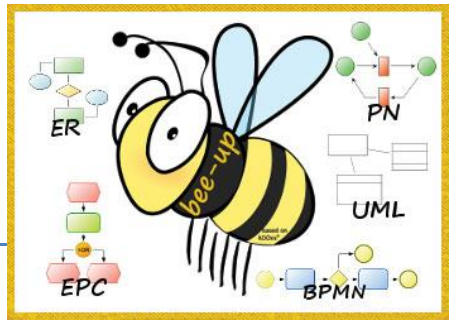
www.OMLAB®.org

Bee-Up
Handbook v1.2



Published: 2017-02-09

Authors: Patrik Burzynski, Dimitris Karagiannis

General information

This Handbook is written for Bee-Up version 1.2 based on the ADOxx 1.5[1] platform. The Bee-Up tool enables modelling according to the following languages and techniques:

- Business Process Model and Notation 2.0 (BPMN),
- Event-driven Process Chains (EPC) and extended EPCs (eEPC),
- Entity-Relationship (ER), and
- Unified Modeling Language 2.0 (UML)
- Petri Nets (PN).

If you should encounter problems, have questions or feature requests which are not covered yet, you can also contact us directly:

- Patrik Burzynski (patrik.burzynski@univie.ac.at)
- Prof. Dimitris Karagiannis

Installation

The Bee-Up tool requires a Windows operating system (XP, Vista, 7, 8, 8.1 or 10). To install it on a different operating system please use virtualization software (e.g. VirtualBox or VMware)[2] and a windows license[3].

To install the Bee-Up tool follow these steps:

1. Download the ZIP-File containing the installation package from OMiLAB.
2. Extract the contents to a folder.
3. Run the setup.exe from the extracted folder.

The setup first informs about prerequisites that will be installed automatically. This includes required frameworks (e.g. .NET) and the creation of a SQL Server instance where necessary. Once those tasks are finished a wizard will guide you through the remainder of the installation.

Note that if the setup automatically created a SQL Server instance, it is called ADOXX15EN and has set the initial 'sa' password to '12+*ADOxx*+34' (without the ' '). If you want to use an already available SQL Server database instance, it has to use "Mixed mode" for authentication. Should you no longer remember the 'sa' password: help on how to reset the 'sa' password can be found at the ADOxx.org community.

Some additional functionality provided by Bee-Up (beyond simple modelling, e.g. RDF Export) also requires a functioning Java 1.8 installation. A download link and installation instructions can be found at https://java.com.

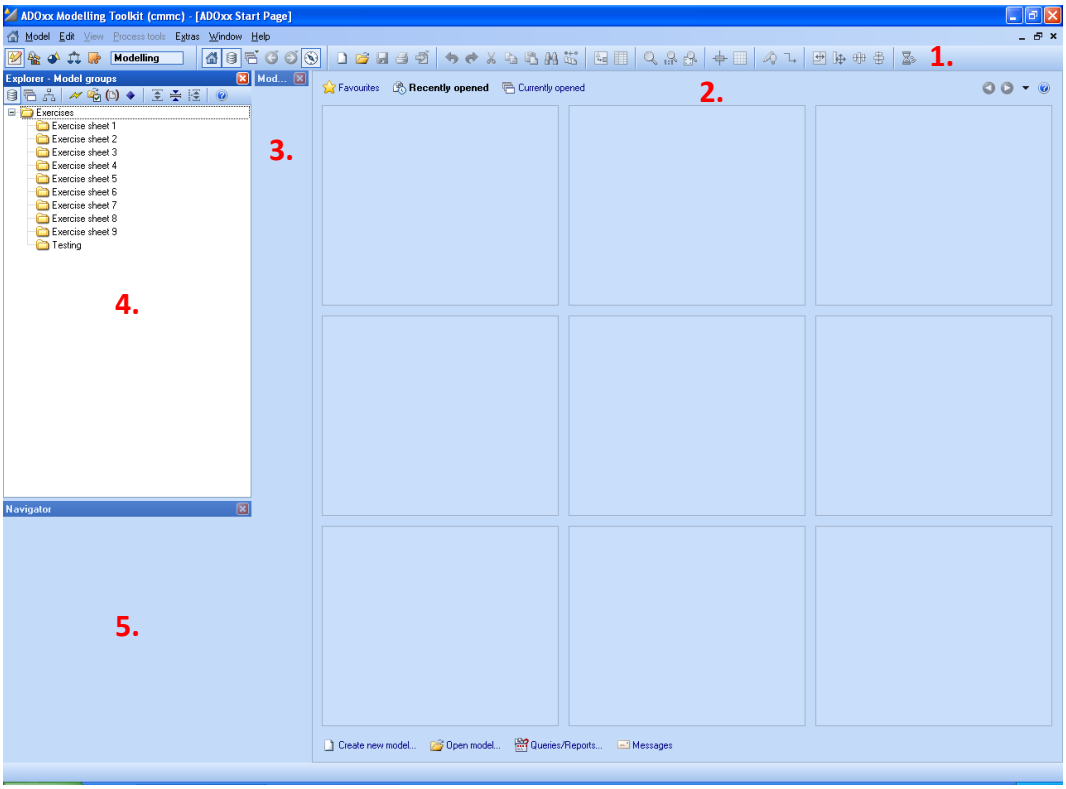
---

[1] http://www.adoxx.org/
[2] Obtainable from https://www.virtualbox.org/ or http://www.vmware.com/ respectively
[3] As a student of computer science on the University of Vienna you can get access to different versions of windows at http://cs.univie.ac.at/students/info/software/msdn/

Modelling with the Bee-Up tool

Tool overview

When first starting the Bee-Up tool you see a screen similar to the following one (without the numbers):



At the top is the menu bar with different menu items for direct access to some of the platforms functionality. The numbered elements of the above picture are:

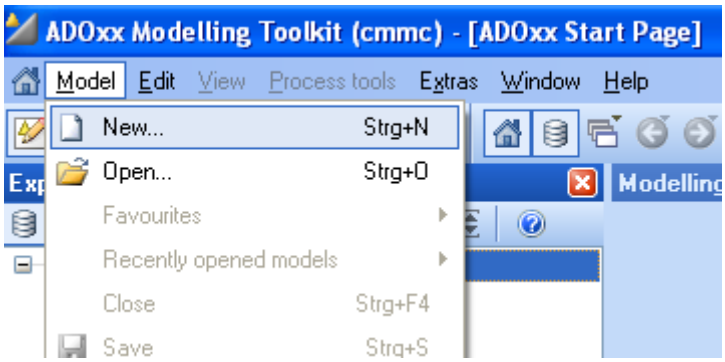
1. The **Toolbar** with icons as shortcuts for different functions. On the left side of the toolbar is the component selection:

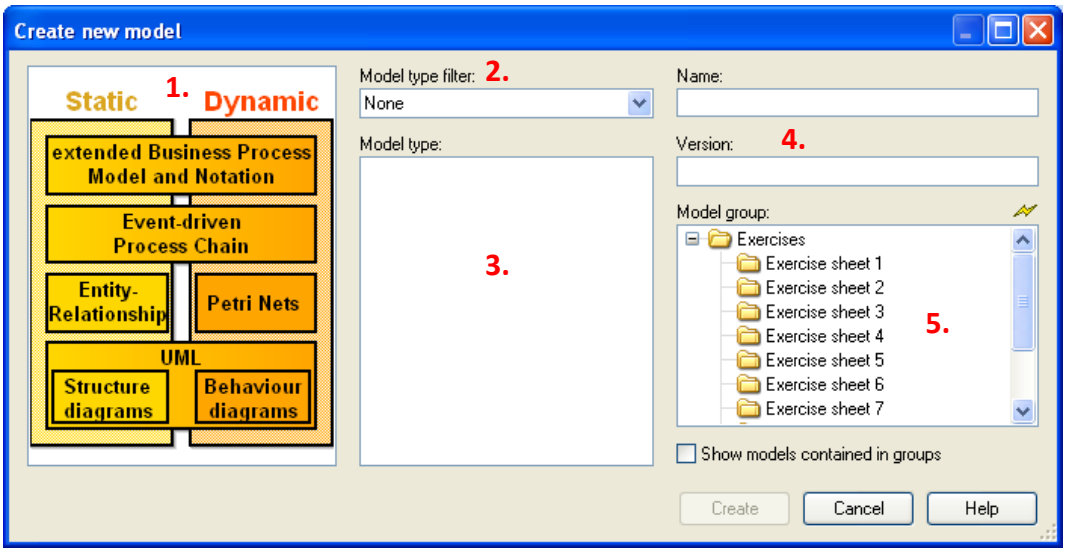
   This changes which menus, menu items and toolbar icons are available. The two important components are "**Modelling**" (left most icon) and "**Import/Export**" (right most icon). The current section of this document focuses on the "Modelling" component, while the next will describe some functionalities of the "Import/Export" component.
2. The **Start Page** is visible, showing recently opened models. The **Start Page** can be accessed through the house icon in the **Toolbar**. Once a model is opened it will be shown instead of the **Start Page**. This area is then referred to as the **Modelling area**.
3. The **Modelling window** shows the modelling objects and relations available for the currently opened model (in the figure above none, because no model is open).
4. The **Explorer window** shows all folders (called model groups) and the models, stored in a model group. Initially, model groups for all exercise sheets are preconfigured, accompanied by a testing model group.
5. The **Navigator window** shows an overview of the currently opened model.

Creating a new model

In order to create a new model select the menu item "Model → New…" while in the "Modelling" component (left most icon in the **Toolbar**).



