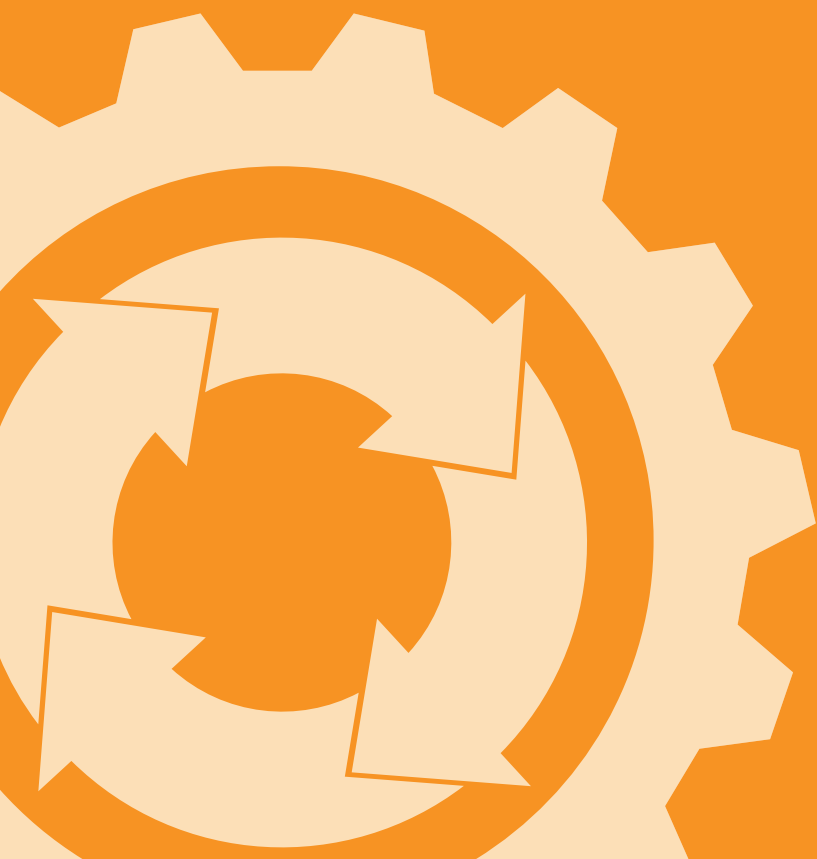




TUTORIAL



PRAGMADEV
modeling and testing tools

Contents

1	Introduction	3
2	The model	4
2.1	Organization	4
2.2	Pools and lanes	5
2.3	Processes	8
3	Execution	14
3.1	Semantic check	14
3.2	Interactive execution	16
3.3	Automatic execution	30
3.3.1	Single-trace execution	30
3.3.2	Multi-trace execution	40
4	Exploration	44
4.1	Complexity check	44
4.2	Reachability	45
4.3	Deadlock check	46
4.4	Property verification	48
5	Conclusion	53

1 Introduction

Before starting this tutorial, it is important to understand the basic concepts used in PragmaDev Process. These concepts derive from the language supported by PragmaDev Process, i.e., **BPMN**.

BPMN stands for **Business Process Model and Notation**. BPMN is a graphical language defined by the Object Management Group (OMG). Its primary goal is to provide a notation that is understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. In doing so, BPMN provides a simple mean of communicating process information to other business users, process implementers, customers, and suppliers.

The following are the most important elements of a BPMN diagram in PragmaDev Process:

- A **Collaboration** is a collection of *Participants* shown as **Pools**, their interactions shown by **Message Flows**, and may include **Processes** within the **Pools**.
- A **Lane** is a sub-partition within a **Pool** that is used to organize and categorize **Activities** of a **Process** within a **Pool**.
- A **Process** describes a sequence or flow of **Activities** in an organization with the objective of carrying out work. In BPMN a **Process** is depicted as a graph of flow elements, which are a set of **Activities**, **Events**, **Gateways**, and **Sequence Flows**.

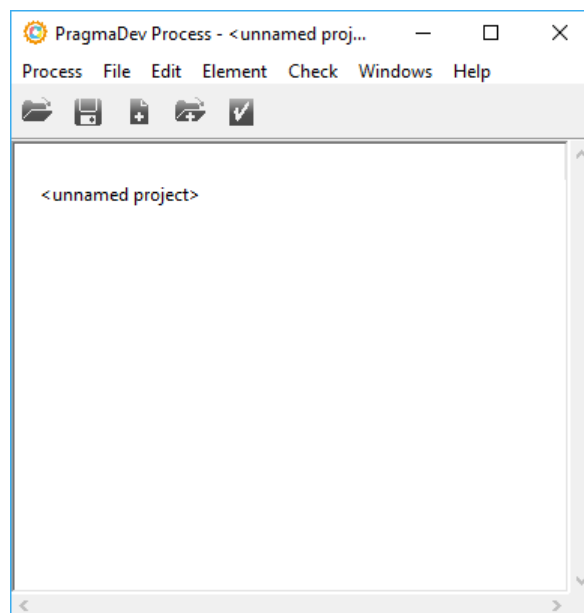
2 The model

The model we have chosen is simple enough to be written from scratch but rich enough to pinpoint the basics of BPMN. It is a pizza delivery model composed of a pizza *Vendor* and a *Customer*. *Vendor*'s activities are categorized in activities carried out by the *Clerk*, *Chef*, and *Delivery Boy*. Upon receiving an order from the *Customer*, the *Clerk* will forward such order to the *Chef*. The *Chef* will then bake the pizza and hand it over to the *Delivery Boy*. At last the pizza will be delivered to the *Customer*.

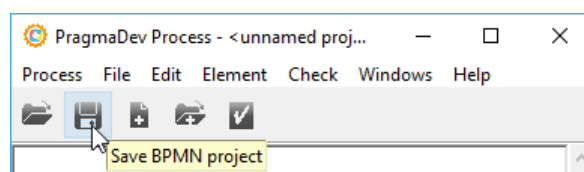
The BPMN model of the pizza delivery used in this tutorial is found in \$PRAGMADEV_PROCESS_HOME/examples/Pizza.

2.1 Organization

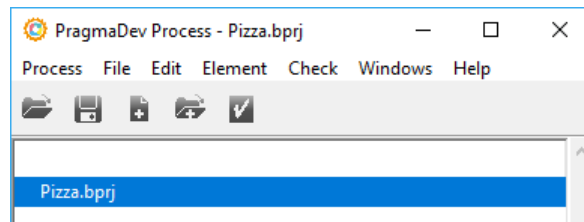
Let's get our hands on the tool! Start PragmaDev Process. The window that appears is called the Project Manager:



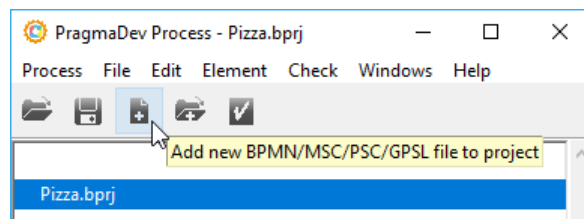
Save the project via the button in the toolbar:



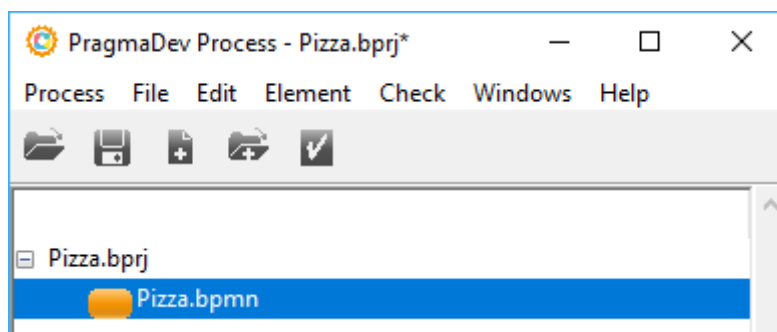
Name the project Pizza:



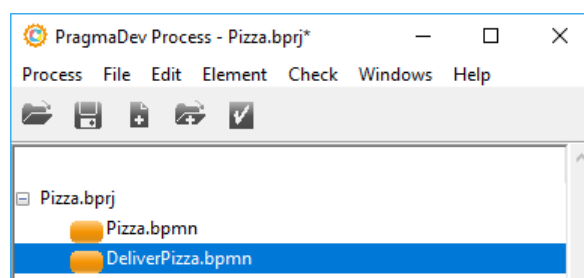
Create and add a new BPMN file to the project via the button in the toolbar:



Name it Pizza:

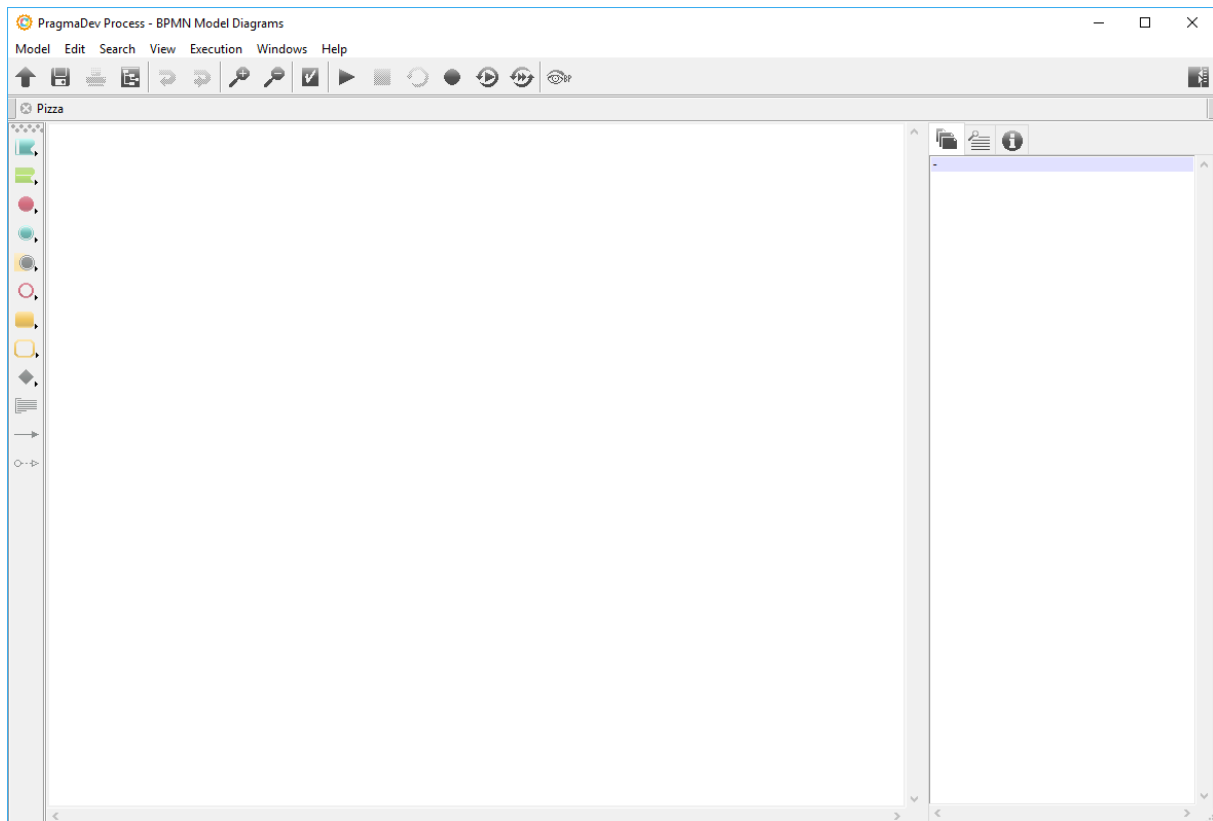


In the same way create and add another BPMN file named DeliverPizza to the project:

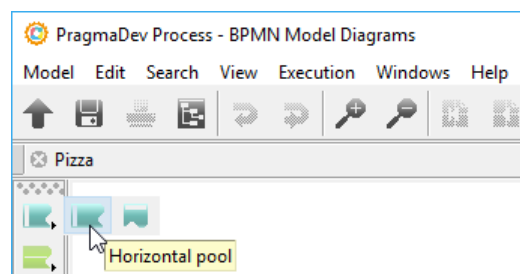


2.2 Pools and lanes

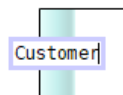
Double click `Pizza.bpmn` in the Project Manager to open it in the BPMN Editor:



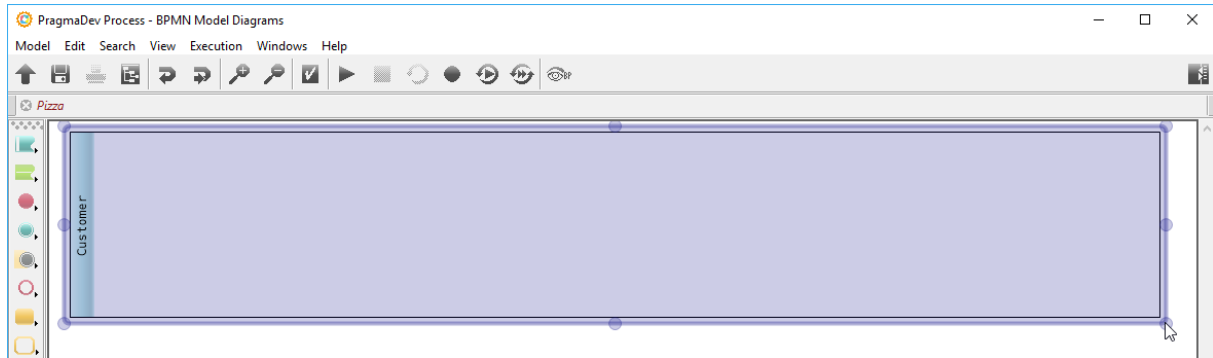
Create a horizontal pool via the tool button:



