SemQuu

User Guide & Documentation

Version 1.0

Please cite as follows:
Fellmann, Michael (2012): SemQuu – User Guide & Documentation. Report on the Semantic Model and Query Utility (SemQuu), developed as part of the dissertation "Semantic Process Engineering – Konzeption und Realisierung eines Werkzeugs zur semantischen Prozessmodellierung" at the University of Osnabrueck. Online: www.semantic-business.org/semquu/UserGuide_v1.pdf.

Contents

1 Introduction

Semantic process modelling as presented here is about connecting the elements of a business process model to the network of machine processable knowledge contained in an ontology. SemQuu provides a showcase of how this connection can be established and what can be done when this connection is established. That is, in a nutshell: The full range of description logic and semantic web technologies can be applied to business process modelling and analysis in a rather usable and friendly way.

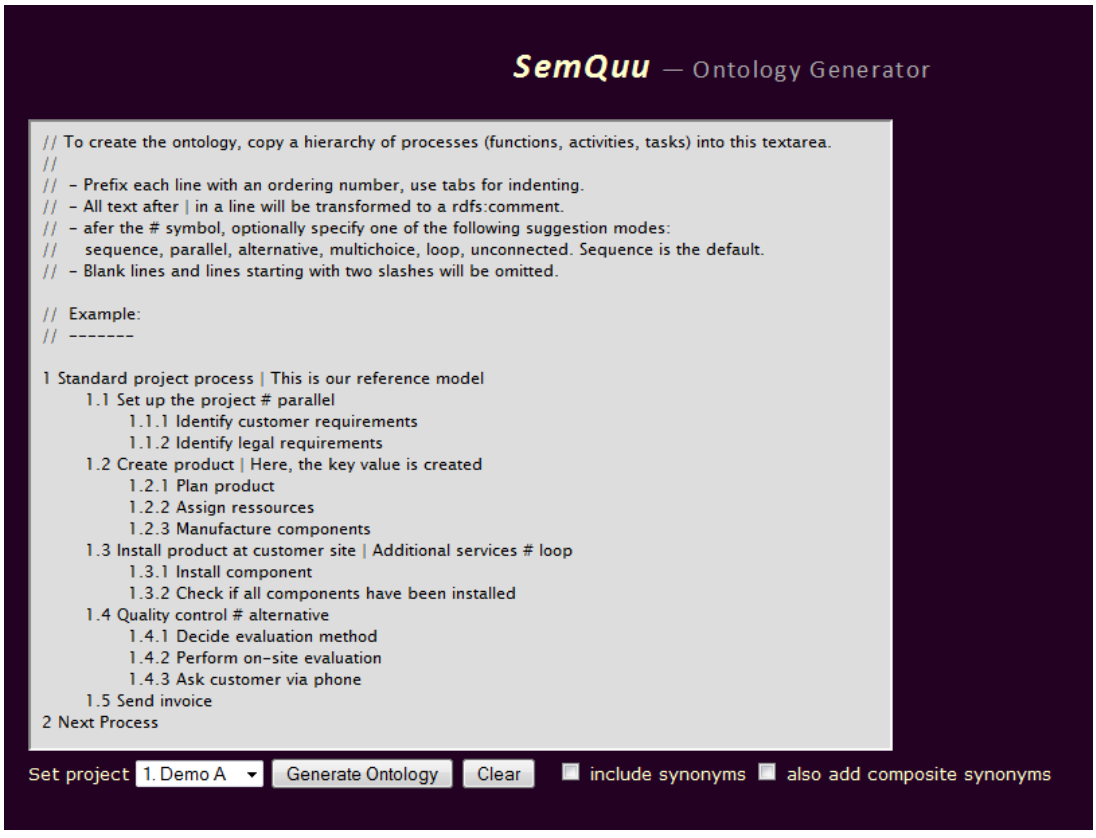
The purpose of this document is threefold, to (a) showcase the functionality and provide step-by-step instructions how to use them to provide, (b) provide a set of test cases to show the regular and intended functionality of the SemQuu prototype and (c) to provide samples of generated data.

2 Ontology Creation with OntGen

2.1 ONTG-1: Generate an Ontology

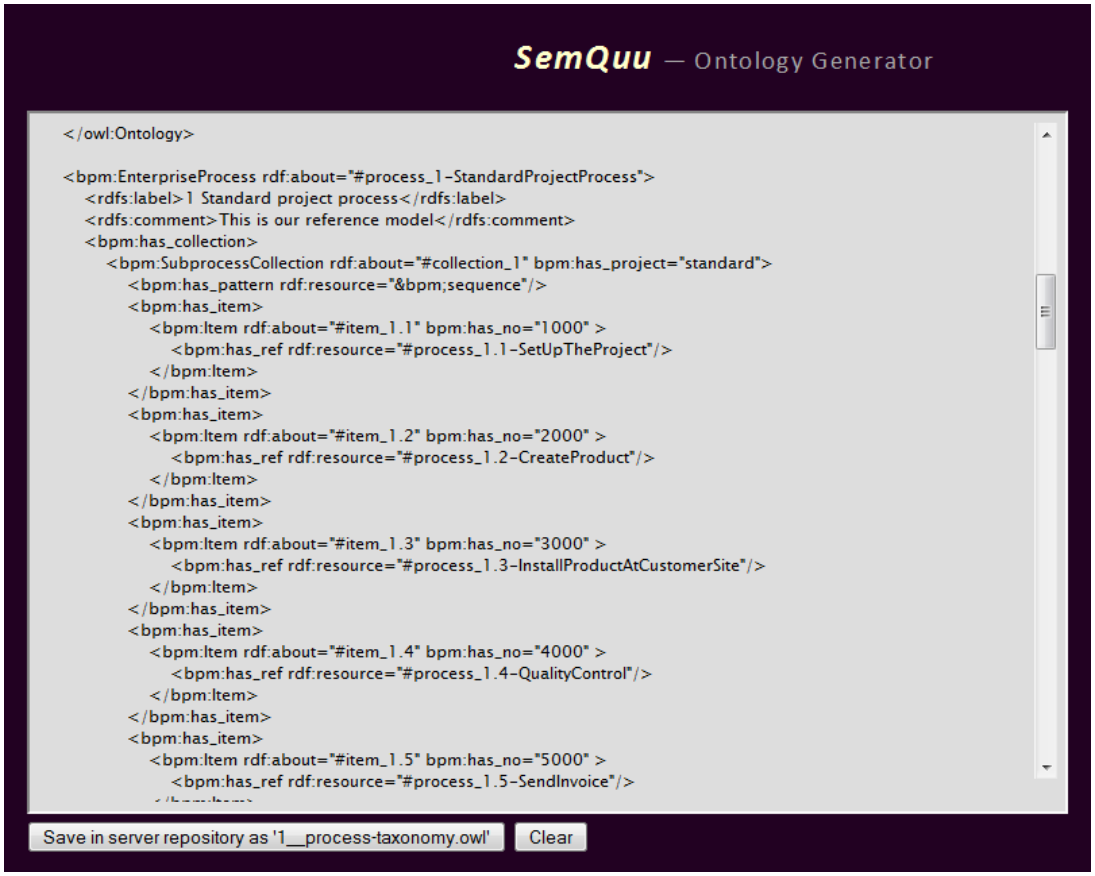
The purpose of OntGen is to facilitate and speed up ontology construction.

Step 1. Start SemQuu Server and point your browser to http://localhost:8080/ontgen. The following screen will appear.



As a default, a simple taxonomy for ontology generation is displayed. From this taxonomy, a simple ontology can be generated in the next step. This ontology is named the *SUN*-ontology throughout this document (Simple Understanding of the New Features). The ontology is intended to be used to explore the semantic annotation features of SemQuu.

Step 2. At the bottom, you may change the project for which the ontology is generated. Check the option “include synonyms”. If this option is set, you can use synonyms when grounding elements of the process model (see ANNO-2). Then Press “Generate Ontology”. The generated ontology will appear as it is shown in the following screenshot.

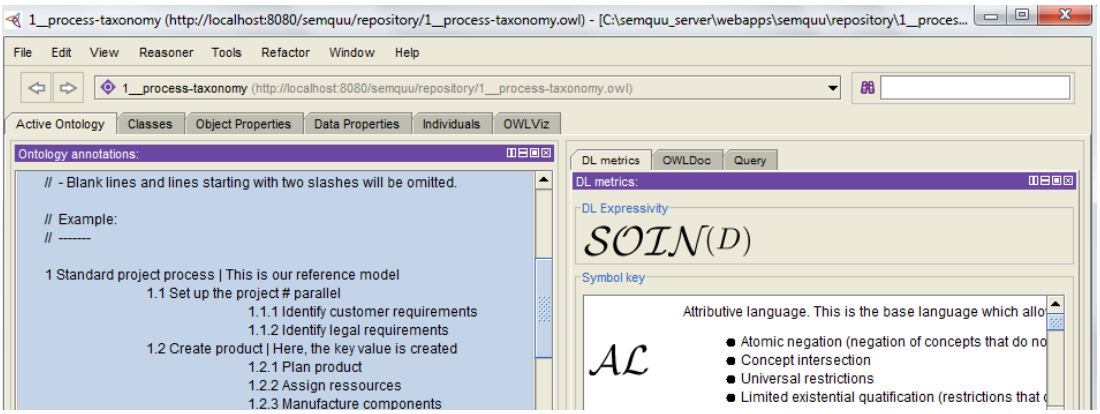


The original text for generating the ontology is included in a CDATA-section in the `rdfs:comment`-element at the beginning of the generated ontology text. This will allow the later reuse of the text to re-generate the ontology if the need arises. The generated data for the SUN-ontology is shown in Appendix 2.

Step 3. Press the „Save in server…“ button. After this, in the textarea a message should appear “Sent content to server…” with the location of the file which has been generated.

Note: The old process taxonomy ontology is backed up automatically on the server in a file with a timestamp in the file name indicating the date and time of backup.

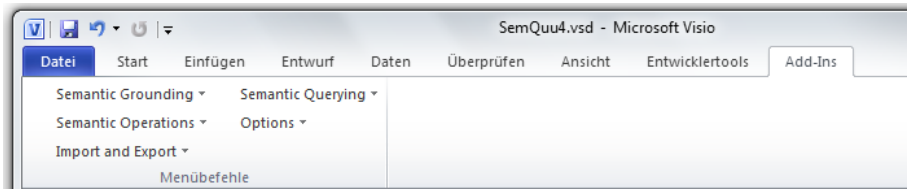
Step 4. To ensure that the generated ontology is correct, open the ontology with the Protégé Ontology Editor 4.x. In the “DL metrics” tab, the expressivity of “SOIN(D)” should be shown (see figure on the next page which shows the Protégé-editor). Use an inference engine such as FACT++ to ensure the validity of the generated ontology.



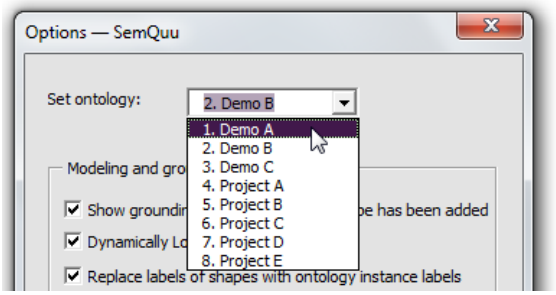
2.2 ONTG-2: Import the Ontology for the SemQuu Add-In

In order to be usable for the grounding of model elements, the ontology must be imported into the SemQuu Add-In.

Step 1. Open the "SemQuu_v<No>.vsd" file with Visio. Activate the ribbon "Add-Ins". The following menu of the SemQuu Add-In appears.



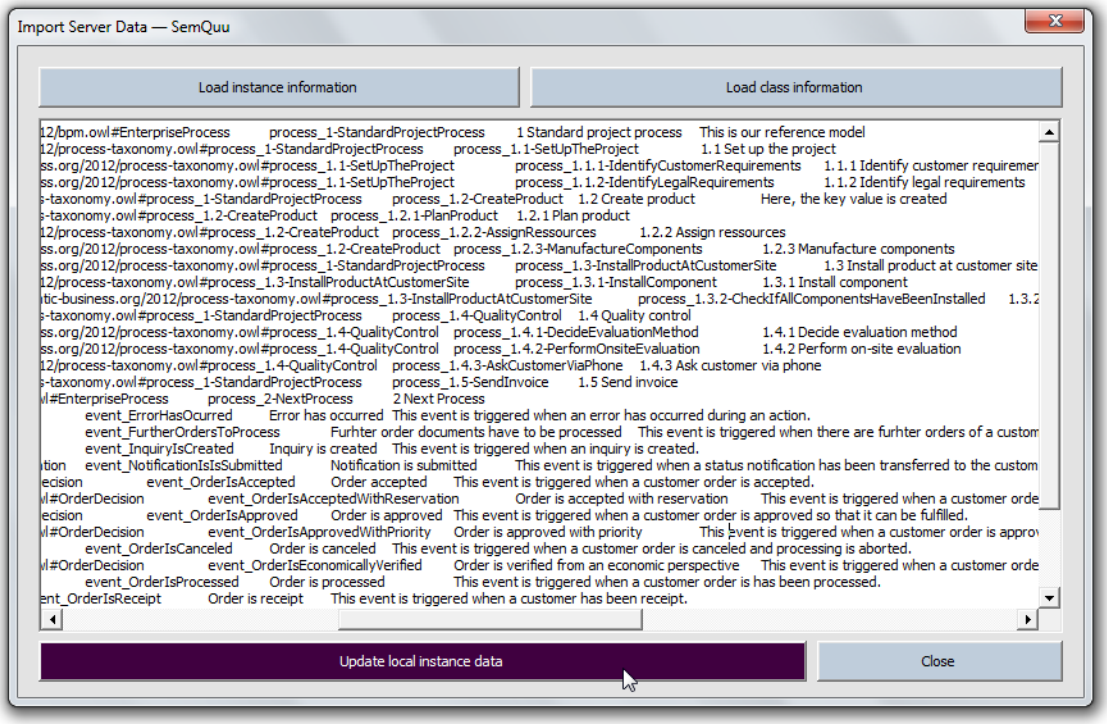
Step 2. Optionally, set the ontology for which the data should be imported. To do so, go to the "Options" menu of the SemQuu Add-In and select "Show options…". The following dialog will appear.



If you have generated data previously with the ontology generator OntGen and now want to use this data, the ontology number (1-8) used to save the data in OntGen should match the number selected in the "Set ontology" drop down field.

Note: Selecting the first ontology "Demo A" will result in importing data from the file ontology "1__process-taxonomy.owl". Selecting the second ontology will result in importing data from the file ontology "2__process-taxonomy.owl" and so on.

Step 3. Go to the "Import and Export" menu of the SemQuu Add-In, then select "Import ontology from server…". A dialog appears which lets you import the data from the server. Press "Load instance information" and "Load class information". After the information is shown in the textarea of the dialog, press the "Update …"-button respectively to save the data.



Note:

- If you try to import data from a non-existing ontology, a message “com.hp.hpl.jena.shared. DoesNotExistException” is displayed inside the textarea.
- In order for the grounding dialog to be updated with the new data, open the dialog and close it using the “x” of the operating system at the topmost right corner. This will cause the data to be read again from the local file system of the client. If the data appears not to be in the current version, restart the server and import the data again in the client.
- Old files are automatically backed up by renaming them with a timestamp indicating the date of creation.

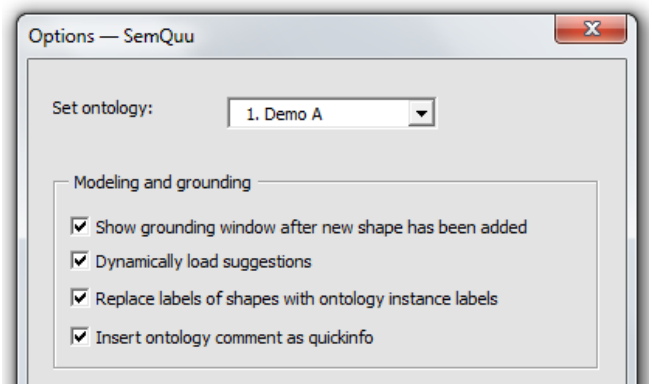
3 Grounding of Model Elements

In the following, it is shown how the model elements can be grounded in the ontology using the SemQuu Add-In. When a model element is grounded in the ontology, an instance of the ontology is linked to the model element which is also referred to as *semantic annotation*.

3.1 ANNO-1: Set Grounding Options

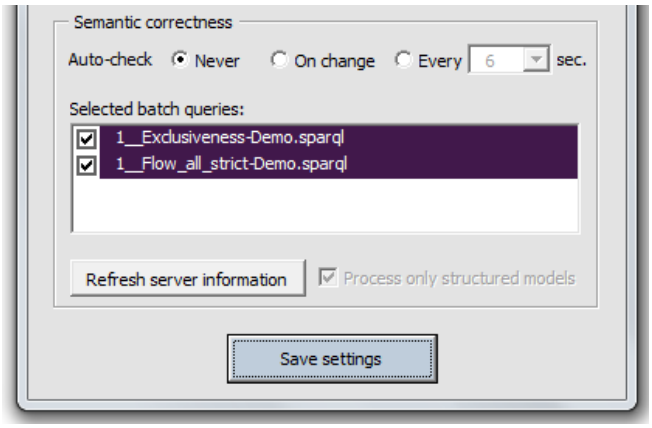
Step 1. Go to the “Options” menu of the SemQuu Add-In, then select “Show options…”.

Step 2. The “Options” dialog has to main areas. In the area “Modeling and grounding”, the following options are available:

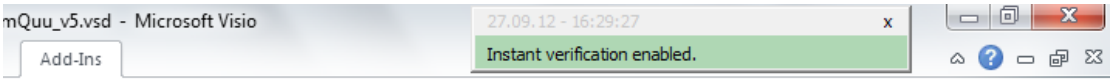


"Show grounding window after new shape has been added" will cause the grounding dialog to appear each time a new model element has been added. The option "Dynamically load suggestions" causes the "Modelling suggestions" dialog to appear automatically each time when modelling suggestions are available. The option "Replace labels of shapes with ontology instance labels" will copy the name of the ontology instance to the model element each time the model element is grounded. The option "Insert ontology comment as quickinfo" will cause the comment of the grounding instance (which his stored in the `rdfs:comment`-element in the ontology) to be copied as a comment for the model element. These comments become visible when hovering with the mouse on a model element.

In the area "Semantic correctness", the following options are available:



The feature "Auto-check" can be triggered with the options "Never", "On change", or "Every <x> sec.". If the first option is selected, then correctness checking will not start automatically and has to be triggered using the menu command.
If the second option is selected, then correctness checking will be executed each time an element has been added or grounded. If the last option is selected, the model is checked in a fixed time interval. To indicate when the next check will be executed, a small modelling status window appears, as shown below. The modelling status window displays messages indicating the status and result of the semantic correctness check.



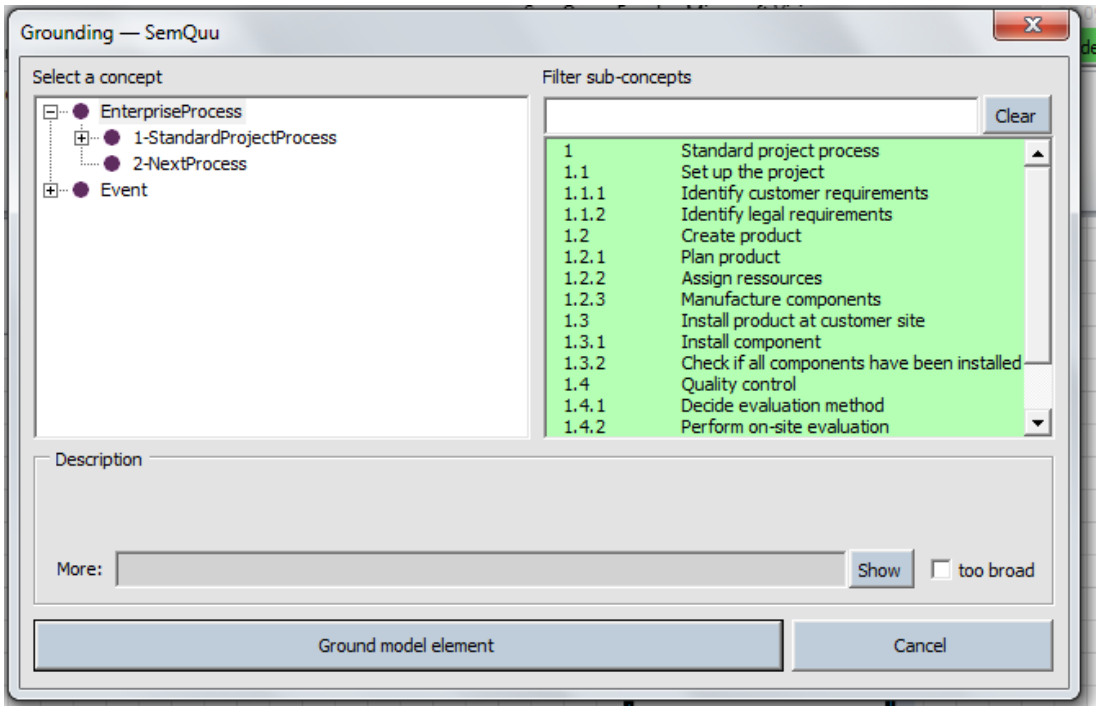
Independent on how the correctness check is triggered, you can specify the batch queries used to check the correctness by checking or unchecking the batch queries shown in the list

"Selected batch queries". The selected queries are stored in local settings. If a new batch query on the server is created, then press "Refresh server information" in order to see the query. If you change the ontology, then the current list of queries is automatically retrieved from the server.

Note: The names of the batch queries on the server must start with "<no>__", where <no> is the number of the ontology. This naming convention is used to associate a batch query to the selected ontology.

3.2 ANNO-2: Ground Model Elements

Step 1. Open the "SemQuu_v<No>.vsd" file with Visio. Create a new page for your model in Visio or use an existing page. Drag a new model element such as a function from the shape collection displayed on the left side onto the document as you do in the usual model construction process. The following grounding dialog appears.

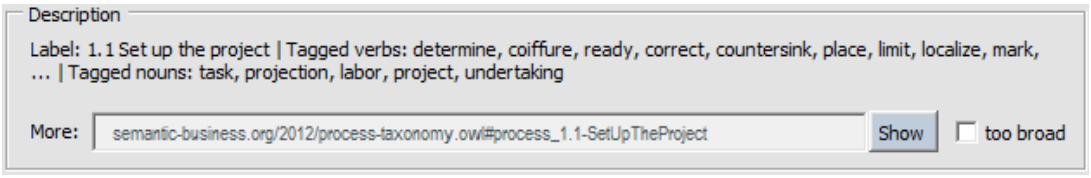


Note: If the dialog does not appear, check the option "Show grounding window after new shape has been added" in the "Options" dialog (see ANNO-1). Alternatively, you can manually activate the grounding dialog using the command "Ground element…" from the menu "Semantic Grounding".

Step 2. To select an ontology instance for grounding, do one of the following steps:
(a) Click on an entry in the leftmost explorer.
(b) Click on an entry in the right list.
(c) Refine the right list by typing a filter text in the input field at the top right position of the dialog. The filter text additionally narrows the list of instances to those containing the filter text. To include synonyms, use the #-symbol in the filter text. If this symbol is present, then the characters before it must match one of the verb synonyms, the characters behind it one of the noun synonyms. For example, to find the instance "1.4.3 Ask customer via phone", type "demand # client" in the input field. Finally, proceed with (b).
(d) First, execute (a), then (b) or (c).

