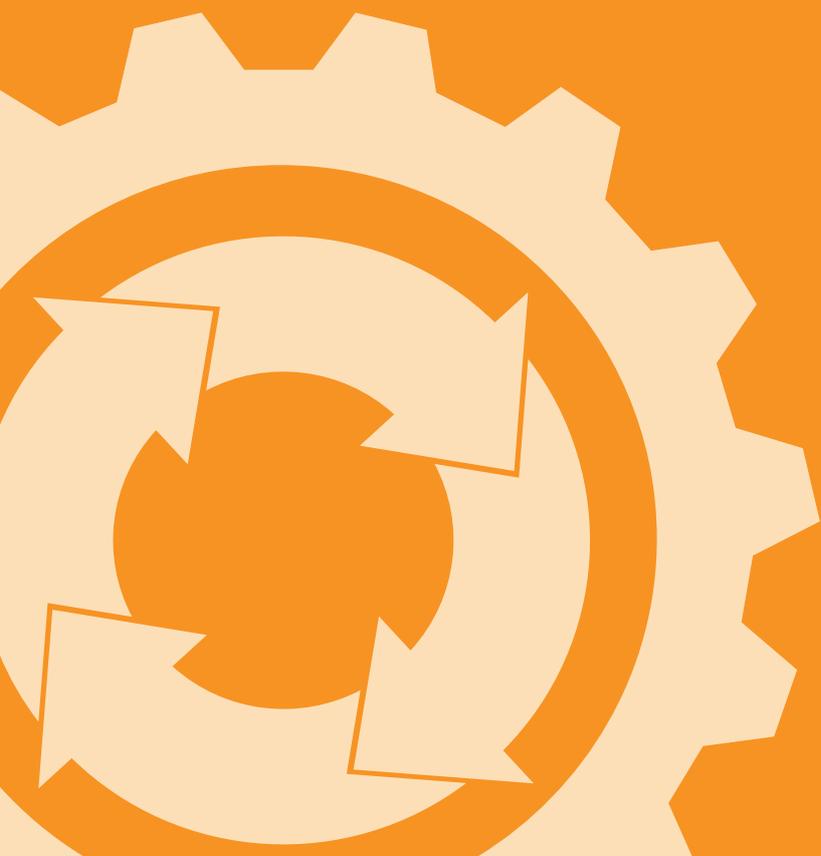




# USER MANUAL



**PRAGMADEV**  
modeling and testing tools

# Contents

- 1 Introduction** **4**
- 2 Supported BPMN constructs** **5**
- 3 Project manager** **7**
  - 3.1 Preferences . . . . . 7
  - 3.2 File manipulations . . . . . 11
  - 3.3 Checking the models . . . . . 13
- 4 BPMN editor** **15**
  - 4.1 Symbols . . . . . 15
  - 4.2 Hierarchy . . . . . 17
  - 4.3 Link with MEGA HOPEX . . . . . 17
  - 4.4 Editor . . . . . 18
    - 4.4.1 Selection modes . . . . . 18
      - 4.4.1.1 Select only . . . . . 18
      - 4.4.1.2 Select or edit . . . . . 19
    - 4.4.2 Re-select last tool . . . . . 19
    - 4.4.3 Handling broken segments . . . . . 20
    - 4.4.4 Modifying symbol types . . . . . 22
    - 4.4.5 Connecting Call activities . . . . . 24
- 5 Executor** **28**
  - 5.1 Underlying principles . . . . . 28
  - 5.2 Controls . . . . . 29
  - 5.3 Behavior . . . . . 30
    - 5.3.1 Start . . . . . 30
    - 5.3.2 Sequence flows . . . . . 30
    - 5.3.3 Message flows . . . . . 30
      - 5.3.3.1 Implicit resolution . . . . . 31
      - 5.3.3.2 Explicit resolution . . . . . 33
    - 5.3.4 Call activities . . . . . 38
    - 5.3.5 Gateways . . . . . 38
      - 5.3.5.1 Inclusive . . . . . 38
      - 5.3.5.2 Exclusive . . . . . 40
      - 5.3.5.3 Parallel . . . . . 42
      - 5.3.5.4 Event . . . . . 44
  - 5.4 Execution traces . . . . . 50
    - 5.4.1 Recording . . . . . 50

5.4.2	Replay . . . . .	51
5.4.2.1	Single-trace execution . . . . .	53
5.4.2.2	Multi-trace execution . . . . .	54
<b>6</b>	<b>MSC and PSC Editor</b>	<b>55</b>
6.1	Overview . . . . .	55
6.2	MSC & PSC reference guide . . . . .	55
6.2.1	General diagram format . . . . .	55
6.2.2	Links . . . . .	56
6.2.2.1	Message links . . . . .	56
6.2.2.2	PSC-specific normal, required and failed message syntax	57
6.2.2.3	Sequence flow . . . . .	58
6.2.2.4	Lifeline components . . . . .	58
6.2.3	Main symbols . . . . .	63
6.2.3.1	Lifeline . . . . .	63
6.2.3.2	Collapsed lifelines . . . . .	63
6.2.3.3	Inline expressions . . . . .	64
6.2.3.4	Absolute times . . . . .	66
6.2.3.5	Comments . . . . .	67
6.2.3.6	Texts . . . . .	67
6.3	MSC editor . . . . .	68
6.3.1	Specific tools . . . . .	68
6.3.2	Symbol creation . . . . .	70
6.3.3	Manipulating components in lifelines . . . . .	72
6.3.4	MSC symbol and link properties . . . . .	73
6.3.5	Message parameters display . . . . .	74
6.3.6	Conformance checking: diagram diff & property match . . . . .	75
6.3.6.1	Basic MSC diff: trace vs. trace, spec. vs. spec., ... . . . .	76
6.3.6.2	Spec vs. trace comparison . . . . .	78
6.3.6.3	Property match . . . . .	79
<b>7</b>	<b>Verifier</b>	<b>82</b>
7.1	Architecture . . . . .	82
7.2	Properties . . . . .	83
7.3	Launch a verification . . . . .	83
7.4	Result analysis . . . . .	87
7.4.1	Full state space exploration . . . . .	87
7.4.2	Property verification . . . . .	89
<b>8</b>	<b>Glossary</b>	<b>93</b>

# 1 Introduction

Complex organizations or systems operations are based on processes described in graphical models. The most popular notation is BPMN. It describes what the different participants in a process do and how they interact with each other. These processes must be thoroughly discussed before they are applied in a real situation. Any misunderstanding of the process might lead to a catastrophic situation in operation.

PragmaDev Process is a set of tools:

- **A project manager**  
The project manager will gather all the files of your project in one place for easy access.
- **BPMN Editor**  
The BPMN editor allows you to design your process graphically. It is also possible to import an existing diagram from another tool through the XML standard BPMN.
- **BPMN Executor**  
The BPMN executor animates your BPMN. The possible path of execution will be graphically displayed.
- **BPMN Tracer**  
When executing the model step by step it is possible to trace the different steps in an MSC. One or several scenario can be replayed automatically against the model.
- **BPMN Verifier**  
The Verifier relies on a third party tool called OBP (Observer Based Prover) developed by ENSTA Bretagne. This tool will automatically try all possible scenarios in the model. There are two possible outcomes of this exploration:
  - Overall number of possible steps in the model.  
If that number is too high compared to the size and complexity of the model, it probably means the model is wrong.
  - Verification of a property  
The property to be verified is expressed with a PSC (Property Sequence Chart) that is a sort of MSC. The Verifier will search if there is a scenario that verifies the property.

## 2 Supported BPMN constructs

PragmaDev Process is a BPMN Viewer, Editor, and Executor. There are a few restrictions to these features as some BPMN constructs can be viewed but not edited or not executed. A summary of supported constructs is given in table 2.1.

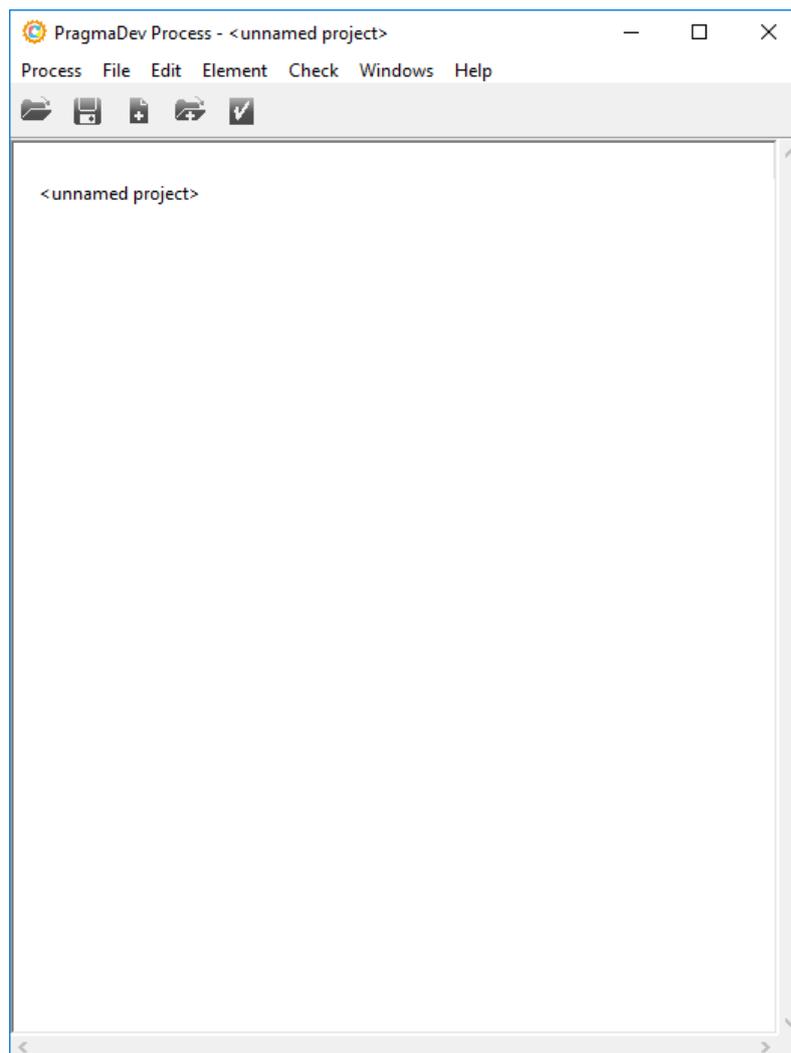
**Note:** Conditional and default flows are not supported yet in the Editor.

Table 2.1: Supported BPMN constructs.

BPMN construct	Viewer	Editor	Executor
Pool	YES	YES	YES
Lane	YES	YES	YES
Task	YES	YES	YES
Sub-Process (Transaction, Ad-Hoc, and Event)	YES	NO	NO
Call-Activity	YES	YES	YES
Data (Object, Input, Output, and Store)	YES	NO	NO
Start Event			
None, Message, Timer, Signal	YES	YES	YES
Conditional, Multiple, Parallel Multiple	YES	NO	NO
End Event			
None, Message, Signal, Terminate	YES	YES	YES
Escalation, Error, Compensation, Multiple, Cancel	YES	NO	NO
Intermediate Event (Throw)			
None, Message, Signal	YES	YES	YES
Escalation, Compensation, Link, Multiple	YES	NO	NO
Intermediate Event (Catch)			
None, Message, Timer, Signal	YES	YES	YES
Link, Conditional, Multiple, Parallel Multiple	YES	NO	NO
Boundary Event	YES	NO	NO
Gateway			
Exclusive, Inclusive, Parallel, Event	YES	YES	YES
Complex, Event Start, Parallel Event Start	YES	NO	NO
Group	YES	NO	-
Text Annotation	YES	NO	-
Choreography	NO	NO	NO
Conversation	NO	NO	NO
Sequence Flow	YES	YES	YES
Message Flow	YES	YES	YES
Association	YES	NO	NO

## 3 Project manager

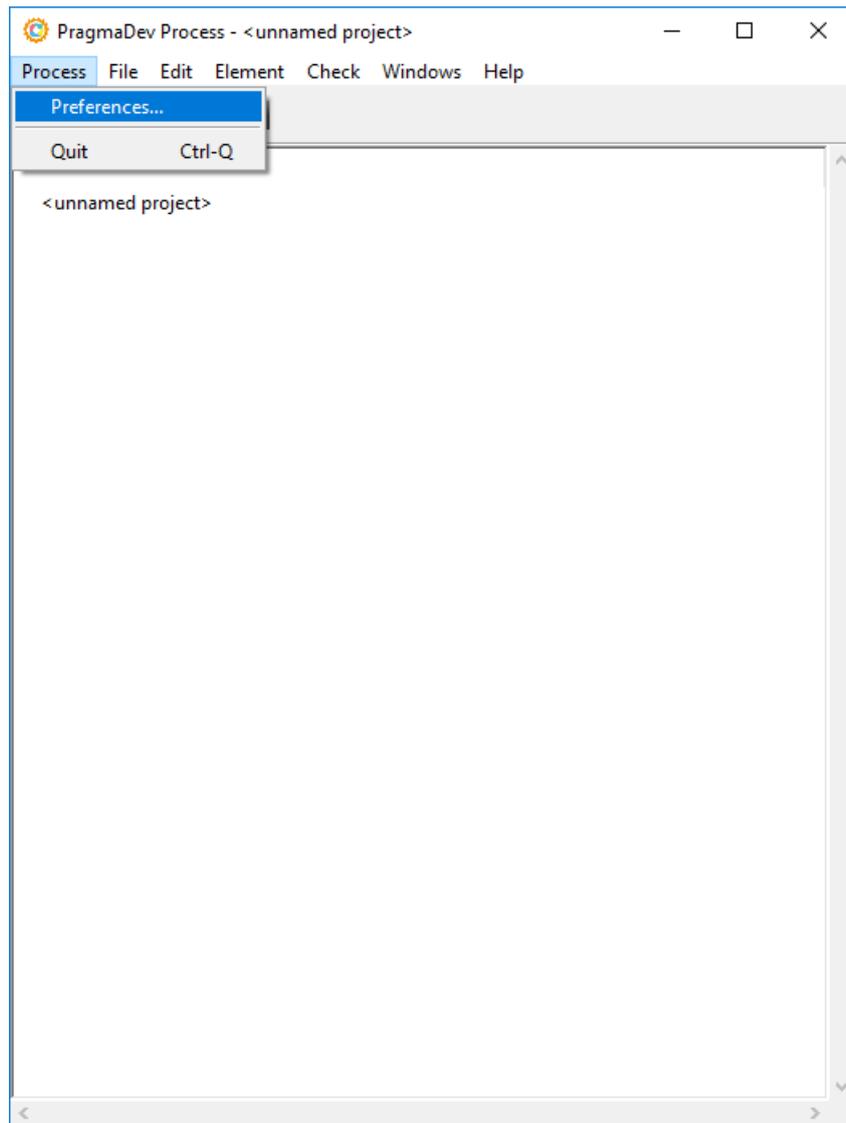
When starting the tool, after setting the licensing mechanism described in the Installation manual, the Project manager pops up.



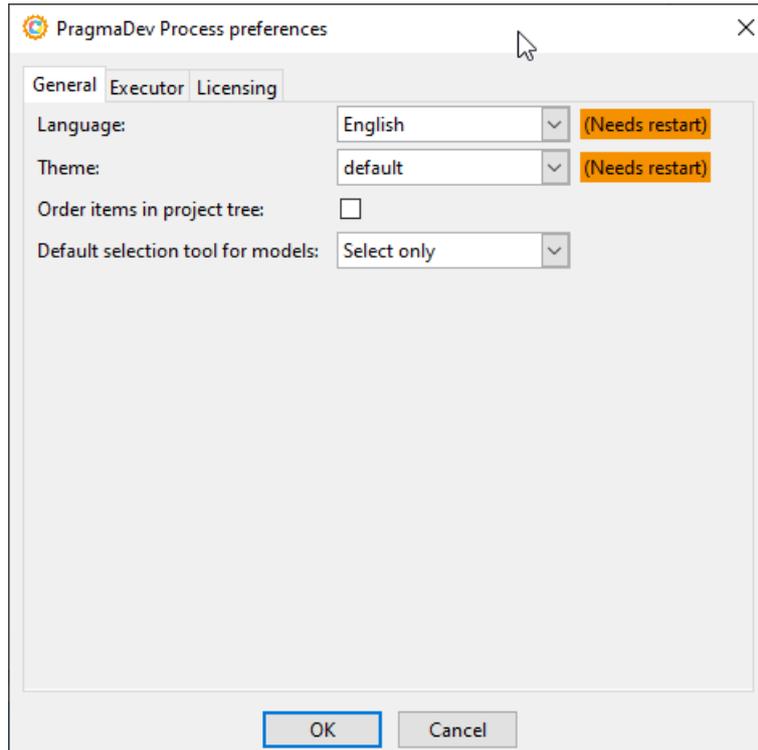
A project allows you to gather all the files related to the on going project: BPMN, traces, and properties. By default an empty unnamed new project is created at startup.

### 3.1 Preferences

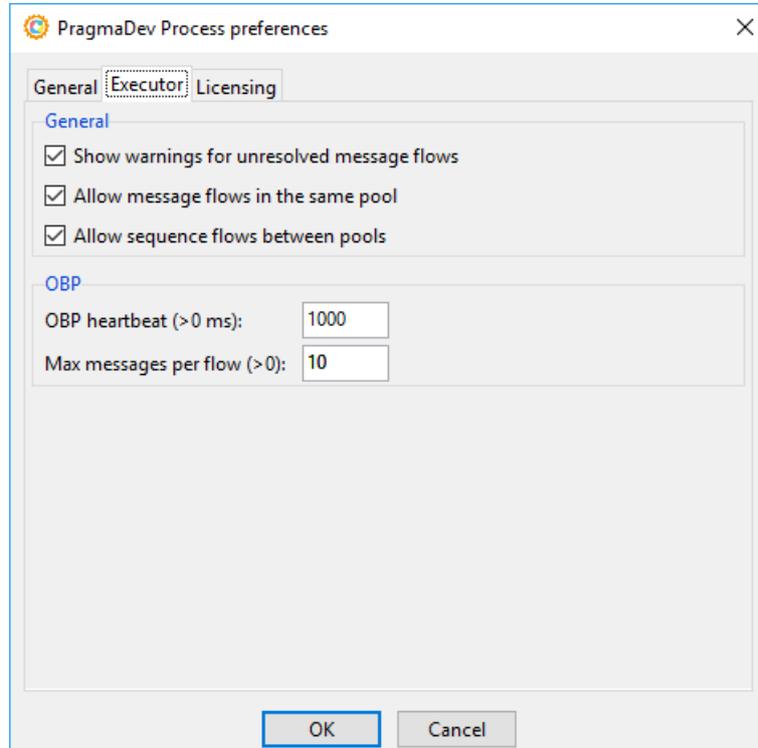
The first menu in the tool gives access to the preferences window:



In the *General* tab you can change the language, switch between themes for the GUI (mostly useful for Linux users), order items in the project by alphabetical order (requires reload), and change the default selection mode of the BPMN editor:



The *Executor* tab contains some rules that can be enabled or disabled during BPMN semantics check and execution, and also the default values used with OBP:



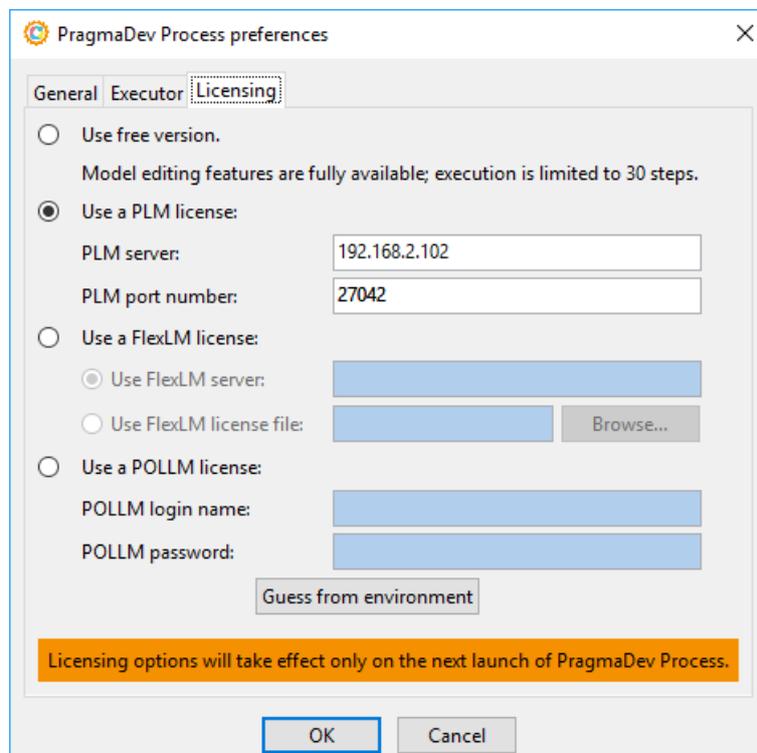
The following *General* rules are available:

- *Show warnings for unresolved message flows*  
When enabled, unresolved messages flows will generate warnings during a semantics check (or execution). This option concerns message flows incoming or outgoing an empty pool or lane (back-box). Message resolution is explained in "Explicit resolution".
- *Allow message flows in the same pool*  
The BPMN semantic does not allow message flows within the same pool. This option allows to override this rule.
- *Allow sequence flows between pools*  
The BPMN semantic does not allow sequence flows to cross pool boundaries, i.e., they are allowed only within the same pool. This option allows to override this rule.

The OBP related values are explained in "Launch a verification". They are:

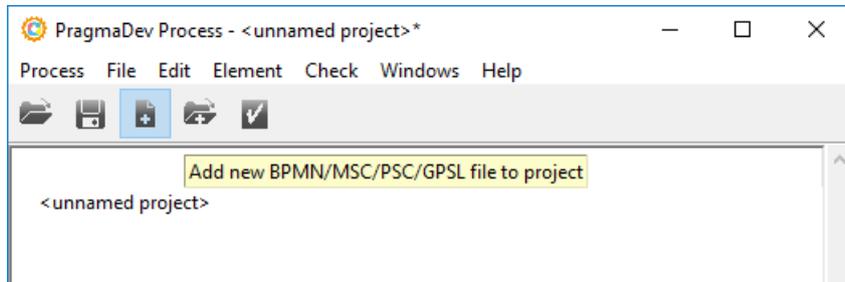
- *OBP heartbeat*  
Allows to change the default refresh value of the status information returned by OBP during exploration.
- *Max messages per flow*  
Upper limit for the number of pending message flows (sent but not received messages).

You can also change the licensing mechanism that is explained in the Installation Manual:

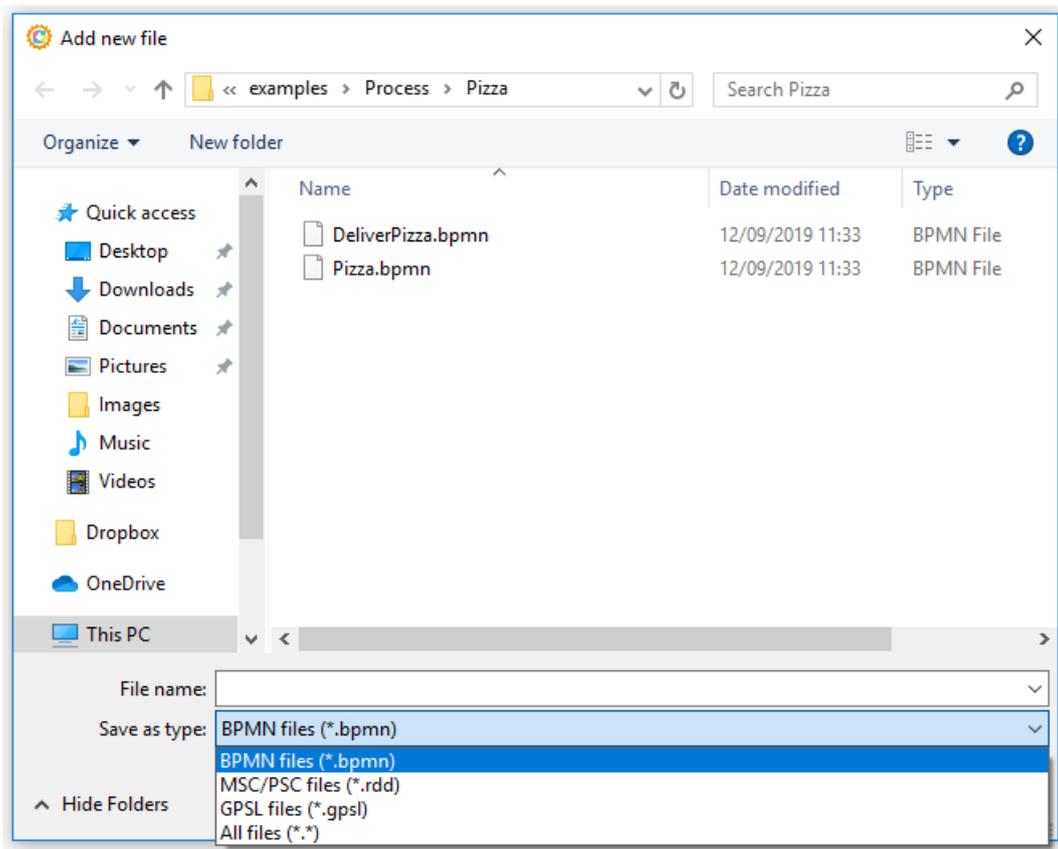


### 3.2 File manipulations

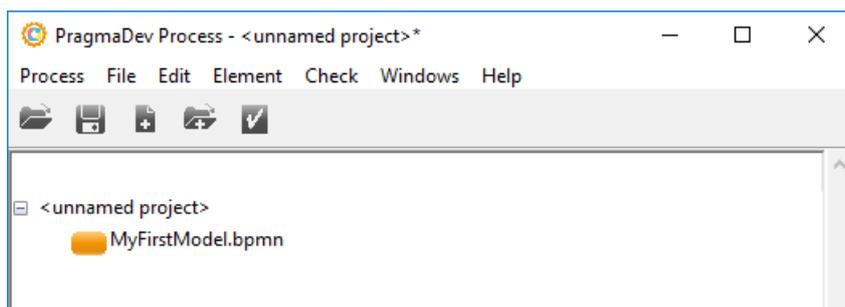
A file is added to the project with the Add file button:



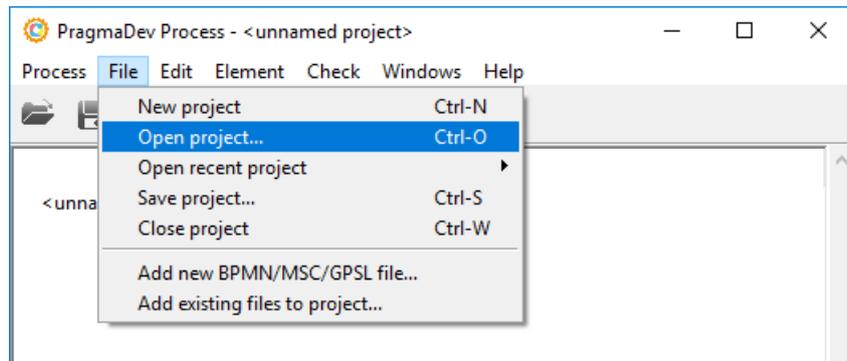
A file browsing window will open with the bpmn, rdd, gpsl and file extensions:



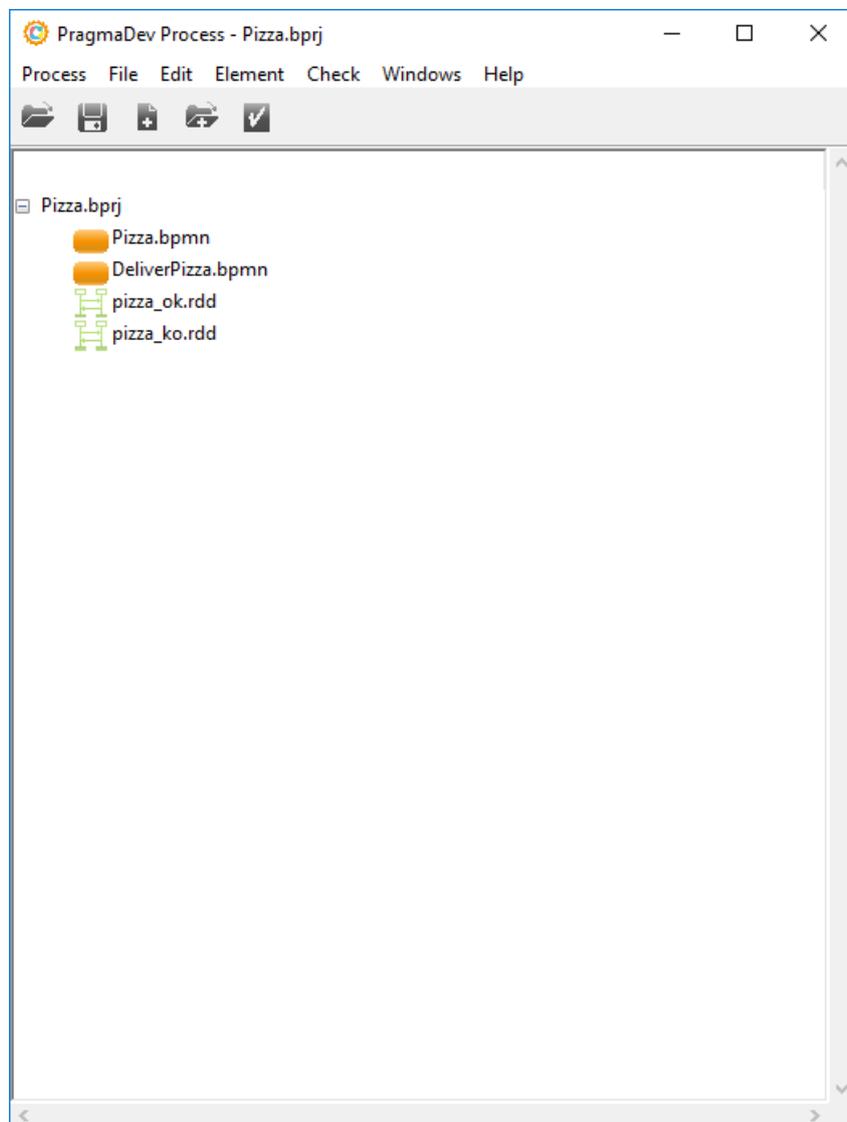
A new file is added to the project:



Or you can directly open a project:

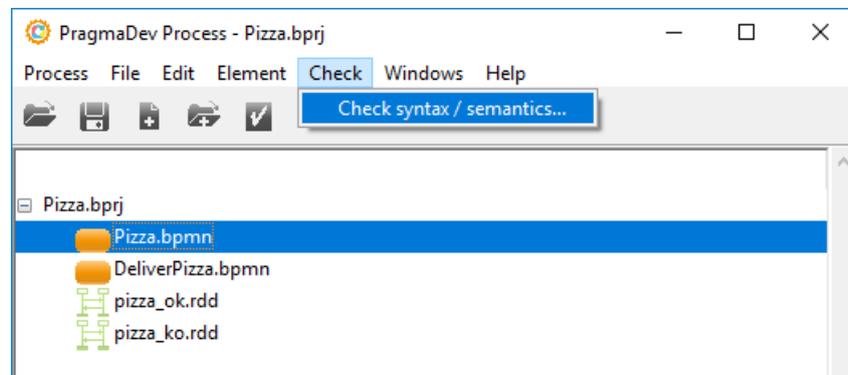


The selected project will display its associated files:

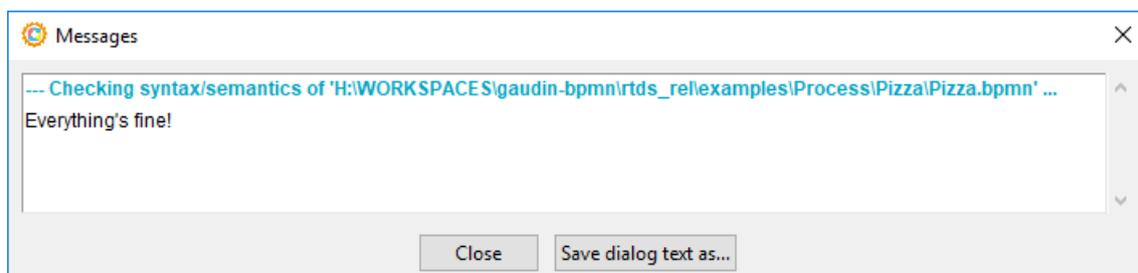


### 3.3 Checking the models

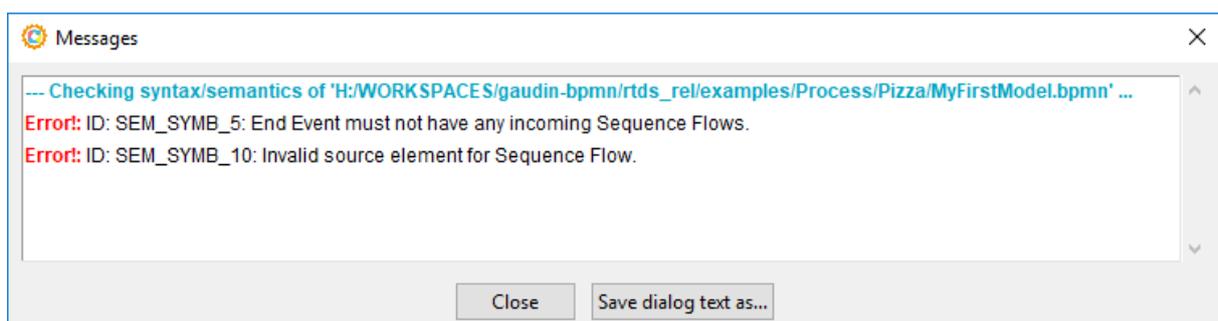
It is possible to check the syntax of a model from the Project manager. Select the diagram and go to the Check menu:



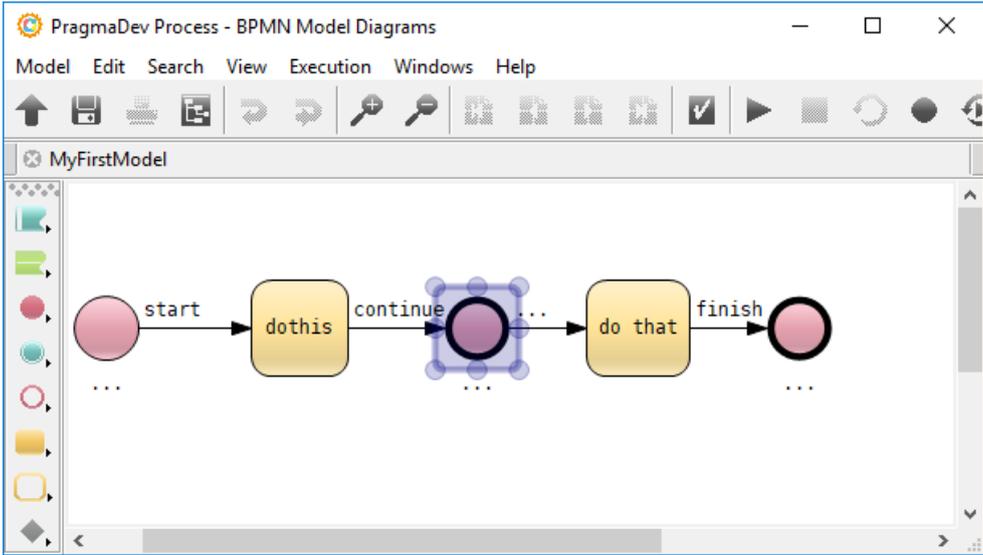
In the case no errors are found:



In the case errors are found:



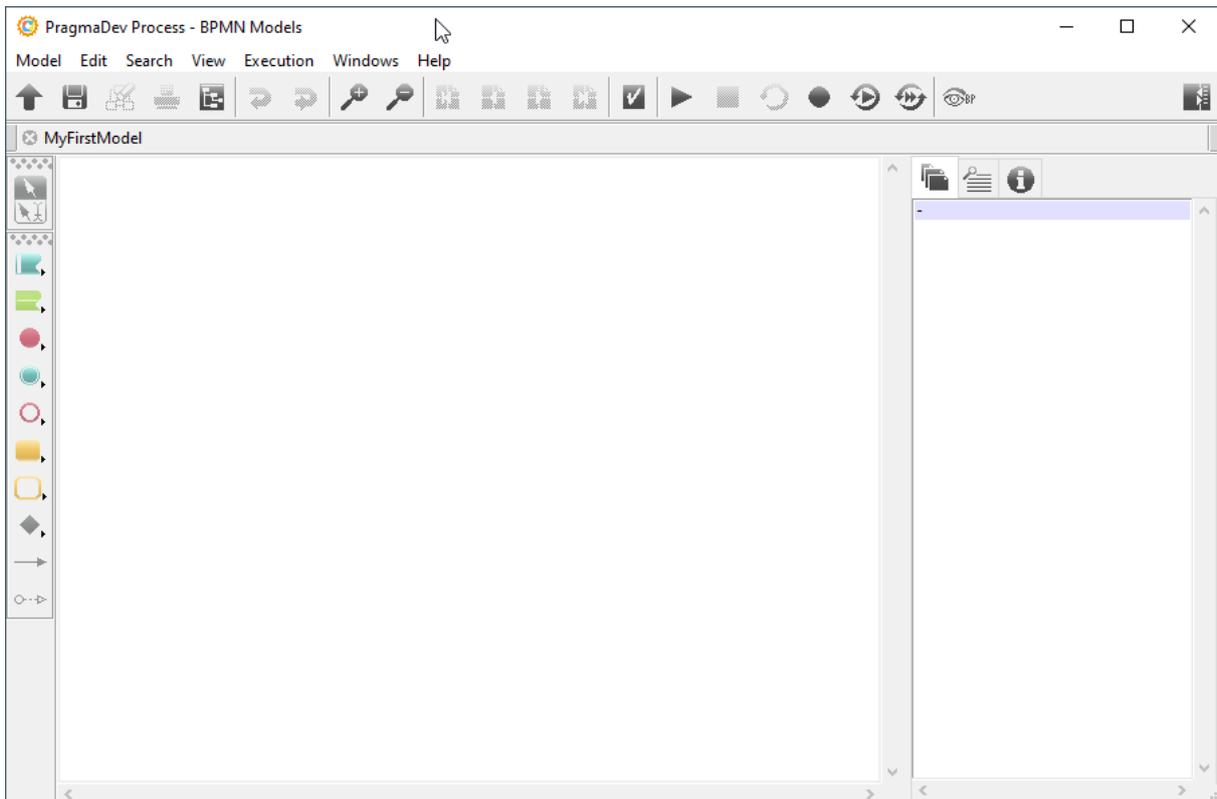
A double click on the error line will open the diagram and selected the symbol related to the error:



## 4 BPMN editor

### 4.1 Symbols

A double click on any BPMN diagram in the Project manager will open the BPMN editor:



The different symbols are organized in categories on the left side of the editor. Below is the list of categories and symbols available in each category:

- Pools



- Horizontal pool
- Vertical pool

- Lane



- Horizontal lanes
- Vertical lanes

- Start events



- Plain start event
- Message catch start event
- Timer start event
- Signal catch start event

- Intermediate events



- Plain intermediate event
- Message throw intermediate event
- Message catch intermediate event
- Timer intermediate event
- Signal throw intermediate event
- Signal catch intermediate event

- End events



- Plain end event
- Message throw end event
- Signal throw end event
- Terminate end event

- Tasks



- Plain task
- Message send task
- Message receive task
- Service task
- User task
- Manual task
- Script task
- Business

- Call activities



- Plain call activity
- User call activity
- Manual call activity
- Script call activity
- Business rule call activity

- Process call activity

- Gateways



- Exclusive gateway
- Parallel gateway
- Inclusive gateway
- Event gateway

- Sequence flow

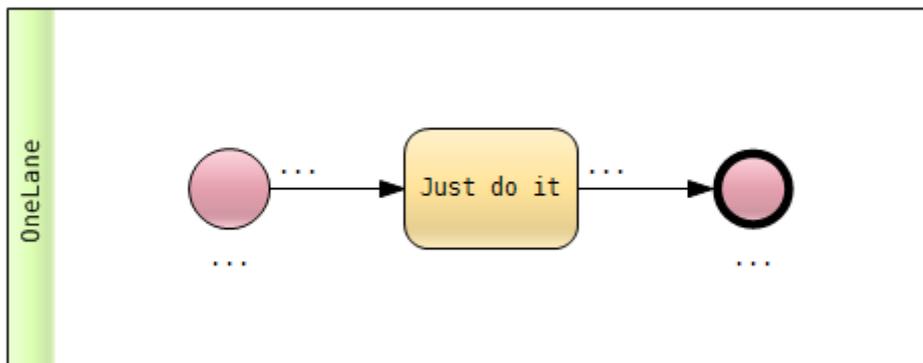


- Message flow



## 4.2 Hierarchy

BPMN has container symbols that include other symbols. Typically Pools and Lanes are containers for execution symbols. Drawing a symbol in a container symbol will automatically associate the contained symbol with its container. Moving the container will move all the contained symbols. Graphically dragging a symbol out of the container will dissociate the symbols. In the example below, the start, the task and the end symbols are contained in the One Lane lane. Moving the lane symbol will move all the contained symbols.



## 4.3 Link with MEGA HOPEX

PragmaDev Process has a specific link with HOPEX tool as we are one of MEGA’s partners. Users of MEGA HOPEX can launch the PragmaDev Process executor from HOPEX on a set of selected diagrams. Diagrams are automatically exported in BPMN and can be viewed in PragmaDev Process viewer. In that specific situation the models can not be edited, they can only be viewed. If a modification should be made on the model it

should be on the source model in HOPEX. For that matter click on the "Go to source model" button at the top of the editor:



Please note this feature only works with HOPEX web front end.

## 4.4 Editor

The BPMN editor is quite straight forward but there are a few particular features that deserve to be explained.

### 4.4.1 Selection modes

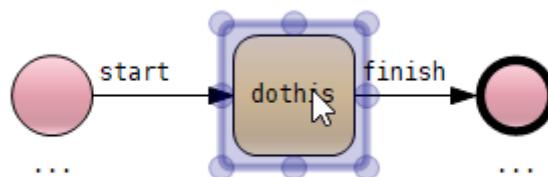
The editor provides 2 modes for selecting and editing. It is possible to toggle from one to the other with this button:



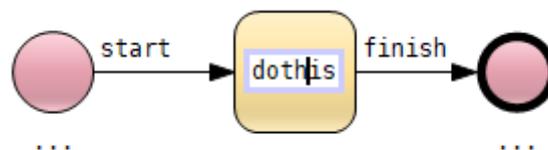
#### 4.4.1.1 Select only



This is the default mode. Symbols have a graphical shape and some text inside. To select the graphical shape of the symbol you can click anywhere in the symbol including the text area:



To edit the text, double click on the text:

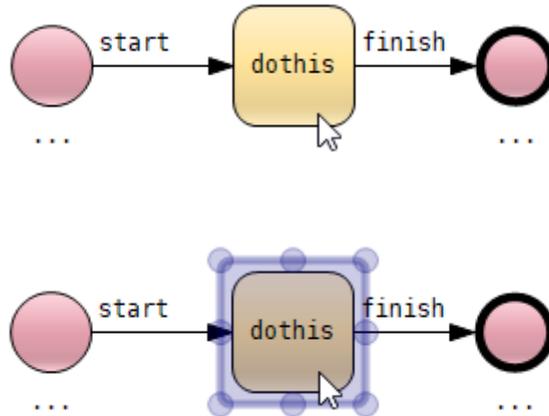


This mode is more efficient if most of the work is to graphically re-organize the diagram.

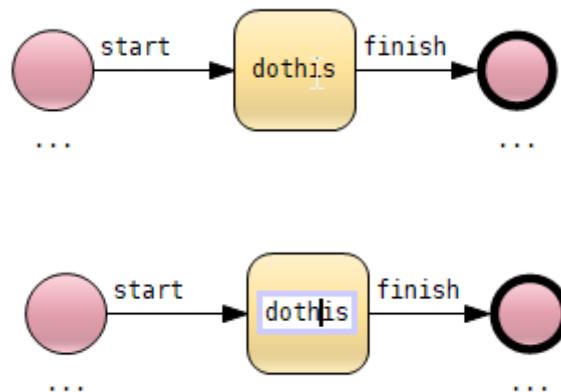
#### 4.4.1.2 Select or edit



Symbols have a graphical shape and some text inside. To select the graphical shape of the symbol you should click between the edge of the symbol and the text:



Clicking on the text will actually edit the text:



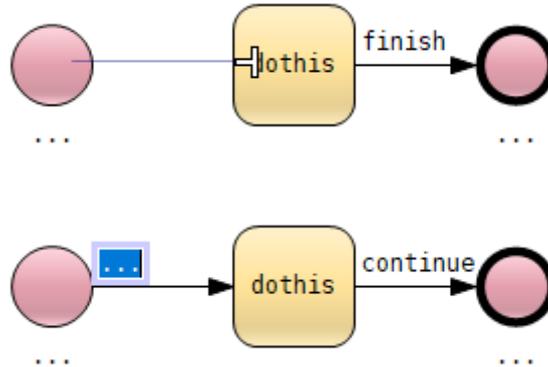
This mode is more efficient if most of the work is to edit the text contents of the symbols.

#### 4.4.2 Re-select last tool

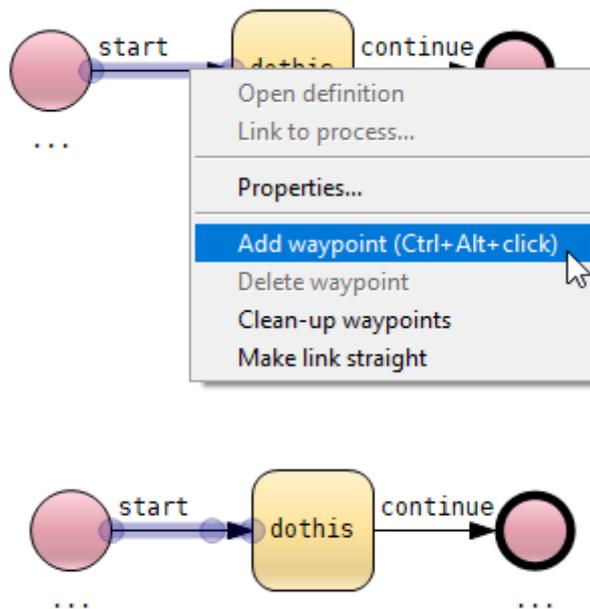
When using the same tool again and again, it might be tedious to select it again and again. For that situation pressing Ctrl + Space bar actually re-selects the last used tool.

### 4.4.3 Handling broken segments

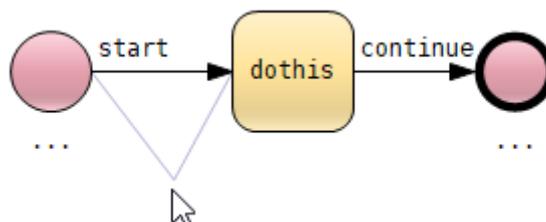
When connecting two symbols with a sequence or a message flow, the link goes straight from one symbol to the other:

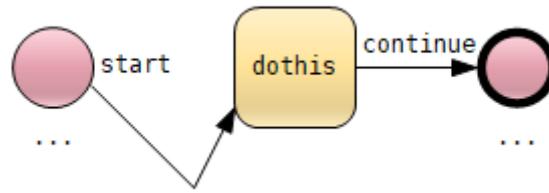


Would you like a broken segment to connect the two symbol it is possible to right click on the segment and add a "waypoint" that is an angle in the connection:

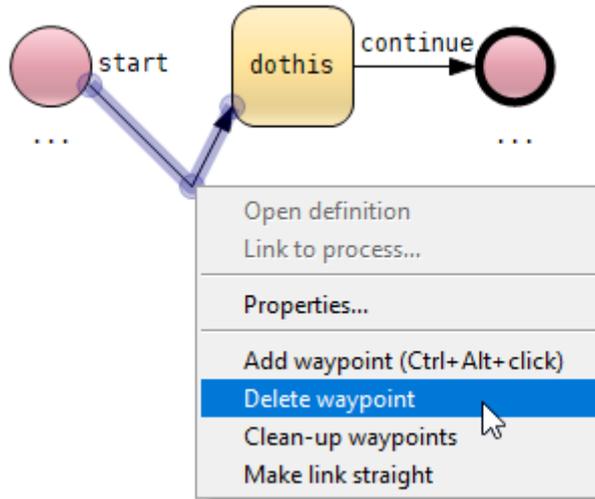


You can just drag the newly created waypoint to where you want the segment to brake:



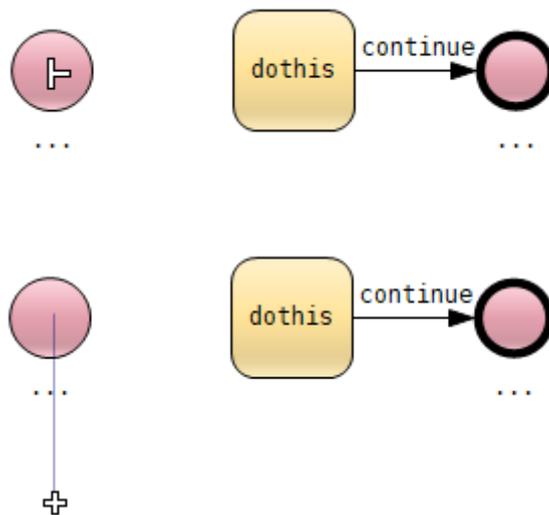


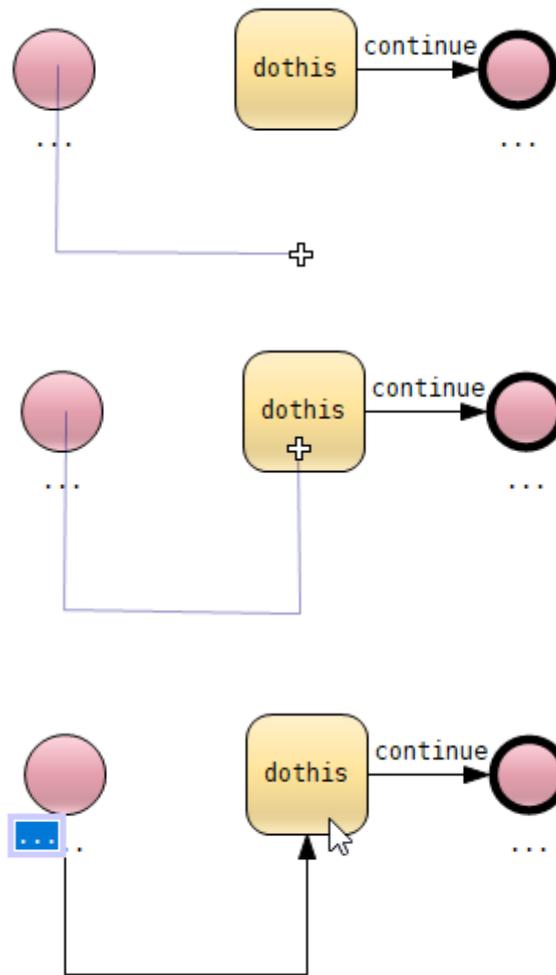
To delete a waypoint, select the waypoint and right click to get the contextual menu:



It is also possible to ask the editor to remove all waypoints in the segment to get a default straight link.

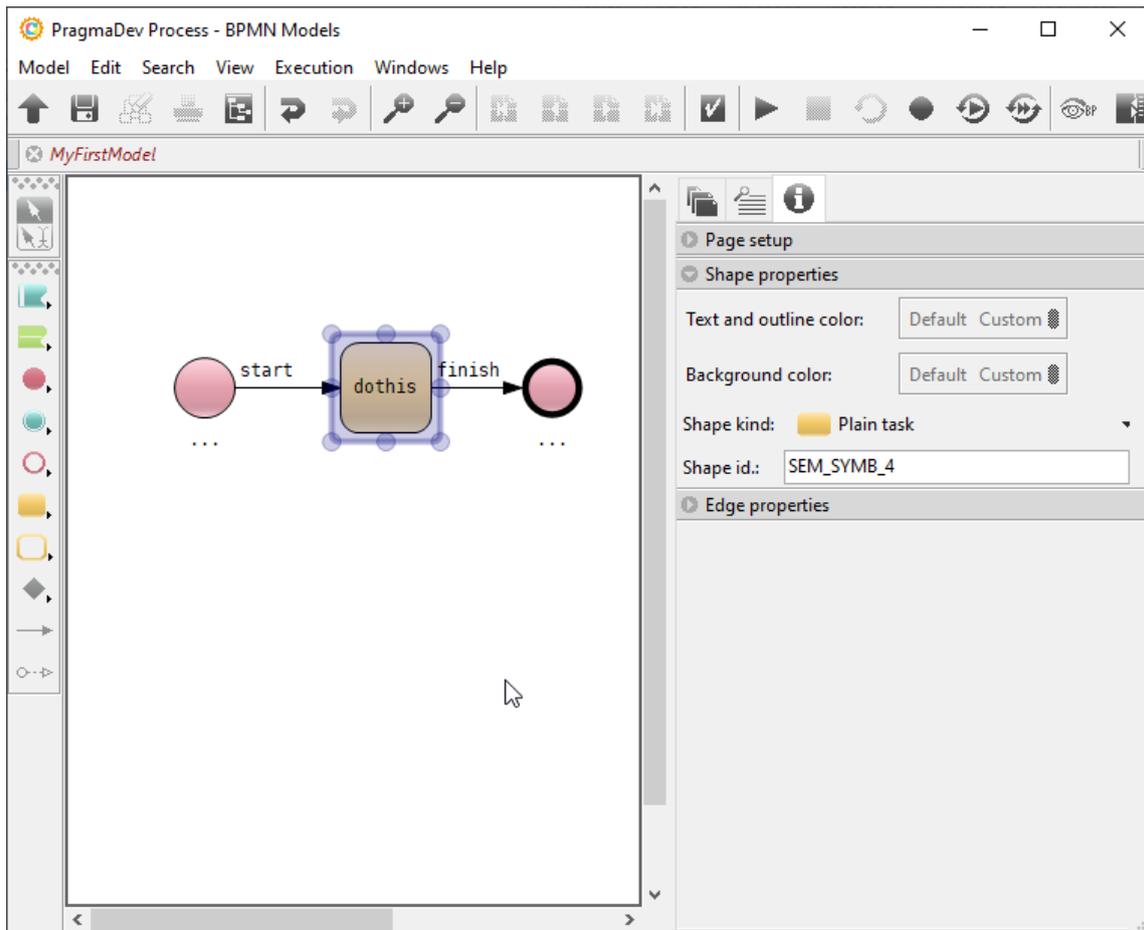
It is also possible to create broken segment from the start, to do so hold the Shift key down while create the link and click where the waypoint should be:



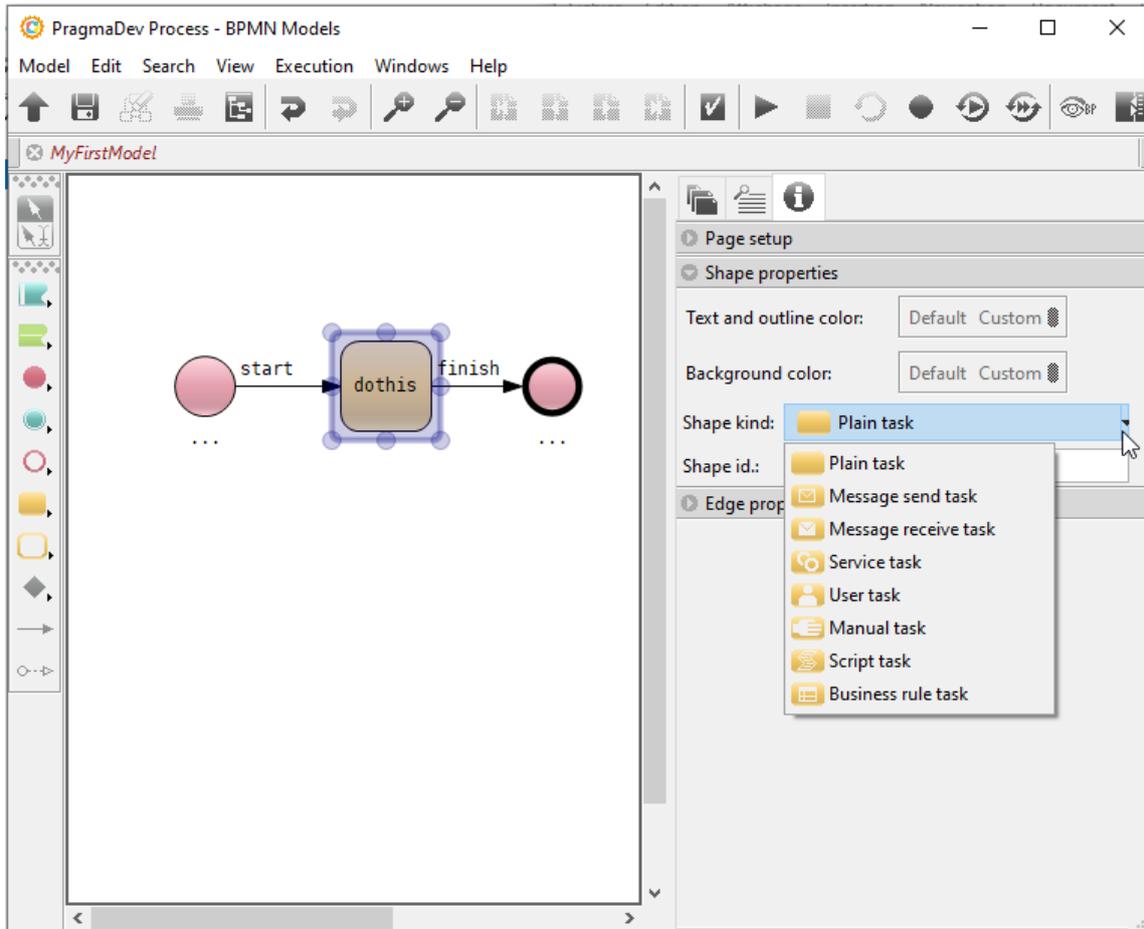


#### 4.4.4 Modifying symbol types

When selecting a symbol in the editor, some complementary information are displayed in the right panel:



Click on the Shape kind to change the symbol type:

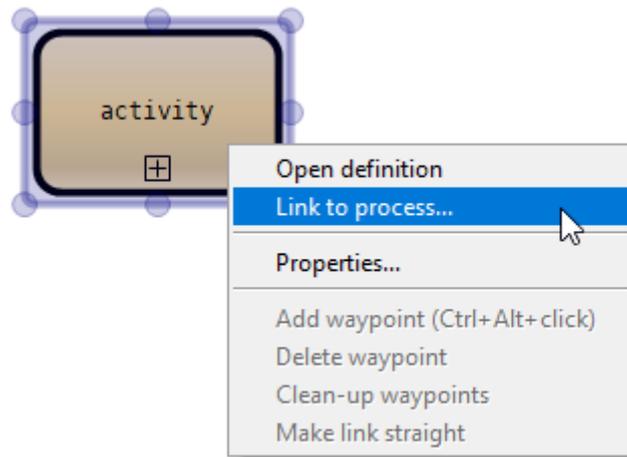


### 4.4.5 Connecting Call activities

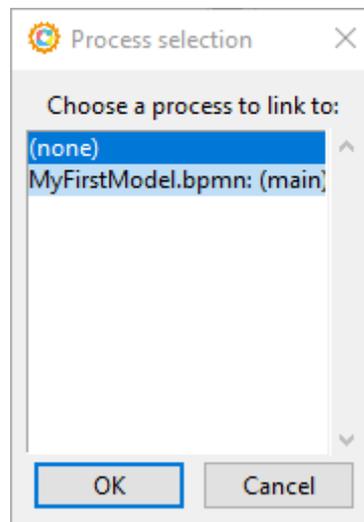
A Call activity references a sub-process.



To link the Call activity to the sub-process description right click on the symbol:



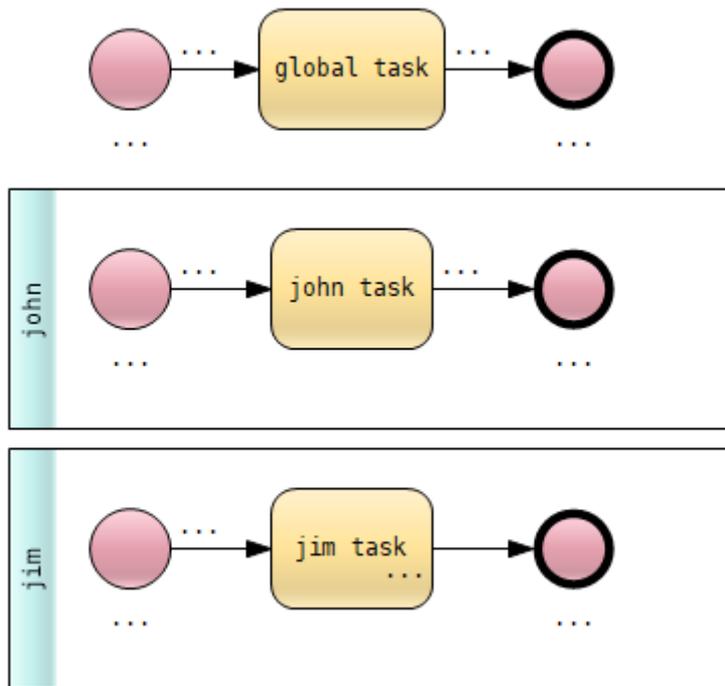
This will open a selection window that will list the possible subprocesses:



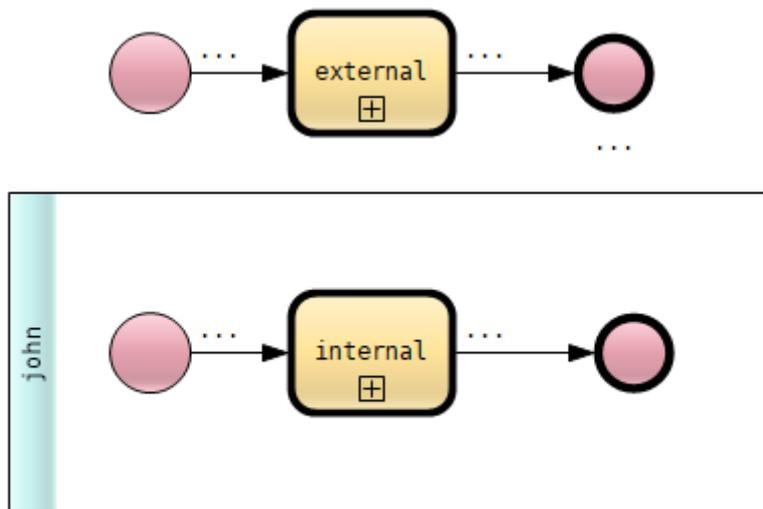
Now it is important to note the list of possible choices depend on the context of the sub-process as well as the context of the caller. The possible connection will list:

- subprocesses that are in the same pool as the caller,
- subprocesses that are in no pool (implicitly the pool of the caller).