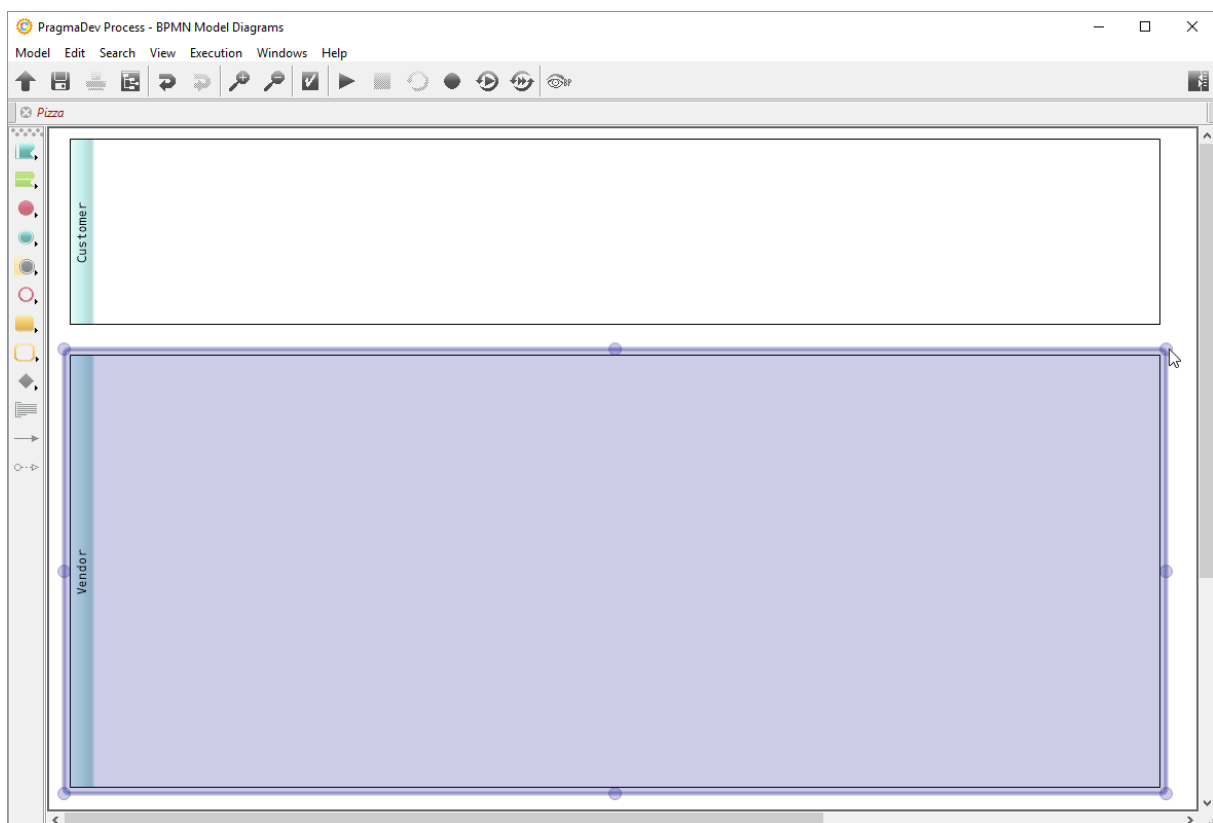
and name it Customer:



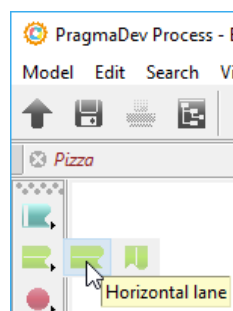
Resize the pool using the points to create space for other elements:



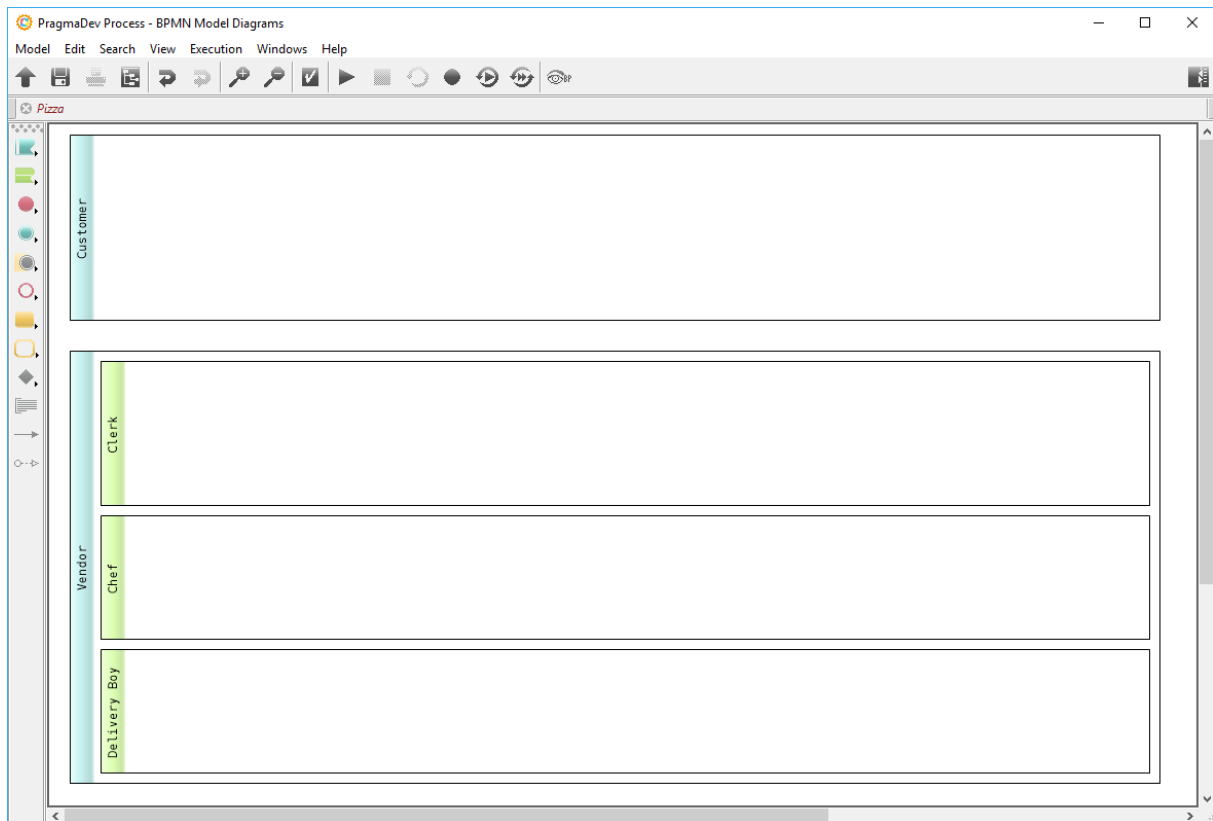
Create another pool named Vendor and resize it:



Create three horizontal lanes inside the Vendor using the tool button:

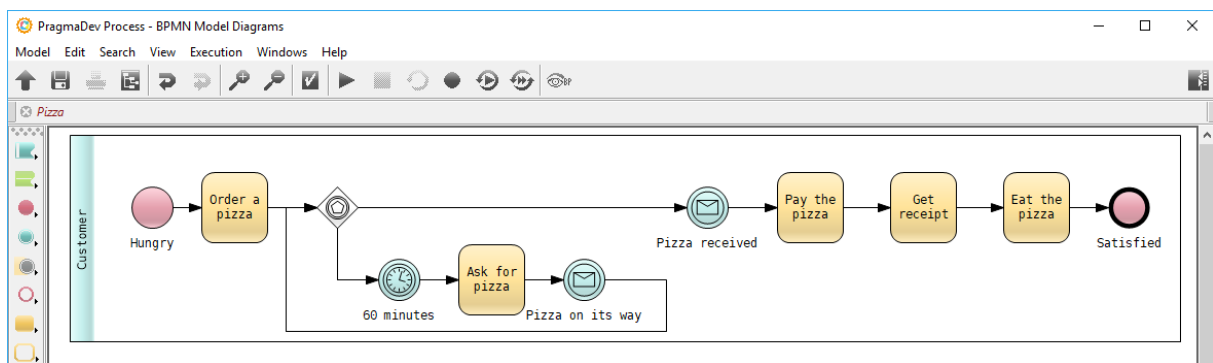


Name them Clerk, Chef, Delivery Boy, and resize them as shown:



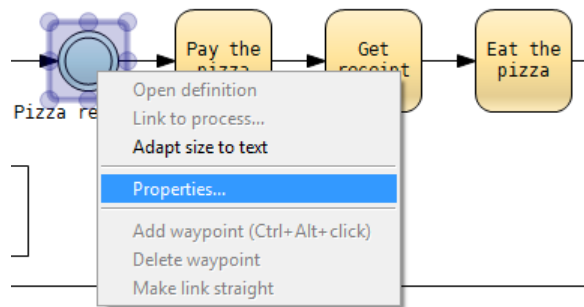
2.3 Processes

Use the tool buttons to draw the Customer process as shown:

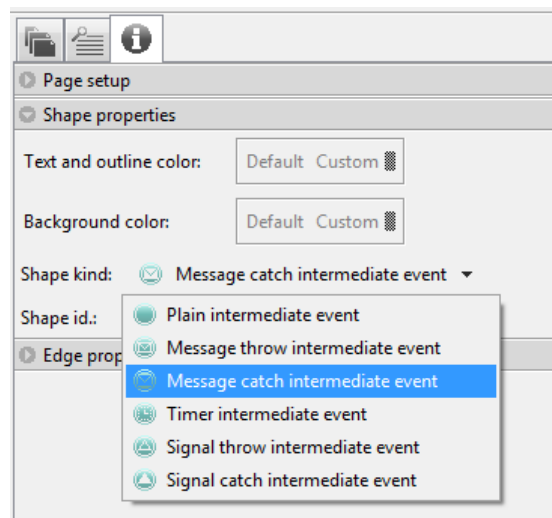


Symbols can either be created with the tools in the toolbar, or via keyboard shortcuts. The insertion shortcuts are all introduced by Control + Space (or Command + Control + Space on macOS), followed by a letter giving the type of symbol to create. For example, to create a start event, press Control + Space followed by "s"; with "t", it creates a task, with "i" an intermediate event, with "g" a gateway, and so on. To see the available letters, you can press Control + Space, then "?".

Symbols created with the keyboard shortcuts will always be "plain" ones. Their type can however be changed afterwards by opening the symbol properties:

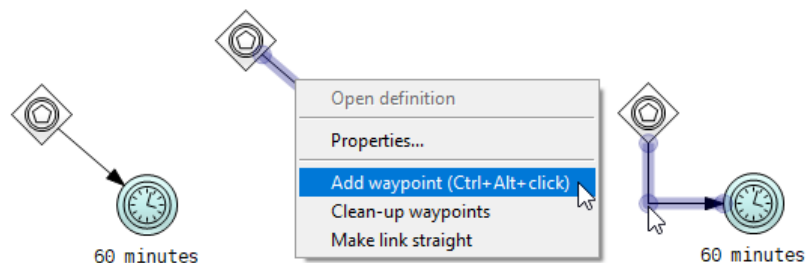


then changing its type in the right-hand side panel in the editor window:



Links can also be created via a Control + Space sequence: with Shift + s, it creates a sequence flow and with Shift + m a message flow. Sequence flows can also be created automatically from the previously selected symbol to the newly inserted one if the option *Automatically create sequence flows* is checked in the *Edit* menu.

Sequence (or message) flows are drawn as a straight line by default. To create non-linear paths, right click a flow to *Add waypoint*, and move the waypoint to the desired position:

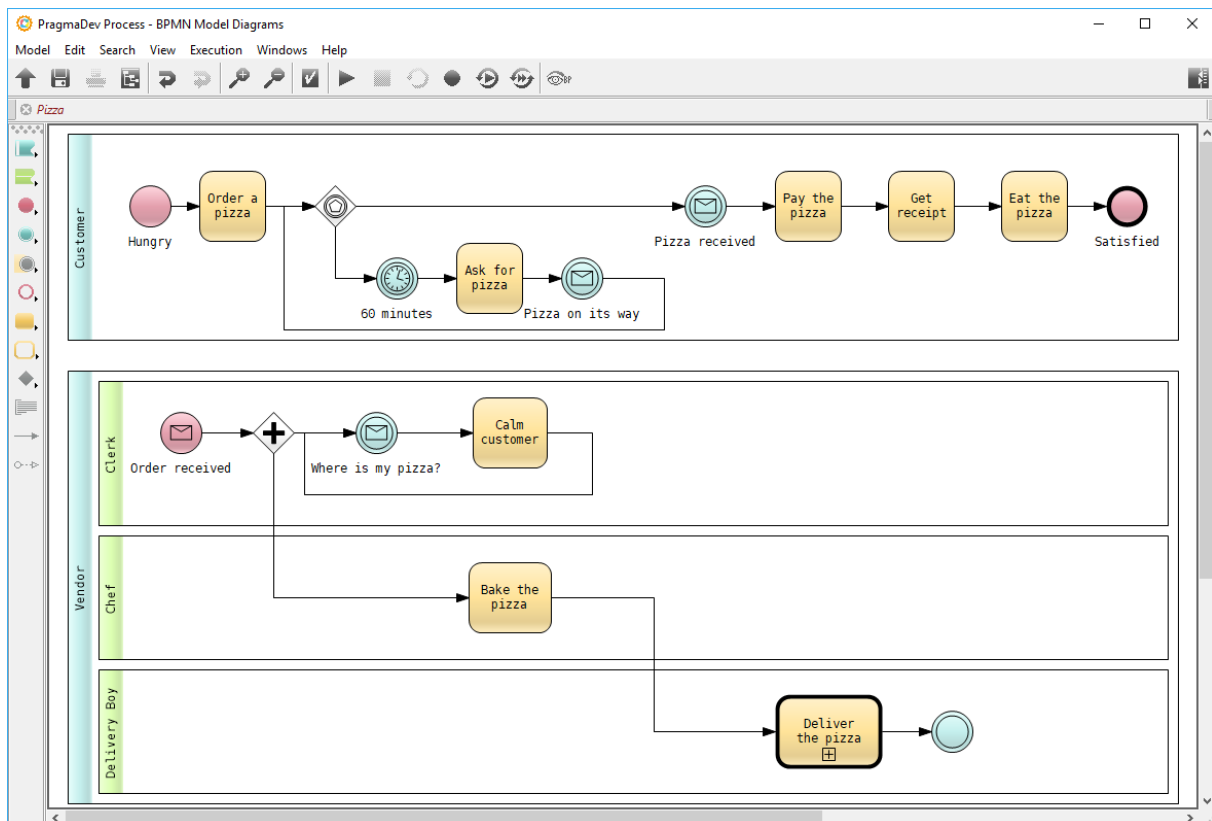


Non-linear paths can be created also by holding *Shift* and clicking to place waypoints while drawing the path from the source to the target symbol.

