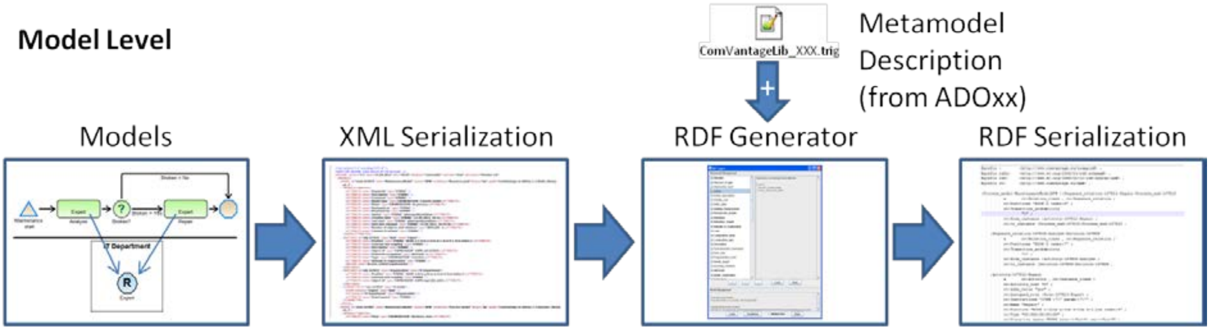
RDF Export

This document provides a simple tutorial on how to use the new (and improved?) RDF Export.

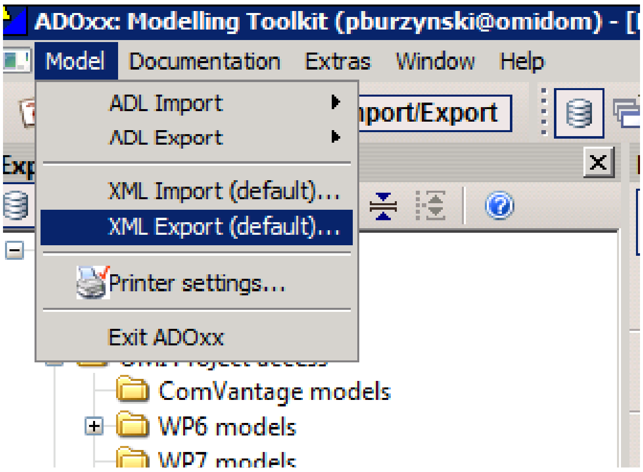
In order to run it a proper java installation is necessary. Java can be obtained at http://www.oracle.com/technetwork/java/javase/downloads/index.html (last accessed 18. Feb. 2013). To run a Java application the JRE is needed (JDK is necessary for development). The implementations have been tested using Java 1.6.0_16, but they'll probably also work with newer versions.

Please not that this has been created on a Computer using the “German” language, therefore certain buttons can have a different label (e.g. “Ja” instead of “Yes” or “Öffnen” instead of “Open” etc.).