Note:
- When an entry is marked in the explorer, it acts as a filter for the instance list. Only instances which are subordinated to the marked entry in the explorer will appear in the instance list.
- Double-click on an entry in the instance list will mark the corresponding entry in the explorer and expand the explorer.

Step 3. After the ontology instance for grounding is selected, you can read the label, the comment and the associated synonyms in the section "Description" of the grounding dialog. An example of this is shown below for the instance `#process_1.1-SetUpTheProject`.

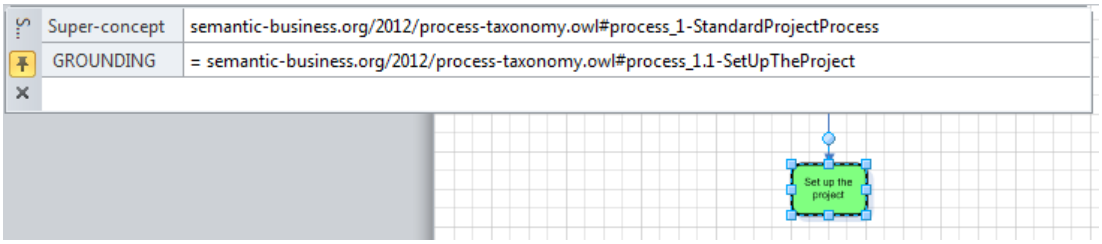


By pressing the "Show"-button, you can open a web browser to read the description of the instance on the web. If the instance is too general and you have a narrower or more specialized notion in mind, check "too broad". In this case, the semantic relation will be "narrower_than" instead of the standard relation "equivalent_to" which is used by default.

Note: Both relations "narrower_than" and "equivalent_to" are sub-properties of the more general "has_annotation" property which is used to represent the grounding of a model element in the ontology-based process representation.

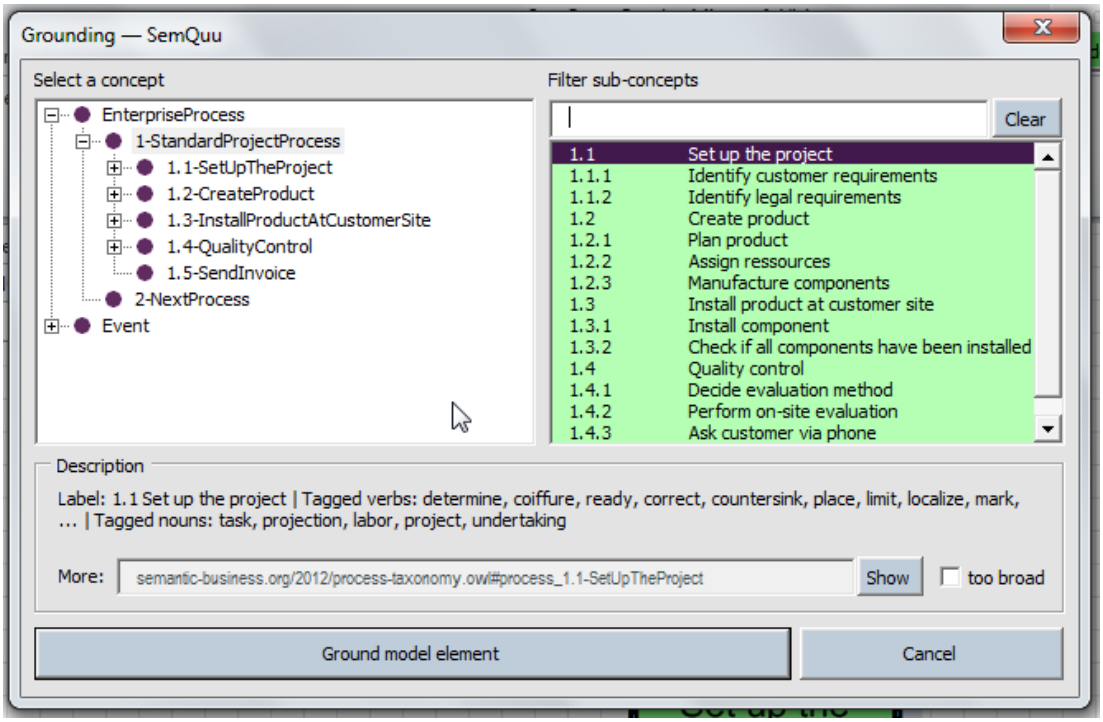
Finally, press "Ground model element". This will ground the model element which is currently shown behind the grounding dialog with the selected ontology instance.

Step 4. Optionally, to review the grounding information, do one of the following:
(a) Click on an arbitrary model element which is grounded. The grounding data will be shown as part of the shape data box which is displayed at the top of the modelling window. The following screenshot shows the grounding data for a selected model element.



You can modify the grounding information manually by altering the entries in the shape data box.

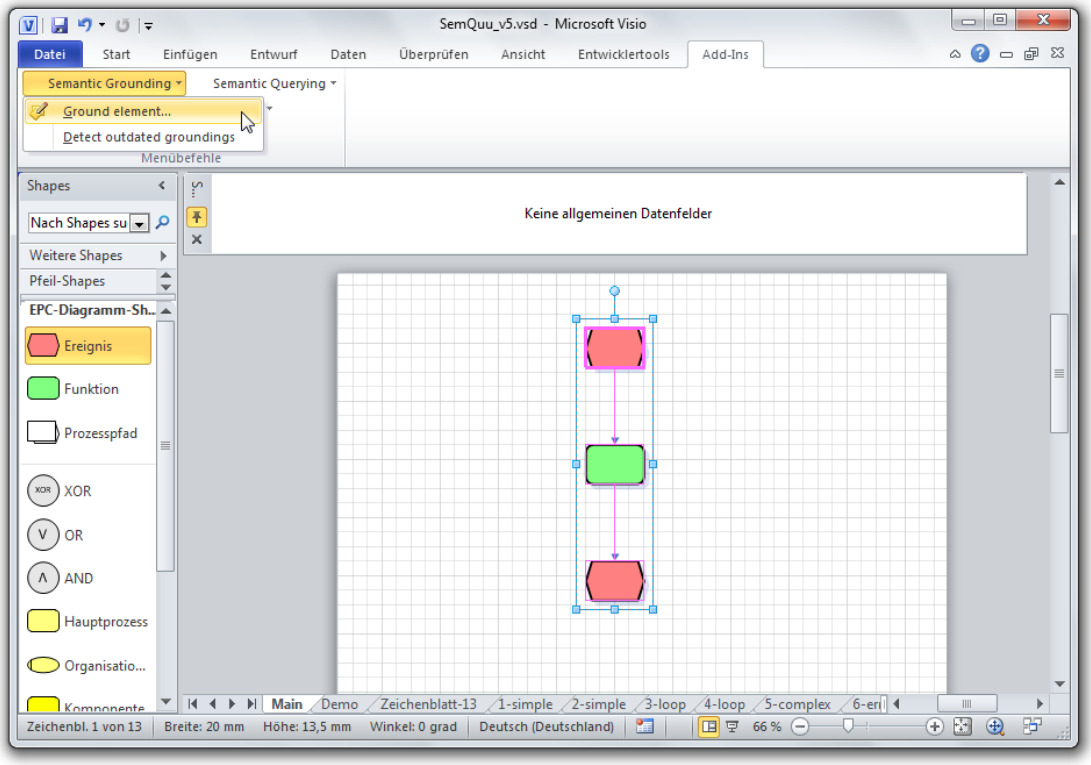
(b) Re-ground a model element. To do so, double-click on the model element. This will bring the grounding dialog back. The dialog is initialized according to the grounding data. The super-concept is selected in the explorer on the left; the instance used to ground the model element is selected in the list on the right.



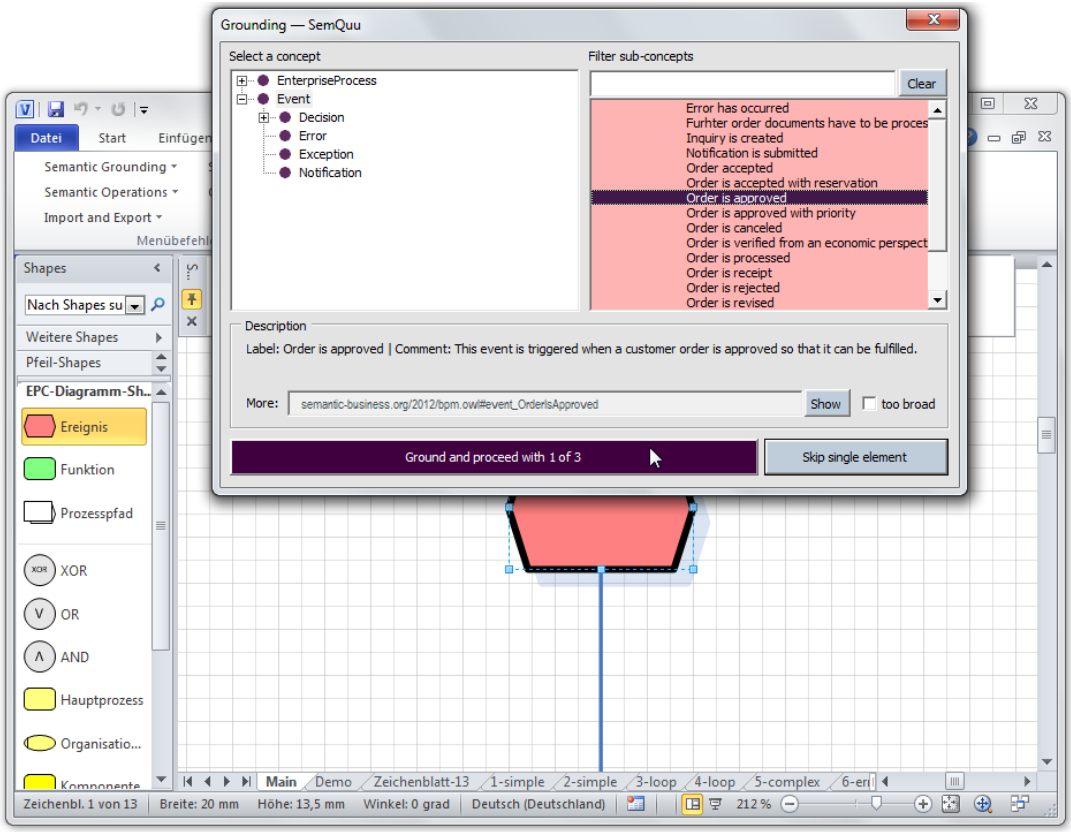
Perform your modifications and then press “Ground model element” again to save changes.

3.3 ANNO-3: Ground Multiple Model Elements

Step 1. Select all the model elements which should be grounded. Then go to the “Semantic Grounding” menu from the SemQuu Add-In and select “Ground element…”.



Step 2. When the grounding dialog appears, ground each model element as described in ANNO-2. Press “Ground model element and proceed with…” at the bottom of the dialog if you want to ground the element and proceed with the next. The next element is displayed in the middle behind the grounding dialog by automatically scrolling and zooming the model.



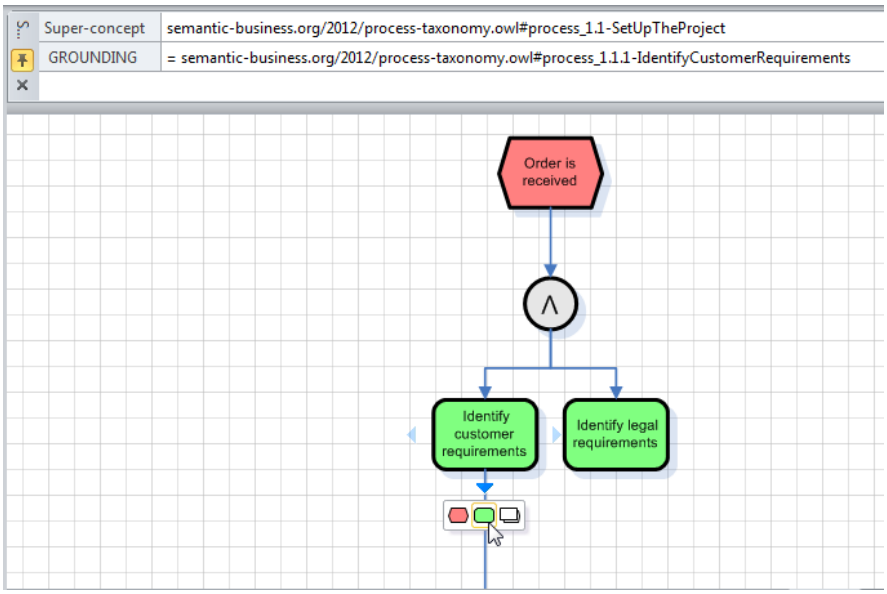
Note: The colour of the right list of ontology instances changes automatically according to the type of the model element which is to be grounded. Events are indicated by a red background, functions by a green. Also, the explorer on the left side of the grounding dialog is expanded according to the type of model element. If you select another type of element such as EnterpriseProcess as the root entry for functions, then the instance list of the right side is empty. In this way, the user is prevented from erroneous groundings.

4 Modelling with Suggestions

The SemQuu Add-In offers a feature for guided modelling whereby the patterns specified in the ontology are used. See ONTG-1 for an example of an ontology containing patterns or the SUN-ontology in the appendix. The following examples are based on this ontology.

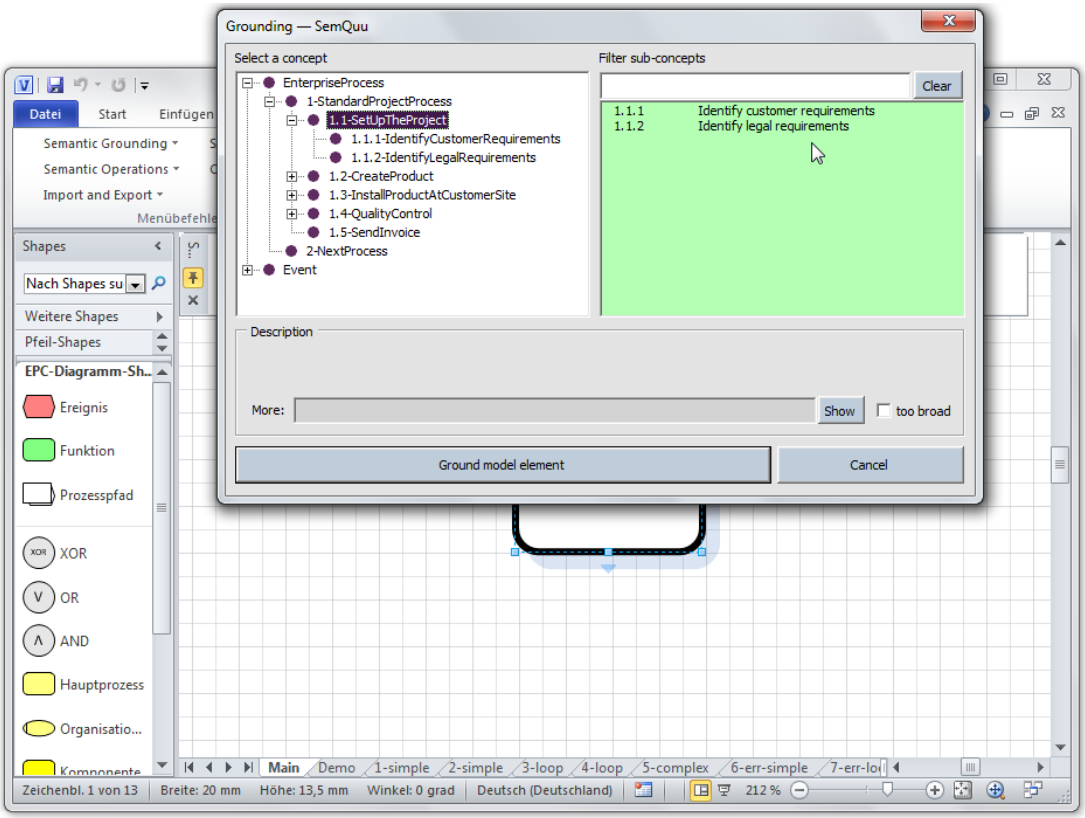
4.1 SUGG-1: Using Ontology-based Suggestions

Step 1. Select a function which is already grounded in the model. For example, “Identify customer requirements” is selected as shown below.



Hover with the mouse to the bottom-middle area of the shape. Visio suggests continuing the model. Select a function as the next element to be inserted. After the function is inserted, the grounding dialog appears.

Step 2. In the grounding dialog, the parent activity “Set up the project” of which “Identify customer requirements” is a part of is automatically selected in the explorer on the left. Thus, in the list on the right, the next activities appear according to the order specified in the ontology.

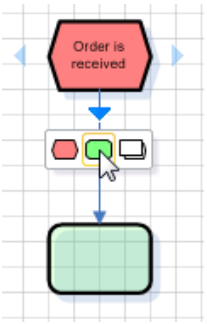


Select the next activity “Identify legal requirements”, if that fits the models content and purpose. In this way, the model can be constructed adhering to the prescribed order of activities specified in the ontology.

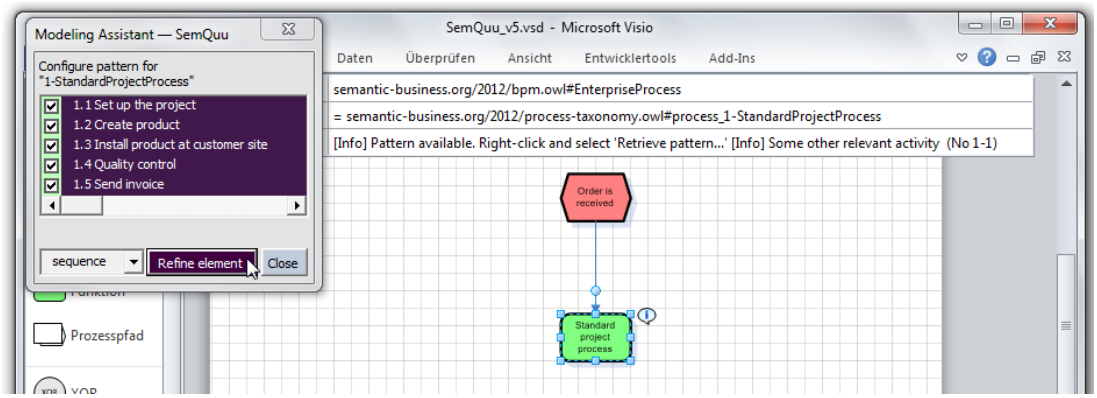
Note: As of now, this feature can only be used if the previous function in the model has no decomposition, i.e. if there are no other activities which are one level below it in the part-of-hierarchy.

4.2 SUGG-2: Top-down Modelling with Suggestions

Step 1. Insert an event in the model. In the grounding dialog which appears, select an arbitrary event such as “Order is received”. Then, hover with the mouse over the bottom-middle area of the event. Visio offers the possibility to proceed with modelling conveniently by selecting a function as the next element (see below). Insert a function which will result in the grounding dialog being shown.

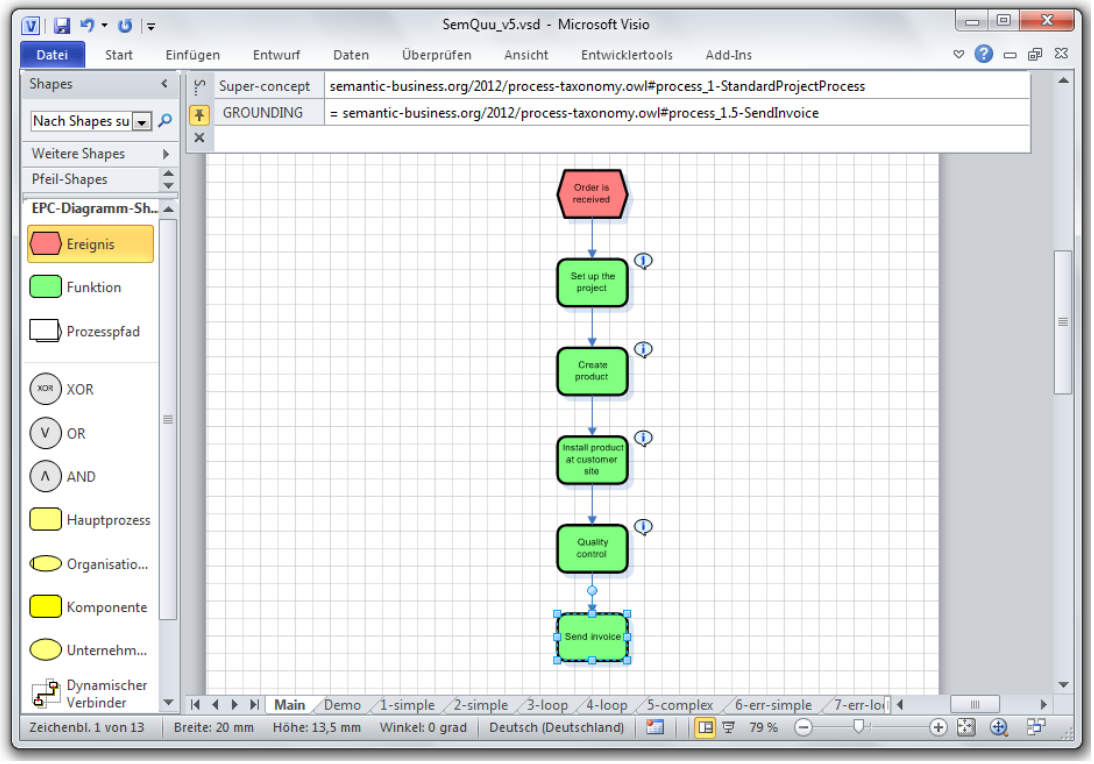


Step 2. In the grounding dialog, select the element “Standard project process” for grounding. After the dialog is closed, a small dialog “Modelling Assistant” is shown. This dialog contains a modelling suggestion, in this case a sequence of five elements.

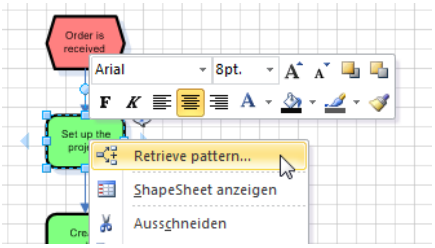


Modify the modelling suggestion e.g. by de-selecting elements or changing the control flow pattern. Finally, click on “Refine element” to insert the pattern.

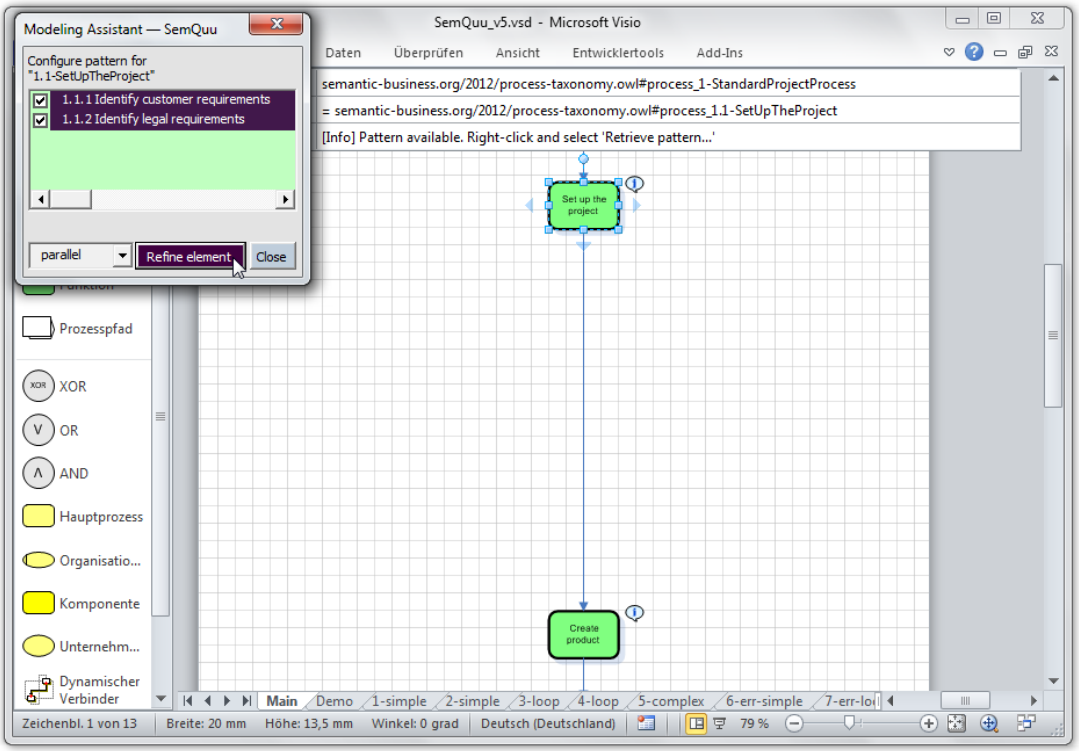
Note:
- The element for which a pattern is suggested is replaced by that pattern. Make sure that this element is still marked before you insert the pattern.
- If the grounding dialog does not appear automatically, right click on the model element and select “Retrieve pattern…”. Alternatively, double-click on the model element and re-ground the element.