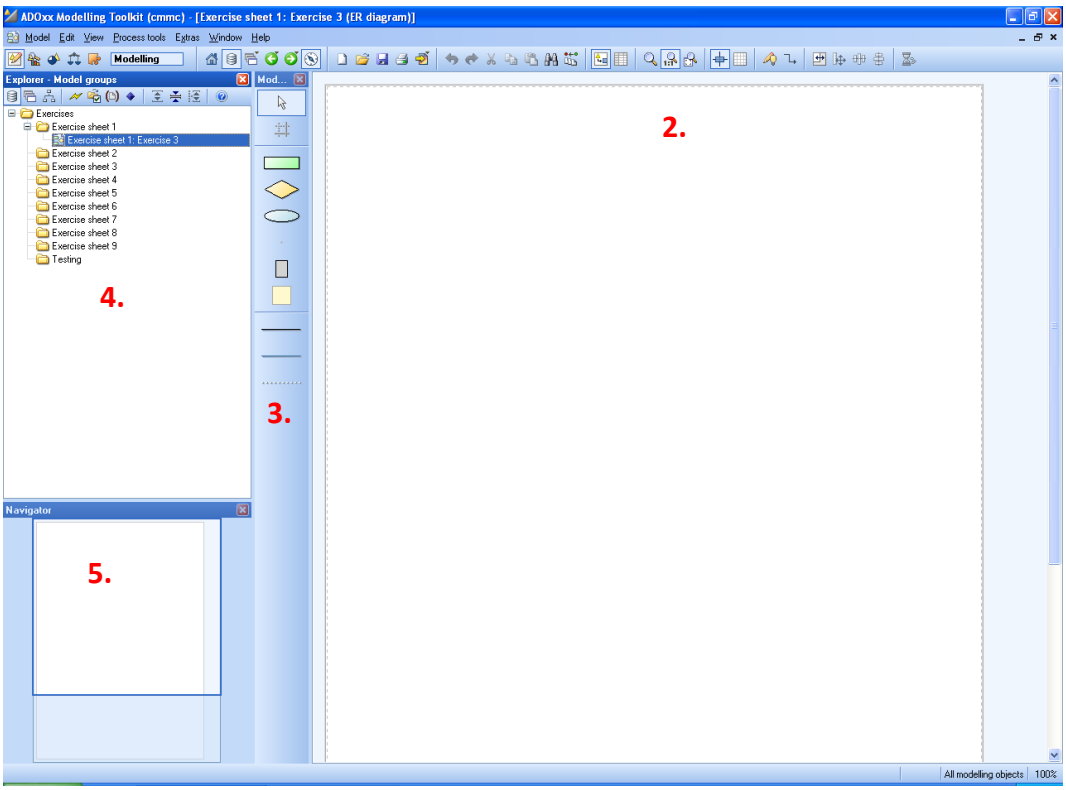
This will open the dialog shown below (without the numbers):



In this dialog first select the appropriate filter for the model types (e.g. Entity-Relationship for ER-models), either by using the graphic on the left side (**1.**) or the dropdown-list in the middle (**2.**). Afterwards, select the desired model type from the list in the middle (**3.**). Then enter a name (mandatory) and a version (optional) on the right side (**4.**). Finally, select to which exercise sheet (model group) the model should be assigned[4] (**5.**) and click on "Create". Please use the appropriate model group for your case (i.e. if it is a solution for a specific assignment, use the corresponding model group).

This will create an empty model of the selected modelling technique, store it in the selected model group and open it, ready to be edited. The name of the model will also automatically be prefixed with the selected model group (i.e. "Exercise sheet 1: " when the model is stored in that group)

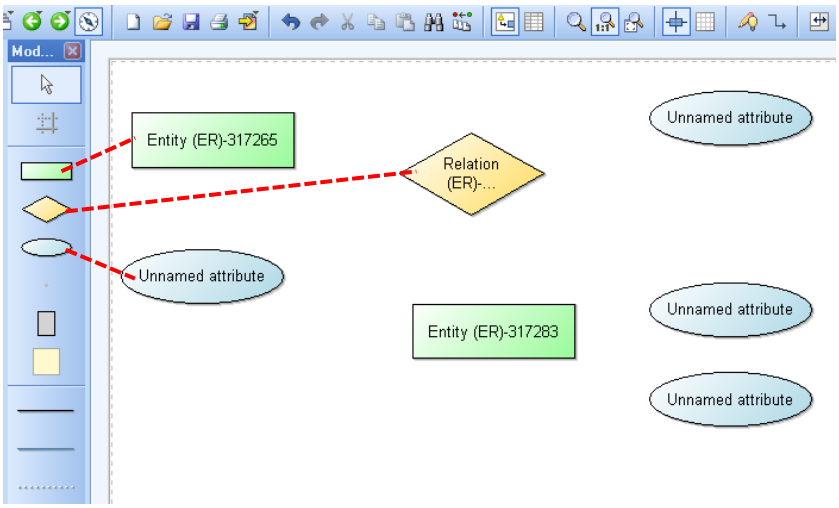
[4] Or select „Testing" (model group) when you are creating a model for testing purposes (e.g. to see how everything works, play around, explore the tool etc.).



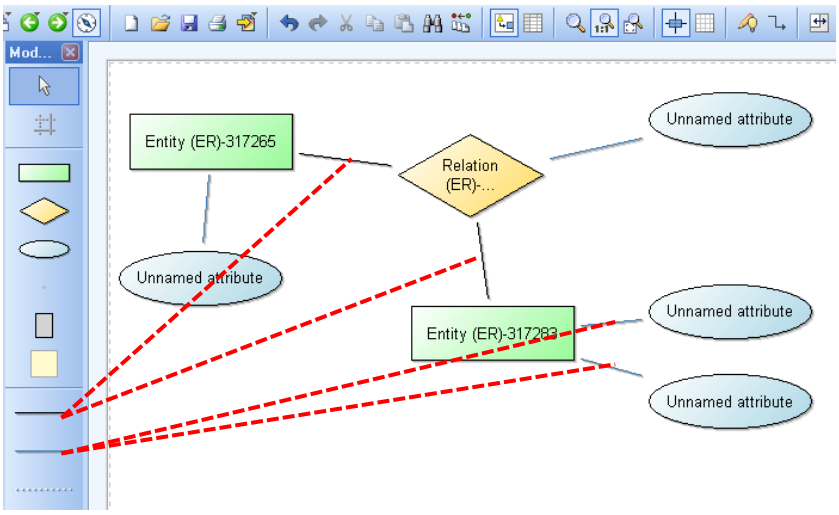
The picture above shows an example for a newly created and opened Entity-Relationship model. The **Modelling area** (2.) now shows the empty **Drawing area** (a white canvas to drag & drop objects and relations) instead of the **Start Page**. The **Modelling window** (3.) shows the available types of objects and relations that can be used for ER modelling, e.g., *Entity*, *Relationship*, *Attribute*. The created model can also be seen in the **Explorer window** (4.) under the selected exercise sheet (model group). The **Navigator window** (4.) shows the complete model, enabling direct navigation and zooming, as well as the portion that is currently shown in the **Modelling area**.

Editing the model

To edit a model it hast to be opened in the Bee-Up tool. The easiest way to open a model is to double click on its name in the **Explorer window**. New objects can be added to the model by selecting the type of object that should be added in the **Modelling window** and then clicking in the **Modelling area** where the object should be placed. If necessary the **Drawing area** will be extended automatically. After adding a few objects, the **Modelling area** could look like this:



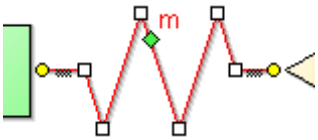
In order to connect objects with relations, first select the relation type from the **Modelling window**. Then click in the **Modelling area** on the object that is the **source** ("starting point") of the relation and then on the object that is the **target** ("ending point") of the relation. Certain types of relations can only be used between specific types of objects (e.g. "has Attribute" always has to target an "Attribute"). The previous example with some relations could look like this:



All objects can be moved in the **Modelling area** by selecting them and moving them accordingly. Some objects can also be resized, which works similar to resizing windows in the operating system (drag the side/corner when it is selected). For both the "Edit" function has to be selected in the **Modelling window** (looks like the default mouse cursor). After creating objects and relations you can quickly switch back to "Edit" by pressing the **right mouse button**. It is also possible to move/resize several objects at once by selecting them first (draw selection box around them, SHIFT+Click, CTRL+Click) and then performing the move or resizing. The difference between SHIFT+Click and CTRL+Click is that using SHIFT will select the object and all of the objects it contains if it is a container (like a "Package", "State", "Combined Fragment", "Lifeline" etc.) while using CTRL will not. This is useful when a container should be moved with all of its contents at once.

