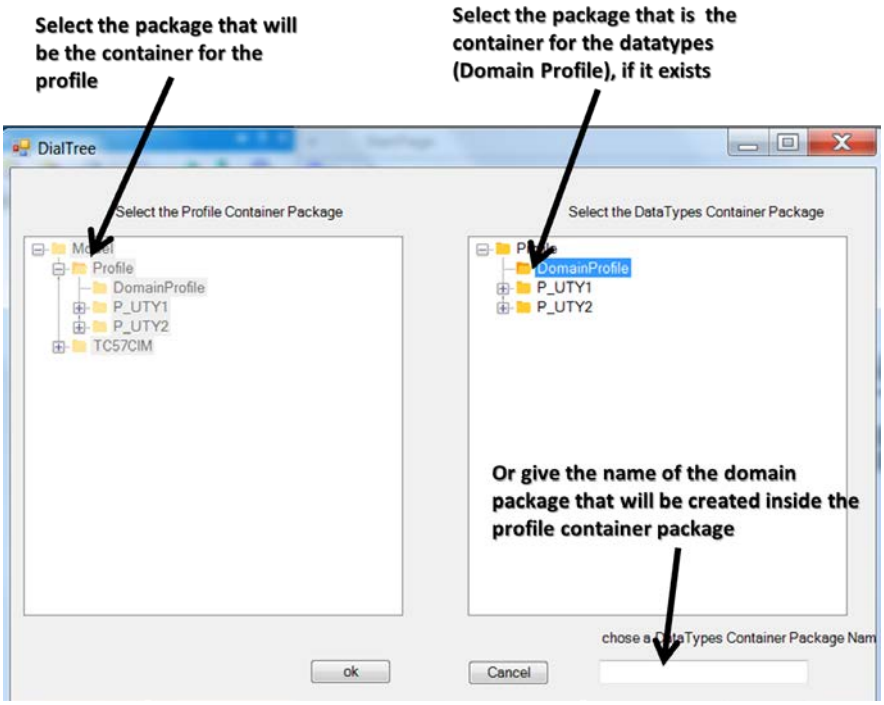
27. Annex F : CreateGlobalProfile description

"*CreateGlobalProfile*" is a feature to speed up profile building, like selecting an information model package and making a profile with all the classes of the package, plus the associated super classes. Same is done when starting from a diagram. These rules are profiling rules for profiles where you want to keep the inheritance path (example is WG13 or CGMES profiling style).

27.1. Starting CreateGlobalProfile

When selecting a package or diagram and launching *CreateGlobalProfile*, a pop up window will open:

- Asking to select the Package that will be the container for the intended profile,
- Asking to select the package that contains the datatypes, if it exists. If it does not exist, give the name of the package that will be created inside the Profile Container and will hold the profile datatypes.



Note: in this version there is only one package structure:

- *Profiles and Domain container package,*
- *Domain profile package (this means that the Domain profile is a global one and will be updated with new datatypes if a new profile is added with CreateGlobalProfile)*
- *Profiles packages.*

Note: in this version, by default the created profile package will be based on the Information Model packages: so check the right dependencies.

27.2. CreateGlobalProfile for a package

27.2.1. Package CreateGlobal Profile rules

The rules for package profiling are:

All selected package classes are copied in a profile package (including attributes), and the hierarchy of each class is replicated also in the profile. The name of the profile package is the name of the package prefixed by "P_" (Example package name is "TestUTY11", the profile package name will be "P_TestUTY11").

And:

- The profile class attribute keep the optional cardinality,
- The profile super classes are concrete,
- The profile super classes:

- o do not get their native attributes, if their associated CIM classes are not in the package that was selected for the CreateGlobalProfile,
- o get their native attributes, if their associated CIM classes are in the same package than the one that was selected for the CreateGlobalProfile.

