INSPIRE

Infrastructure for Spatial Information in Europe

Consolidation Team

Connecting to the public INSPIRE UML repository using Enterprise Architect

Title	Connecting to the public INSPIRE UML repository using Enterprise Architect
Creator	M. Lutz
Date (last change)	2011-02-11
Subject	Instructions on how to connect to the public INSPIRE UML repository from Enterprise Architect
Publisher	JRC CT
Type	Text
Description	Instructions on how to connect to the public INSPIRE UML repository from Enterprise Architect
Contributor	JRC CT
Format	PDF
Source	None
Rights	Public
Identifier	UML_repository_public_branch_instructions.docx
Language	En
Relation	n/a
Coverage	Project duration

These are Dublin Core metadata elements. See for more details and examples http://www.dublincore.org/.

Version history:

Version number	Date	Modified by	Comments
0.1	2011-02-11	MLU	Created document based on UML & EA training document

Table of contents

Preface

This document is a short tutorial on how to connect to the public branch of the consolidated UML repository developed as part of the INSPIRE data specification development.

It is assumed that the reader is familiar with UML modelling and using Enterprise Architect and Subversion. Annex I gives a short introduction to subversion. Annex II introduces the UML profile used in INSPIRE and some of the naming and documenting conventions used.

The screenshots in the text are based on EA version 7.5 on Windows XP.

1 Getting started with EA and Subversion

EA has integrated functionality for working with a version control system. However, since an EA stores its models in a binary, Microsoft Access, file, it cannot simply be uploaded to a version control repository. Thus, the handling of setting up a repository and checking out/in versions is not completely straightforward. Since the EA file itself is binary, EA will allow import/export of each package in XML Metadata Interchange format (which is an ASCII-file) that will then be checked out/in of EA. This means that the packages themselves are version controlled but the EA file is not. You may also import several packages from several different version control systems in one EA file or simply have some packages which are version controlled and some which are not.

In INSPIRE, subversion (SVN) is used as a version control software. For a brief introduction to basic SVN operations, see Annex A.

NOTE EA requires a command line executable SVN client tool (such as the *CollabNet Subversion Command-line Client*[1]) to be installed on your computer. *TortoiseSVN*[2] does *not* provide such a command line tool.

2 Setting up Subversion on your computer

Figure 2.1 gives a schematic overview of the procedure required to set up EA with the INSPIRE UML repository. During this step, a local copy has to be created on the client machine using a SVN check-out (steps 1 and 2). The checked out XMI files are then imported into an EA project (steps 3 and 4).

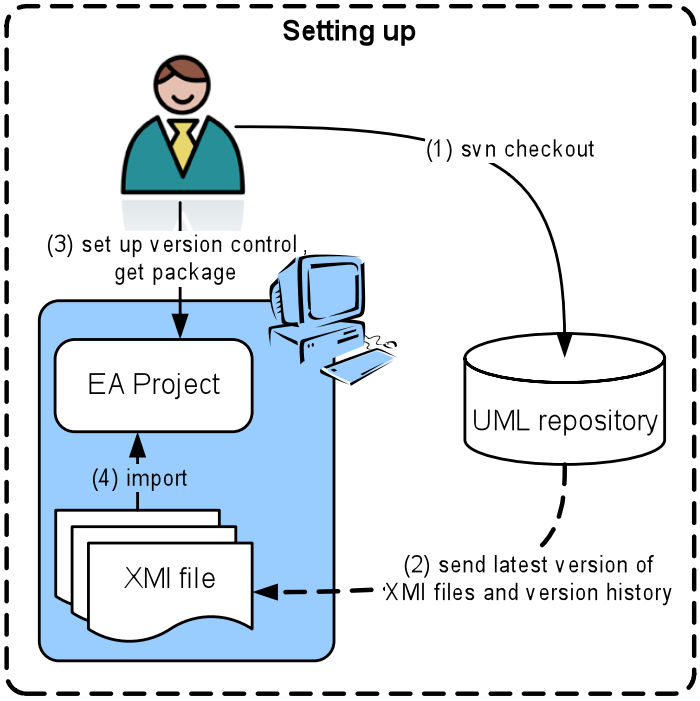


Figure 2.1: Schematic overview of setting up EA with the UML repository.

[1] Available at http://www.collab.net/downloads/subversion/
[2] Available at http://tortoisesvn.tigris.org/

The public repository of the INSPIRE UML application schemas is available at https://inspire-twg.jrc.it/svn/inspire-model/branches/public. Several of the INSPIRE application schemas refer to types defined in ISO TC 211 models. The ISO models are not included in our INSPIRE repository. They are located in a separate repository: https://inspire-twg.jrc.it/svn/iso/ (access is restricted – an open access mirror is available at https://www.seegrid.csiro.au/mirrors/iso-harmonized-model/).

First, you need to create local copies of these two repositories on your file system. Creating the local copy is done by an SVN checkout from the existing repository.

To check out the **INSPIRE model**, issue the following command in the directory where you would like to store the local copy of the repository.

```
svn checkout https://inspire-twg.jrc.ec.europa.eu/svn/inspire-
model/branches/public --username <username>
```

If you have never connected to the `inspire-twg.jrc.ec.europa.eu` server, the output for this command should be the following:

```
Error validating server certificate for 'https://inspire-
twg.jrc.ec.europa.eu:443':
 - The certificate is not issued by a trusted authority. Use the
   fingerprint to validate the certificate manually!
Certificate information:
 - Hostname: inspire-twg.jrc.ec.europa.eu
 - Valid: from Mon, 23 Mar 2009 08:17:38 GMT until Thu, 21 Mar 2019 08:17:38
GMT
 - Issuer: SDI, JRC, Ispra, Varese, IT
 - Fingerprint: 4e:d4:dd:63:95:ee:ff:6a:12:f4:18:26:0e:3d:6e:7f:8d:0f:6d:21
(R)eject, accept (t)emporarily or accept (p)ermanently?
```

Press `p` for accepting and storing the certificate permanently on your computer and then enter your password.