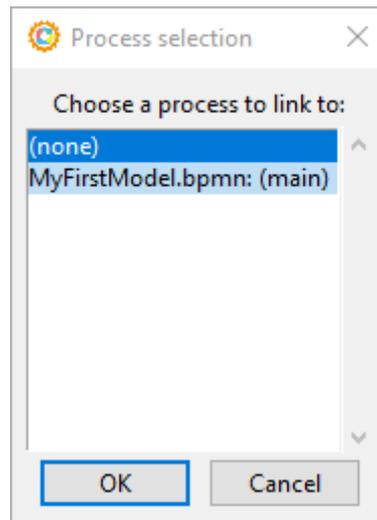
To illustrate this let's take the example of three processes described in MyFirstModel diagram, one is defined outside of any pool, one is defined in john pool, and the last one is defined in jim pool:



In another diagram we have two call activities, one outside of any pool, and one in john's pool:

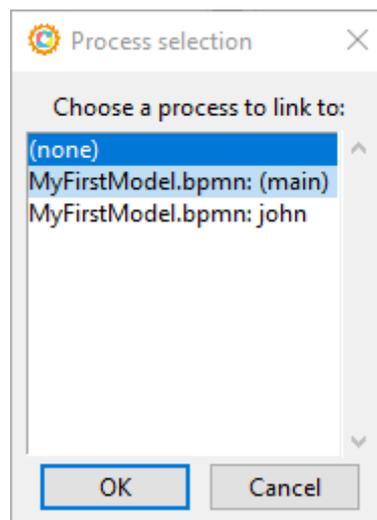


The possible links for external are:



The only possible choice is (main) of MyFirstModel meaning the process defined out of the pools.

The possible links for internal are:



The two possible choices are:

- the process that has been defined outside of any pool,
- the process defined in john's lane.

Once the link is done it will be used for navigation: double click will open the sub-process definition, and for execution: click on the call activity will execute the sub-process.

# 5 Executor

## 5.1 Underlying principles

Each BPMN element in the model has a state of execution:

- *None*  
The element does not accept any action from the user, and it has never been enabled or disabled.
- *Active*  
The element is waiting for either an enabling or disabling action from the user.
- *Ready*  
An enabling action was issued on the element, but the element cannot be enabled yet because it depends on the state of other elements.
- *Enabled*  
An enabling action was previously issued on the element, and all enabling conditions have been fulfilled (i.e., the other elements it depends on are in the required state).
- *Disabled*  
A disabling action was issued on the element.

There are two types of actions, that can be issued only on *Active* elements:

- *Enabling*: click on active element.
- *Disabling*: right-click on active element.

In general, these actions can be issued on sequence flows, message flows, and process call-activities.

Since an element can be enabled or disabled several times (i.e., many flows of execution can go through the same element), each element has actually a list of states of execution. During execution the most recent state of the element is displayed in color as show in the following table.

Table 5.1: Color representation of execution states.

State	Color	Example
None	No color	
Active	Blue	
Ready	Orange	
Enabled	Green	
Disabled	No color	

Having the most recent state shown, an implicit priority is created in displaying execution states, i.e., *Active* has the highest priority, then *Ready*, and last *Enabled* and *Disabled*. For example, given an element that is *Active* in a flow and *Ready* in another flow, its *Active* state will be shown in blue color.

## 5.2 Controls

The executor is controlled from the diagram editor with the following tools:



It is started with the *Start* button

Once started, it can be stopped with the *Stop* button

Once started it can be reset with the *Reset* button

The execution scenario can be recorded with the *Record* button

A single execution scenario can be replayed with the *Replay* button

Several execution scenarios can be replayed with the *Replay all* button

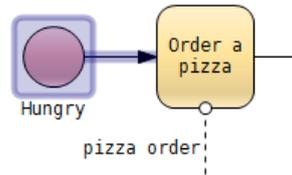
Last execution step can be undone with the *Undo* button

Last undone execution action can be re-executed with the *Redo* button

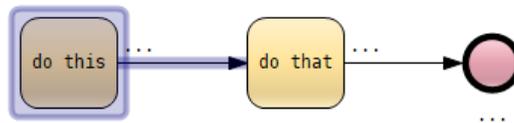
## 5.3 Behavior

### 5.3.1 Start

When starting, the executor will look for explicit and implicit starting points in the model. An explicit starting point is... the sequence flow following the *Start* symbol:



Any outgoing flow (message or sequence) from a task without incoming flows can be an implicit starting point. Here is a typical example:



### 5.3.2 Sequence flows

Sequence flows can be either enabled or disabled. However, even though most sequence flows can be enabled (if they are *Active*), disabling an *Active* sequence flow follows the following rules:

- A normal (uncontrolled) sequence flow can be disabled only if it is an outgoing flow of an inclusive gateway, and at least one other outgoing flow of the gateway is still *Active* or has been enabled.
- In absence of normal (uncontrolled) flows, an outgoing conditional sequence flow can be disabled only if at least one other outgoing conditional flow is still *Active* or has been enabled.
- Default sequence flows can be always disabled if they are *Active*.

### 5.3.3 Message flows

Message flows accept only enabling action, i.e., they cannot be disabled. The *resolution* of a message flow is the identification of both its *endpoints* (sender and receiver). The Executor supports two kinds of message flow resolution:

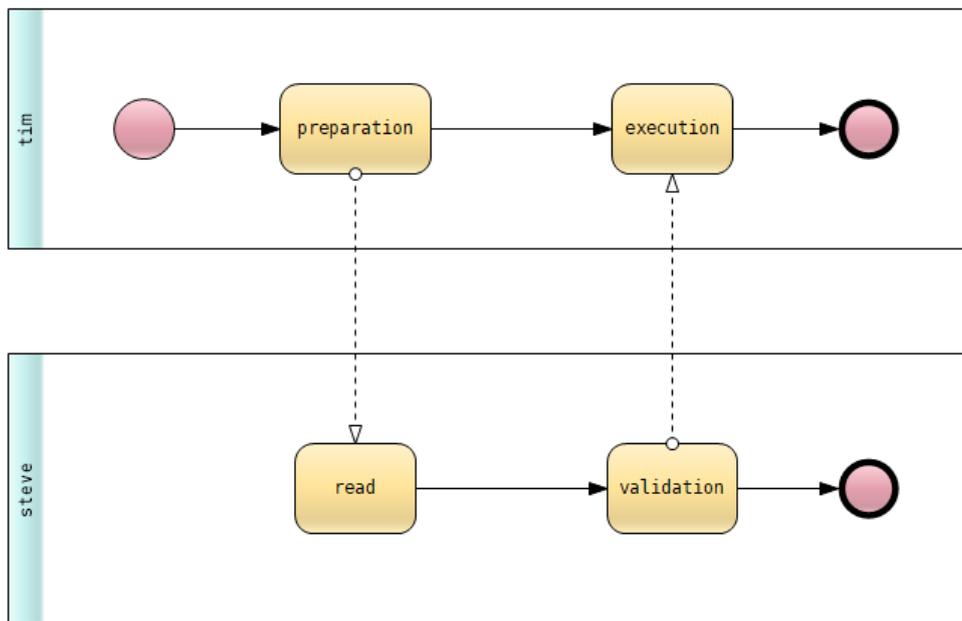
- **Implicit**  
Both sender and receiver are flow elements (e.g., task, event, etc.). The Executor automatically identifies these flow elements as message flow endpoints.
- **Explicit**  
Either the sender or the receiver is a *black-box* participant (i.e., an empty pool with no flow elements). The Executor expects user action for identifying the endpoints.

### 5.3.3.1 Implicit resolution

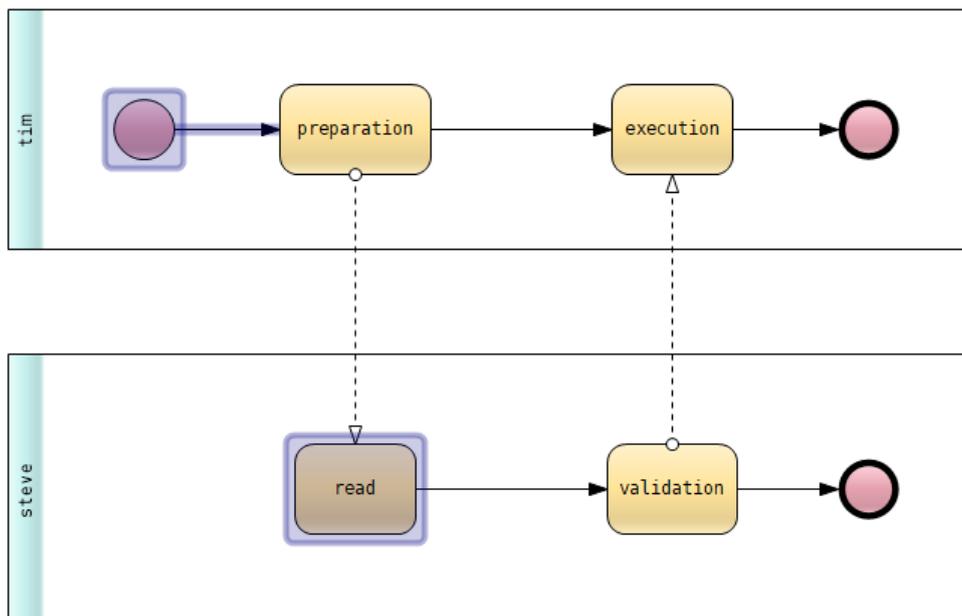
The general message flow semantic is as follows:

- All outgoing message flows must be sent before execution can move to the outgoing flows, i.e., all outgoing sequence flows become *Active* when *Enabling* actions have been issued on all outgoing message flows.
- All incoming message flows must be present (*Ready*) before executing the receiver of said message flows, i.e., outgoing message or sequence flows become *Active* when *Enabling* actions have been issued on all incoming message flows.

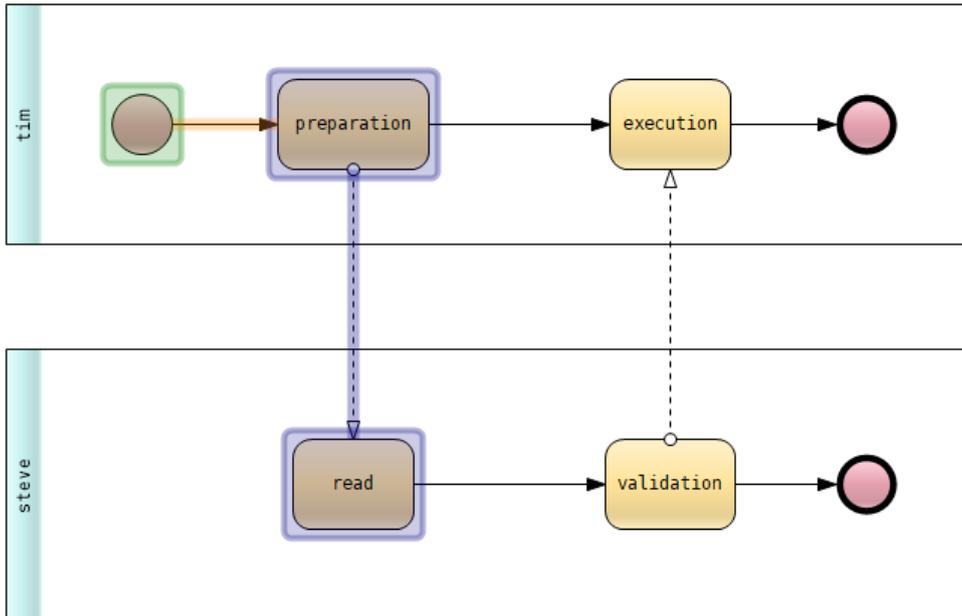
To illustrate this let's consider the following example:



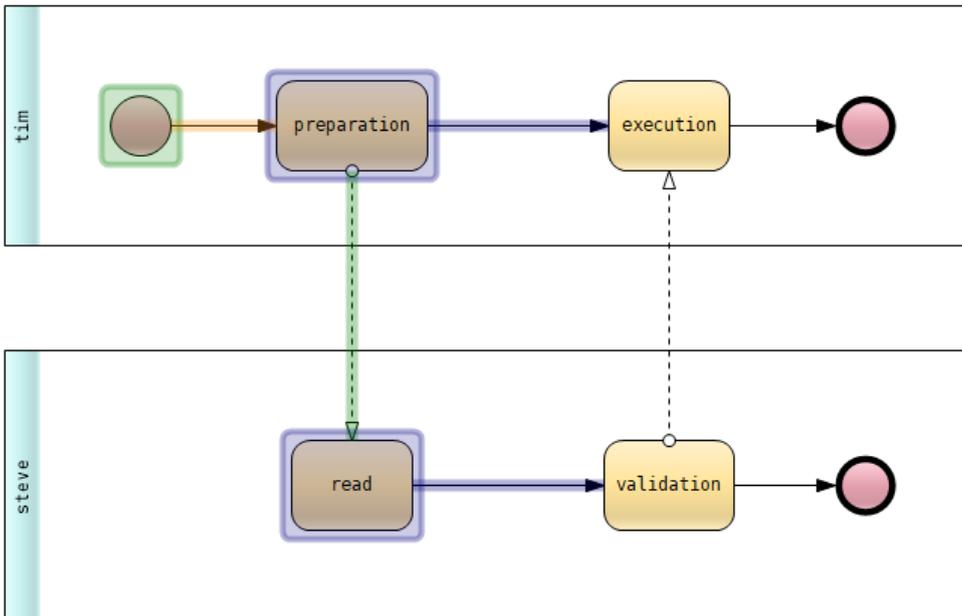
When executing the first possible action is to go from the *Start* symbol to the *Task*:



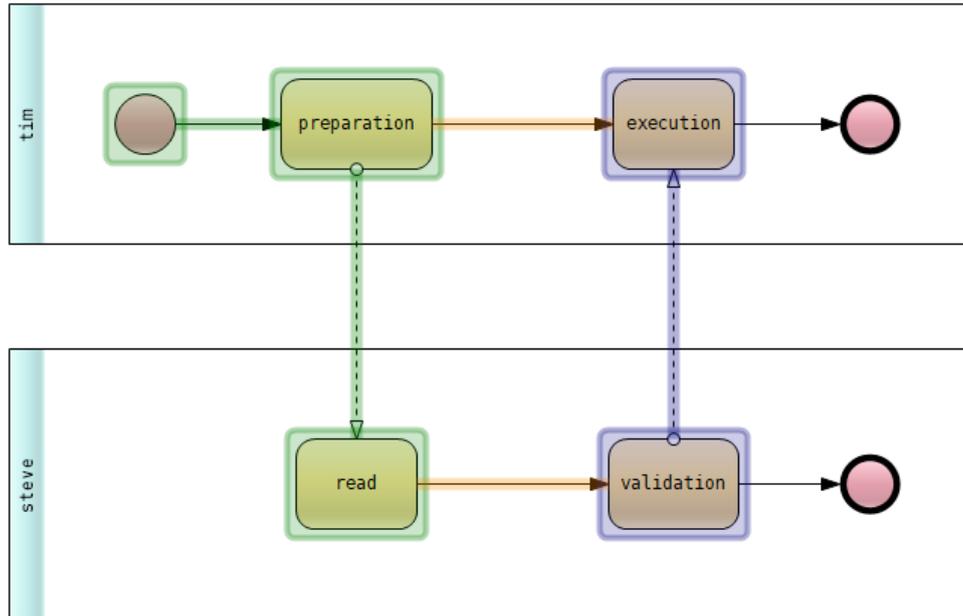
Then the only possible action is to send the message from tim to steve:



Once the message has been sent tim and steve can move on to the next execution step:



If tim and steve execute a single step, again the only possible action is to send the message from steve to tim:



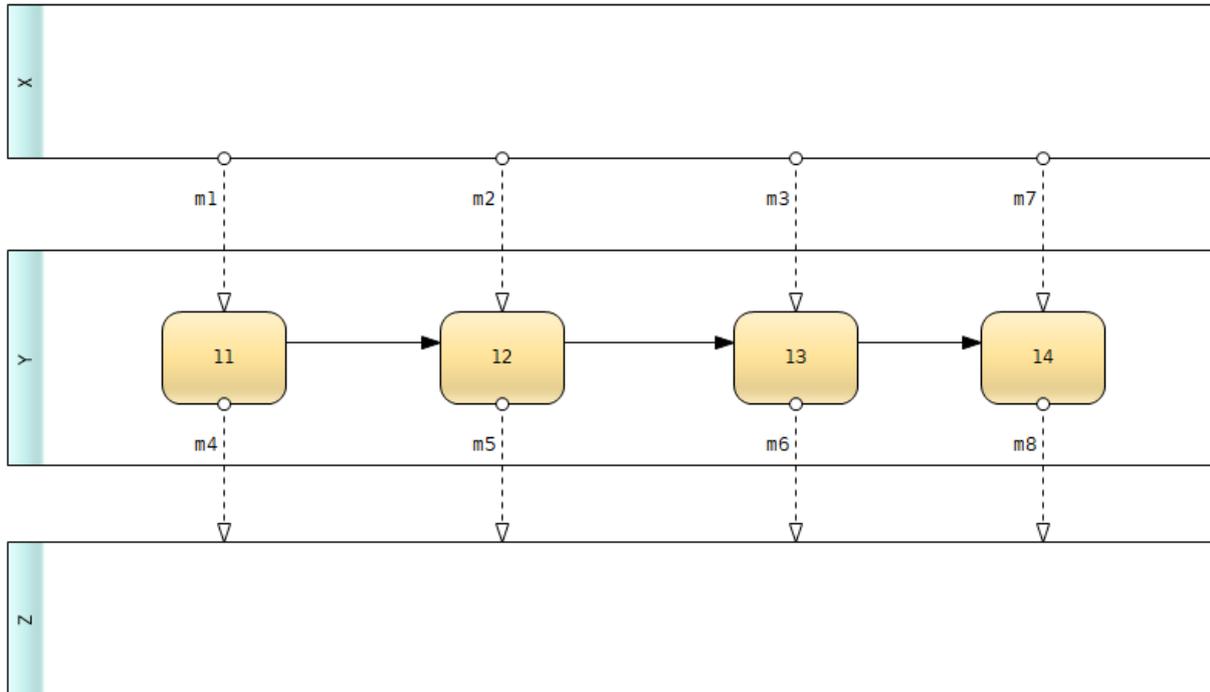
### 5.3.3.2 Explicit resolution

Large models are described in several diagrams written by different modelers. In these situations the pools that are defined by the other modelers are represented by empty pools. We call these empty pools *black-boxes*. During execution these black-boxes might be defined or not, and even if they are defined one might not necessarily want to execute them. The Executor provides a way to handle all these situations with the concept of *gates*.<sup>1</sup> A gate is a small rectangle on the border of the pool where the message endpoint is found. This means that the message flow goes through the gate. At startup, for each gate, the Executor will look for a process that handles the same message flow from and to the same pools. If it finds such a definition, the gates can be enabled or disabled by the user (hence explicit resolution). If enabled it will link both processes, and if disabled it will act like the black-box is undefined. When the black box is undefined, it is possible to send as many messages as wanted. An undefined black-box will always receive all messages sent to it.

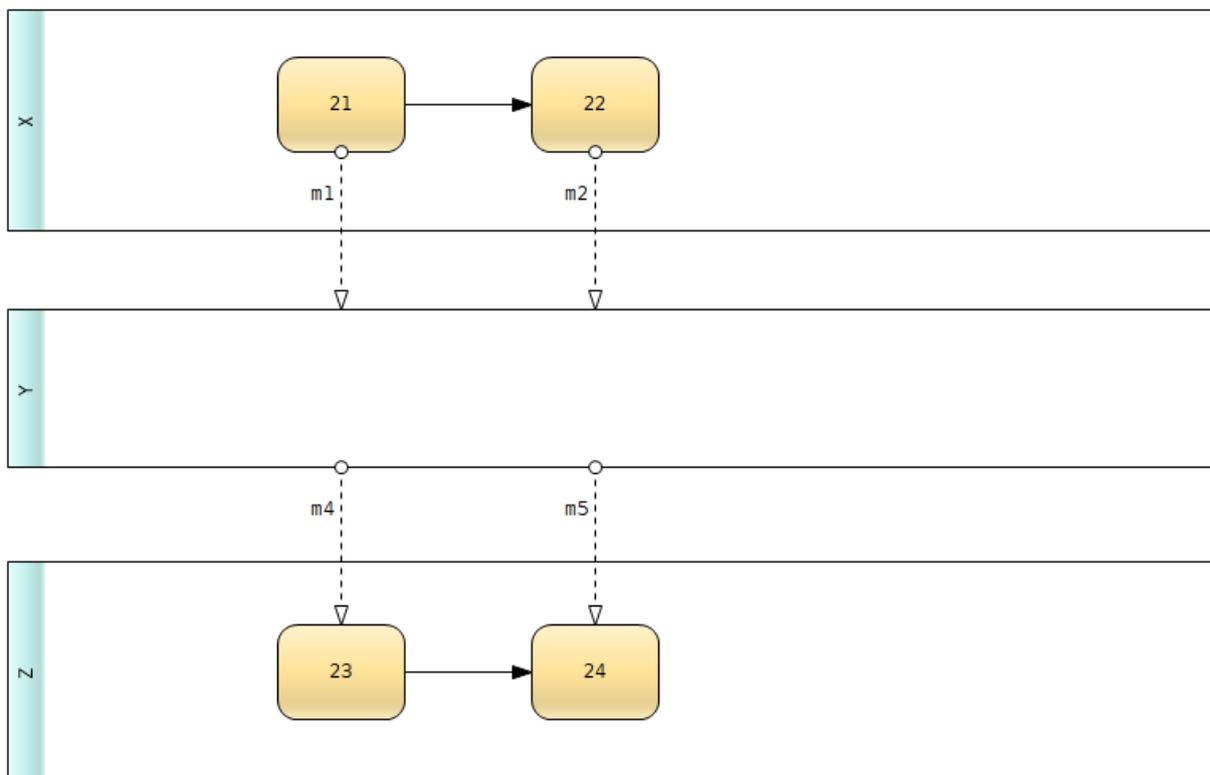
These concepts are illustrated by the following example with three diagrams. The first diagram describes a process in pool Y with messages exchanged with pool X and pool Z.

<sup>1</sup>The gate is not a BPMN symbol; it is a hidden symbol that may become visible only during execution to support explicit message flow resolution.

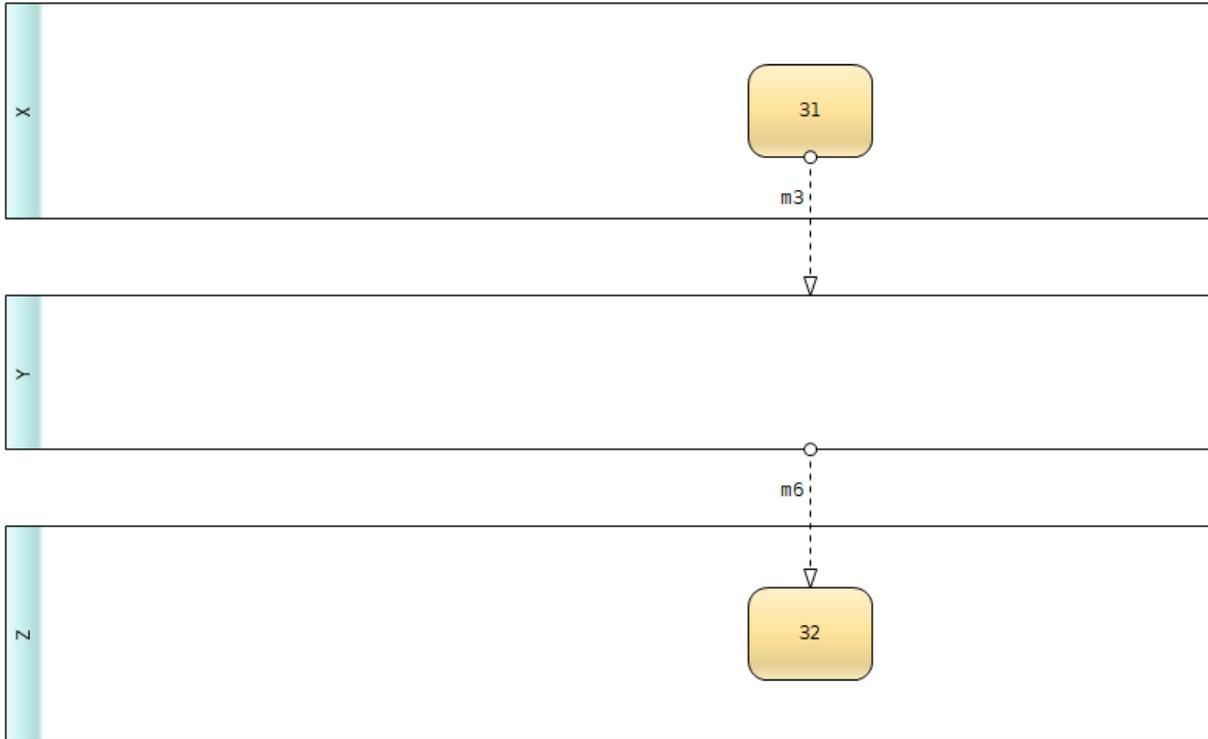
**User manual**



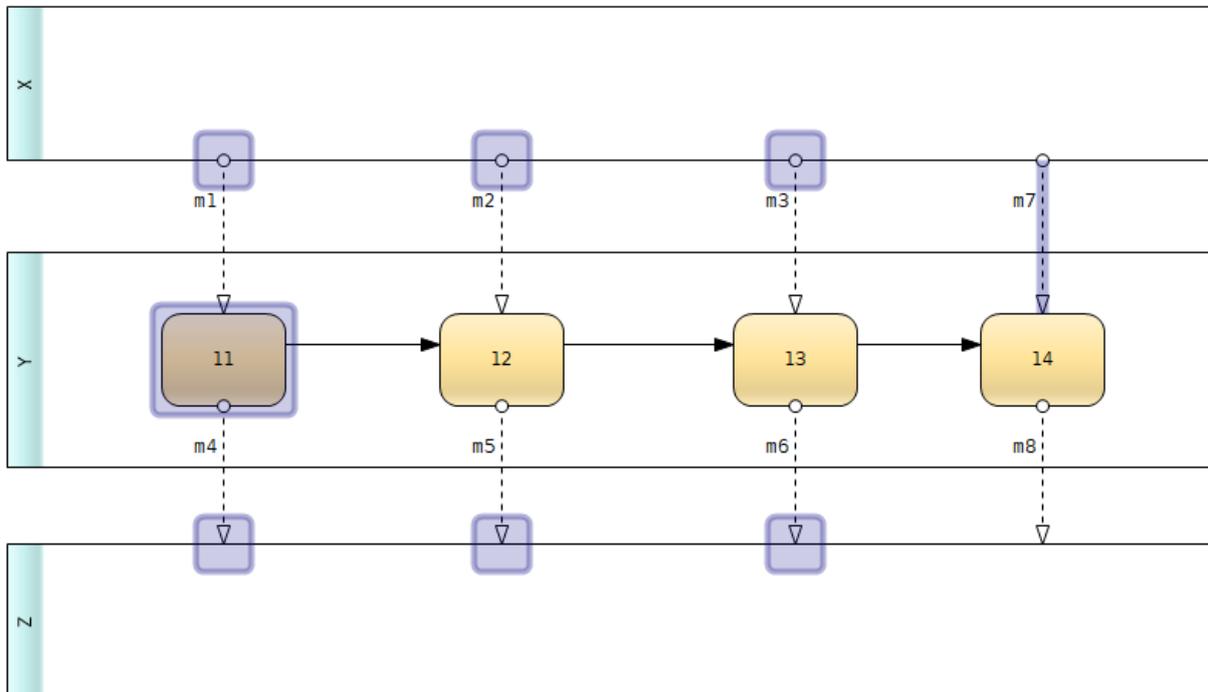
The m1, m2, m4, and m5 interactions between X and Y, and X and Z are described in a second diagram:



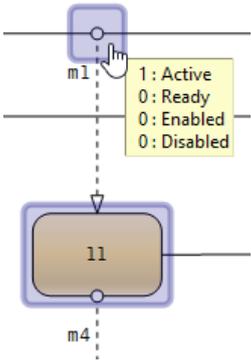
The m3 and m6 interactions are described in a third diagram:



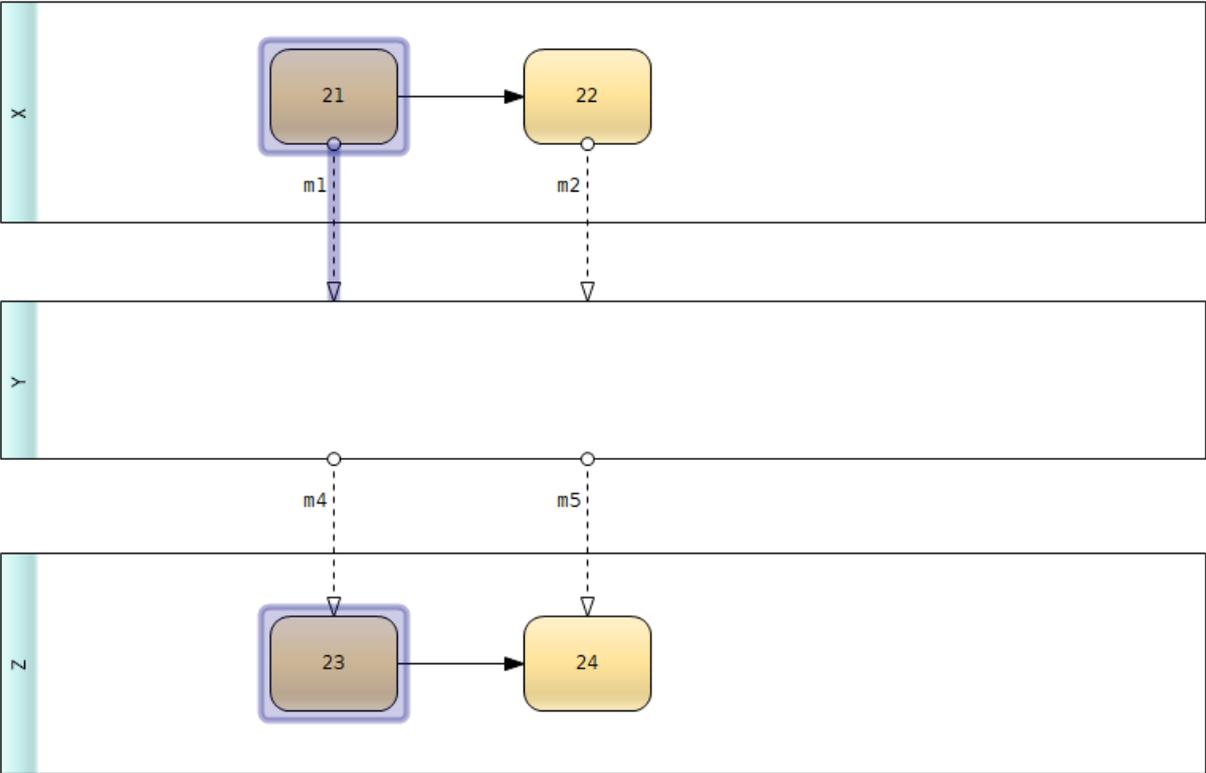
When starting the Executor the links between the three diagrams are analyzed. The m1, m2, m3, m4, m5, and m6 related gates are *Active* meaning a diagram definition has been found for these message flows. It is possible to either enable or disable them.