For associations:

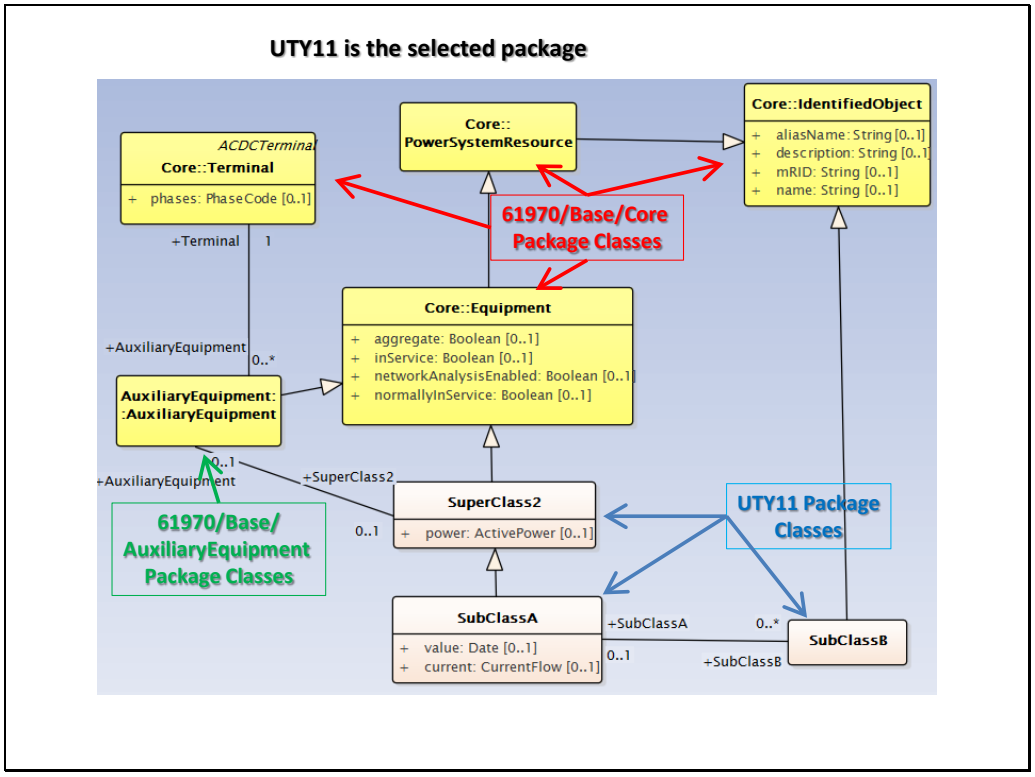
- associations between selected package classes are drawn without restrictions,
- association between a selected package class and a class outside the selected package, this outside class is put in a package of the profile, and the association is drawn without restrictions,
- all other associations are discarded,
- to refine association, you must use one of the Edit connector menu.

For diagrams:

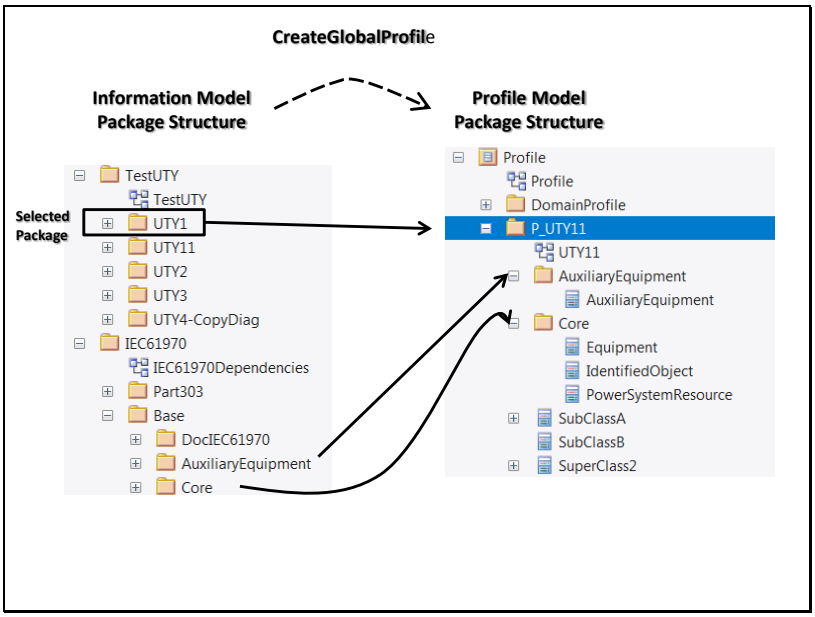
- Profile diagram is a duplicate of the selected package ones, with the above applied rules.