The Customer process says the following:

- The Customer is Hungry for pizza.
- He/She decides to Order a pizza.
- The Customer will wait until his/her pizza is delivered (Pizza received).
- If the pizza is not delivered within 60 minutes, the Customer will check what's going on with his/her pizza (Ask for pizza).
- Upon receiving the pizza, the Customer will Pay the pizza, Get receipt, and finally Eat the pizza to be Satisfied.

Draw the Vendor process as shown:



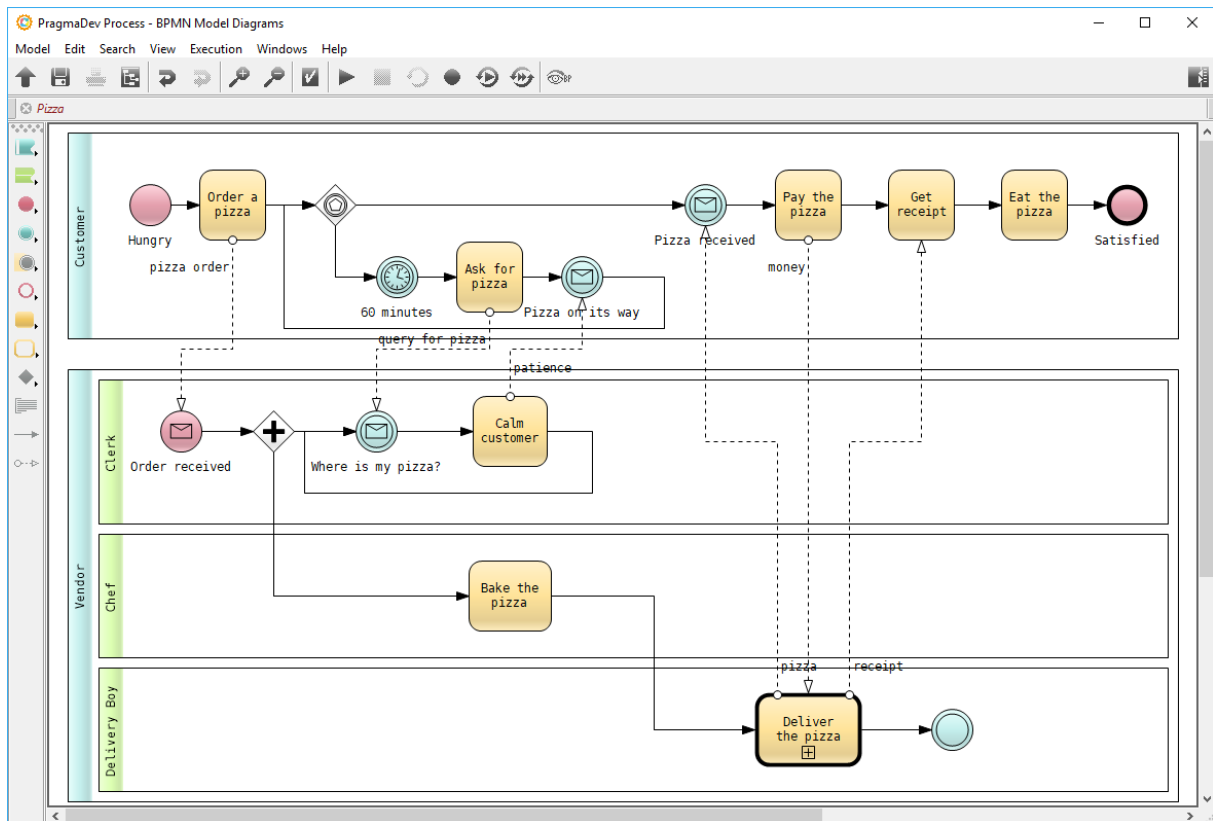
The Vendor process says the following:

- Upon receiving an order from the Customer, the Clerk will enter a loop waiting for "complaints" (Where is my pizza?), and in case of a "complaint" he/she will try to Calm customer.
- The Clerk will also forward the received order to the Chef, so that he/she can Bake the pizza.
- When ready, the Chef will hand over the pizza to the Delivery Boy so that he can Deliver the pizza.

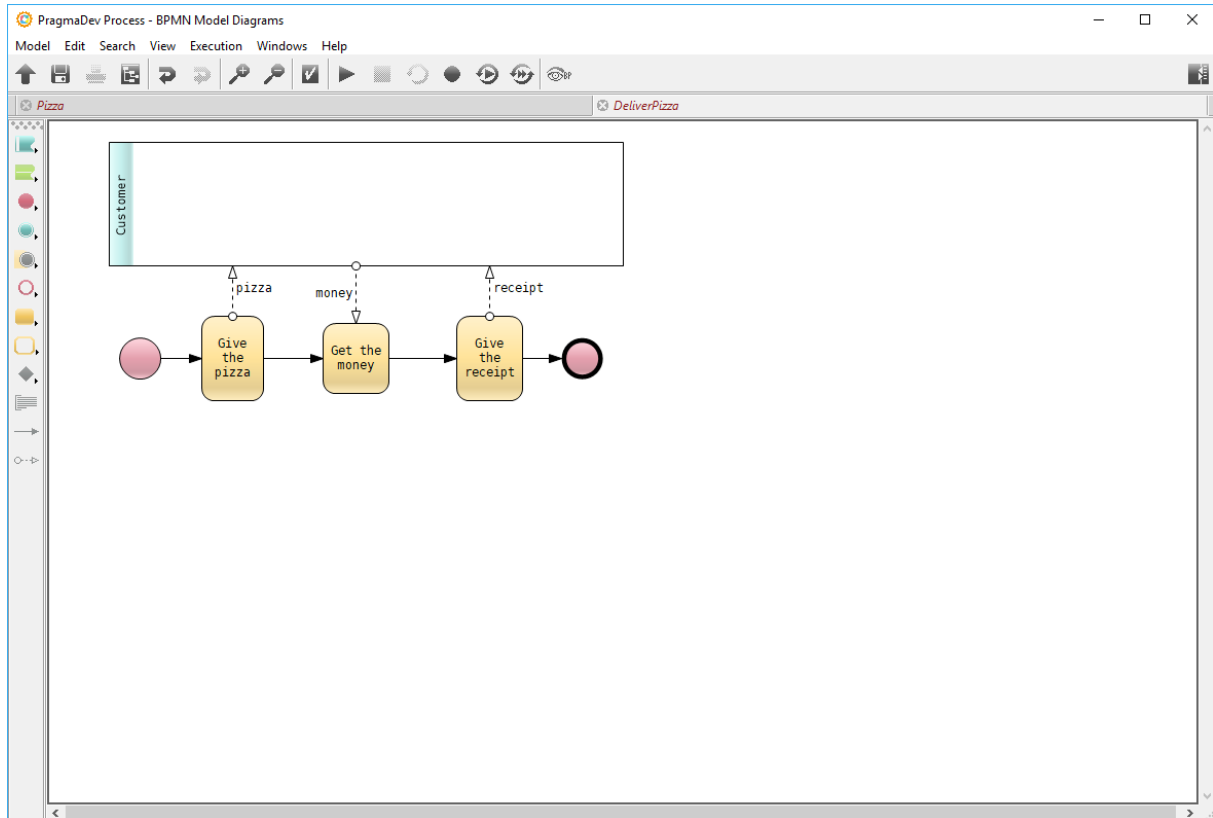
Notes:

- Deliver the pizza is a *Process call activity*.
- The process ends with an intermediate event. This error is intentional, and will be corrected later in the tutorial.

To complete the Customer-Vendor *collaboration* draw the following message flows:

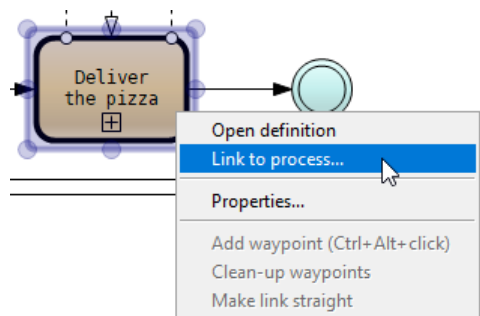


Let's define now the call-activity **Deliver the pizza**. In the Project Manager double click **DeliverPizza.bpmn** to open it in the editor. Draw the following:

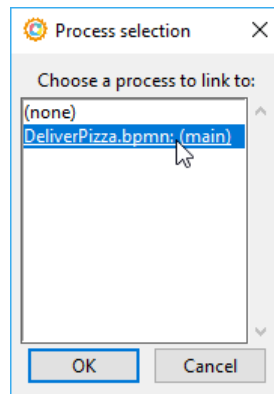


When creating the message flows be sure to draw starting from / ending to either the border or header of the pool Customer.

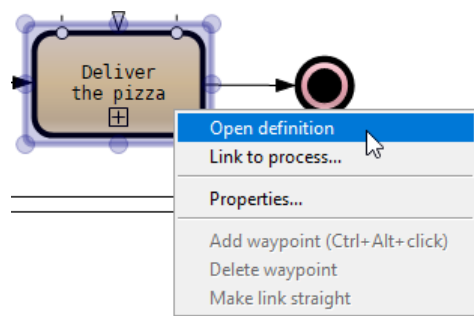
Save the diagram and go back to the Pizza.bpmn. Right click the call-activity and then *Link to process...*:



Select the single available option (except *none*) in the dialog and hit *OK*:



Right click the call-activity and then *Open definition*:



This should display the `DeliverPizza.bpmn` in the editor.

The model of the pizza delivery is now complete. Save everything before closing the editor.

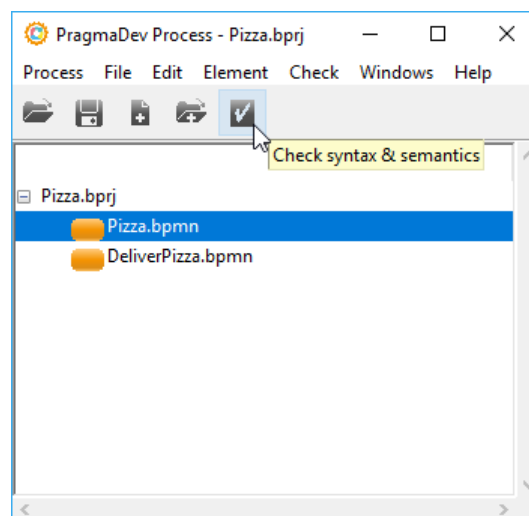
3 Execution

Now we will execute the model using PragmaDev Process' BPMN Executor. There are two kinds of execution:

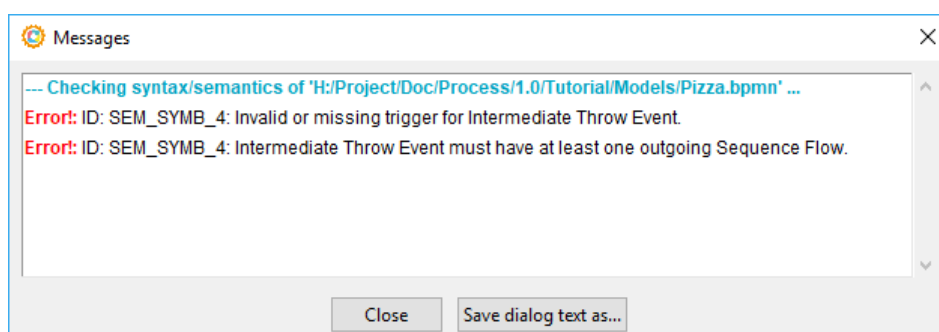
- *Interactive*: allows the user to have full control over the execution. The user interacts with the model at every step of its execution.
- *Automatic*: enables model execution without user input. The execution is guided by MSC traces.