General Procedure



1. Create the models in the Prototype
2. Export the desired models as XML

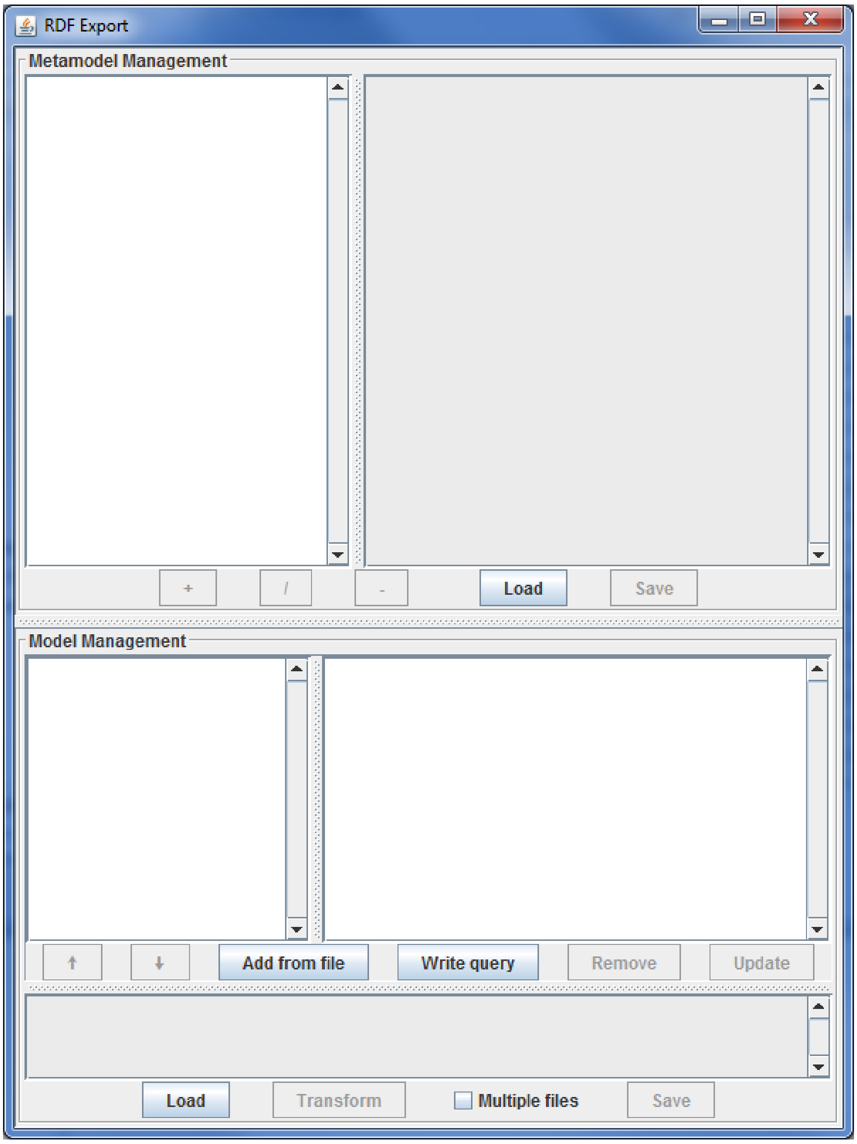


3. Run the .jar file of the RDFExport

IMPORTANT: The “adoxml31.dtd” has to be in the same folder as the “RDFExport.jar”!

Using the RDF Export

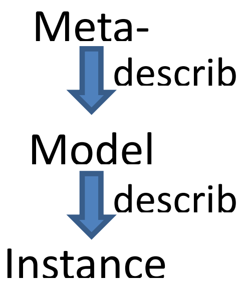
When starting the RDF Export you are greeted with the following window:



It can be divided into two major parts:

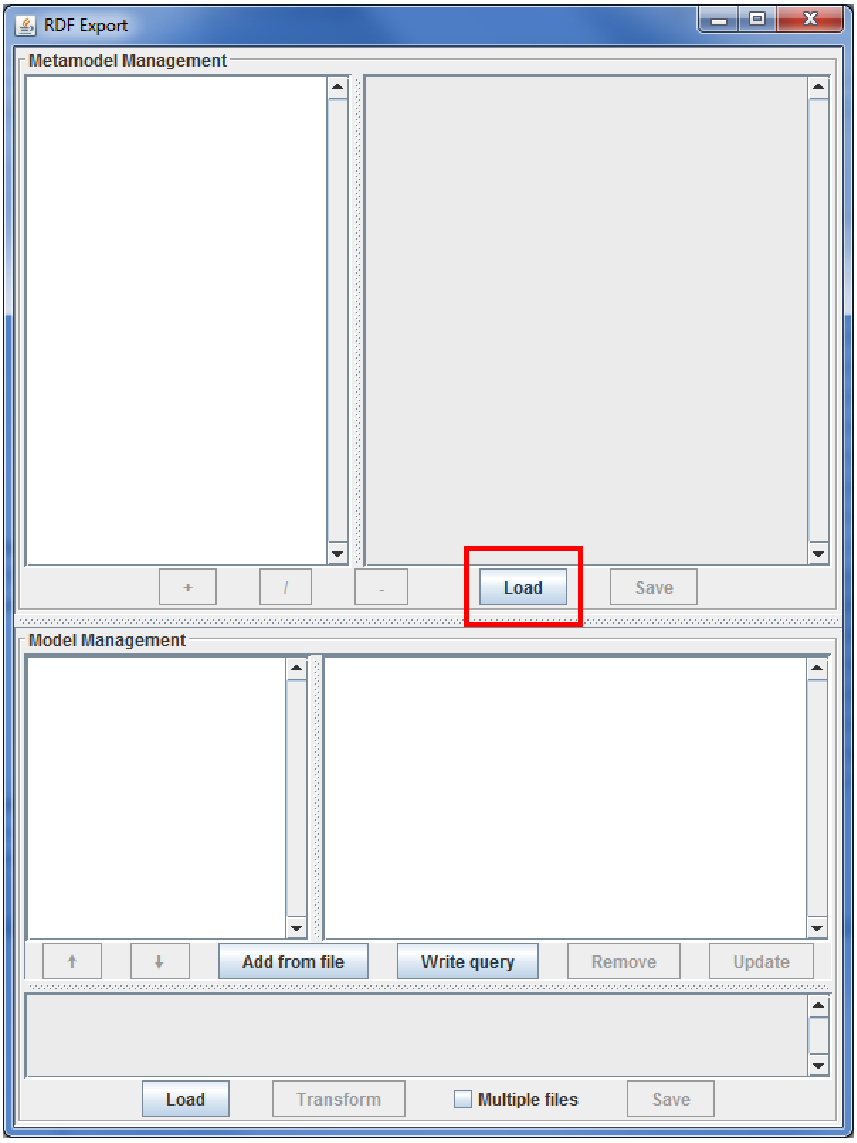
1. Top – Metamodel management
2. Bottom – Model management (requires data from the Metamodel management)