NOTE SVN complains that the certificate is not from a trusted source because our certificate is self-signed.

The check out command will create a directory named `publuc` in the folder where you issued the command containing all subdirectories of the repository.

To check out the **ISO TC211 model**, issue the following command in the directory where you would like to store the local copy of the repository:

```
svn checkout https://www.seegrid.csiro.au/mirrors/iso-harmonized-model/
```

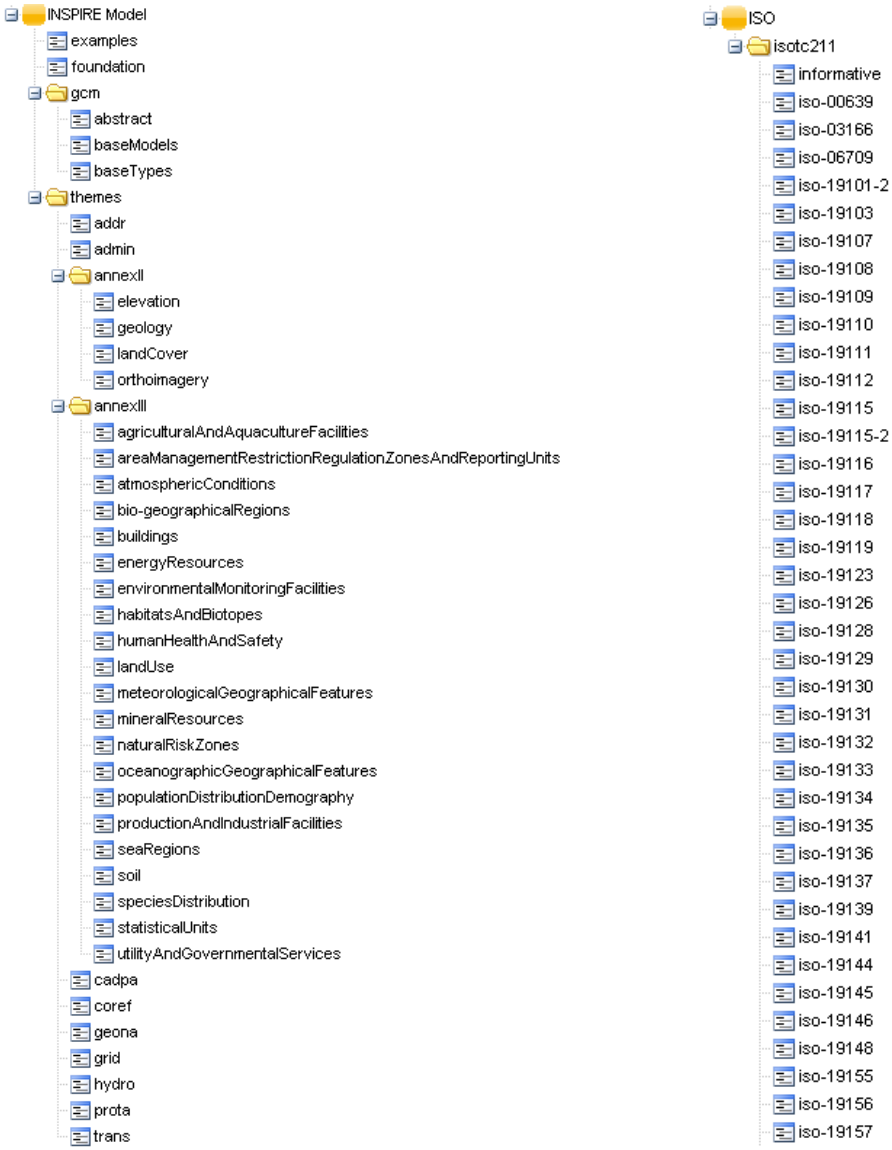
The check out command will create a directory named `iso-harmonized-model` in the folder where you issued the command containing all subdirectories of the repository.

NOTE Since the command line utility is required by EA in any case, it is used here for illustration. However, alternatively the above procedure can be done by using a graphical user interface for

subversion such as the *TortoiseSVN*. In this case it is important that the SVN client used supports the same SVN version (1.5.x or 1.6.x – the third digit can be different) as the command line utility (otherwise you risk receiving the following error message: *"This client is too old to work with working copy"*).

3 Repository structure

The structure of the repositories is the following:



The INSPIRE model contains a directory for the foundation packages (`/foundation`), which currently only contains the ISO TC211 19100 models. Furthermore, a directory for the generic conceptual model exists (`/gcm`) with three subdirectories: `abstract`, `baseModels`, `baseTypes`. The `abstract` directory contains nothing for the moment but is intended for abstract models of INSPIRE. The `baseModels` directory shall contain any cross-themes models, e.g., the Generic Network Model. The `baseTypes` directory is intended for all cross-themes data types. Finally, a `/themes` directory exists, which contains all the 34 INSPIRE annex I, II and III themes.

4 Creating the Structure in EA

Now that your system is connected with the repository, you can set up EA to work with it. When this is done you can import each package contained in the repository into EA. It is important to mention that EA considers a file in the repository ending with `.xml` containing XMI a UML package. To import packages from the repository you need to have an EA project file – it can be empty (newly created) or it can be an existing file already containing model/packages. EA does not care whether it imports packages from a version control system in an EA file already containing models.

