Demonstrator guidelines for demo **ENTERKNOW PN-III-P2-2.1-PED-2016-1140** - early iteration draft

Contents

Part 1. Set Up the Modelling Tool

Preparing the EnterKnow tool

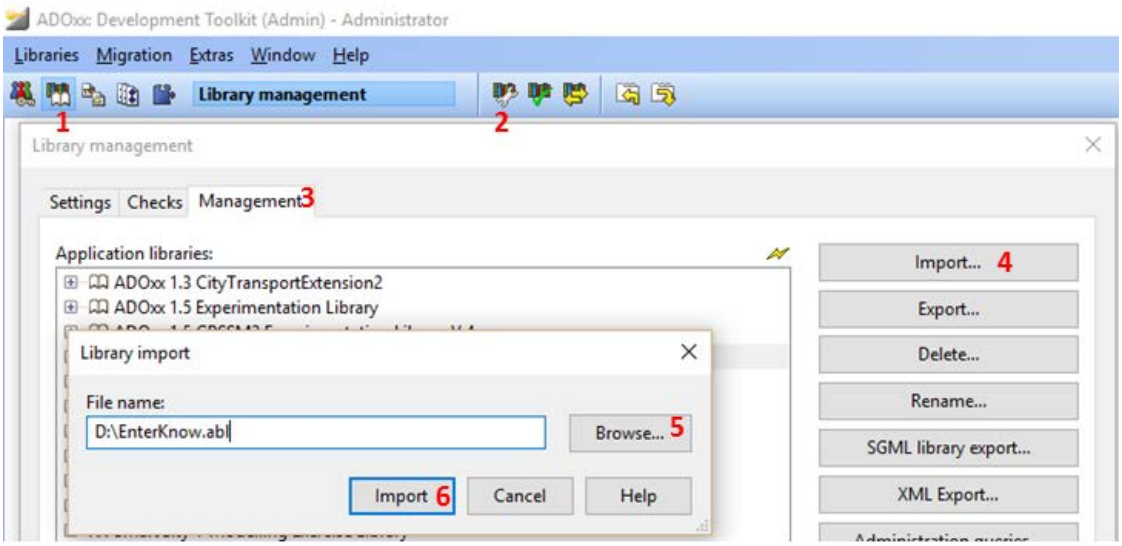
The EnterKnow modelling tool is provided as ADOxx source library. This means that, in order to actually use the tool, you must compile this source on the ADOxx (http://adoxx.org) platform by following several simple steps:

Step 1. Log in

Open ADOxx Development Toolkit and log in with the default credentials (user=*Admin*, password=*password*)

Step 2. Import the source

1. Open the Library Management feature
2. Open Library Settings – you should see the list of method implementations (called "libraries") which are currently deployed on your ADOxx installation. If this is a fresh installation you should only see an "Experimentation Library". We need to add to this list the EnterKnow implementation source by importing it.
3. Open the Management Tab
4. Press Import
5. Press Browse and select the EnterKnow.abl file from where you have saved it.
6. Press Import

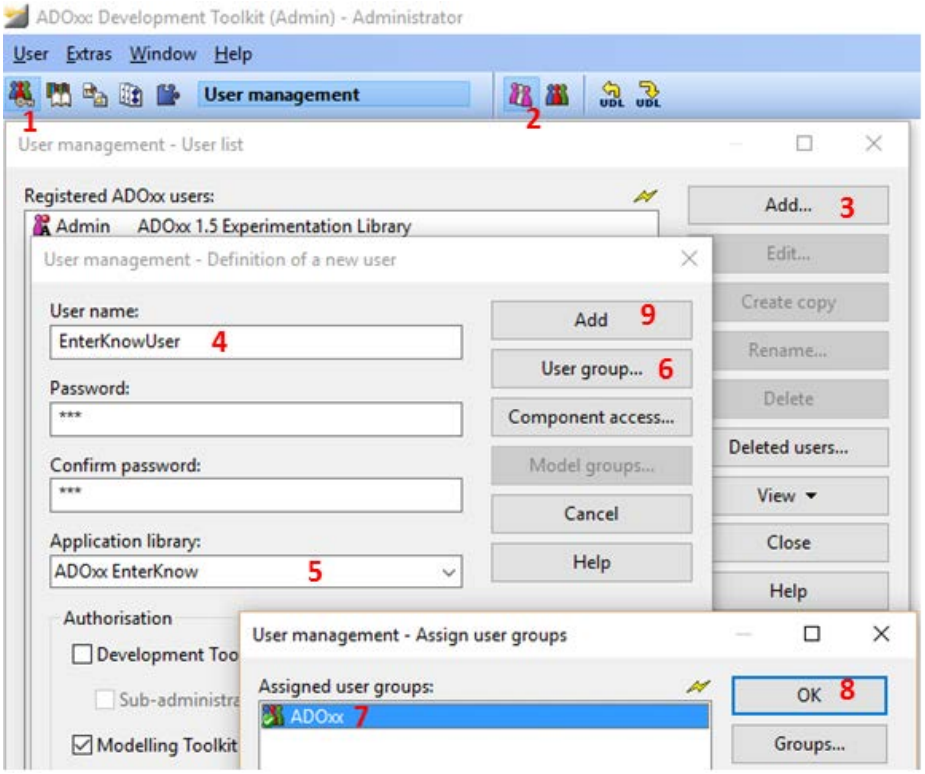


If this is the first time you import it, you will be asked "Create a default model group?". Answer "Yes" and you should see a success message, then the EnterKnow library will be visible in the list of implementations.
If this is not the first time you import it (e.g., you use a computer where someone else already imported this library) then you might get a warning that you must change the name of the newly imported library. You will be prompted three times to change the name (once for each of the 3 components of an ADOxx library). Do it three times to avoid the name conflict.

From now on, in the screenshots throughout this document the imported library will be visible with the default name "EnterKnow" (if you have to import again and to rename the library, on your computer it will have the changed name – for example "EnterKnow - demo").

Step 3. Define a user for the modelling tool

1. Open the User Management feature
2. Open User Settings – you should see the list of defined users
3. Press Add to create a new user
4. In the new window create a name, a password, confirm the password
5. After the password is created, assign it to the EnterKnow implementation (in the Application library dropdown list)
6. Press User Group to assign the user rights
7. In the new Window press on the default ADOxx user group, thus assigning the default rights
8. Press OK to finish the user rights assignment
9. Press Add to finish creating the new user (and remember the name and password)



Close all windows.

Step 4. Start the modelling tool

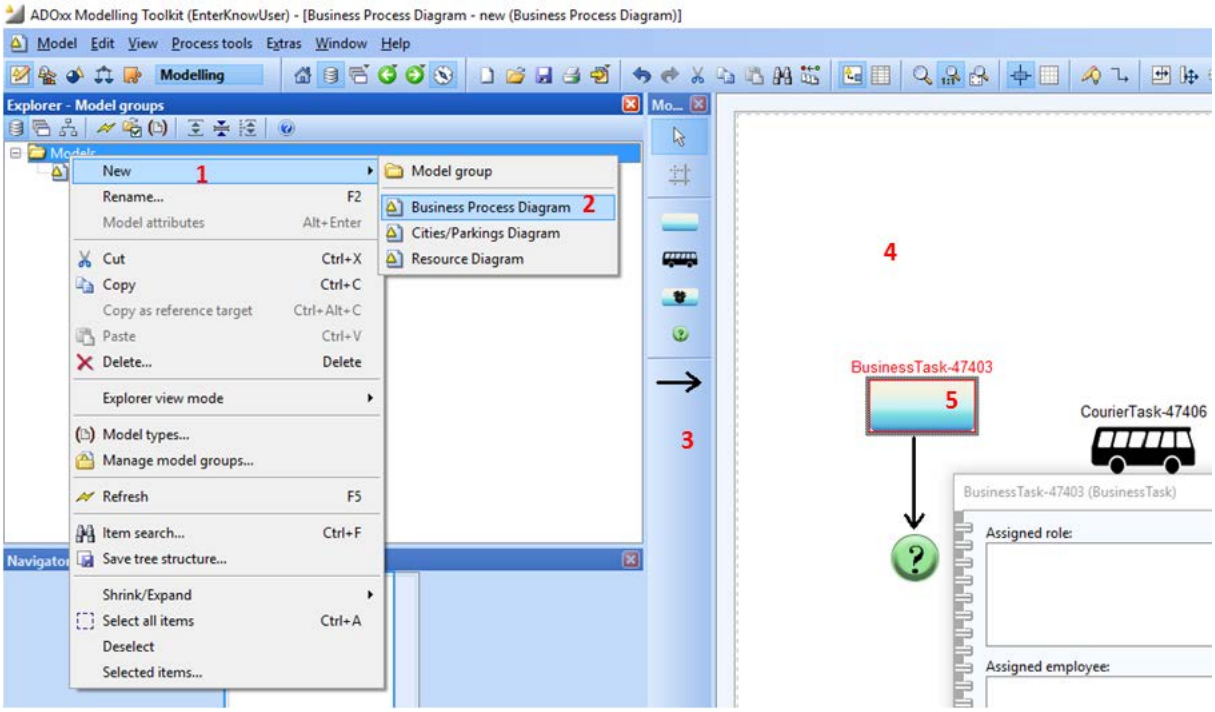
If you are familiar with ADOxx method development, and you want to study and/or extend the implementation of the EnterKnow modelling tool, you will continue working in the ADOxx Development Toolkit (in Library Management).

However, for this exercise we only want to use the modelling tool, not to extend it. Therefore you can close the ADOxx Development Toolkit.

To see the compiled tool, you need to start ADOxx Modelling Toolkit. This time, you need to log in with the credentials you've just created (in the previous step).

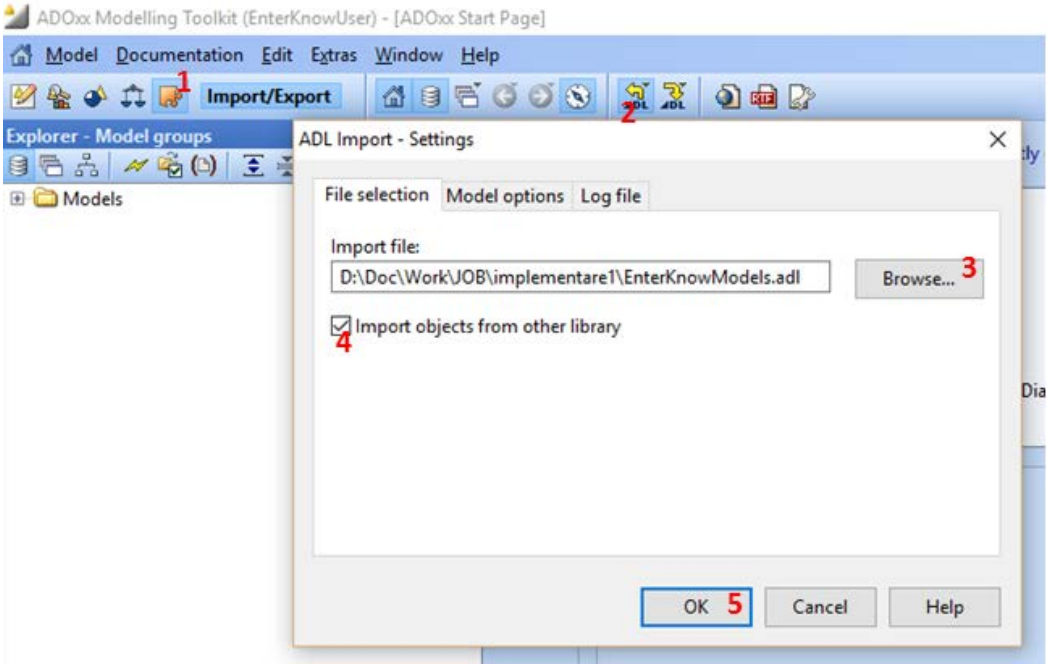
After logging in, the EnterKnow tool is ready for use:

1. Right-click on the Models folder and press New
2. You will see the types of models provided by the EnterKnow modelling language. Select one (we will come back later to explain some of them)
3. You will see the list of concepts and connectors (relations) made available for the selected model type
4. Position concepts and connectors on the modelling canvas. For now your model does not have to make sense, just get used to the user interface and the drawing surface
5. Double click on a symbol and you should see its prescribed, machine-readable properties