Adding edges to relations

It is also possible to add **bend points** to relations. Those force the relation to be drawn through that point and can increase readability of the created models. The following image shows a relation with six bend points (small white rectangles):

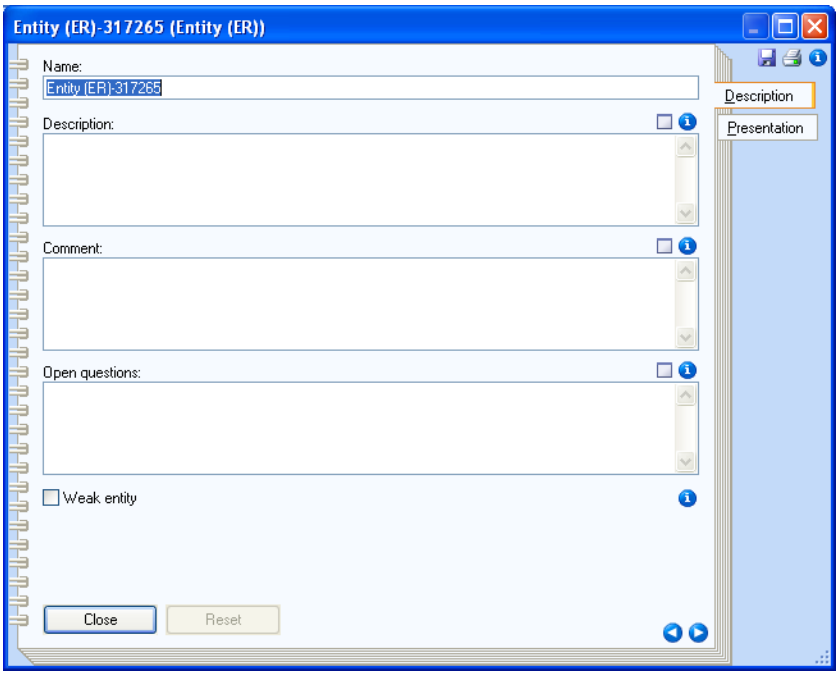


Bend points can be added either during the creation of the relation, or afterwards. To add **bend points** during the creation first click on the source, then click on the desired points in the **Modelling area** where a bend point (i.e. an edge) should be drawn, and finally click on the target object. To add them after the relation has been created, select the relation first and then click and drag a point of the relation (that isn't already a bend point) to the desired place on the **Modelling area**.

The source or the target of a relation can also be changed by selecting the relation and then clicking the yellow circle at the beginning or the end and dragging it to the new object that should be the source or target. The **green diamond** that is visible when a relation is selected can be used in many cases to move the text that is visualized next to it (e.g. the cardinality of the "Links" relation; 'm' in the figure above).

Editing attributes

All objects, models and relations can have editable attributes, which can also influence their visualization in the **Modelling area** (e.g. weak entities have a double outline instead of a single one). Those attributes can be accessed by double-clicking on the object or relation (or selecting it and then pressing Enter). This opens up the ADOxx **Notebook**, which contains the attributes that can be edited. To access the attributes of a model it has to be opened first and then select the menu item "Model → Model attributes" or press ALT+Enter. The following picture shows an example of a **Notebook**:



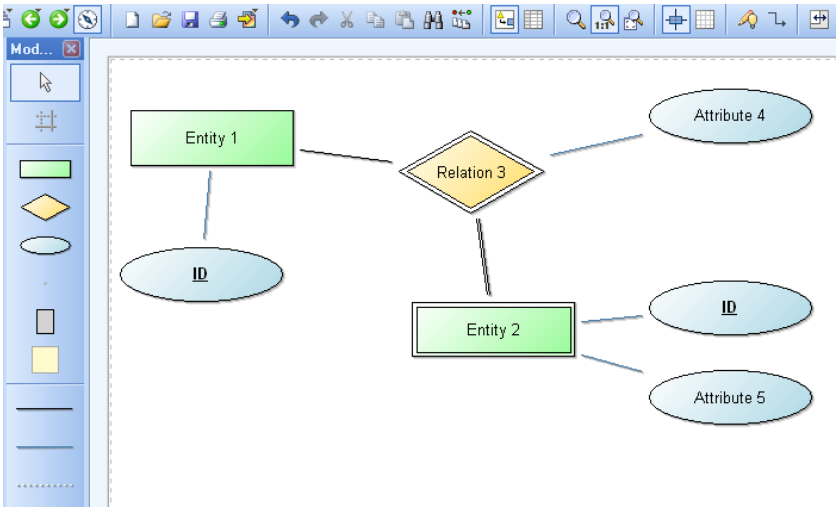
The attributes take up the largest portion of the **Notebook** and are categorized in different tabs, available on the right side of the **Notebook**. Depending on the attribute type different editors for the attribute value are available (e.g. single-line text field for the Name, multi-line text area for the Description, checkbox for the Weak entity etc.). For some attributes an alternative editor can be accessed through the button. Also additional information is available for most of the attributes and can be accessed by clicking on the icon. A similar icon can be found at the top right (underneath the X for closing the window), which provides information about the type of the object. The two buttons at the lower right are an alternative to switching between the different categories. They are also used to switch between pages of one category, in the case that more attributes are available than can be shown in the **Notebook** window.

There are two special types of attributes that have to be described in more detail:

1. Inter-model references – they allow to link (reference) to one or several other objects or models and have three special icons: . The first one (+) allows to add new references, while the second one (X) removes the selected references. The third one (→) is like a hyperlink that jumps to the referenced object. Often when the attribute value is visualized it also works as a hyperlink in the Modelling area.
2. Tables – they allow to store more complex attribute values in a structured way. They also have two special icons: . The first one (+) is used to add a new row at the end and the second one (X) removes the selected rows. Note that in order to select rows the number on the left side has to be clicked. The context menu also provides several options to handle rows in a table (e.g. insert row, move row, etc.)

It is also possible to edit some of the attribute values that are visualized in the **Modelling area** without using the **Notebook**. For this simply select the object and then click on the visualized attribute value (e.g. the name: "Entity (ER)-317265"). Note that this prevents opening the **Notebook** although the attribute is being edited.

The previous example with edited attributes (mostly names, but also some other ones) looks like this:



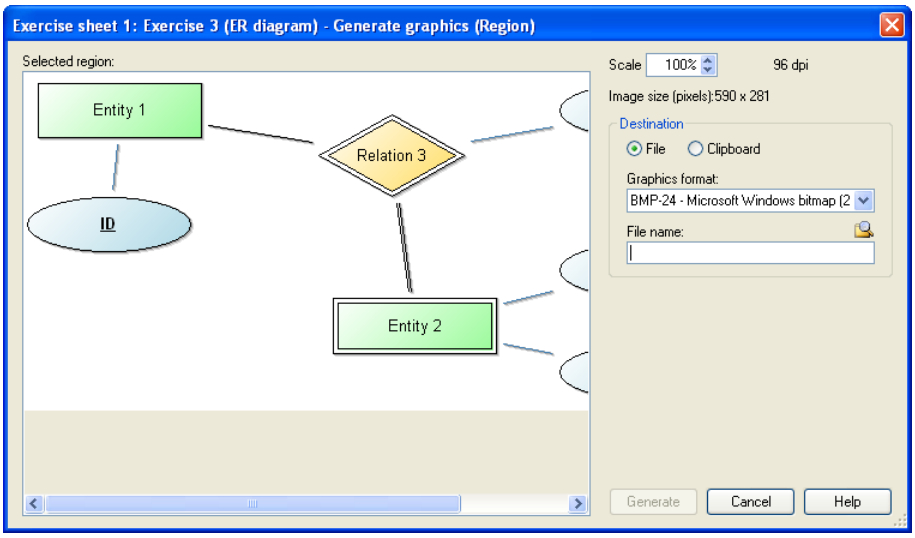
Exporting models

The tool also provides functionality to export the created models in different formats. Some of those will be described here. Most of them are available in the "Import/Export" component ( )

Creating a graphic from a model

It is possible to create a graphic of the currently opened model and store it in a file using the provided "**Generate graphics**" functionality (icon in the **Toolbar** available when using the "Modelling" component). This opens a dialog showing the region of the model for which the graphic will be created as well as options to scale the created picture and to either save it in a file (with different formats available) or copy it to the clipboard. Clicking on the "**Generate**" button will finish the process.

The region can be set beforehand by holding down the ALT key and click-and-dragging a box in the **Modelling area** with the mouse. This will create a teal rectangle (that can also be resized) showing what region will be used for generating the graphic. The following picture shows the dialog for the previous example model, where a fitting region has been set:



Exporting the exercise

It is also possible to export an entire exercise at once. This can be achieved by using the menu item "Model → Export Exercises" (or using the icon in the **Toolbar**). This will show a dialog where the model group containing all of the solutions for the exercise can be selected. Afterwards a new dialog asks for a folder where the results should be stored. Once it is finished a message will inform you about it.

This functionality exports all of the models contained in the selected model group or one of its descendent sub-groups in ADL format and also creates individual graphic files for all of the models. The creation of the graphic can sometimes fail when the name and/or version contain a character that is not supported by the operating system as a filename[5]. **IMPORTANT**: It also removes empty space from the right and the bottom in the drawing area (in the **Modelling area**) AND saves the model before generating the graphic.

Exporting and importing models

One or several models as well as model groups can be exported to/imported from either the ADL format (proprietary) or XML format by using the according menu items in the "Model" menu (e.g. "Model → XML export (default)…").

For the export a simple dialog is show where the models and/or model groups are selected at the top. The middle of the dialog contains some checkboxes to control what should be exported (e.g. "Including models", "Including model groups" etc.) and at the bottom the file is specified where the models/model groups should be exported to. The export is started by clicking on the "Export" button and at the end a success or error message is shown.

[5] Common ones like „:", „/", „*" etc. are replaced by a „-" for the file name.

For the import there are several tabs available. In the “File selection” tab the file containing the models/model groups is selected. The “Model options” tab provides some choices on how to deal with collisions (e.g. what to do when two models have the same name). The last tab “Log file” allows logging the process in a file. After everything is selected click on the “OK” button. This will prepare the data from the previously selected file and open another dialog. Here the left side shows which models and/or model groups have been found in the file and you can select which of those should be imported. On the right side select into which model group the contents should be imported and click on the “Import” button. At the end a success or error message is shown.