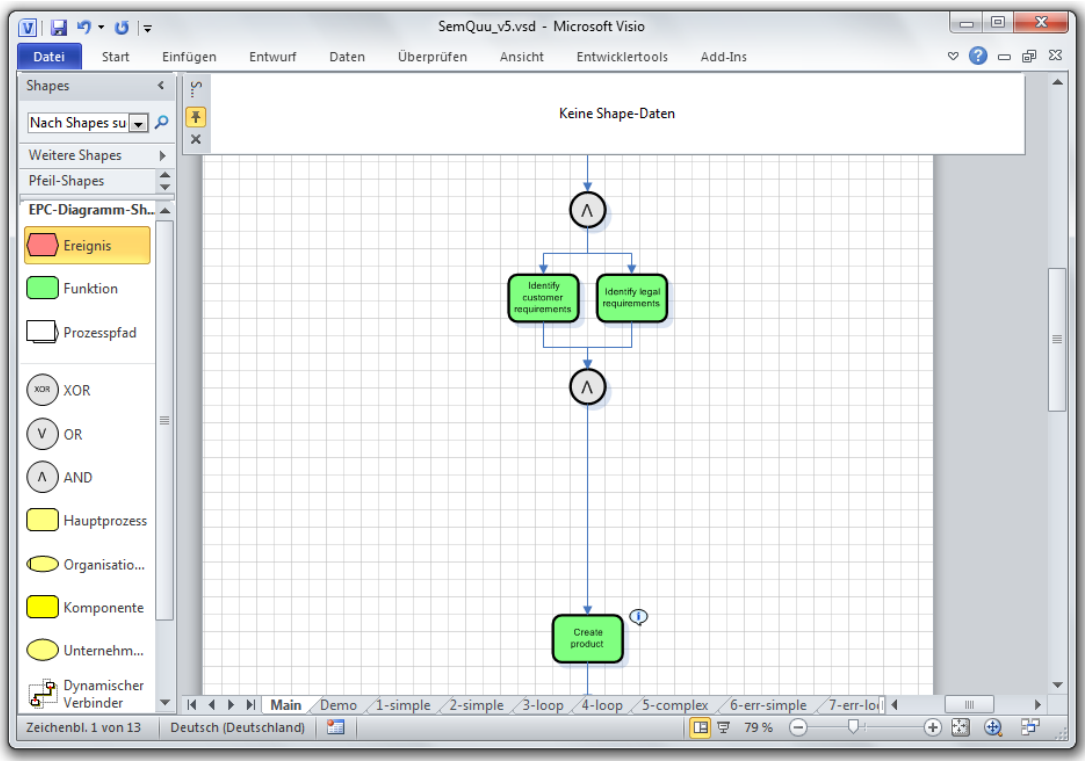
Step 3. After the pattern has been inserted, each model element that is displayed with an additional info-symbol on the top-right corner can be refined again. To do so, right click on the element and select “Retrieve pattern…”.



The pattern is retrieved from the server and the “Modelling Assistant” is shown again. Before inserting the pattern, make sure that there is enough space in the model. Take into account the amount and shape of the pattern which can easily be inferred by looking at the patterns content shown in the “Modelling Assistant”. It is important to provide enough space for the pattern in the model since the size of the pattern is NOT automatically calculated. If model elements already present in the model and the newly inserted pattern overlap, the automatic connectors get “confused” and it is tedious to “untangle” the connectors. The following screenshot shows how to ensure enough space by moving the elements downwards which are positioned below the element which will be replaced by the pattern. To move elements downwards, mark them and drag & drop while pressing the SHIFT-key. In the example shown in the screenshot, “Create product” is moved down.



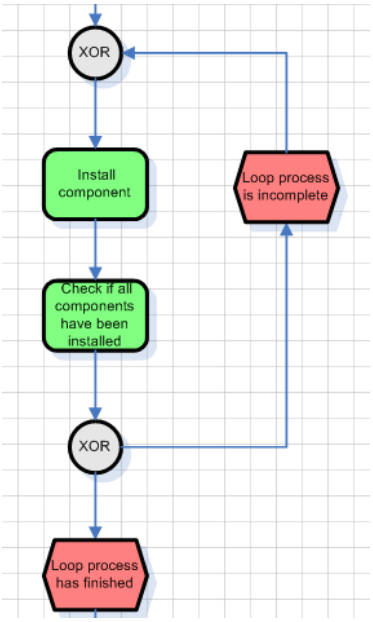
If you have finished re-arranging the model elements, activate the model element which should be replaced by the pattern again (in the example, “Set up the project”). Then press “Refine element” in the “Modelling Assistant” dialog. The pattern will be inserted as shown below.



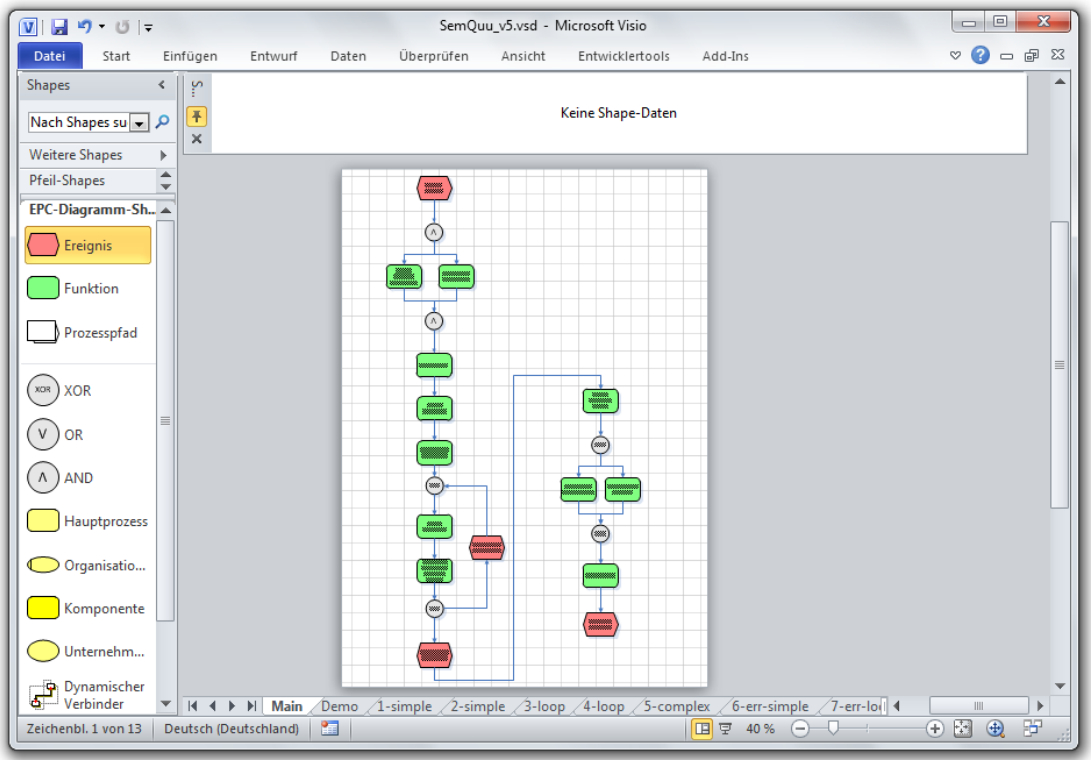
Step 4. After that, you can move the next following element upwards. As this element can also be refined by a pattern which is indicated by the info-symbol, repeat Step 3 until all infor-

mation that you wish to include in the model has been inserted.

When you construct the model, pay special attention to loops. It is important to model events after the XOR-split opening a loop. Since the suggested loop patterns do not contain these events, add them manually. You can use the events “Loop process has finished” and “Loop process is incomplete” for this purpose, as shown below.



The final model which is constructed exclusively using the elements from the SUN-ontology and the suggestion features is shown below.

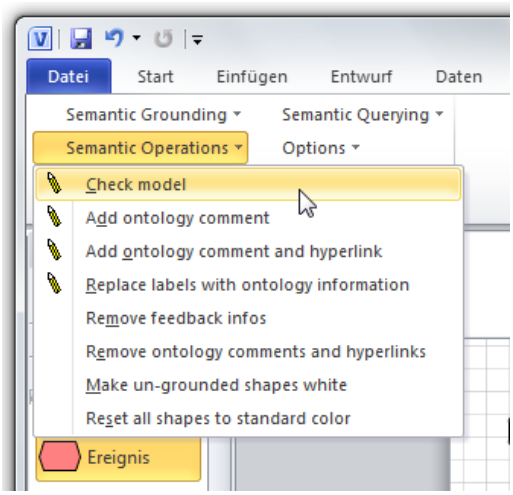


5 Semantic Operations provided by the SemQuu Add-In

This section describes additional operations that can be performed using the SemQuu Add-In on model and model elements.

5.1 SOPR-1: Perform a Semantic Correctness Check

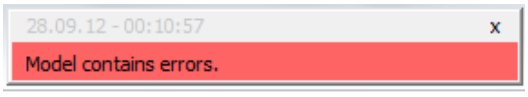
Step 1. To check the correctness of a model, activate the page of the model you want to check in Visio. Then go to the "Semantic Operations" menu of the SemQuu Add-In and select "Check model".



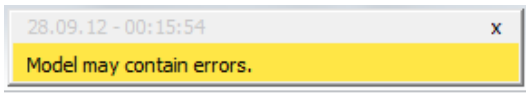
Note:

- The model is checked thereby SPARQL queries are executed against the model (see section 7.1 for an introduction on the available properties and query patterns). The queries for the active ontology are contained in text files stored in the "$root\webapps\semquu\queries" directory on the server ($root is the directory of the Tomcat server).
- The queries follow the same naming conventions as the ontologies. The correctness queries for ontology one (Demo A) are contained in a file with a name starting with "1_", for ontology two (Demo B) in a file starting with "2_" and so on.

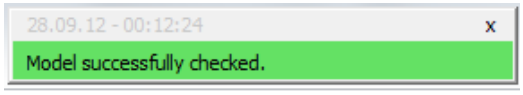
The aggregated results of the correctness check are displayed in a small modelling status window that appears at the top of the screen. It displays the result of the check as well as the time the check has been performed, as shown below. The window changes the colour to indicate the result of correctness checking.



Red = The model could not be converted in the ontology-based representation OR there are errors in the model. If ossible, a short textual hint provides more detail on the cause of the error (e.g. „Cannot find start event", „Model is unstructured", „Loop error"). If there is at least one semantic error detected, „Model contains errors" is diaplyed and the errors are indicated by callout data graphics in the model (see Step 2 for a description of callout data graphics).

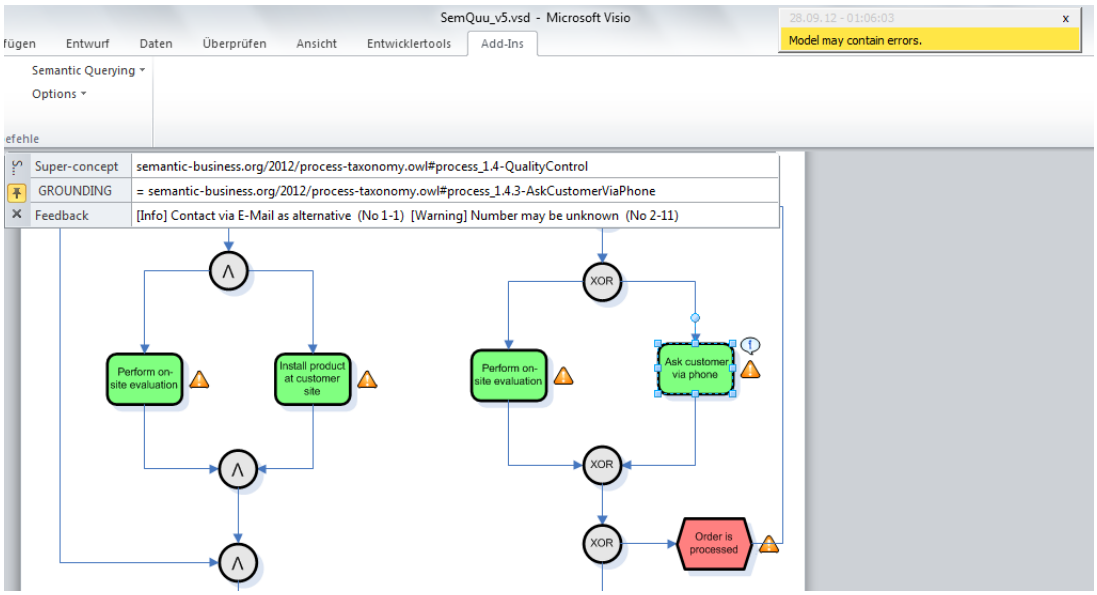


Yellow = Some of the queries have results indicating that the model contains critical constructs that may cause the model to be inappropriate or inaccurate.



Green = The queries did not match any of the model elements, so the model seems to be correct.

Step 2. You may inspect the results of the correctness check in more detail by looking at the model. Errors are signified by , warnings by and information by . By activating a model element, you can read messages corresponding to the issues in the shape data box in the field “Feedback”.



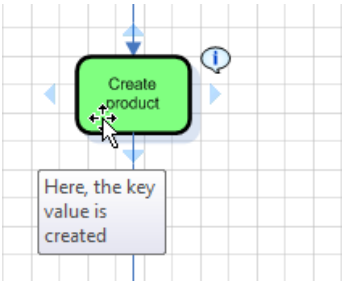
Note: The messages refer to problems or errors arising from the use of the model element at the *specific position* in a model. So it may be the case that model elements grounded with the same ontology instance but that are on different positions in the model produce different messages due to the semantic correctness checking.

Step 3. Optionally, you can activate “Instant verification” in the “Options” dialog (see ANNO-1) of the SemQuu Add-In. If activated, the model is sent to the server each time you add a new model element. The model is re-checked on the server and the result is displayed exactly the same way then activated manually.

5.2 SOPR-2: Enrich the Model with Ontology Information

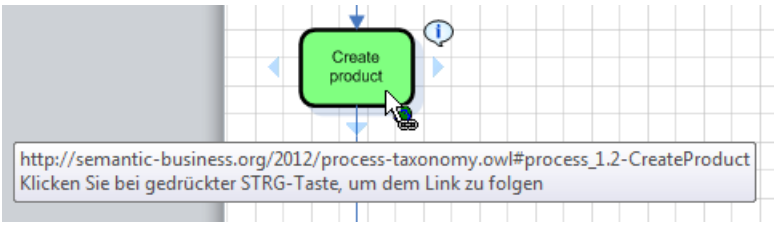
All of the following commands can be selected in the "Semantic Operations" menu of the SemQuu Add-In.

Step 1. Select "Add ontology comment". This will enrich the model with comments which are copied from the `rdfs:comment`-elements of the ontology. The comments appear when hovering with the mouse over the model element.



You can remove the comments by selecting "Remove ontology comments and hyperlinks" from the "Semantic Operations" menu.

Step 2. Select "Add ontology comment and hyperlink". This will enrich the model by comments as in the step before. Additionally, a hyperlink to the web page describing the semantics of the model element is created. To show the hyperlinks, deselect all model elements and hover with the mouse over a model element. To show the comments, select at least one arbitrary model element. Then hover with the mouse over other model elements to show their comments.

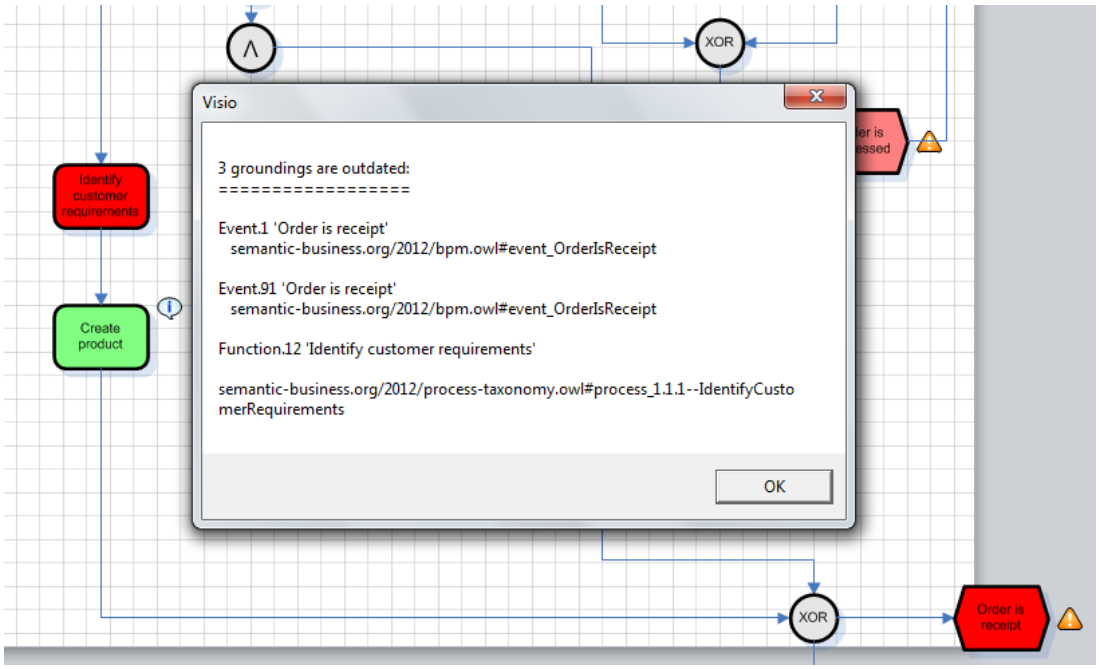


You can remove the comments and hyperlinks by selecting "Remove ontology comments and hyperlinks" from the "Semantic Operations" menu.

Step 3. Select "Replace labels with ontology information" to overwrite all labels of the model elements with the labels of the ontology.

Step 4. Select "Make un-grounded shapes white" to detect shapes that have not been grounded. To revert to the original default colours, select "Reset all shapes to standard color".

Step 5. To detect model elements grounded with ontology instances that are no longer part of the ontology select "Detect outdated groundings". Shapes with an outdated grounding are filled with red colour as shown below.

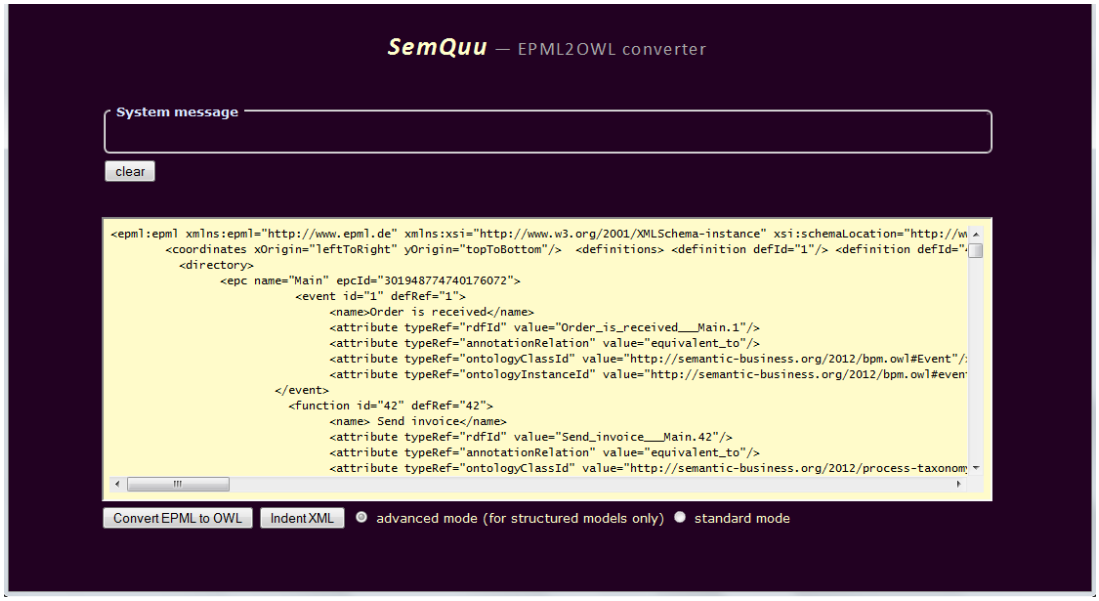


You can reset the colour of the shape as described in Step 4.

6 SemQuu Server Repository

6.1 EXPT-1: Import a Model into the Repository

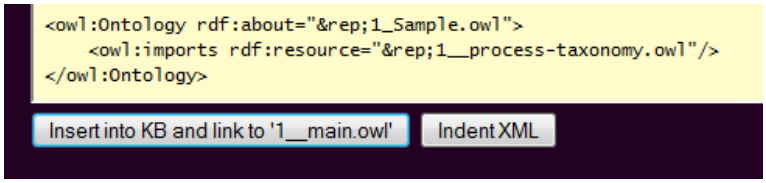
Step 1. Select "Export sEPML to SemQuu" from the "Import and Export" menu of the SemQuu Add-In. The SemQuu Server will be loaded in a web browser displaying the exported data in sEPML intermediate format.



Note: The sEPML is an intermediate format, „s" means EPML with semantic attributes.

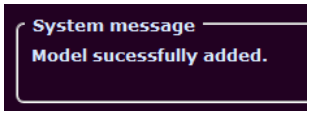
Step 2. Press the "Convert EPML to OWL"-button. After the conversion is executed, the result is displayed in the textarea and the button below changes to "Insert into KB and link to

‘1__main.owl’”.



Note: Depending on the selected ontology in the “Options” dialog in SemQuu Add-In, the ontology (Knowledge Base) is selected. If you use the ontology “Demo A” with the associated number “1”, the corresponding process taxonomy file “1__process-taxonomy.owl” is imported in the generated ontology. After generation, the new ontology will be imported in the corresponding main ontology “1__main.owl”.

Step 3. Press the “Insert into KB and link to ‘1__main.owl’”-button. The generated ontology is imported in the main ontology and a message appears.

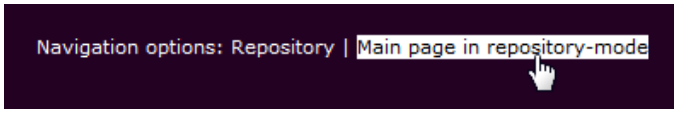


If the ontology is already present in the repository, instead of the message shown above a warning message appears and the button of the form changes to “Update KB (…)”.