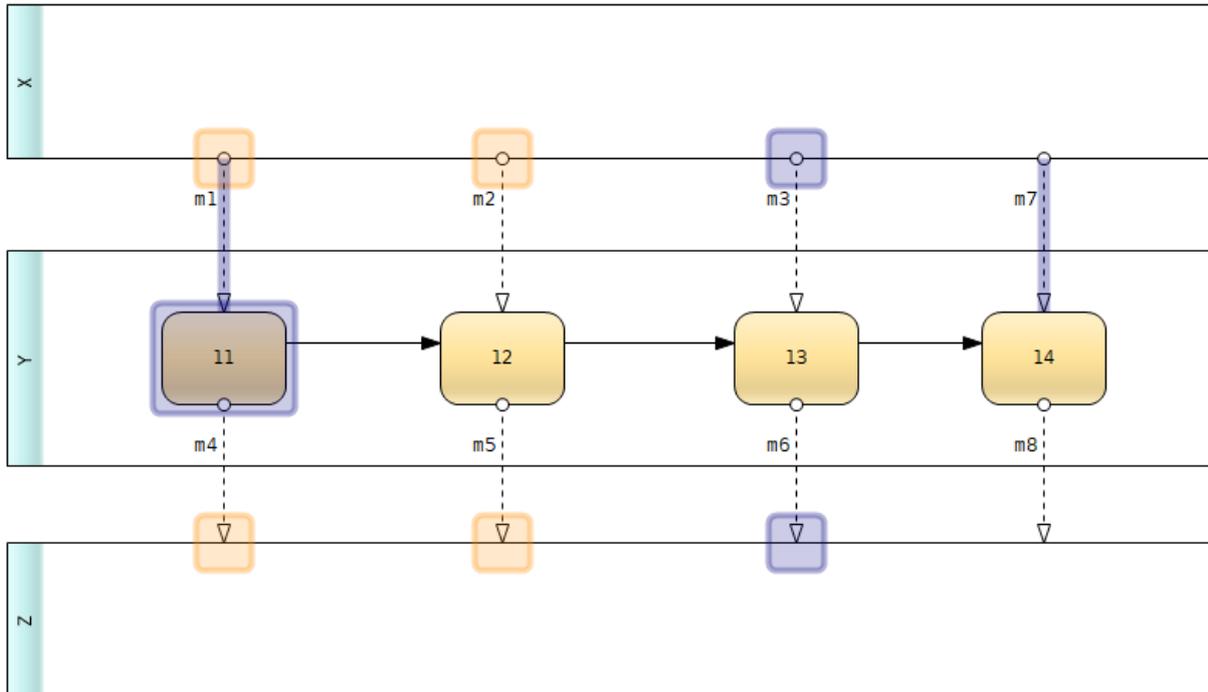
If we enable the m1 related gate with a simple click on it:



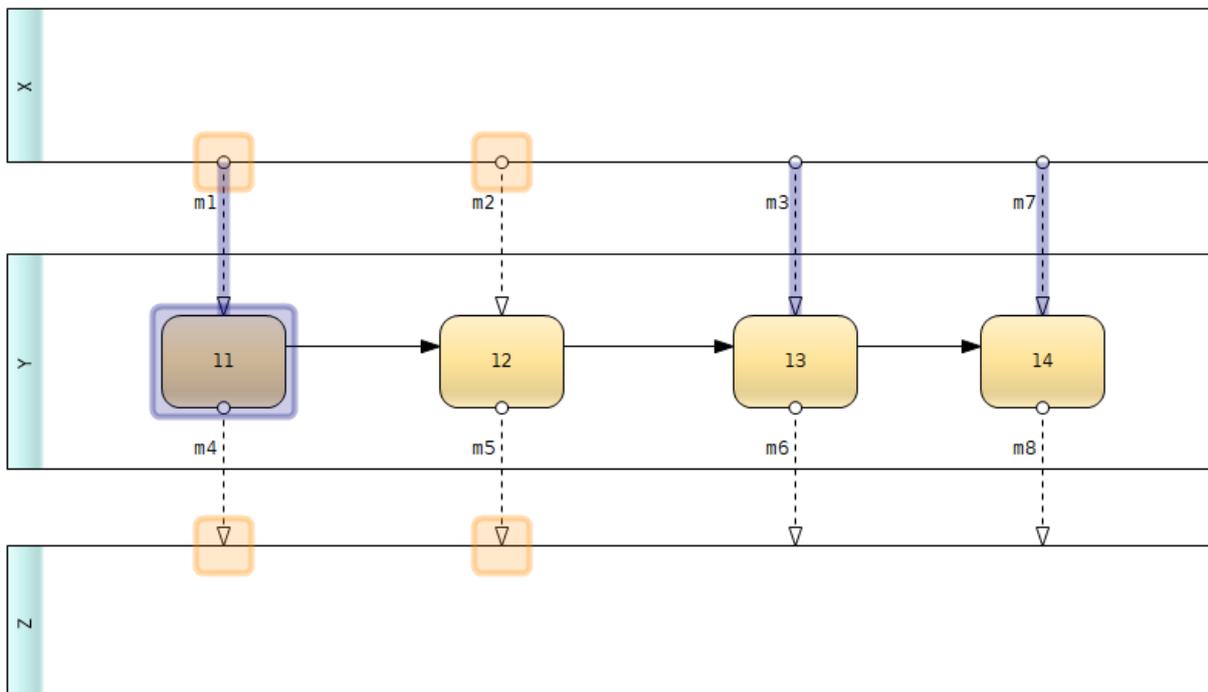
It will automatically include the related diagram in the execution:



The first diagram will display the following:



If we disable the m3 related gate with a right click on it, the related diagram (the third one) is then ignored and pool X is considered a black-box for m3. There is no diagram that describes the m7 interaction, so for m7 pool X is also considered a black-box. In practice that means the message flows can be always enabled in the Executor:



### 5.3.4 Call activities

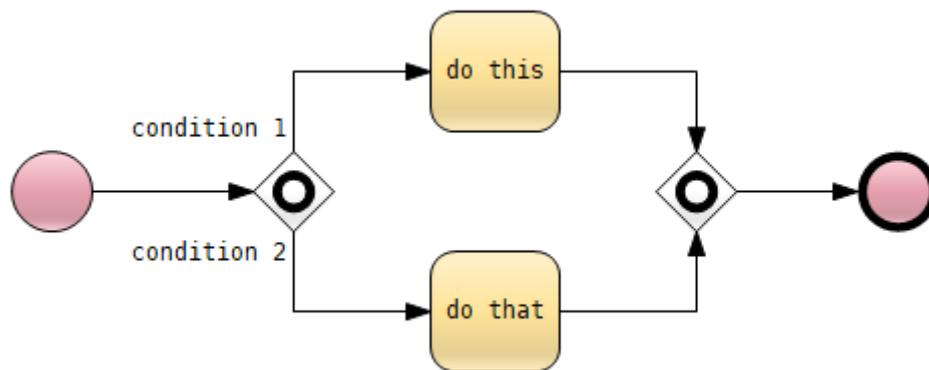
Process call-activities accept enabling or disabling actions during execution. Enabling a process call-activity means including the referenced process' diagram (if it exists) in the execution; this can be seen as a *step-into* the call-activity. Disabling a process call-activity means ignoring the referenced process and treating the activity as a simple task; this can be seen as a *step-over* the call-activity. So, when a process call-activity becomes Active during execution, a left-click will step-into it, while a right-click will step-over it.

### 5.3.5 Gateways

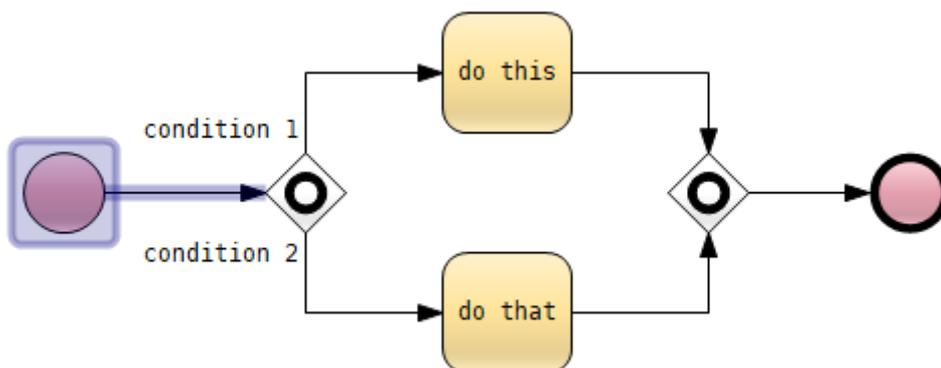
The behavior of the gateways is conform to the standard. As a reminder some typical cases are described in the following paragraphs.

#### 5.3.5.1 Inclusive

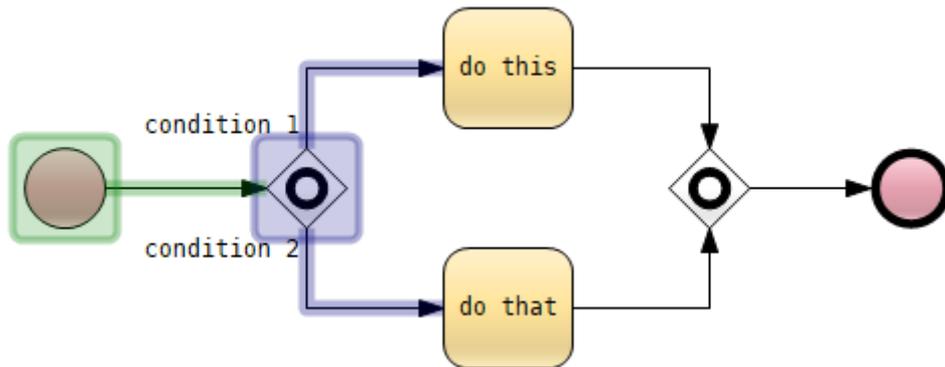
Let's consider a forking and a merging inclusive gateway:



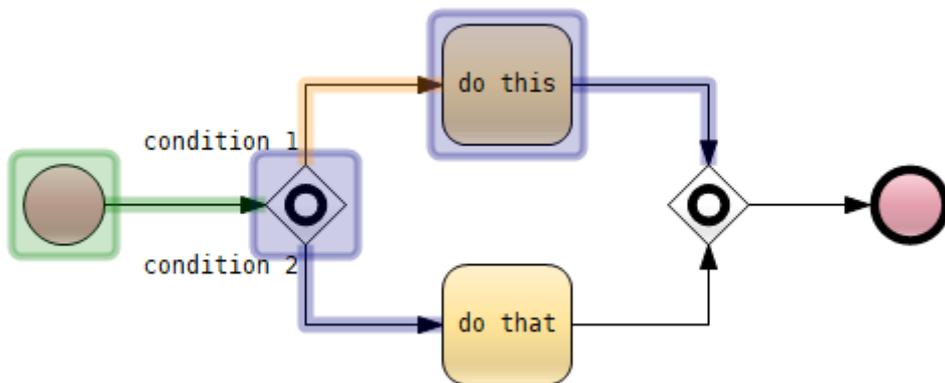
The forking gateway allows to enable or disable condition 1 and condition 2. Not both branches can be disabled because no further action could be done. To enable a branch left click on it, to disable a branch right click on it. Let's execute our example, in the first step the start sequence flow can be enabled:



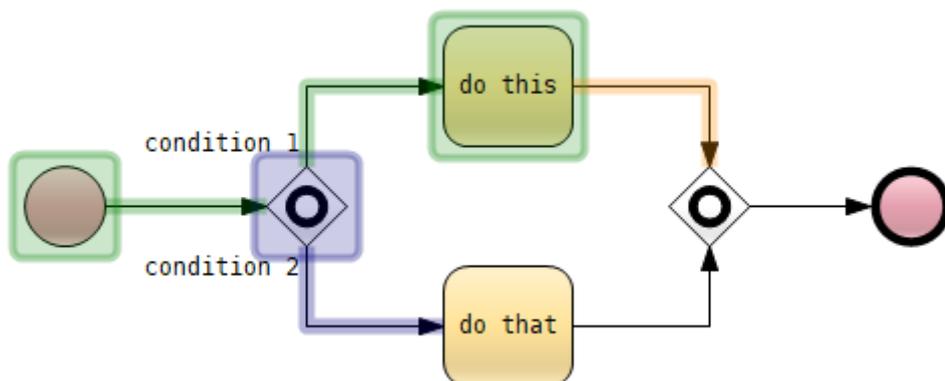
Then it is possible to enable any of the branches:



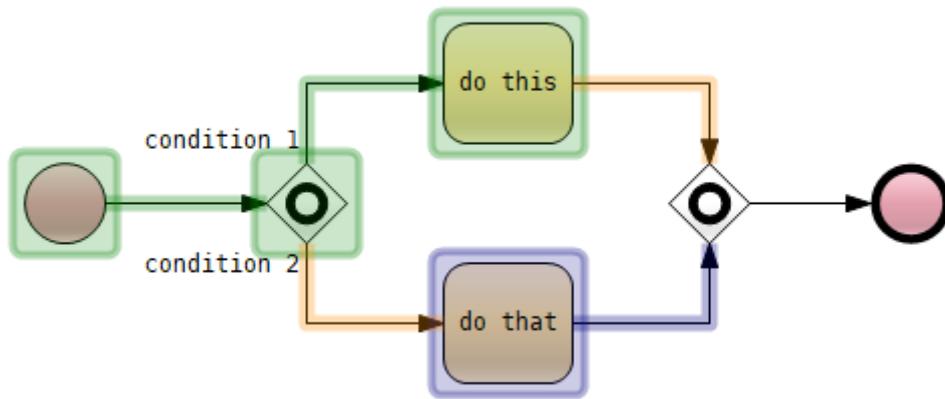
Let's enable condition 1 branch:



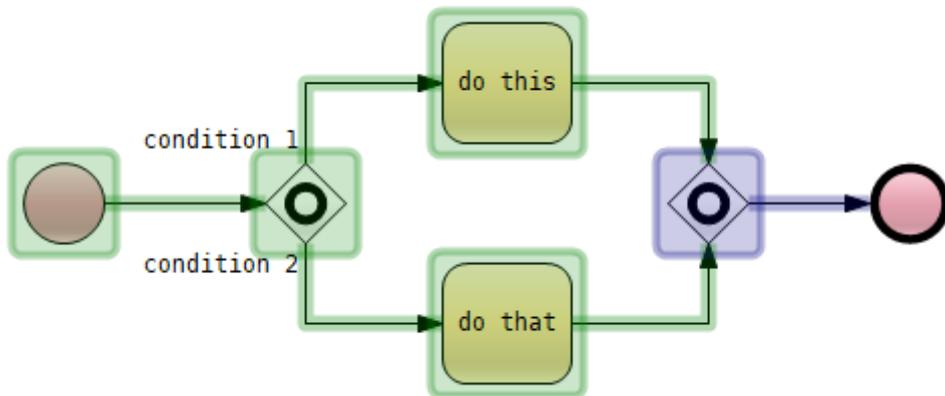
and the sequence flow after do this:



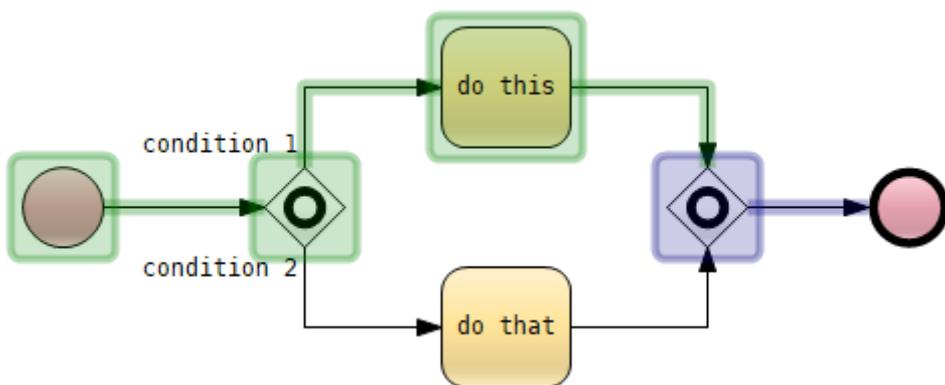
The merging inclusive gateway is waiting for the other to activate its outgoing sequence flow. It is either possible to enable condition 2 with a left click:



and the outgoing flow will be activated:

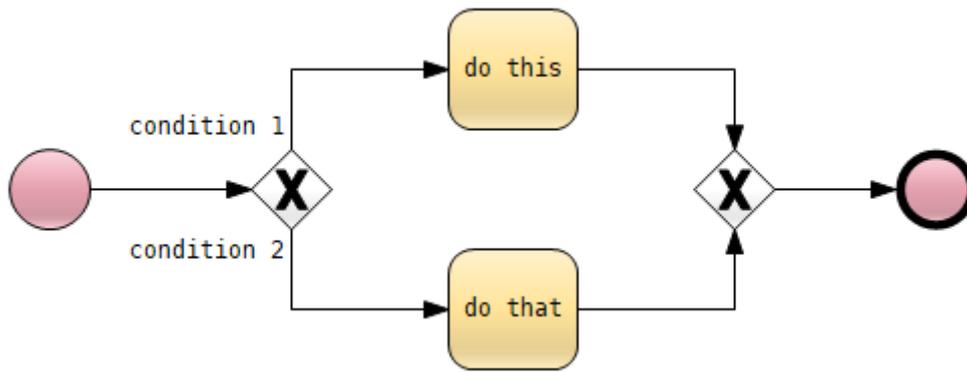


or it is possible to disable condition 2 with a right click, and the outgoing flow will be activated:

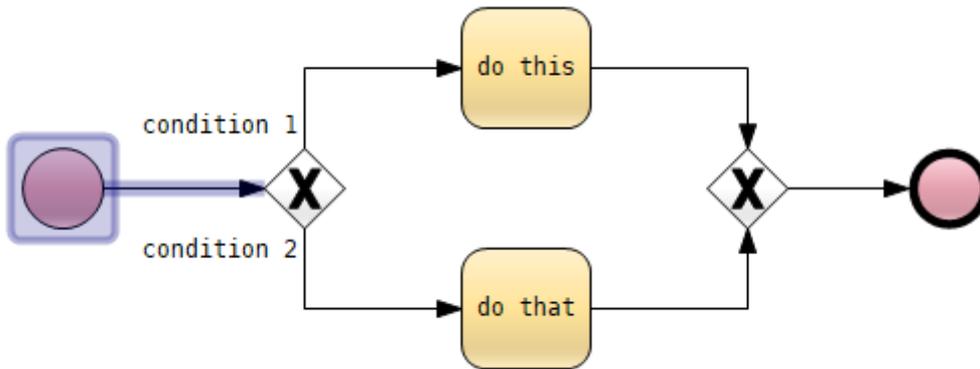


### 5.3.5.2 Exclusive

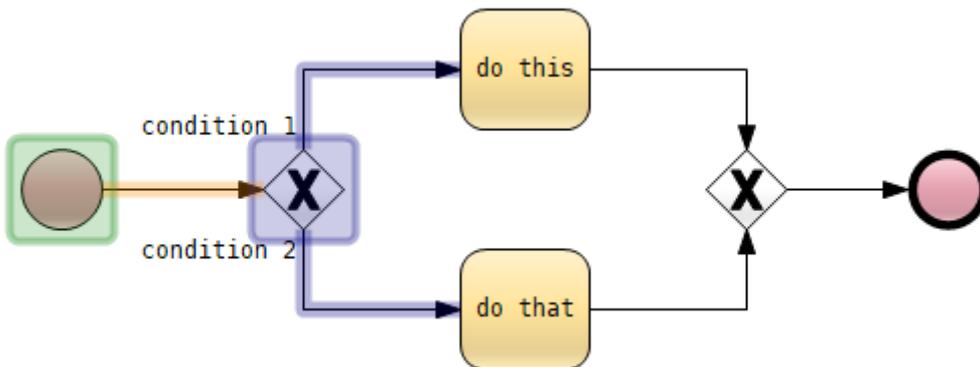
Let's consider a forking and merging exclusive gateway:



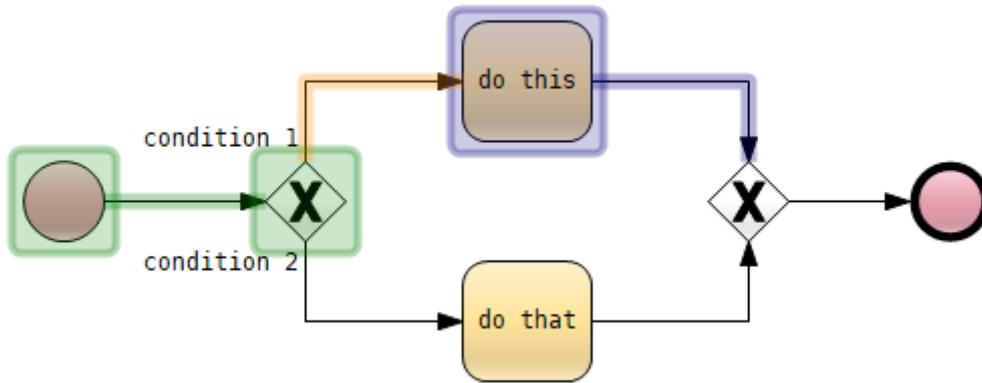
Execution of this simple diagram will first propose the sequence flow following the start event:



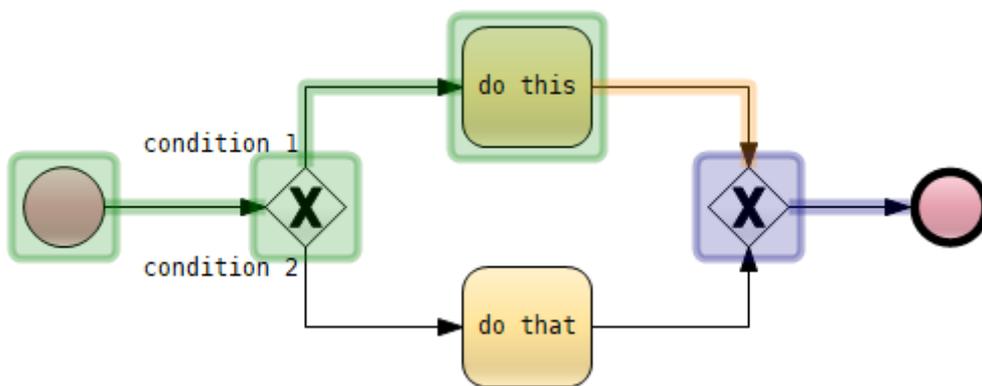
Getting to the gateway will propose any of the outgoing branches:



When enabling one of the branches, the other ones get disabled:

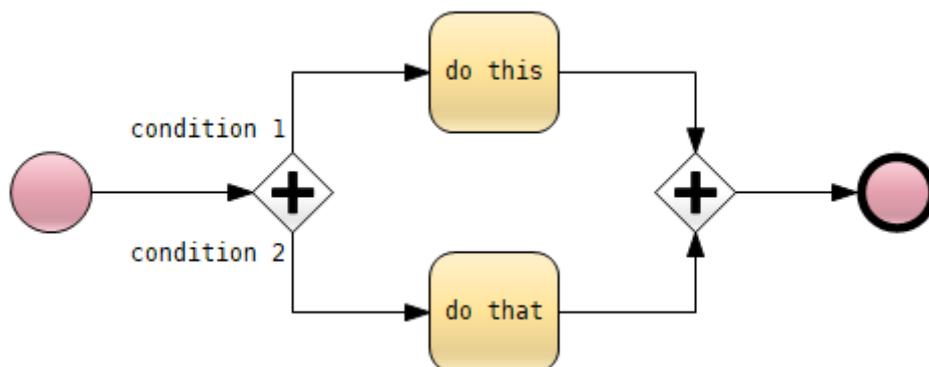


The merging gateway does not expect any other active branches to let the flow go through:

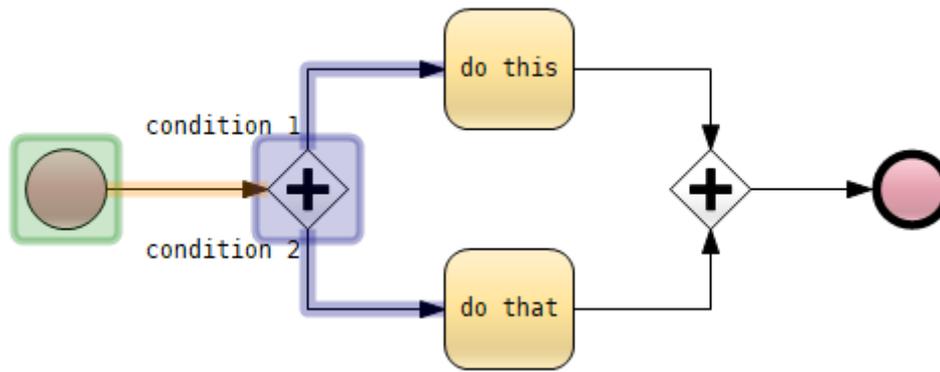
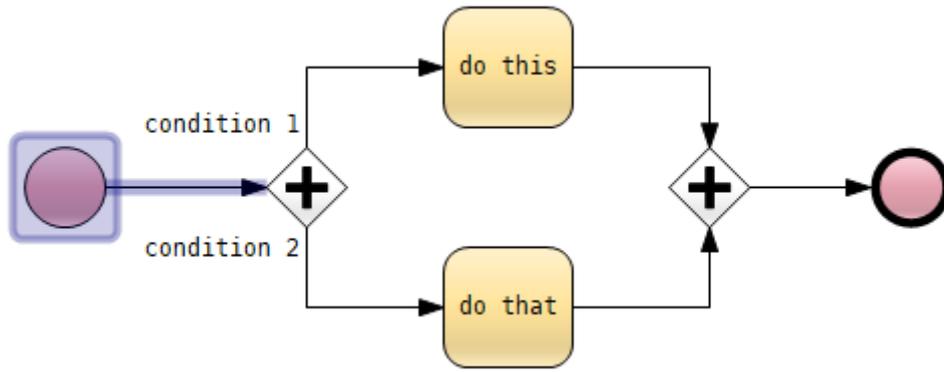


### 5.3.5.3 Parallel

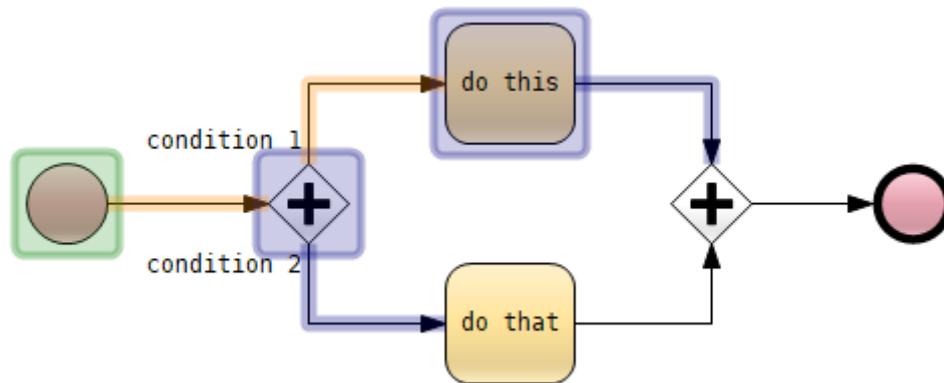
Let's take a simple parallel forking and merging gateway:

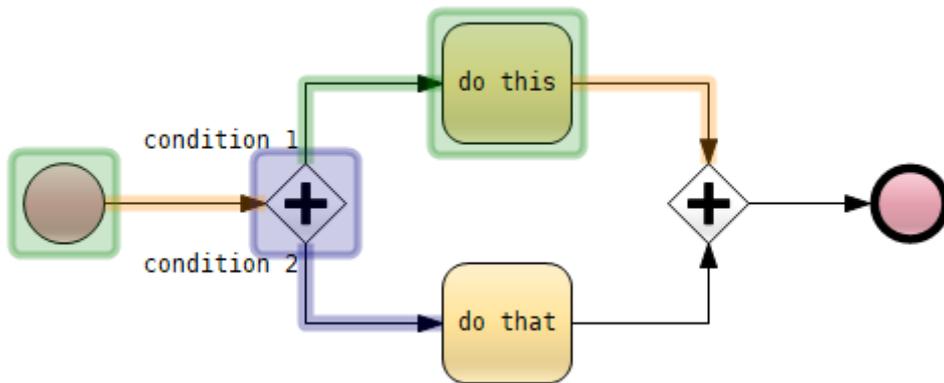


Let's proceed until the forking gateway will activate both branches:

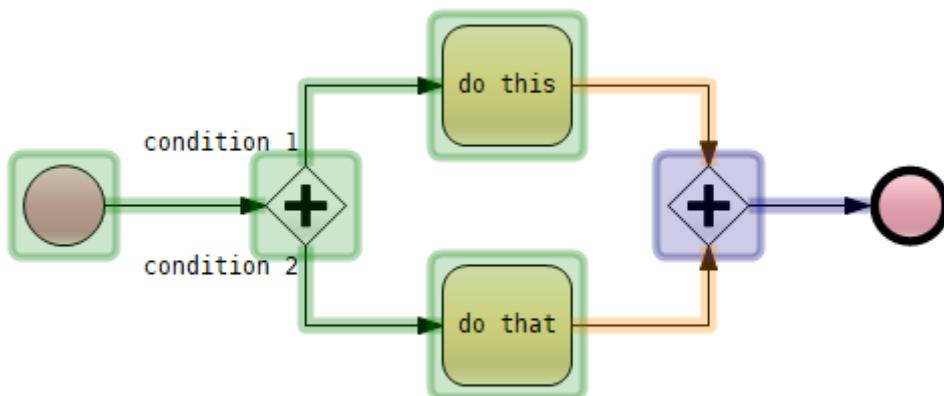
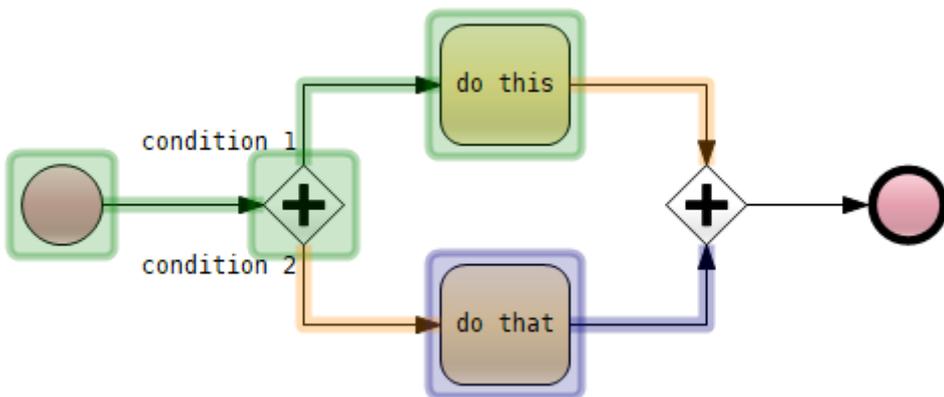


There is no way to disable a branch, both paths have to be enabled for the merging gateway to allow further execution:



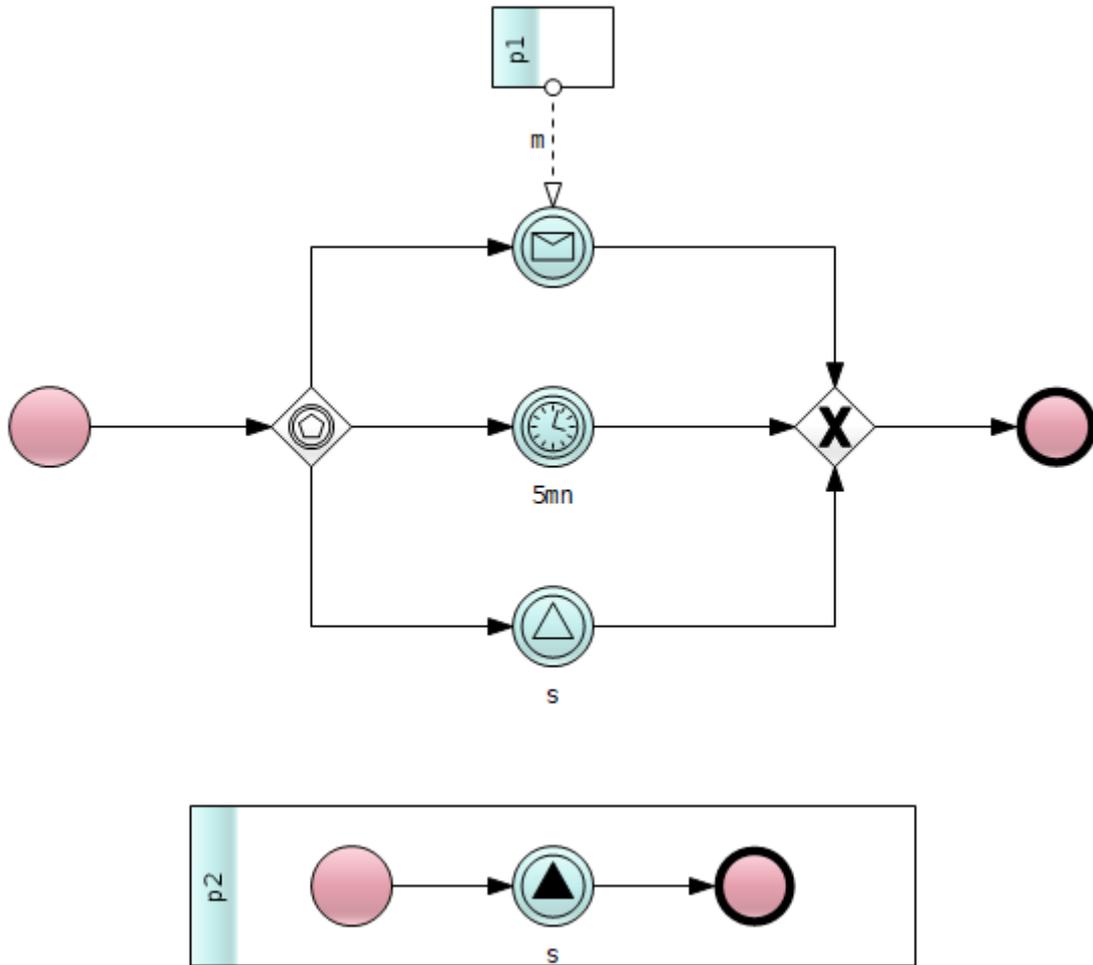


The upper branch has been enabled. The merging gateway is waiting for the lower branch to allow further execution:

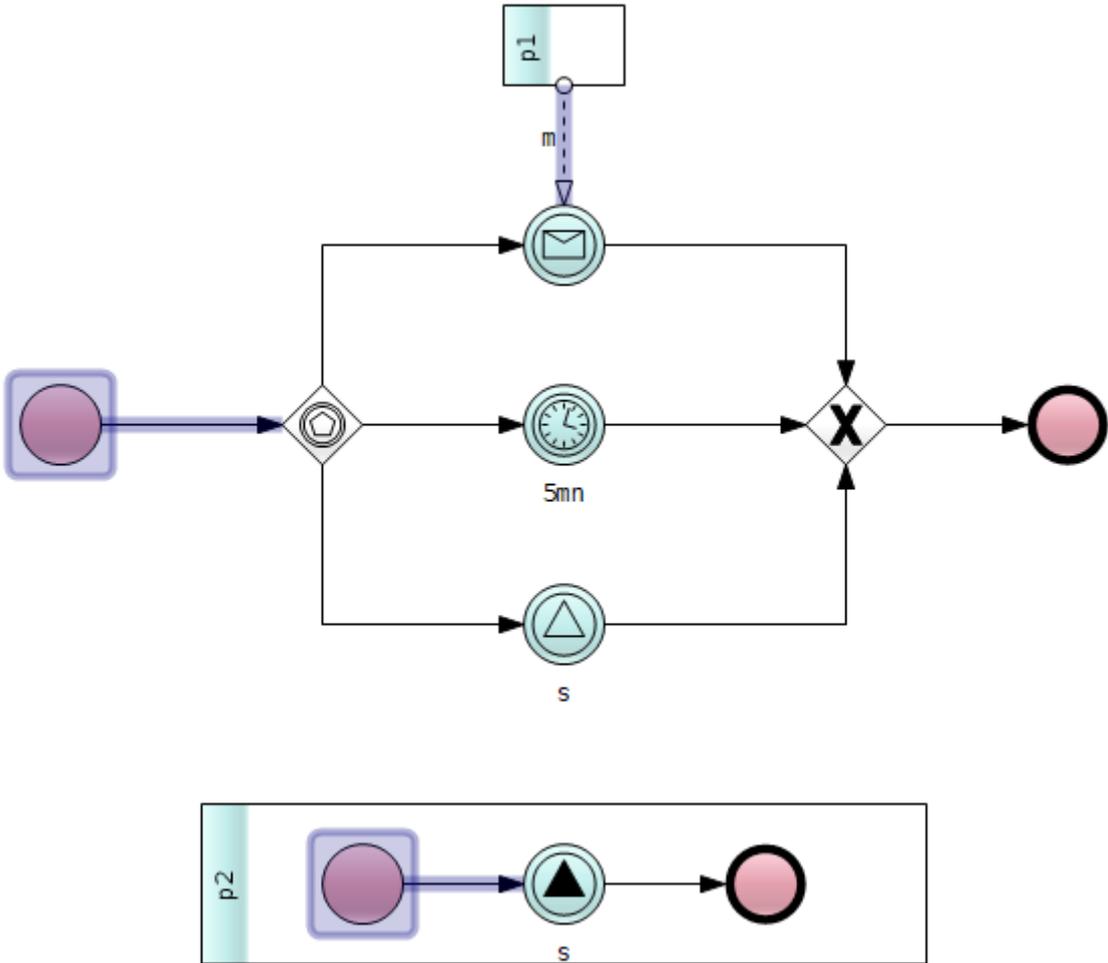


### 5.3.5.4 Event

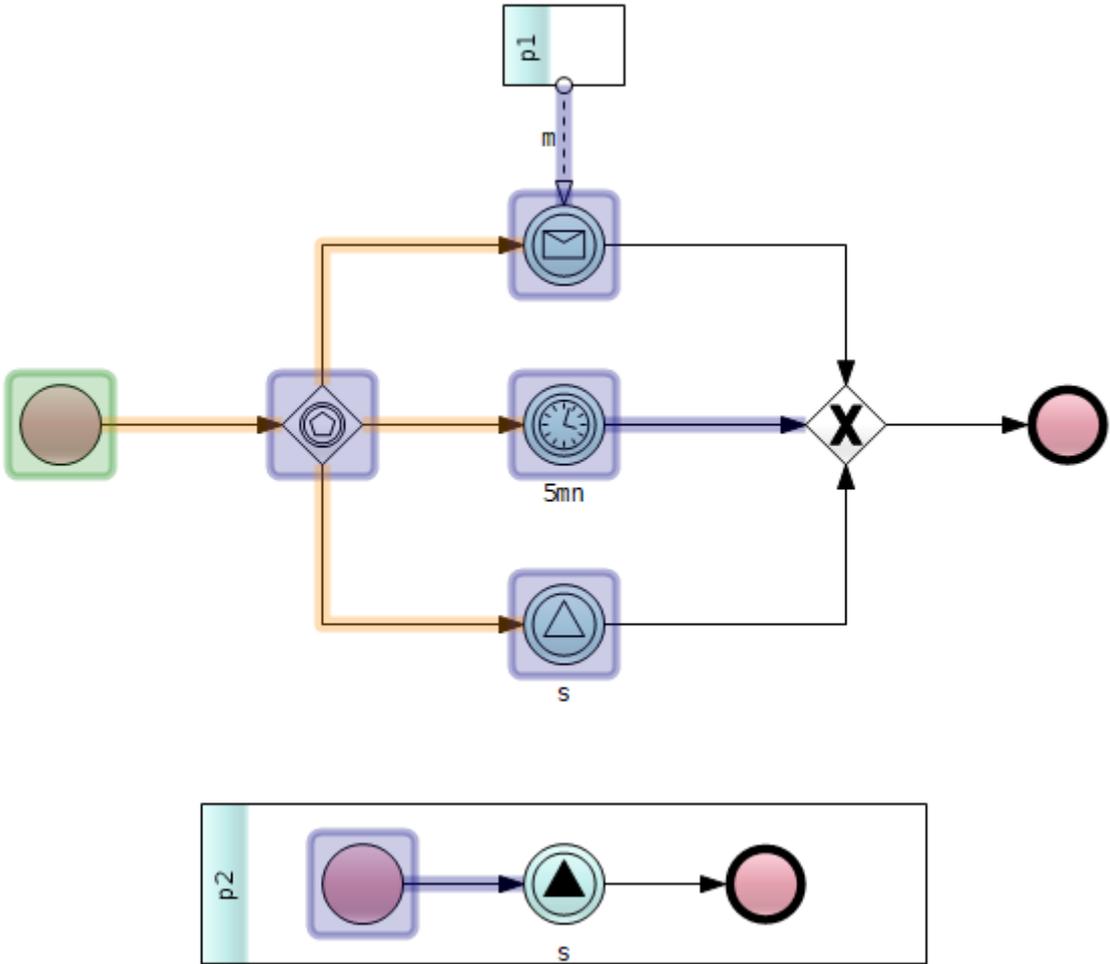
Let's take the following simple example:



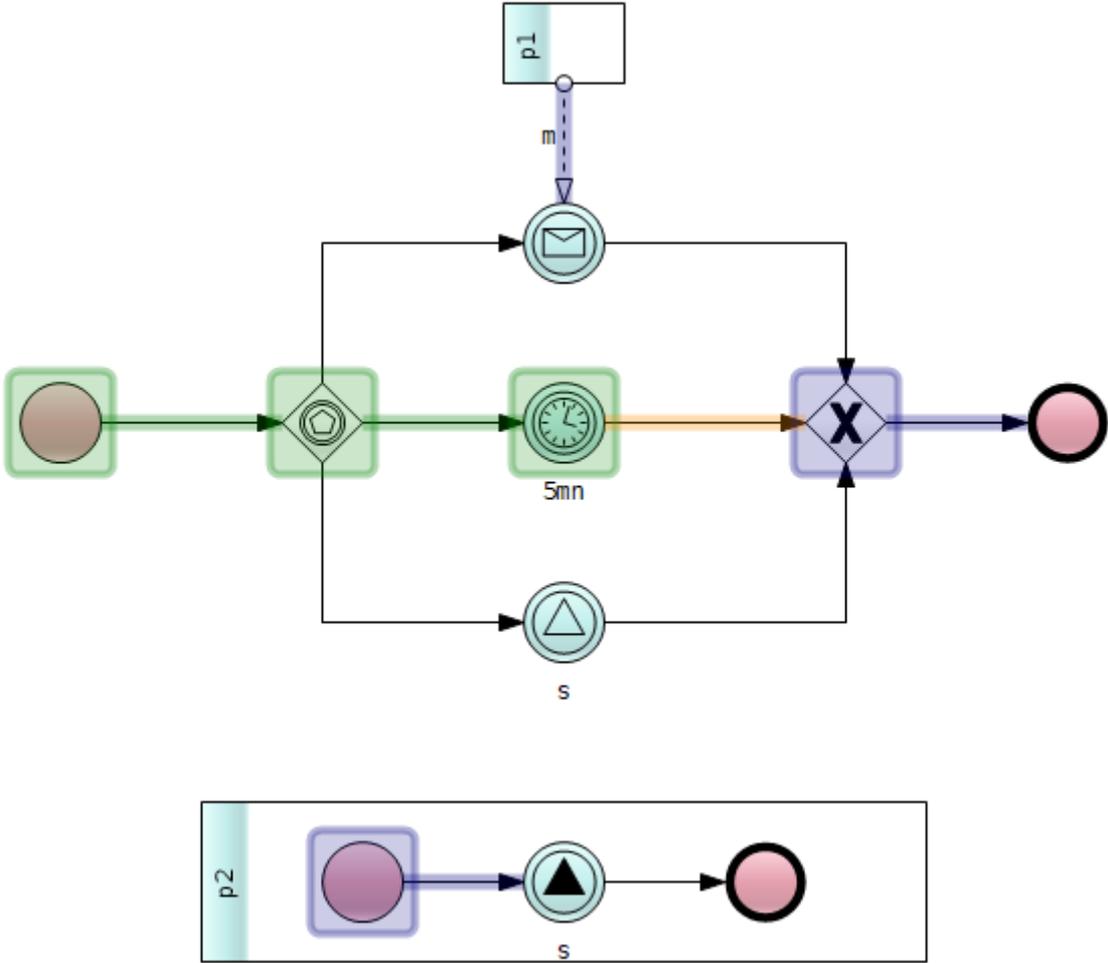
The event gateway can be triggered by the *m* message coming from pool *p1*, or a time out from the 5mn timer, or the *s* event that could be thrown by the process in pool *p2*. Let's start execution:



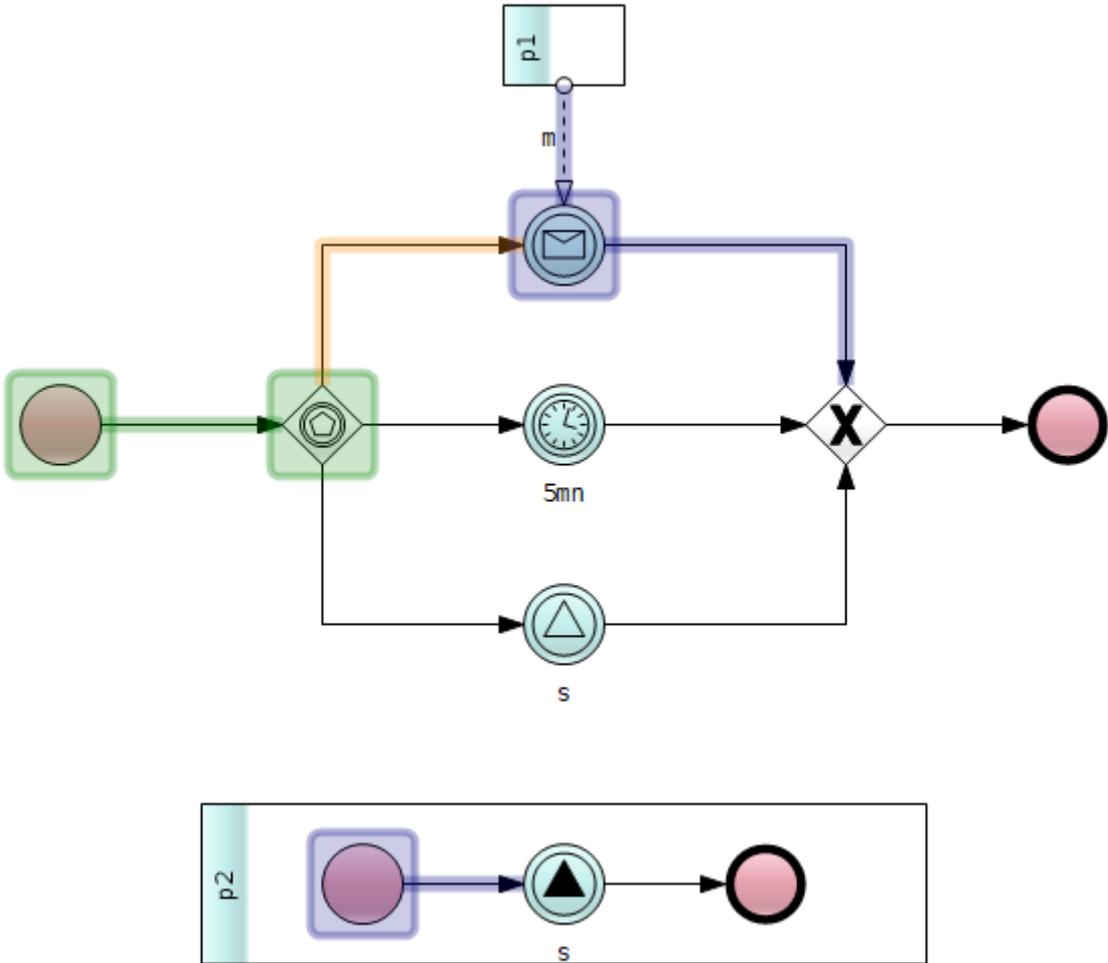
And enable the sequence flow after the start event:



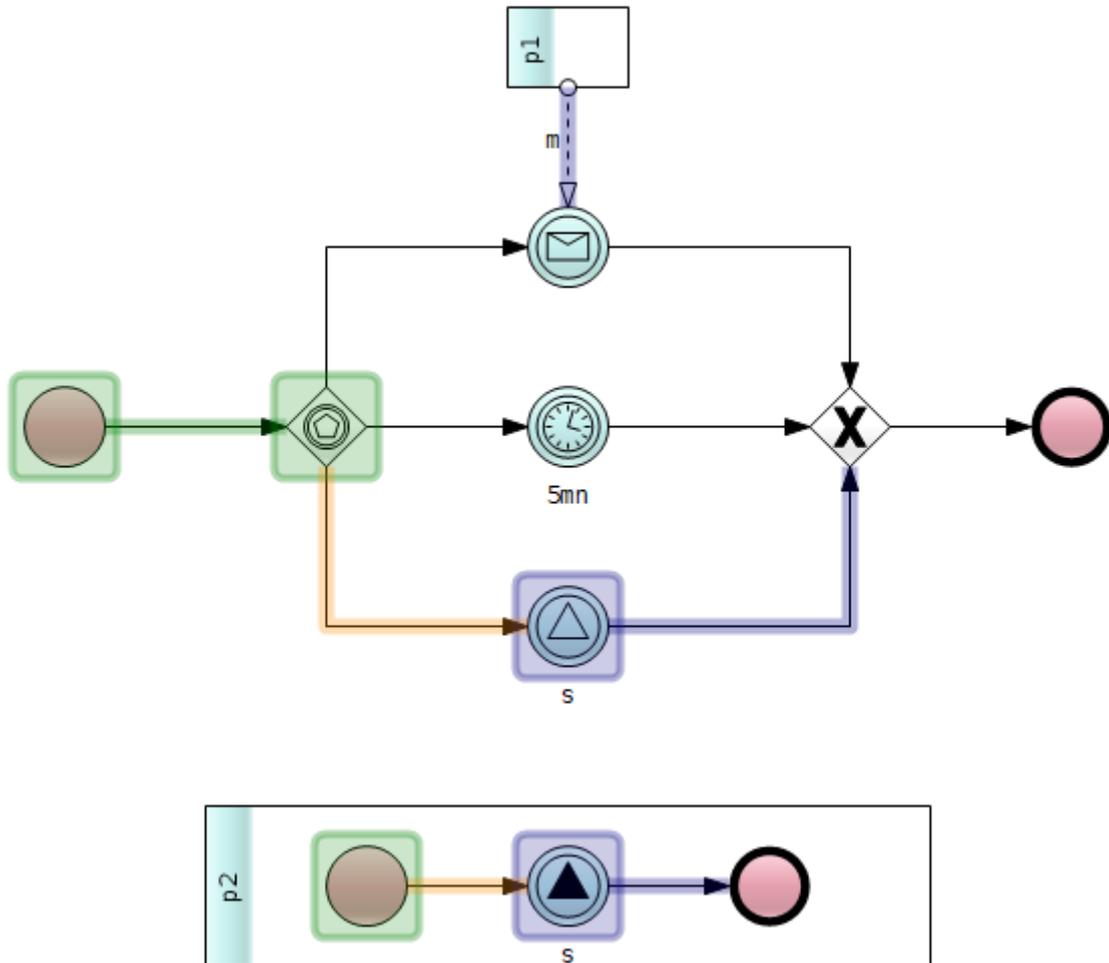
A click on the sequence after the timer will activate the sequence after the exclusive gateway. Note that all other branches are disabled:



A click on the m message flow will activate the upper sequence flow and disable the others:



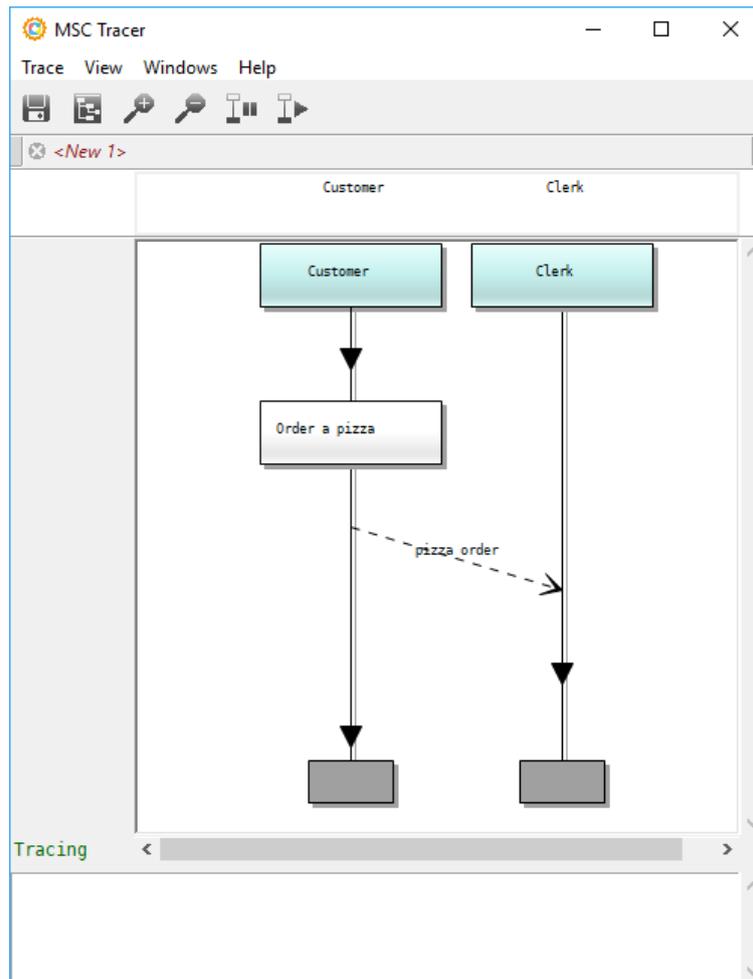
A click on the start sequence after the start event in pool p2 to throw the event that will be caught by the lower branch of the gateway. Again the other branches will be disabled:



## 5.4 Execution traces

### 5.4.1 Recording

An execution scenario can be recorded with the *Record* button . Clicking the button will open the *MSC Tracer* and record the execution events, e.g.:



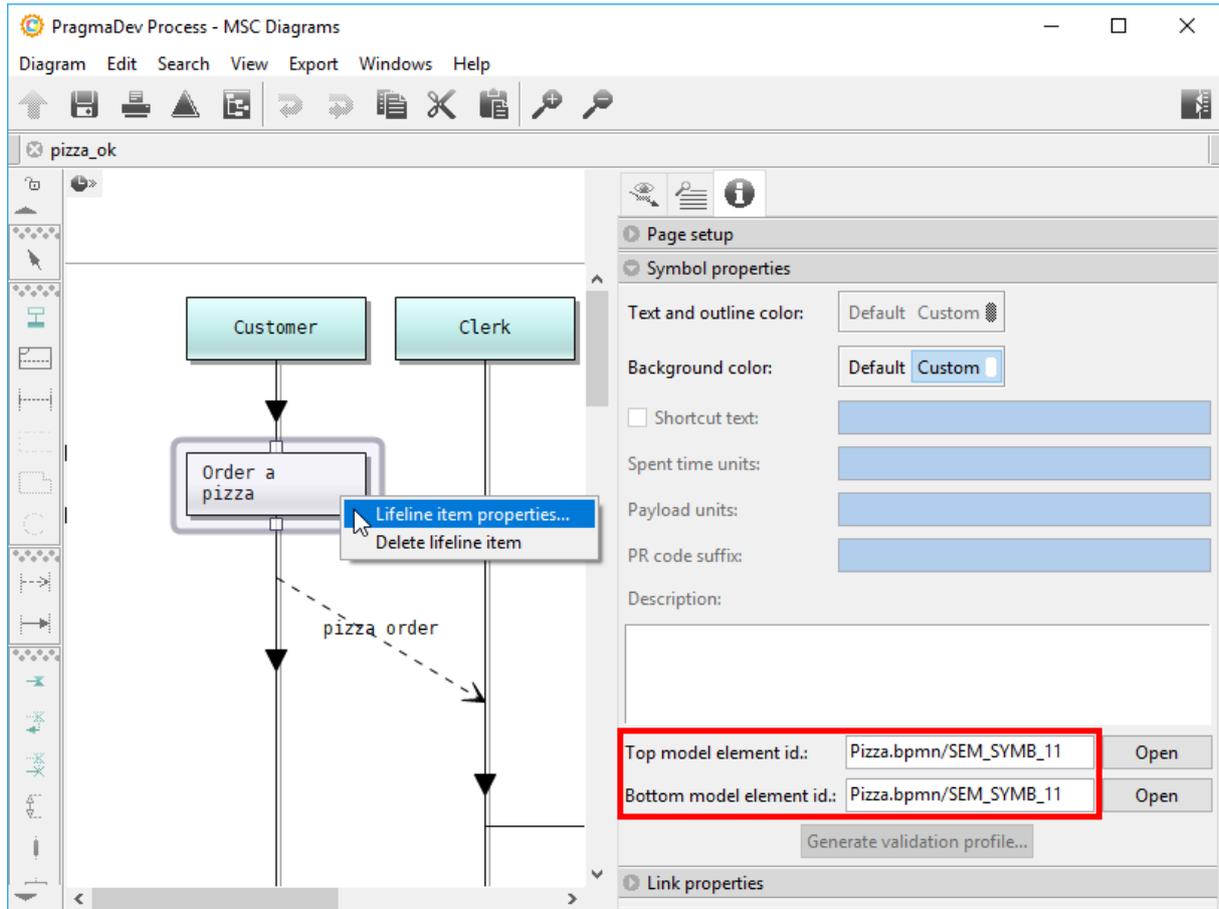
The description of the events that can be traced and the BPMN element (and execution action) they represent is given in MSC & PSC reference guide.

## 5.4.2 Replay

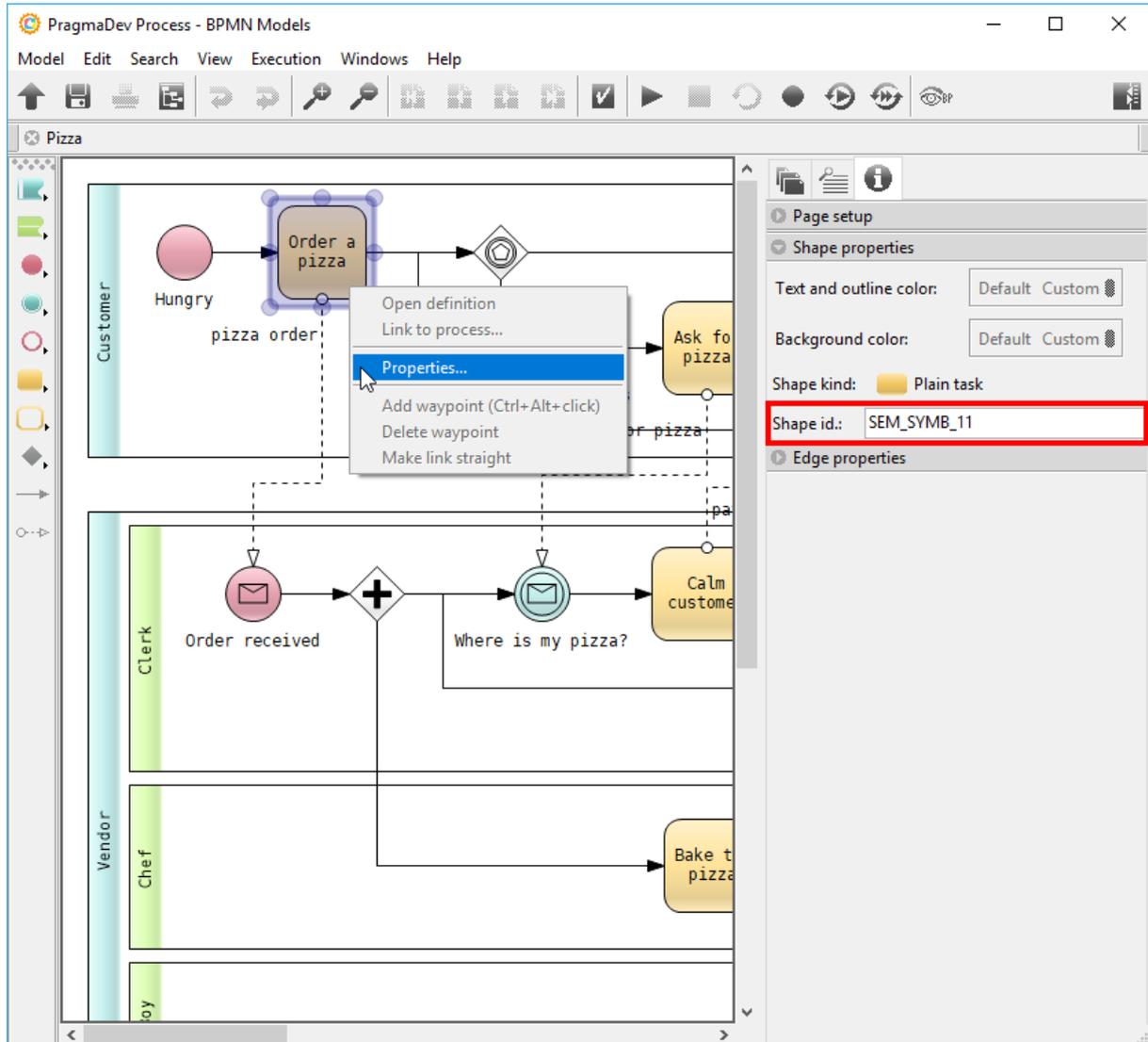
A recorded execution trace can be saved for replay. Replay means executing a trace against the model, while checking for differences between the two at every step. There are two checks done at every step:

- The referenced model ID of the current element in the MSC trace is checked against the ID of the current element in the BPMN model.
- The text of both current elements (MSC and BPMN) are checked to see whether they are the same.

The referenced model ID of an element in a trace can be accessed in the MSC editor by right-clicking the element and *Lifeline item properties* or *Link properties*:



Note that the referenced ID is preceded by <name-of-bpmn-file>/.  
In the BPMN editor the element ID can be accessed in a similar way:



### 5.4.2.1 Single-trace execution

The *Replay* button  in the BPMN editor allows replaying of a single trace. After choosing the trace to replay, the MSC editor will be opened in execution mode. The execution is controlled from the MSC editor with the following tools:



A trace can be executed till its end *Run* button .

A single event of the trace can be executed with the *Step* button .

When running, execution can be paused with the *Pause* button .