27.2.2. Example of rules for a package CreateGlobalProfile

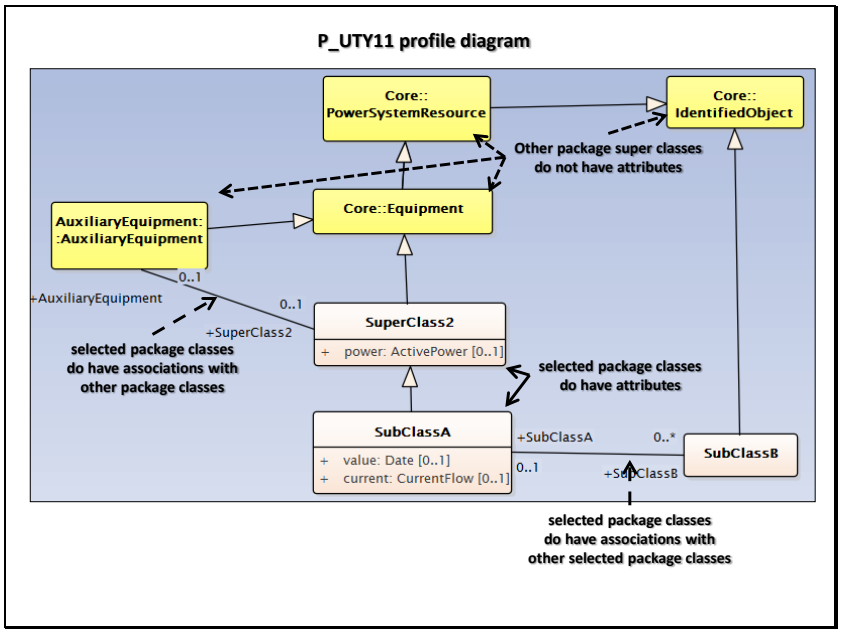
If we have a UTY11 package in the information model, that has some classes (SuperClass2, SubClassA and SubClassB): these are the selected package classes. These classes have relations with classes from other packages of the information model (Example: inheritance between SuperClass2 and Equipment or association between SubClass2 and AuxiliaryEquipment).



When launching the CreateGlobalProfile on the UTY11 package, a profile will created as P_UTY11 package in the profile container with appropriate sub packages (in the example Core and AuxiliaryEquipment):



And a profile diagram will show some of the profile elements:



27.3. CreateGlobalProfile for a diagram

27.3.1. Diagram CreateGlobalProfile rules

The rules for diagram profiling are:

All selected diagram classes are copied in a profile package: the name of the profile package is the name of the diagram prefixed by "P_" (Example diagram name is

"TestUTY", the profile package name will be "P_TestUTY"). Only classes from the selected diagram will be used in the profile.

And:

- The profile class (or super class) keep their attributes with the cardinality they have in the selected diagram,
- The profile classes keep the status they have in the selected diagram: concrete or abstract,

For associations:

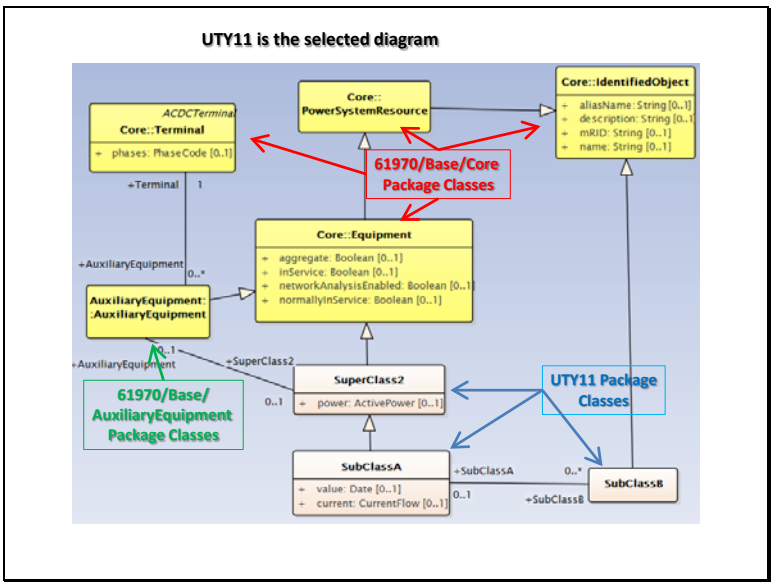
- associations between diagram classes are kept in the profile,
- association between a selected diagram class and a class outside the selected diagram, is discarded,
- to refine association, you must use one of the Edit connector menu.

For diagrams:

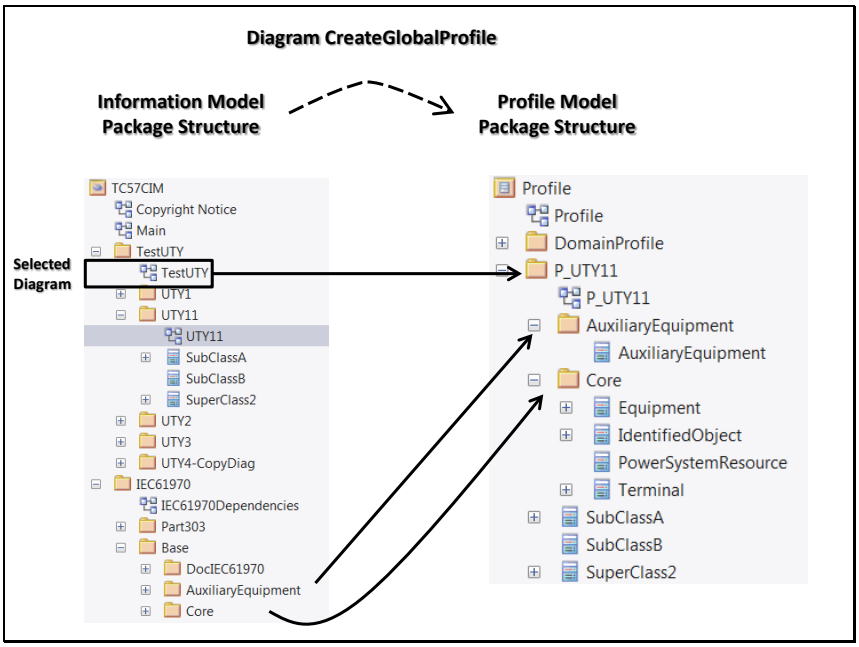
- Profile diagram classes are a duplicate of the selected diagram ones, with the above applied rules.

27.3.2. Example of results:

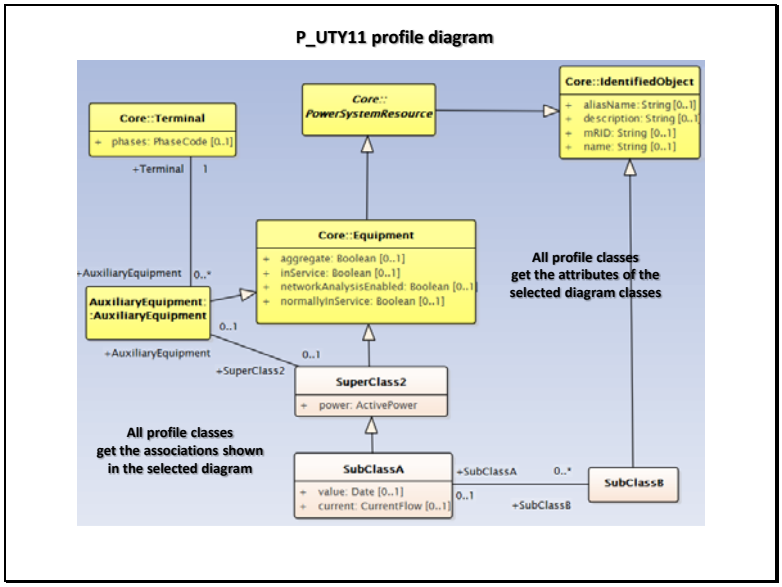
If we have a UTY11 diagram in the information model that displays some classes either from the package that contains the diagram (SuperClass2, SubClassA and SubClassB) or from another packages (IdentifiedObject, PowerSystemResource, Equipment, Terminal from a Core package or AuxiliaryEquipment from AuxiliaryEquipment package): these are the selected diagram classes. These classes have properties like attributes and relations shown in the diagram. All these properties will be kept in the profile.