The recommended procedure on how to work with this implementation is to first use the Metamodel management to load data/information about the metamodel and then use the Model management to load the model data/information and transform it.



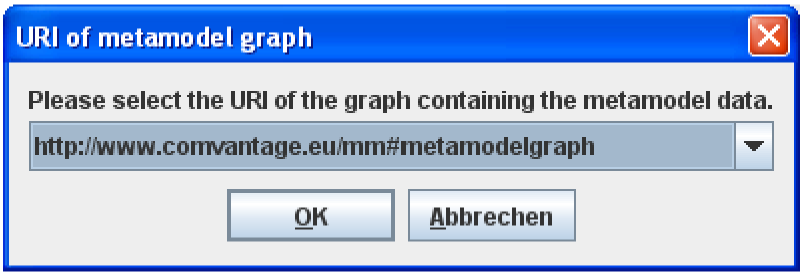
Metamodel Management

First the Metamodel description should be loaded in the Metamodel management part by using the "Load" button.



This in turn will open up a file selection window. Here select from which file to load the descriptions. Different types are supported like TriG and XML among others. A TriG file (ComVantageLib_XXX.trig) containing the metamodel description for one of the ComVantage prototypes has been provided together with the RDF Export.

If the selected file contains more than one graph then another window will ask which graph contains the metamodel description (as in the picture below).[1]