Press the “Update KB (…)”-button if you want to overwrite the ontology on the server.

Step 4. After the newly generated or updated ontology is stored in the repository and linked to the main ontology, a link “Main page in repository mode” is displayed. Using this link, you can go to the main page of the SemQuu Server component for querying the complete ontology importing the newly generated ontology.

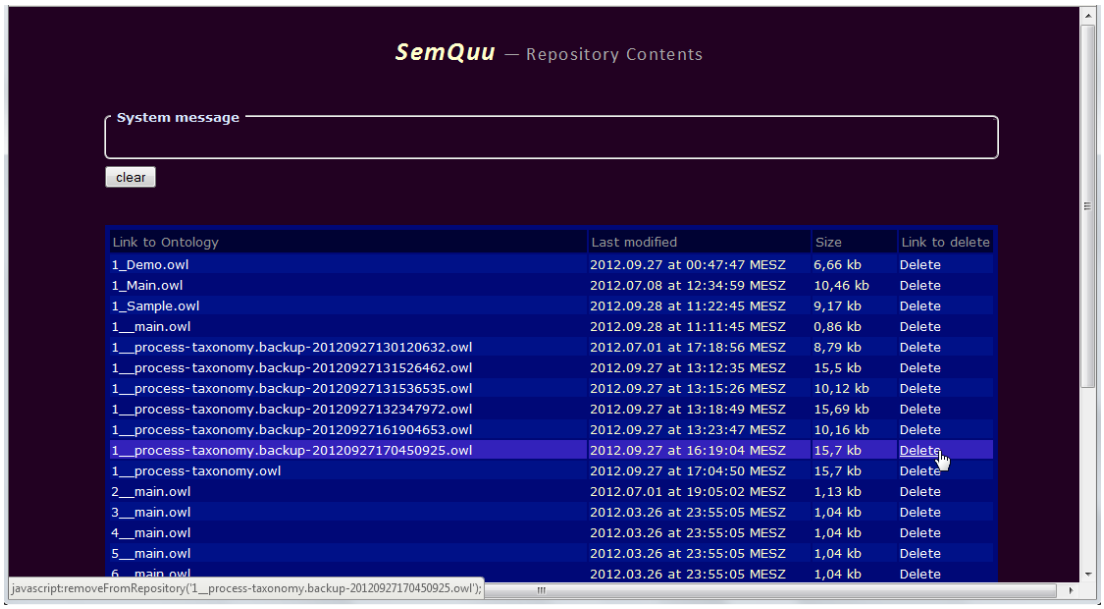
Note: Due to a performance issue, it is not recommended to use this link for large repositories at the time of this writing. If you follow the link, the page is displayed but the server is working in the background for several minutes before you can use the page. For querying, see SEMV-1 and -2.



You can also navigate to the repository page by following the link “Repository”.

6.2 EXPT-2: Manage Repository Contents

Step 1. After ontology generation, follow the link “Repository” (see EXPT-1). You can also directly access the repository page by pointing your browser to http://localhost:8080/semquu/list/. The repository page will be displayed as shown below.



Step 2. Perform the operations you need.

(a) Display ontologies by clicking on the links to display ontologies in the leftmost column of the repository contents table. The ontology will be displayed in raw, textual representation in the browser.

(b) Delete ontologies by clicking on the links to delete an ontology. If you delete an ontology, not only the file will be removed from the server, but also the import statement in the corresponding main ontology file.

7 Introduction to Semantic Process Model Verification

7.1 Query Structure and Ontology Properties

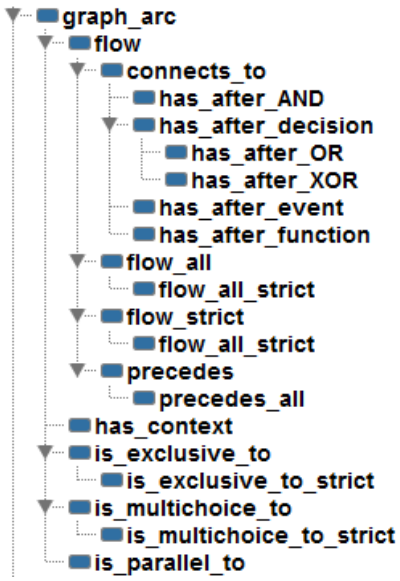
To query the process knowledge that has been stored as an OWL-DL ontology inside the SemQuu Server you can use the W3C standard query language SPARQL. The basic SPARQL syntax is relatively simple. After a SELECT keyword, the variables are listed which contain the return values (comparable to columns of a table). In the following WHERE clause, a graph pattern can be specified for describing the desired result by using triple patterns. Below, a query illustrating some of the triple patterns and abbreviations used in this manual is shown.

```
SELECT ?var_1 ... ?var_n
WHERE{
  s p o .
  s1 p1 o1 ; p2 o2 .
  s3 p3 [ p4 o4 ] .
  s5 a o5 .
}
```

Each triple pattern consists of three parts (subject, predicate, object), each separated by a blank. The subject and object of a triple pattern usually specify known individuals, classes or blank nodes, the predicate specifies a property between them that must be present in the ontology for the query to match. Each triple pattern is ended with a dot, variables are prefixed with a question mark. If a triple

pattern shares the first part (subject) of the previous pattern, it can be appended with a semicolon. If triple patterns have a subject which is the object of a preceding pattern, then the following patterns can be abbreviated by writing just the predicate and the object inside square brackets which signify a blank node. If the predicate of a triple pattern is the rdf:type relation, it can be abbreviated to "a". The upcoming 1.1 version of SPARQL will additionally support NOT EXISTS for detecting missing triples and specifying a cardinality for predicates, e.g. *node1 connectsTo{1,5} node2* will match if 1 to 5 *connectsTo*-predicates are in between the subject and object of the triple pattern. For more information, see the SPARQL specification for an introduction to the query language (www.w3.org/TR/rdf-sparql-query).

In order to provide an expressive yet simple way of querying process knowledge, various predicates ("object properties" in the OWL jargon) are present in the generated OWL-based representation of process models (for an example of a model and the generated representation see Appendix A4). These properties can be used at the predicate position in the query patterns of a SPARQL query. The hierarchy of these properties is as shown below.



The hierarchy means that for example, if two activities are connected by *flow_all_strict*, then they are also connected by *flow_strict*, *flow_all*, *flow* and *graph_arc*. The properties are further explained in the table below. The column *Property Name* contains the names of the properties as shown in the hierarchy above (these names are also used in the OWL-DL encoding of process models, see Appendix 4 for an example). *Short Description* describes the semantics of the property using an alternative name, *Description* provides an abstract notion of the semantics and *Example* gives a concrete example of two nodes (or elements) of a process model that are connected via the property. The column *BP* finally indicates whether the property is a behavioral property or not. Behavioral properties are related to the *execution semantics* of a process model. That is, in order to decide whether a property holds between two nodes (elements) of a process model, all possible execution traces of the process model have to be considered. In order to retrieve elements of processes that are related to each other via one of the properties introduced in the table above, a SPARQL graph pattern has to be constructed. Following the description of the properties in the table, some simple query patterns are provided as illustrative examples of how simple queries may be composed.

Property Name	*Short Description*	*Description*	*Example*	*BP*
flow	Path relation	"A flow B" means that there is a path between node A and B.	"Receive order" is somewhere followed by "Notify customer".	-
connects_to	Direct connection	"A connects_to B" means that node A is directly connected to node B. Depending on the type of the successor node, some subproperties such as e.g. has_after_AND are available.	The event "Receive order" is directly followed by the function "check order".	-
flow_all	Guaranteed execution	"A flow_all B" means that all tokens that pass node A will also pass node B.	Each time "Send goods" is performed, "Send invoice" is performed too.	-
flow_all_strict	Guaranteed subsequent execution	"A flow_all_strict B" means that "A flow_all B" holds and "A flow_strict B" holds.	Each time "Prepare meal" is executed, "Serve meal" is executed later.	✓
flow_strict	Subsequent execution	"A flow_strict B" means that there is a path between node A and node B but not vice versa.	"Perform pre-check" has to be executed always before "Perform in-depth check".	✓
precedes	Some preceding execution	"A precedes B" means that each token that passes node B has passed Node A at least once before.	When services are executed on behalf of the customer, a contract must have established before.	✓
precedes_all	Always preceding execution	"A precedes_all B" means that "A precedes B" holds and that node A has been passed each time before.	When a public event takes place, the local administration has to be informed some time before.	✓
has_context	Member of a logical context	"A has_context B" means that node A is a member of the logical context B (a logical context is the set of nodes on a single branch between a split and a join node in a structured process model).	"Reject order" is part of the context representing activities devoted to order rejection.	-
is_exclusive_to	Observed exclusive execution (XOR)	"A is_exclusive_to B" means that either node A or node B is executed depending on a previous decision.	"Express delivery" is executed exclusive to "Standard delivery" for each line item.	-
is_multichoice_to	Observed alternative execution (OR)	"A is_multichoice _to B" means that either node A or node B or both nodes are executed depending on a previous decision.	One or both of the activities "Show premium products" and "Show design products" are executed.	-
is_exclusive_to _strict	Global exclusive execution (XOR)	"A is_exclusive_to_strict B" means that "A is_exclusive_to B" holds and that there is no path between the two nodes.	To register, either "Create institutional account" or "Create private user account" has to be performed.	✓
is_multichoice_to _strict	Guaranteed exclusive execution (OR)	"A is_multichoice_to_strict B" means that "A is_multichoice_to B" holds and that there is no path between the two nodes.	When a new product is launched, one or both of the activities "Announce in electronic media" and "Announce in print media" has to be executed once.	✓
is_parallel_to	Parallel execution	"A is_parallel_to B" implies that node A and node B are executed concurrently.	A demonstration is observed by the police.	-

7.2 Query Patterns

7.2.1 Node Selection Patterns

In order to select nodes within queries, different patterns can be used depending on what is already known about the nodes. The namespace ":" indicates the represented process model, "pt:" the process taxonomy (see also ONTG-1) and "bpm:" the namespace of the process ontology (see also Appendix 4 for an example where all namespaces are used).

- **Known node.** In the simplest form, a graph pattern may be constructed using one already known node from the process model and one variable, as shown in the following query.

```
SELECT ?node WHERE { :knownNode bpm:property ?node }
```

  The query returns all nodes as values of the variable `?node` that are connected to the instance `:knownNode` via `bpm:property`.

- **Known annotation instance.** If instead of a specific node the annotation of a node is known, the previously presented query can be modified as follows.

```
SELECT ?node1 ?node2 WHERE {
    ?node1 bpm:annotation pt:knownInstance ; bpm:property ?node2
}
```

  The query returns all pairs of nodes for which the following conditions hold: The first node is annotated via `bpm:annotation` with `pt:knownInstance` and the second node is connected via `bpm:property` to the first node.

- **Known abstract annotation instance.** If instead of a specific annotation of a node a more abstract annotation is known, the previously presented query can be modified as follows.

```
SELECT ?node1 ?node2 WHERE {
    ?node1 bpm:annotation/bpm:is_subprocess_of* pt:knownInstance
}
```

  The query returns all pairs of nodes for which the following conditions hold: The first node is annotated via `bpm:annotation` with an unknown instance that connected to a known `pt:knownInstance` via an arbitrary steps of the `bpm:is_subprocesss_of` property.

- **Known annotation class.** If instead of a specific annotation the class of the annotated instance is known, the previously presented query can be modified as follows.

```
SELECT ?node1 ?node2 WHERE {
    ?node1 bpm:annotation [ a bpm:KnownClass ] ; bpm:property ?node2
}
```

  The query returns all pairs of nodes for which the following conditions hold: The first node is annotated via `bpm:annotation` with an instance being a member of `bpm:KnownClass` (class membership might be inferred) and the second node is connected via `bpm:property` to the first node.

7.2.2 Logical Context Pattern

To retrieve nodes that are e.g. executed in parallel to a known node, use the following pattern.

```
SELECT ?node WHERE {
    bpm:knownNode bpm:has_context ?c1 . ?node bpm:has_context ?c2 .
    FILTER(?c1 != ?c2) . ?c1 bpm:is_parallel_to ?c2
}
```

The query returns all nodes that are executed in parallel to `bpm:knownNode`. This pattern can also be used for other types of logical contexts such as `bpm:is_exclusive_to`, `bpm:is_exclusive_to_strict`, `bpm:is_multichoice_to` and `bpm:is_multichoice_to_strict`. This pattern can be combined with patterns for node selection such as "Known instance", "Known annotation" and "known annotation class".

7.2.3 Advanced Patterns

The following patterns show how to use logical expressions such as AND (intersection), OR (union) and NOT (complement).

- **Known annotation alternative.** To require the grounding instance to be one of a predefined set of alternatives, a filter can be used as follows.

  ```
  SELECT ?node WHERE {
      ?node bpm:annotation ?x .
      FILTER(?x = pt:instance1 || ?x = pt:instance2 || ?x = pt:instanceN)
  }
  ```

  The query returns all pairs of nodes being annotated via `bpm:annotation` with `pt:instance1` or `pt:instance2` or `pt:instanceN`.

- **Known annotation class intersection.** To require the grounding instance to belong to several classes, a conjunction of classes written as a list separated by a commas can be used as follows.

  ```
  SELECT ?node WHERE {
      ?node bpm:annotation [ a bpm:ClassA , bpm:ClassB , bpm:ClassC ]
  }
  ```

  The query returns all nodes annotated via `bpm:annotation` with an individual that is a member of each of the classes `ClassA`, `ClassB` and `ClassC`.
  Note: If you define in your ontology what it takes to belong to class A (e.g. a time span), to class B (e.g. an amount of money) or to class C (e.g. a specific stakeholder is involved) you can combine these aspects into the query without introducing detailed domain knowledge. Rather, the inference engine will decide/infer if the grounding instance fulfils all the conditions which do not have to be described explicitly.

- **Known annotation class union.** To require the grounding instance to belong to some known classes, a union of the classes can be used as follows.

  ```
  SELECT ?node WHERE {
      ?node bpm:annotation ?x . {?x a bpm:ClassA} UNION {?x a bpm:ClassB}
  }
  ```

The query returns all nodes annotated via `bpm:annotation` with an individual that is a member of `bpm:ClassA` or `bpm:ClassB` or both.

- **Known annotation class negation.** To restrict the grounding instance to be not a known instance use negation as failure which is expressed by the keyword NOT EXISTS as follows.

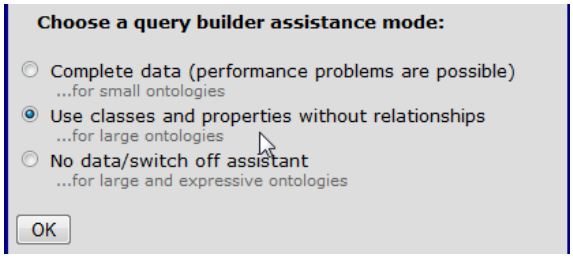
```
SELECT ?node WHERE { NOT EXISTS {?node bpm:annotation pt:knownInstance} }
```

The query returns all nodes not being annotated with `pt:knownInstance` via `bpm:annotation`. If you combine this pattern with other patterns you can e.g. require a grounding instance to be of a specified type but not to be a specific known instance ("subtraction"), or you can specify that all groundings are allowed except that of instances of a known class ("complement").

8 Executing Semantic Verification Queries

8.1 SEMV-1: Run a Single Query against a Repository Ontology

Step 1. Open the main page of the SemQuu Server component by pointing your browser to http://localhost:8080/semquu. After the page has been load, a dialog appears to select the level of query support. For the simple demo-ontology "inferencetest.owl" you can choose the first option "Complete data".



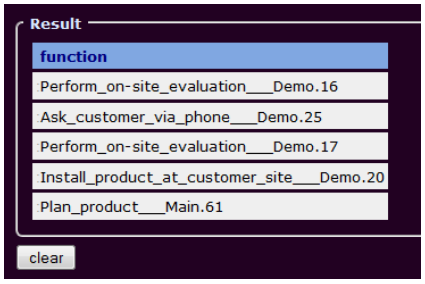
If this option is chosen, all possible information that can be inferred from the knowledge represented in the ontology is used to assist query construction. Hence for more comprehensive ontologies such as the "main"-ontologies (e.g. "1__main.owl") importing several ontologies, use the second mode ("Use classes and properties without relation-ships"). This mode will only assist you with the completion of class- and property-names.

Step 2. Optionally, change the URI in the "Doc"-section if you do not want to use the pre-selected "inferencetest.owl"-ontology. Set the URI to the repository ontology you want to use (e.g. http://localhost:8080/semquu/repository/1__main.owl which corresponds to the ontology "Demo A"). Then press the "Load and validate"-button beside the text input field. The dialog as described in Step 1 shows up again. After that, the ontology is ready to be queried with SPARQL and machine inference.

Step 3. Construct your SPARQL query using the textarea in the middle of the page. Construction of correct queries is simplified by a suggestion functionality which suggests you suitable names of constructs, depending on the respective position in the query. As an alternative to the text-based construction, you may use the form above the central textarea field in order to have more guidance in the construction of your first queries.

Note:
- The link “show prefixes” provides information in regard to the available namespaces.
- The link “help” provides information on the extensions of SPARQL available in SemQuu. The most notable extensions are (1) the extensions provided from the Jena ARQ library, (2) the Lucene full text search and (3) GLEEN. To use the Lucene full text search, you have to enable indexing when the ontology is constructed. To do so, expand the options section by clicking on “show options” below the “Execute”-button. Check the option “create index”. Then load and validate the ontology again. Lucene provides some advanced features like fuzzy search that you can use to search the models content. The extension GLEEN provides the capability to specify constraints on paths beyond SPARQL 1.1 and to return the set of nodes contained in a Subgraph.
- You can also insert predefined queries using the link “insert queries”.

Step 4. Make sure, all return variables that you want to include in the result are checked in the “Return Vars”-section on top of the textarea field. Then press “Execute” below the textarea field to get the query results as shown below.



Note: While you specify the graph pattern in an SELECT query which follows the WHERE-keyword, variables with a name longer than 2 letters are appended automatically to the return variable list. This list is specified after the SELECT-keyord of the query. Additionally, a check box appears for each return variable above the textarea where each return variable contained in the query is checked. If you uncheck a variable in this list, then it is automatically removed from the list after the SELECT-keyword. With this feature, you can conveniently exclude unimportant variables which you only need to specify your patterns by naming them with a single letter such as e.g. ?x, ?y or ?z.

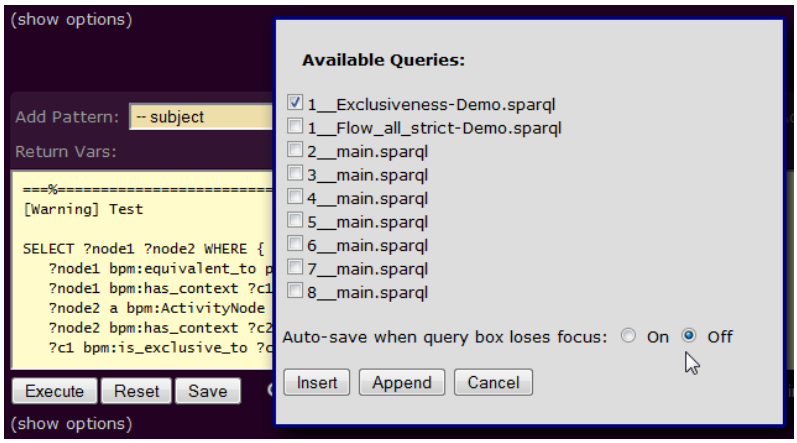
8.2 SEMV-2: Specify and Run Batch Queries

Batch queries are multiple queries within one single file on the server.

Step 1. Perform the Steps 1-2 of SEMV-1 to load the ontology for which you want to specify a batch query for instant verification.

Step 2. Construct the batch query for performing instant verification. A batch query consists of two or more separate SELECT queries. Prefix each query with two lines. The first line should start with "===%===". The second line should start with the type of the issue being one of "[Info]", "[Error]" or "[Warning]" followed by a natural language description of the issue within one single line.

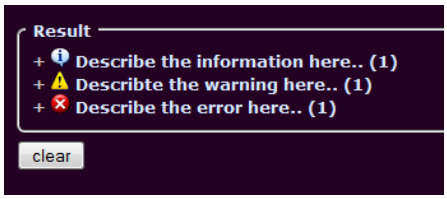
You can use templates for batch query construction. If you click on "Insert queries" below the textarea field, a list of available queries is displayed as shown below. You can select one or more queries to be inserted or appended into the textarea field.



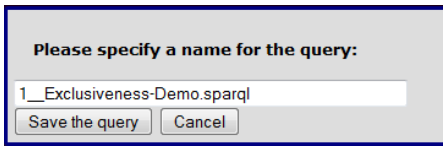
Queries used for instant verification are contained in files named in the scheme `<no>__description-of-the-query-without-blanks.sparql`. For the placeholder `<no>`, use the number of the selected ontology in the "Options" dialog of the SemQuu Add-In. For example, if you have set the first ontology "Demo A" in the SemQuu Add-In, then your models are checked using queries contained in files starting with `1__`.

Note:
- If you select a single query, the "Auto-save"-feature is turned on automatically so that the query is saved each time you click outside the textarea field. If you do not want this, then select "off" for this option. You can see the file name which is used to save the query when you expand the options-section by clicking on "(show options)" below the textarea field.
- If you select multiple queries, the "Auto-save"-feature is not available. The reason for this is that the file name is not specified if multiple queries of multiple files have been selected.
- if you want to compose a query using multiple, previously stored queries, insert or append all the queries. Then save the query to a single file using the "Save"-button.

Step 3. View the result of your batch query by pressing the "Execute"-button. Unlike conventional queries, the results of batch queries are displayed in an aggregated form. Click on the "+"-symbol to expand the results for each query type.



Step 4. If the “Auto-save”-feature is not enabled, press the “Save”-button to store the batch query on the server. If a query with this name already exists, it is overwritten without notification.



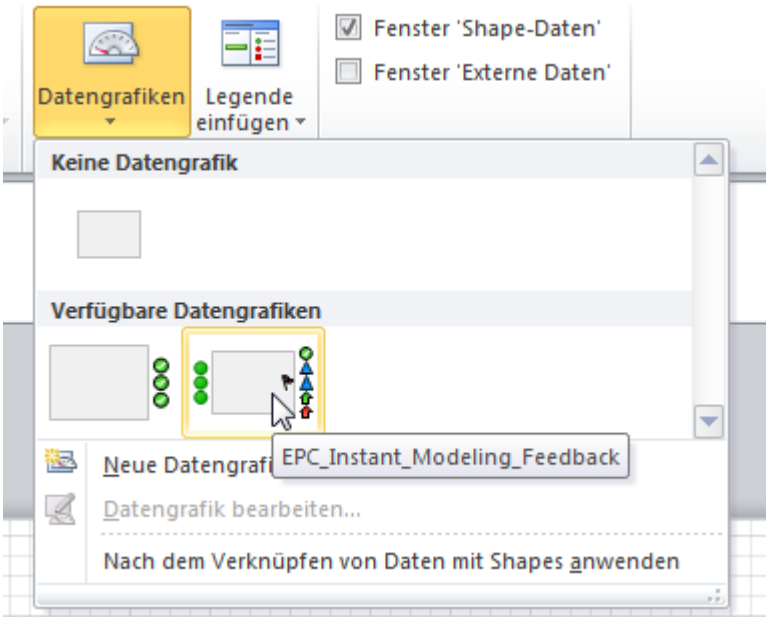
Note: To use the batch query for automatic verification, see SOPR-1 and ANNO-1.