3.1 Semantic check

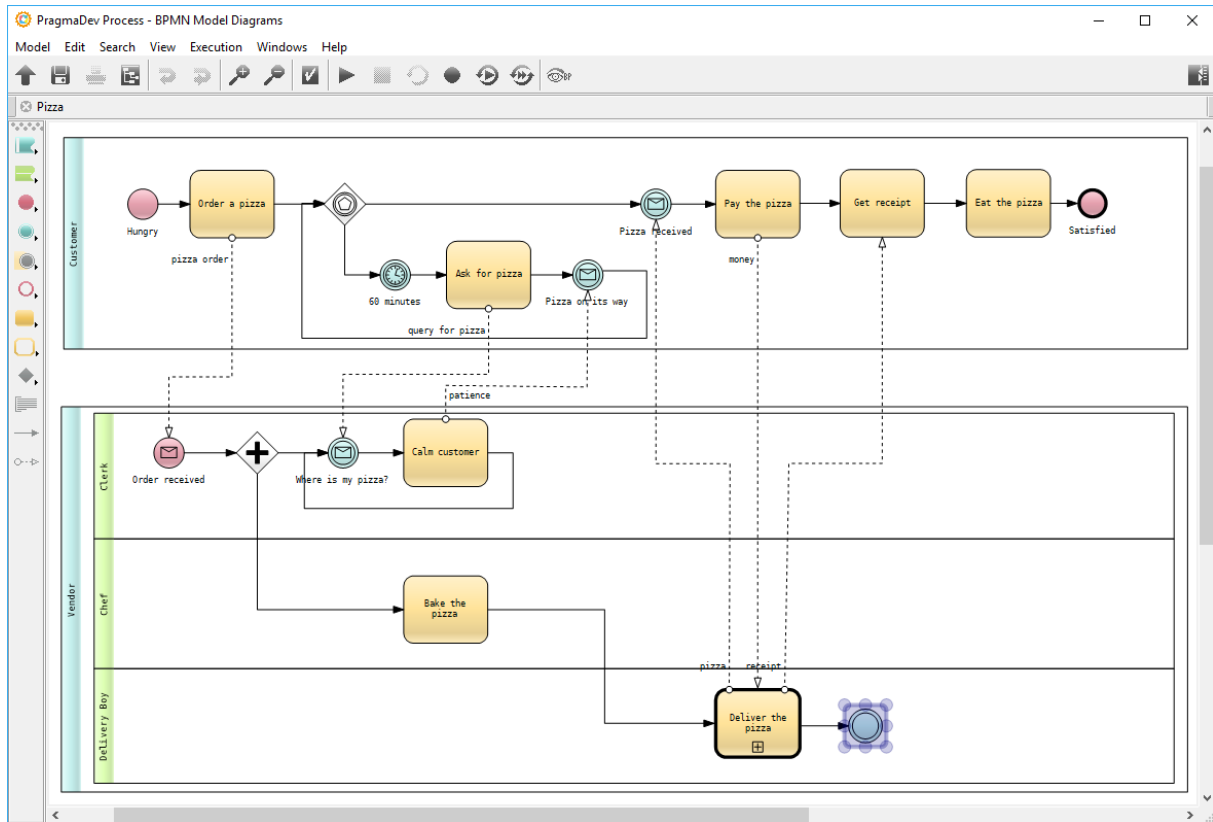
The model can be executed only if it conforms to BPMN 2.0 semantics. To make sure this is true, select *Pizza.bpmn* in the project manager and click the *Check syntax & semantics* button:



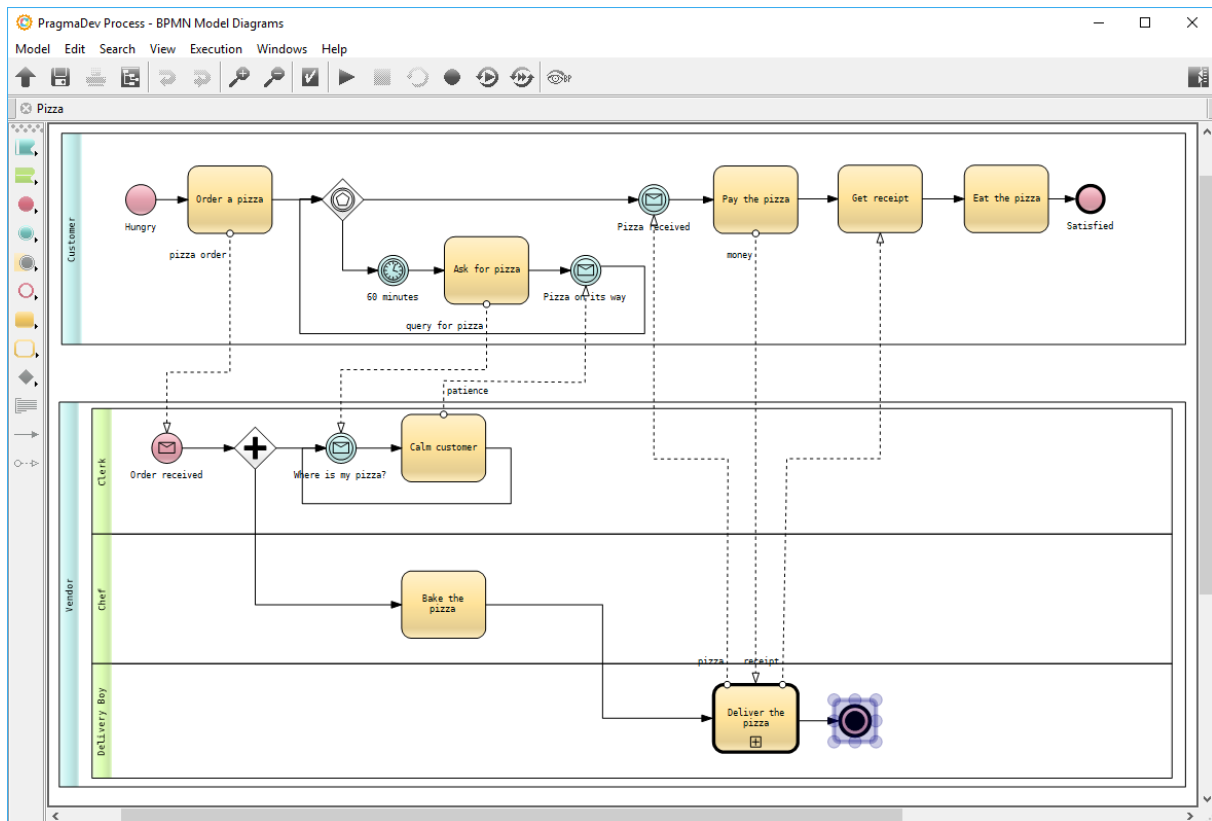
A window will pop-up showing the result of the semantic check on the model:



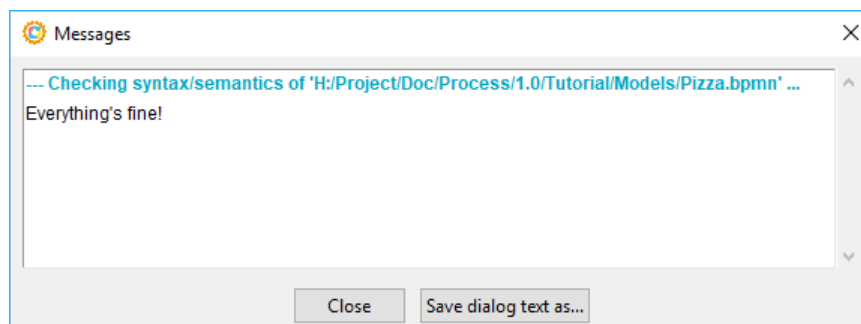
Double-clicking either of the listed errors will open the model in the editor and the concerned element will be selected:



Replace the intermediate event with a terminate end event to correct the errors:

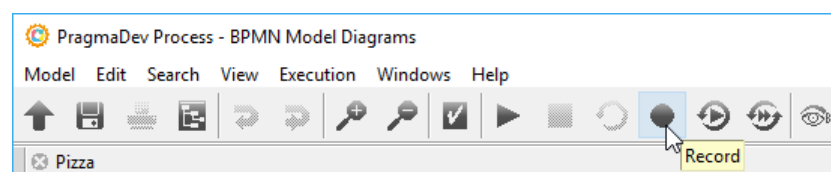


Re-running a semantic check on the model should not list any errors:

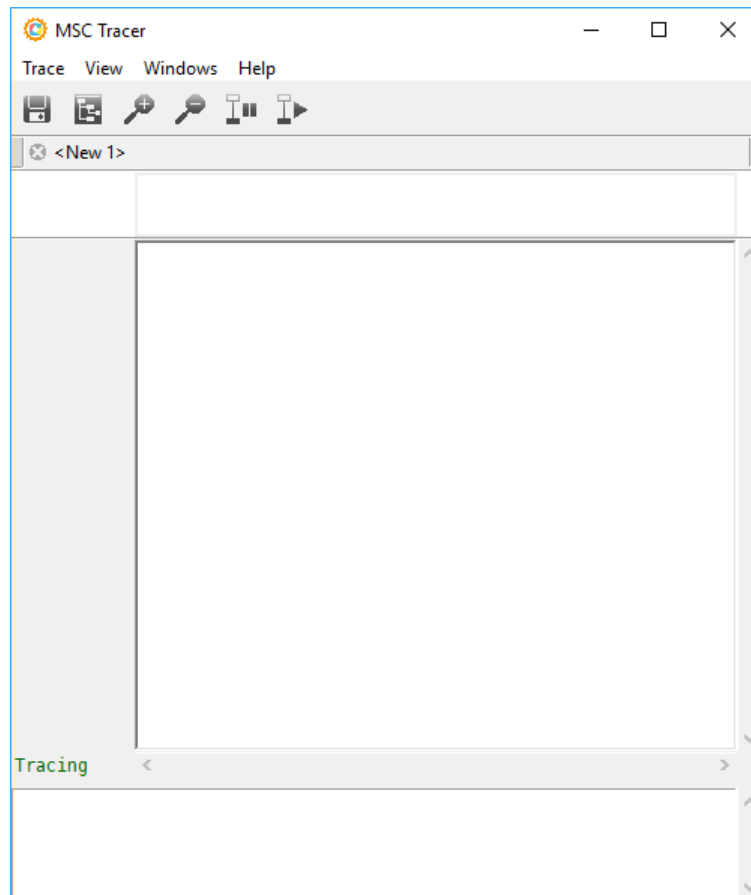


3.2 Interactive execution

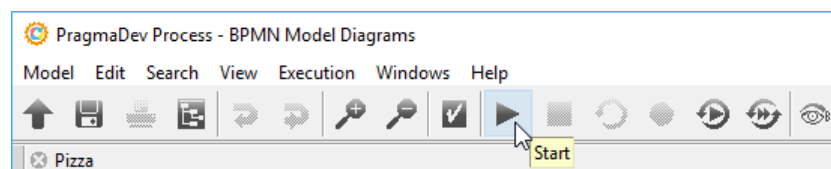
With `Pizza.bpmn` opened in the editor (double-click on it in the project manager), click the *Record* button:



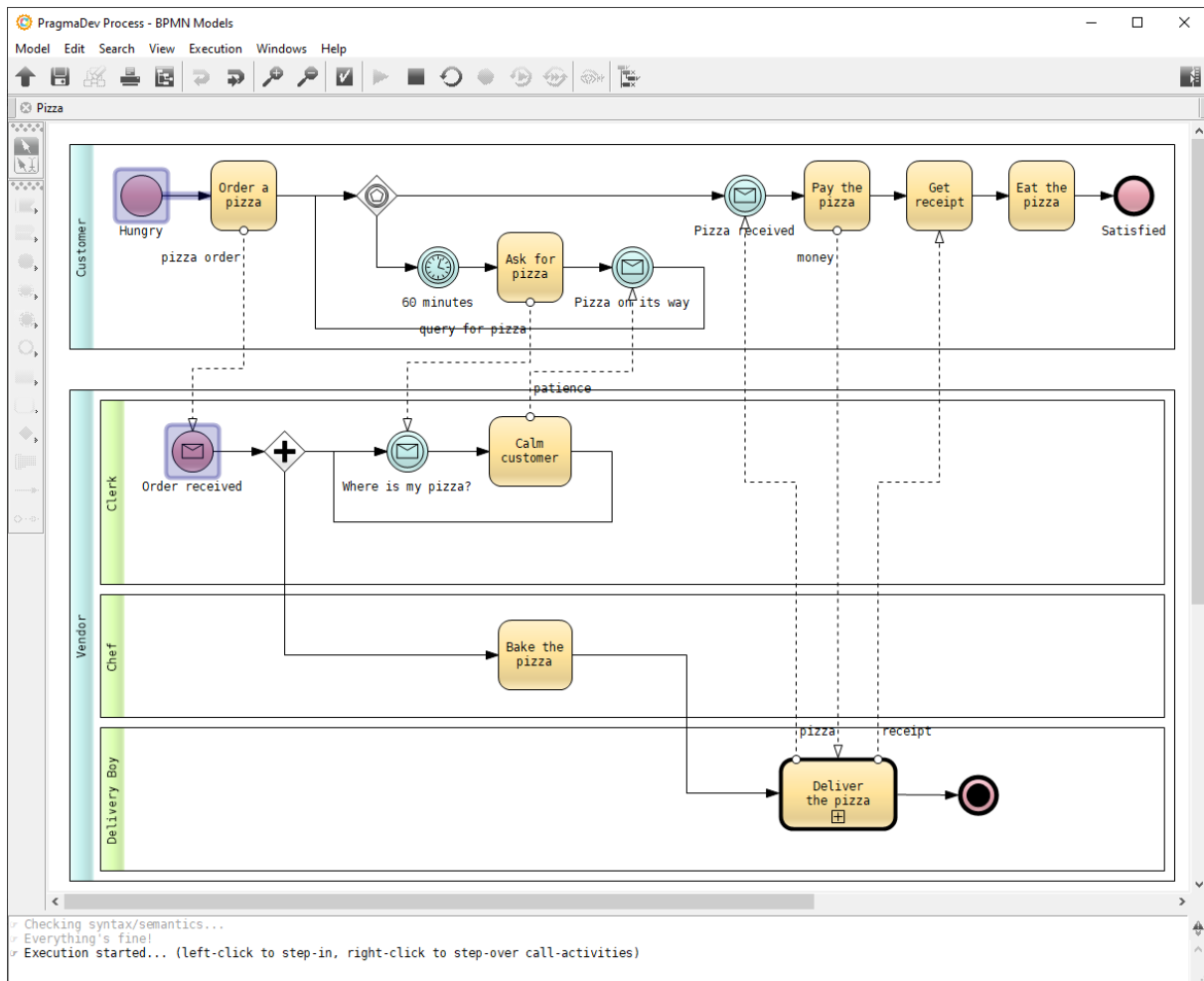
This will allow us to record the execution in the *MSC Tracer*:



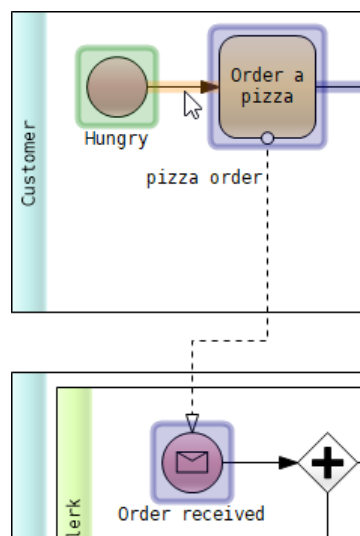
We can now start the execution via the *Start* button:



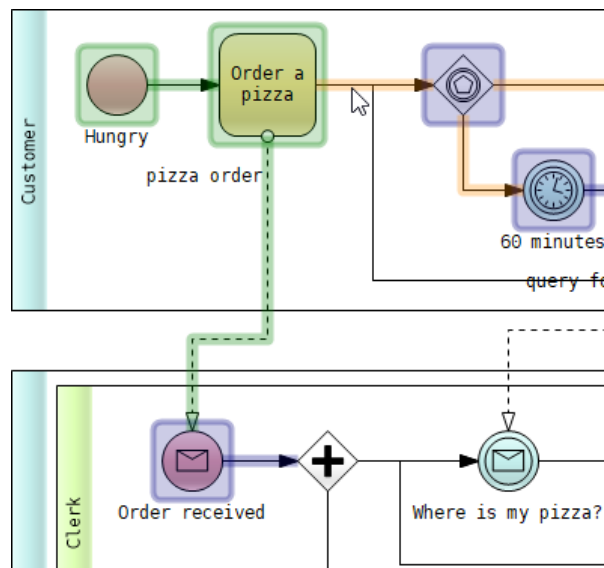
Start events and executable flows will be marked in blue:



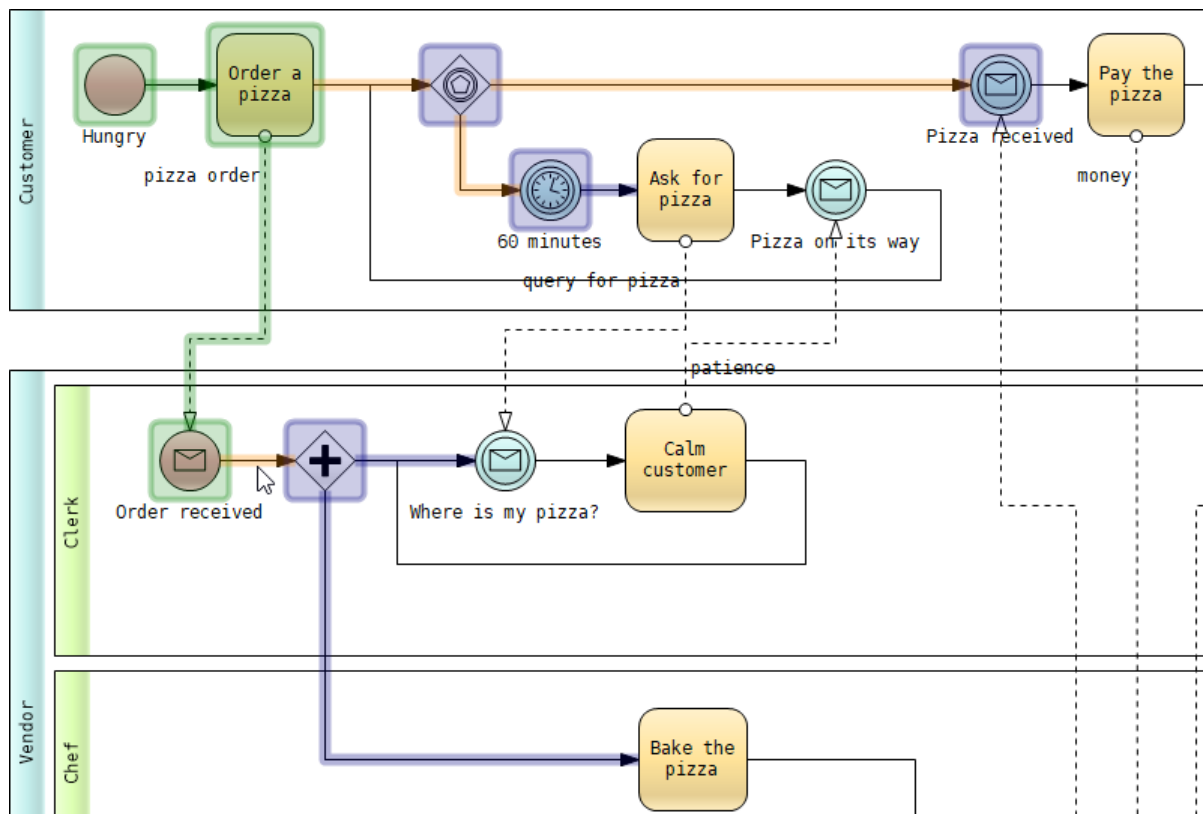
Please note that generally speaking only sequence flows, some message flows, or call-activities can be clicked. No other symbol can be clicked even though they appear in blue. In our example there is a single sequence flow that can be executed. Clicking on it will advance execution, i.e., make the Customer order a pizza:



The next (and only) step will be for the Customer to send the pizza order to the Clerk; to continue click the sequence flow:



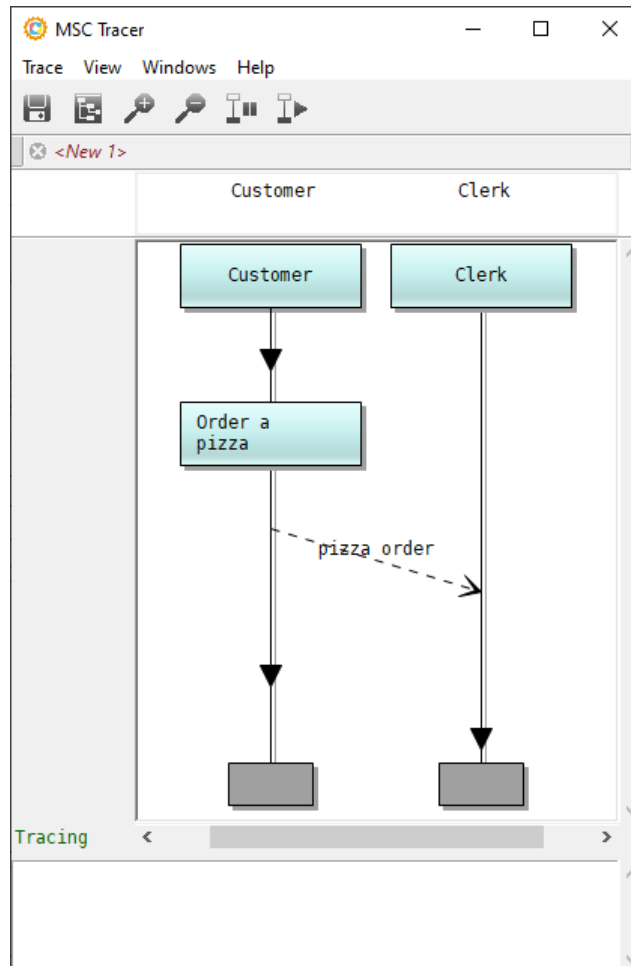
The pizza order will start the Vendor process. Clicking the sequence flow will result in:



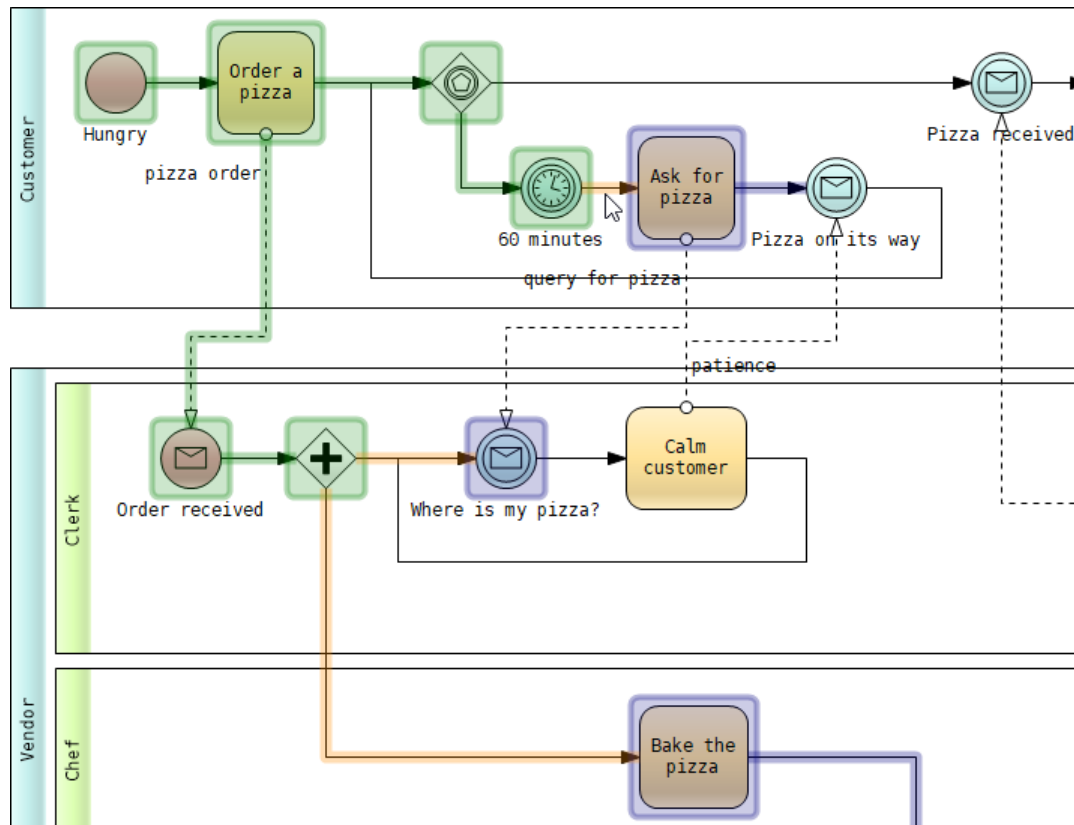
By entering the event gateway, the Customer will wait for its pizza to be delivered (i.e., the Pizza received event) and start a 60 minutes timer. If the timer fires before the

pizza is delivered, the Customer will ask the Clerk for the pizza. On the other hand, the parallel gateway in the Vendor process will allow the Clerk to listen to any Customer queries, while the Chef starts baking the pizza.

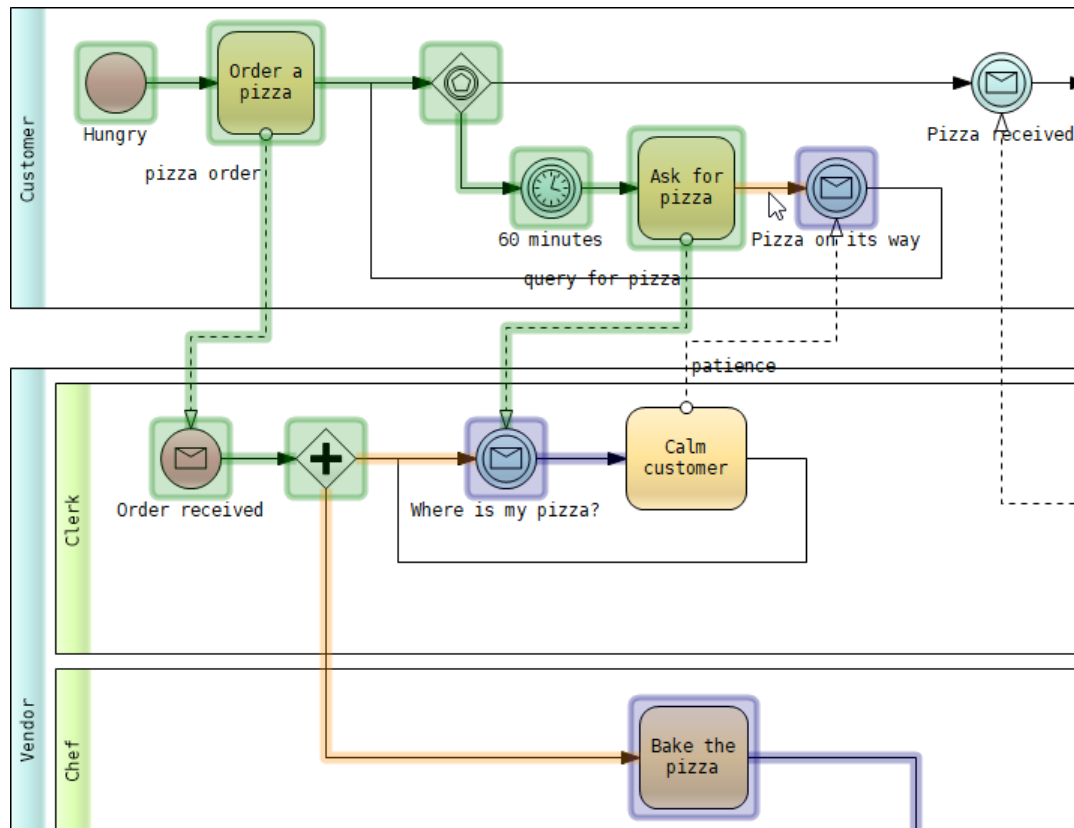
While execution advances in the editor, all steps are also being recorded in the *MSC Tracer*:



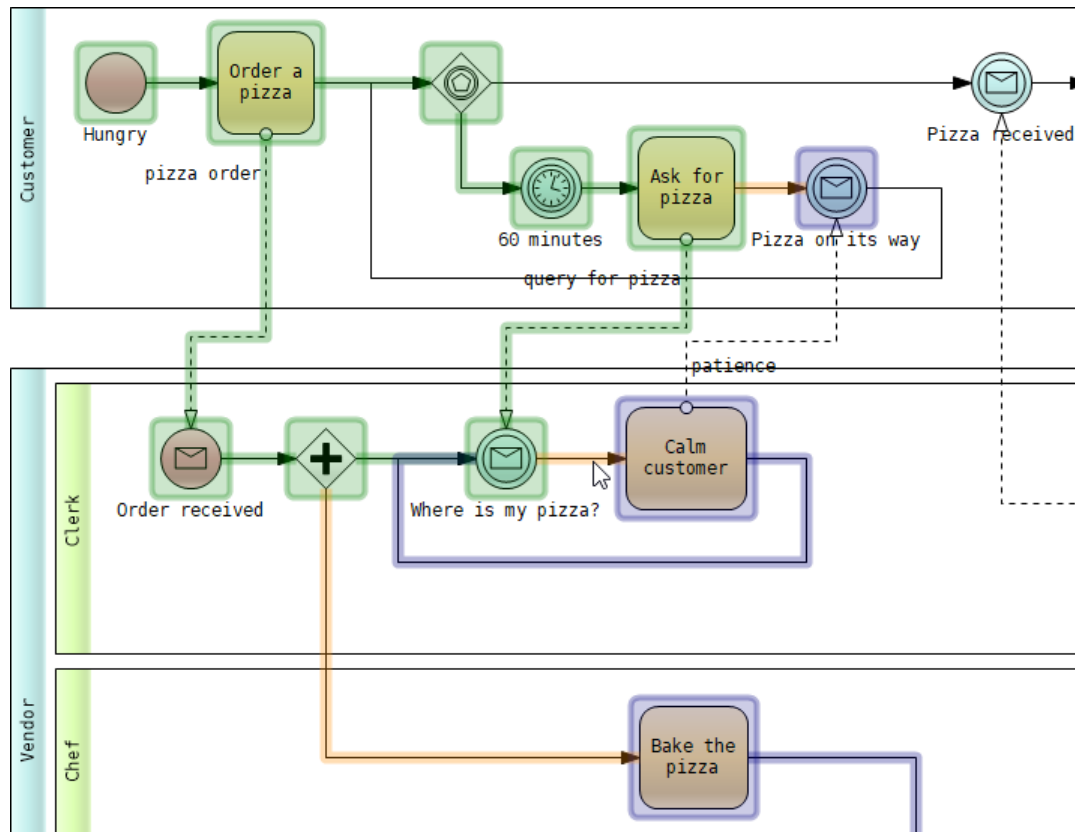
Let's make the timer fire, the Clerk listen for Customer queries, and the Chef start baking the pizza by clicking all three sequence flows:



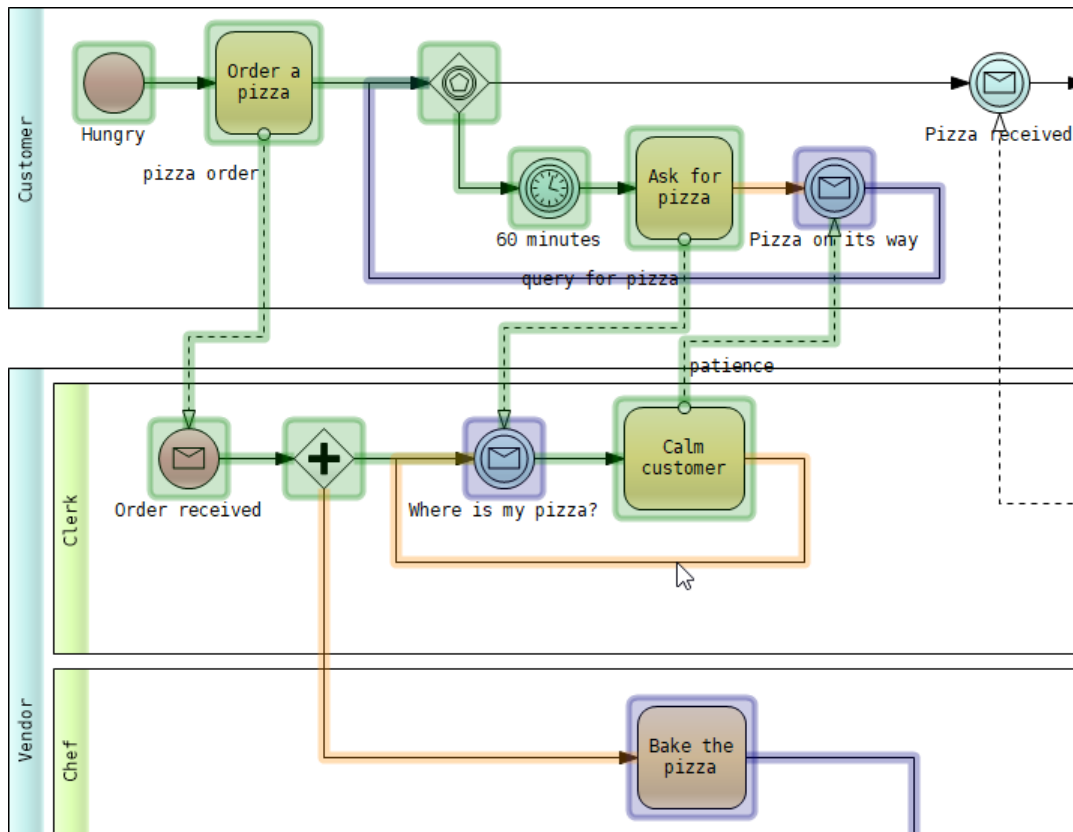
Notice that the Pizza received event is no longer active because the 60 minutes event was triggered. In this situation the Customer is asking the Clerk for his/her pizza, while the pizza is being baked by the Chef. If query for pizza is sent:



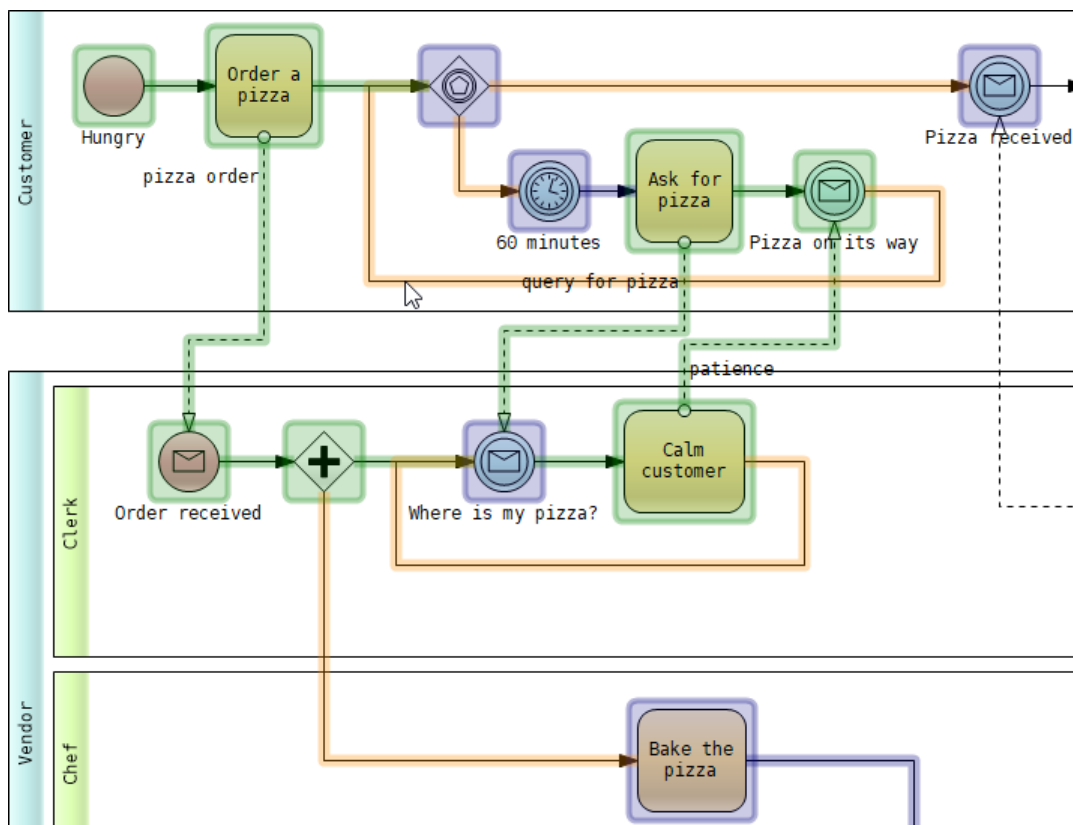
By clicking the sequence flow, the Clerk will Calm customer while the later will be waiting for a feedback for his/her query (i.e., the Pizza on its way event):



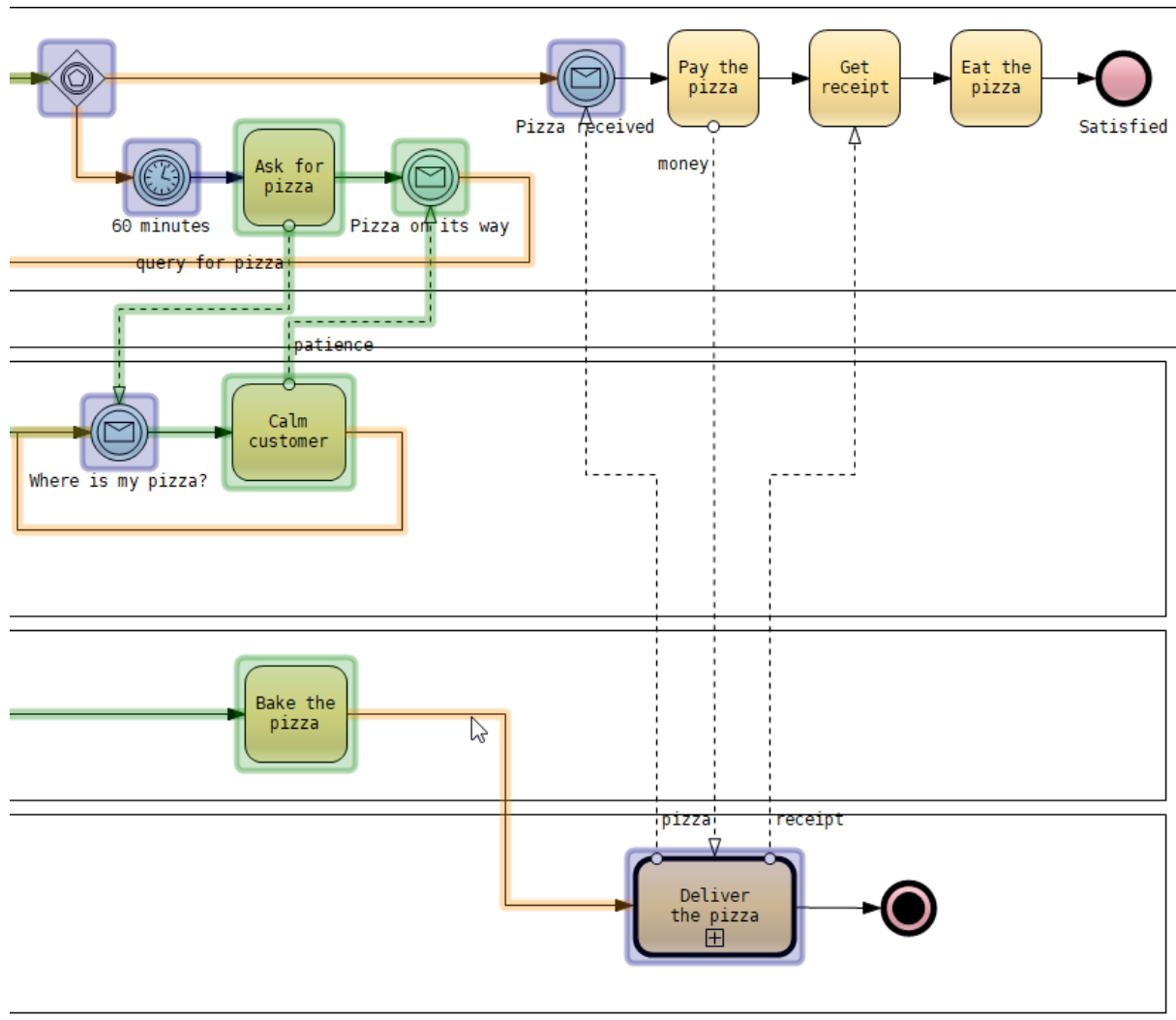
Make the Clerk respond to the Customer's query by clicking the sequence flow which will automatically send the patience message:



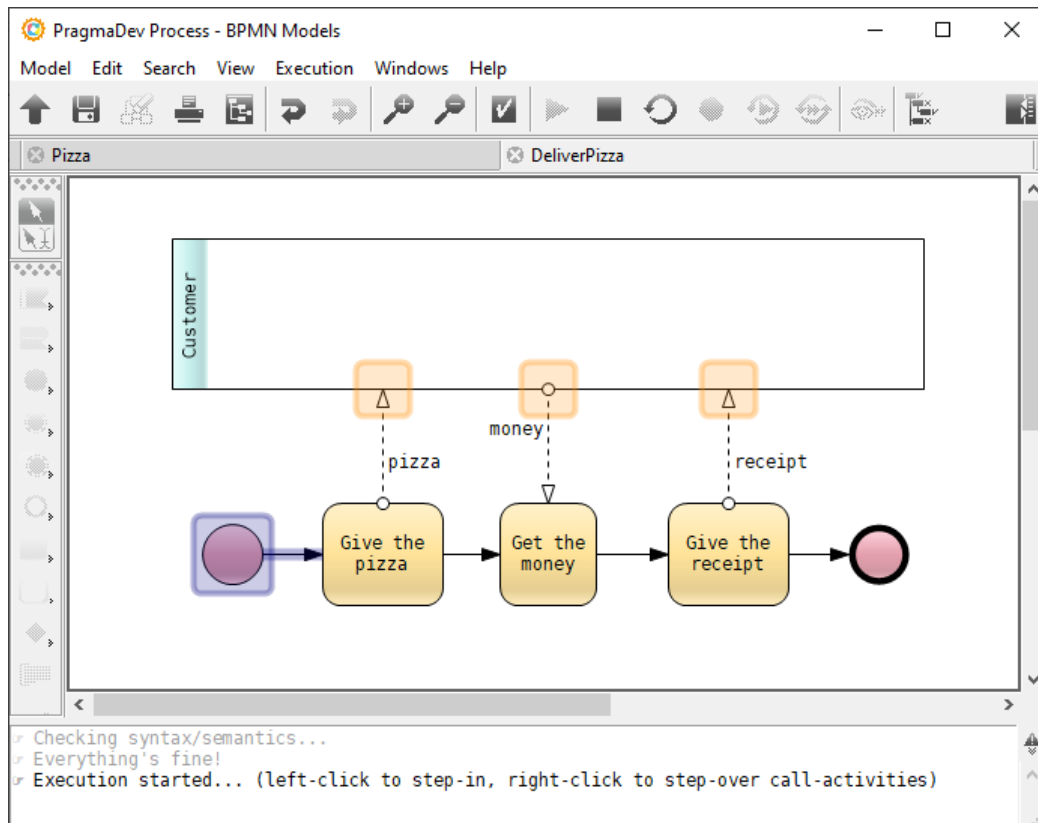
Continue by clicking the looping sequence flow in the Customer process:



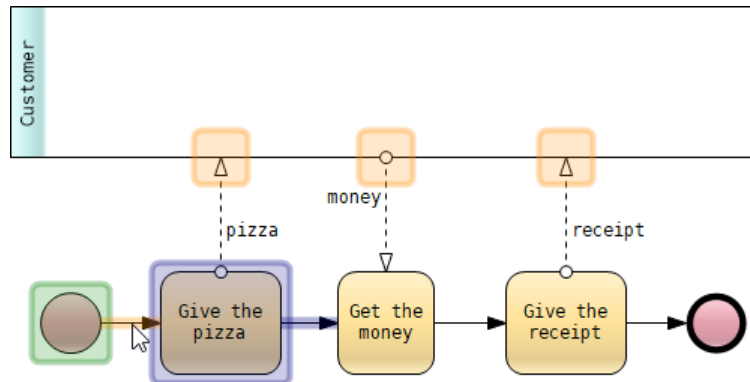
The Customer will get back to waiting for the pizza or for another 60 minutes to pass before querying the Clerk again. We will not let the Customer wait any longer, so let's make sure the pizza is ready by clicking the sequence flow outgoing Bake the pizza:



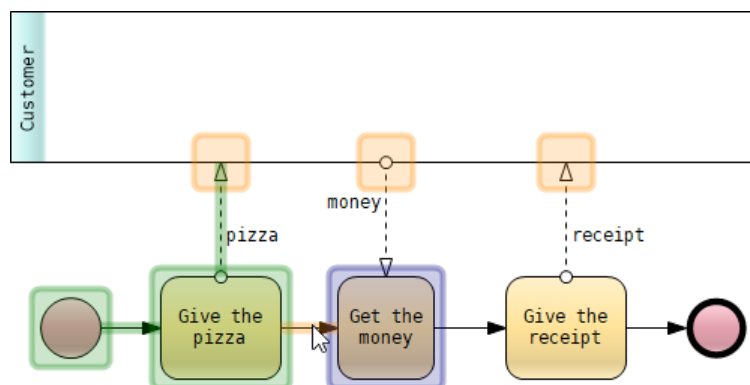
Now it's time to Deliver the pizza. This is a BPMN call-activity which calls the process defined in `DeliverPizza.bpmn`. We can either step-over (right-click) or step-in (left-click) the call-activity. Stepping over the call-activity will not execute the called process, but will consider the activity as being a simple BPMN task. In this case the execution will block because the activity will wait for the receipt message before sending the pizza message. Stepping in the call-activity will open `DeliverPizza.bpmn` in the editor and start the called process:



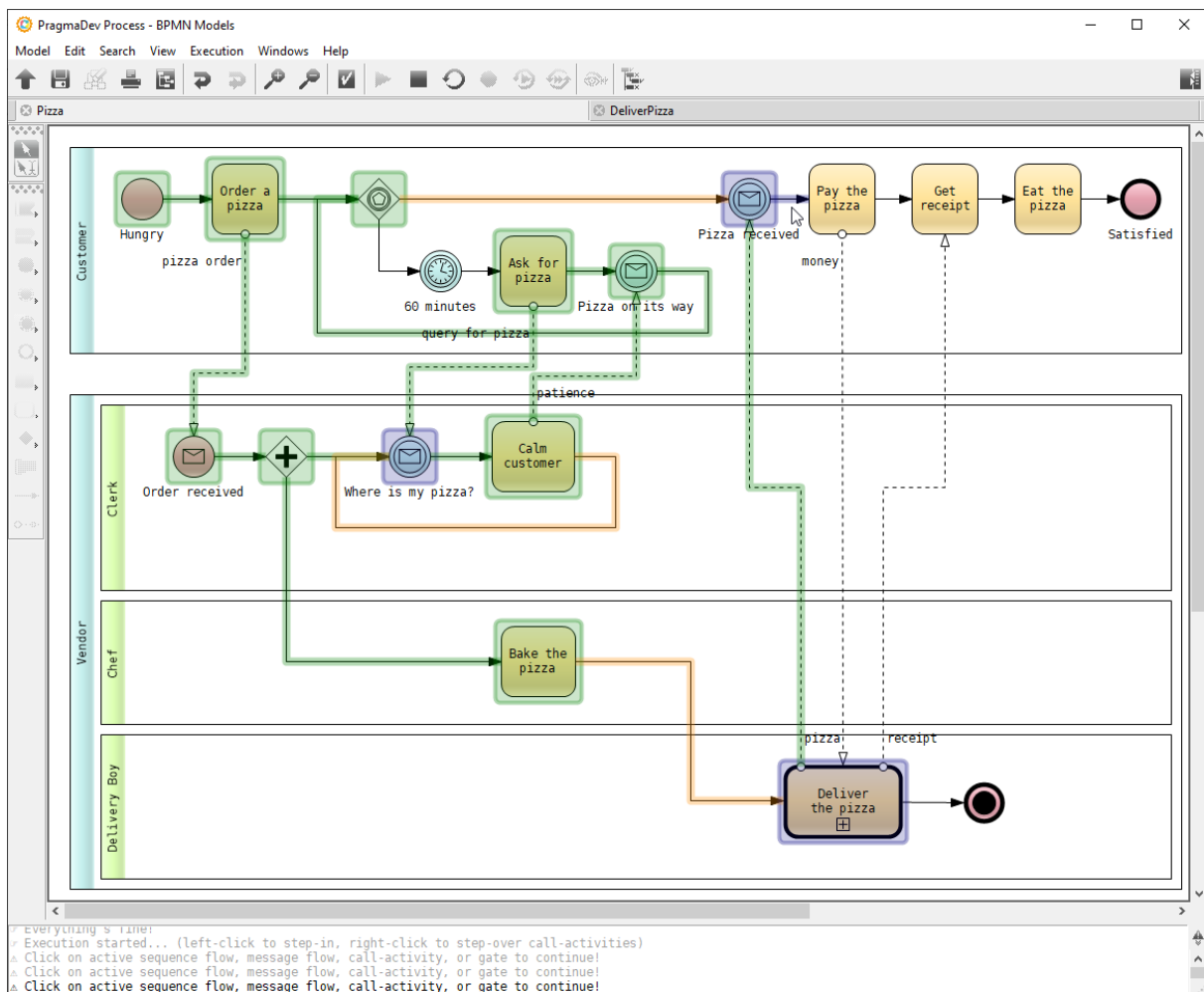
We can Give the pizza to the Customer by clicking the sequence flow:



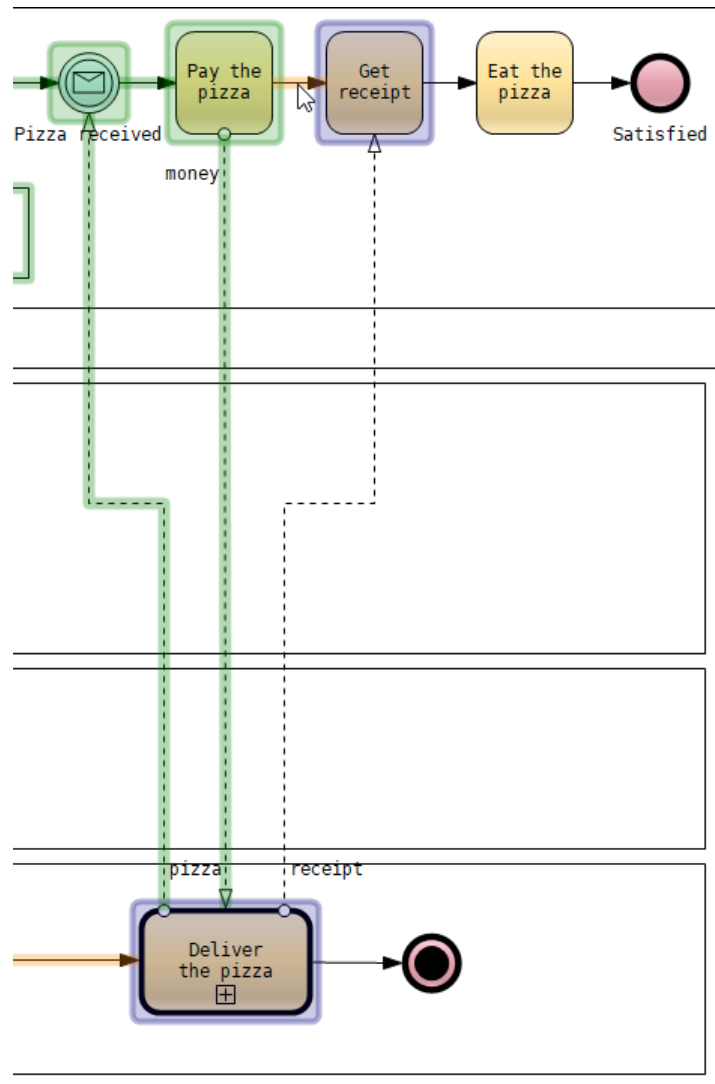
and then the following sequence flow which will automatically send the pizza message:



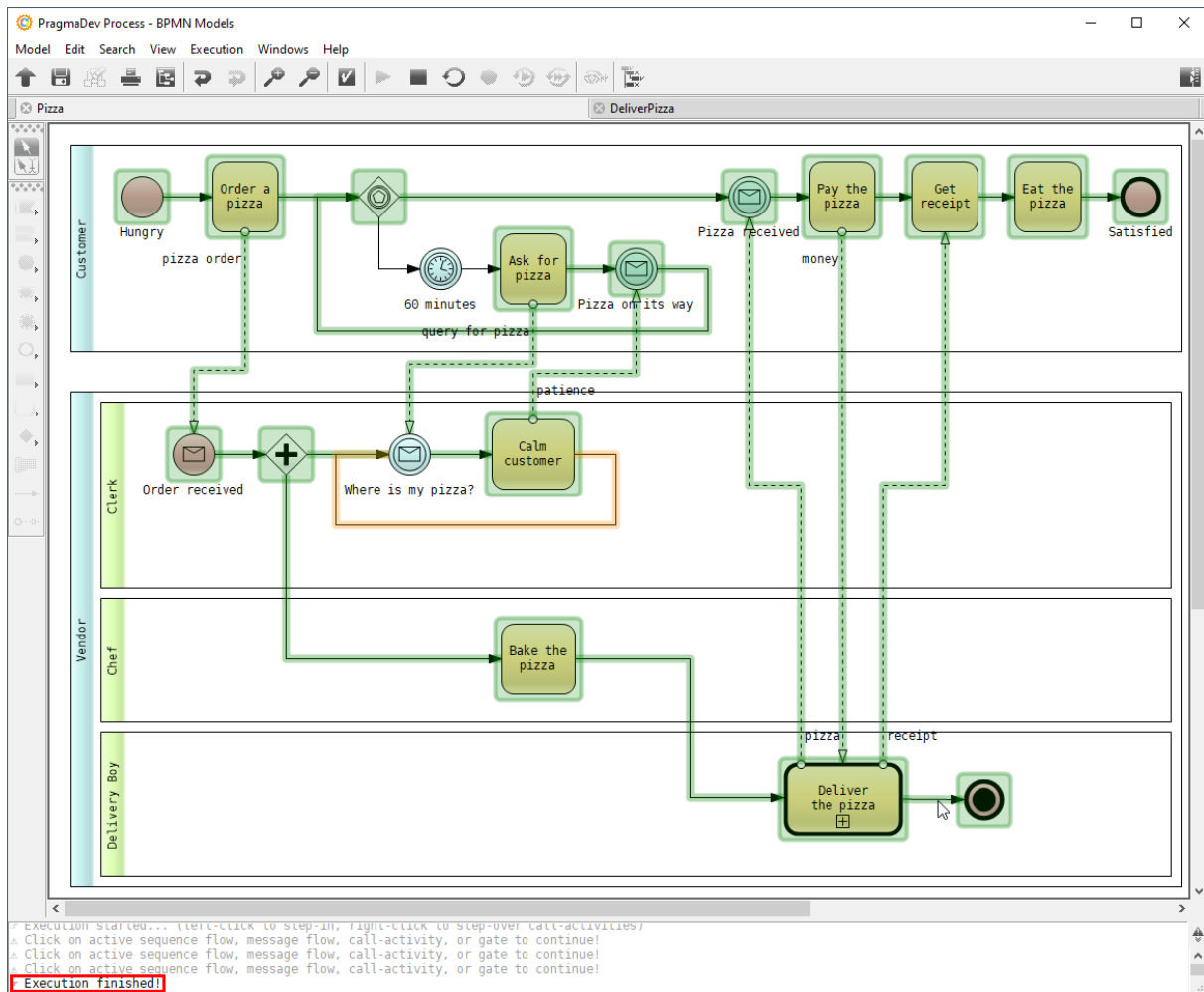
If we switch to the Pizza.bpmn in the editor we can see that the Customer has indeed received the pizza and can proceed with the payment:



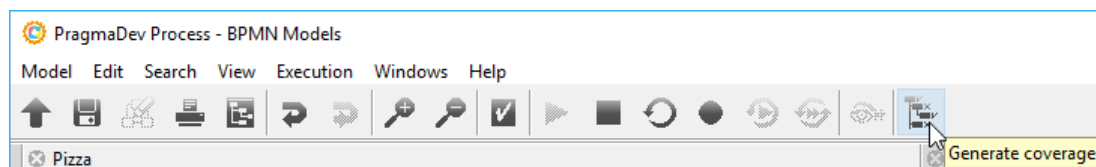
Click the next two sequence flows to give the money to the Delivery boy:



Continue by clicking all possible flows in both diagrams until there is nothing left to do, i.e., execution is finished:



To generate model coverage click the *Generate coverage* button:

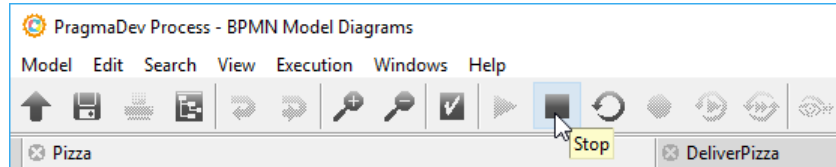


Coverage information will be shown as follows:

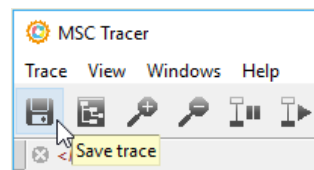
Diagram/element	Hits
Pizza.bprj	1 - 2
Pizza.bpmn	1 - 2
DeliverPizza.bpmn	1

Coverage information helps to identify complementary scenarios in order to verify the process. Please note several coverage information can be merged to make sure a set of scenarios do actually cover all symbols.