Exporting models as RDF

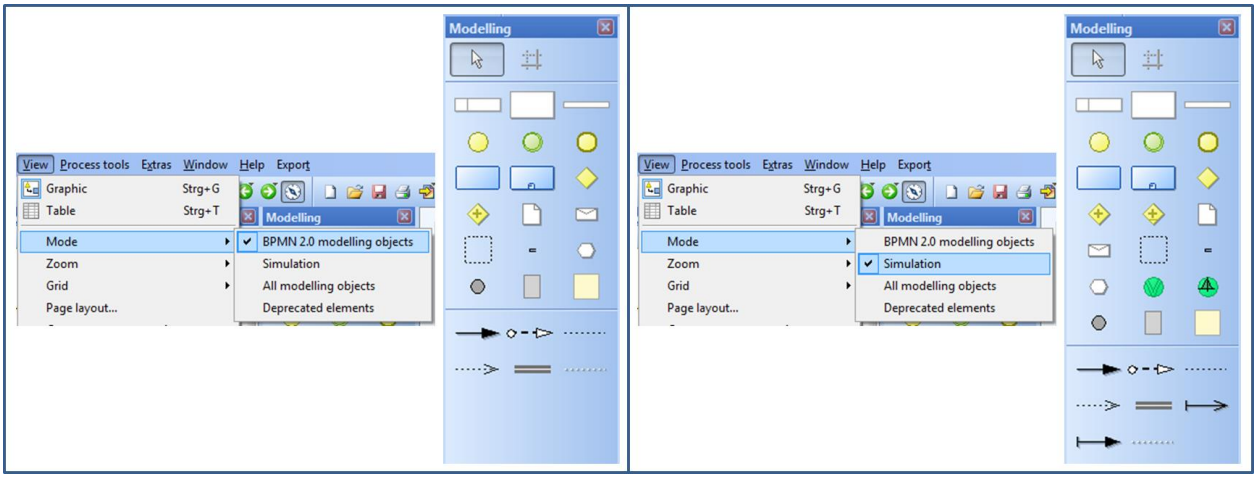
A new function in Version 1.1 adds the possibility to export one or several models in RDF Format. This can be simply accessed through the menu item “Model → RDF Export”. The first dialog asks which models should be exported. Here either directly the models or entire model groups can be selected. Afterwards a file selection dialog will ask where the RDF data should be stored and allows a choice of different formats (.trig is recommended). A third dialog will ask for a base URI to be used in the identifiers as a prefix. Here it is recommended to use a valid URL without the fragment (for example http://www.omilab.org/example# or http://www.example.net/#). It is not necessary for the URL to be actually used (i.e. the URL can return an error code like 404), just that the URL is valid. Once it is finished a message will inform you about it.

With Version 1.2 additional attributes have been added to (almost) all elements and models to enhance the RDF Export. These are found in the “RDF properties” tab of the **Notebook**. The “URI” attribute allows specifying a specific URI to be used for the element/model instead of automatically generating a URI. The “Additional Triples” table allows specifying additional triples (Subject, Predicate and Object) that will be added to the graph, where one row represents one triple. If a cell is left empty in the “Additional Triples” table, then it will be substituted with the URI of the element/model it is located in. Note that the values provided in those attributes will be treated as is as a complete URI (ignoring prefixes etc.). Therefore, it is necessary to enter the entire URI. Also with Version 1.2 the names of elements and models are exported explicitly as rdfs:label statements.

Additional hints and information

Specific information for BPMN modelling

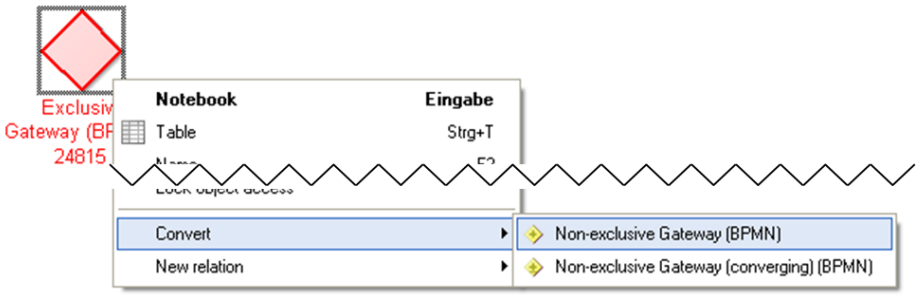
The BPMN implementation provides concepts to describe processes, as well as for describing input, output and execution of "Tasks". Two different modes are available, which limit the available concepts. By default the "**BPMN 2.0**" mode is selected, which contains the typical BPMN concepts. However, the mode can be changed (through the menu "View → Mode") to "**Simulation**". This mode further adds concepts which are necessary to perform simulation of processes in the tool (e.g. converging gateways as their own types). The following picture shows the two modes and which types of elements they use:



The majority of BPMN should be straight forward and some constraints are enforced by the tool (e.g. "Start Events" cannot have any incoming "Subsequent" relations). However, due to certain platform restrictions the Gateways (Exclusive, Inclusive, Parallel etc.) are handled a bit differently[6]. In the standard BPMN mode the Exclusive Gateway is available as its own type ("Exclusive Gateway"), however the Inclusive and Parallel Gateway are modelled through the "Non-exclusive Gateway". The type (Inclusive, Parallel or Complex) is then set through the attribute "Gateway type" (in the Notebook)[7].

Previously the Intermediate event was split into two different types: "Intermediate Event (boundary)" and "Intermediate Event (sequence)". Since Version 1.1 the two have been merged into one and are distinguished through setting the "Attached to" Attribute. If the attribute has a value it will be considered on the Boundary of the set "Task". A new Mode has been added called "Deprecated", which allows the use of the two old Intermediate events in order to not destroy previously created models. Those events can easily be transformed into the new "Intermediate Event" by right clicking on them and selecting "Convert → Intermediate Event (BPMN)".

Certain types of objects can be converted into other types (e.g. "Exclusive Gateway" to "Non-exclusive Gateway") by selecting them and then using "Conversion" in the context menu. An object will become greyed out and cannot be selected, if it is converted to a type that is not available in the current mode. To transform it back (or delete it or change it etc.), simply change the mode to one that makes use of the type (e.g. "All modelling objects"). The picture below shows the available options for converting the "Exclusive Gateway":



[6] This is due to the way simulation is handled by the platform.

[7] "Simulation" mode additionally has a "Non-exclusive Gateway (converging)", which is necessary for the simulation.

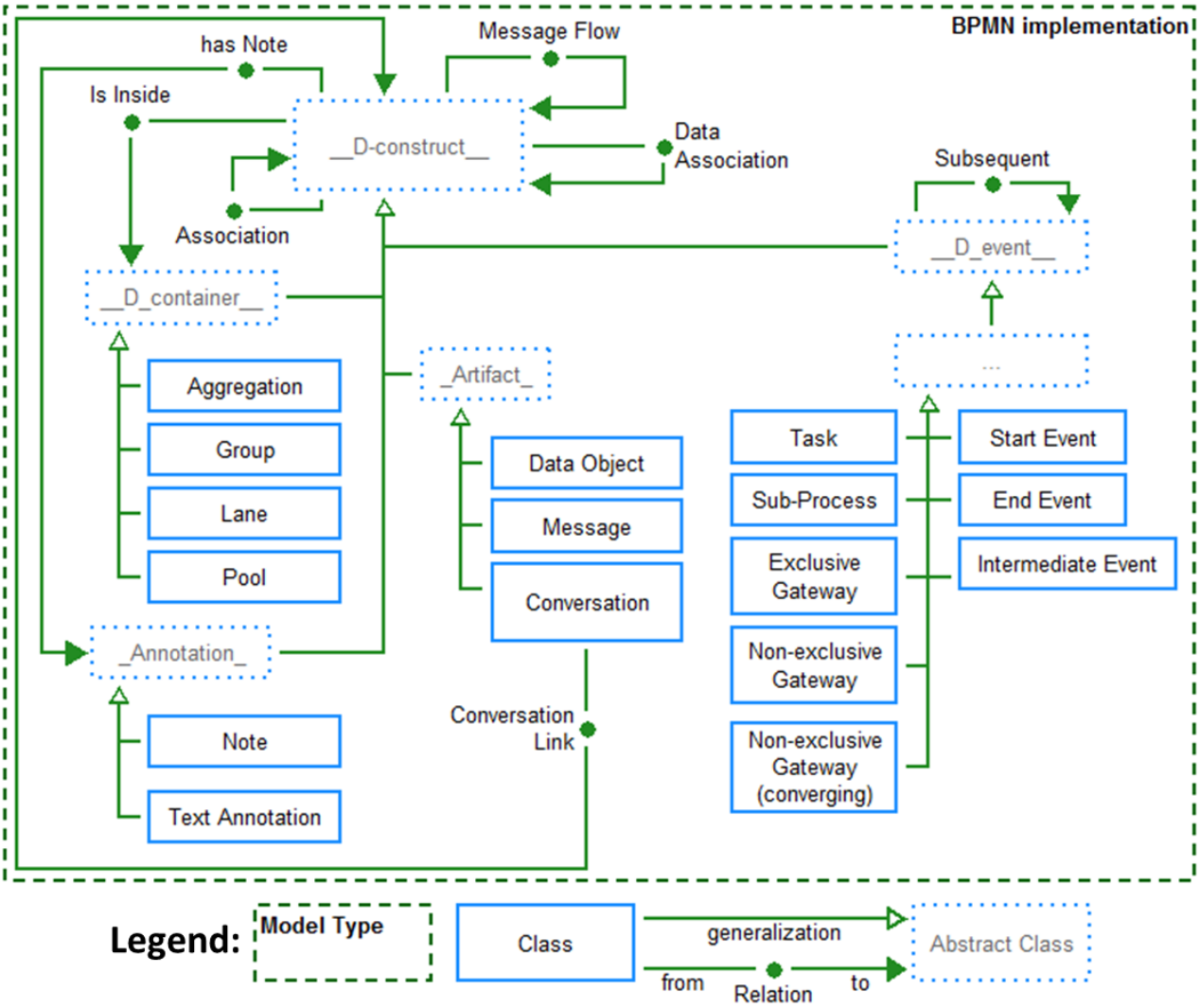
The availability of some attributes (in the Notebook) is dependent on the values of others. This is to prevent setting wrong values or changing irrelevant attributes. For example the available “Triggers” of a “Start Event” depend on its “Type” to prevent wrong selections. Another example is the “Loop condition (standard)” attribute of a “Task”, which is only available when the “Loop type” is set to “Standard” (otherwise it is irrelevant).