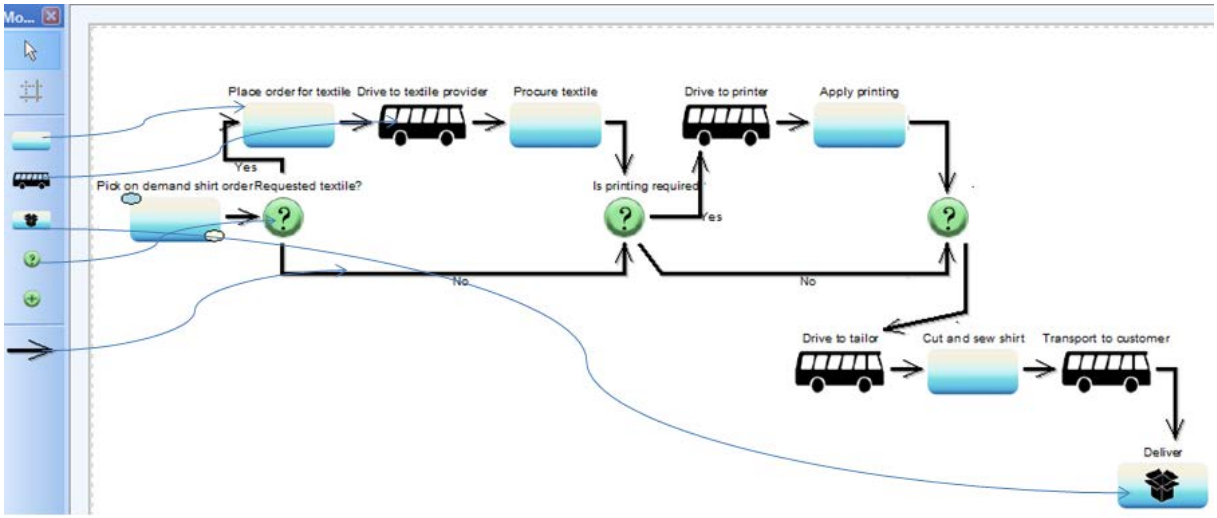
Part 2. Create Models

The exemplified models have already been created ("My Business Model", "My Organization Model", "My City/Region Model") and you can see them if you import the EnterKnowModels.adl file into ADOxx Modelling Toolkit. Log in in ADOxx Modelling Toolkit with your credentials (here our user was "EnterKnowUser", but yours can have different name) for EnterKnow library (imported before into ADOxx Development Toolkit) and then follow the steps emphasized in the picture below:



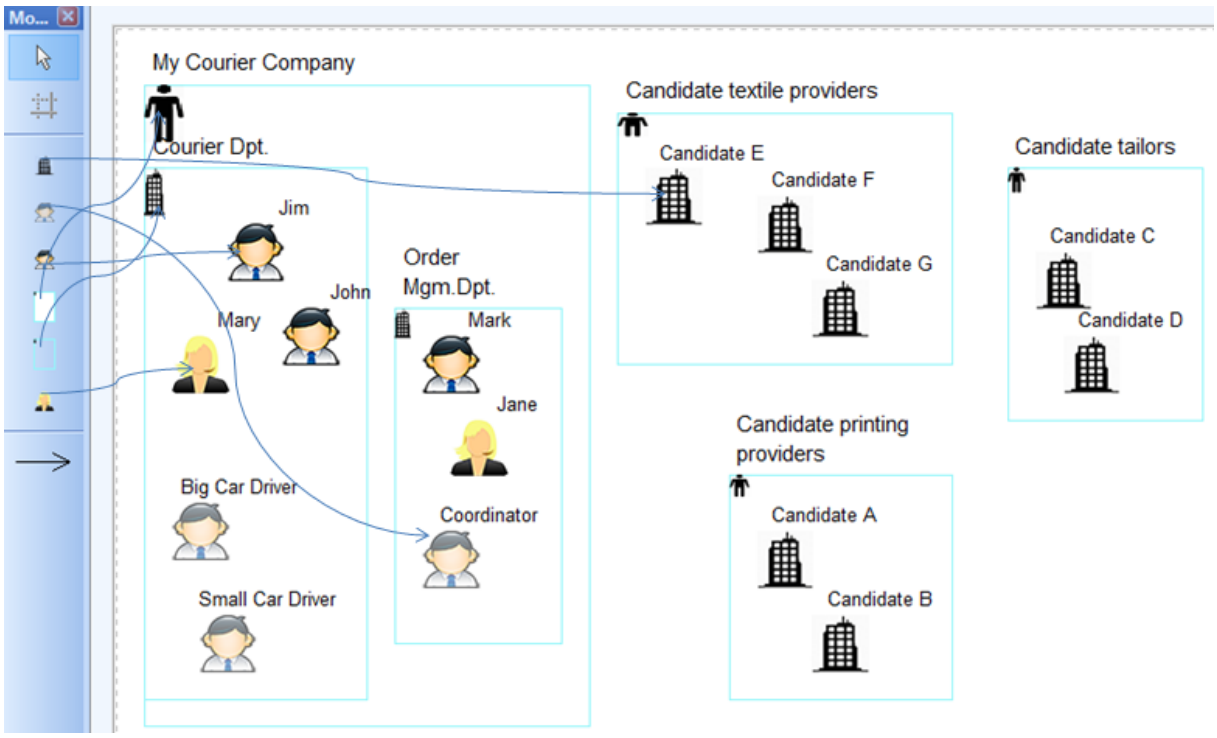
If you want to create the diagrams from the very beginning just follow the instructions given below (just click on "Models", select the type of diagram and pick and place objects on the modelling canvas):

Create a model named "My Business Model" of type "Business Process Diagram", as in the following example:



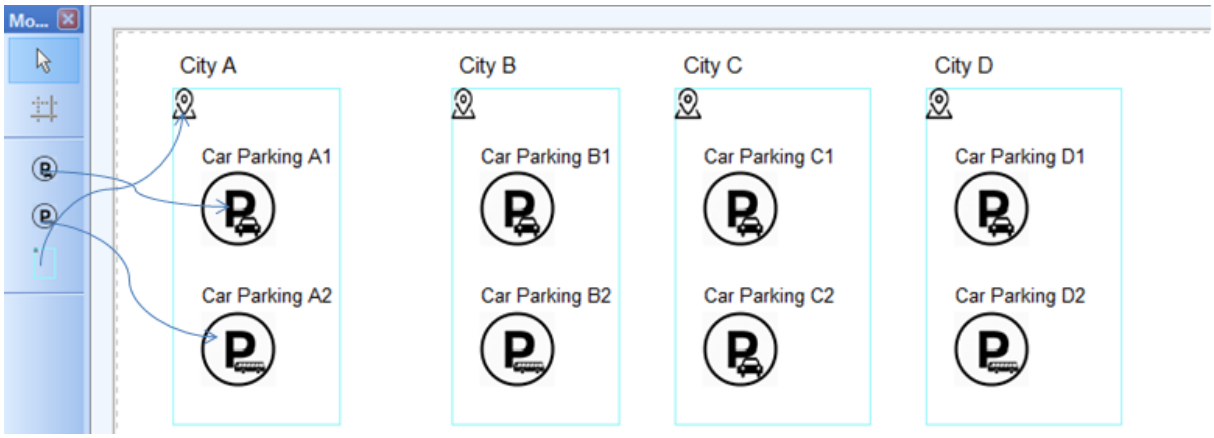
In this model you have three types of tasks (i.e., Business Task, Courier Task and Delivery Task), each one depending on who executes it (you will see later in the role diagram). The Yes/No annotations on the arrows outgoing from the decision node must be written in the "Name" attribute of the “followed By” relation/connector; the decision question must be written in the "Name" attribute of the Decision element.

Create a model named "My Organization Model" of type "Resource Diagram", as in the following example:



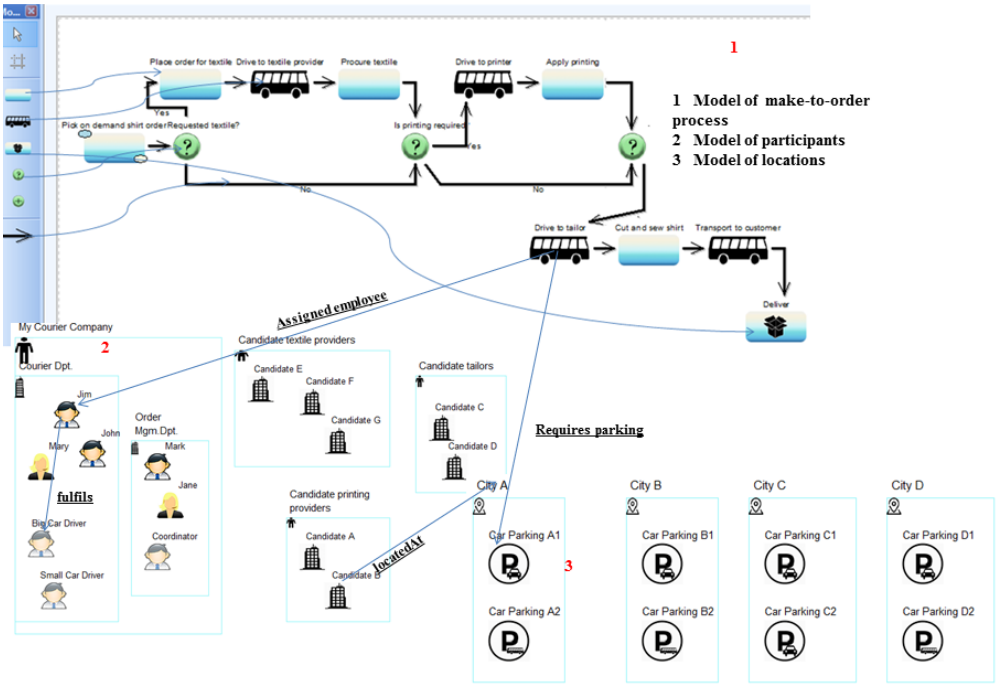
In this model you can see two types of containers (i.e., Organizational Unit, Organizational Aggregate), both containing some elements. The Organizational Unit contains Roles (e.g., Big Car Driver) and Employees (e.g., John) and the Organizational Aggregate contains Business Partners (e.g., Candidate A) and also Organizational Units.

Now, create the last type of model, “My City/Region Model”, of type “Cities/Parkings Diagram”, as in the following example:



The model is made up from several elements: City/Region (e.g., City A), Small Parking Area (e.g., Car Parking A1), Big Parking Area (e.g., Car Parking D2).

Up to this point, the semantics expressed in the model is partly left to human interpretation (as you see in the picture below):



- We need to figure out that the task (regardless of type), has to be executed by some role or employee;
- We need to think that the tasks also require parking areas and also that the business partners are located in different cities;
- We need also to figure out that an employee can fulfill any role from the model.