4.1 Connecting EA to the repository

EA can work with a number of different repositories. It handles all connections giving them separate unique IDs. It is IMPORTANT that all users **use the same identifier** for a given repository (subset). If different identifiers are used with packages that contain version-controlled sub-packages, these sub-packages will lose their connection to the repository. Therefore, you have to use the following identifiers:

- *inspire-model* for the repository with the INSPIRE application schemas
- *isotc211* for the ISO repository

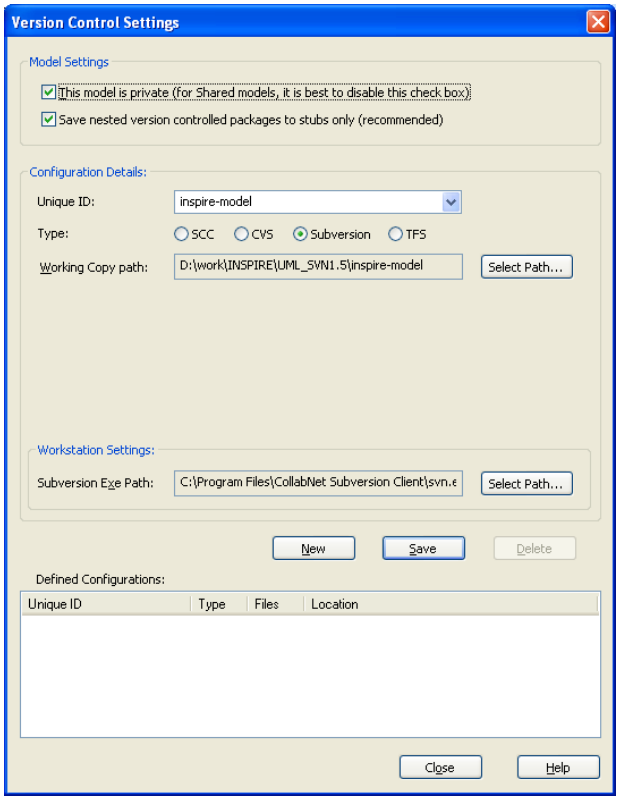


Figure 4.1: Creating an identifier called for the INSPIRE repository.

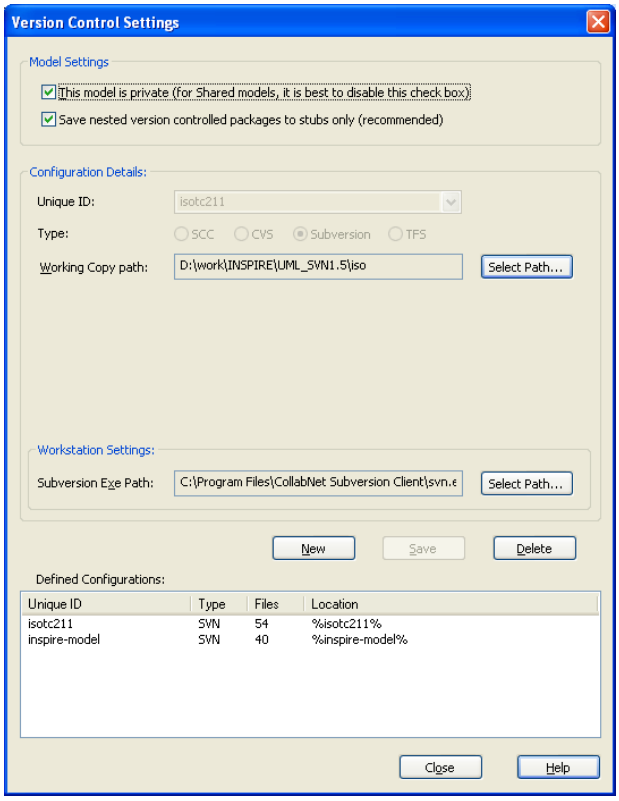


Figure 4.2: Final version control settings.

Working with a repository and EA can follow certain schemes. The main ones are private and shared models. We will follow the private model scheme which has several advantages to the shared one. The shared model scheme is highly dependent on 100% server up-time and high speed internet connection. The private model scheme allows for flexibility and that each editor works with his own EA file. Settings

related to the EA projects connection to version control systems are found in PROJECT – VERSION CONTROL – VERSION CONTROL SETTING (Figure 4.1).

Here, you need to specify the Unique ID and the type of repository (select *Subversion*). You also need tp specify the path to which you have checked out the repository, e.g. `D:\work\INSPIRE\UML_SVN\public` in the example shown in Figure 4.1. Finally, you need to specify the path to the SVN executable `svn.exe`, e.g. `C:\Program Files\CollabNet Subversion Client\svn.exe` in the example.

NOTE The path for the isotc211 model is `<some path>\iso-harmonized-model`, *not* `<some path>\ iso-harmonized-model\isotc211`.

After adding the identifier for the INSPIRE repository, repeat these steps for the ISO repository. The final settings are shown in Figure 4.2.

4.2 Manually importing packages

One way to import the packages existing in the repository is to import each one separately. To do this, right-click in the *Project Browser* on the package into which you want to import the package. It could be the top view. Then select PACKAGE CONTROL – GET PACKAGE as shown in Figure 4.3.

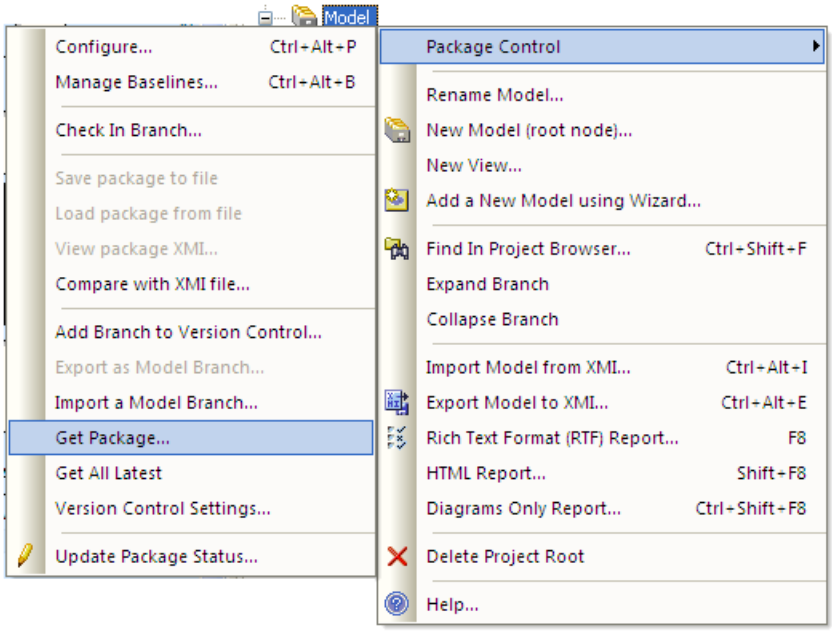


Figure 4.3: Get package.