The relation “Subsequent” has an attribute “Visualized values”, which controls which attribute values are shown. Should the desired value not be shown on the drawing area (e.g. “Transition condition”) then it might be because of the “Visualized values” attribute. “Subsequent” is also used in several different model types (e.g. EPC, UML Activity Diagram). Therefore it also contains attributes used in those model types. They are however grouped in their own categories (e.g. “UML properties”).

For many different types of objects (e.g. “Start Event”, “Exclusive Gateway” etc.) the visualization of the name can be controlled through the attribute “Show name”. In some cases this is a simple choice if the name should be displayed (e.g. “Start Event”). In other cases more options are available (e.g. “Exclusive Gateway”).

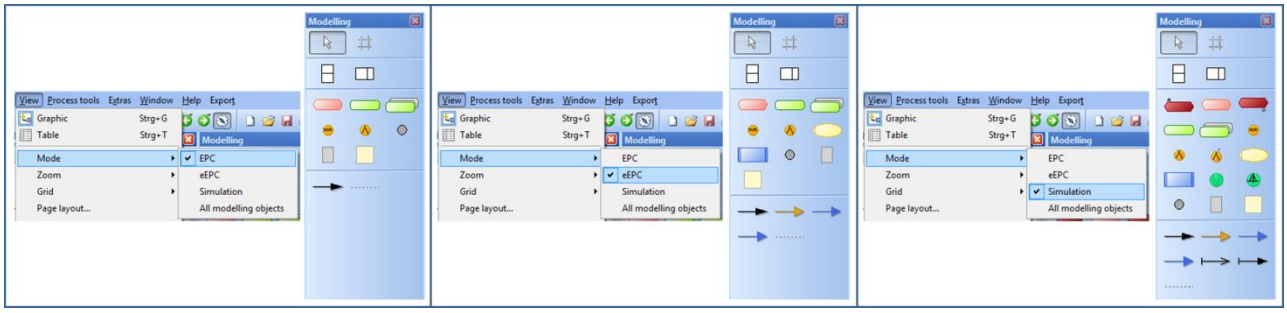
Version 1.2 also added the option to further describe Service Tasks through Petri Nets or Flowcharts using the “Automated service details” attribute. The attribute should reference the starting point in the Petri Net or Flowchart.

The following picture provides some detailed information about the implementation of BPMN in Bee-Up. More specifically it shows an excerpt of how the BPMN meta-model is implemented. Certain parts are provided by the platform to allow specific functionality, like __D_event__ and Subsequent used for process simulation. The “…” abstract class is used to represent complex generalization structures in the meta-model in a simplified manner.



Specific information for EPC modelling

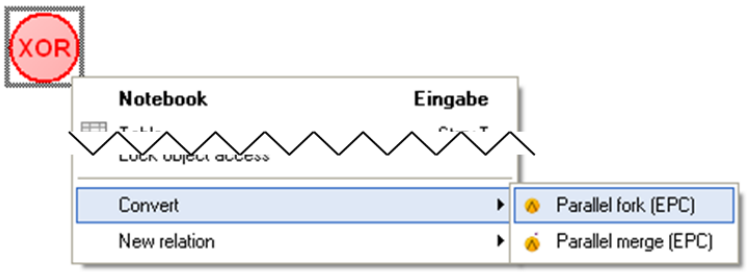
The EPC implementation provides the core concepts from Event-driven Process Chains to describe processes ("Event", "Function", logical operators), as well as some additional ones for describing input, output and execution of "Functions". Different modes can be selected, which limit the available concepts. By default the "**EPC**" mode is selected, which contains "Events", "Functions" and the basic logical operators from EPC (also some additional "general" concepts). However, the mode can be changed (through the menu "View → Mode") to "**eEPC**" or "**Simulation**". "eEPC" (extended EPC) additionally contains "Organizational units", "Information objects" and relations for those new object types. The relations for denoting inputs and outputs for "Functions" are implemented as separate types. "Simulation" mode further adds concepts which are necessary to perform simulation of processes in the tool (e.g. "Start Event" which explicitly denotes the start of the process). The following picture shows the three modes and which types of elements they use:



The majority of EPC should be straight forward and some of the constraints of an EPC model are enforced by the tool (e.g. between two "Functions" there has to be an "Event"). However, due to certain platform restrictions the typical logical operators (XOR, OR, AND) are handled a bit differently[8]. In the basic "EPC" and "eEPC" the XOR operator is available as its own type ("XOR operator"), however the AND and OR operators are modelled through the "Parallel fork". The type (AND or OR) is then set through the attribute "Type" (in the Notebook)[9]. In "EPC" and "eEPC" mode the "Parallel fork" is used both for splitting and merging paths.

The relation "Subsequent" has an attribute "Visualized values", which controls which attribute values are shown. Should the desired value not be shown on the drawing area (e.g. "Transition condition") then it might be because of the "Visualized values" attribute. "Subsequent" is also used in several different model types (e.g. BPMN, UML Activity Diagram). Therefore it also contains attributes used in those model types. They are however grouped in their own categories (e.g. "UML properties").

Certain types of objects can be converted into other types (e.g. "Event" to "Start Event" or "End Event", "XOR operator" to "Parallel fork" etc.) by selecting them and then using "Conversion" in the context menu. An object will become greyed out and cannot be selected, if it is converted to a type that is not available in the current mode. To transform it back (or delete it or change it etc.), simply change the mode to one that makes use of the type (e.g. "All modelling objects"). The picture below shows the available options for converting the "XOR operator":

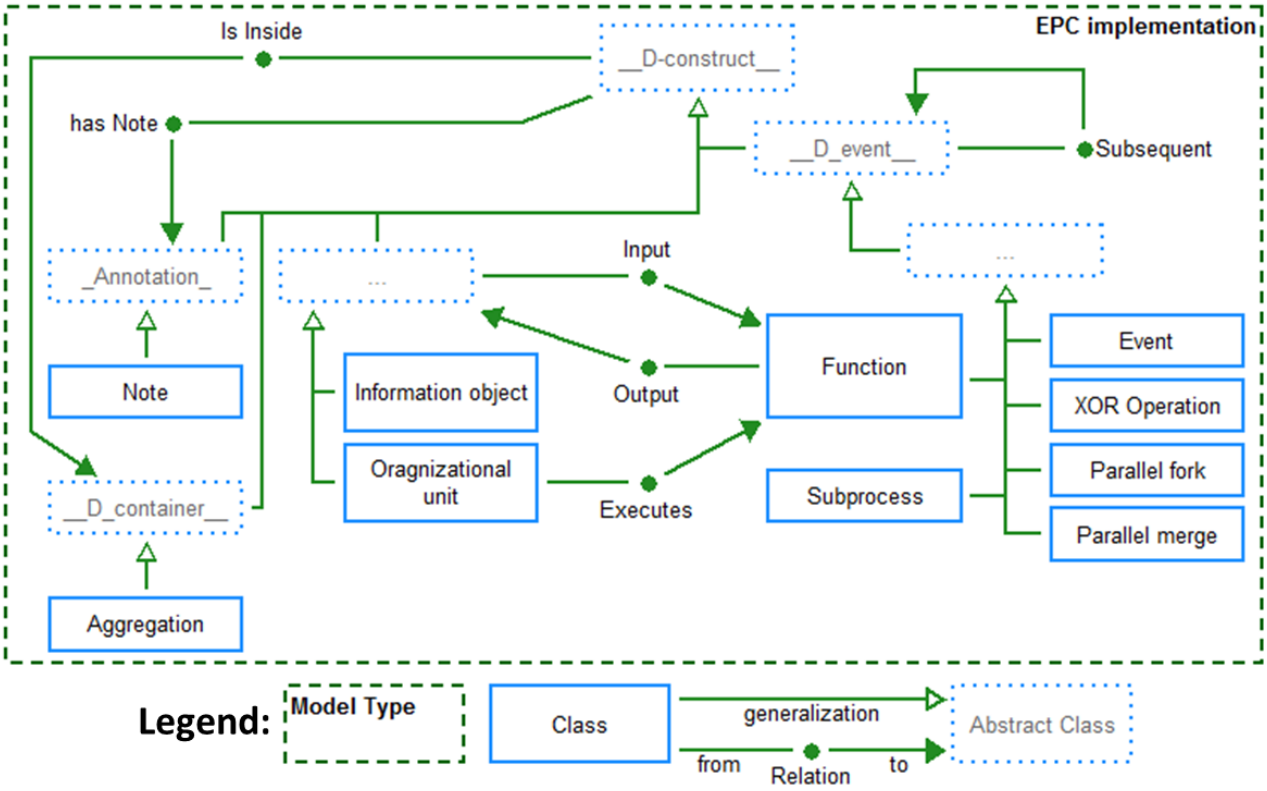


[8] This is due to the way simulation is handled by the platform.

[9] "Simulation" mode additionally has a "Parallel merge". This distinction between fork and merge is necessary for the simulation algorithm.

Version 1.2 also added the option to further describe Functions through Petri Nets or Flowcharts using the "Automation details" attribute. The attribute should reference the starting point in the Petri Net or Flowchart.