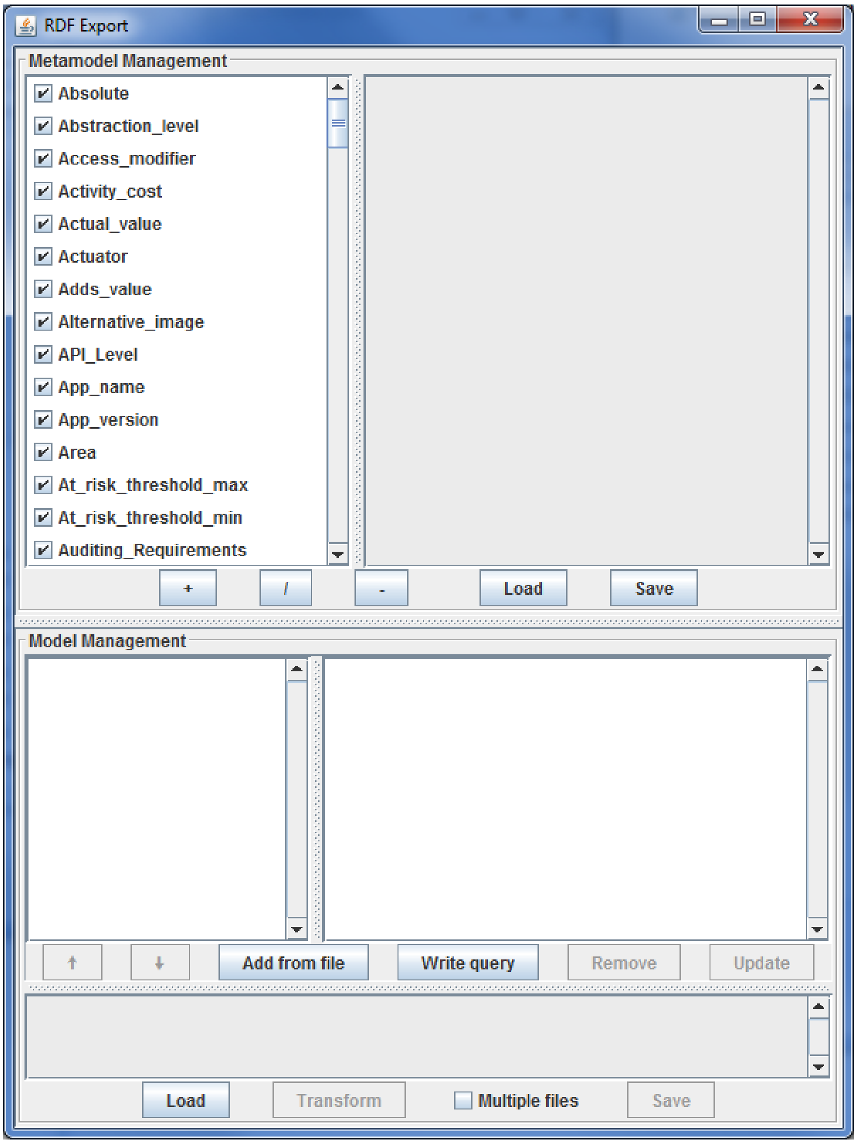
Please select the URI of the graph containing the metamodel data.

http://www.comvantage.eu/mm#metamodelgraph

---

[1] Other windows might appear depending on the selected file type, but they should be self explanatory.

If the right graph has been loaded then the main window should change to something resembling the following:



☑ Absolute
☑ Abstraction_level
☑ Access_modifier
☑ Activity_cost
☑ Actual_value
☑ Actuator
☑ Adds_value
☑ Alternative_image
☑ API_Level
☑ App_name
☑ App_version
☑ Area
☑ At_risk_threshold_max
☑ At_risk_threshold_min
☑ Auditing_Requirements

☐ Multiple files

In the Metamodel management part the left list contains all the available attributes. If no attributes could be determined from the loaded graph (because it is following an unknown schema or because there simply are none) then the list will contain one entry telling the user this. This usually also means that the Model management won't work properly.

When the metamodel description has been properly loaded and an attribute is selected (as in the above picture) then the right part will contain its full URI, a short description about the attribute (if available) and in what element types it is used.

The +, - and / buttons below the list (to the left of the "Load" button) as well as those keys on the keyboard can be used to remove (-), add (+) or switch (/) the selected attributes[2]. Deactivated attributes (the ones without the tick to their left) will not be exported. This is useful to prevent exporting unwanted attributes like for example "Show name" which is used in the modelling tool only for visualization purposes.