This will open a dialog as shown in Figure Figure 4.4. Select the identifier for the version control system: (*inspire-model* or *isotc211*), which will then show all UML packages available in the repository. Unfortunately, in this dialogue it is only possible to select one package at a time, so that this procedure is rather cumbersome. On the other hand, it allows you to have all packages imported in a structure that you decide yourself. You could, e.g., import all theme packages in the root of your model.

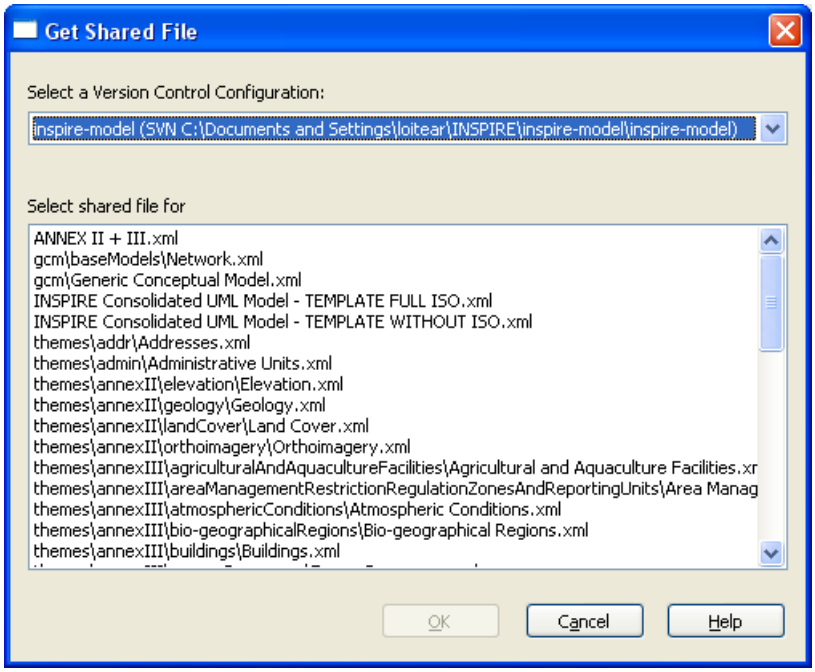


Figure 4.4: Select package to import.

4.3 Automatically importing all packages

If you want to import all INSPIRE application schemas, you should import one of the following "template" files containing the whole structure in EA:

- `INSPIRE_with_ISO.xml` (in the *inspire-model* repository)

  This template contains all INSPIRE themes packages and all the ISO models, i.e. a rather large number of packages. So the import will take some time (around 5-10 minutes) – but you only have to do this once. The advantage is that you have all the ISO models and can refer to them in your models.

- `INSPIRE_without_ISO.xml` (in the *inspire-model* repository)

  This template contains all INSPIRE themes packages, but none of the ISO packages. This means that you have to import those ISO packages (standards) that you need in your model work separately (either manually as described in section 4.2 or using the template `ISO TC211.xml` – see below).

  This template is appropriate if you already have included the ISO svn repository in your project or if you need to use only few ISO standards and want to keep your EA model small.

  NOTE Some ISO standards have more than one package. These typically represent different versions of the standard (e.g. ISO 19115:2003 and ISO:19115:2006 Corrigendum). When importing standards manually, make sure that you include the latest version of a particular standard.

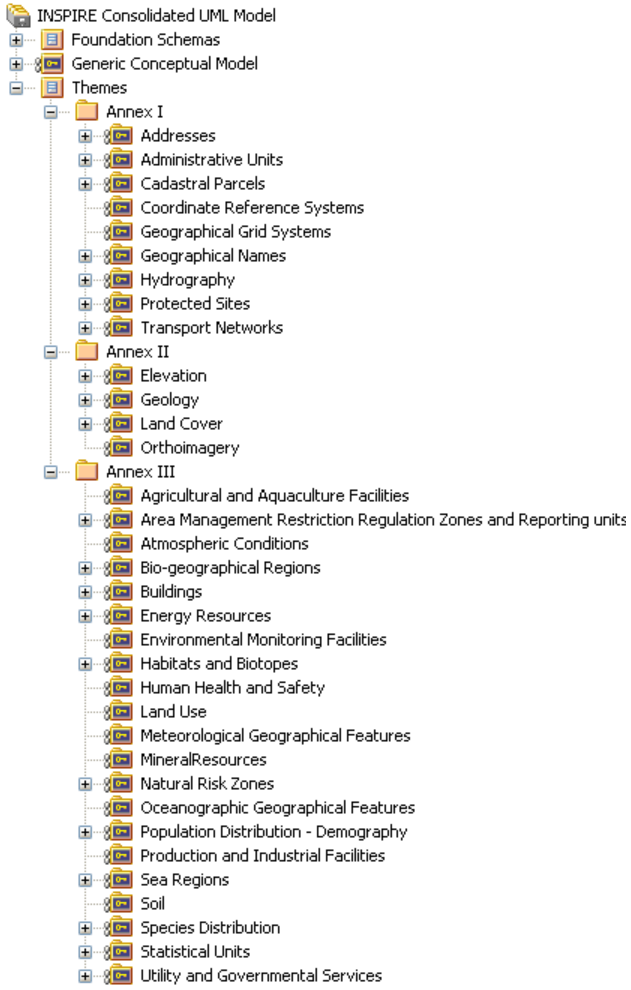
- `ISO TC211.xml` (in the *isotc211* repository)

  This template contains all the ISO models. You can use this template to include the ISO models if you have previously imported only the INSPIRE themes packages. To add all the ISO models, go to

a package in your model, choose PACKAGE CONTROL – GET PACKAGE from the *isotc211* repository and select the `ISO TC211.xml` package.