8.3 SEMV-3: Visualize Node Relations

In order to visualize the relations between nodes, the SemQuu Client provides an extra set of symbols (in the MS Visio jargon: “data graphics”) that can be used to show the type of relation that holds between two nodes. For the sake of clarity, it is recommended to only use one node with a known annotation (cf. section 7.2.1 for patterns related to node selection). All relations of this node to other nodes can then be shown using the procedure described below.

Step 1. Start the SemQuu Client (Visio) and activate the demo model “maintest”. Go to the Options menu of SemQuu and select knowledge base “Demo B”. In the area “Semantic correctnesss”, ensure that the queries “main.context.sparql” and “main.flow.sparql” are checked.

Step 2. Make sure that the correct set of data graphics is available and is named *EPC_Instant_Modeling_Feedback*. You can rename a set of data graphics by right-clicking on the name of the data graphics entry in the list “available data graphics” (see below).



Step 3. Optional. Go to the SemQuu Server Component and insert and modify each of the queries contained in “2__main.context.sparql“ and “2__main.flow.sparql”.

Step 4. Select “Check model” from the “Semantic operations”-menu of the SemQuu Add-in. The model is sent to the server, converted and the results are shown using the following symbols.

 **flow**

 **flow_all**

 **precedes_all**

 **flow_all_strict**

 **precedes**

 **flow_strict**

 **parallel_to**

 **exclusive_to**

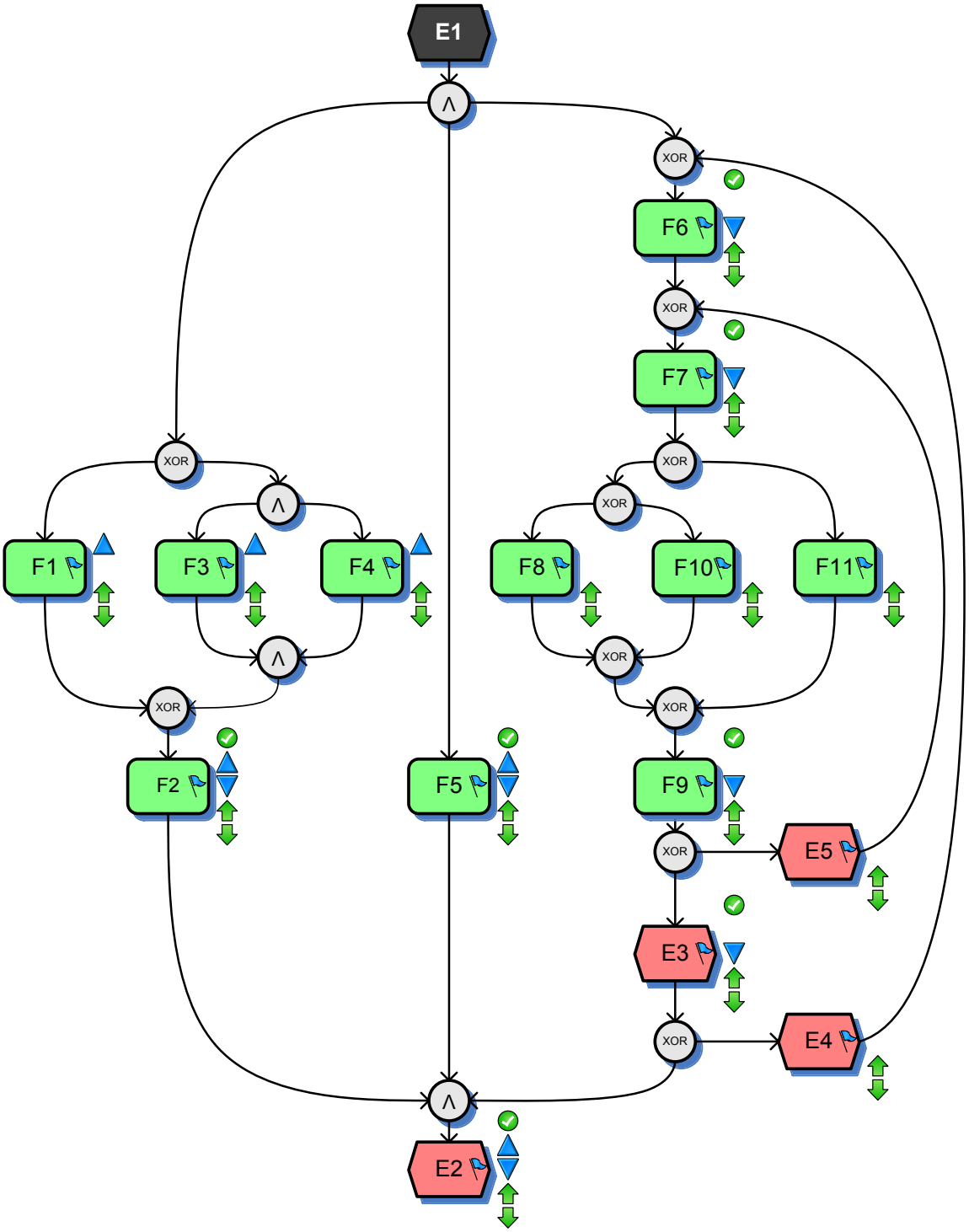
 **exclusive_to_strict**

Note: Symbols for *multichoice_to* and *multichoice_to_strict* are omitted since relations associated to these logical contexts occur similarly to the exclusive contexts.

In the following, some examples are shown. The examples make use of the queries described in Step 3. The queries essentially contain two node variables ?node1 and ?node2. The queries which are executed on the SemQuu Server test if a relation between the two nodes exists and if so, the binding of variable ?node2 is returned to the SemQuu Client. In the examples, ?node1 has been visualized by a grey shaded model element, whereas ?node2 is visualized with the above symbols.

Put simply, the grey shaded model element is the fixed node. The nodes that exhibit a symbol are arbitrary nodes that have a relation to the fixed node. The type of the relation is indicated by one of the symbols introduced above.

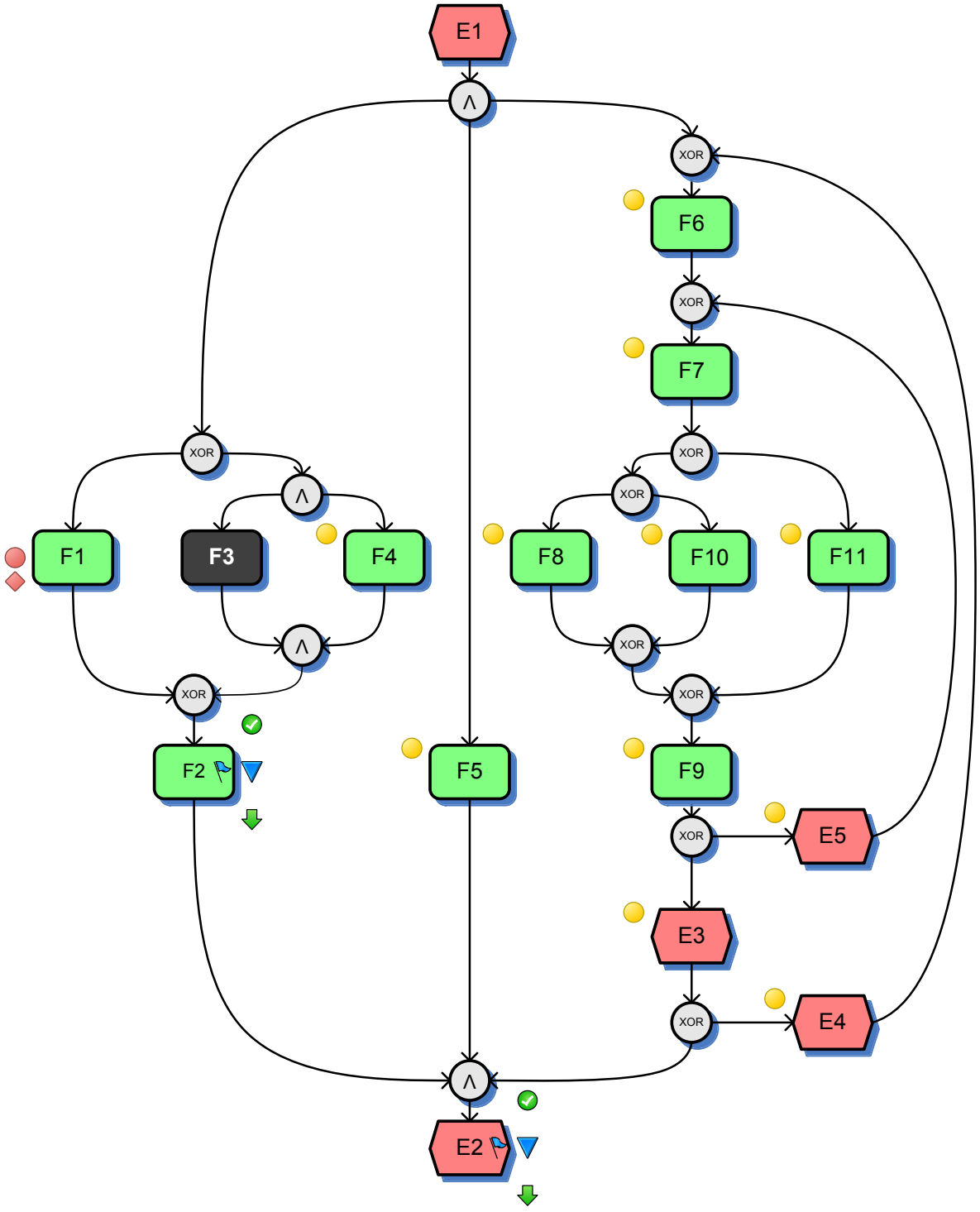
Example 1: Show nodes that can be reached from E1.



Legend:

= flow, = flow_all, = precedes_all, = flow_all_strict, = precedes, = flow_strict

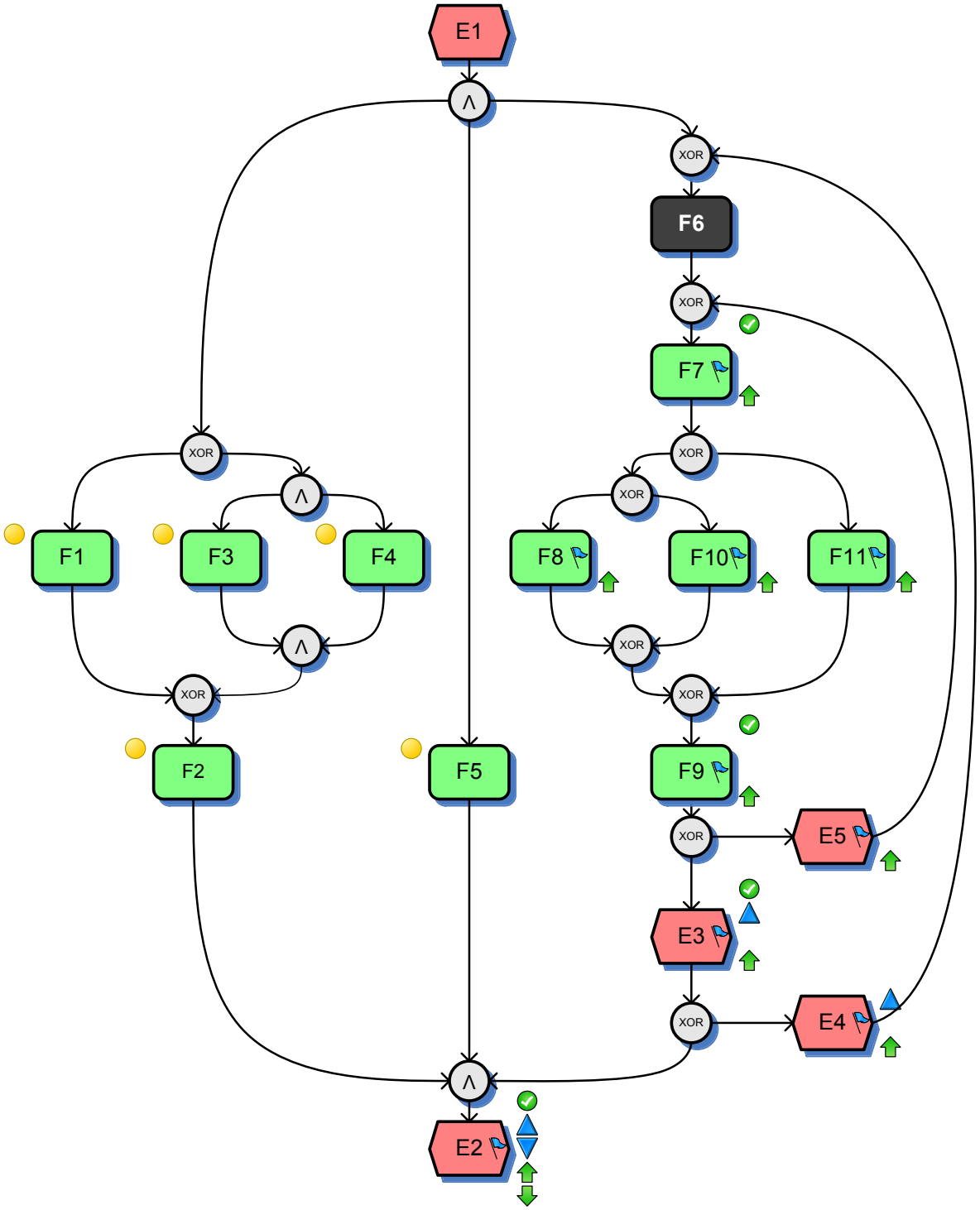
Example 2: Show nodes that can be reached from F3.



Legend:

= flow, = flow_all, = precedes_all, = flow_all_strict, = precedes, = flow_strict,
= parallel_to, = exclusive_to, = exclusive_to_strict

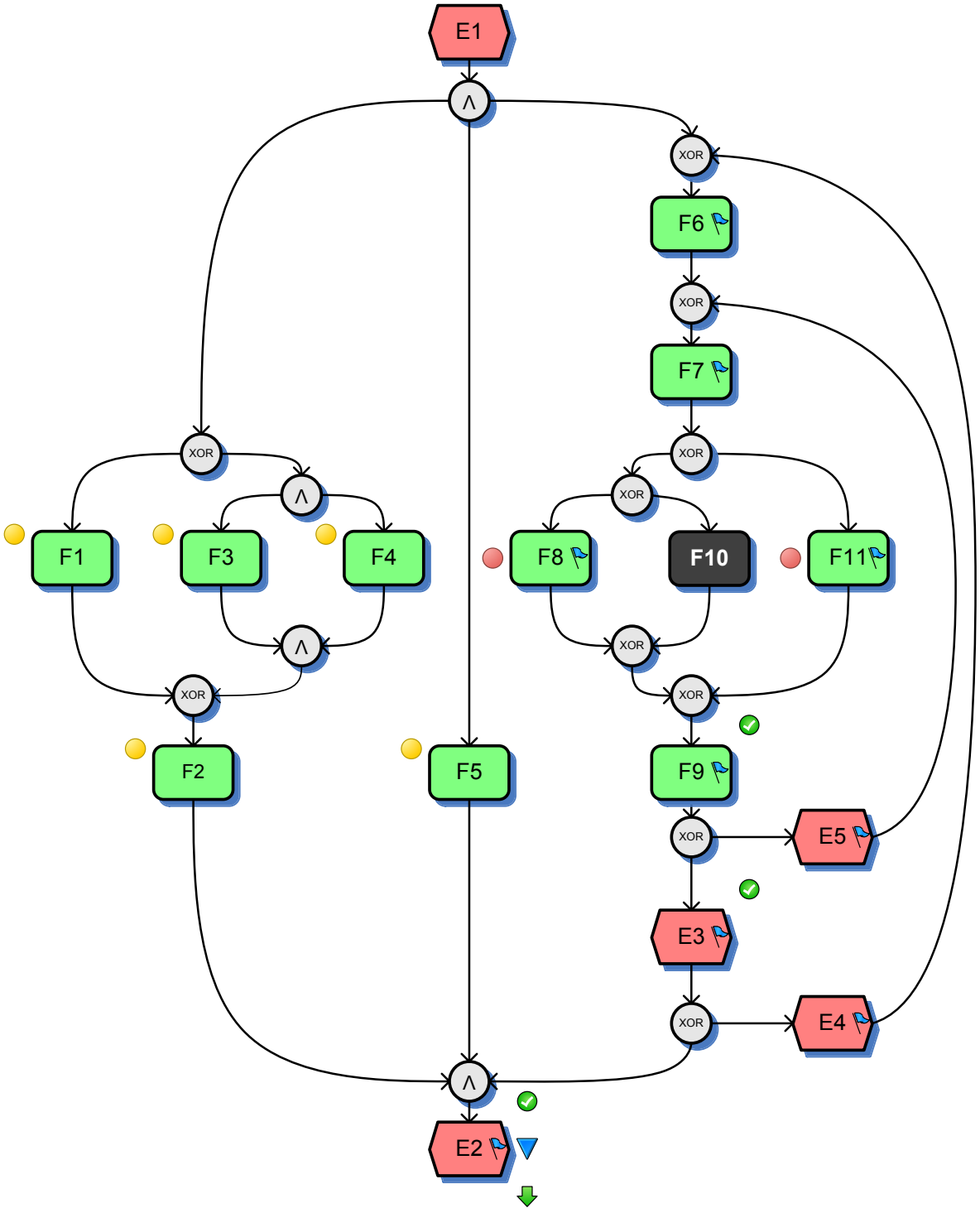
Example 3: Show nodes that can be reached from F6.



Legend:

= flow, = flow_all, = precedes_all, = flow_all_strict, = precedes, = flow_strict, = parallel_to

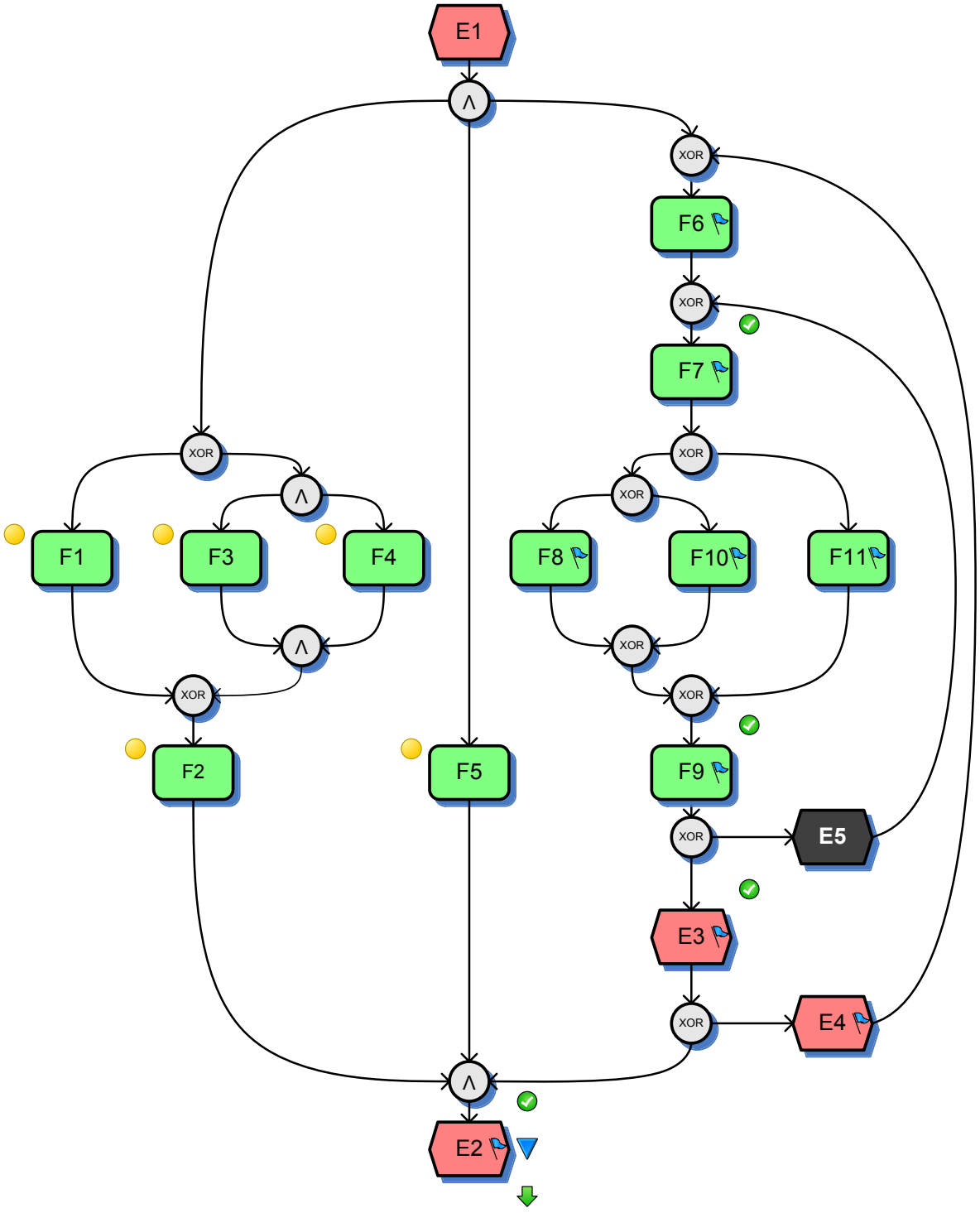
Example 4: Show nodes that can be reached from F10.



Legend:

= flow, = flow_all, = flow_all_strict, = flow_strict, = parallel_to, = exclusive_to

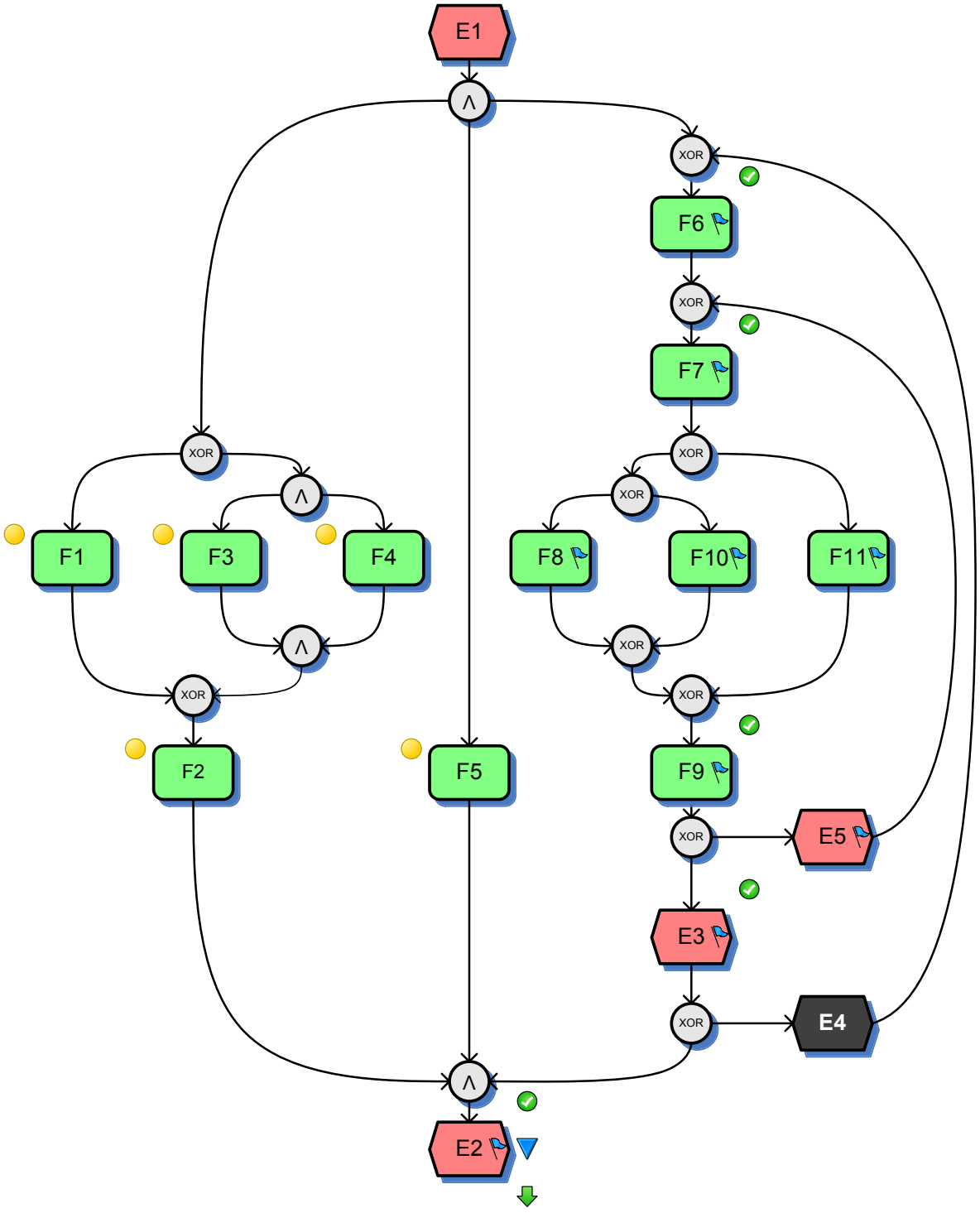
Example 5: Show nodes that can be reached from E5.