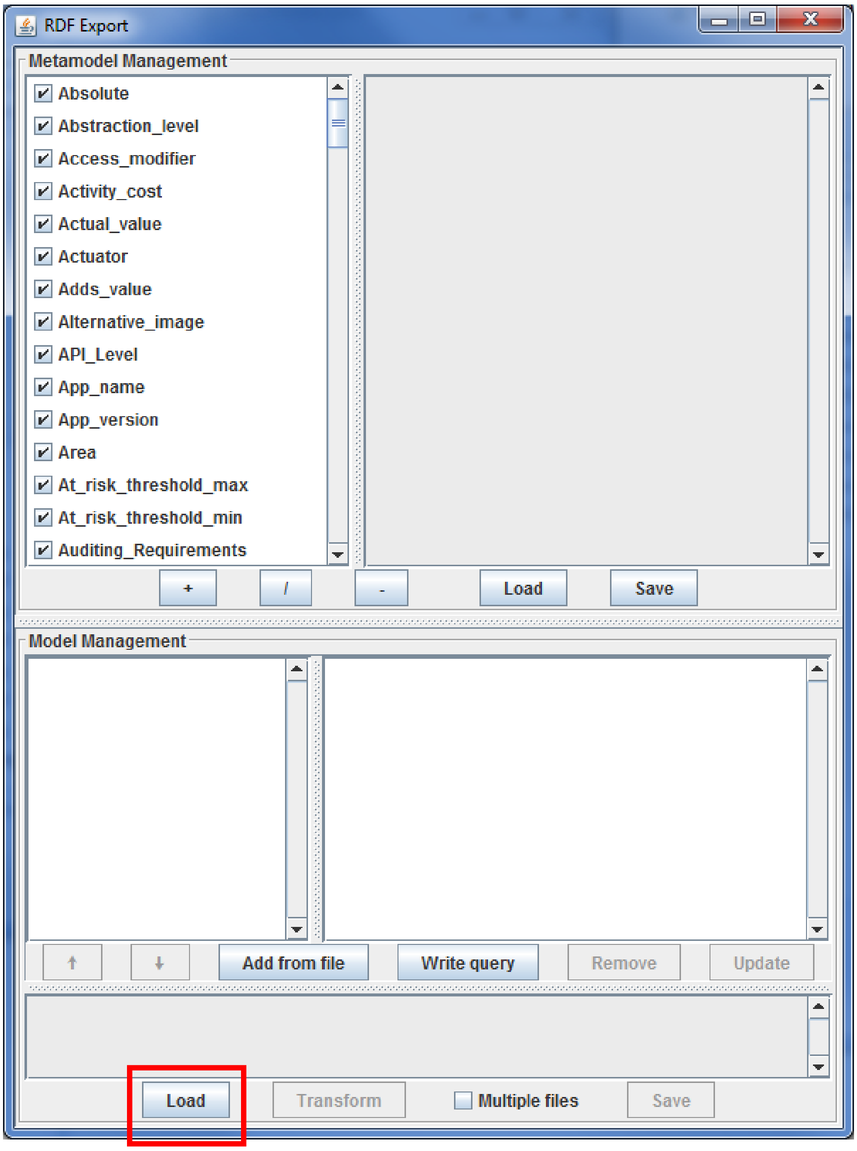
---

[2] Using Shift and Ctrl more than one attribute can be selected at a time. Some shortcuts are automatically supported by Java (e.g. Ctrl+A selects everything).

The “Save” button (to the right of the “Load” button) can be used to save those changes in a new file and also maybe using a new file format. This will remove the statements about deactivated attributes in the saved file. It should be noted that the implementation makes use of the file endings (.trig for TriG syntax, .trix for TriX syntax, .ttl for Turtle syntax etc.). Therefore if something goes wrong during saving, this can be one of the reasons why.

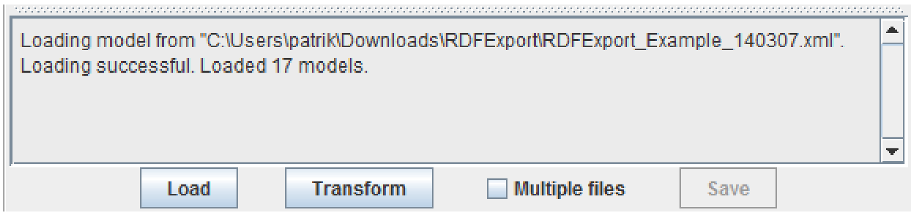
Model Management

After the metamodel has been loaded, and if needed its attributes adapted, the mode should be loaded using the “Load” button. The area above the buttons is used to provide information to the user (“Loading successful”, “Transformation failed” etc.) and will be here called “Info box”.



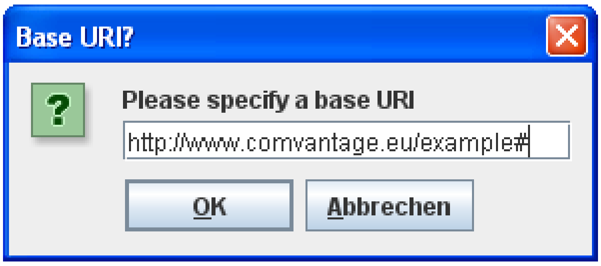
Again this will open a file selection window where the file containing model data in XML format should be selected. This model data has to correspond to the previously loaded metamodel. Otherwise the result of the transformation might not be useful.