After importing one or several of the templates, all sub-packages can be retrieved by right-clicking on a package in the *Project Browser* and selecting PACKAGE CONTROL – GET ALL LATEST in the context menu. Figure 4.5 shows the Project Browser after this step if you imported the `INSPIRE_with_ISO.xml` template (with the INSPIRE packages are expanded on the left, and all ISO packages expanded on the right).

INSPIRE



ISO

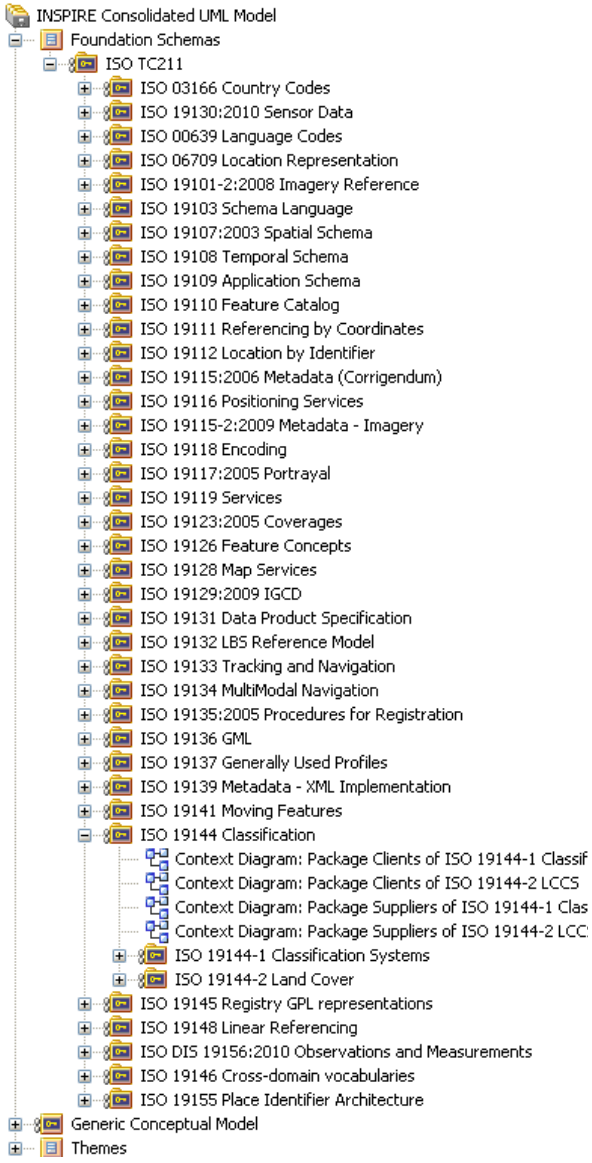


Figure 4.5: Project browser after importing all packages.

To see changes that have been made to the version-controlled models, right-click on a version-controlled package in the *Project Browser* and select PACKAGE CONTROL – GET LATEST (to update only this package) or PACKAGE CONTROL – GET ALL LATEST (to update all version-controlled packages in the EA project – this may take some time!) in the context menu.

Annnex A: Working with Subversion

SVN can be used to maintain current and historical versions of files such as source code, web pages, and documentation. Its goal is to be a mostly-compatible successor to the widely used Concurrent Versions System (CVS). In the following the most basic SVN operations are described. For a comprehensive overview see the “Subversion book” at http://svnbook.red-bean.com/.

The core of a SVN system is a central repository that stores all the files and their revisions (as versions are called in SVN). In the case of the modular data specifications template, this repository is located at https://inspire-twg.jrc.ec.europa.eu/svn/ds_twg/. You can browse the repository content by simply using a web browser[3].

It is important to remember that SVN stores all historical versions of a file. So even if a file is deleted at one point, it is still possible to retrieve the latest (or indeed any previous) version of that file from the repository. It is also possible to revert to previous versions, e.g. in order to undo a change that was committed by accident.

A.1 Basic operations

The first thing you need to do (and if all goes well you need to do this only once) is to create a local copy of the contents of the SVN repository on your local machine (Figure 4.6). This is done using the `svn checkout` command. As a result, the latest revision (r114 in the example shown in Figure 4.6) of all files in the repository will be copied to the selected folder on your local machine.

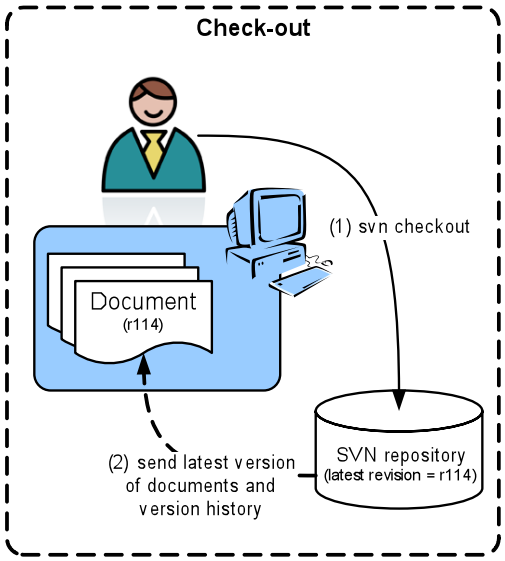


Figure 4.6: Creating a copy of the repository content on your local machine using svn checkout.