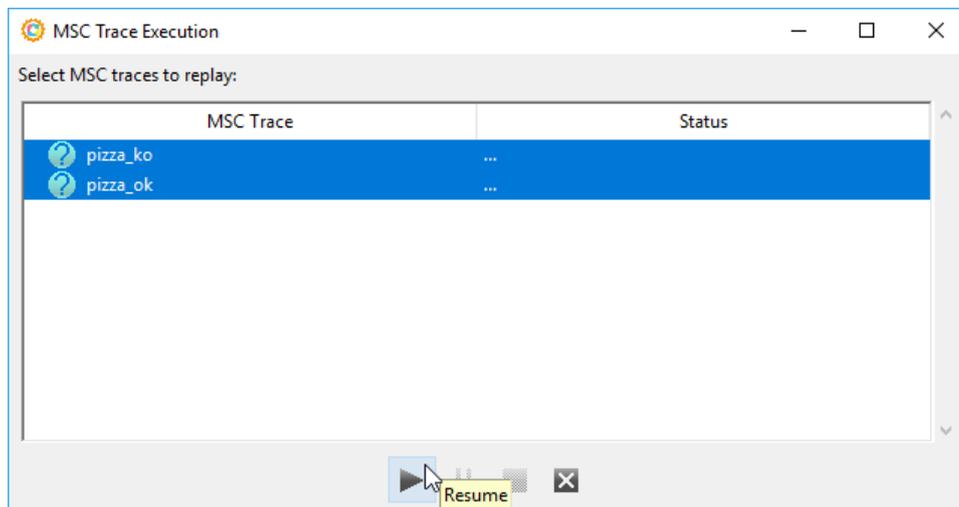
Execution mode can be exited with the *Stop* button .

Execution is reset with the *Reset* button .

When executing a trace either step-by-step or until the end, the state of execution will be updated also in the BPMN editor. The current event in the trace will be checked and compared against the BPMN model at every execution step. The execution will stop either if a difference is found between the trace and the model or if the end of the trace was reached.

#### 5.4.2.2 Multi-trace execution

The *Replay all* button  allows replaying multiple traces. Clicking the button will show following window:



To select several traces, hold down *Ctrl* or *Shift* and click on them.

Left to the trace name is a symbol representing the result of execution. Possible symbols are:

- 🔍 = NONE, i.e., trace has not been executed yet.
- ✅ = PASS, i.e., no differences were found between the trace and the BPMN model.
- ❌ = FAIL, i.e., differences were found between the trace and the BPMN model.
- ❗ = ERROR, i.e., could not execute trace due to errors.

If the result of execution for a given trace is FAIL, then double-clicking the trace will open it and the BPMN model in the editors, and select the symbols that caused the fail.

## 6 MSC and PSC Editor

### 6.1 Overview

PragmaDev Tracer is one of PragmaDev Process modules. PragmaDev Tracer allows to generate execution traces when executing the BPMN model. The traces are based on a variant of the standard ITU-T MSC representation.

Online traces are created directly by the model being executed which sends trace information.

Other features of PragmaDev Process may be used in conjunction with the tracer itself to get a full-featured tracing utility with conformance checking against specification or property verification:

- Trace diagrams can be directly created for documentation purposes;
- Trace diagrams can be compared in a visual way;
- Specification MSC diagrams can be created, then can be compared to execution traces for conformance checking;
- Property diagrams can be created using the Property Sequence Chart (PSC) format, which can be:
  - matched against execution traces;
  - automatically verified with OBP explorer;
- All diagrams can be easily documented, for example by exporting them fully or partially to common image formats, allowing to insert them in a document.

The general graphical form of the diagrams supported by PragmaDev Tracer is described in section MSC & PSC reference guide below.

The tracing feature is described in the Executor chapter. The MSC and PSC editor is described in MSC editor. The available checks that can be performed - trace against trace, specification against trace or property matches - are described in Conformance checking: diagram diff & property match.

### 6.2 MSC & PSC reference guide

#### 6.2.1 General diagram format

A MSC or PSC diagram represents the interaction going on between entities called instances over time. Instances will typically be a participant or a lane. Instances are represented by symbols called lifelines, that look like follows:



The lifeline always starts with a head that specifies the instance name.

All events happening on the instance are then displayed on a vertical line under the lifeline head. These events are described below in “Lifeline components” on page 358.

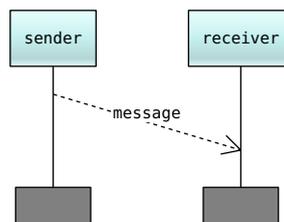
The lifeline terminates by a lifeline tail, that can take several forms depending on the status of the instance at the end of the diagram.

Lifelines are distributed along the horizontal axis, and the vertical axis represents the time, flowing from top to bottom. Events happening between lifelines are mostly represented by links, described in “Links” on page 355. Other symbols allow to further describe the diagram or add semantics to specification MSCs or PSCs; they are described in “Main symbols” on page 366.

## 6.2.2 Links

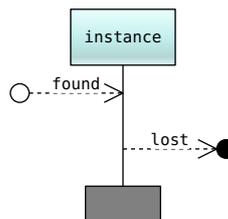
### 6.2.2.1 Message links

An asynchronous message sent by an instance and received by another is represented by a dashed line with an outlined arrow at its end:



Note that a message is a plain line in the genuine MSC format.

A message can also be received from an unknown source, or sent to an unknown target. In this case, they are called a found message and a lost message respectively:



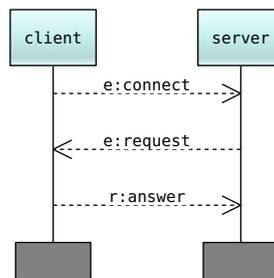
The syntax for the message link text is free.

### 6.2.2.2 PSC-specific normal, required and failed message syntax

In PSC diagrams, texts for message links are supposed to be prefixed with one of the following:

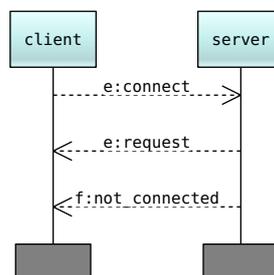
- ‘e:’ indicates the message is a regular one. This means that the message is part of the precondition for the property: all messages prefixed with ‘e:’ must appear to trigger a property match. If any of these messages do not appear in the checked diagram, the preconditions for the property are not satisfied, and no matching is attempted. Regular messages should appear first in the PSC diagram.
- ‘r:’ indicates a required message. This means that if all the regular messages preceding this message are present in the checked diagram, this message must be present for the property to match. If it does not, the property is violated.
- ‘f:’ indicates a fail message. This means that if all the regular messages preceding this message are present in the checked diagram, this message must not appear for the property to match. If it does appear, the property is violated.

Here is an example of a required message in a property:



This means that if the client has sent a connect message to the server, then sends a request message, the server must send back an answer message, or the property is violated.

Here is an example with a fail message:

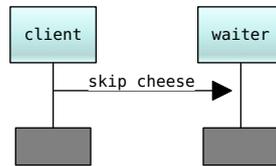


This means that if the client has sent a connect message to the server, then sends a request message, the server must not send back a not\_connected message, or the property is violated.

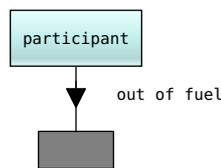
Note that in PragmaDev Process, PSC wanted or unwanted constraints will also appear in the message link text. For more details, see 6.2.2.4.

### 6.2.2.3 Sequence flow

A sequence uses a plain arrow that can go from one participant to another:



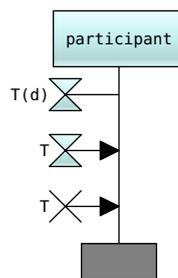
Or that can stay on the participant:



### 6.2.2.4 Lifeline components

Lifeline components are events impacting a single lifeline. They appear as symbols attached to the lifeline.

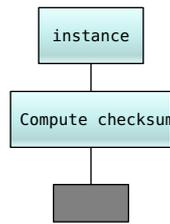
**Timer events** An instance can start timers, that will time-out in a given amount of time. A timer can also be canceled by the instance that created it. The symbols for timers are the following ones:



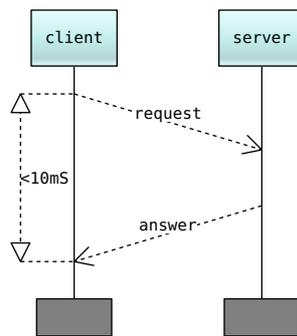
First symbol: the timer named T starts for a duration of d. Second symbol: time out for timer named T,

Last symbol: cancellation of timer named T.

**Action symbol** Action symbols describe actions performed by the lifeline. In the current version of PragmaDev Tracer, this description is informal. For example:



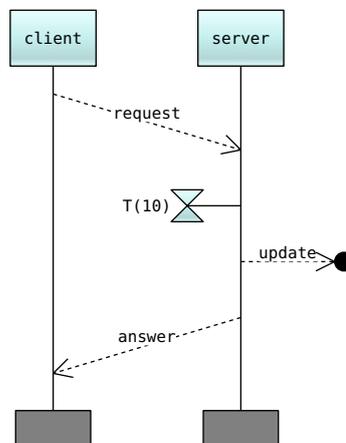
**Relative time constraints** A relative time constraint appearing in a specification MSC diagram or a PSC diagram indicates that the sequence of events it encloses must happen within a given time. For example:



This specifies there must be less than 10 ms between the time when Client sends the request message and the time when it receives the answer message.

During conformance checking, relative time constraints are compared to absolute times in the compared diagram (see 6.3.6 and 6.2.3.4). Please note that units are not yet supported: relative time constraints can only contain a valid comparison operator (<, >, <=, >=, ...) followed by a real number.

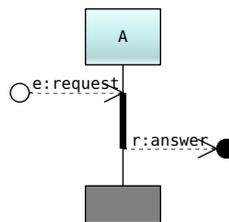
**Co-regions** A co-region on a lifeline specifies that all events happening on this lifeline can happen in any order, and not only the order specified graphically. For example:



The co-region, indicated by the dotted line on the server lifeline, indicates that the timer and the outputs of messages update and answer can happen in any order.

Note that co-regions are not supported in the conformance checking feature of PragmaDev Tracer (6.3.6). The same semantics can usually be specified by using inline expressions; see 6.2.3.3 for details.

**PSC strict operator** Events specified on lifeline in PSC diagrams are supposed to be loosely ordered by default. This means that if anything happens between two of these events, the property is matched anyway. It is however possible to specify a strict ordering for a set of events, meaning that these events must happen in this order without anything in between. This is done with the strict operator, that looks like follows:



This means that a request message received by A must be immediately followed by the output of an answer message, without anything in between (see 6.2.2.2 for the PSC-specific link text syntax).

**PSC constraints: wanted and unwanted messages & chains** PSC diagrams allow to specify on a message a set of messages that must or must not appear before or after it for the property to match. Unlike other messages, the messages in these constraint appear in what PSC calls the intra-message format, i.e as a text formatted like: <sender instance name>.<message name>.<receiver instance name>.

In the PSC specification, the constraint itself is represented by a symbol appearing under one end of the message link:

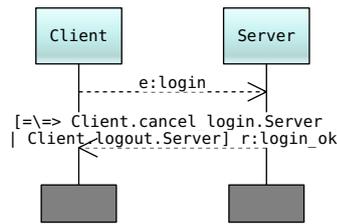
- If it appears under the link start (message output on sender), it is a past constraint, meaning it must be satisfied before the message is sent for the property to match;
- If it appears under the link end (message input on receiver), it is a future constraint, meaning it must be satisfied after the message has been received for the property to match.

In PragmaDev Process, the constraint is actually specified directly in the text of the link. So a past constraint will appear in square brackets before the text for the message itself: [constraint] message\_name(parameters...) and a future constraint will appear after the text for the message: message\_name(parameters...) [constraint]

Constraints can have several forms:

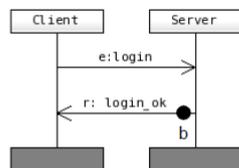
- An unwanted message constraint specifies a set of messages that should not appear. If any of the specified messages appear, the constraint is not satisfied and the property does not match. In PragmaDev Process, this kind of constraint is

represented as follows:



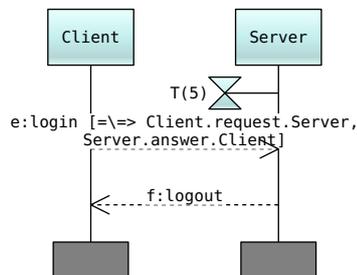
The brackets isolate the constraint from the message itself, the “=\=>” is the standard prefix for an unwanted constraint in PragmaDev Process, and the messages that should not appear before the login\_ok message are separated by a “|”, meaning that if Client sends a login message to Server, Server must answer by sending back a login\_ok message, unless either the message cancel\_login has been sent from Client to Server before, or the message logout has been sent by Client to Server before.

Note that is standard PSC, the representation would be something like:



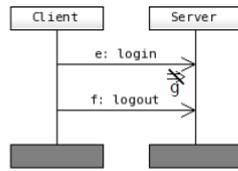
b = {Client.cancel\_login.Server, Client.logout.Server}

- An unwanted chain constraint specifies a sequence of events that should not appear. If all messages in the constraint appear in the order specified in the constraint, then the property does not match. This kind of constraint is represented in PragmaDev Process as follows:



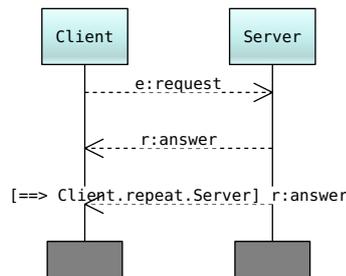
The brackets and prefix are the same as in the unwanted message constraint above, but the separator between the messages in the constraint is now a “,”, denoting a sequence. The constraint also appears after the message text, so this is a future constraint. So this means that if Client sends a login message to Server, it is a property failure if it sends a logout message after it, unless it has sent the request message and Server has sent back the answer message in-between.

Note that in standard PSC, the representation would be something like:

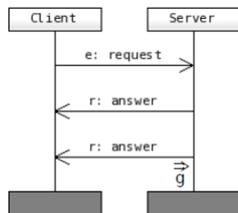


g = (Client.request.Server, Server.answer.Client)

- A wanted chain constraint specifies a sequence of events that must appear. If any of the messages in the constraint does not appear, or the messages appear in a different order than the one specified in the constraint, then the property does not match. This kind of constraint is represented in PragmaDev Process as follows:



The constraint appears before the message name, so it's a past constraint. The prefix "[]=>" is the standard one for all wanted constraints in PragmaDev Process. So this specifies that if Client sends a request message to Server, Server must send back an answer message, and then another one if Client sends the repeat message after the first answer. Note that the standard PSC representation would be something like:



g = (Client.repeat.Server)

Note: PragmaDev Process actually supports more general types of constraints called wanted and unwanted alternative chain constraint. These merge the message and chain constraints described above. The general syntax for these constraints is:

[<constraint type prefix> I1.m1.J1,I2.m2.J2,... | In.mn.Jn,... | Im.mm.Jm,... ]

where <constraint type prefix> can be either []=> for a wanted constraint, or []=> for an unwanted constraint.

- If the constraint is unwanted, this specifies that neither the sequence I1.m1.J1, I2.m2.J2, ..., nor the sequence In.mn.Jn, ..., nor the sequence Im.mm.Jm, ... should appear for the property to match.
- If the constraint is wanted, this specifies that either the sequence I1.m1.J1, I2.m2.J2, ..., or the sequence In.mn.Jn, ..., or the sequence Im.mm.Jm, ... must appear for the property to match.

This allows to represent all the PSC constraint kinds:

- An unwanted message constraint {I1.m1.J1, I2.m2.J2} will be represented as: [=\<=> I1.m1.J1 | I2.m2.J2]
- An unwanted chain constraint (I1.m1.J1, I2.m2.J2) will be represented as: [=\<=> I1.m1.J1, I2.m2.J2]
- A wanted chain constraint (I1.m1.J1, I2.m2.J2) will be represented as: [==> I1.m1.J1, I2.m2.J2]

## 6.2.3 Main symbols

### 6.2.3.1 Lifeline

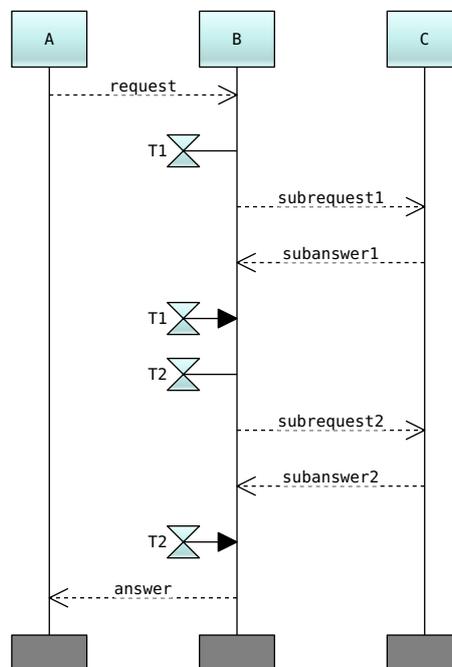
A lifeline represents an interacting entity in a MSC or PSC diagram. PragmaDev Tracer allows the instance name appearing in the lifeline head to have the following format:

<instance name>[:<class name>][(<instance identifier>)]

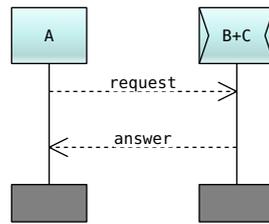
Lifelines can appear in all kinds of diagrams: MSC trace or specification diagrams, as well as PSC diagrams.

### 6.2.3.2 Collapsed lifelines

Collapsed lifelines are a PragmaDev Tracer extension and result from a ‘collapse’ operation. This allows to represent a set of lifelines as a single lifeline, events happening between the lifelines in the set being hidden. For example, after collapsing the instance B and C in the following diagram:



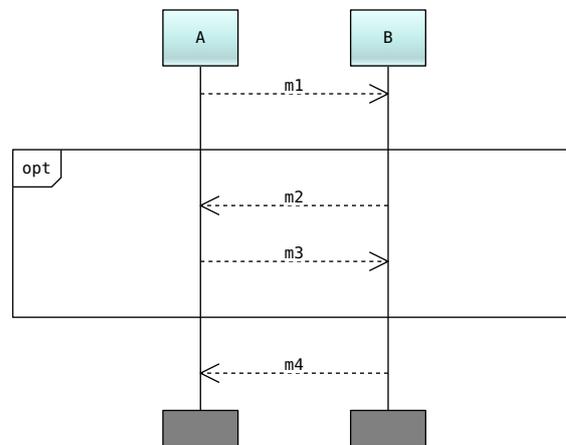
the diagram appears as follows:



### 6.2.3.3 Inline expressions

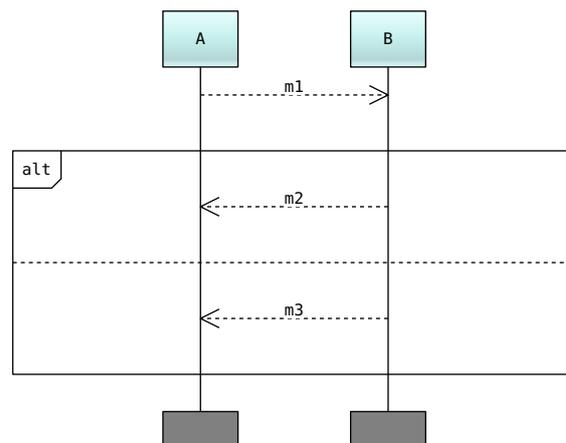
An inline expression in a specification or PSC diagram is a way to specify specific semantics for a group of events. The semantics depend on the kind of inline expression:

- An 'opt' inline expression specifies an optional set of events. For example:



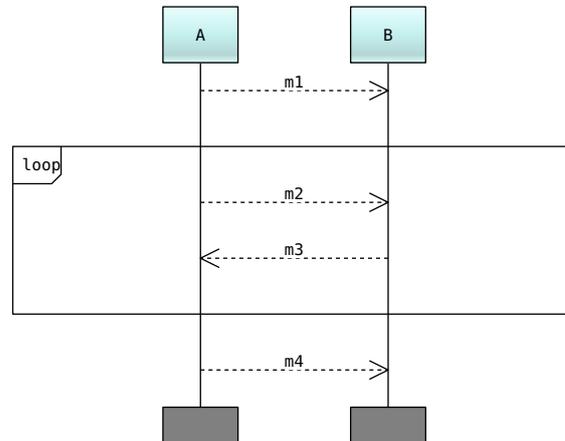
specifies that the message m1 is sent from A to B, then B may send m2 to A, which answers m3, then B sends m4 to A. So the sequences m1-m2-m3-m4, and m1-m4 are both valid.

- An 'alt' inline expression specifies a set of alternative behaviors. For example:



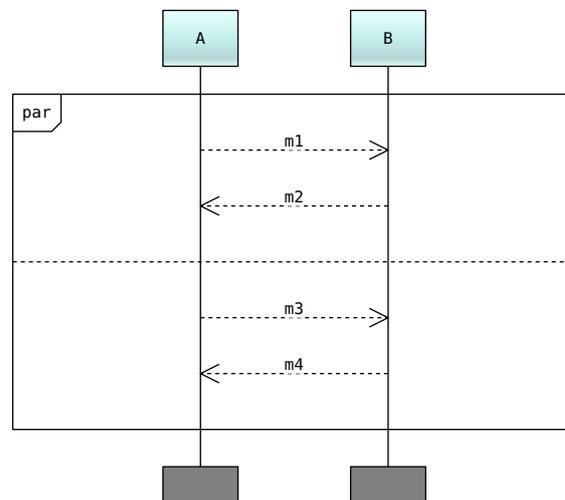
specifies that when A sends m1 to B, B may answer by sending back m2, or m3. So the sequences m1-m2 and m1-m3 are both valid. An 'alt' inline expression must have at least two compartments in it, and can have as many as needed.

- A 'loop' inline expression specifies a set of events that might be repeated several times. For example:  
 specifies that after A has sent the message m1 to B, it may send any number of messages m2, to which B will answer by the message m3, until A sends the message m4 to B. So the sequences m1-m4, m1-m2-m3-m4, m1-m2-m3-m2-m3-m4, and so on, are all valid.



Note that the MSC standard allows to indicate minimum and maximum number of repeats in loop inline expressions. This feature is not yet available in PragmaDev Tracer.

- A 'par' inline expression specifies a set of event sequences that must all happen, but in any order. For example:

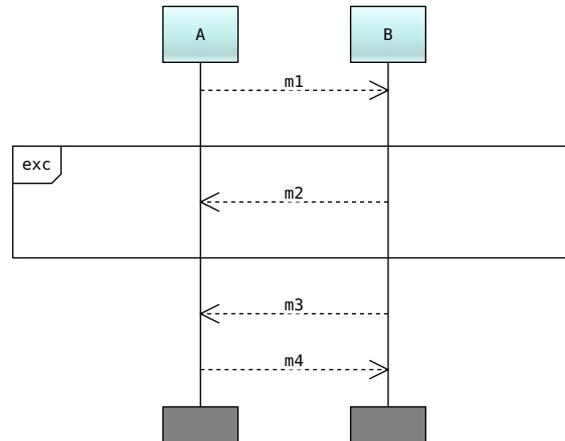


specifies that the two sequences A sending m1 to B and B answering m2, and A sending m3 to B and B answering m4 must both happen, but that the order is not significant between the sequences. So the global sequences m1-m2-m3-m4 and m3-m4-m1-m2 are both valid.

A 'par' inline expression must have at least 2 compartments, and can have as many as needed.

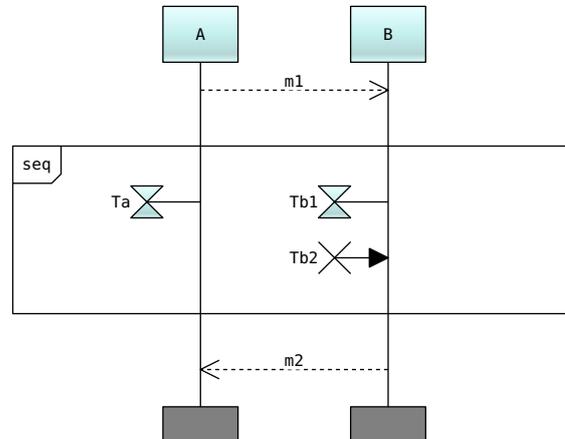
- An 'exc' inline expression represents an exception. This means the sequence of

events in the inline expression is an error case and terminates the scenario. For example:



specifies that when A sends m1 to B and B answers m2, there is an error and the scenario should stop. So the sequence m1-m2 is valid, but is an error case, and the sequence m1-m3-m4 is valid and is a normal execution. Note that the MSC standard represents an ‘exc’ inline expression with a dotted bottom line. PragmaDev Tracer uses a solid line in the current version.

- A ‘seq’ inline expression represents a weak sequence. This means that within such an inline expression, the events on a specific lifeline must happen in the given order, but the general ordering can be anything. For example:

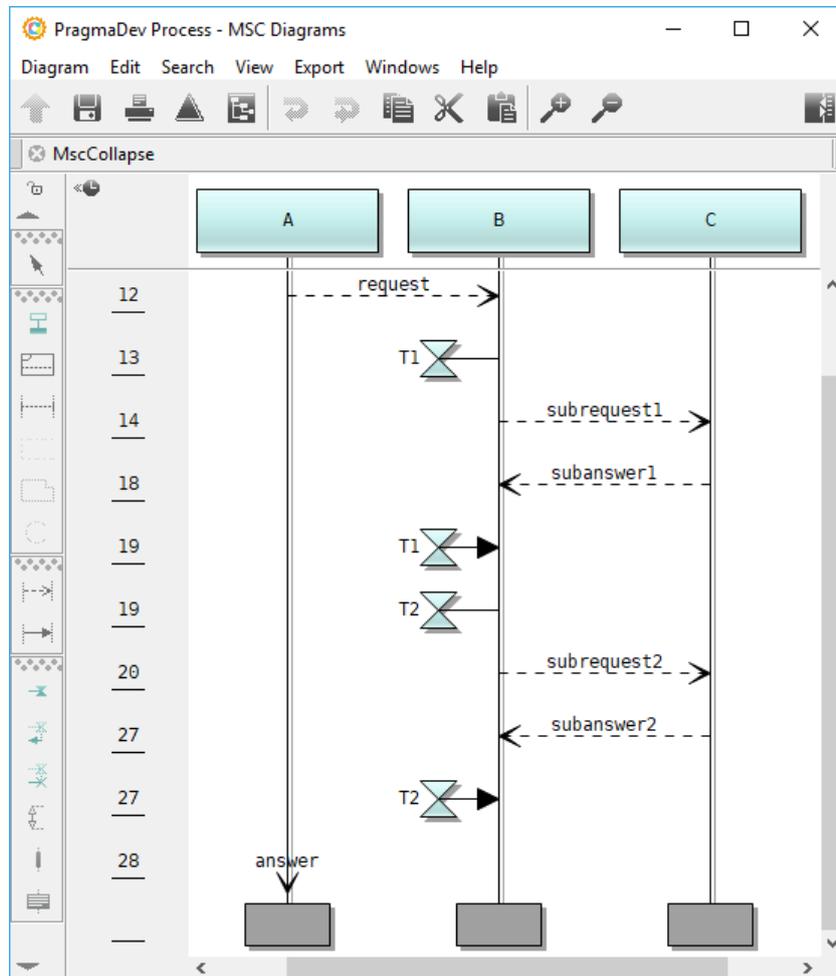


This means that on lifeline B, the starting of Tb1 has to happen before the canceling of Tb2, but that the starting of Ta by A can happen at anytime: before the starting of Tb1, after the canceling of Tb2 or between the two. Note that this kind of inline expression is not supported in conformance checking (6.3.6).

### 6.2.3.4 Absolute times

In the MSC standard, absolute times can be associated to any event in the diagram by using a symbol consisting only in a dashed underline under the text for the time. PragmaDev Tracer supports absolute times, but only associated to complete ‘event rows’:

the times are displayed in the left margin of the diagram and are associated to all events with the same y coordinate, instead of any event. To keep the same representation as in the MSC standard, each absolute time is displayed with a dashed underline:



These absolute times are the reference when verifying relative time constraints during conformance checking (see 6.2.2.4 and 6.3.6). Please note that units are not yet supported: absolute time constraints must be written as a real number only.

**6.2.3.5 Comments**

A comment symbol just contains a documentation text for the item it is attached to. Comment symbols are not yet supported in PragmaDev Tracer.

**6.2.3.6 Texts**

A text symbol contains informal text usually describing global items in the diagram. Text symbols are not supported yet in PragmaDev Tracer.

## 6.3 MSC editor

This kind of editor is used for Message Sequence Charts. A MSC diagram describes a sequence of events happening in a system, with a set of “lifelines” represented as vertical lines, with symbols representing events attached to them.

There are 3 main kinds of MSC diagrams, which are all recognized by PragmaDev Process:

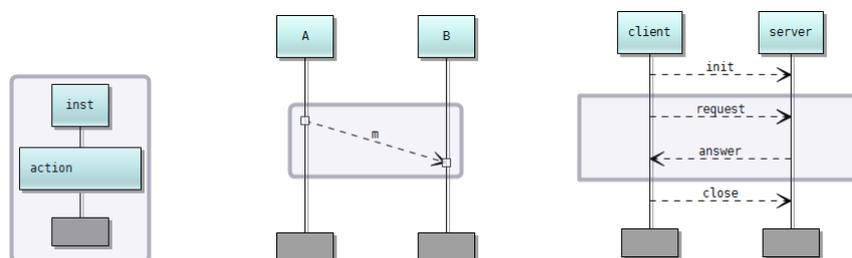
- Basic MSCs represent a sequence of events that have actually happened during a system execution. They will contain a lifeline for each participant or lane, and events will be sequence flows, message exchanges, and timeouts. They are typically obtained by using the MSC tracing functionality in the executor.
- Specification MSCs will contain the same kind of events, but can group them within other symbols with attached semantics. For example, a sequence of events can be isolated in another MSC diagram that will be referenced via a “MSC reference” symbol. Or a sequence of events can be enclosed in an “inline expression”, allowing to specify this sequence is optional, or can be repeated several times.
- Property Sequence Charts are another kind of specification MSCs that are used to describe “if/then” conditions: if a given sequence of events appear in a diagram, then another sequence must appear behind it, or must not appear behind it.

The whole format for MSCs - basic & specification - and PSCs is described in 6.2. Note that there is no specific editor for each kind of diagrams: all symbols are available in the editor, and the kind of diagram is recognized automatically from what it contains.

The features of the MSC editor and their availability are described in the following paragraphs.

### 6.3.1 Specific tools

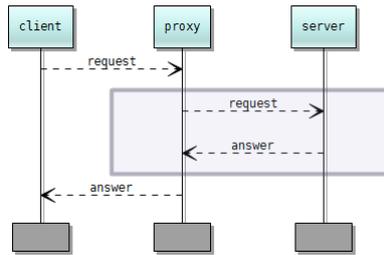
The selection tool in MSC diagrams allows to select symbols and links, just as in other editors. It also allows to select a rectangular zones, that can be copied and pasted and exported as image files:



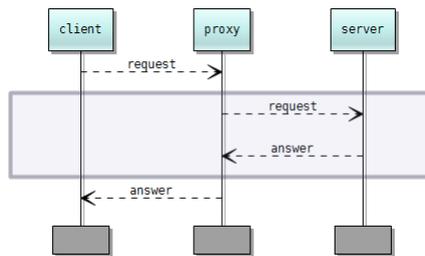
*Selected lifeline   Selected message link   Rectangular selection*

Note that copying and pasting rectangular zones will work only for full “horizontal slices” of the diagram: if a rectangular zone is selected, but does not span the full diagram width, it will be automatically extended when copied or cut.