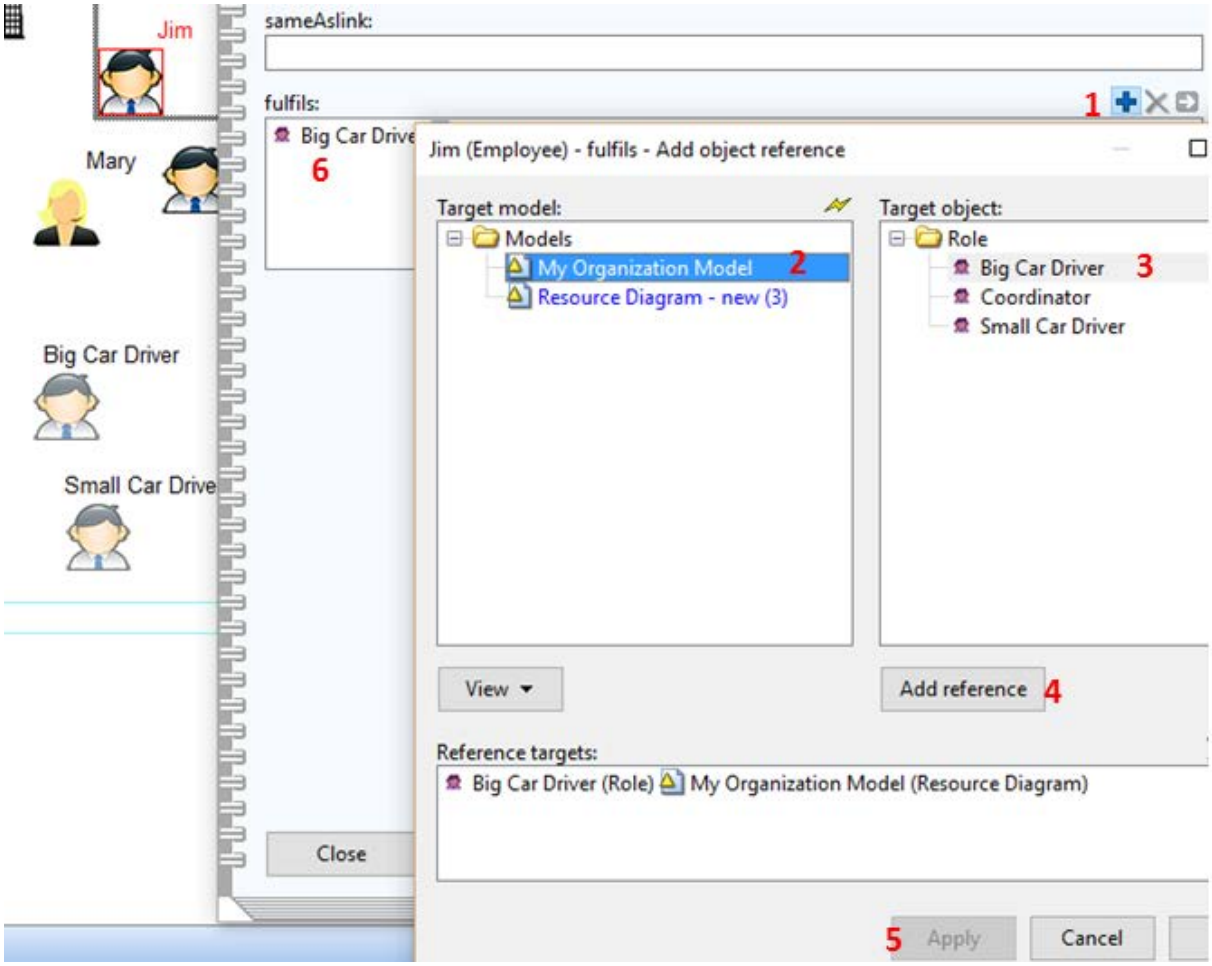
All these are relations left to human interpretation and the more straightforward way is to create from the very beginning semantic hyperlinks that express these relations in a structured way.

Part 3. Create Links between Models

Semantic hyperlinks are "hyperlinks with meaning". In other words:

- They can be used in the modelling tool to navigate between related models or from one element to another in the same model
- Each of them has a well-defined meaning, event their own properties - unlike HTML hyperlinks whose meaning is typically left to the human interpretation

First, create a link connecting two elements within the same model. Such links are useful as an alternative to visual connectors, to avoid visual cluttering. This means that certain semantics will not be communicated on a visual level (their understanding is ensured by interacting with the model, not just by looking at it).
An example of an in-model link is the one between an Employee and the associated Role.

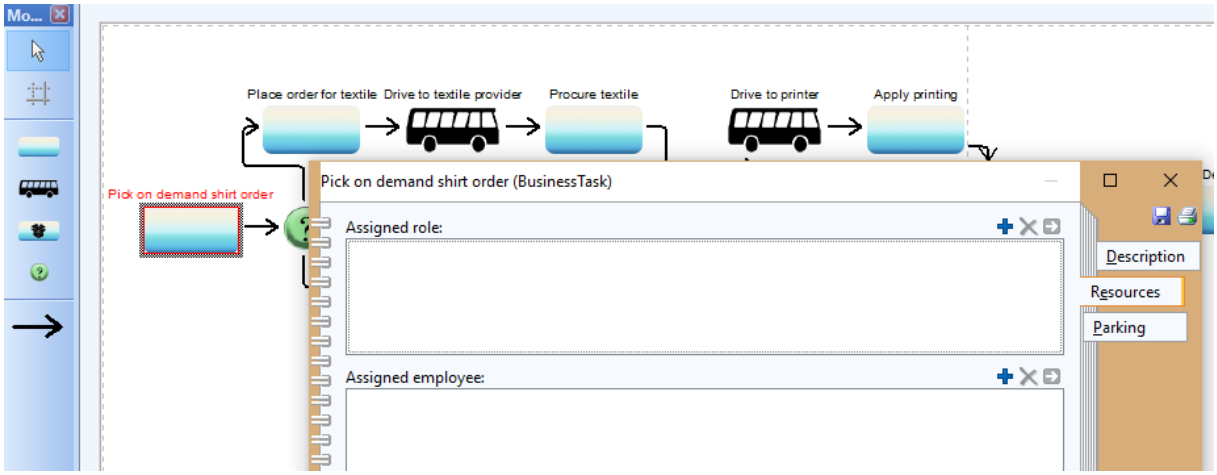


So, double click on "Jim" who is an Employee in “My Organization Model” to see its prescribed properties.

1. Click on “Description” field, then click on the “plus” sign located right above “fulfils” property
2. A new window appears, click on the model from where you want to choose the role (in this case is the same model “My Organization Model”)
3. Select a role (e.g., Big Car Driver)
4. Create the link (Add reference)

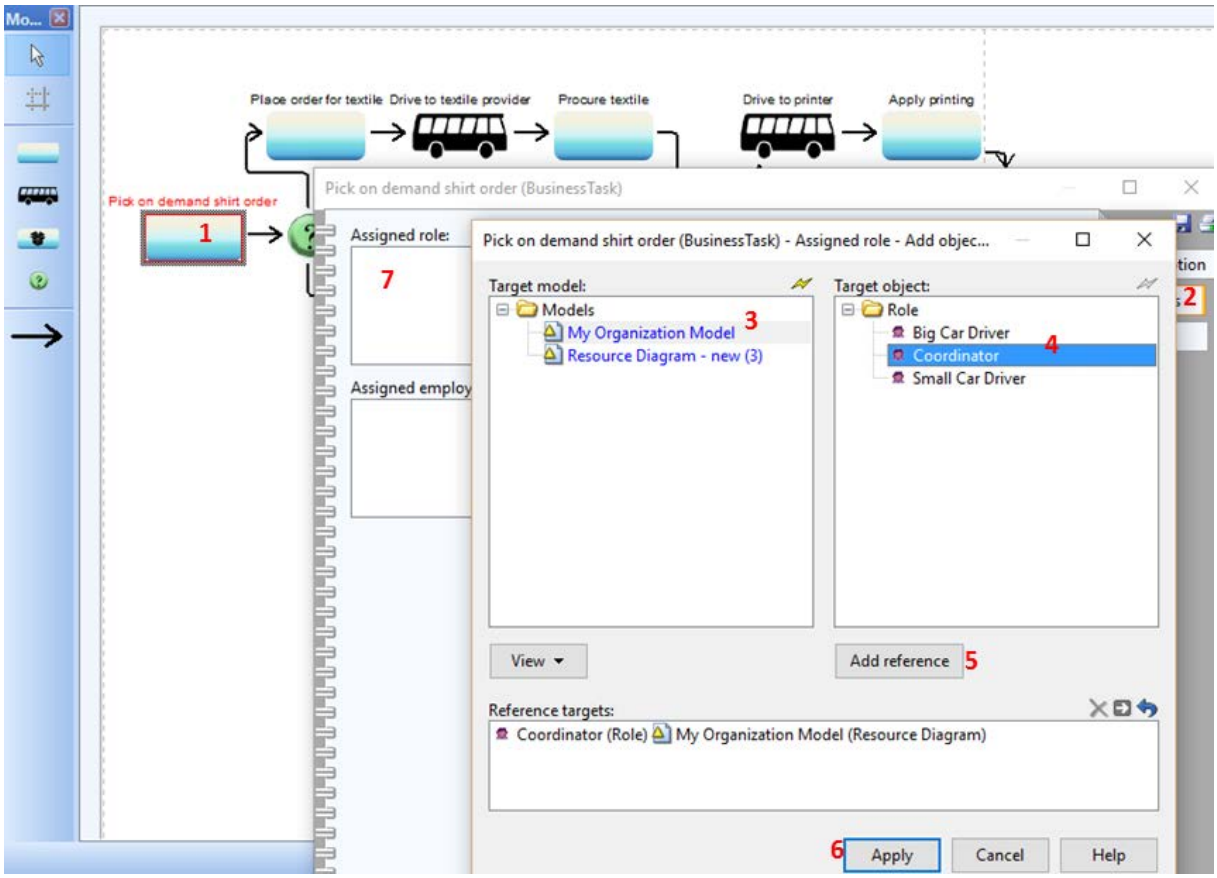
5. Apply
6. See the created link on the space allocated for “fulfils” property

Next we create a link from process activities to the roles that should perform them. Any type of task can be performed either by an employee or by a role (as seen in the picture below):



To assign a role for example, you should follow the steps:

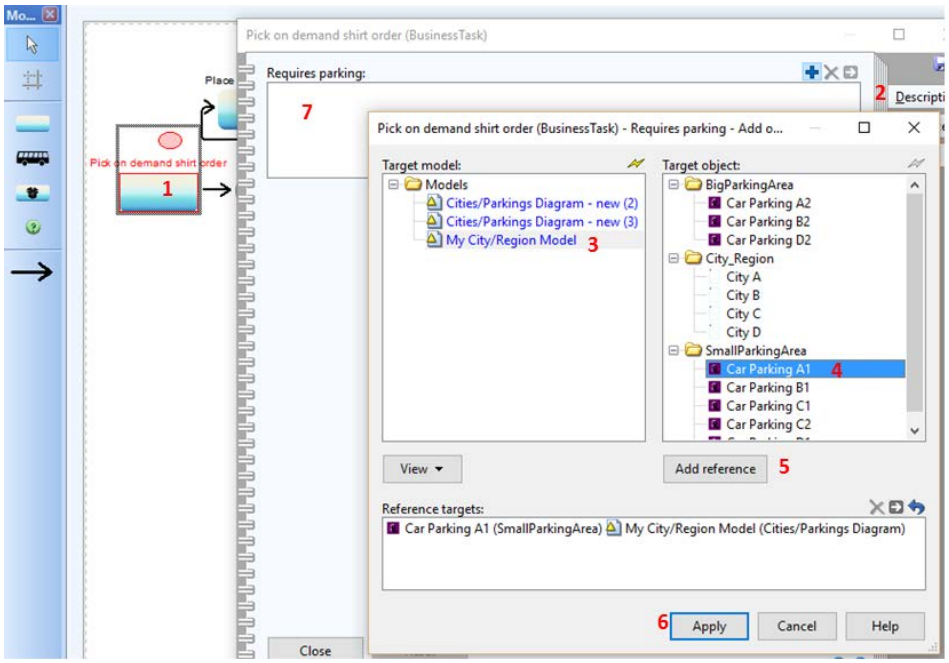
1. Double-click on a task (e.g., “Pick on demand shirt order”)
2. Select the “Resources” tab
3. Choose the resource diagram from which you want to pick the role (in this case “My Organization Model”)
4. You should see a list of roles, click on that one that you want
5. Add reference
6. Apply
7. See the assigned role on the dedicated field



Do the same for the activity if you want to assign an employee. Notice that a task can have assigned both the role and the employee. You will notice that the target role is displayed as a hyperlink anchor in each activity.

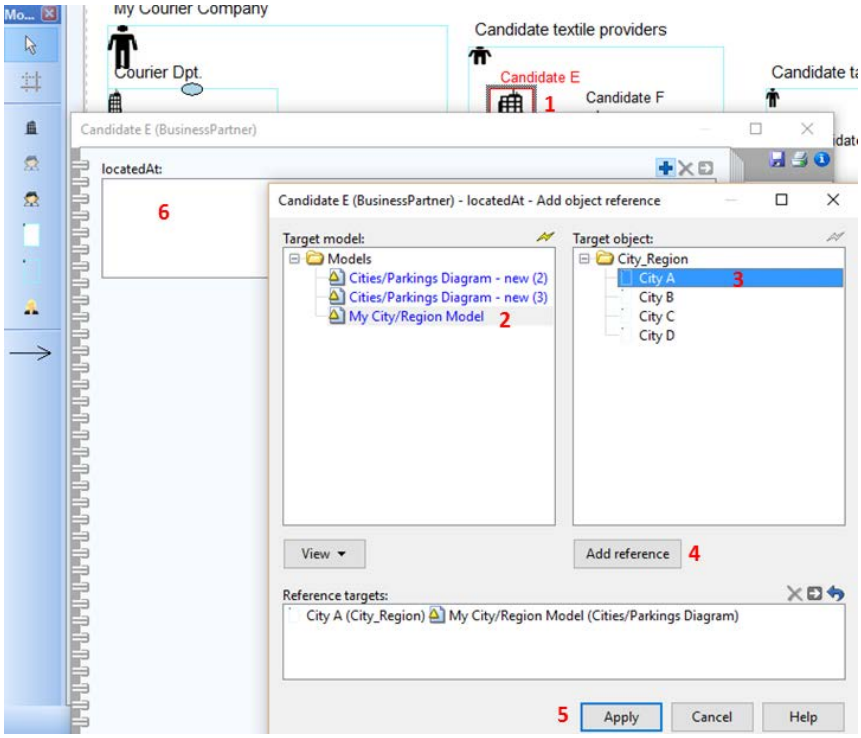
Next we move to the “My Business Model” to create a link between the task and the parking area.