Legend:

= flow, = flow_all, = flow_all_strict, = flow_strict, = parallel_to

Example 6: Show nodes that can be reached from E4.



Legend:

= flow, = flow_all, = flow_all_strict, = flow_strict, = parallel_to

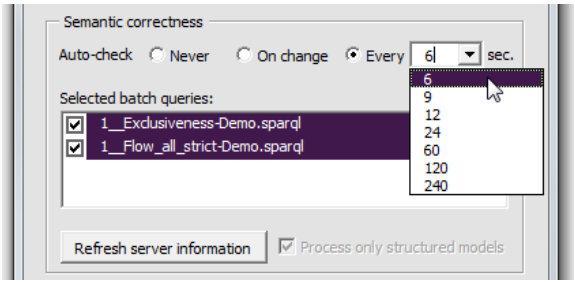
8.4 SEMV-4: Querying with Auto-save and Auto-check

Using this setting allows you to explore the impact of your queries using both the SemQuu Server and the SemQuu Add-In in a highly interactive manner.

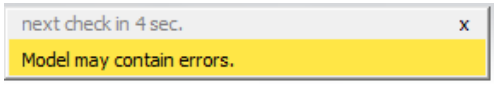
Step 1. Start Visio with the SemQuu Add-In and load the model which you want to use.

Step 2. Start the SemQuu Server. Perform the Steps 1-2 of SEMV-2 in order to load an ontology for which you want to construct and run a batch query. Make sure that "Auto-save" is enabled when inserting the batch query.

Step 3. Go to the "Options" dialog of the SemQuu Add-In. In the "Semantic correctness"-section, select "Every" and then the default setting of "6 sec." (or more) as shown below.



After closing the dialog, the model is checked automatically in the specified interval (e.g. 6 sec.). In order to show when the model is checked the next time, the status window on top of the Viso window displays the seconds left before the next check (at the top of the small window, in light grey), as show below.



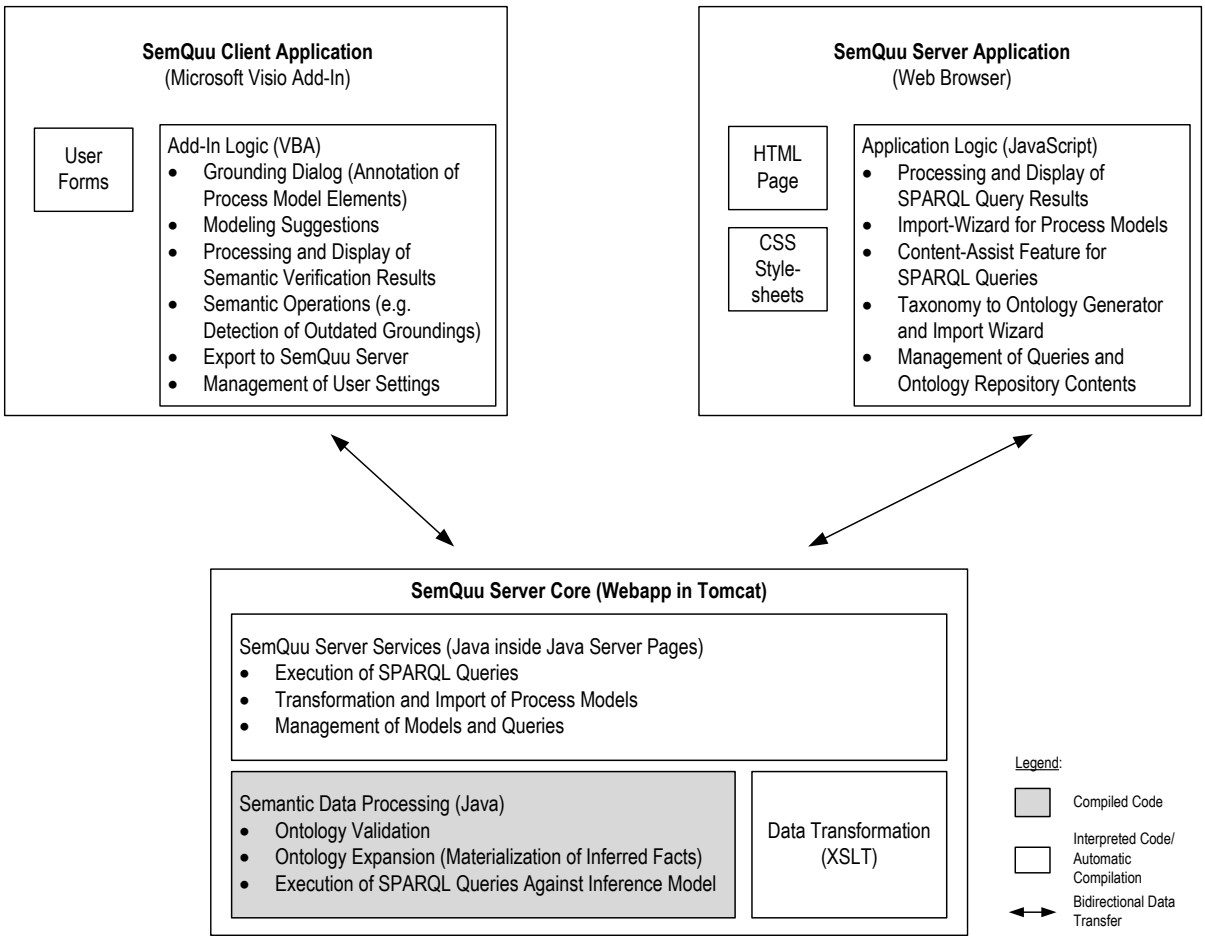
You can easily switch off Auto-check by simply closing the status window or by disabling the option going the SemQuu "Options" dialog.

Step 4. Activate the SemQuu Server window. Modify the query as you want. After modification, click outside the textarea. The query is saved which is indicated by a star "*" appearing for a short time in the button label of the "Save"-button, as shown below.

Step 5. Activate the Visio window with the SemQuu Add-In and see the nodes in the model which are matched by the query (for more information, see also SOPR-1).

9 Software Architecture and Extensibility of SemQuu

The logical architecture of SemQuu shown below provides an overview of the main code components that can be divided into three parts – SemQuu Client Application, SemQuu Server Application and SemQuu Server Core.



In the architecture shown above, the main functions and features of the components have been listed inside the rectangle representing a component using bullet points. SemQuu can be extended or adapted easily since most of the functions and features are implemented in code that is interpreted or compiled automatically. The only code that has to be compiled manually are the Java classes for "Semantic Data Processing". To compile these classes after changes have been made, you will need to include all the .jar-libraries in your Java classpath that are located in the Tomcat folder "$TomcatBaseDir/webapps/semquu/WEB-INF/lib". At the time of this writing it has not yet been decided if there will be a public version of SemQuu that will be hosted on a platform such as GitHub in the future. In the meantime, if you are interested in testing and/or extending SemQuu, please write to michael@mfellann.net.

APPENDIX

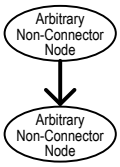
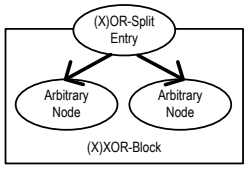
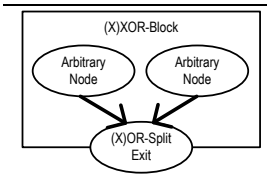
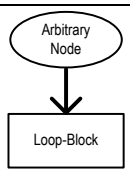
A1 Connection Rules of the Ontology-based Process Representation

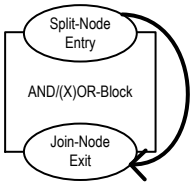
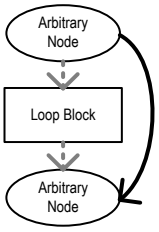
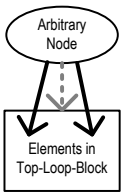
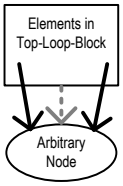
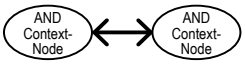
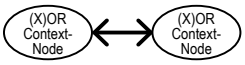
The following table illustrates the rules applied when a process model is transformed to a corresponding ontology-based representation. In the leftmost column, constructs of process models are visualized such as single nodes (oval circles) or blocks (rectangles). The latter are an abstraction that represents a combination of a split and a join node that form e.g. a parallel block (AND), an alternative (XOR, OR) or a Loop. Black arrows symbolize arbitrary flow relations. The columns *No.*, *Name* and *Description* provide a number, a short hand name and a description of the rule.

Whether a specific flow relation is inserted in the ontology-based representation depending on the nodes being either inside or outside a loop as well as on additional conditions is indicated in the three rightmost columns of the table named *Condition*, *Outside Loop*, and *In Loop*. The properties that are inserted into the ontology-based representation are identified by the following symbols:

= flow_all, = precedes_all, = flow_all_strict, = precedes, = flow_strict, = parallel_to, = exclusive_to/multichoice_to, = exclusive_to_strict/multichoice_to_strict.

Note that only the most specific properties have to be inserted into the ontology-based representation. The more general ones can be inferred from the more specific ones, see the property hierarchy in section 7.1. Note too that all properties are transitive. In addition, parallel_to, exclusive_to, multichoice_to, exclusive_to_strict, multichoice_to_strict are transitive and symmetric.

Visualization	*No.*	*Name*	*Description*	*Condition*	*Outside Loop*	*In Loop*
	R1	**Default Flow Rule**	Connect nodes occurring in a sequence.	–	▲ ▼	● ▲
	R2	**(X)OR-Block Entry to Sucessors**	Connect the (X)OR-split entry of the (X)OR-block to all its successors.	–	▲ ▼	▲
	R3	**Predecessors to (X)OR-Block Exit**	Connect the (X)OR-split exit of the (X)OR-block to all its predecessors.	–	▼	●
	R4	**Predecessor to Loop-Block**	Connect the predecessor to the loop-block.	–	▼ ▲	● ▲

	R5	**Bypass AND/(X)OR Block**	Connect the split-node entry of the block to the join-node exit of the block.	–		
	R6	**Bypass Loop Block**	Connect the predecessor to the successor node of the loop-block.	–		
	R7	**Predecessor of Top-Loop-Block to all Nodes Within**	Connect the predecessor node to all nodes within the top-loop-block. Different connections have to be set depending on the nodes inside the top-loop being located in nested (X)OR-blocks or not.	Outside nested (X)OR-Block Inside nested (X)OR-Block	 	(Occurs only outside a loop) (Occurs only outside a loop)
	R8	**All Nodes within Top-Loop-Block to Successor**	Connect all nodes inside the top-loop-block to the successor node of this block.	–	(Occurs only inside a loop)	
	R9	**Connection between Logical AND-Contexts**	Connect nodes representing logical AND contexts (i.e. branches of parallelism) with each other.	–		
	R10	**Connection between Logical (X)OR-Contexts**	Connect nodes representing logical (X)OR contexts (i.e. branches of decisions) with each other.	–		

We briefly summarize the rules contained in the table above informally using natural language. The representation of the control flow relations depends on whether the constructs are part of a loop or not. Inside loops, no strict order of the elements can be guaranteed since the loop may be executed several times. As a rule of thumb, all “strict”-relations do not occur inside loops.

R1 is executed if the source node and the target node should be connected by a flow property and R2-R4 are not applicable. R2 expresses that in contrast to R1 relations with the suffix “_all” (indicating that all tokens will pass that path) should not be established since the source node is a logical (X)OR decision. Hence not all branches will receive a token. R3 expresses that in contrast to R1 *precedes* and *precedes_all* does not hold since from the perspective of the target node it cannot be determined which source node will receive a token inside the XOR-block. R4 expresses that in contrast to R1 *precedes_all* does not hold if the source node is outside the loop and the target inside the loop.

This is due to the fact that inside a loop nodes may be executed more than once so that not all executions are preceded with the execution the predecessor of the loop.

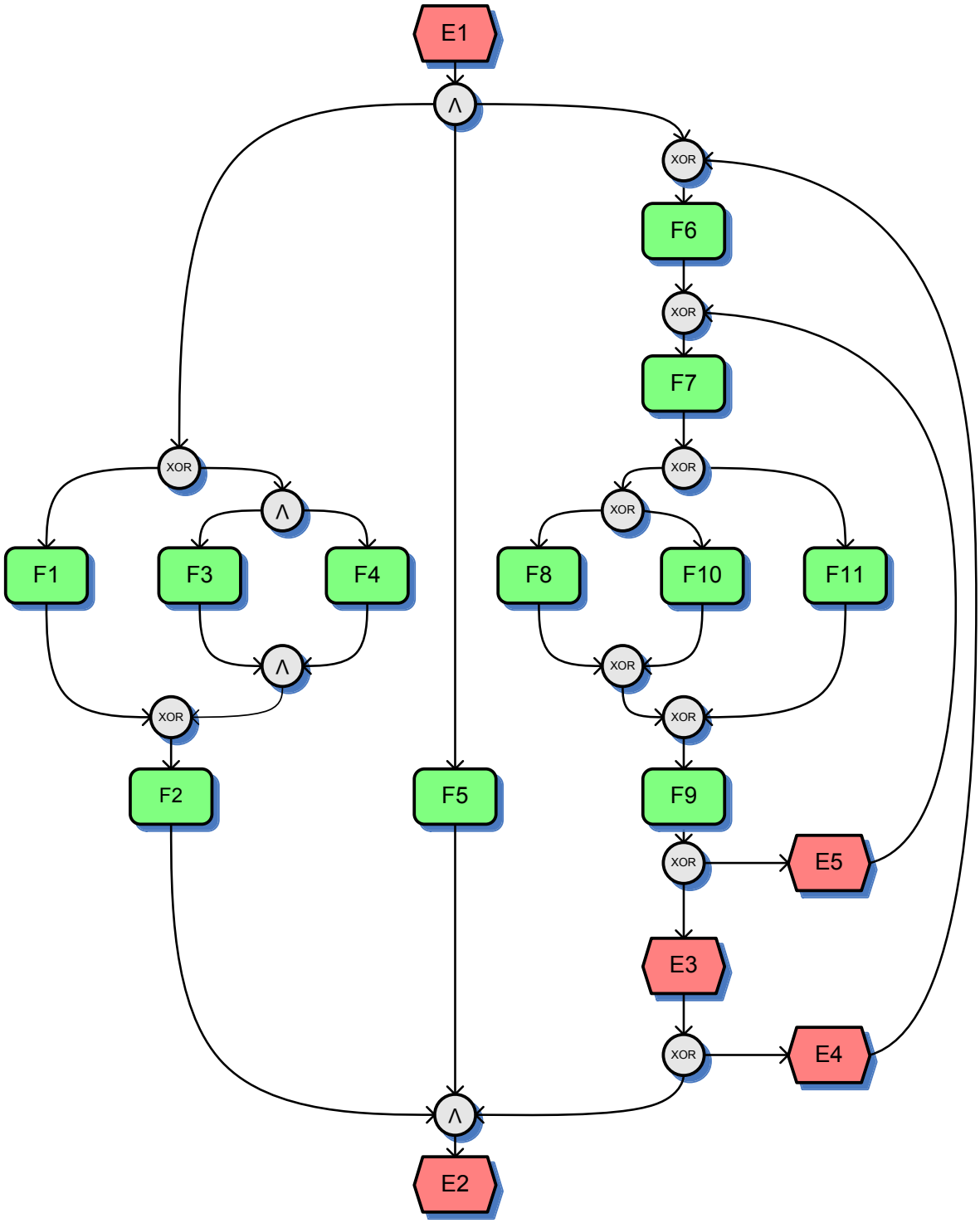
Control flow connections that are inserted into the ontology-based process representation due to the Rules R5 and R6 have no counterpart in the original process model. R5 and R6 ensure that the connections specified by R1 are continued after an (X)OR-block or a loop-block and propagate through the model. In R6, the predecessors and successors are used instead of the split- and join nodes since the latter are inside the loop.

Similarly to R5 and R6, control flow connections that are inserted into the ontology-based process representation due to the Rules R7 and R8 have no counterpart in the original process model. R7 expresses that there is a strict order relation between the predecessor of a loop (the source) and all the nodes inside the loop (the targets). Thereby it has to be differentiated if the nodes in the loop are nested in a (X)OR-context. If not, then all tokens passing the predecessor of the loop (the source) are also passing the node inside the loop (the target). If yes, then this cannot be guaranteed, hence the relation with suffix "_all" cannot be established. Moreover, since the source node is outside and the target node is inside the loop, *precedes_all* does not hold (analogous to R4). R8 expresses that each token circulating inside the loop will eventually exit the loop (provided that livelocks are prohibited) and hence pass the successor of the loop-block.

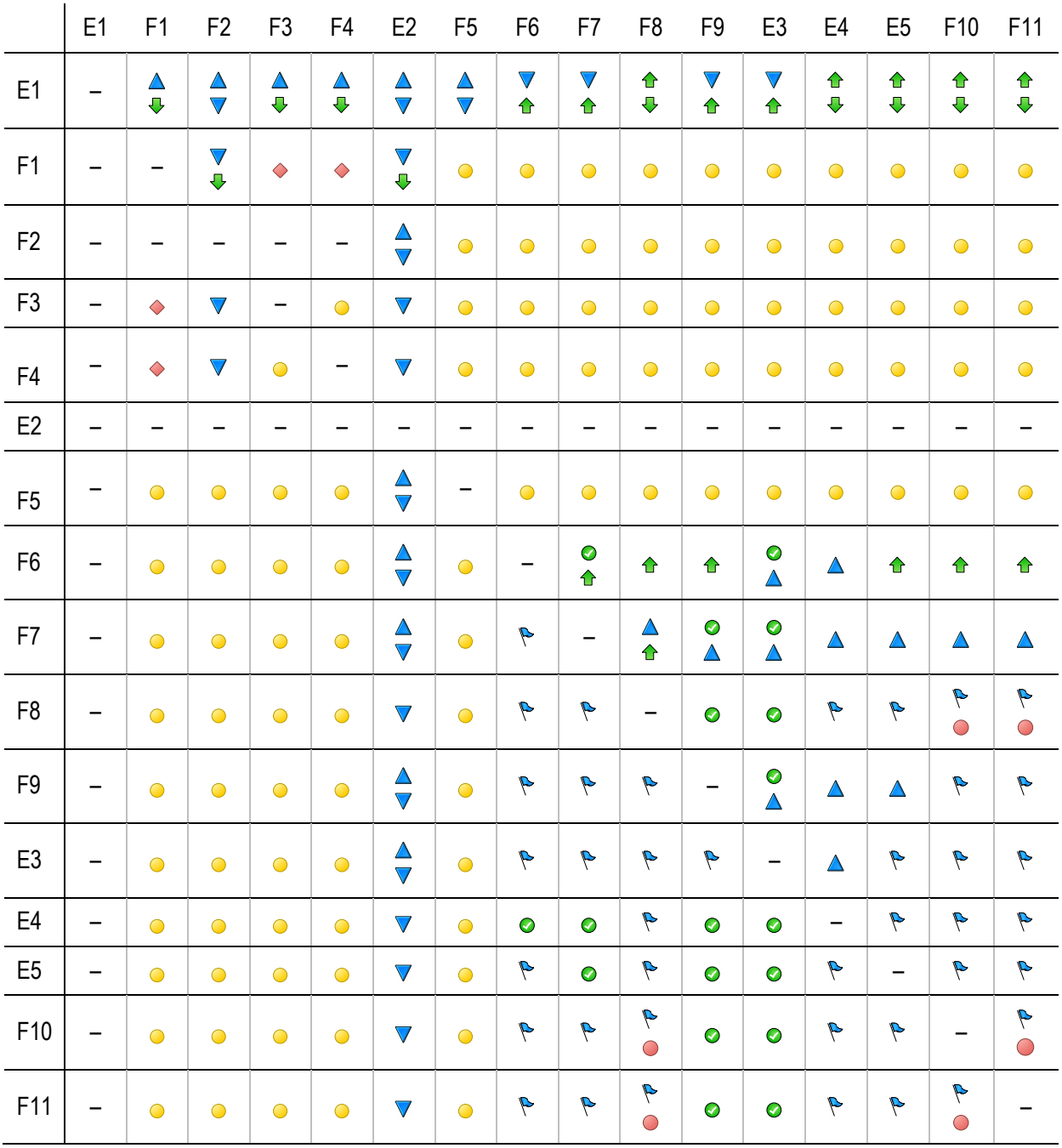
Similarly to R5-8, control flow connections that are inserted into the ontology-based process representation due to the Rules R9 and R10 have no counterpart in the original process model. R9 and R10 represent relations between nodes representing logical contexts, that is, branches inside an AND/(X)OR-block. Whereas the parallel relation is independent of loops, the relations specifying an exclusive or inclusive alternative are dependent on being inside a loop or not. If yes, then relations with the suffix "_strict" do not hold since the loop can be executed multiple times enabling the subsequent activation of all nodes on all branches (logical contexts) inside the loop-block.

A2 Sample Ontology-based Process Logic Representation

In the following tables, the complete control flow for the process model shown below is given in the form of a matrix. The process model is the same as the sample used in 8.3 “SEMV-3: Visualize Node Relations”.



The table below shows the explicitly declared relationships between the nodes. Note that due to the hierarchy of properties (see section 7.1.) additional relations can be inferred by the inference engine.



Legend:

= flow_all, = precedes_all, = flow_all_strict, = precedes, = flow_strict, = parallel_to, = exclusive_to/multichoice_to, = exclusive_to_strict/multichoice_to_strict.

In the next table, the complete information that is available for querying when the ontology is expanded by a reasoner is shown.