After the model data has been successfully loaded the Info box should look something like this:



```
Loading model from "C:\Users\patrik\Downloads\RDFExport\RDFExport_Example_140307.xml".
Loading successful. Loaded 17 models.
```

Now the models can be transformed by pressing the "Transform" button (to the right of the "Load" button). A new window will asking for a base URI[3] to use.

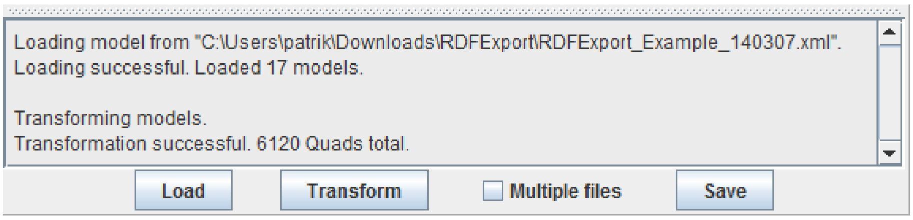


Please specify a base URI

http://www.comvantage.eu/example#

After the base URI has been provided the Model management will take the current version of the Metamodel description from the Metamodel management part and create the RDF statements according to this metamodel out of the loaded model data. Therefore it is not necessary to load the model data again when something on the metamodel has been changed (i.e. attributes activated/deactivated). Instead just press the "Transform" button again.

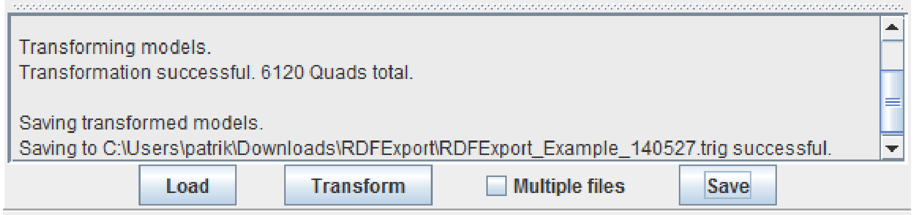
After a successful transformation the Info box should look something like this:



```
Loading model from "C:\Users\patrik\Downloads\RDFExport\RDFExport_Example_140307.xml".
Loading successful. Loaded 17 models.

Transforming models.
Transformation successful. 6120 Quads total.
```

Now the graphs created from the models can be saved by pressing the "Save" button. Either all graphs in one file (no tick next to "Multiple files") or each graph in a separate file (a tick next to "Multiple files")[4]. Like always this will open a file selection window where either a file or a folder as the save location can be selected. However folders can only be selected when more than one file could be generated (i.e. "Multiple files" has a tick). Again remember that the implementation makes use of the file endings, which can be a source for errors. Success looks something like this:



```
Transforming models.
Transformation successful. 6120 Quads total.

Saving transformed models.
Saving to C:\Users\patrik\Downloads\RDFExport\RDFExport_Example_140527.trig successful.
```