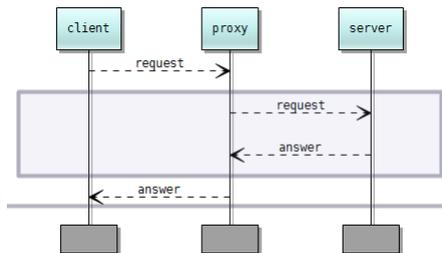
For example, if this zone is selected:



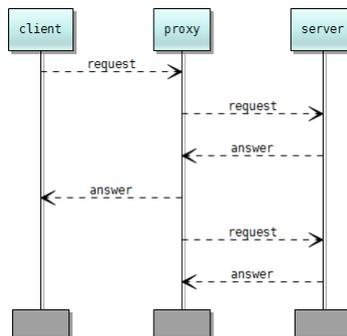
copying it will automatically extend the zone to the full width of the diagram and display a warning:



When pasting a rectangular zone, a horizontal insertion line will be displayed:



Clicking in the diagram while the insertion line is displayed will paste the copied zone at this position:

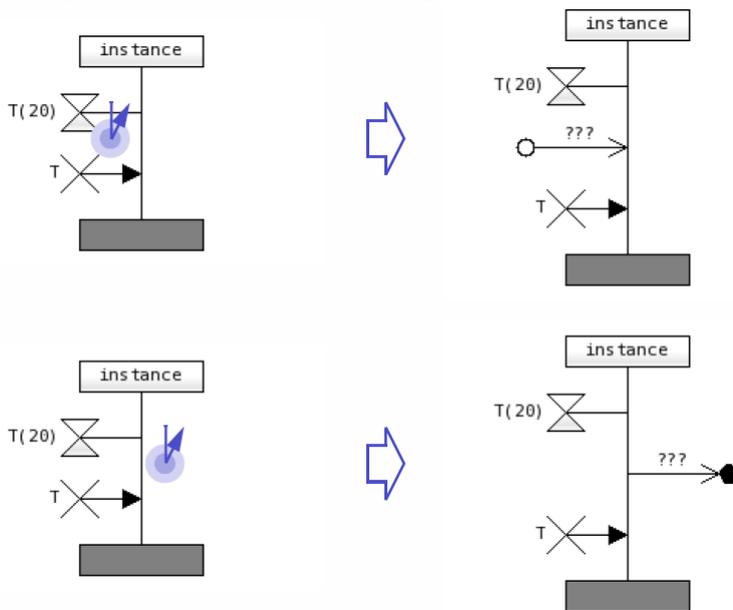


Note that the copy will fail if any object has an end within the slice but the other end outside it, such as a lifeline starting before the slice and ending in it. The paste will fail if one of the copied lifelines does not exist at the paste position.

### 6.3.2 Symbol creation

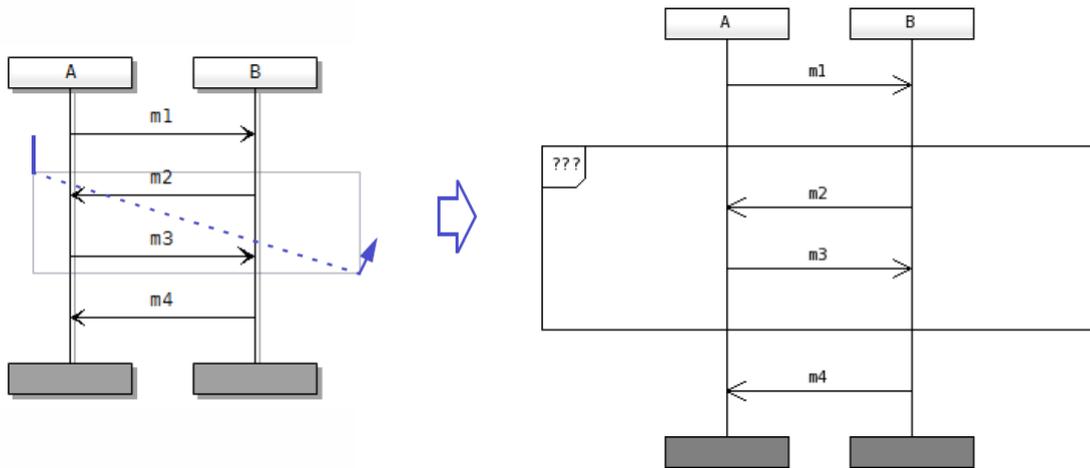
Creating symbols in MSC diagrams is pretty straight forward. But some of the tools have a special behavior, mostly because of the nature of the MSC diagrams, which describes mostly a sequence of events, and not individual symbols:

- When creating lifelines, only the horizontal position will be considered: lifelines are always created starting from the top of the diagram and going to the bottom.
- The creation of a lost (resp. found) message is done by selecting the message creation tool and clicking on the right side (resp. left side) of the lifeline sending it (resp. receiving it). For example:

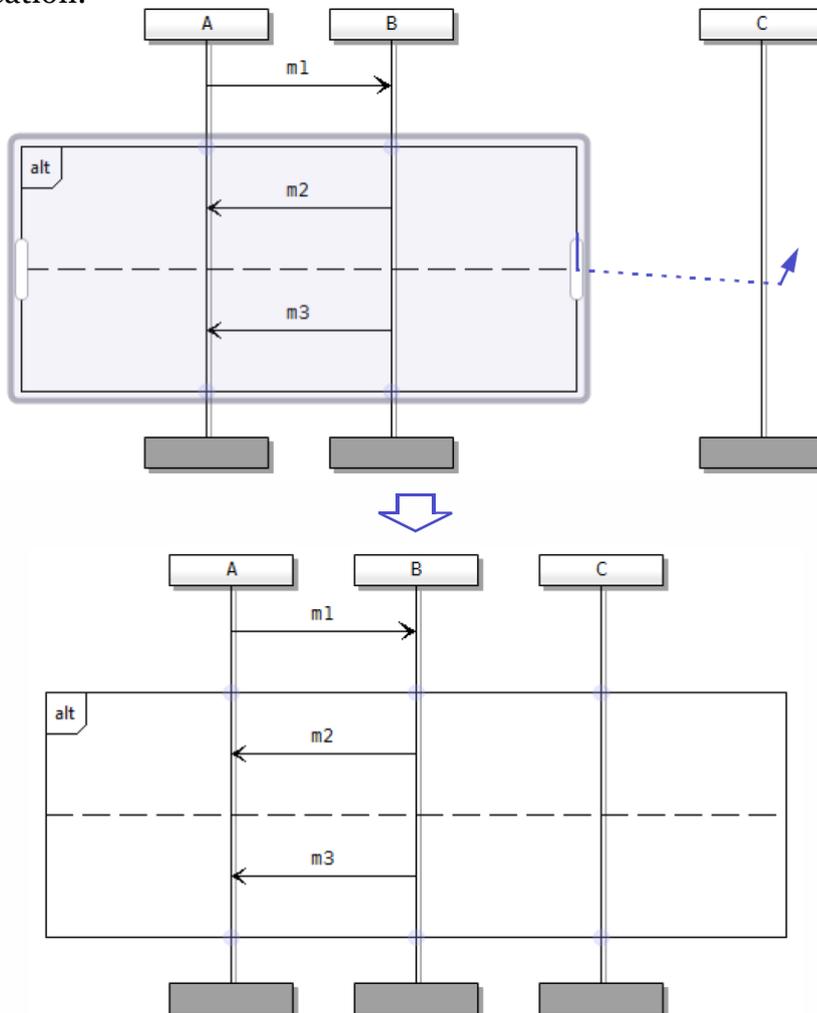


Note that in legacy diagrams, lost and found messages have their own specific symbol that must be created the usual way, and a message link has to be created between the lost (resp. found) message symbol and its sender (resp. receiver) lifeline.

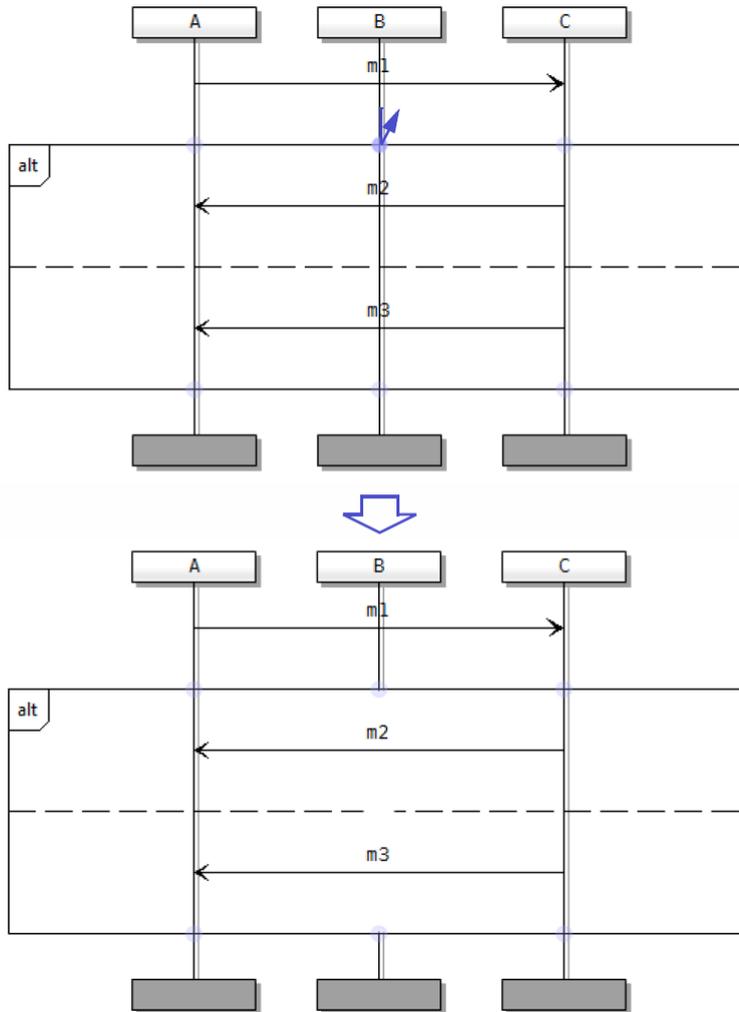
- For conditions, MSC references and inline expressions, they must be created over the lifelines they impact. This is done simply by making them span these lifelines, and optionally all the events that must be included in them:



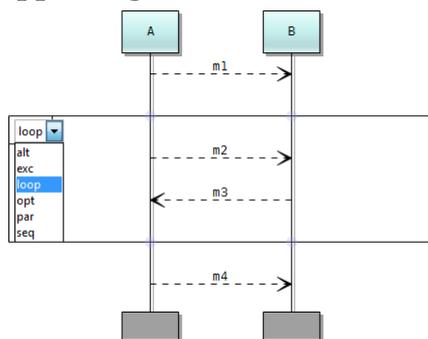
Once created, this symbols can be moved up or down by dragging them, and re-sized horizontally via the handles appearing on their sides when they are selected, which is the way to make them impact other lifelines than the ones setup at their creation:



Excluding from the symbol a lifeline included in it can be done via the circular handles appearing at the connection points between the symbols and the lifelines:



For inline expressions, their kind can then be changed by selecting it in the box appearing when the mouse cursor is over its text:



### 6.3.3 Manipulating components in lifelines

To the difference of all other symbols, lifeline are composite symbols: they may include several components like segments, timers or time constraints. They may also die before the end of the diagram or survive it.

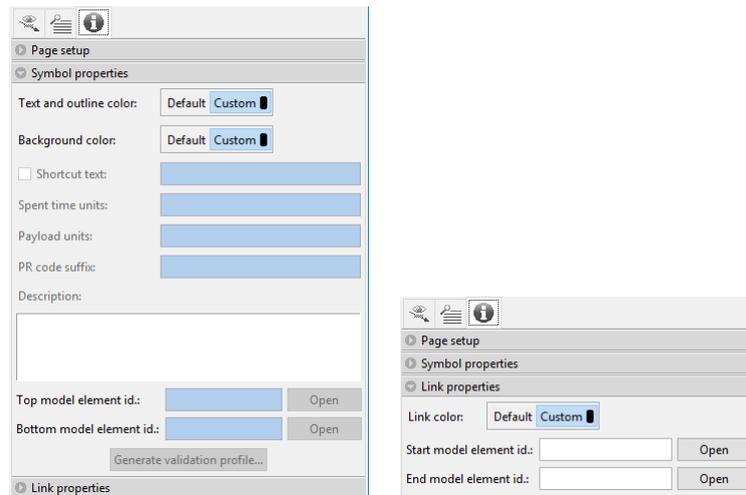
In normal diagrams these features are managed via the toolbar:



The buttons add to the lifeline a segment, an action symbol, a timer or a time constraint. After selecting an item in the toolbar, press the mouse button at the desired position in the lifeline, and drag to its end position (if applicable). To cancel the insertion, hit the Esc key or select the selection tool.

#### 6.3.4 MSC symbol and link properties

Symbols and links in MSC diagrams display internal information on the right panel. Please note some of the information may not be relevant to BPMN execution trace yet:



The information might be recorded automatically if the MSC diagram is a trace from an execution. They can also be specified “manually” via the symbol or link properties. PragmaDev Process will open itself the model elements that it has recorded in traces.

### 6.3.5 Message parameters display

Please note this feature is not used in the current version of PragmaDev Process.

A specific sub-menu in the “View” menu controls the message parameter visibility:

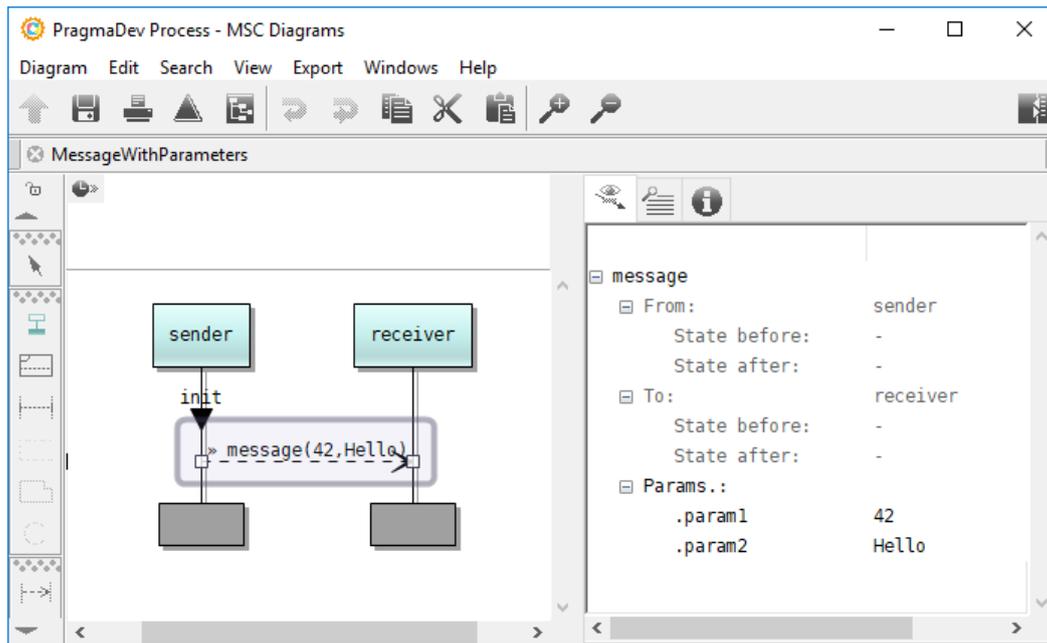
- A visibility set to “Full” displays the full text for the message parameters as it is recorded in the diagram file. The parameters for structured messages are then displayed in a flat textual format which can be quite difficult to read as this format is quite complex;
- A visibility set to “Abbreviated” still displays completely parameters for non-structured messages, but only displays the first level of parameter values in structured parameters. An example of this visibility can be seen below;
- A visibility set to “None” hides all message parameters.

This visibility setting is stored with the diagram. Please note it is only possible to modify the text for the message parameters if the visibility is set to “Full”.

When the visibility is set to “None” or “Abbreviated”, structured messages are indicated by a “»” before their name. Their parameters may be displayed by clicking on the message link: a panel then appears in the right part of the editor window displaying the parameters as a tree. For example, for a message with the full text:

```
mParams(|{param1|=42|,param2|=Hello|})
```

the display with parameter visibility set to “Abbreviated” and the link selected is:



Other information is also displayed in the panel:

- The sender and receiver process;
- The states of the sender and receiver processes before and after they sent or received the message (not relevant with current version);

### 6.3.6 Conformance checking: diagram diff & property match

PragmaDev Studio offers 3 levels of conformance checking:

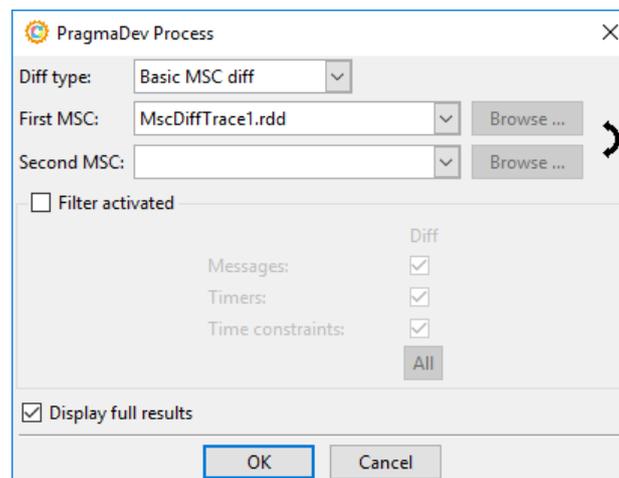
- A MSC trace can be compared to another MSC trace, used as a reference. This can typically be used for regression testing, the reference trace giving the wanted behavior, and being compared to a newly obtained trace. In this kind of comparison, all events in both diagrams are compared one by one without any interpretation of any kind. This is mainly intended for trace comparisons, but it also works on other diagram kinds, as items normally only present in specification or PSC diagrams are taken into account too, e.g inline expressions or relative time constraints.
- A MSC trace can be compared to a specification diagram. For this comparison, the semantics in the specification is taken into account. For example, if there is an 'opt' inline expression in the specification containing a sequence of message exchanges, the comparison will interpret it, and consider that the diagrams are matching if the sequence is there, or if it is not there at all. This allows to describe expected scenarios in a powerful way via specification MSC diagrams and match the execution traces against them later.
- Occurrences of a property described in a PSC diagram can be found in a MSC trace. In this case, the semantics are considered in the PSC diagram, as well as the PSC specific elements. Note that this is different from a specification vs. trace comparison, as properties describe a small part of a scenario that can actually match

several times in a trace. MSC specification diagrams describe a whole scenario, and will be matched entirely on the trace. Properties are a good and powerful way to specify wanted and unwanted behavior in the designed system.

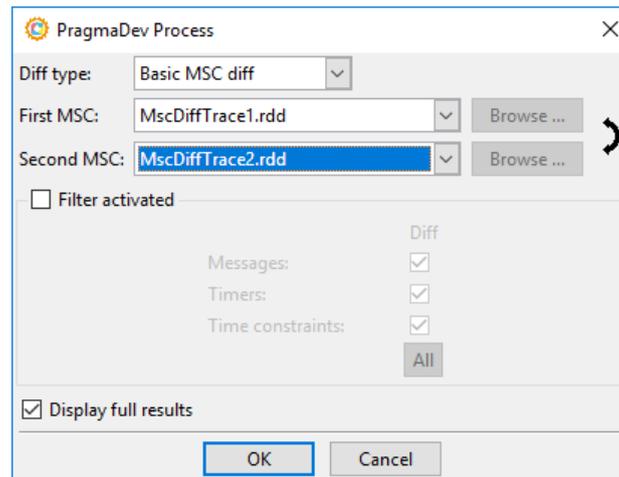
Important note: the current implementation of the algorithm used for specification vs. trace comparisons and property matches is limited in the number of events it can handle after a matched one and before the next one. The current limitation is 200 events in the trace, so if an event matches and the next event that should match is more than 200 events away from it, it won't be found. This limitation will be removed in a future version.

### 6.3.6.1 Basic MSC diff: trace vs. trace, spec. vs. spec., ...

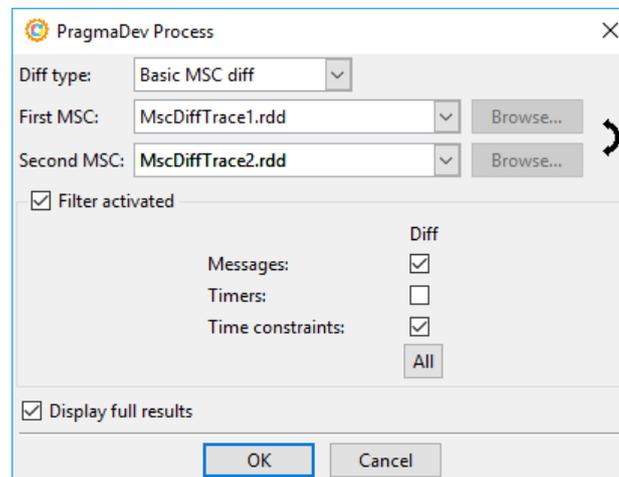
The basic MSC diff just compare two diagrams events by events and reports the found differences. This kind of comparison is launched by selecting 'Compare diagrams...' in the 'Diagram' menu, or by clicking the  button in the toolbar. The following dialog then appears:



Selecting the basic MSC diff is done by selecting the corresponding value in the 'Diff type' field. The name for first MSC will be automatically set to the name of the currently displayed diagram. For the MSC to compare, it can be either selected in the list attached to the 'Second MSC' field, or loaded from a file via its 'Browse...' button. Once selected, the arrow in the right part of the dialog allows to exchange the two MSCs if the comparison must be done the opposite way.



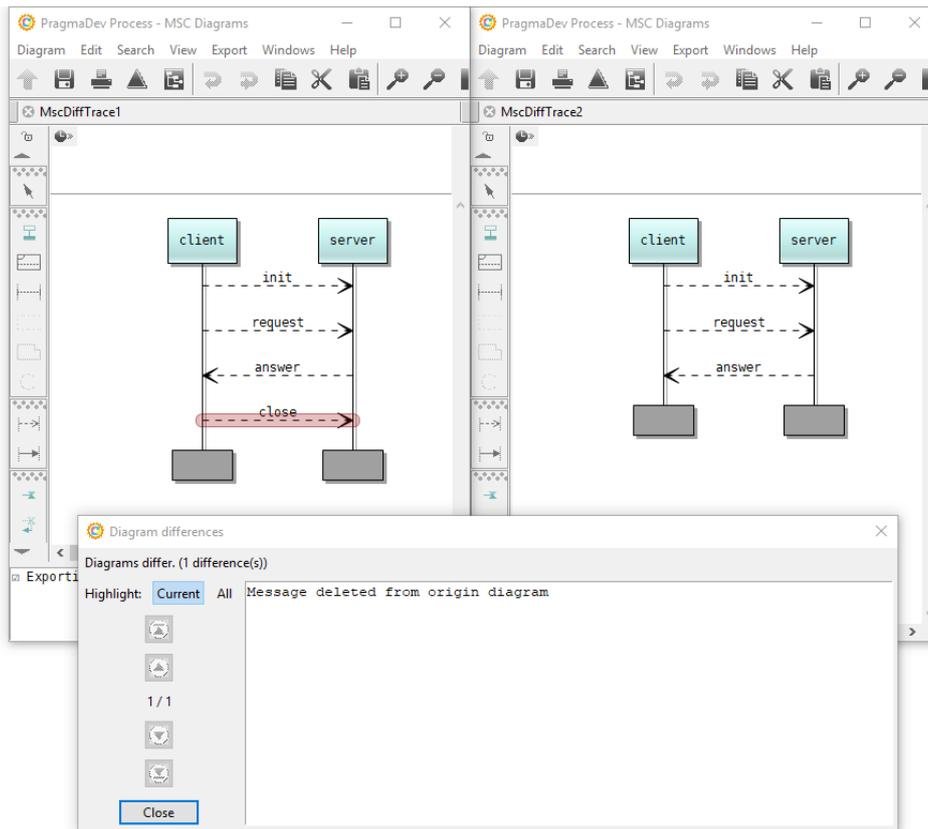
PragmaDev Studio allows to exclude some elements from the comparison based on their type. This is done by checking the 'Filter activated' option:



All the shown element types can be included or excluded from the comparison. The 'All' button will check all the boxes if any of them is unchecked, and uncheck them if all are checked.

The option 'Display full results' at the bottom of the dialog allows to display only a summary of the comparison results instead of the full set of differences. To display the summary, just uncheck the box.

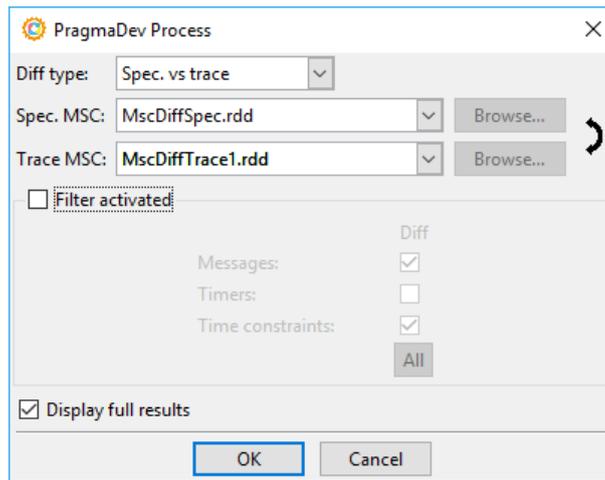
If this option is checked and after validating the dialog, PragmaDev Studio puts each diagram in its own window and displays them side by side. A dialog also appears at the bottom of the screen, allowing to browse through the found differences:



A summary of the differences is displayed at the top. Each difference will be highlighted in red in the diagram displayed on the left, and in blue in the diagram displayed on the right. The text in the dialog gives a short description of the identified difference. The arrows allow to browse through the differences. The option 'Highlight' allows to highlight all differences in both diagrams to get a quick view of what differs without having to browse through all the differences.

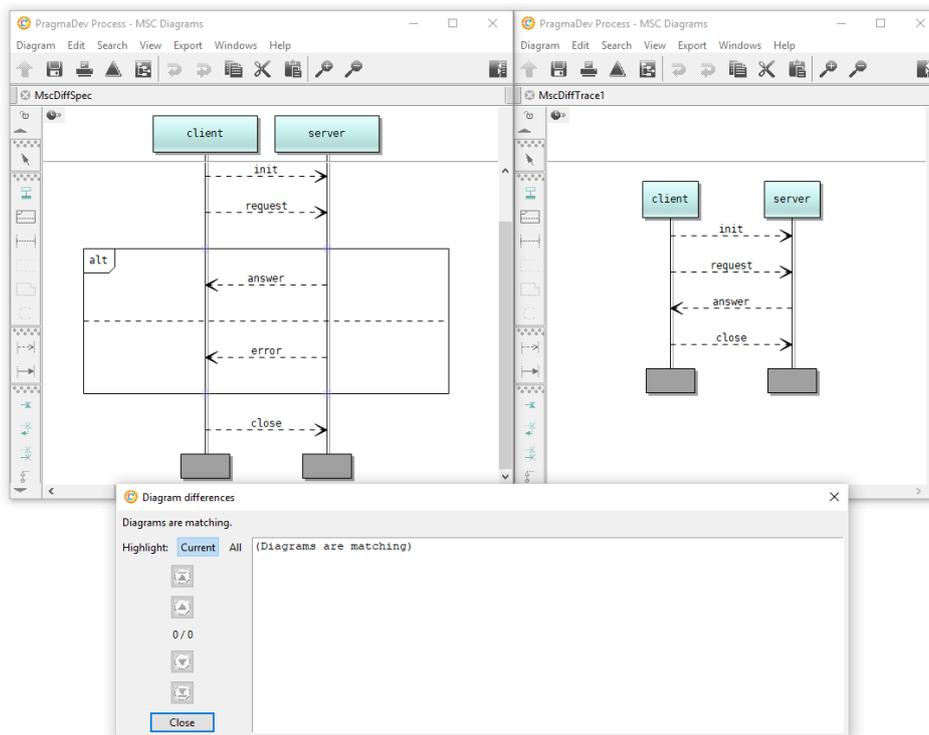
### 6.3.6.2 Spec vs. trace comparison

Comparing a specification diagram to an actual trace is done the same way as for a basic MSC diff, except the diff type has to be set to 'Spec. vs. trace' in the dialog:



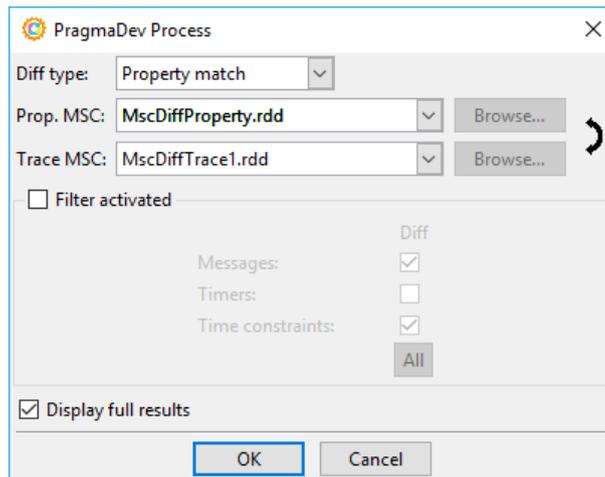
Note also that the specification diagram must be the one specified in the field ‘Spec MSC’ in the dialog, which is always the first one. If needed, the diagrams can be swapped by using the arrow button on the dialog’s right side. The same filters are provided as for a basic MSC diff.

Once validated, the found differences are displayed in the same way as for a basic MSC diff; only the way to perform the comparison changes, as semantics in the specification is taken into account where it wouldn’t be in a basic MSC diff:



### 6.3.6.3 Property match

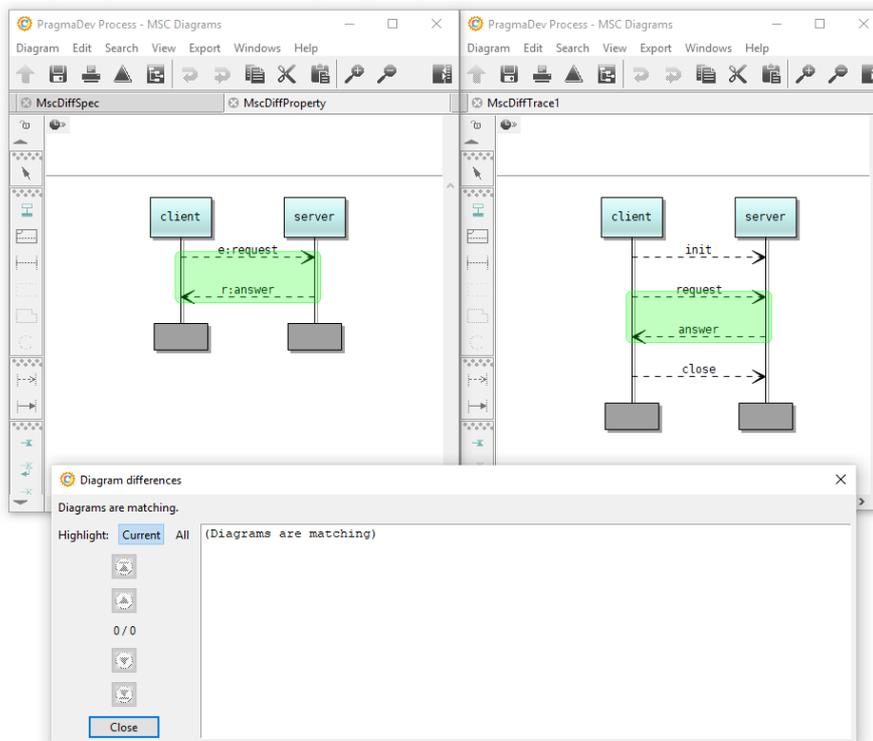
Matching a PSC diagram against a MSC trace is done the same way as for the other kinds of comparisons, except the diff type has to be set to ‘Property match’:



Note that the PSC diagram has to be the one specified in the ‘Prop. MSC’ field, which is always the first one. If needed, the diagrams can be swapped with the arrow button on the right side of the dialog. The same comparison options are provided as for basic MSC and specification vs. trace comparisons, but they are less significant here, as a property diagram is always partial.