	E1	F1	F2	F3	F4	E2	F5	F6	F7	F8	F9	E3	E4	E5	F10	F11
E1	–															
F1	–	–														
F2	–	–	–	–	–											
F3	–			–												
F4	–				–											
E2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
F5	–						–									
F6	–							–								
F7	–								–							

	E1	F1	F2	F3	F4	E2	F5	F6	F7	F8	F9	E3	E4	E5	F10	F11
F8	–	●	●	●	●	⚑ ✓ ▼ ↓	●	⚑	⚑	–	⚑ ✓	⚑ ✓	⚑	⚑	⚑ ●	⚑ ●
F9	–	●	●	●	●	⚑ ✓ ▲ ▼ ↑ ↓	●	⚑	⚑	⚑	–	⚑ ✓ ▲ ↑	⚑ ▲ ↑	⚑ ▲ ↑	⚑	⚑
E3	–	●	●	●	●	⚑ ✓ ▲ ▼ ↑ ↓	●	⚑	⚑	⚑	⚑	–	⚑ ▲ ↑	⚑	⚑	⚑
E4	–	●	●	●	●	⚑ ✓ ▼ ↓	●	⚑ ✓	⚑ ✓	⚑	⚑ ✓	⚑ ✓	–	⚑	⚑	⚑
E5	–	●	●	●	●	⚑ ✓ ▼ ↓	●	⚑	⚑ ✓	⚑	⚑ ✓	⚑ ✓	⚑	–	⚑	⚑
F10	–	●	●	●	●	⚑ ✓ ▼ ↓	●	⚑	⚑	⚑ ●	⚑ ✓	⚑ ✓	⚑	⚑	–	⚑ ●
F11	–	●	●	●	●	⚑ ✓ ▼ ↓	●	⚑	⚑	⚑ ●	⚑ ✓	⚑ ✓	⚑	⚑	⚑ ●	–

A3 Ontology Generation Example

Below is the listing of the SUN-ontology (Simple Understanding of the New Features Ontology) used throughout this document. Note that the textual input is contained in the generated ontology in a CDATA-section inside a `rdfs:comment`-element.

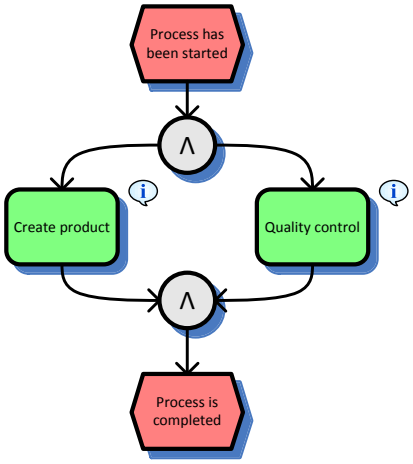
```
<!DOCTYPE rdf:RDF [ <!ENTITY bpm "http://semantic-business.org/2012/bpm.owl#"> ]>

<rdf:RDF xmlns="http://semantic-business.org/2012/process-taxonomy.owl#"
     xml:base="http://semantic-business.org/2012/process-taxonomy.owl"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:bpm="http://semantic-business.org/2012/bpm.owl#">

   <owl:Ontology rdf:about="http://localhost:8080/semquu/repository/1__process-taxonomy.owl">
      <owl:imports rdf:resource="http://localhost:8080/semquu/repository/bpm.owl"/>
      <rdfs:comment>
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<![CDATA[
// To create the ontology, copy a hierarchy of processes (functions, activities, tasks) into this textarea.
//
//  - Prefix each line with an ordering number, use tabs for indenting.
//  - All text after | in a line will be transformed to a rdfs:comment.
//  - afer the # symbol, optionally specify one of the following suggestion modes:
//     sequence, parallel, alternative, multichoice, loop, unconnected. Sequence is the default.
//  - Blank lines and lines starting with two slashes will be omitted.

//  Example:
//  -------

1 Standard project process | This is our reference model
  1.1 Set up the project # parallel
      1.1.1 Identify customer requirements
      1.1.2 Identify legal requirements
  1.2 Create product | Here, the key value is created
      1.2.1 Plan product
      1.2.2 Assign ressources
      1.2.3 Manufacture components
  1.3 Install product at customer site | Additional services # loop
      1.3.1 Install component
      1.3.2 Check if all components have been installed
  1.4 Quality control # alternative
      1.4.1 Decide evaluation method
      1.4.2 Perform on-site evaluation
      1.4.3 Ask customer via phone
  1.5 Send invoice
2 Next Process
]]>
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
      </rdfs:comment>
   </owl:Ontology>

   <bpm:EnterpriseProcess rdf:about="#process_1-StandardProjectProcess">
      <rdfs:label>1 Standard project process</rdfs:label>
      <rdfs:comment>This is our reference model</rdfs:comment>
      <bpm:has_collection>
         <bpm:SubprocessCollection rdf:about="#collection_1" bpm:has_project="standard">
            <bpm:has_pattern rdf:resource="&bpm;sequence"/>
            <bpm:has_item>
               <bpm:Item rdf:about="#item_1.1" bpm:has_no="1000" >
                  <bpm:has_ref rdf:resource="#process_1.1-SetUpTheProject"/>
               </bpm:Item>
            </bpm:has_item>
            <bpm:has_item>
               <bpm:Item rdf:about="#item_1.2" bpm:has_no="2000" >
                  <bpm:has_ref rdf:resource="#process_1.2-CreateProduct"/>
               </bpm:Item>
            </bpm:has_item>
            <bpm:has_item>
               <bpm:Item rdf:about="#item_1.3" bpm:has_no="3000" >
                  <bpm:has_ref rdf:resource="#process_1.3-InstallProductAtCustomerSite"/>
               </bpm:Item>
            </bpm:has_item>
            <bpm:has_item>
               <bpm:Item rdf:about="#item_1.4" bpm:has_no="4000" >
```

```
                <bpm:has_ref rdf:resource="#process_1.4-QualityControl"/>
              </bpm:Item>
            </bpm:has_item>
            <bpm:has_item>
              <bpm:Item rdf:about="#item_1.5" bpm:has_no="5000" >
                <bpm:has_ref rdf:resource="#process_1.5-SendInvoice"/>
              </bpm:Item>
            </bpm:has_item>
          </bpm:SubprocessCollection>
        </bpm:has_collection>
    </bpm:EnterpriseProcess>

    <bpm:EnterpriseProcess rdf:about="#process_1.1-SetUpTheProject">
        <rdfs:label>1.1 Set up the project</rdfs:label>
        <bpm:has_collection>
          <bpm:SubprocessCollection rdf:about="#collection_1.1" bpm:has_project="standard">
            <bpm:has_pattern rdf:resource="&bpm;parallel"/>
            <bpm:has_item>
              <bpm:Item rdf:about="#item_1.1.1" bpm:has_no="1000" >
                <bpm:has_ref rdf:resource="#process_1.1.1-IdentifyCustomerRequirements"/>
              </bpm:Item>
            </bpm:has_item>
            <bpm:has_item>
              <bpm:Item rdf:about="#item_1.1.2" bpm:has_no="2000" >
                <bpm:has_ref rdf:resource="#process_1.1.2-IdentifyLegalRequirements"/>
              </bpm:Item>
            </bpm:has_item>
          </bpm:SubprocessCollection>
        </bpm:has_collection>
    </bpm:EnterpriseProcess>

    <bpm:EnterpriseProcess rdf:about="#process_1.1.1-IdentifyCustomerRequirements">
        <rdfs:label>1.1.1 Identify customer requirements</rdfs:label>
    </bpm:EnterpriseProcess>

    <bpm:EnterpriseProcess rdf:about="#process_1.1.2-IdentifyLegalRequirements">
        <rdfs:label>1.1.2 Identify legal requirements</rdfs:label>
    </bpm:EnterpriseProcess>

    <bpm:EnterpriseProcess rdf:about="#process_1.2-CreateProduct">
        <rdfs:label>1.2 Create product</rdfs:label>
        <rdfs:comment>Here, the key value is created</rdfs:comment>
        <bpm:has_collection>
          <bpm:SubprocessCollection rdf:about="#collection_1.2" bpm:has_project="standard">
            <bpm:has_pattern rdf:resource="&bpm;sequence"/>
            <bpm:has_item>
              <bpm:Item rdf:about="#item_1.2.1" bpm:has_no="1000" >
                <bpm:has_ref rdf:resource="#process_1.2.1-PlanProduct"/>
              </bpm:Item>
            </bpm:has_item>
            <bpm:has_item>
              <bpm:Item rdf:about="#item_1.2.2" bpm:has_no="2000" >
                <bpm:has_ref rdf:resource="#process_1.2.2-AssignRessources"/>
              </bpm:Item>
            </bpm:has_item>
            <bpm:has_item>
              <bpm:Item rdf:about="#item_1.2.3" bpm:has_no="3000" >
                <bpm:has_ref rdf:resource="#process_1.2.3-ManufactureComponents"/>
              </bpm:Item>
            </bpm:has_item>
          </bpm:SubprocessCollection>
        </bpm:has_collection>
    </bpm:EnterpriseProcess>

    <bpm:EnterpriseProcess rdf:about="#process_1.2.1-PlanProduct">
        <rdfs:label>1.2.1 Plan product</rdfs:label>
    </bpm:EnterpriseProcess>

    <bpm:EnterpriseProcess rdf:about="#process_1.2.2-AssignRessources">
        <rdfs:label>1.2.2 Assign ressources</rdfs:label>
    </bpm:EnterpriseProcess>

    <bpm:EnterpriseProcess rdf:about="#process_1.2.3-ManufactureComponents">
        <rdfs:label>1.2.3 Manufacture components</rdfs:label>
    </bpm:EnterpriseProcess>

    <bpm:EnterpriseProcess rdf:about="#process_1.3-InstallProductAtCustomerSite">
        <rdfs:label>1.3 Install product at customer site</rdfs:label>
```

```
    <rdfs:comment>Additional services</rdfs:comment>
    <bpm:has_collection>
      <bpm:SubprocessCollection rdf:about="#collection_1.3" bpm:has_project="standard">
        <bpm:has_pattern rdf:resource="&bpm;loop"/>
        <bpm:has_item>
          <bpm:Item rdf:about="#item_1.3.1" bpm:has_no="1000" >
            <bpm:has_ref rdf:resource="#process_1.3.1-InstallComponent"/>
          </bpm:Item>
        </bpm:has_item>
        <bpm:has_item>
          <bpm:Item rdf:about="#item_1.3.2" bpm:has_no="2000" >
            <bpm:has_ref rdf:resource="#process_1.3.2-CheckIfAllComponentsHaveBeenInstalled"/>
          </bpm:Item>
        </bpm:has_item>
      </bpm:SubprocessCollection>
    </bpm:has_collection>
  </bpm:EnterpriseProcess>

  <bpm:EnterpriseProcess rdf:about="#process_1.3.1-InstallComponent">
    <rdfs:label>1.3.1 Install component</rdfs:label>
  </bpm:EnterpriseProcess>

  <bpm:EnterpriseProcess rdf:about="#process_1.3.2-CheckIfAllComponentsHaveBeenInstalled">
    <rdfs:label>1.3.2 Check if all components have been installed</rdfs:label>
  </bpm:EnterpriseProcess>

  <bpm:EnterpriseProcess rdf:about="#process_1.4-QualityControl">
    <rdfs:label>1.4 Quality control</rdfs:label>
    <bpm:has_collection>
      <bpm:SubprocessCollection rdf:about="#collection_1.4" bpm:has_project="standard">
        <bpm:has_pattern rdf:resource="&bpm;alternative"/>
        <bpm:has_item>
          <bpm:Item rdf:about="#item_1.4.1" bpm:has_no="1000" >
            <bpm:has_ref rdf:resource="#process_1.4.1-DecideEvaluationMethod"/>
          </bpm:Item>
        </bpm:has_item>
        <bpm:has_item>
          <bpm:Item rdf:about="#item_1.4.2" bpm:has_no="2000" >
            <bpm:has_ref rdf:resource="#process_1.4.2-PerformOnsiteEvaluation"/>
          </bpm:Item>
        </bpm:has_item>
        <bpm:has_item>
          <bpm:Item rdf:about="#item_1.4.3" bpm:has_no="3000" >
            <bpm:has_ref rdf:resource="#process_1.4.3-AskCustomerViaPhone"/>
          </bpm:Item>
        </bpm:has_item>
      </bpm:SubprocessCollection>
    </bpm:has_collection>
  </bpm:EnterpriseProcess>

  <bpm:EnterpriseProcess rdf:about="#process_1.4.1-DecideEvaluationMethod">
    <rdfs:label>1.4.1 Decide evaluation method</rdfs:label>
  </bpm:EnterpriseProcess>

  <bpm:EnterpriseProcess rdf:about="#process_1.4.2-PerformOnsiteEvaluation">
    <rdfs:label>1.4.2 Perform on-site evaluation</rdfs:label>
  </bpm:EnterpriseProcess>

  <bpm:EnterpriseProcess rdf:about="#process_1.4.3-AskCustomerViaPhone">
    <rdfs:label>1.4.3 Ask customer via phone</rdfs:label>
  </bpm:EnterpriseProcess>

  <bpm:EnterpriseProcess rdf:about="#process_1.5-SendInvoice">
    <rdfs:label>1.5 Send invoice</rdfs:label>
  </bpm:EnterpriseProcess>

  <bpm:EnterpriseProcess rdf:about="#process_2-NextProcess">
    <rdfs:label>2 Next Process</rdfs:label>
  </bpm:EnterpriseProcess>
</rdf:RDF>
```

A3 Simple EPC to OWL Conversion Example

The conversion of an EPC in EPML-format to the ontology-based process representation is illustrated by a simple example. The graphical model is shown below.



EMPL generated for the example model:

```
<epml:epml xmlns:epml="http://www.epml.de" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.epml.de EPML_111_draft.xsd">
 <coordinates xOrigin="leftToRight" yOrigin="topToBottom"/>
<definitions>
 <definition defId="1"/><definition defId="24"/><definition defId="6"/><definition defId="12"/>
 <definition defId="8"/><definition defId="9"/></definitions><attributeTypes><attributeType typeId="rdfId"/>
 <attributeType typeId="annotationRelation"/><attributeType typeId="ontologyClassId"/>
 <attributeType typeId="ontologyInstanceId"/></attributeTypes>
  <directory>
    <epc name="simpledemo" epcId="561040169448539137">
          <event id="1" defRef="1">
             <name>Process has been started</name>
             <attribute typeRef="rdfId" value="Process_has_been_started___simpledemo.1"/>
             <attribute typeRef="annotationRelation" value="equivalent_to"/>
             <attribute typeRef="ontologyClassId"
             value="http://semantic-business.org/2012/bpm.owl#Event"/>
             <attribute typeRef="ontologyInstanceId"
             value="http://semantic-business.org/2012/bpm.owl#ProcessHasBeenStarted"/>
        </event>
         <event id="24" defRef="24">
             <name>Process is completed</name>
             <attribute typeRef="rdfId" value="Process_is_completed___simpledemo.24"/>
             <attribute typeRef="annotationRelation" value="equivalent_to"/>
             <attribute typeRef="ontologyClassId" value="http://semantic-business.org/2012/bpm.owl#Event"/>
             <attribute typeRef="ontologyInstanceId"
             value="http://semantic-business.org/2012/bpm.owl#event_ProcessIsCompleted"/>
        </event>
         <function id="6" defRef="6">
             <name>Create product</name>
             <attribute typeRef="rdfId" value="Create_product___simpledemo.6"/>
             <attribute typeRef="annotationRelation" value="equivalent_to"/>
             <attribute typeRef="ontologyClassId"
             value="http://semantic-business.org/2012/process-taxonomy.owl#process_1-StandardProjectProcess"/>
             <attribute typeRef="ontologyInstanceId"
             value="http://semantic-business.org/2012/process-taxonomy.owl#process_1.2-CreateProduct"/>
        </function>
         <function id="12" defRef="12">
             <name>Quality control</name>
             <attribute typeRef="rdfId" value="Quality_control___simpledemo.12"/>
             <attribute typeRef="annotationRelation" value="equivalent_to"/>
             <attribute typeRef="ontologyClassId"
             value="http://semantic-business.org/2012/process-taxonomy.owl#process_1-StandardProjectProcess"/>
             <attribute typeRef="ontologyInstanceId"
             value="http://semantic-business.org/2012/process-taxonomy.owl#process_1.4-QualityControl"/>
        </function>
         <and id="8" defRef="8">
             <name>AND</name>
             <attribute typeRef="rdfId" value="AND___simpledemo.8"/>
```

```
            </and>
              <and id="9" defRef="9">
                  <name>AND</name>
                  <attribute typeRef="rdfId" value="AND___simpledemo.9"/>
            </and>
             <arc id="7"><flow source="6" target="9"/></arc>
            <arc id="13"><flow source="12" target="9"/></arc>
            <arc id="2"><flow source="8" target="12"/></arc>
            <arc id="4"><flow source="1" target="8"/></arc>
            <arc id="3"><flow source="8" target="6"/></arc>
            <arc id="5"><flow source="9" target="24"/></arc>
        </epc>
    </directory>
</epml:epml>
```

The transformation into the ontology-based representation (see EXPT-1) will result in the following OWL ontology:

```
<!DOCTYPE rdf:RDF [
    <!ENTITY bpm "http://semantic-business.org/2012/bpm.owl#">
    <!ENTITY pt "http://semantic-business.org/2012/process-taxonomy.owl#">
    <!ENTITY rep "http://localhost:8080/semquu/repository/">
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
]>

<rdf:RDF xmlns="&rep;2__main.owl#" xml:base="&rep;2__main.owl"
    xmlns:bpm="http://semantic-business.org/2012/bpm.owl#"
    xmlns:pt="http://semantic-business.org/2012/process-taxonomy.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<owl:Ontology rdf:about="&rep;2_simpledemo.owl">
    <owl:imports rdf:resource="&rep;2__process-taxonomy.owl"/>
</owl:Ontology>

<bpm:ProcessGraph rdf:ID="2_simpledemo">
    <bpm:is_structured rdf:datatype="&xsd;boolean">true</bpm:is_structured>
</bpm:ProcessGraph>

<bpm:StartEvent rdf:ID="Process_has_been_started___simpledemo.1">
    <rdfs:label>Process has been started</rdfs:label>
    <bpm:graphPart rdf:resource="#2_simpledemo"/>
    <bpm:equivalent_to rdf:resource="&bpm;ProcessHasBeenStarted"/>
    <!-- to: AND -->
    <bpm:has_after_AND rdf:resource="#AND___simpledemo.8"/>
    <bpm:flow_all_strict rdf:resource="#AND___simpledemo.8"/>
    <bpm:precedes_all rdf:resource="#AND___simpledemo.8"/>
</bpm:StartEvent>

<bpm:ParallelSplit rdf:ID="AND___simpledemo.8">
    <rdfs:label>AND</rdfs:label>
    <bpm:graphPart rdf:resource="#2_simpledemo"/>
    <!-- to: Create product -->
    <bpm:has_after_function rdf:resource="#Create_product___simpledemo.6"/>
    <bpm:precedes_all rdf:resource="#Create_product___simpledemo.6"/>
    <bpm:flow_all_strict rdf:resource="#Create_product___simpledemo.6"/>
    <!-- to: Quality control -->
    <bpm:has_after_function rdf:resource="#Quality_control___simpledemo.12"/>
    <bpm:precedes_all rdf:resource="#Quality_control___simpledemo.12"/>
    <bpm:flow_all_strict rdf:resource="#Quality_control___simpledemo.12"/>
</bpm:ParallelSplit>

<bpm:ContextNode rdf:ID="CONTEXT___simpledemo.a8-1">
    <bpm:is_parallel_to rdf:resource="#CONTEXT___simpledemo.a8-2"/>
</bpm:ContextNode>

<bpm:ActivityNode rdf:ID="Create_product___simpledemo.6">
    <rdfs:label>Create product</rdfs:label>
    <bpm:graphPart rdf:resource="#2_simpledemo"/>
    <bpm:equivalent_to rdf:resource="&pt;process_1.2-CreateProduct"/>
    <!-- to: AND -->
    <bpm:has_after_AND rdf:resource="#AND___simpledemo.9"/>
    <bpm:flow_all_strict rdf:resource="#AND___simpledemo.9"/>
    <bpm:precedes_all rdf:resource="#AND___simpledemo.9"/>
    <!-- context -->
```

```
    <bpm:has_context rdf:resource="#CONTEXT___simpledemo.a8-1"/>
</bpm:ActivityNode>

<bpm:Synchronization rdf:ID="AND___simpledemo.9">
    <rdfs:label>AND</rdfs:label>
    <bpm:graphPart rdf:resource="#2_simpledemo"/>
    <!-- to: Process is completed -->
    <bpm:has_after_event rdf:resource="#Process_is_completed___simpledemo.24"/>
    <bpm:flow_all_strict rdf:resource="#Process_is_completed___simpledemo.24"/>
    <bpm:precedes_all rdf:resource="#Process_is_completed___simpledemo.24"/>
</bpm:Synchronization>

<bpm:EndEvent rdf:ID="Process_is_completed___simpledemo.24">
    <rdfs:label>Process is completed</rdfs:label>
    <bpm:graphPart rdf:resource="#2_simpledemo"/>
    <bpm:equivalent_to rdf:resource="&bpm;event_ProcessIsCompleted"/>
</bpm:EndEvent>

<!-- Bypass control flow block -->
<owl:NamedIndividual rdf:about="#AND___simpledemo.8">
    <bpm:flow_all_strict rdf:resource="#AND___simpledemo.9" />
    <bpm:precedes_all rdf:resource="#AND___simpledemo.9" />
</owl:NamedIndividual>

<bpm:ContextNode rdf:ID="CONTEXT___simpledemo.a8-2"></bpm:ContextNode>

<bpm:ActivityNode rdf:ID="Quality_control___simpledemo.12">
    <rdfs:label>Quality control</rdfs:label>
    <bpm:graphPart rdf:resource="#2_simpledemo"/>
    <bpm:equivalent_to rdf:resource="&pt;process_1.4-QualityControl"/>
    <!-- to: AND -->
    <bpm:has_after_AND rdf:resource="#AND___simpledemo.9"/>
    <bpm:flow_all_strict rdf:resource="#AND___simpledemo.9"/>
    <bpm:precedes_all rdf:resource="#AND___simpledemo.9"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___simpledemo.a8-2"/>
</bpm:ActivityNode>

</rdf:RDF>

<!-- Generated with SemQuu (transformation mode: advanced) -->
```