Once you have created a local copy, you can modify files and synchronize them with the repository using the `svn update` and `svn commit` commands (Figure 4.7). With `svn update` you make sure that your local copy is up-to-date. As a response to this command, all files that have been added, modified or deleted by somebody else in the meantime will be updated in your local copy. It is important to update before editing a file because working on an out-of-date copy can lead to conflicts when committing a file (see below). In the example shown in Figure 4.7, the local working copy is updated to revision r123.

[3] A slightly more user-friendly interface is available at https://inspire-twg.jrc.ec.europa.eu/voilaSVN/ (select DS_TWG as a repository).

With `svn commit` you send modified files back to the repository. You can apply a commit to just one file or several files or folders at once. Through this command your local changes (new, modified or deleted files) will be propagated to the repository, and a new revision (r124 in the example) will be created.

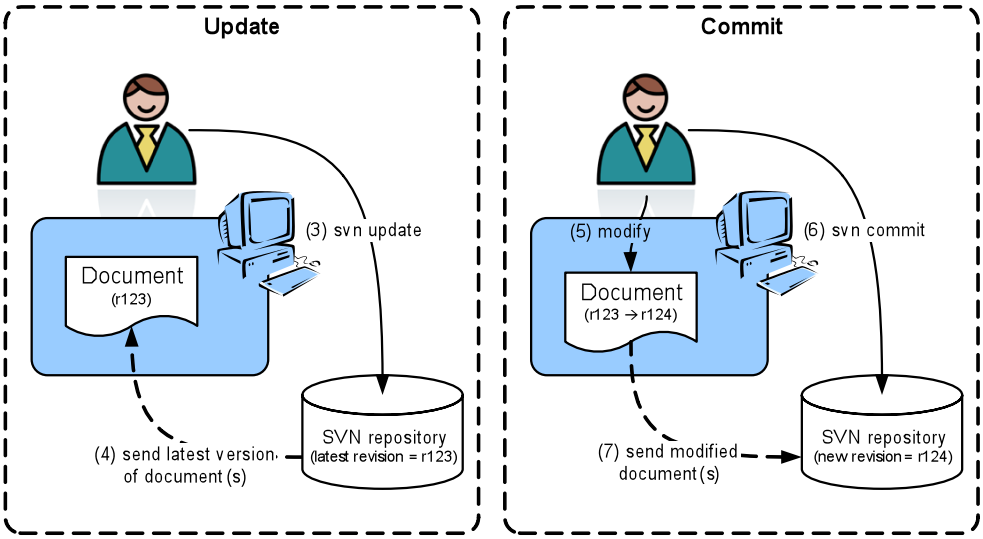


Figure 4.7: Creating a copy of the repository content on your local machine using svn checkout.

A.2 Concurrent editing and dealing with conflicts

Concurrent editing () can lead to conflicts. In the example shown in Figures 3 and 4, two users are working on the same revision (r123) of a document (Figure 4.8). User 1 submits his edits first (steps 5-7 in Figure 4.9), and a new revision (r124) is created. User 2's document is now out-of-date with respect to the repository, and when he tries to commit his changes, a conflict occurs[4] and the commit fails.

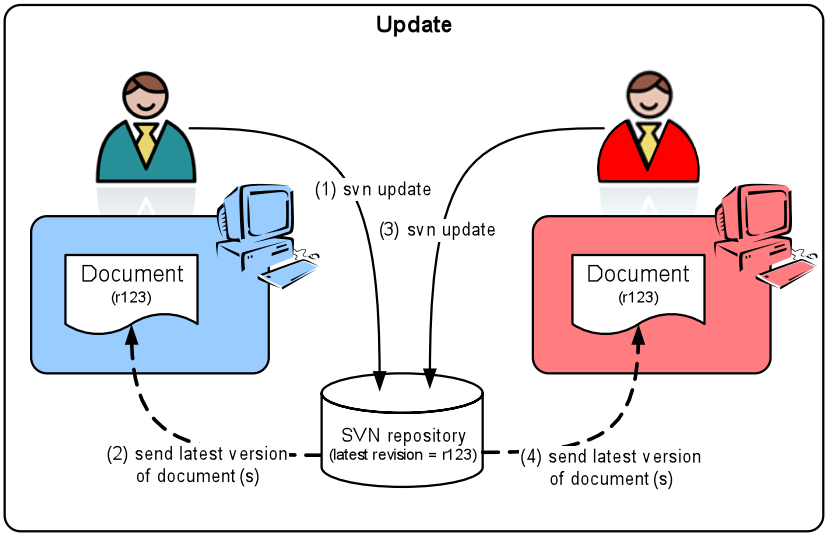


Figure 4.8: Two users are concurrently updating a document.

[4] Note that for text files, SVN is able to merge concurrent edits of the same file, and conflicts only occur when the same parts of the text file are edited concurrently. For binary files like Word documents, however, concurrent editing will always lead to a conflict.

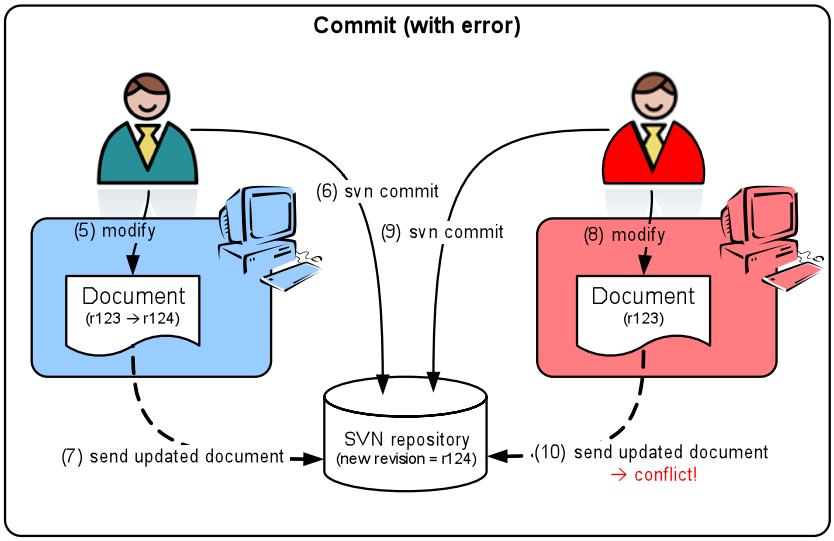


Figure 4.9: Two users are concurrently modifying a document and a conflict occurs when the second user tries to commit his modifications.