1. Double-click on a task (e.g., “Pick on demand shirt order”)
2. Select the “Location” tab
3. Choose the resource diagram from which you want to pick the role (in this case “My City/Region Model”)
4. You should see a list of cities and parking areas, click on that one that you want
5. Add reference
6. Apply
7. See the assigned city/parking area on the dedicated field



Next we move to “My Organization Model” in order to choose a city for any business partner.

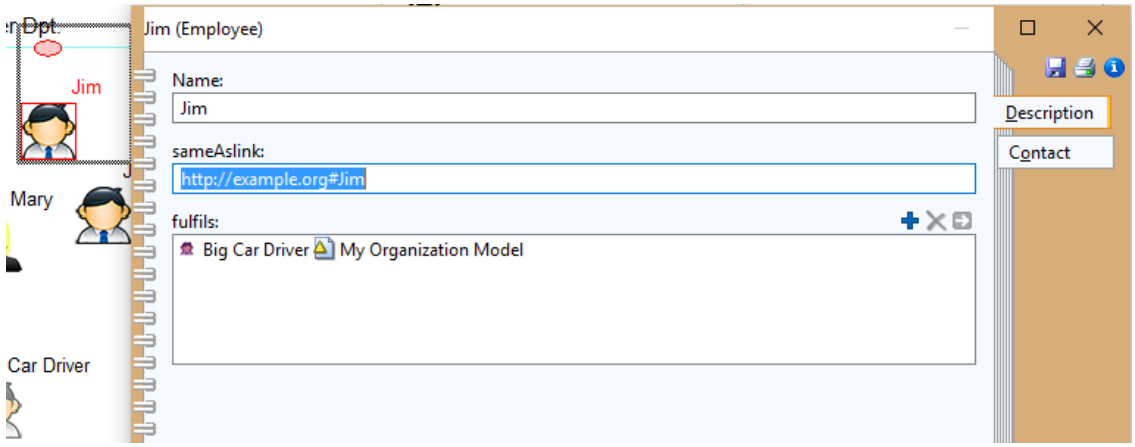
1. Double-click on a business partner (e.g., “Candidate E”)
2. Choose the location diagram from which you want to pick the role (in this case “My City/Region Model”)
3. You should see a list of cities, click on that one that you want
4. Add reference
5. Apply
6. See the assigned city/parking area on the dedicated field



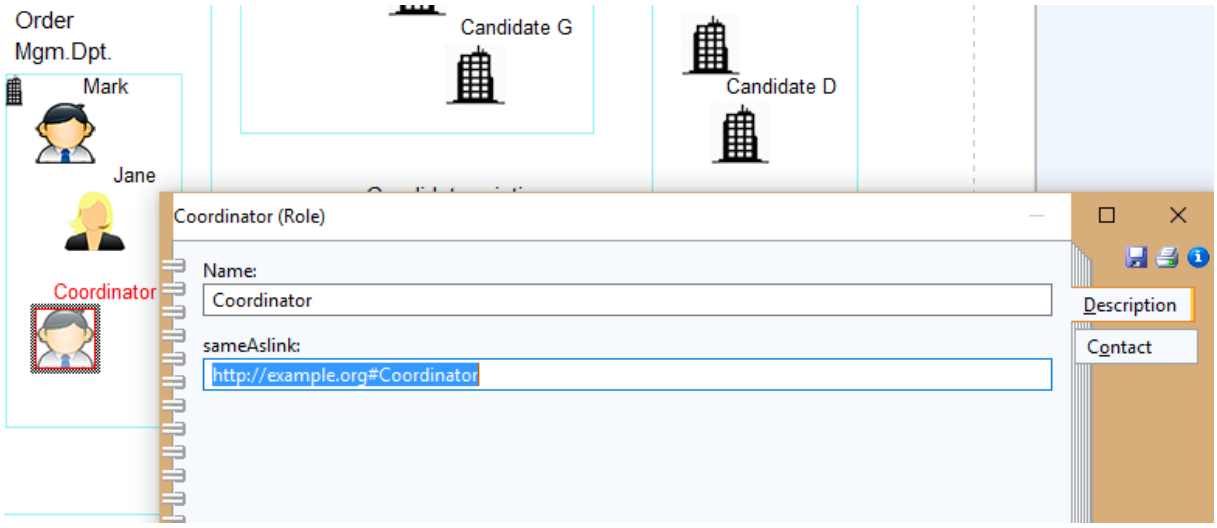
Now there's a number of meaningful hyperlinks that make the model semantics available not only for human interpretation, but also for machine readability and further processing. In other words, the models are not just some drawings to be included in a documentation – they are a "knowledge base" which may be queried.

Part 4. Create Links between Models and External Knowledge Graphs

Besides the links between models or model elements, we may also add some links to external resources, for examples equivalence links to things that already exist in an existing graph database or ontology:
In the "sameAslink" slot provide http://example.org#Jim. We assume that this is the employee identifier that Jim has in some existing human resources database. By doing this, we state that the modelled Jim Smith is the same as the employee in the database having the identifier **:Jim** (we will use http://example.org# as a namespace for everything to keep the example simple).



Next, we assume that, for the company positions/roles, an ontology already exists externally. In that ontology, the remote expert role has the fixed class identifier http://example.org#Coordinator.

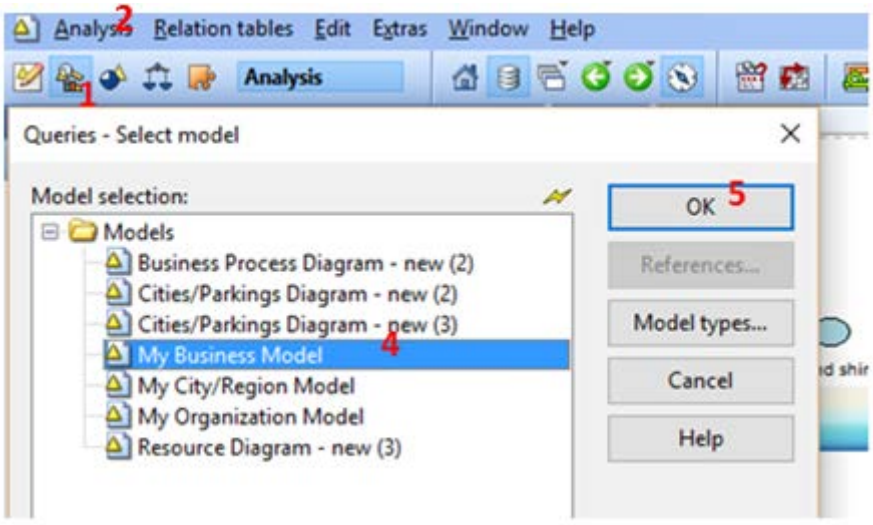


P.S. As you probably guessed, for all model elements, the "sameAslink" allows us to override the identifiers that otherwise would be generated by the modelling tool, with stable identifiers that already exist in some external system, for some of the modelled things (in this examples, we did this for employee identifiers and a role identifier).

Part 5. Querying Models in the Modelling Environment

The "Knowledge Base" formed by models may be analysed through querying mechanisms provided directly in the modelling environment: the AQL language and a query builder providing some query templates. The query feature may be accessed as follows:

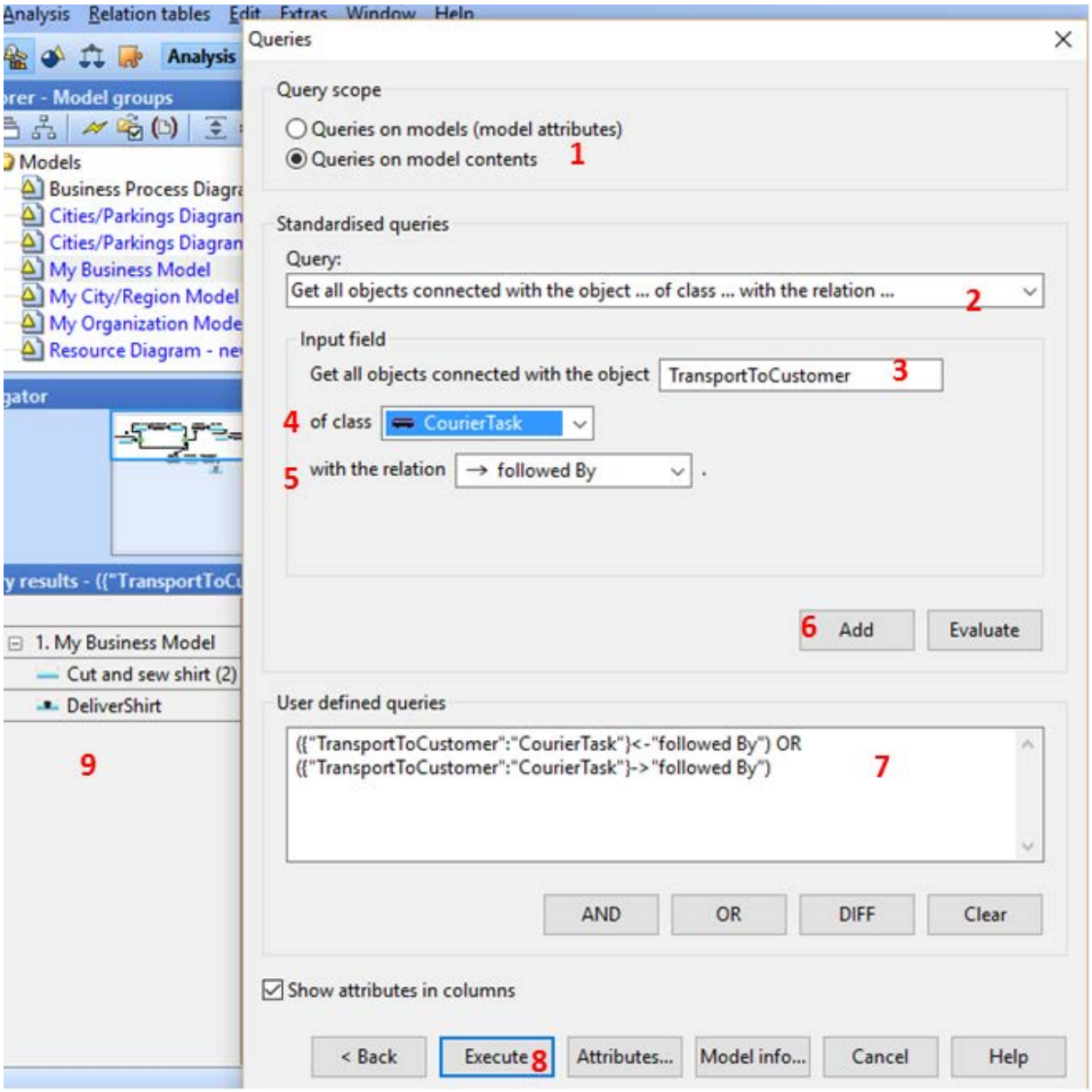
1. Activate the "Analysis" component (from Modelling Toolkit)
2. Open the Analysis menu
3. Select Queries/Reports (not visible in the screenshot)
4. Select the model to be queried (My Business Model)
5. Press OK



The Query window will appear, providing means for both building a template-based query and for writing a query with AQL statements. In the next figure, the query retrieves from the business model all tasks and decisions of TransportToCustomer task in relation to “followedBy”. Notice the steps:

1. Select Queries on model contents
2. Select the template that best fits the intended query
3. Indicate the target object ("TransportToCustomer" types exactly as the name it was given)
4. Select the type of the target object ("CourierTask")
5. Select the type of connector whose targets should be retrieved ("followedBy")
6. Press Add
7. In the box below you will see the AQL query generated by the template. This can be customized by those who know AQL (full documentation is available at https://www.adoxx.org/live/adoxx-query-language-aql)
8. Press Execute