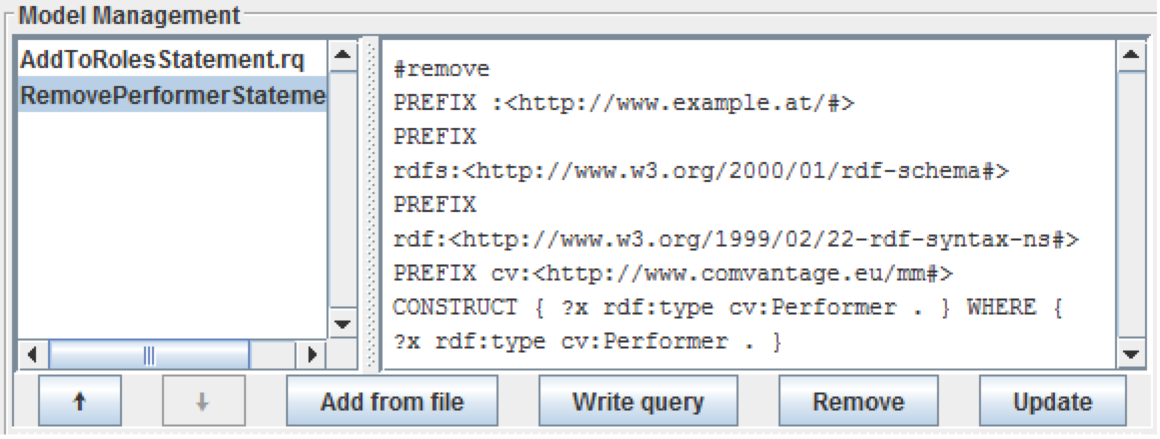
---

[3] The base URI is used to create the unique identifier for all the elements by using it as a prefix (in this example: MaintenanceModel → http://www.comvantage.eu/example#MaintenanceModel).

[4] In certain cases like saving in Turtle format this is ignored, since every graph has to be in a separate file anyway.

Minor support for transformation

A new feature of the improved RDF Export is a limited support for automatically adding or removing certain statements. This feature uses SPARQL Construct queries for both cases (add and remove). The important component in the GUI is shown in the following picture (the picture shows a remove query):



```
#remove
PREFIX :<http://www.example.at/#>
PREFIX
rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX
rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX cv:<http://www.comvantage.eu/mm#>
CONSTRUCT { ?x rdf:type cv:Performer . } WHERE {
?x rdf:type cv:Performer . }
```

Here the list on the left contains the names of all the loaded Queries. Queries can be loaded either from a file using the “Add from file” button (the filename is then used as a name) or can be written directly in the GUI using the “Write query” button (then a name can be specified). The “Add from file” button shows a file selection window where one or several files and/or directories can be selected[5]. The “Write query” button creates an entry with an empty query. When an entry is selected in the list the actual query will be shown in the field on the right. This field can be used to further edit the query. Press the “Update” button once the query has been adapted to notify the GUI about its change. Queries can also be removed from the GUI by selecting them in the list and using the “Remove” button.

When transforming a model, all loaded queries will be executed on each created graph, i.e. if 3 graphs have been created and 2 queries are available, then each of the two queries will be executed on each of the three graphs separately. They are however executed in the order that is shown in the list on the left (queries further up are executed before queries further down). This order can be manipulated using the up and down arrow buttons. Note that the tool does not check if it is a valid query (i.e. following the SPARQL syntax etc.)! It will try to execute the query and if an error occurs the information will be added to the info box below.

The tool distinguishes between add and remove queries by looking at their beginning. If it starts with a “#remove”, then all statements that are created by the construct query will be removed/deleted from the graph. Otherwise (if it doesn’t start with “#remove”) all statements created by the construct query are added to the graph.

Please note that the supported query concepts are restricted by the library used in the implementation and therefore not all queries that are valid SPARQL can necessarily be executed in this tool.

---

[5] Directories searched recursively, i.e. all the queries contained

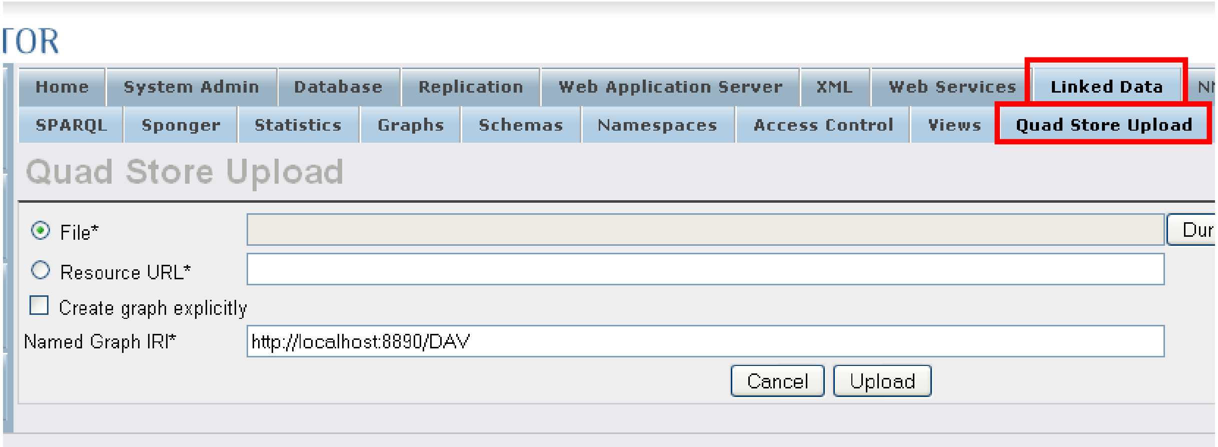
Use with a Quad Store

The generated results can also be published to a Quad Store like Virtuoso or Sesame.