A4 Complex Process Model Example

In the following, a conversion of a model to the OWL-based encoding which is generated as described in EXPT-1 is shown. The model is the same as in section 8.3 and in Appendix A2. Since the model only serves to illustrate the correspondence of the graphical model with the constructs of the ontology-based representation, the functions and events have been named with symbolic names. Functions have been annotated with instances having almost the same name like the label. Events have been annotated using slightly more meaningful instances. Below, the OWL-based representation is shown in RDF/XML-syntax. To improve readability, comments have been inserted.

```
<!DOCTYPE rdf:RDF [
   <!ENTITY bpm "http://semantic-business.org/2012/bpm.owl#">
   <!ENTITY pt "http://semantic-business.org/2012/process-taxonomy.owl#">
   <!ENTITY rep "http://localhost:8080/semquu/repository/">
   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
]>
<rdf:RDF xmlns="&rep;2__main.owl#" xml:base="&rep;2__main.owl"
   xmlns:bpm="http://semantic-business.org/2012/bpm.owl#"
   xmlns:pt="http://semantic-business.org/2012/process-taxonomy.owl#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<owl:Ontology rdf:about="&rep;2_maintest.owl">
   <owl:imports rdf:resource="&rep;2__process-taxonomy.owl"/>
</owl:Ontology>

<bpm:ProcessGraph rdf:ID="2_maintest">
   <bpm:is_structured rdf:datatype="&xsd;boolean">true</bpm:is_structured>
</bpm:ProcessGraph>

<bpm:StartEvent rdf:ID="E1___maintest.1">
   <rdfs:label>E1</rdfs:label>
   <bpm:graphPart rdf:resource="#2_maintest"/>
   <bpm:equivalent_to rdf:resource="&bpm;ProcessHasBeenStarted"/>
   <!-- to: AND -->
   <bpm:has_after_AND rdf:resource="#AND___maintest.8"/>
   <bpm:flow_all_strict rdf:resource="#AND___maintest.8"/>
   <bpm:precedes_all rdf:resource="#AND___maintest.8"/>
</bpm:StartEvent>

<bpm:ParallelSplit rdf:ID="AND___maintest.8">
   <rdfs:label>AND</rdfs:label>
   <bpm:graphPart rdf:resource="#2_maintest"/>
   <!-- to: XOR -->
   <bpm:has_after_XOR rdf:resource="#XOR___maintest.18"/>
   <bpm:precedes_all rdf:resource="#XOR___maintest.18"/>
   <bpm:flow_all_strict rdf:resource="#XOR___maintest.18"/>
   <!-- to: F5 -->
   <bpm:has_after_function rdf:resource="#F5___maintest.22"/>
   <bpm:precedes_all rdf:resource="#F5___maintest.22"/>
   <bpm:flow_all_strict rdf:resource="#F5___maintest.22"/>
   <!-- to: XOR -->
   <bpm:has_after_XOR rdf:resource="#XOR___maintest.77"/>
   <bpm:precedes rdf:resource="#XOR___maintest.77"/>
   <bpm:flow_all_strict rdf:resource="#XOR___maintest.77"/>
</bpm:ParallelSplit>

<bpm:ContextNode rdf:ID="CONTEXT___maintest.a8-1">
   <bpm:is_parallel_to rdf:resource="#CONTEXT___maintest.a8-2"/>
</bpm:ContextNode>

<bpm:ExclusiveChoice rdf:ID="XOR___maintest.18">
   <rdfs:label>XOR</rdfs:label>
   <bpm:graphPart rdf:resource="#2_maintest"/>
   <!-- to: AND -->
   <bpm:has_after_AND rdf:resource="#AND___maintest.15"/>
   <bpm:precedes_all rdf:resource="#AND___maintest.15"/>
   <bpm:flow_strict rdf:resource="#AND___maintest.15"/>
   <!-- to: F1 -->
   <bpm:has_after_function rdf:resource="#F1___maintest.19"/>
   <bpm:precedes_all rdf:resource="#F1___maintest.19"/>
   <bpm:flow_strict rdf:resource="#F1___maintest.19"/>
   <!-- context -->
   <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-1"/>
</bpm:ExclusiveChoice>

<bpm:ContextNode rdf:ID="CONTEXT___maintest.x18-1">
   <bpm:is_exclusive_to_strict rdf:resource="#CONTEXT___maintest.x18-2"/>
</bpm:ContextNode>

<bpm:ParallelSplit rdf:ID="AND___maintest.15">
   <rdfs:label>AND</rdfs:label>
   <bpm:graphPart rdf:resource="#2_maintest"/>
   <!-- to: F3 -->
   <bpm:has_after_function rdf:resource="#F3___maintest.2"/>
   <bpm:precedes_all rdf:resource="#F3___maintest.2"/>
   <bpm:flow_all_strict rdf:resource="#F3___maintest.2"/>
   <!-- to: F4 -->
   <bpm:has_after_function rdf:resource="#F4___maintest.3"/>
   <bpm:precedes_all rdf:resource="#F4___maintest.3"/>
   <bpm:flow_all_strict rdf:resource="#F4___maintest.3"/>
   <!-- context -->
   <bpm:has_context rdf:resource="#CONTEXT___maintest.x18-1"/>
   <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-1"/>
</bpm:ParallelSplit>

<bpm:ContextNode rdf:ID="CONTEXT___maintest.a15-1">
   <bpm:is_parallel_to rdf:resource="#CONTEXT___maintest.a15-2"/>
</bpm:ContextNode>

<bpm:ActivityNode rdf:ID="F3___maintest.2">
   <rdfs:label>F3</rdfs:label>
   <bpm:graphPart rdf:resource="#2_maintest"/>
   <bpm:equivalent_to rdf:resource="&pt;process_3-F3"/>
   <!-- to: AND -->
   <bpm:has_after_AND rdf:resource="#AND___maintest.14"/>
   <bpm:flow_all_strict rdf:resource="#AND___maintest.14"/>
   <bpm:precedes_all rdf:resource="#AND___maintest.14"/>
   <!-- context -->
   <bpm:has_context rdf:resource="#CONTEXT___maintest.a15-1"/>
   <bpm:has_context rdf:resource="#CONTEXT___maintest.x18-1"/>
   <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-1"/>
</bpm:ActivityNode>

<bpm:Synchronization rdf:ID="AND___maintest.14">
   <rdfs:label>AND</rdfs:label>
   <bpm:graphPart rdf:resource="#2_maintest"/>
   <!-- to: XOR -->
   <bpm:has_after_XOR rdf:resource="#XOR___maintest.20"/>
   <bpm:flow_all_strict rdf:resource="#XOR___maintest.20"/>
   <!-- context -->
   <bpm:has_context rdf:resource="#CONTEXT___maintest.x18-1"/>
   <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-1"/>
</bpm:Synchronization>

<bpm:SimpleMerge rdf:ID="XOR___maintest.20">
   <rdfs:label>XOR</rdfs:label>
   <bpm:graphPart rdf:resource="#2_maintest"/>
   <!-- to: F2 -->
   <bpm:has_after_function rdf:resource="#F2___maintest.49"/>
   <bpm:flow_all_strict rdf:resource="#F2___maintest.49"/>
   <bpm:precedes_all rdf:resource="#F2___maintest.49"/>
   <!-- context -->
   <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-1"/>
</bpm:SimpleMerge>
```

```
127
128 <bpm:ActivityNode rdf:ID="F2___maintest.49">
129    <rdfs:label>F2</rdfs:label>
130    <bpm:graphPart rdf:resource="#2_maintest"/>
131    <bpm:equivalent_to rdf:resource="&pt;process_2-F2"/>
132    <!-- to: AND -->
133    <bpm:has_after_AND rdf:resource="#AND___maintest.9"/>
134    <bpm:flow_all_strict rdf:resource="#AND___maintest.9"/>
135    <bpm:precedes_all rdf:resource="#AND___maintest.9"/>
136    <!-- context -->
137    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-1"/>
138 </bpm:ActivityNode>
139
140 <bpm:Synchronization rdf:ID="AND___maintest.9">
141    <rdfs:label>AND</rdfs:label>
142    <bpm:graphPart rdf:resource="#2_maintest"/>
143    <!-- to: E2 -->
144    <bpm:has_after_event rdf:resource="#E2___maintest.17"/>
145    <bpm:flow_all_strict rdf:resource="#E2___maintest.17"/>
146    <bpm:precedes_all rdf:resource="#E2___maintest.17"/>
147 </bpm:Synchronization>
148
149 <bpm:EndEvent rdf:ID="E2___maintest.17">
150    <rdfs:label>E2</rdfs:label>
151    <bpm:graphPart rdf:resource="#2_maintest"/>
152    <bpm:equivalent_to rdf:resource="&bpm;event_OrderIsApproved"/>
153 </bpm:EndEvent>
154
155 <!-- Bypass control flow block -->
156 <owl:NamedIndividual rdf:about="#AND___maintest.8">
157    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
158    <bpm:precedes_all rdf:resource="#AND___maintest.9" />
159 </owl:NamedIndividual>
160
161 <!-- Bypass control flow block -->
162 <owl:NamedIndividual rdf:about="#XOR___maintest.18">
163    <bpm:flow_all_strict rdf:resource="#XOR___maintest.20" />
164    <bpm:precedes_all rdf:resource="#XOR___maintest.20" />
165 </owl:NamedIndividual>
166
167 <!-- Bypass control flow block -->
168 <owl:NamedIndividual rdf:about="#AND___maintest.15">
169    <bpm:flow_all_strict rdf:resource="#AND___maintest.14" />
170    <bpm:precedes_all rdf:resource="#AND___maintest.14" />
171 </owl:NamedIndividual>
172
173 <bpm:ContextNode rdf:ID="CONTEXT___maintest.a15-2"></bpm:ContextNode>
174
175 <bpm:ActivityNode rdf:ID="F4___maintest.3">
176    <rdfs:label>F4</rdfs:label>
177    <bpm:graphPart rdf:resource="#2_maintest"/>
178    <bpm:equivalent_to rdf:resource="&pt;process_4-F4"/>
179    <!-- to: AND -->
180    <bpm:has_after_AND rdf:resource="#AND___maintest.14"/>
181    <bpm:flow_all_strict rdf:resource="#AND___maintest.14"/>
182    <bpm:precedes_all rdf:resource="#AND___maintest.14"/>
183    <!-- context -->
184    <bpm:has_context rdf:resource="#CONTEXT___maintest.a15-2"/>
185    <bpm:has_context rdf:resource="#CONTEXT___maintest.x18-1"/>
186    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-1"/>
187 </bpm:ActivityNode>
188
189 <bpm:ContextNode rdf:ID="CONTEXT___maintest.x18-2"></bpm:ContextNode>
190
191 <bpm:ActivityNode rdf:ID="F1___maintest.19">
192    <rdfs:label>F1</rdfs:label>
193    <bpm:graphPart rdf:resource="#2_maintest"/>
194    <bpm:equivalent_to rdf:resource="&pt;process_1-F1"/>
195    <!-- to: XOR -->
196    <bpm:has_after_XOR rdf:resource="#XOR___maintest.20"/>
197    <bpm:flow_all_strict rdf:resource="#XOR___maintest.20"/>
198    <!-- context -->
199    <bpm:has_context rdf:resource="#CONTEXT___maintest.x18-2"/>
200    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-1"/>
201 </bpm:ActivityNode>
202
203 <bpm:ContextNode rdf:ID="CONTEXT___maintest.a8-2">
204    <bpm:is_parallel_to rdf:resource="#CONTEXT___maintest.a8-3"/>
205 </bpm:ContextNode>
206
207 <bpm:ActivityNode rdf:ID="F5___maintest.22">
208    <rdfs:label>F5</rdfs:label>
209    <bpm:graphPart rdf:resource="#2_maintest"/>
210    <bpm:equivalent_to rdf:resource="&pt;process_5-F5"/>
211    <!-- to: AND -->
212    <bpm:has_after_AND rdf:resource="#AND___maintest.9"/>
213    <bpm:flow_all_strict rdf:resource="#AND___maintest.9"/>
214    <bpm:precedes_all rdf:resource="#AND___maintest.9"/>
215    <!-- context -->
216    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-2"/>
217 </bpm:ActivityNode>
218
219 <bpm:ContextNode rdf:ID="CONTEXT___maintest.a8-3"></bpm:ContextNode>
220
221 <bpm:SimpleMerge rdf:ID="XOR___maintest.77">
222    <rdfs:label>XOR</rdfs:label>
223    <bpm:graphPart rdf:resource="#2_maintest"/>
224    <!-- to: F6 -->
225    <bpm:has_after_function rdf:resource="#F6___maintest.16"/>
226    <bpm:flow_all rdf:resource="#F6___maintest.16"/>
227    <bpm:precedes_all rdf:resource="#F6___maintest.16"/>
228    <!-- context -->
229    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
230 </bpm:SimpleMerge>
231
232 <bpm:ActivityNode rdf:ID="F6___maintest.16">
233    <rdfs:label>F6</rdfs:label>
234    <bpm:graphPart rdf:resource="#2_maintest"/>
235    <bpm:equivalent_to rdf:resource="&pt;process_6-F6"/>
236    <!-- to: XOR -->
237    <bpm:has_after_XOR rdf:resource="#XOR___maintest.10"/>
238    <bpm:flow_all rdf:resource="#XOR___maintest.10"/>
239    <!-- context -->
240    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
241 </bpm:ActivityNode>
242
243 <bpm:SimpleMerge rdf:ID="XOR___maintest.10">
244    <rdfs:label>XOR</rdfs:label>
245    <bpm:graphPart rdf:resource="#2_maintest"/>
246    <!-- to: F7 -->
247    <bpm:has_after_function rdf:resource="#F7___maintest.21"/>
248    <bpm:flow_all rdf:resource="#F7___maintest.21"/>
249    <bpm:precedes_all rdf:resource="#F7___maintest.21"/>
250    <!-- context -->
251    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
252 </bpm:SimpleMerge>
253
254 <!-- Connect the predecessor to the XOR-element closing a do-while loop -->
255 <owl:NamedIndividual rdf:about="#F6___maintest.16">
256    <bpm:precedes rdf:resource="#XOR___maintest.10" />
257 </owl:NamedIndividual>
258
259 <!-- Byepass loop with precedes_all relation -->
260 <owl:NamedIndividual rdf:about="#F6___maintest.16">
261    <bpm:precedes_all rdf:resource="#E3___maintest.13" />
262 </owl:NamedIndividual>
263
264 <bpm:ActivityNode rdf:ID="F7___maintest.21">
265    <rdfs:label>F7</rdfs:label>
266    <bpm:graphPart rdf:resource="#2_maintest"/>
267    <bpm:equivalent_to rdf:resource="&pt;process_7-F7"/>
268    <!-- to: XOR -->
269    <bpm:has_after_XOR rdf:resource="#XOR___maintest.72"/>
270    <bpm:flow_all rdf:resource="#XOR___maintest.72"/>
271    <bpm:precedes_all rdf:resource="#XOR___maintest.72"/>
272    <!-- context -->
273    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
274 </bpm:ActivityNode>
275
276 <bpm:ExclusiveChoice rdf:ID="XOR___maintest.72">
277    <rdfs:label>XOR</rdfs:label>
278    <bpm:graphPart rdf:resource="#2_maintest"/>
279    <!-- to: XOR -->
280    <bpm:has_after_XOR rdf:resource="#XOR___maintest.4"/>
281    <bpm:precedes_all rdf:resource="#XOR___maintest.4"/>
282    <!-- to: F11 -->
```

```
    <bpm:has_after_function rdf:resource="#F11___maintest.60"/>
    <bpm:precedes_all rdf:resource="#F11___maintest.60"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
</bpm:ExclusiveChoice>

<bpm:ContextNode rdf:ID="CONTEXT___maintest.x72-1">
    <bpm:is_exclusive_to rdf:resource="#CONTEXT___maintest.x72-2"/>
</bpm:ContextNode>

<bpm:ExclusiveChoice rdf:ID="XOR___maintest.4">
    <rdfs:label>XOR</rdfs:label>
    <bpm:graphPart rdf:resource="#2_maintest"/>
    <!-- to: F8 -->
    <bpm:has_after_function rdf:resource="#F8___maintest.5"/>
    <bpm:precedes_all rdf:resource="#F8___maintest.5"/>
    <!-- to: F10 -->
    <bpm:has_after_function rdf:resource="#F10___maintest.6"/>
    <bpm:precedes_all rdf:resource="#F10___maintest.6"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___maintest.x72-1"/>
    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
</bpm:ExclusiveChoice>

<bpm:ContextNode rdf:ID="CONTEXT___maintest.x4-1">
    <bpm:is_exclusive_to rdf:resource="#CONTEXT___maintest.x4-2"/>
</bpm:ContextNode>

<bpm:ActivityNode rdf:ID="F8___maintest.5">
    <rdfs:label>F8</rdfs:label>
    <bpm:graphPart rdf:resource="#2_maintest"/>
    <bpm:equivalent_to rdf:resource="&pt;process_8-F8"/>
    <!-- to: XOR -->
    <bpm:has_after_XOR rdf:resource="#XOR___maintest.7"/>
    <bpm:flow_all rdf:resource="#XOR___maintest.7"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___maintest.x4-1"/>
    <bpm:has_context rdf:resource="#CONTEXT___maintest.x72-1"/>
    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
</bpm:ActivityNode>

<bpm:SimpleMerge rdf:ID="XOR___maintest.7">
    <rdfs:label>XOR</rdfs:label>
    <bpm:graphPart rdf:resource="#2_maintest"/>
    <!-- to: XOR -->
    <bpm:has_after_XOR rdf:resource="#XOR___maintest.61"/>
    <bpm:flow_all rdf:resource="#XOR___maintest.61"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___maintest.x72-1"/>
    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
</bpm:SimpleMerge>

<bpm:SimpleMerge rdf:ID="XOR___maintest.61">
    <rdfs:label>XOR</rdfs:label>
    <bpm:graphPart rdf:resource="#2_maintest"/>
    <!-- to: F9 -->
    <bpm:has_after_function rdf:resource="#F9___maintest.74"/>
    <bpm:flow_all rdf:resource="#F9___maintest.74"/>
    <bpm:precedes_all rdf:resource="#F9___maintest.74"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
</bpm:SimpleMerge>

<bpm:ActivityNode rdf:ID="F9___maintest.74">
    <rdfs:label>F9</rdfs:label>
    <bpm:graphPart rdf:resource="#2_maintest"/>
    <bpm:equivalent_to rdf:resource="&pt;process_9-F9"/>
    <!-- to: XOR -->
    <bpm:has_after_XOR rdf:resource="#XOR___maintest.11"/>
    <bpm:flow_all rdf:resource="#XOR___maintest.11"/>
    <bpm:precedes_all rdf:resource="#XOR___maintest.11"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
</bpm:ActivityNode>

<bpm:ExclusiveChoice rdf:ID="XOR___maintest.11">
    <rdfs:label>XOR</rdfs:label>
    <bpm:graphPart rdf:resource="#2_maintest"/>
    <!-- to: E5 -->
    <bpm:has_after_event rdf:resource="#E5___maintest.12"/>
    <bpm:precedes_all rdf:resource="#E5___maintest.12"/>
    <!-- to: outside the loop -->
    <bpm:flow_all rdf:resource="#E3___maintest.13"/>
    <!-- to: E3 -->
    <bpm:has_after_event rdf:resource="#E3___maintest.13"/>
    <bpm:precedes_all rdf:resource="#E3___maintest.13"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
</bpm:ExclusiveChoice>

<bpm:IntermediateEvent rdf:ID="E5___maintest.12">
    <rdfs:label>E5</rdfs:label>
    <bpm:graphPart rdf:resource="#2_maintest"/>
    <bpm:equivalent_to rdf:resource="&bpm;event_OrderIsReceived"/>
    <!-- to: XOR -->
    <bpm:has_after_XOR rdf:resource="#XOR___maintest.10"/>
    <bpm:flow_all rdf:resource="#XOR___maintest.10"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
</bpm:IntermediateEvent>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_strict rdf:resource="#E5___maintest.12" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#E5___maintest.12">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<bpm:IntermediateEvent rdf:ID="E3___maintest.13">
    <rdfs:label>E3</rdfs:label>
    <bpm:graphPart rdf:resource="#2_maintest"/>
    <bpm:equivalent_to rdf:resource="&bpm;event_OrderIsRejected"/>
    <!-- to: XOR -->
    <bpm:has_after_XOR rdf:resource="#XOR___maintest.76"/>
    <bpm:flow_all rdf:resource="#XOR___maintest.76"/>
    <bpm:precedes_all rdf:resource="#XOR___maintest.76"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
</bpm:IntermediateEvent>

<bpm:ExclusiveChoice rdf:ID="XOR___maintest.76">
    <rdfs:label>XOR</rdfs:label>
    <bpm:graphPart rdf:resource="#2_maintest"/>
    <!-- to: AND -->
    <bpm:has_after_AND rdf:resource="#AND___maintest.9"/>
    <bpm:precedes_all rdf:resource="#AND___maintest.9"/>
    <!-- to: E4 -->
    <bpm:has_after_event rdf:resource="#E4___maintest.75"/>
    <bpm:precedes_all rdf:resource="#E4___maintest.75"/>
    <!-- to: outside the loop -->
    <bpm:flow_all rdf:resource="#AND___maintest.9"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
</bpm:ExclusiveChoice>

<bpm:IntermediateEvent rdf:ID="E4___maintest.75">
    <rdfs:label>E4</rdfs:label>
    <bpm:graphPart rdf:resource="#2_maintest"/>
    <bpm:equivalent_to rdf:resource="&bpm;event_OrderIsProcessed"/>
    <!-- to: XOR -->
    <bpm:has_after_XOR rdf:resource="#XOR___maintest.77"/>
    <bpm:flow_all rdf:resource="#XOR___maintest.77"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
</bpm:IntermediateEvent>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_strict rdf:resource="#E4___maintest.75" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#E4___maintest.75">
```

```
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_all_strict rdf:resource="#XOR___maintest.76" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#XOR___maintest.76">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_all_strict rdf:resource="#E3___maintest.13" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#E3___maintest.13">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_all_strict rdf:resource="#XOR___maintest.11" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#XOR___maintest.11">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_all_strict rdf:resource="#F9___maintest.74" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#F9___maintest.74">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<!-- Bypass control flow block -->
<owl:NamedIndividual rdf:about="#XOR___maintest.72">
    <bpm:flow_all rdf:resource="#XOR___maintest.61" />
    <bpm:precedes_all rdf:resource="#XOR___maintest.61" />
</owl:NamedIndividual>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_all_strict rdf:resource="#XOR___maintest.61" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#XOR___maintest.61">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<!-- Bypass control flow block -->
<owl:NamedIndividual rdf:about="#XOR___maintest.4">
    <bpm:flow_all rdf:resource="#XOR___maintest.7" />
    <bpm:precedes_all rdf:resource="#XOR___maintest.7" />
</owl:NamedIndividual>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_strict rdf:resource="#XOR___maintest.7" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#XOR___maintest.7">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_strict rdf:resource="#F8___maintest.5" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#F8___maintest.5">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<bpm:ContextNode rdf:ID="CONTEXT___maintest.x4-2"></bpm:ContextNode>

<bpm:ActivityNode rdf:ID="F10___maintest.6">
    <rdfs:label>F10</rdfs:label>
    <bpm:graphPart rdf:resource="#2_maintest"/>
    <bpm:equivalent_to rdf:resource="&pt;process_10-F10"/>
    <!-- to: XOR -->
    <bpm:has_after_XOR rdf:resource="#XOR___maintest.7"/>
    <bpm:flow_all rdf:resource="#XOR___maintest.7"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___maintest.x4-2"/>
    <bpm:has_context rdf:resource="#CONTEXT___maintest.x72-1"/>
    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
</bpm:ActivityNode>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_strict rdf:resource="#F10___maintest.6" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#F10___maintest.6">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_strict rdf:resource="#XOR___maintest.4" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#XOR___maintest.4">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<bpm:ContextNode rdf:ID="CONTEXT___maintest.x72-2"></bpm:ContextNode>

<bpm:ActivityNode rdf:ID="F11___maintest.60">
    <rdfs:label>F11</rdfs:label>
    <bpm:graphPart rdf:resource="#2_maintest"/>
    <bpm:equivalent_to rdf:resource="&pt;process_11-F11"/>
    <!-- to: XOR -->
    <bpm:has_after_XOR rdf:resource="#XOR___maintest.61"/>
    <bpm:flow_all rdf:resource="#XOR___maintest.61"/>
    <!-- context -->
    <bpm:has_context rdf:resource="#CONTEXT___maintest.x72-2"/>
    <bpm:has_context rdf:resource="#CONTEXT___maintest.a8-3"/>
</bpm:ActivityNode>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_strict rdf:resource="#F11___maintest.60" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#F11___maintest.60">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_all_strict rdf:resource="#XOR___maintest.72" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#XOR___maintest.72">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
```

```
    <bpm:flow_all_strict rdf:resource="#F7___maintest.21" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#F7___maintest.21">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_all_strict rdf:resource="#XOR___maintest.10" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#XOR___maintest.10">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_all_strict rdf:resource="#F6___maintest.16" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#F6___maintest.16">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

<!-- From the last element before current top-loop to an element inside  -->
<owl:NamedIndividual rdf:about="#AND___maintest.8">
    <bpm:flow_all_strict rdf:resource="#XOR___maintest.77" />
</owl:NamedIndividual>

<!-- From an element inside current top-loop to the first element outside  -->
<owl:NamedIndividual rdf:about="#XOR___maintest.77">
    <bpm:flow_all_strict rdf:resource="#AND___maintest.9" />
</owl:NamedIndividual>

</rdf:RDF>

<!-- Generated with SemQuu (transformation mode: advanced) -->
```