9. You should see the query results



More complex queries may be constructed with the AQL language. We only provide here one examples:

(<"Employee">[?"fulfils" = "REF mt:\"Resource Diagram\" m:\"My Organization Model\" c:\"Role\" i:\"Big Car Driver\"
"])
This query will retrieve all the employees with the role “Big Car Driver”

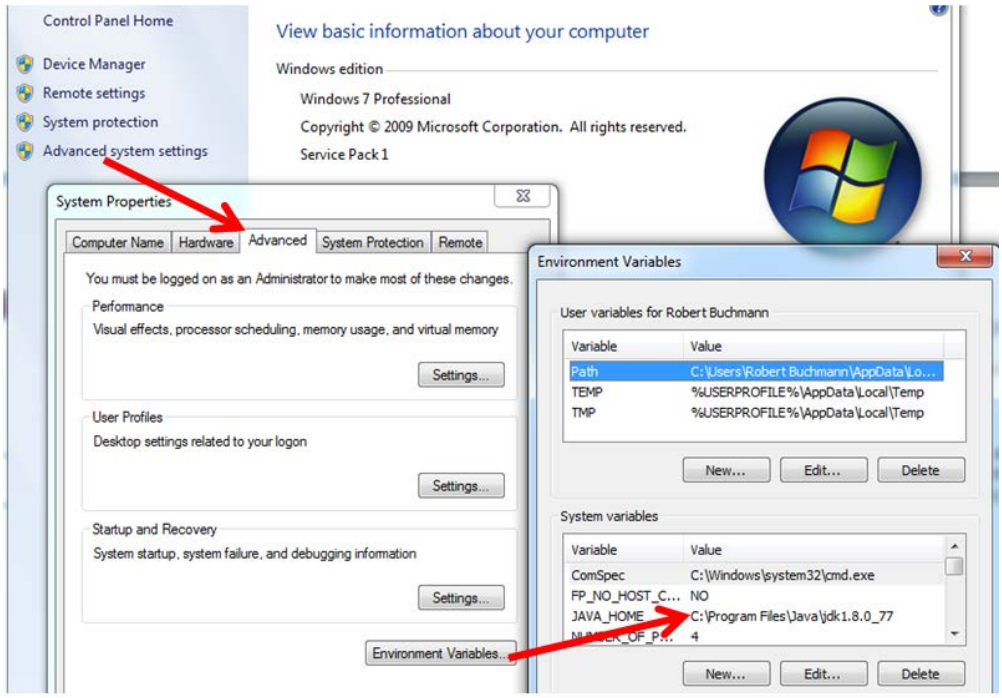
However, AQL queries are limited to model analysis within the modelling environment. In the next step we will employ semantic technology to expose model contents to external systems – specifically, this can be achieved with the Resource Description Framework and its graph query language SPARQL.

Part 6. Setup a Graph Database with OWL Ontological Capabilities

In the following we will use the GraphDB server due to its RDF graph database management capabilities, including support for OWL ontologies and inferences.

Some preparation is required after the download (https://ontotext.com/products/graphdb/):

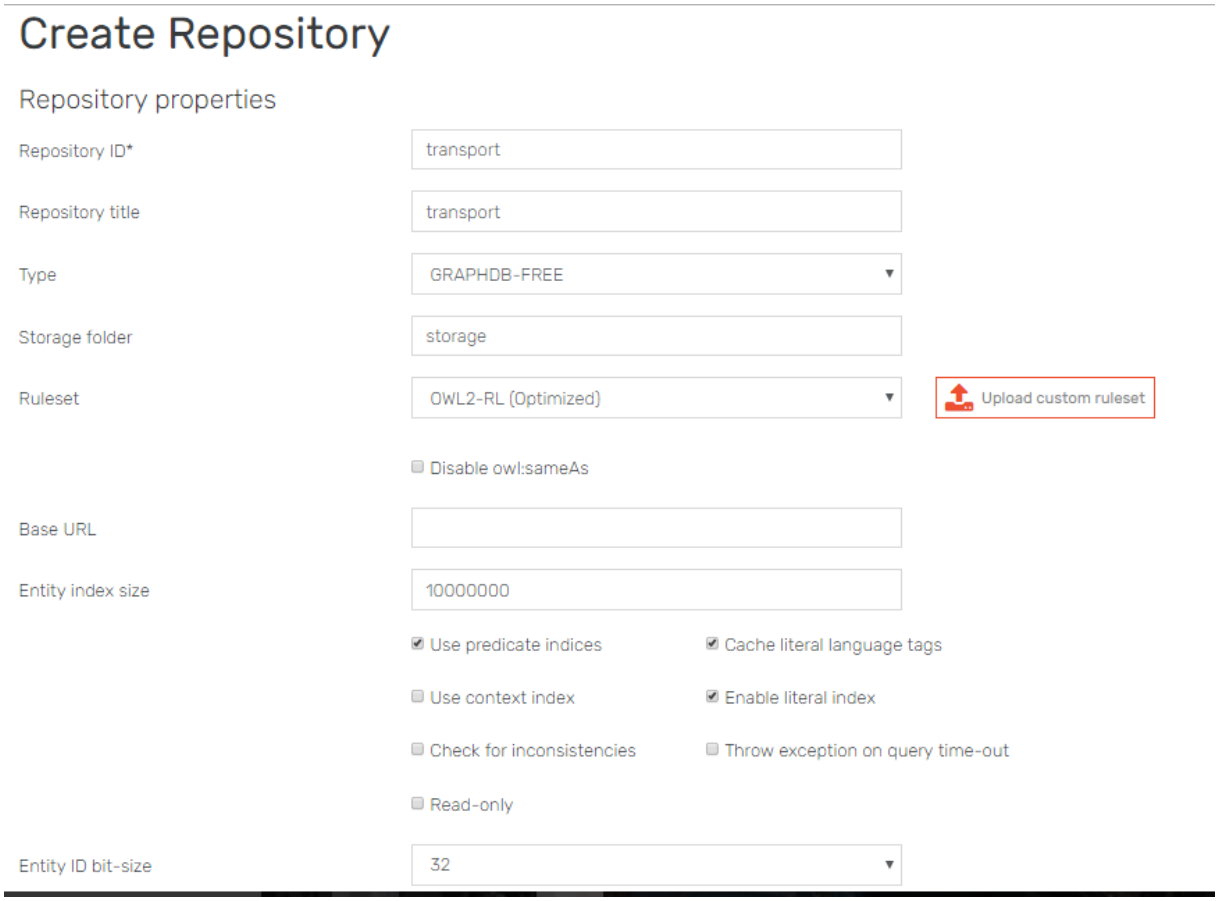
Make sure that Java is properly installed and configured. This means that after installation, you need to create an Environment Variable to indicate where you installed Java. This variable should be named JRE_HOME (if you installed a JRE variant of Java) or JAVA_HOME (if you installed a JDK variant of Java). On a Windows 7 machine with JDK installed this is set as shown in the figure:



If you downloaded the standalone version of GraphDB, no installation is required. Just go to the folder where you extracted its folder, go to BIN and run the file graphdb.cmd.

After it starts, use the browser to access the server interactive console at http://localhost:7200.

Select "Create a repository" and use the settings visible below:



This will create a graph database with OWL capabilities, thus also allowing us to add ontological axioms (based on the OWL2-RL profile and inference patterns).

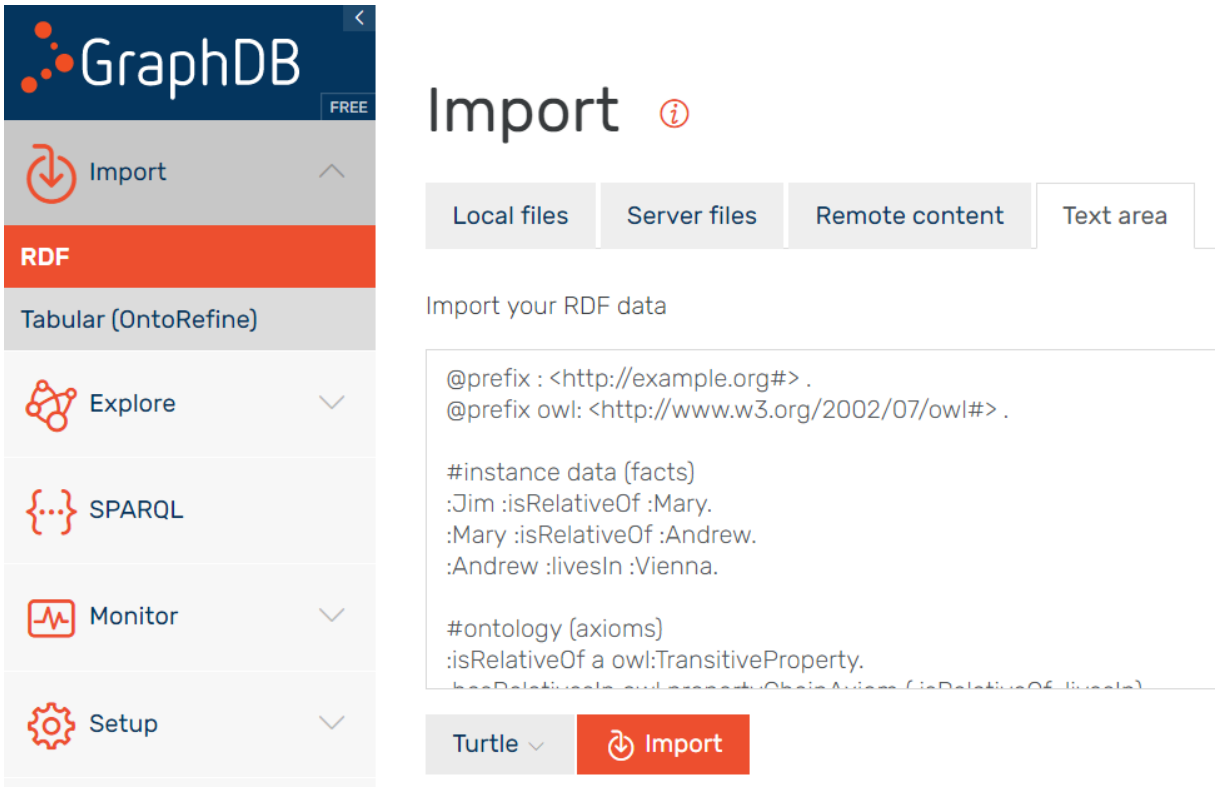
Let's add some minimal sample data to become familiar with graph queries.

@prefix : <http://example.org#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

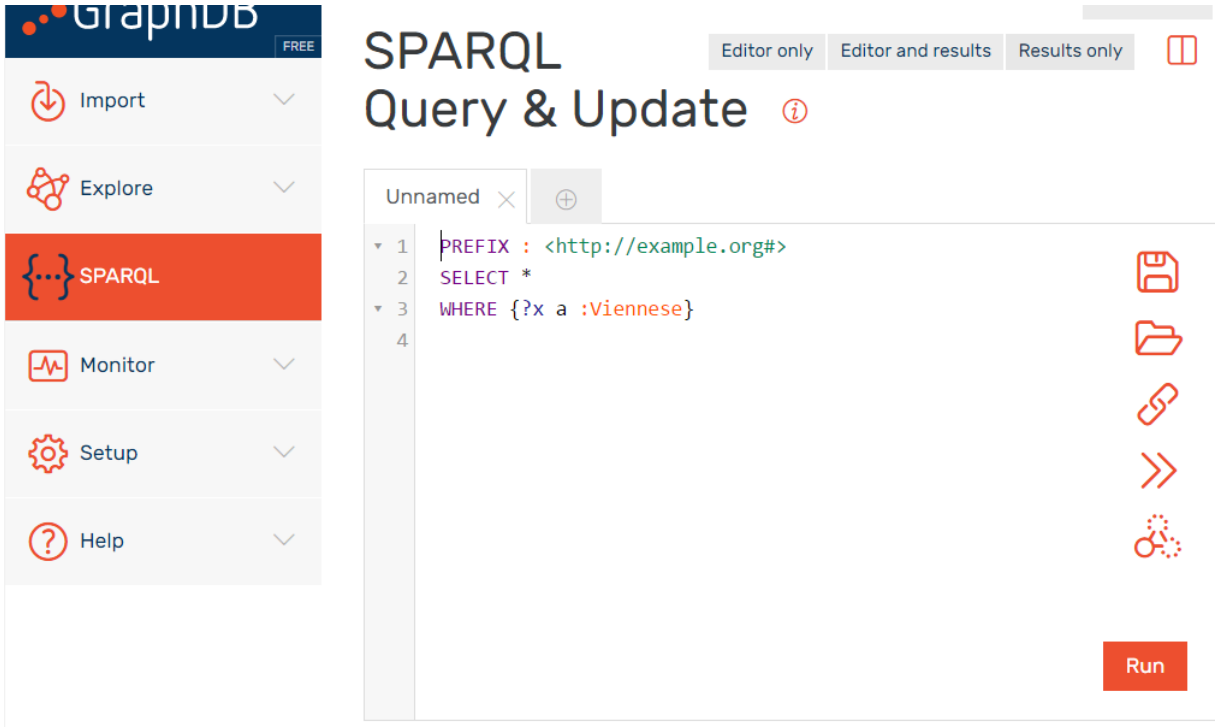
#instance data (facts)
:Jim :isRelativeOf :Mary.
:Mary :isRelativeOf :Andrew.
:Andrew :livesIn :Vienna.