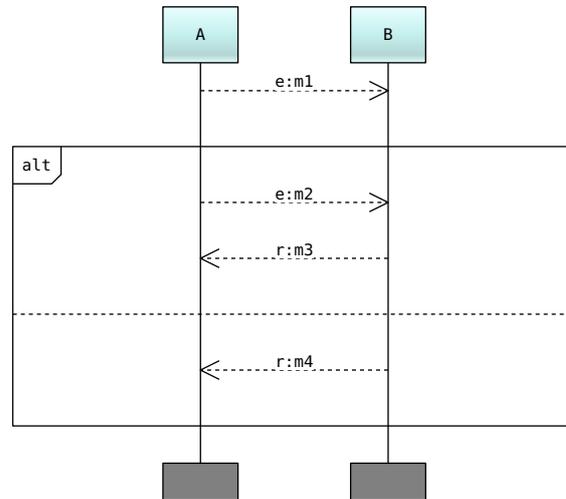
Once validated, the property matches and violations are displayed in a similar way to the display of differences in the other kinds of comparisons. Mostly the colors and the difference descriptions differ: each matched element in the property or the MSC diagram will be displayed as green, and each unmatched one as red. The difference description will be:

- ‘Property match’ if the property matches:



- ‘Violated property!’ if the property does not match.

- ‘Possibly violated property’ in some very specific cases where it is impossible to tell if the property is matched or not. A typical example where this case happens is the following:



If the trace contains a message m1 from A to B, followed neither by a message m2 from A to B, nor by a message m4 from B to A, there’s no way to know which part of the alternative should have matched. But if it was the first part, the message m2 is not there, so the property does not apply, and if it was the second one, the required message m4 is not there either, so the property is violated. In this case, a possible property violation will be reported. Note that a property is not necessarily violated if something does not match in it. Typically, an unmatched fail message means the property is matched.

## 7 Verifier

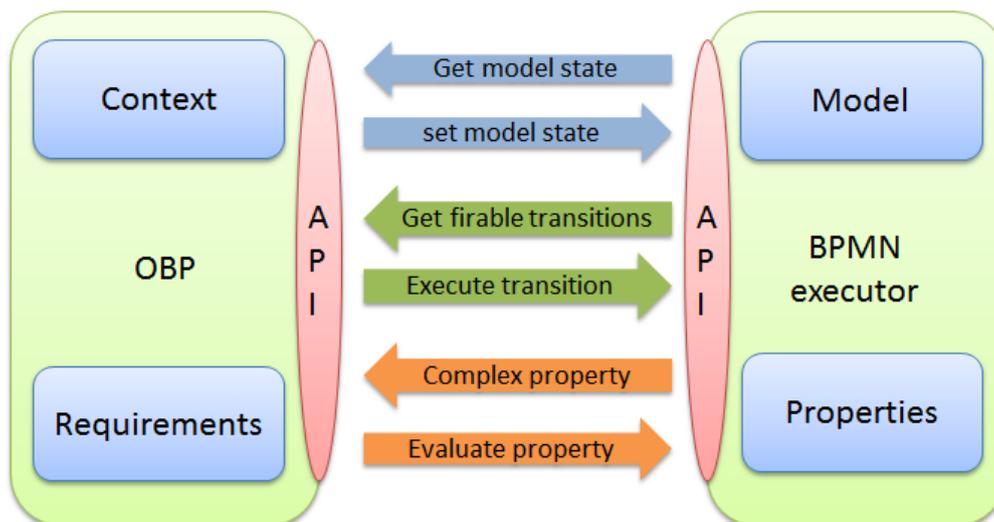
The Verifier automatically explores all possible execution path of the model. The possible paths and their intermediate states build what is called a state space. There are two interesting results for this feature:

- The size of the state space. That is basically the number of possible execution steps in the model. If it is too high that means the model is complex and might not be doing what is expected.
- Verify a property in order to make sure some scenario actually can not happen whatever the scenario.

Please note OBP requires Java to be installed on the computer.

### 7.1 Architecture

The state space exploration is done with OBP (Observer Based Prover) tool developed by ENSTA Bretagne research lab. To do so OBP relies on PragmaDev Executor. At each step OBP asks the executor what are the possible paths of execution (what we call transitions). OBP will then try all possible paths and the executor will provide the resulting state for each one. For each resulting state OBP will ask for the possible transitions and so on. The key aspect here is that OBP does not execute the model, it relies on the executor. This is quite unique as usually verification tools have their own semantic and executor.



While exploring OBP can verify user defined properties defined with a PSC (Property Sequence Chart) diagram. Internally the PSC is translated to a Büchi automaton and

sent to OBP. The Büchi automaton is based on what we call atomic properties of the model. At each step of execution OBP asks the BPMN executor to evaluate the atomic properties to evaluate the overall Büchi automaton. While atomic properties are static and usually boolean, the Büchi automaton can express quite complex sequence of events.

## 7.2 Properties

The properties are defined using the PSC format. For the time being only the following features are supported in the PSC:

- e (regular message),
- r (required message),
- f (failed message),
- Alt in-line expression,
- Strict operator.

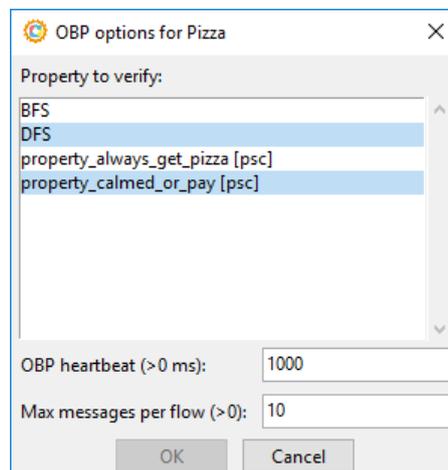
Please refer to the 6.2.2.2 pages for more information.

## 7.3 Launch a verification

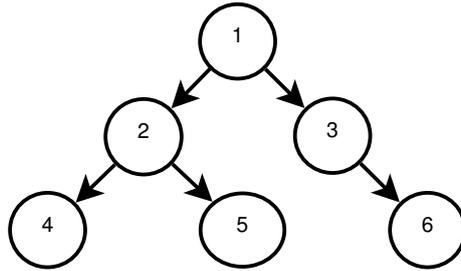
The Verifier is launched from the editor windows with the following button:



The following window will then open:



The first 2 lines are always present, they are the default full state space exploration policies. To explain these two exploration techniques let us consider the following state space:



The exploration policies will do the following:

- BFS: Breadth First Search: 1 2 3 4 5 6
- DFS: Depth First Search: 1 2 4 5 3 6

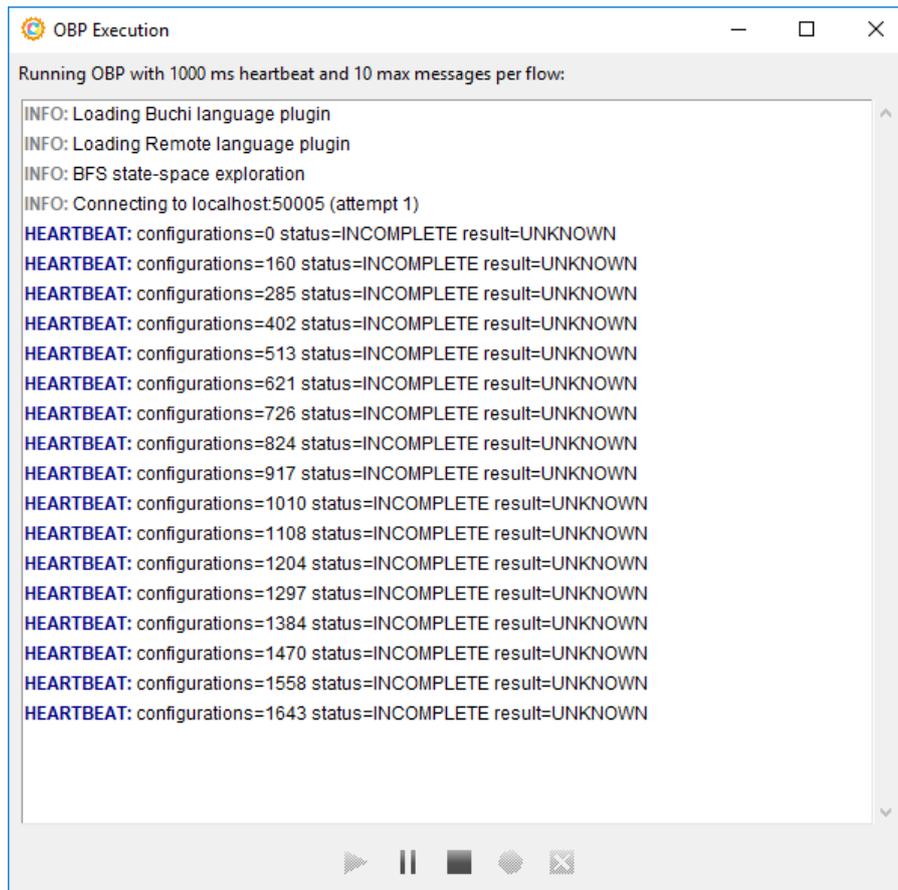
Depending on your model organization, one or the other might be better choice.

The lines after that are the PSCs found in the project. A PSC defines a single property. OBP can only explore for one property at a time.

An exploration might take a substantial amount of time. For that reason, every heartbeat, some status information is displayed in the exploration window. It is possible to change the refresh value of the status information.

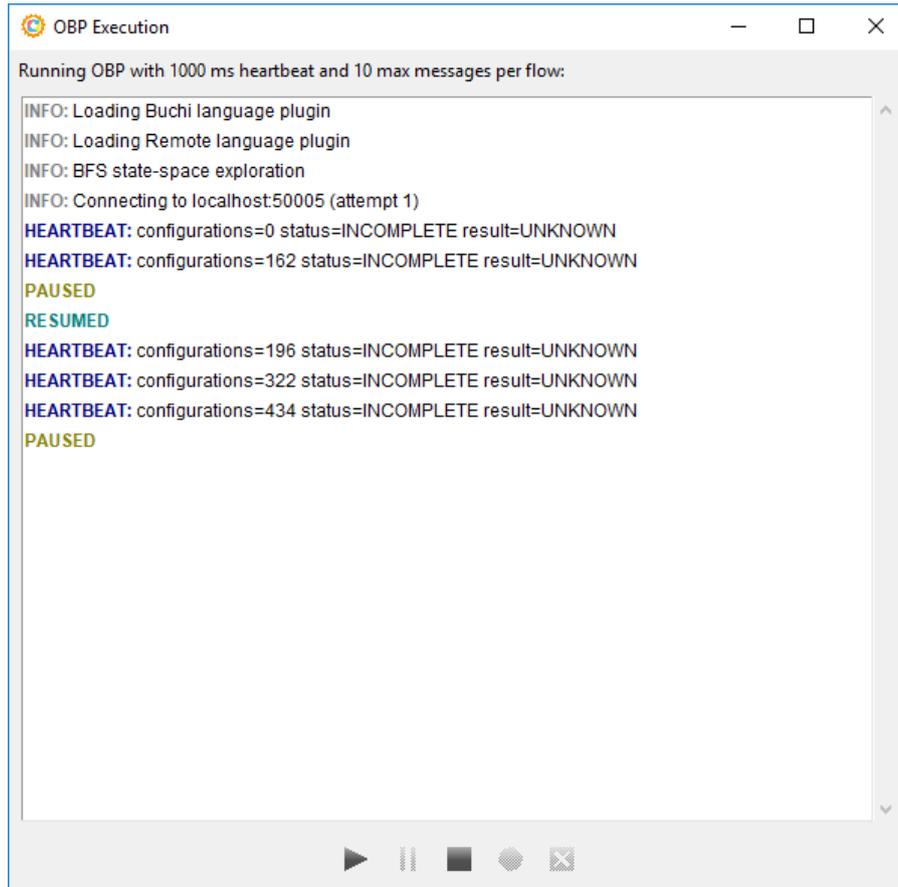
During exploration the history states are cleaned up because they do not impact the upcoming execution of the model. For example a loop executed once or a hundred times will end up in the same state. But if a message is sent from the loop this creates a different configuration that is to remember. Since this can potentially create an infinity of possible configuration (one per number of messages in the queue), the exploration window offers an upper limit for the number of pending messages in the queue.

Once the exploration is launched, the following window will show what the status is:

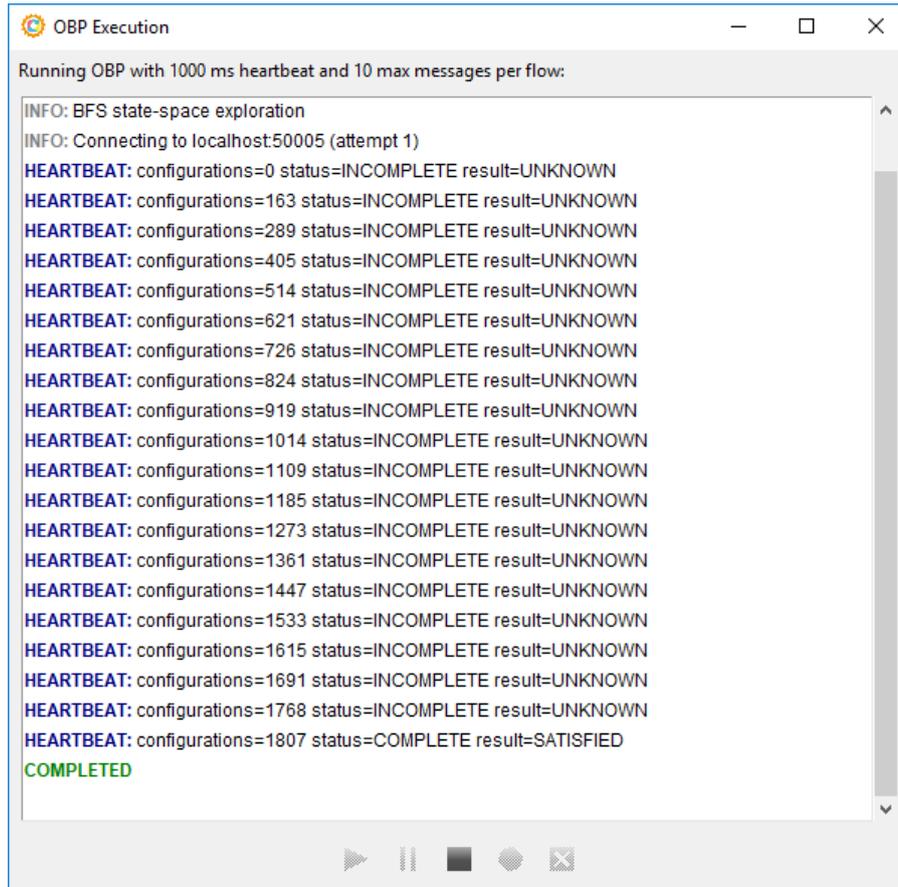


Since exploration can take quite some CPU and memory, it is possible to pause the exploration with the Pause button: 

The exploration can then be resumed or paused again:



It is also possible to interrupt the state space exploration with the Stop button:  Once this button pressed, the exploration is interrupted and can not be resumed. At the end of the exploration the exploration window will indicate a COMPLETE status:



The screenshot shows a window titled "OBP Execution" with a standard Windows-style title bar (minimize, maximize, close). Below the title bar, there is a status bar that reads "Running OBP with 1000 ms heartbeat and 10 max messages per flow:". The main area of the window is a text log with the following content:

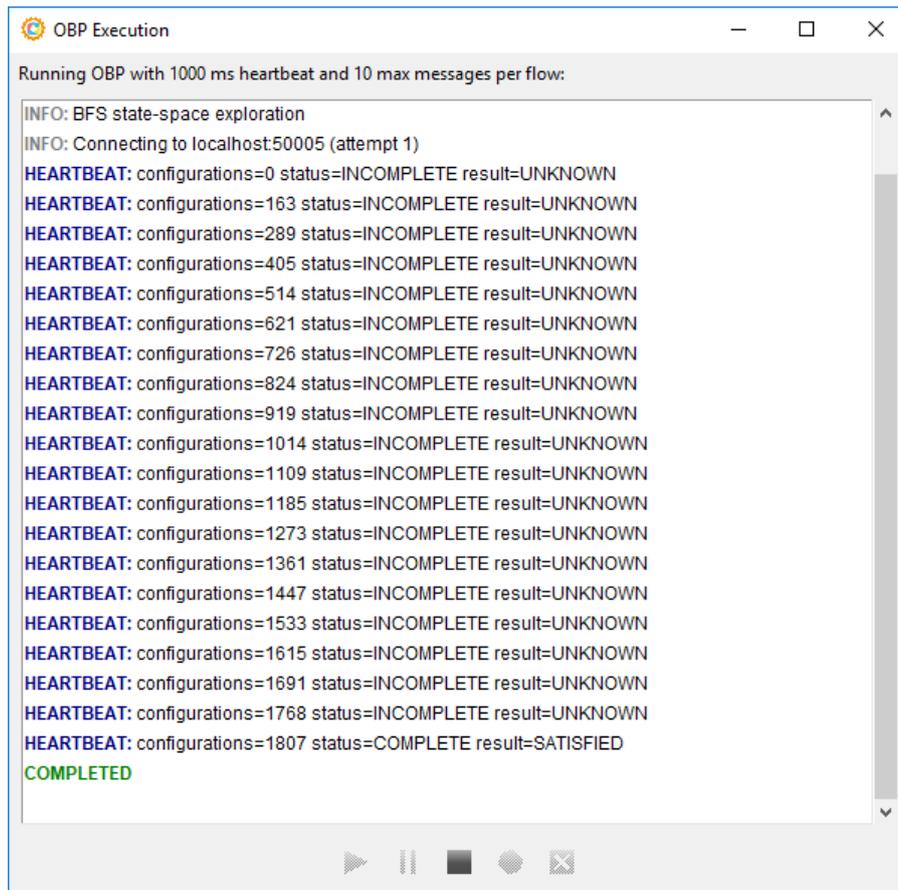
```
INFO: BFS state-space exploration
INFO: Connecting to localhost:50005 (attempt 1)
HEARTBEAT: configurations=0 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=163 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=289 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=405 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=514 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=621 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=726 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=824 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=919 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=1014 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=1109 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=1185 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=1273 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=1361 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=1447 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=1533 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=1615 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=1691 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=1768 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=1807 status=COMPLETE result=SATISFIED
COMPLETED
```

At the bottom of the window, there is a control bar with several icons: a play button, a pause button, a stop button, a refresh button, and a close button.

## 7.4 Result analysis

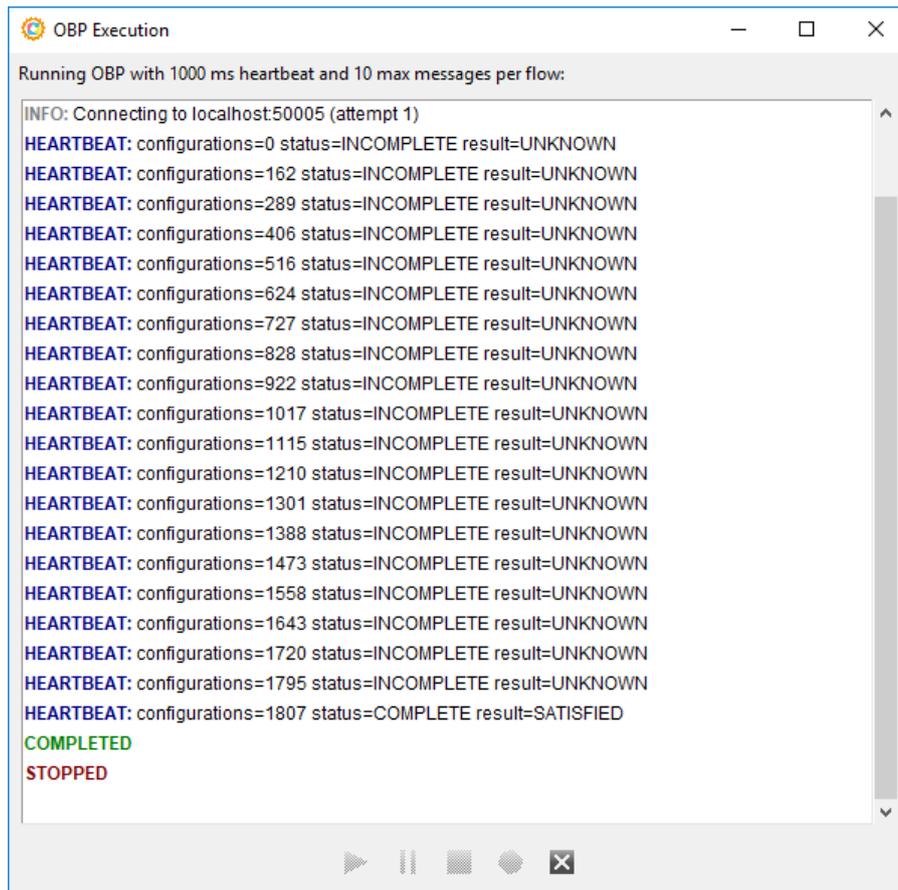
### 7.4.1 Full state space exploration

The full state space exploration does not verify any property, it is only a full exploration of all the possible configurations of the model. At the end of the exploration the only valuable information is the number of configurations that have been explored. In the example below:



1807 different configurations have been explored. This has to be compared to the model complexity to find out if that result is too high or normal. A simple plain sequence is usually less than a hundred configurations. If it is too high, there is probably a mis-construct in the model. But if there are messages exchanged in loops in the model the number of configurations can go way up a thousand.

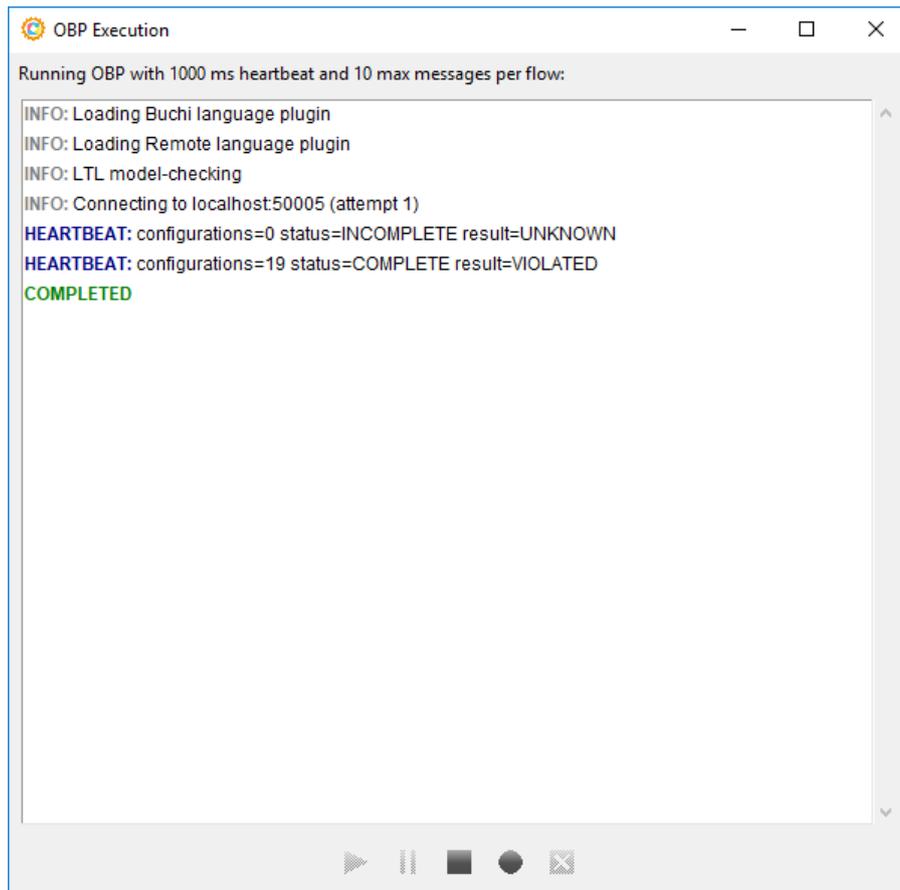
The Stop button  will stop OBP in the background:



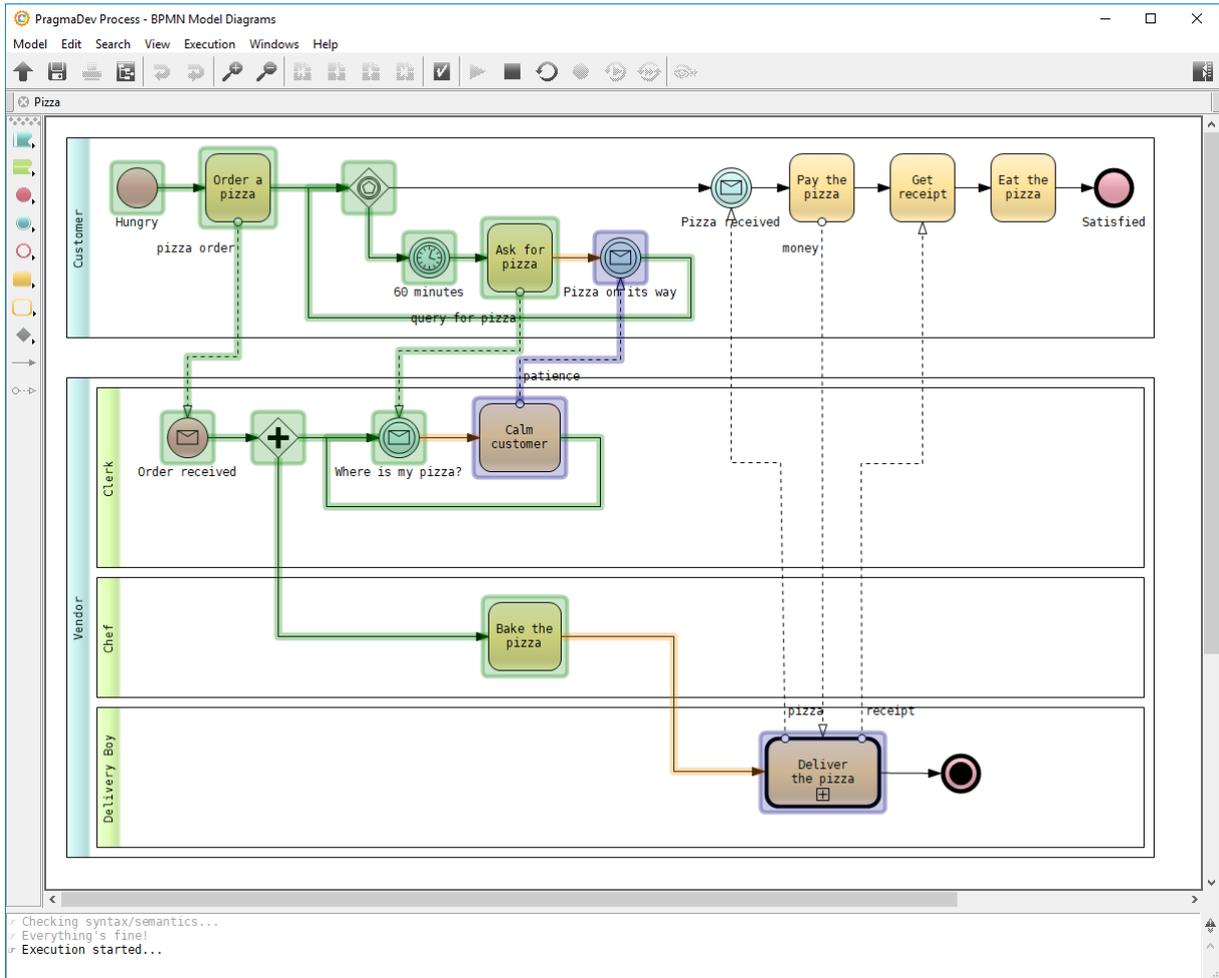
The Quit button  will propose to save the exploration information and then close the exploration window.

### 7.4.2 Property verification

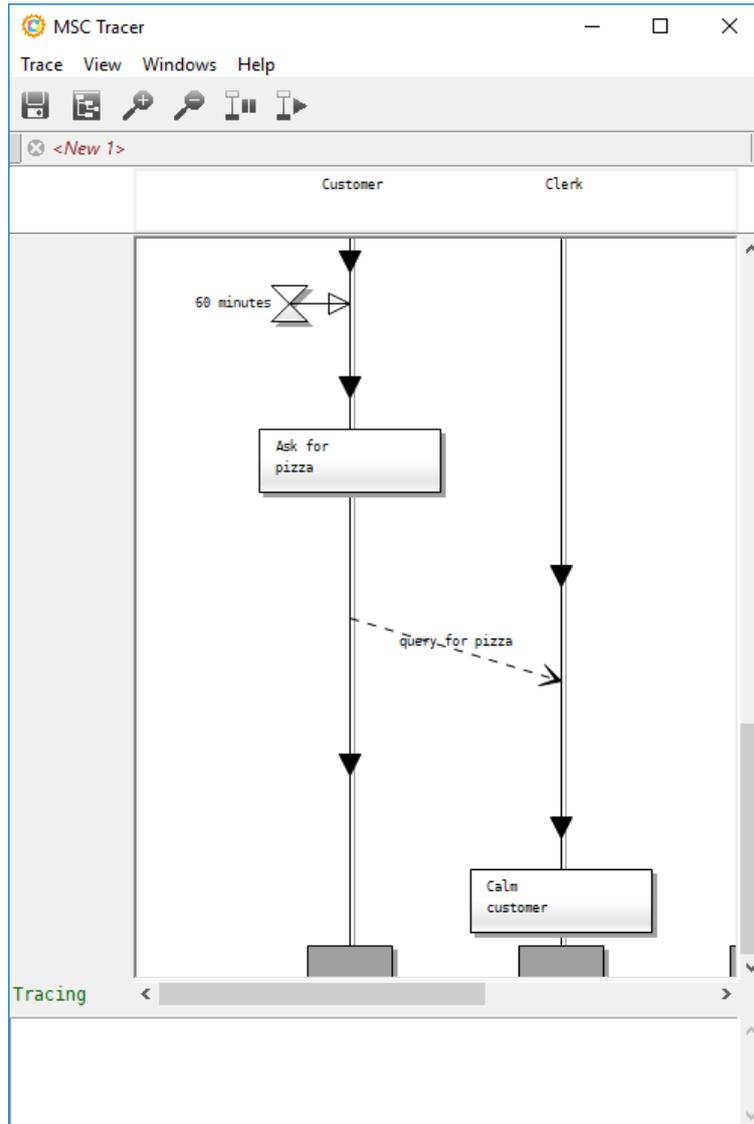
At the end of the exploration, the window will indicate if the property has been violated or not. In the example it has been violated quickly after starting the execution:



Pressing the Record button  to replay the scenario that violated the property:



An execution traced is also generated with all the execution steps:



The problem can then be fully analyzed and the model corrected.

## 8 Glossary

Acronym	Meaning
BPMN	Business Model Process Notation
MSC	Message Sequence Chart
PSC	Property Sequence Chart
OBP	Observer Based Prover
BFS	Breadth First Search
DFS	Depth First Search