When launching the CreateGlobalProfile on the UTY11 diagram, a profile will created as P_UTY11 package in the profile container with appropriate sub packages (in the example Core and AuxiliaryEquipment):



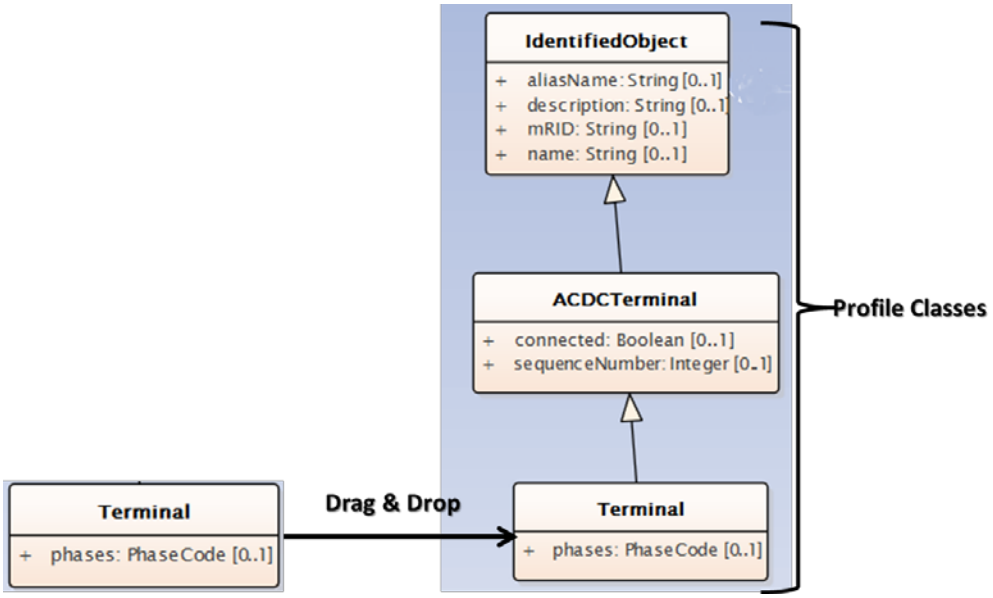
And a profile diagram will show what is in the profile:



28. Annex H: Enhanced Drag and Drop for IsBasedOn classes

This kind of feature is used for example by WG13 or CGMES profiling style. This feature is activated by the checking of the option "WG13AutomaticAnscesterInProfile" in the configuration file or in the option menu.