#ontology fragment (OWL axioms)
:isRelativeOf a owl:TransitiveProperty.
:hasRelativesIn owl:propertyChainAxiom (:isRelativeOf :livesIn).
:Viennese owl:onProperty :livesIn; owl:hasValue :Vienna.

Use the Import->RDF option in Graph DB, as shown below. Copy-Paste the data in the Text area Tab and press Import (twice).



Use the SPARQL option to run queries.



Queries:

Retrieve the list of all Viennese:

PREFIX : <http://example.org#>

```
SELECT ?x
WHERE {?x a :Viennese}
```

Who has relatives in Vienna?

```
PREFIX : <http://example.org#>
SELECT ?x
WHERE {?x :hasRelativesIn :Vienna}
```

Notice that although we did not add explicitly statements such as "x a :Viennese" or "x :hasRelativesIn :Vienna", they were generated by reasoning, based on the uploaded axioms and instance data.

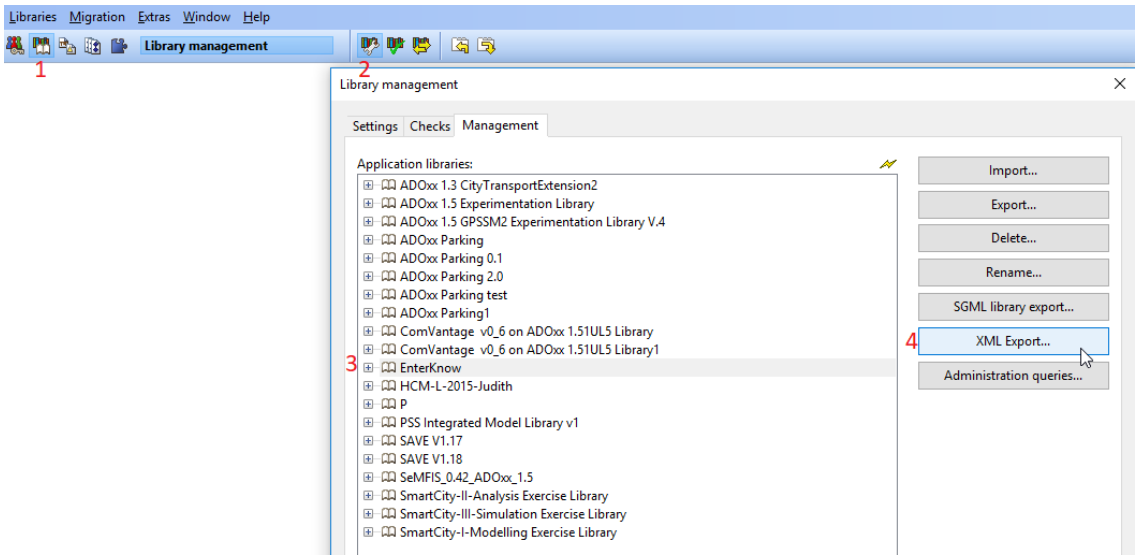
Next, such mechanisms will be applied to diagrammatic contents. First, we need to export models into RDF graphs and upload them in GraphDB.

Part 7. Export models

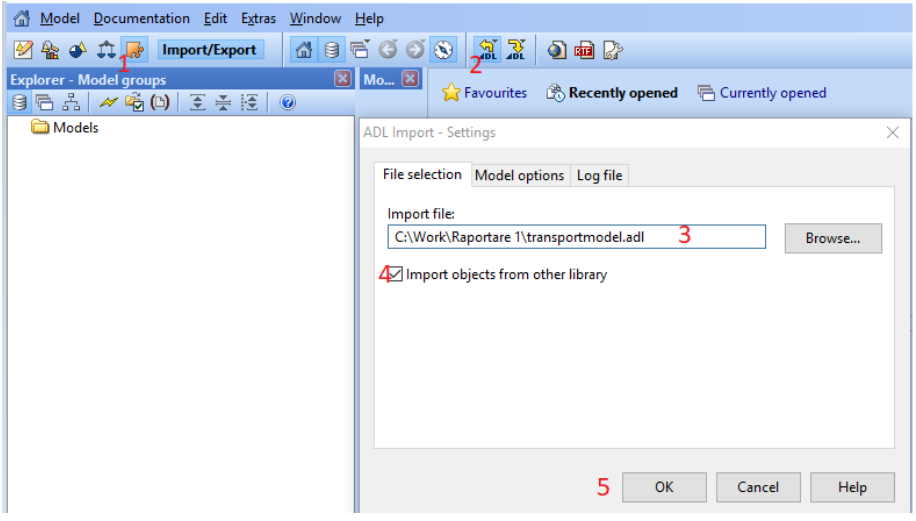
If you want to go all the way through this conversion step, further read the details in this section. The export steps allow are the following:

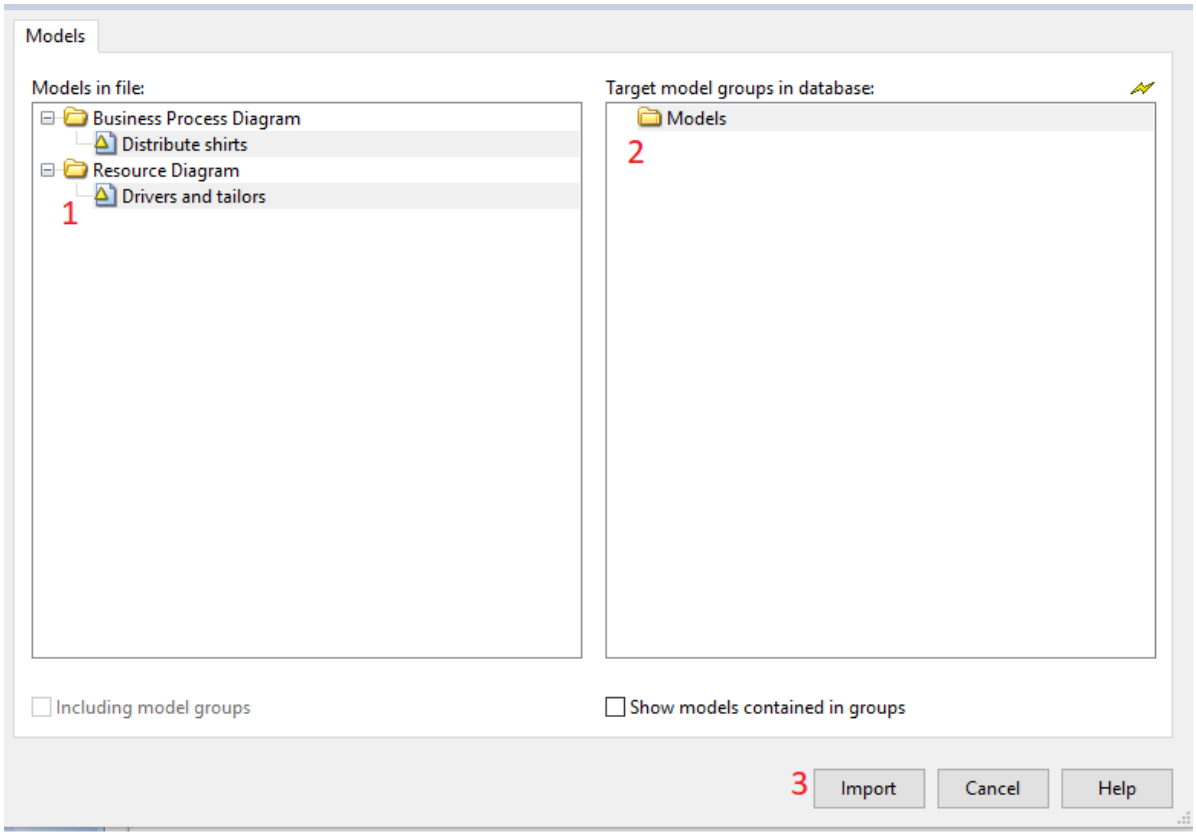
1. Go to the ADOxx Development Toolkit and export the modelling language definition as XML
2. Go to the modelling tool and export the model contents as XML
3. Use the RDF transformer (provided in the archive) to transform the models from XML to RDF. In the following screenshots we detail these steps.

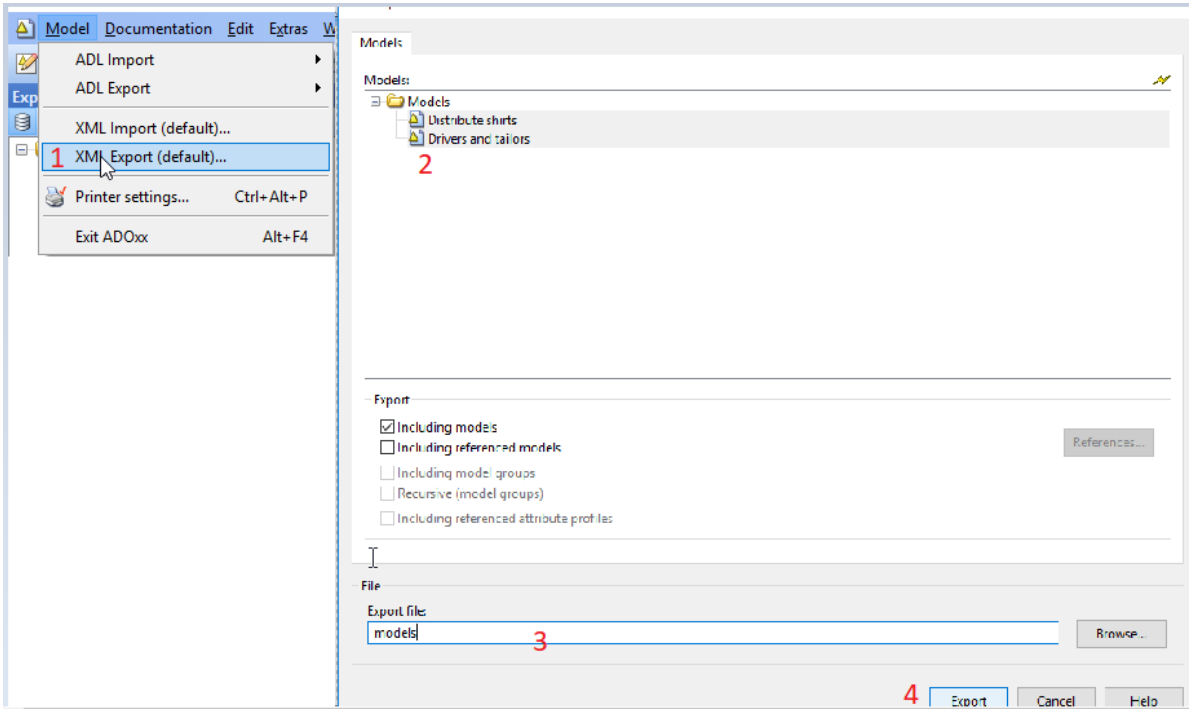
Step 1: Start the ADOxx Development Toolkit, go to Library Management, select the Transport (EnterKnow) implementation source and select XML Export.