The following picture provides some detailed information about the implementation of EPC in Bee-Up. More specifically it shows an excerpt of how the EPC meta-model is implemented. Certain parts are provided by the platform to allow specific functionality, like __D_event__ and Subsequent used for process simulation. The "…" abstract class is used to represent complex generalization structures in the meta-model in a simplified manner.



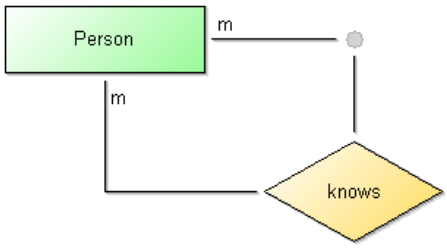
Specific information for ER modelling

The ER Model implementation provides the general concepts used ("Entity", "Relation" and "Attribute") as well as the necessary connectors[10] ("Links" and "has Attribute") among other common elements ("Note", "has Note" etc.). Restrictions are set for the "Links" connectors to prevent creating wrong models. A "Links" connector has to start from either a "Relation" or a "Relation Node" and target an "Entity", a "Relation" or a "Relation Node".

The finer details are controlled through the attributes found in the notebook, which in some cases also influence the visualization (notation) of the objects. For example to show a "Weak Entity" use a normal "Entity" and check its "Weak entity" attribute in the Notebook. Also to specify the "Relation" that indicates on which stronger entity it relies use a "Relation" and set its "Relation type" attribute to "Weak entity dependency".

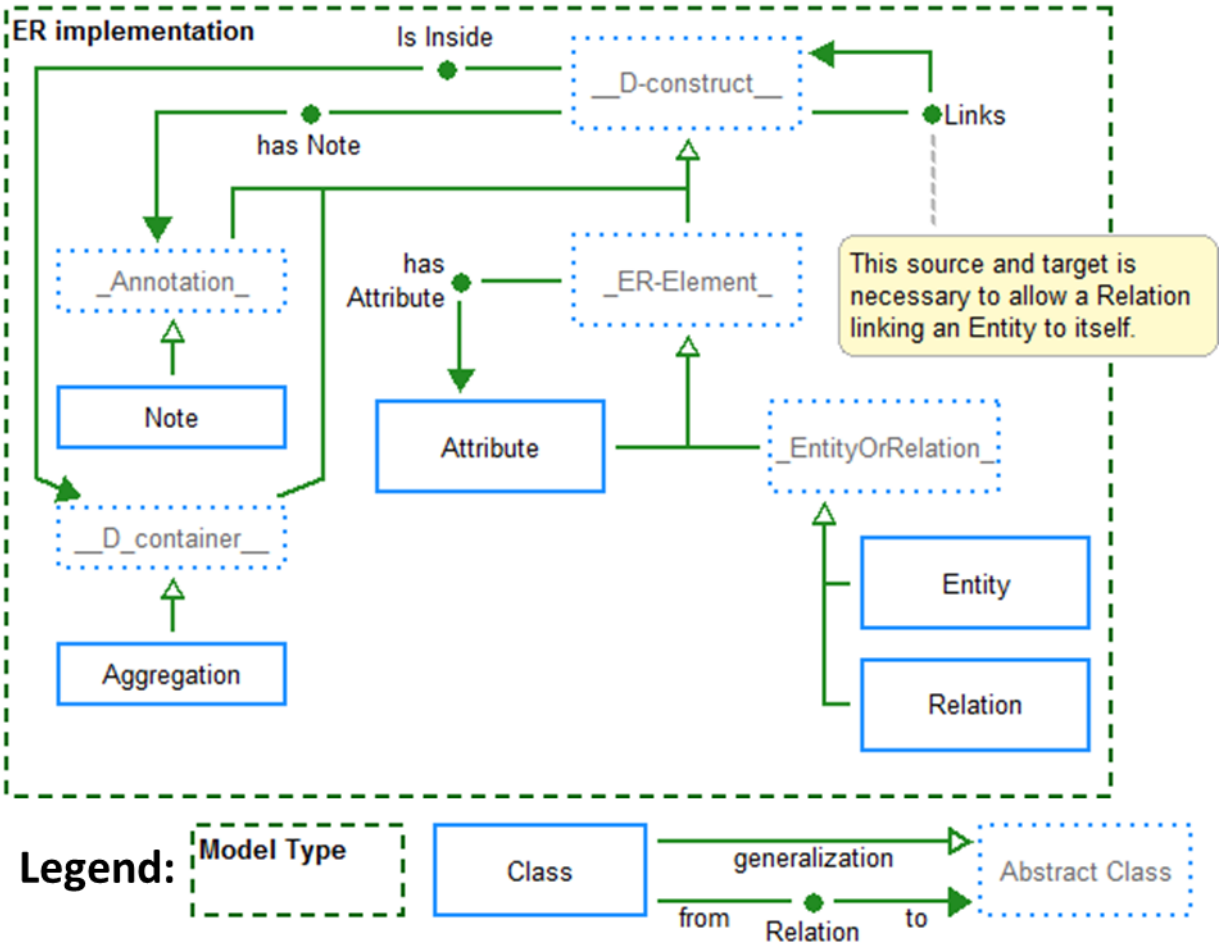
Should a "Relation" be between the same "Entity" (e.g. Person knows Person) then use the "Relation Node" on one of the connections. For a binary relation (e.g. Person knows Person): First connect the "Relation" to the "Entity" directly, then connect the "Relation" to a "Relation Node" and then connect the "Relation Node" to the "Entity". This is necessary because of how identifiers of connectors work (identified by their type, their source object and their target object). An example can be seen below:

[10] In this one section we refer to the lines as "connectors" instead of "relations" to not confuse them with the objects of type "Relation"



Functionality for creating SQL statements from an ER Model is also provided. It can be accessed through the "Model" menu of the "Import/Export" component. A description with the quirks and details can also be found in the same menu. The functionality uses the currently active model and will ask for a file to store the created SQL code in. If the selection of a file is cancelled it will instead show the SQL code in a pop-up window from where it can be copied to the clipboard. Version 1.2 added two SQL properties to "Attribute": 1) one for directly entering the data type of the attribute and 2) to specify auto-increment (only works for MySQL).

The following picture provides some detailed information about the implementation of ER in Bee-Up. More specifically it shows an excerpt of how the ER meta-model is implemented. Certain parts are provided by the platform to allow specific functionality, like __D_container__ used to automatically derive "Is Inside" relations.



Specific information for UML modelling

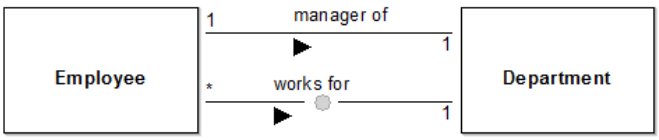
UML and its implementation in the tool are big. Addressing all of the peculiarities would be difficult and also a lot of text to read. Therefore, they are addressed in general and some examples are provided.

Notations[11] are generally influenced by the attribute values that are specified for them (in the notebook):

- Most of the attributes that deal only with the visualization are located in a category called "Presentation". Examples for such attributes are "Color" (of the object background), "Representation" (of text) and "Presentation" (of class details).
- The "Subsequent" relation and the "Activity edge" use the attribute "Visualized values" to control which attribute values should be shown (e.g. Denomination, Transition condition, Weighting etc.).
- Relations often have an option on where the text should be shown, handled through a "Representation" attribute. In general "above/below" value should be used for parts of relations going horizontally and "left/right" value for parts of relations going vertically. As an example the "Association" used in "Class / Object Diagrams" can have text in three parts: at the start, at the middle and at the end. For the start and the end a different "Representation" value can be set. If for example the association class starts going from the object towards the right (horizontal) and then turns towards the bottom (vertical) then the "Representation start" should use "above/below", the "Representation end" should use "left/right". In most cases the middle part uses a notation that works well with both horizontally and vertically drawn relations.
- UML Specific attributes (e.g. "IsAbstract", "Visibility" etc.) are usually located in a category called "UML properties". In some cases they are located in the "Description" category (e.g. the "Type" of a "Final Node") for quicker access or have their own category (e.g. "Properties/Operations" of a "Class"). Some of them also influence the notation, like the "Final type" attribute of a "Final Node" in an "Activity Diagram" or the "Properties" entered in a "Class".

Certain relations, like the "Message" from a "Sequence Diagram", have their sub-types controlled mostly through the attributes. So the typical types like "synchronous call", "asynchronous call" and "reply" are handled through the "Message sort" attribute of the "Message" relation.

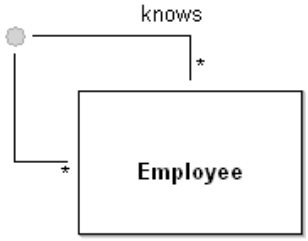
In order to draw several relations between the same two objects in the same direction (e.g. several "Associations" between the same two "Classes") the "Relation Node" has to be used. For every additional relation beyond the first one it is necessary to create two relations: one has to go from the source object to a "Relation Node" and the other from that "Relation Node" to the target object. This is necessary because of how identifiers of relations work[12]. For example when there are the classes "Employee" and "Department" and two associations "works for" and "manager of" between the two classes. The "manager of" association can go directly from "Employee" to "Department". However, the "works for" association has to be split in two: one association going from "Employee" to a "Relation Node" and another from the same "Relation Node" to the "Department". The attributes should also be split among those two relations accordingly (i.e. the multiplicity for the "Employee" side of "works for" has to go to the first relation, the multiplicity for the "Department" side of "works for" has to be in the second relation and the name can be in one of those). The example can be seen below:



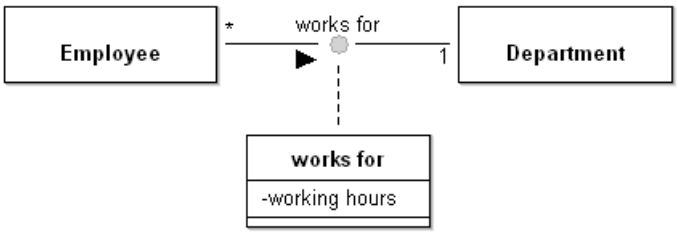
[11] The look of an object on screen or on paper.