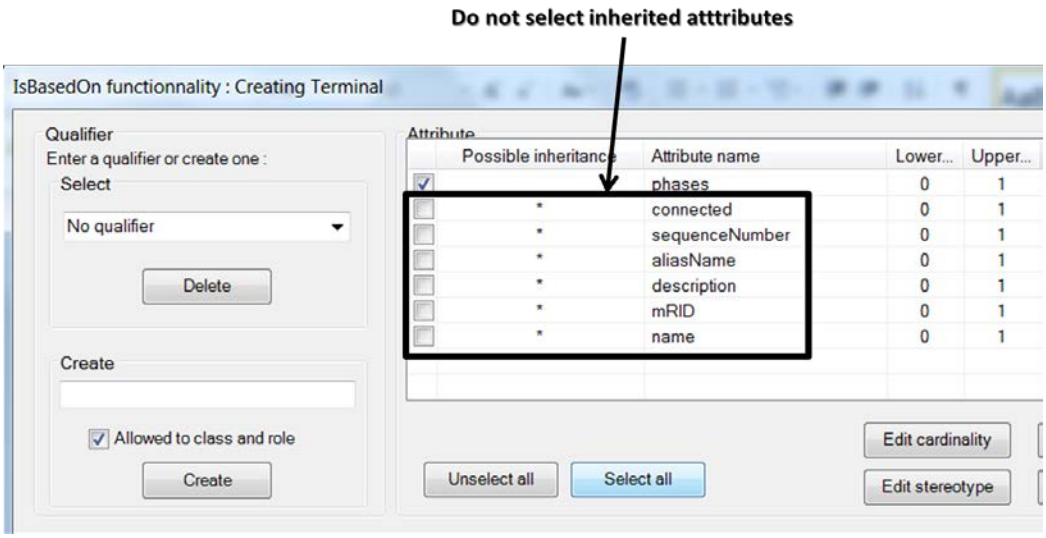
The main difference with the basic drag and drop function is that, at the end of drag and drop operation, the profile class is created with all its super classes. Example: drag and drop of Terminal Class will lead to have three profile classes: Terminal, inheriting from ACDCTerminal class, inheriting from IdentifiedObject.



Note: in this version of CimContextor:

- The super classes are not abstract, so with the EditIsBasedOn feature they should be made abstract,
- Each super class is coming with its attributes,
- Classes are created in the same package,
- The result is different from the one you get with CreateGlobalProfile.

Apart from the fact that at the end of the process the profile class is created with its super class(es), the procedure to specify the drag and drop profile is the same as the one described in section 10, except that you should not select inherited attributes (as shown here after):



Note: you will speed up your design if you always drag and drop the leaf classes first.

29. Annex G : Adapting profile to a new CIM version

When a new version of CIM is outputted, profiles based on a previous version might be impacted and aligned with some elements of the new version. Here are some proposals and procedure to handle changes:

- Export profiles package and extension package as xmi files,
- Import those files in the eap that has the new CIM version (because of the IsBasedOn features, all the relations between profiles and information model will be restored),
- Check with the excel sheet that comes with the new CIM version what are the changes and if these changes impact the profiles (may be all changes will not be reported, a utility like Integrity check could also be used, but the current version need some upgrade to check everything). Changes could be:
    - Class name change: use EA class properties to change profile class name manually,
    - Attribute name change: use EA class attribute properties to change profile class attribute name manually,
    - Notes change: use CopyAllNotes for CGMES profile (For EntsoE ESMP, alignment is forbidden, open question for other profiles),
    - Deleted class: delete profile class manually,
    - Deleted attribute: delete profile class attribute manually,
    - Deleted association: delete profile association manually, (or use EditConnector),
    - Added class, update profile if necessary using Drag & Drop feature and EditConnector,
    - Added Attribute, update profile if necessary using EditIsBasedOn,
    - Added Association, update profile if necessary using EditConnector.
    - Change or update of TaggedValues are done with EditIsBasedOn,
    - Change or update of other items like Constraint, initial value, stereotype must be done manually for classes
    - Change or update of CIMDatatype supplementary attribute fixed initial value is done using EditIsBasedOn on the profile CIMDatatype. (note: for default value, it must be done manually with EditIsBasedOn,
    - Attribute type change (TBD)
    - CIMDatatype change: (TBD).

A future version of CimConteXtor will offer some solutions for these issues.

The *CimConteXtor* user guide has been updated under the governance of ENTSO-E from a previous version written by Zamiren.

For any issue or comment, send mail to:

contact@zamiren.fr or FSD@energinet.dk

Or go to the Entso-E website

https://www.entsoe.eu/