Publishing to Virtuoso

In order to publish the data in Virtuoso you have to save the files in Turtle (.ttl) format. Putting a tick next to "Multiple files" allows you to select a folder to save to instead of a file, which should make things easier. This generates one file for each graph.

Now start the Virtuoso Conductor, log in and go to the "Quad Store Upload" found in the "Linked Data" part.



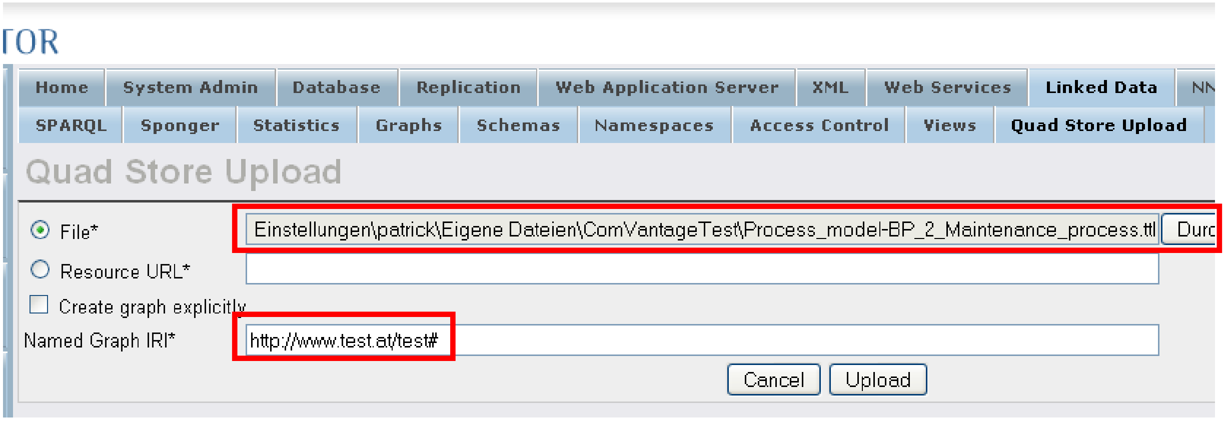
● File*

○ Resource URL*

☐ Create graph explicitly

Named Graph IRI* http://localhost:8890/DAV

Once there select one Turtle file containing a graph and specify the URI for the graph under "Named Graph IRI". Then click on "Upload" to upload the graph.



● File* Einstellungen\patrick\Eigene Dateien\ComVantageTest\Process_model-BP_2_Maintenance_process.ttl

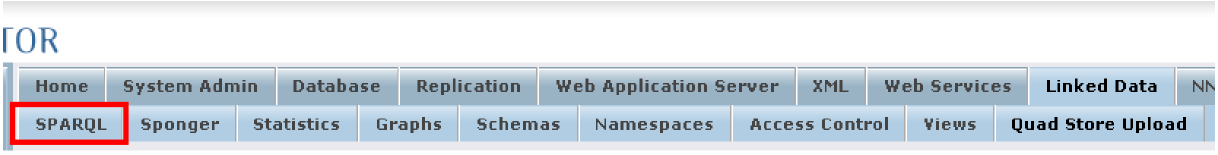
○ Resource URL*

☐ Create graph explicitly

Named Graph IRI* http://www.test.at/test#

Now repeat the previous step (select file, provide URI and click Upload) for every graph you want to publish.

Once finished, you can query the data from the SPARQL component of the Virtuoso Conductor.


TOR
Home
System Admin
Database
Replication
Web Application Server
XML
Web Services
Linked Data
NN
SPARQL
Sponger
Statistics
Graphs
Schemas
Namespaces
Access Control
Views
Quad Store Upload