In such a case, SVN will create 3 files in the local copy of user 2:

(1) the latest revision of the document in the repository (r124 in the example)
(2) the revision of the document that was created in the local copy of user 2 by the last update (r123)
(3) the current document in the local copy of user 2 (with all local changes)

Usually, it is possible to resolve the conflict by merging (using Word's "Compare and Merge Documents" functionality) documents (1) and (3).

Once the conflict has been resolved the merged file can be committed to the repository, and a new revision (r125) is created.

Annnex B: UML modelling conventions in INSPIRE

This section gives a short introduction on conventions on how to document UML models in INSPIRE.

Convention: INSPIRE modelling conventions are highlighted like this.

B.1 The UML INSPIRE profile

For INSPIRE, a number of specific stereotypes (e.g. «featureType» and «voidable») have been defined in a UML profile. The profile (current version 1.1) is available as an XML document in the `uml_profile` folder of the INSPIRE svn repository.

In order to use this profile in EA, import the XML document by selecting IMPORT PROFILE from the *UML Profiles* folder in the *Resources* tab (Figure 4.10). In the following dialogue, keep all the default settings. After the import, the *UML Profiles* folder should look like shown in Figure 4.11.

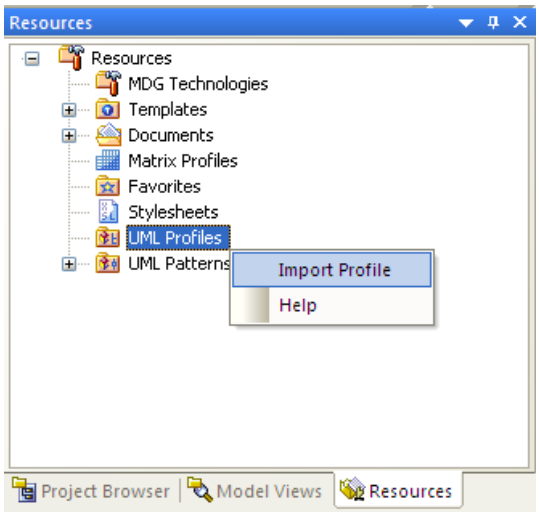


Figure 4.10: Importing a profile.

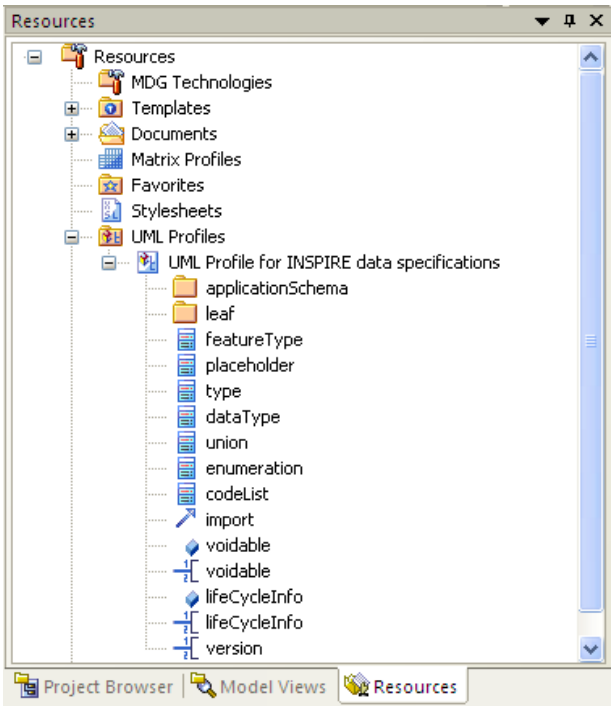


Figure 4.11: Resources tab after the import.

NOTE The stereotypes included in the INSPIRE UML profile reflect all the stereotypes that were required for the data specifications work in Annex I. If, during the work on the Annex II & III data specifications, further stereotypes are required, these should be added to the profile and documented in the Generic Conceptual Model.

B.2 Documenting classes, attributes and association roles

Classes are documented using the Notes text field. The documentation shall include:

- a natural language name for that class, e.g. “speed limit” for the class SpeedLimit;
- a definition; and
- (optionally) a description, including further explanations, examples or references for the definition.

The documentation in the Notes field needs to follow a certain convention (see below). Based on this convention, the feature catalogue in the INSPIRE data specification and much of the content of the Implementing Rule are auto-generated.

Following this convention as closely as possibly greatly reduces the effort required for manual editing.

Convention: The three elements shall be documented in the Notes field as follows:

- The natural language name
  - is introduced by the tag "-- Name --"
  - shall be given in lower case.

- The definition
  - is introduced by the tag "-- Definition --"
  - shall be as concise as possible and not contain examples or further explanations (these shall go into the description)
  - shall not start with "An address component is …"
  - shall end with a full stop (".")
- If the definition uses another term that needs to be defined, this additional definition shall be included using the keyword DEFINITION (all upper case).