[12] A relation is identified by its type, its source object and its target object. Duplicate identifiers are not allowed.

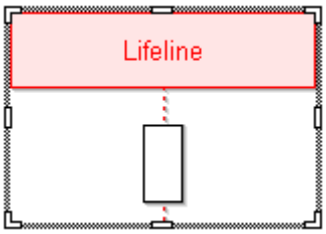
There are also cases where the source and the target of a relation should be the same object (e.g. an “Association” from a “Class” to the same “Class” or a “Transition” from a “State” to the same “State”). This also requires a “Relation Node”, since the same object cannot be the source and the target of one relation. For this case simply make a relation from the object to the “Relation Node” and then from the “Relation Node” to the same object. For example when a relation “knows” should be from and to the class “Employee” first create the “Relation Node”, then make an “Association” from “Employee” to the “Relation Node” and then from the “Relation Node” back to the “Employee”. The example can be seen below:



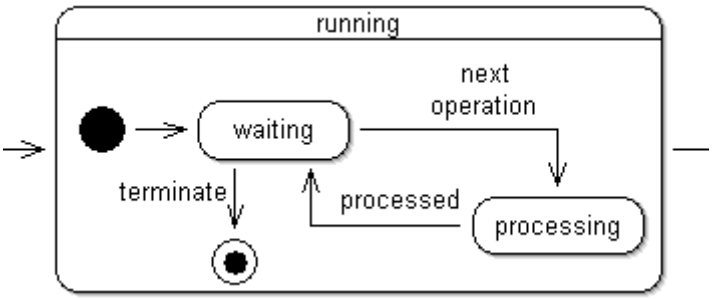
In UML it is also sometimes necessary to have a relation which originates or targets another relation. Again this is solved by using the “Relation Node”. Simply put the “Relation Node” on the relation that should be the source or the target (this will split the relation in two) and use the “Relation Node” as the source or target of the other relation. For example when the association “works for”, between “Employee” and “Department” should be linked to a class “works for” to indicate it is an association-class (so it can contain attributes like “working hours”): first put the “Relation Node” on the “works for” association and then make the “is Associationclass” relation from that “Relation Node” to the desired “works for” class. The example can be seen below:



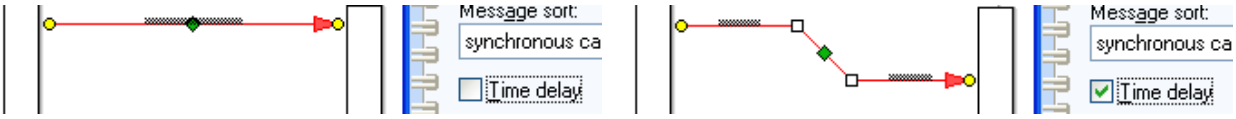
The boundary of “Lifelines” should not overlap, due to the automatic assignment of “Execution Specifications” based on being inside of a “Lifeline”. The exact boundary of an object is visible when the element is selected and is represented by the thick-chequered line as seen in the picture below:



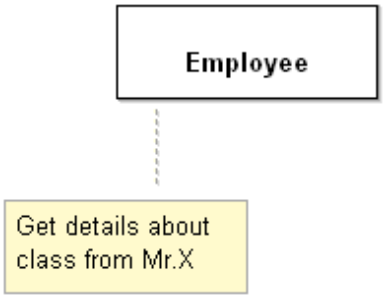
To create a “Composite State” (i.e. a “State” that contains other states) use the “State” type and set the attribute “Number of regions” to a value larger than 0, depending on how many regions are available. A simple example of a “Composite State” with only one region can be seen below:



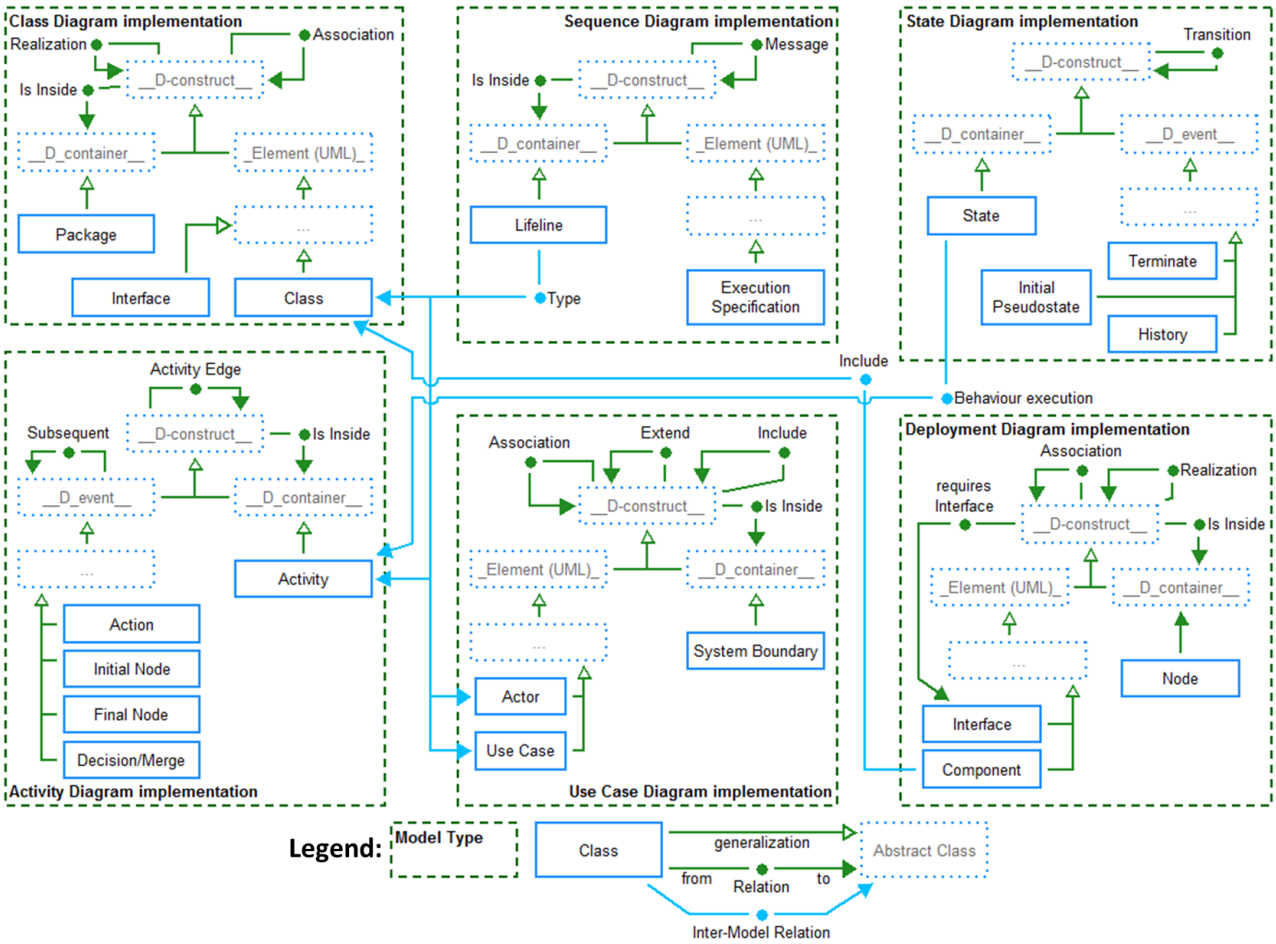
In “Sequence Diagrams” it is sometimes necessary to show a time delay by drawing “Message” relations diagonally. This is generally achieved by adding bend points to a relation. However, adding bend points can be difficult since the tool tries to draw horizontal (and vertical) relations. Therefore the “Message” relation contains an attribute called “Time delay”. Putting a check mark in this attribute will automatically add two bend points to the relation. Those can then be moved and other bend points can also be added more easily. Removing the check mark will also remove the bend points again. The two pictures below show a “Message” relation with the two possible states of the “Time delay” attribute:



It is possible to leave notes and comments in the models by using the “Note” class and also assigning those notes to any object using the “has Note” relation. The text displayed is specified through the “Description” attribute of the “Note”. An example can be seen below:

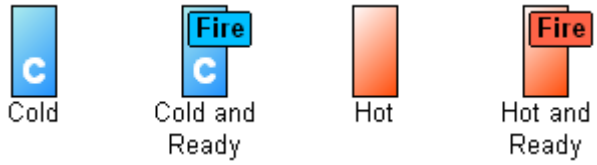


The following picture provides some detailed information about the implementation of UML in Bee-Up. More specifically it shows an excerpt of how the UML meta-model is implemented. Certain parts are provided by the platform to allow specific functionality, like __D_event__ and Subsequent used for process simulation (e.g. of Activity Diagrams). The “…” abstract class is used to represent complex generalization structures in the meta-model in a simplified manner.