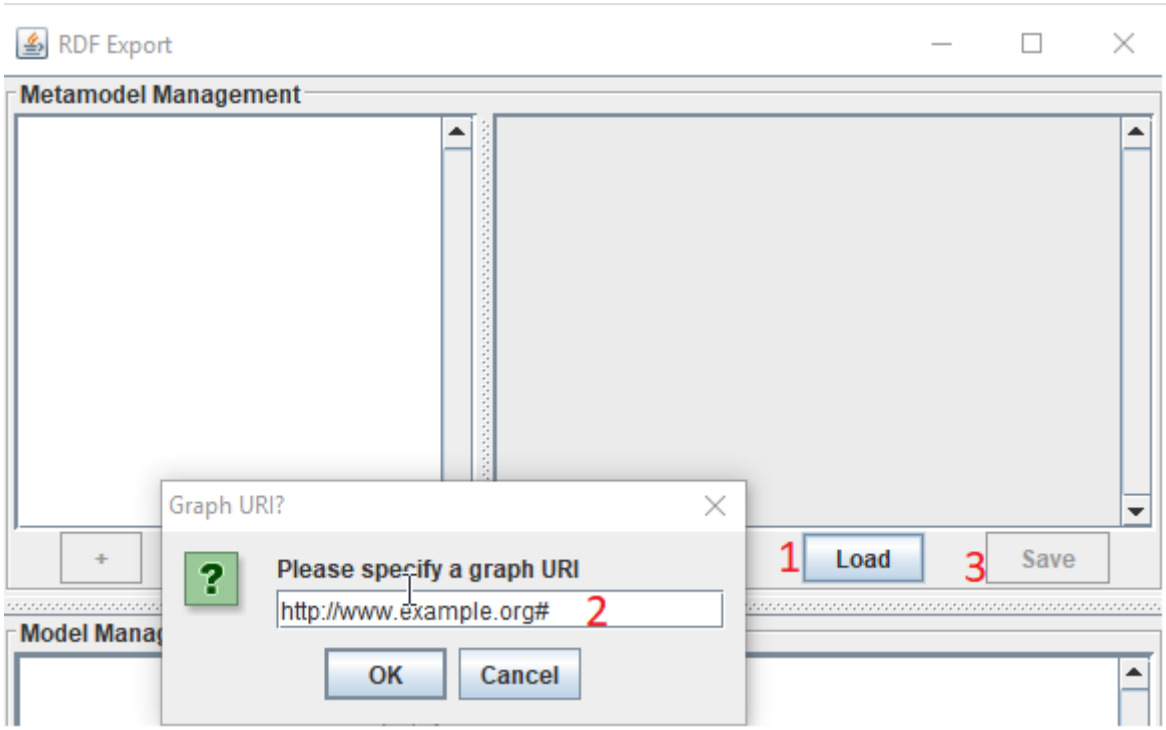
Step 2: Go to the modelling tool, select the Import/Export component and go to the Model menu. At first, import the ADL file (ADL Import) attached in the archive to have the models that will be used for the application (transportmodel.adl), then select the XML Export (not visible in the screenshot). Select **all of** the models (all of them will be saved in the same file!), use the Browse button to set the target file and press Export.



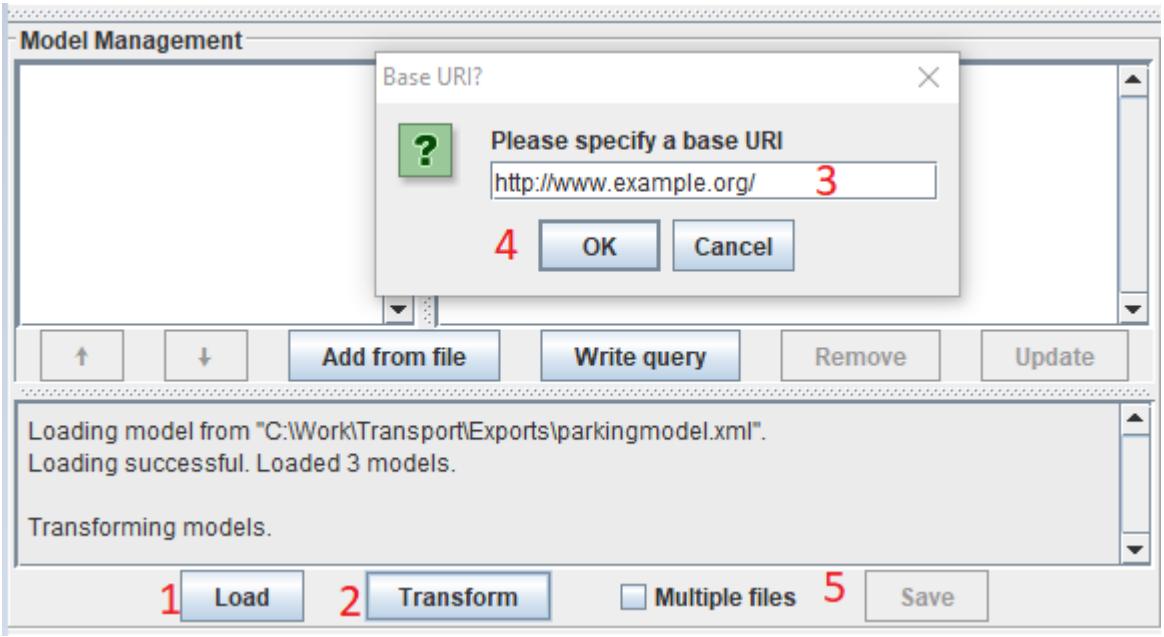




Step 3: Execute the RDF Transformer and use the top half of its window to load the language vocabulary (the XML file exported from ADOxx Development Toolkit). You will be prompted to specify a URI for the graph where the language vocabulary will be stored.



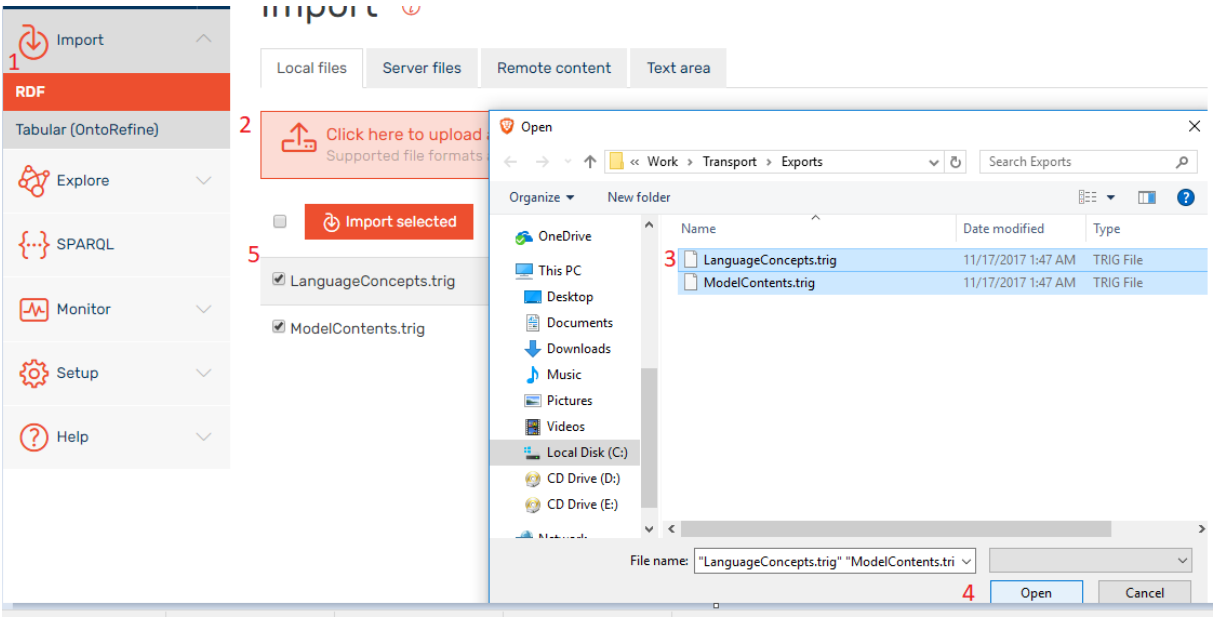
The URI in the figure will also be used in the queries, so type the same to keep things simple. After setting the URI, the properties discovered in the models are displayed. The users have the chance of disabling the properties that they don't want to export.
Use the Save button to save the language vocabulary, preferably in the TriG format.
After the language was exported, use the Load button in the bottom half of the Window to do the same for model contents. Use the Transform button to convert the XML to RDF. Again you will be prompted to customize the URI for model elements (use the one in the figure for simplicity). In the end use the Save button, again with TriG as file format.



Important: For the URI, use the ‘ / ‘ character instead of ‘ # ‘. Otherwise, there will be problems later at application level.

Part 8. Processing the model contents in GraphDB

For this step, please go to the location where you saved the export files. However, this time select the TriG format instead of Turtle, before pressing Import (a Trig file contains multiple graphs, a Turtle file may contain a single graph; in our export case, each model is a separate graph).



The statements should now be imported and queryable from GraphDB.
Let's try some queries to make sure the data was imported correctly:

Query 1: Get all the tailors:

```
PREFIX : <http://www.example.org#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?y where {
?x  :Name 'Tailor' .
?y :fulfils ?x .
}
```

The result should be (different ID numbers):

1	:Employee-50400-Ioana
2	:Employee-50403-Geta

Query 2: Get all the tailors along with their phone numbers and email addresses:

```
PREFIX : <http://www.example.org#>
```

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?y ?z where {
?x  :Name 'Tailor' .
?y :fulfils ?x .
?y :Phone_Number ?z . }
```

The result should be:

	y	z
1	:Employee-50400-Ioana	0747384738
2	:Employee-50403-Geta	0743937477

Query 3: Get the task assigned to the tailors:

```
PREFIX : <http://www.example.org#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?y WHERE
{
?x :Name "Tailor" .
?y :Assigned_role ?x .
}
```

The result should be:

	y
1	:BusinessTask-49809-Tailor_sweaters

Query 4: Get the task that follows after the one resulted from Q3 (change the ID – bolded in the query- according to the result from Q3):

```
PREFIX : <http://www.example.org#>
PREFIX example: <http://www.example.org/>
SELECT ?x WHERE
{example:BusinessTask-49809-Tailor_sweaters :followed_By ?x}
```

The result should be:

1	:CourierTask-58263-Take_sweaters_to_warehouse

Part 9. Adding ontological axioms (OWL)

For the application level to work, it is necessary to have direct relations between the tasks, as they can be seen in the models. Because the “followed by” arrows also have properties, they are not exported as predicates between the subject and object of the statement, but as entities mapped to the instance they come from to the instance they go to. For the tasks to have direct relations, it is necessary to use OWL ontological axioms which automatically create statements based on existing ones. In the following lines we will see a few examples of such axioms which create this relationship.

```
@prefix cv: <http://www.comvantage.eu/mm#> .
@prefix : <http://www.example.org#> .
 :source owl:inverseOf cv:from_instance.
 :target owl:inverseOf cv:to_instance.
 :followed_By owl:propertyChainAxiom (:source cv:to_instance).
```

These axioms create bidirectional relationships, creating inverses for some predicates (inverses will be explained in the next example), while also generating the followed_By predicate when finding a succession of occurrences (a source task that is mapped to a target task by the ‘to_instance’ predicate).
It would also be possible to create this relationship by using SPARQL Insert queries, but these would need to be executed at each import, while these axioms are rules that will apply every time new statements are added. The SPARQL query for this purpose would be:

```
PREFIX : <http://www.example.org#>
PREFIX cv: <http://www.comvantage.eu/mm#>
INSERT {?x :followed_By ?y} WHERE {?node cv:from_instance ?x; cv:to_instance ?y; a :followed_By}
```

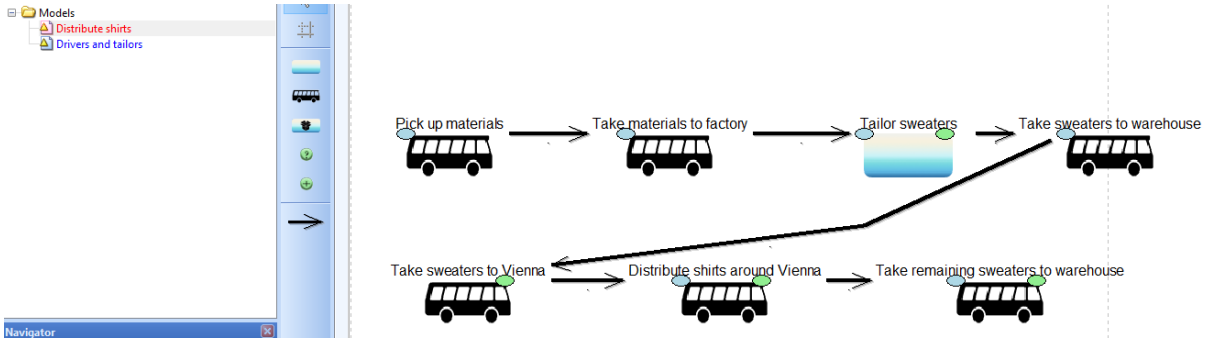
As it can be seen in the previous section, the export creates one-way relations, while for the application it would be useful to have bidirectional ones in order to have the information when accessing each URI involved in the statement.
For this, we will also use OWL ontological axioms. It is useful to have bidirectional relations for two types of relations: Followed By, which says what task needs to be performed after the subject, and Assigned Employee, which specifies which employee is responsible for the subject.
This can be achieved by using the owl:inverseOf property, which creates properties for the object of a statement based on the predicate.
More specifically, for the “followed_By” property we will generate “preceded_By” properties, meaning that an object that follows another is also preceded by the one it follows, which will be useful when dereferencing terms. The same is available for the “Assigned_employee” property, so when a task has an assigned employee for it, the database also stores the information that the employee is responsible for that task. It is

possible to add many more axioms in order to make the application even richer semantically, these are a few examples needed at this stage of the implementation.

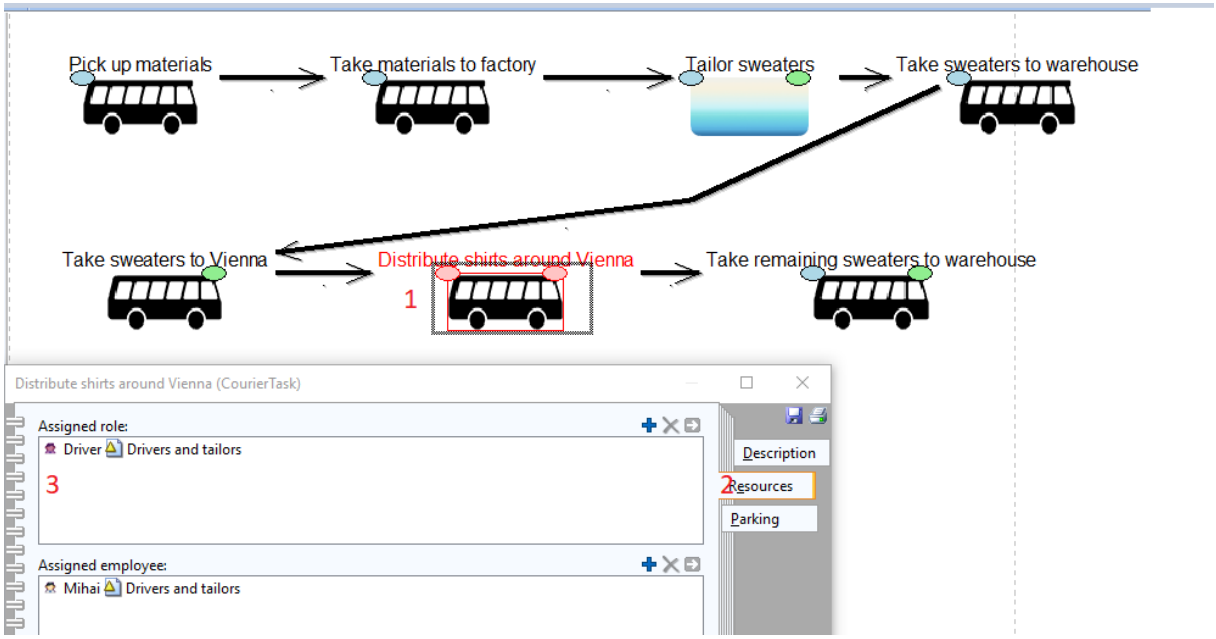
@prefix : <http://www.example.org#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
:preceded_By owl:inverseOf :followed_By .
:responsible_For owl:inverseOf :Assigned_employee .

Part 10. Understanding the models

Before going further to the application level, let's take another look in the modelling tool to understand exactly what properties have been exported and how the workflow management application needs to manage the information.
Step 1: Log into the modelling tool with the user assigned.
Step 2: Access the "Distribute shirts" diagram.



Step 3: Let's take a look at a task, for example "Distribute shirts around Vienna", and see what properties it has, remembering the types of information resulted from the queries.



This task is assigned for a Driver, but is also sets a specific user to do this specific Task (the driver specialised for the Vienna route). At step 3, double click on “Mihai” and close this window.

At this point, we see this employee in the Resource Diagram. Double click on the highlighted employee to see the types of data queried earlier.

As these models are already exported to the database, we can go forward and set up the application.

IMPORTANT: For now, if you wish to create and export new models to the application following the steps explained, please only used the types of Tasks/Employees that can be seen in these pictures, as the other ones are not yet available in this early iteration draft.

Part 11. Setting up the application

Before running the jars, we need to set up the localhost to respond to “http://www.example.org”, otherwise this would access an external address which does not respond with our data.

Step 1: Go to C:\Windows\System32\drivers\etc and open the hosts file with an editor.

Step 2: Write the following line after the commented ones:

127.0.0.1 www.example.org

```
# localhost name resolution is handled within DNS itself.
#   127.0.0.1       localhost
#   ::1             localhost

127.0.0.1       www.example.org
```

Step 3: Save and close the files. If it requests administrator rights, reopen it as an administrator, do step 2, and save.

Now the URI dereferencing service used to get data from the database at application level should work. Do the following steps:
Step 1: Go to the folder where you saved the elements from the archive and open a command line (shift + right click, choose the option to open command line).
Step 2: In the command line, write the following command:
java -jar dereference.jar

There should be a long log and the last line should look like this:

```
2017-11-18 18:03:11.725  INFO 24448 --- [           main] Dereference.Application                  : Started Application
 in 5.99 seconds (JVM running for 6.968)
```

Step 3: Go to the GraphDB query endpoint and do the following query:
PREFIX : <http://www.example.org#>
SELECT ?x WHERE
{?x :Name "Mihai"}

For us, the result is: :Employee-49815-Mihai . If it is another, copy it from there, but without the “ : “ .
Step 4: Go to a browser of your choice, and add it to http://www.example.org/ .
For example: http://www.example.org/Employee-49815-Mihai .The browser should respond like this:


```
[ {
  "@id" : "http://www.example.org/BusinessTask-49840-Take_sweaters_to_Vienna",
  "http://www.example.org#Assigned_employee" : [ {
    "@id" : "http://www.example.org/Employee-49815-Mihai"
  } ]
}, {
  "@id" : "http://www.example.org/BusinessTask-49843-Distribute_shirts_around_Vienna",
  "http://www.example.org#Assigned_employee" : [ {
    "@id" : "http://www.example.org/Employee-49815-Mihai"
  } ]
}, {
  "@id" : "http://www.example.org/BusinessTask-49846-Take_remaining_sweaters_to_the_warehouse",
  "http://www.example.org#Assigned_employee" : [ {
    "@id" : "http://www.example.org/Employee-49815-Mihai"
  } ]
}, {
  "@id" : "http://www.example.org/Employee-49815-Mihai",
  "@type" : [ "http://www.w3.org/2002/07/owl#Thing", "http://www.comvantage.eu/mm#Instance_class", "http://www.example.org#Employee" ],
  "http://www.comvantage.eu/mm#described_in" : [ {
    "@id" : "http://www.example.org/Resource_Diagram-Resource_Diagram_-_new_"
  } ],
  "http://www.example.org#E-Mail_Address" : [ {
    "@value" : "mihaidriver@example.org"
  } ],
  "http://www.example.org#Name" : [ {
```


(more lines)

This means that the dereferencing service the application uses to get data is working.
In order to start the application, it is necessary to have an SQL database that stores login data and connects the SQL users to the RDF data.
Step 1: Open a command line and log in as an admin. it should work with the following command:
mysql -u root -p
The command line will ask for a password, which, by default, is null (just press enter).

Step 2: When logged in, type the following commands:
create database transport;
create user 'tsport'@'localhost' identified by 'pass';
\q
mysql -u tsport -p
(when prompted for password, use ‘pass’ as the password);
use transport
create table user (id int auto_increment primary key, email varchar(45), name varchar(30), password varchar(255), rdfid varchar(30), role int);
insert into user values (1, 'admin@example.org', 'admin', '$2a$10$/DQ7O/oqqJaeVsqqOYx85uLPZrTgIeDUVGBiyRczeIfBHGWJo6TW2', 'noid', 0);
(this is a predefined admin for the application, which will then be used to set it up).

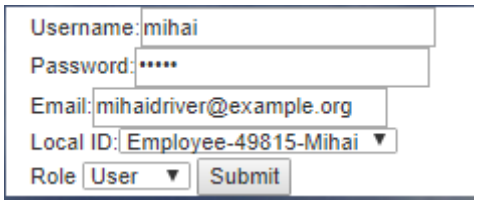
After setting up the database, go back to the archive, and run the following command:
java -jar transport.jar

Now, from the browser, go to localhost:8080/login.html and login with the following credentials:
Username: admin
Password: password
The registration page should appear.

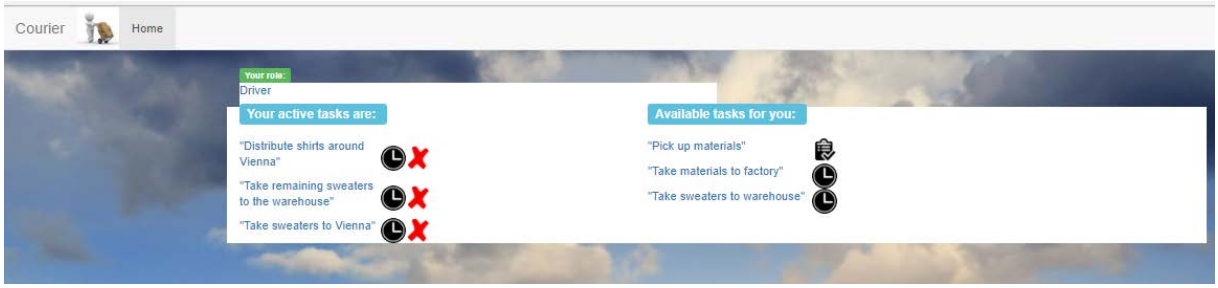
If you wish to create another admin, you can do it from here and delete the one that already exists from the database.

What we need to do is to create a user that matches the local ID of an Employee from the Graph Database. In this case, it is mandatory to create users for two employees, because they have tasks that are already assigned: Ioana (Tailor), and Mihai (Driver). Select the Local ID from the dropdown list.

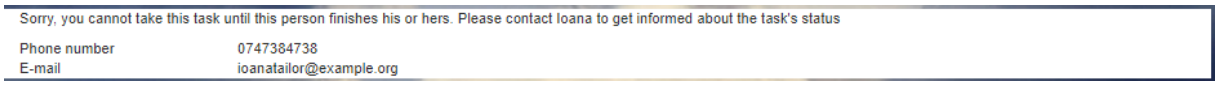
Add a username and password, which will be the credentials, the Local ID, and select Employee from the dropdown list. It should look like this:



Do this for both users.
Click submit, then access http://localhost:8080/login.html again.
Let’s log in as the driver (Mihai). Enter your credentials and hit the login button. You should be prompted with the following page:



Pending task: It cannot be taken as active or completed yet, because another task needs to be performed before it. Some appear as active because they were assigned from the modelling level; if a user drops it, he/she cannot take it again until it is available. By clicking this button, the application responds with contact information about the user/department assigned for the task that precedes this one, so the user can enquire about the status of that task. The responses look like this:



Give up task: When a user decides he/she cannot complete the task anymore, this option makes the task available again for other users.

Take task: By clicking this button, a user takes this task as active, making it unavailable for other users (the active tasks are only visible for the user that is responsible of achieving them, the available tasks are visible for all of the users).

Complete task: This task is active and can be completed, so it is removed from the list of active tasks and the application writes data to the database about the task's completion.
Sometimes, a user might be in the situation when all of the tasks are pending, which means that another department has to finish a task before he/she can take another one (for example a Driver must wait before a Tailor tailors the sweaters before taking them to the warehouse, moving them to Vienna and so on).
The purpose is to finish all the tasks for each process. A user cannot see what processes are the tasks described in (company privacy issues), but when all of the tasks from a process are finished, the application writes to the database that the process is done. This can be viewed in the Administration Page when logging in as an admin, along with the time of fulfilment and the finished processes, which are the groups of tasks described in the models.