- The description:
  - is introduced by the tag "-- Description --"
  - is optional. If there is not further description, do not include the -- Description -- tag.
- The source reference of the definition can be included using the keyword SOURCE, after which a short name of the Source shall be included. The full reference shall be included in the INSPIRE data specification.
- Additional explanations in the descriptions shall be introduced by the keyword NOTE (all upper case). If there is more than one note, the notes shall be numbered: NOTE 1, NOTE 2, etc.
- Examples shall be introduced by the keyword EXAMPLE (all upper case). If there is more than one example, the examples shall be numbered: EXAMPLE 1, EXAMPLE 2, etc.

The following (partly fictional) example for the spatial object type "AddressComponent" shows how the convention should be applied:

> -- Name –
> address component
>
> -- Definition --
> Identifier or geographic name of a specific geographic area, location, or other spatial object which defines the scope of an address.
>
> DEFINITION Geographic name: A proper noun applied to a real world entity.
>
> -- Description --
> SOURCE [UPU-S21]
>
> NOTE 1 Four different subclasses of address components are defined:
> - Administrative unit name, which may include name of country, name of municipality, name of district
> - Address area name like e.g. name of village or settlement
> - Thoroughfare name, most often road name
> - Postal descriptor

In order to construct an address, these subclasses are often structured hierarchically.

NOTE 2 It is the combination of the address locator and the address components, which makes a specific address spatial object readable and unambiguous for the human user.

EXAMPLE The combination of the locator "13" and the address components "Calle Mayor" (thoroughfare name), "Cortijo del Marqués" (address area name), "41037" (postal descriptor), "Écija", "Sevilla" and "España" (administrative unit names) makes this specific address.

The convention for documenting classes also applies for documenting attributes and association roles.

NOTE Since the INSPIRE data specifications specify a data model for data exchange, operations are typically not specified. If this is required for some reason, they can be specified in the same way as attributes.

B.3 Documenting constraints

Constraints are documented as follows:

Convention: For constraints in INSPIRE, select *OCL* in the *Type* drop down box and specify a name in the text field *Constraint.* The name should ideally give some indication on the content of the constraint.

The actual constraint is specified in the larger text box below. According to the GCM, OCL constraints should also be reported in natural language. In OCL, this can be done using the comment syntax: `/* ... */`. The OCL constraint itself should be specified below, starting with `inv:` for invariants. For example, Figure 4.12 specifies that an address shall have exactly one default geographic position. During the generation of the feature catalogue (Section 4), the natural language and the OCL expressions are extracted from this text.

For more information on OCL, see e.g. the *OCL 2.2 specification*[5], a *brief OCL 2.0 syntax*[6] or this *OCL tutorial*[7] (including an *OCL syntax checker*[8]).

Convention: When referring to spatial objects of a specific type, use one of the following options:

- (preferred) you can refer to a FeatureTypeName spatial object, e.g. an AddressComponent (UpperCamelCase) spatial object, or
- your can refer to the natural language name of the feature type, e.g. an address component.

When referring to an attribute or association role of a spatial object, use one of the following options:

- (preferred) you can refer to the attributeName (lowerCamelCase) attribute or associationRoleName association (role), e.g. the name attribute of the Address spatial object, or
- you can refer to the attributeName (lowerCamelCase) or associationRoleName, e.g. the name of the address.

If the choice is between an elegant but ambiguous and a slightly awkward, but unambiguous phrasing, go for the latter. The people having to interpret and translate your text will thank you!

[5] http://www.omg.org/spec/OCL/2.2/PDF/
[6] http://www.csci.csusb.edu/dick/samples/ocl.html
[7] http://atlanmod.emn.fr/atldemo/oclturorial/
[8] http://atlanmod.emn.fr/atldemo/oclturorial/simpleOCLWebTester

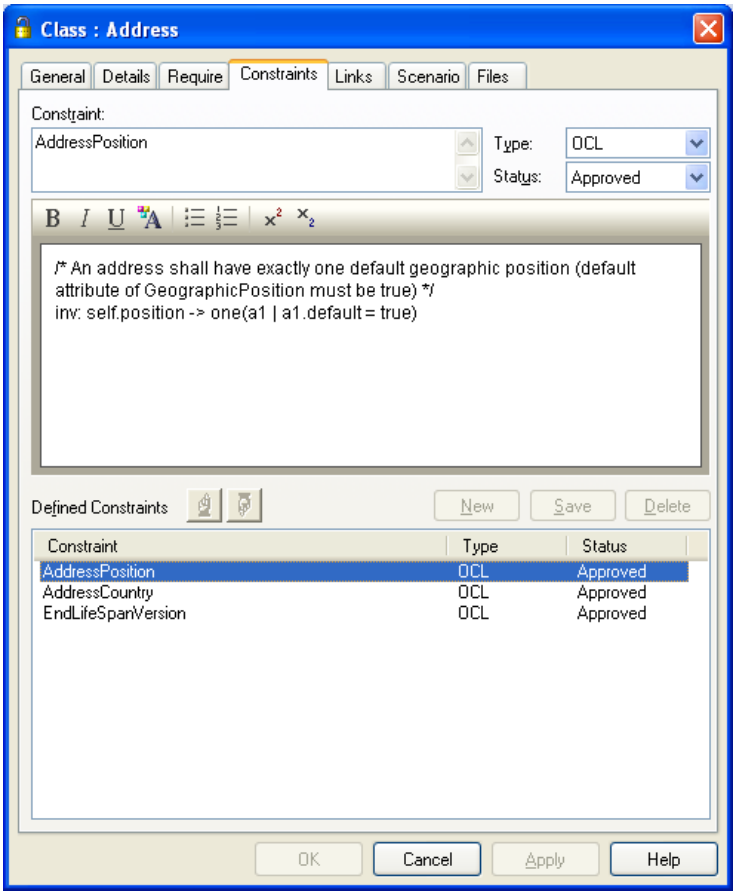


Figure 4.12: Adding a constraint to a class.