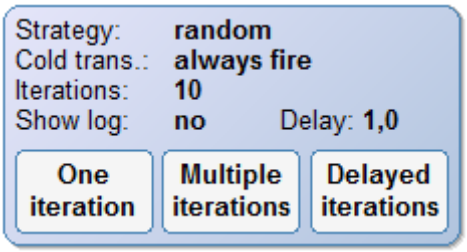


Specific information for PN modelling

The Petri Net implementation provides the base concepts ("Place", "Transition" and a connector called "Arc") as well as some additional ones for simulation and state storage. Tokens are modelled through the "Tokens" attribute of "Place" and are also visualized in them through small black circles and a number if there isn't enough room to draw all tokens. "Transitions" are also categorized into "Hot" (drawn in red color) and "Cold" (drawn in blue color with a white "C") transitions which is handled through the "Type" attribute. "Arcs" contain one attribute called "Weight" which is used to denote how many tokens should be consumed/created by the attached "Transition". The picture below shows the different notations of a Transition:



When the conditions to fire a transition are met (i.e. enough tokens in all preceding places) then a "Fire" button will appear on the transition (see picture above). Clicking on this button will fire the transition, meaning that the necessary tokens will be consumed in preceding "Places" and new ones will be added to the succeeding "Places". In Version 1.1 the "Arcs" have been extended with additional "Ready behaviour" for their following transitions, which allows firing a transition only when certain conditions are met without consuming any tokens. For more information check the "Ready behaviour" attribute information of an "Arc".

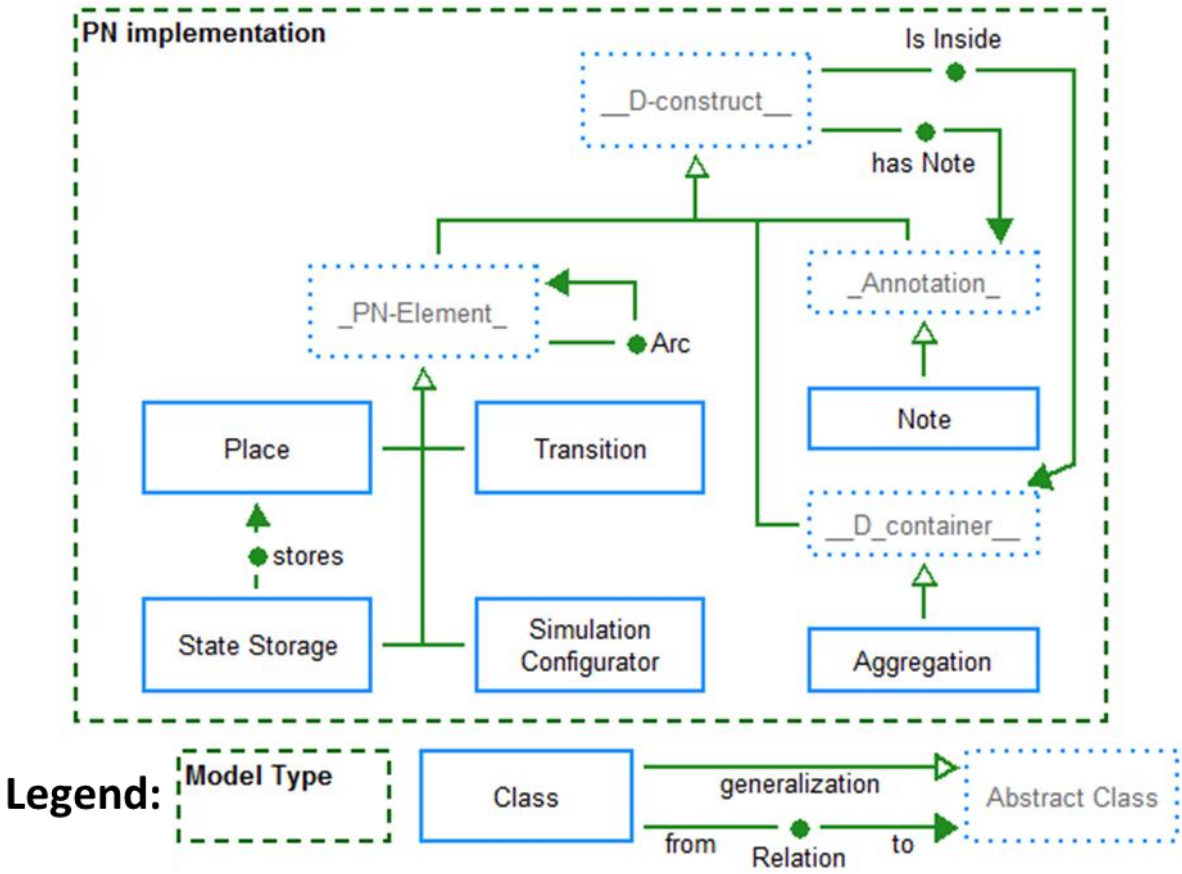


To simulate the net a special concept called "Simulation Configurator" is used (see picture above). It contains the configuration for a simulation run and is also used to start the simulation. The configuration is handled through the attributes of the notebook. See the additional information available for each attribute to find out more. The simulation can be started either by using the buttons on the drawing area or the buttons in the notebook. Through them either one iteration, multiple iterations or a slow simulation with delays between each iteration can be run. One iteration tries to fire all ready "Transitions". Should there not be enough tokens to fire all ready "Transitions" (e.g. several transitions requiring a token from a "Place" that only has one) then the selected "Transition conflict strategy" will be employed.

It is also possible to store the current state of a Petri Net and later restore it using "State Storage" (see picture below). In this context the state of the whole net is considered to be the amount of tokens in all the known places. When a "State Storage" is first added to the model it will store the state at that time in its attribute "Storage". This stored state can also be manipulated manually through that attribute. The notebook also provides two buttons: one to store the current state of the model (i.e. update the "State Storage" object with new values) and one to restore the state based on the "State Storage".

Version 1.1 also added two Model attributes: "Visualize priorities" and "Visualize probabilities". Selecting them changes the notations of transitions. "Visualize priorities" shows their relative priority in the model with a green bar on the left. "Visualize probabilities" shows a yellow bar on the right of cold "Transitions".

The picture below provides some detailed information about the implementation of PN in Bee-Up. More specifically it shows an excerpt of how the PN meta-model is implemented. Certain parts are provided by the platform to allow specific functionality, like __D_container__ used to automatically derive "Is Inside" relations.



General information for modelling

- Don't forget to save (so you are safe from data loss).
- Context menus are available for many things (e.g. objects in the **Modelling area**, entries of the **Explorer window** etc.). Making use of them can make work easier.
- Should a window be gone/missing (e.g. **Explorer window**, **Modelling window** etc.) → They can be toggled on and off through the menu "Window → Tools"
- Most icons have a tool tip, which provide a hint on what an icon is about. In case of the icons of the **Modelling window** the tool tip show the name of the type (e.g. Entity, Relation, has Attribute etc.).
- The tool also provides some functions for convenience. They can be accessed through the **Toolbar** using the  icons. From left to right they toggle the functionalities:
  - Align objects on the grid. The grid can be configured through the menu "View → Grid → Settings…"
  - Show the grid.
  - Use the modelling assistant. It supports the creation of new objects and relations from an existing object.
  - Automatically add bend points to relations when creating them to use using right angles.
- Notations can contain in some cases hyperlinks to other models/objects if the proper attributes are set. For example if a "Class" has the "Referenced class" attribute set, then the visualized name will be based on the referenced class and also a hyperlink to that class.
- The size of the **Drawing area** is represented by the white rectangle with the grey border in the **Modelling area** and can be resized similar to a window. Note that it is automatically extended as needed to fit any new objects that are created or old elements when their position is changed.
- Some model types (e.g. EPC, BPMN) have different modes. Those control which types of objects are available and visualized in the **Modelling area**. They can be changed through the menu "View → Mode"

- Object access locks can be changed through the menu item “View → Object access locks…”
- The tool has certain restrictions due to the things it uses as identifiers and also some limitations:
  - Models are identified through their type and a combination of their name and version (“[name] [version]”). Therefore two ER models, one with the name “Exercise” and version “3” and the other with the name “Exercise 3” are not allowed.
  - Objects in a model are identified through their type and their name. Therefore **no two objects of the same type in the same model can have the same name**. Because of that the “Attribute” in ER models uses “Denomination” and hides the name.
  - Relations in a model are identified by their type, their source object and their target object. Therefore **two relations of the same type linking the same objects in the same direction in the same model cannot exist**.
  - The **source and the target of a relation cannot be the same object**.
  - Relations cannot be the source or the target of other relations.
- To work around the limitations of relations the object type “**Relation Node**” (a small grey circle) is available in all model types:
  - It can be used to create multiple relations of the same type between the same two objects (e.g. several “Message flows” between two “Pools” in a BPMN model) by linking the first object to the “Relation Node” and then the “Relation Node” to the second object (this has to be done for each relation of the same type, between the same two objects, beyond the first direct relation).
  - It can be used to draw relations with the same source and target, by going through the “Relation Node” instead (e.g. when a “Class” is associated with itself). Place the “Relation Node”, then draw the relation from the object to the “Relation Node” and then from the “Relation Node” back to the object. Kindly add bend points to the created relations to increase the readability.
  - It allows the use of relations as the source or target of another relation by using the “Relation Node” instead. Freely place the “Relation Node” on an existing relation (e.g. association between two “Classes” in UML) and create the new relation (e.g. “is Associationclass”) from/to this “Relation Node” to/from the desired Object (e.g. the third “Class”).

Development Team

The Bee-Up modelling tool has been realized by the following team:

- Patrik Burzynski (patrik.burzynski@univie.ac.at): chief developer
- Dimitris Karagiannis: project owner

Additional used Tools

The following additional tools, implementations, binary codes etc. are used/included in Bee-Up and their according licenses apply:

- Apache Jena 3.1.0 – is used by the RDF Export functionality. Apache Jena website is available here: http://jena.apache.org/
- JDOM 2.0.6 – Developed by the JDOM Project (http://www.jdom.org/), it is used in the RDF Export functionality.