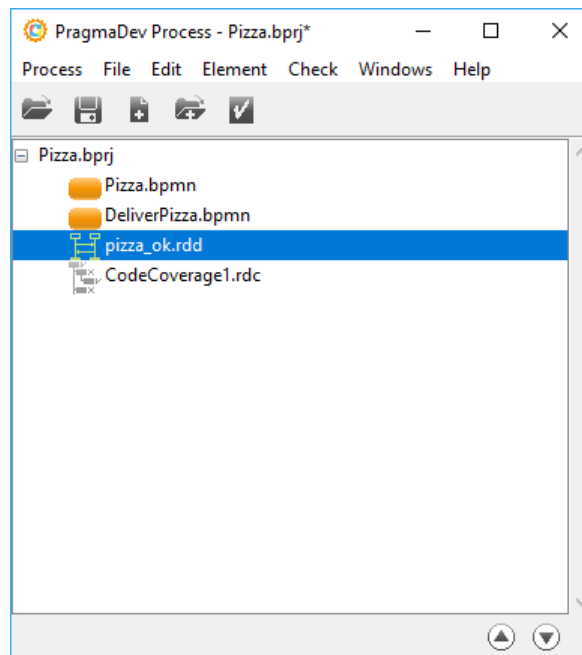
Close the coverage window and stop the execution via the *Stop* button:



Save the recording via the *Save trace* button in the *MSC Tracer*:



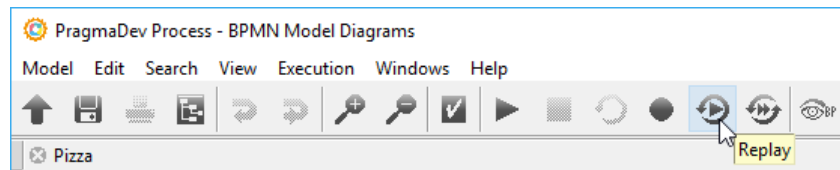
Give it a name (e.g., `pizza_ok`), and it will be added automatically to the project:



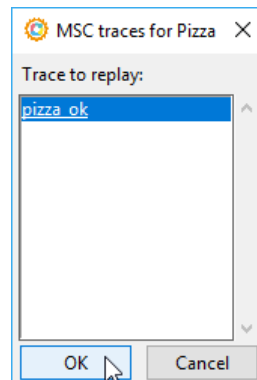
3.3 Automatic execution

3.3.1 Single-trace execution

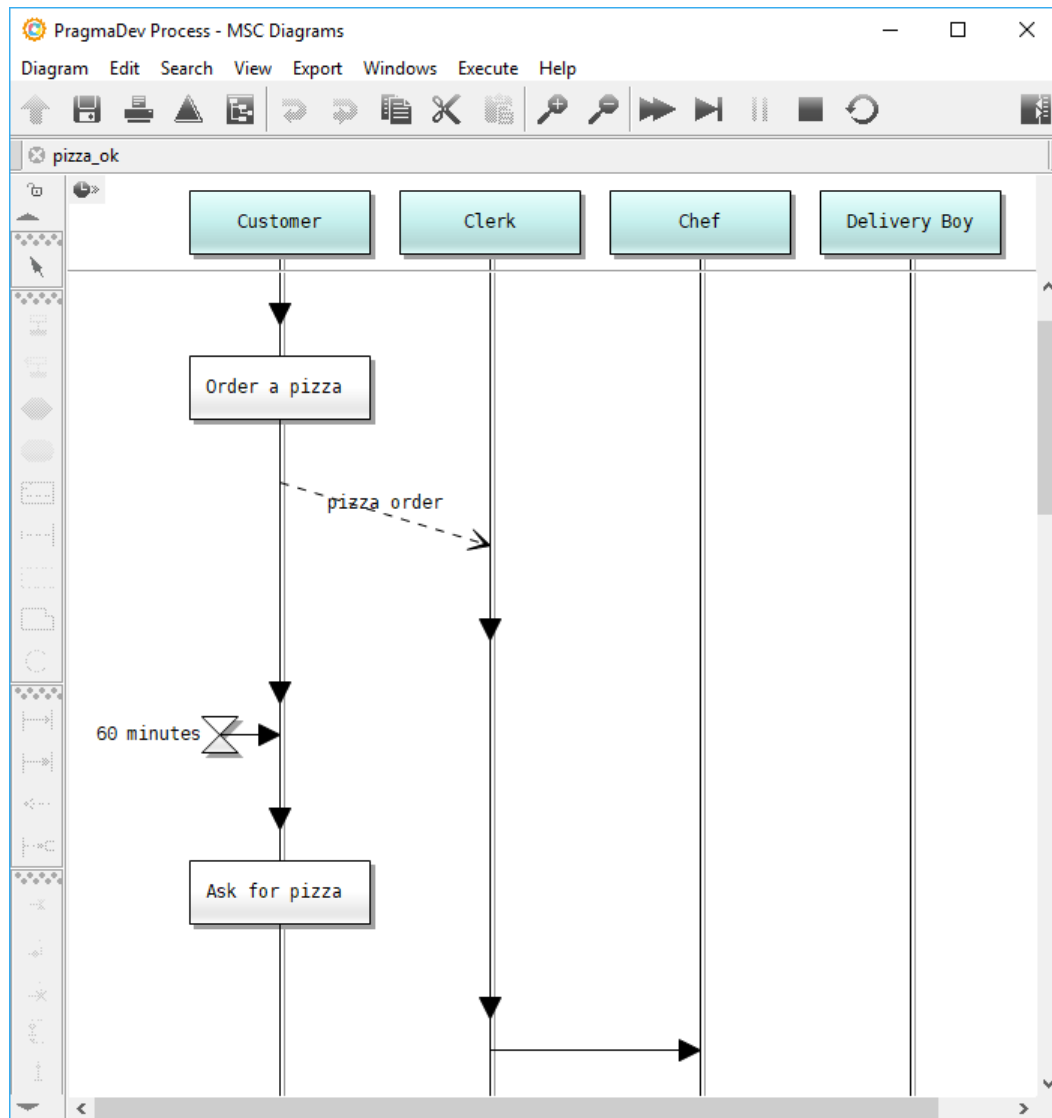
It is possible to replay a scenario. With `Pizza.bpmn` opened in the editor (double-click on it in the project manager), click the *Replay* button:



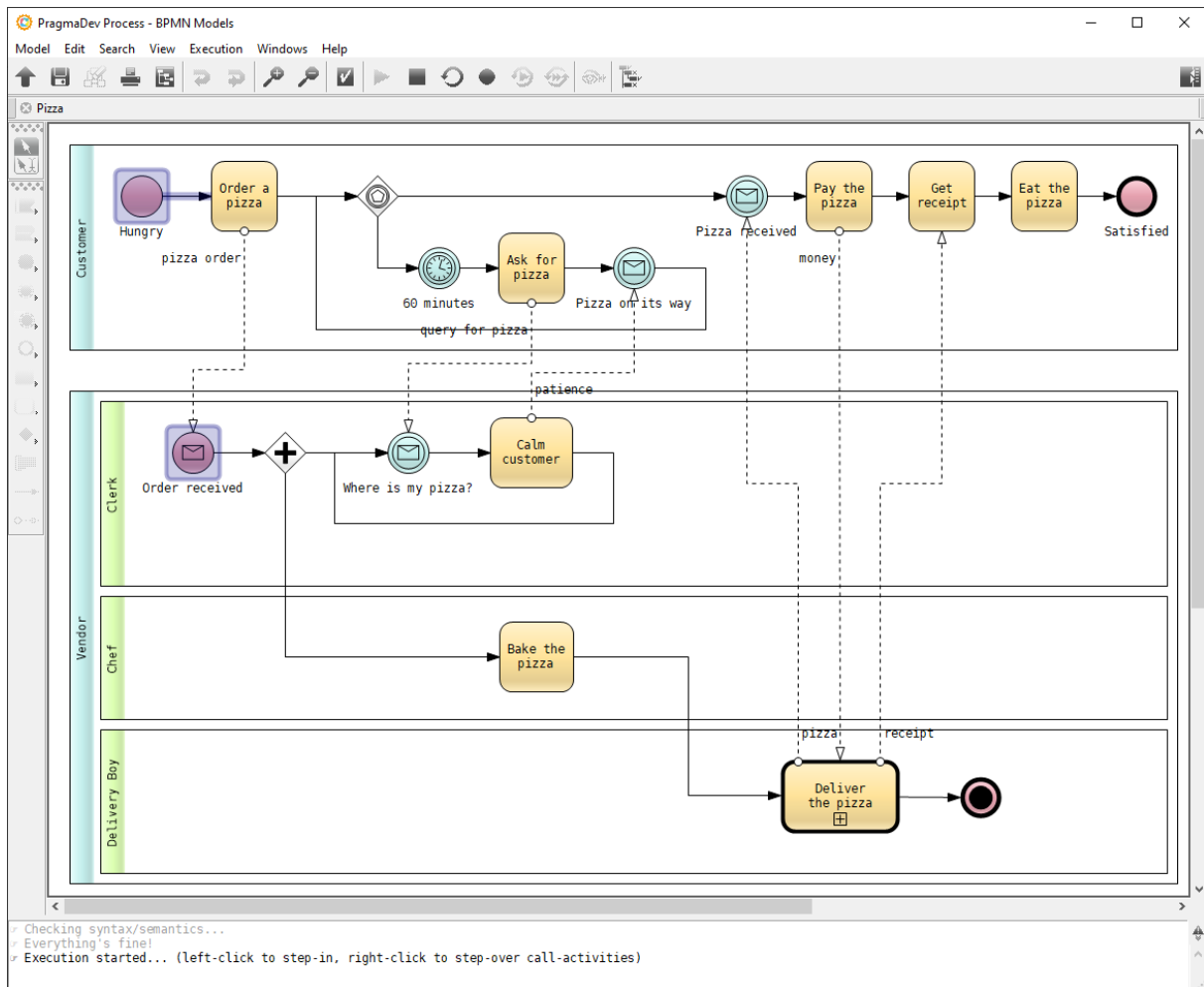
All recordings (MSC traces) found in the project will be listed; select `pizza_ok` and click the *OK* button:



The `pizza_ok` will be opened in the MSC editor:



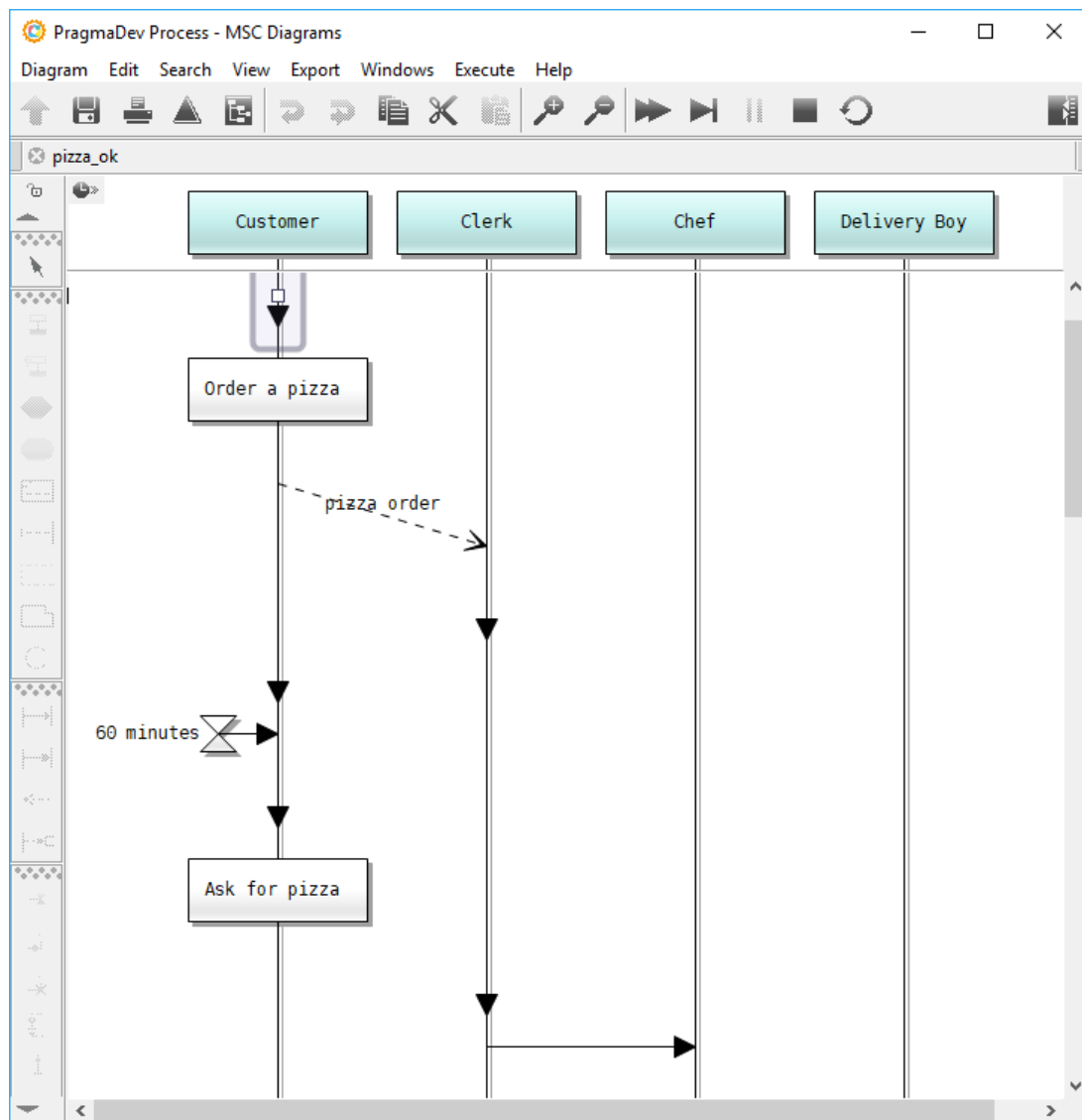
while the execution will start automatically in the BPMN editor:



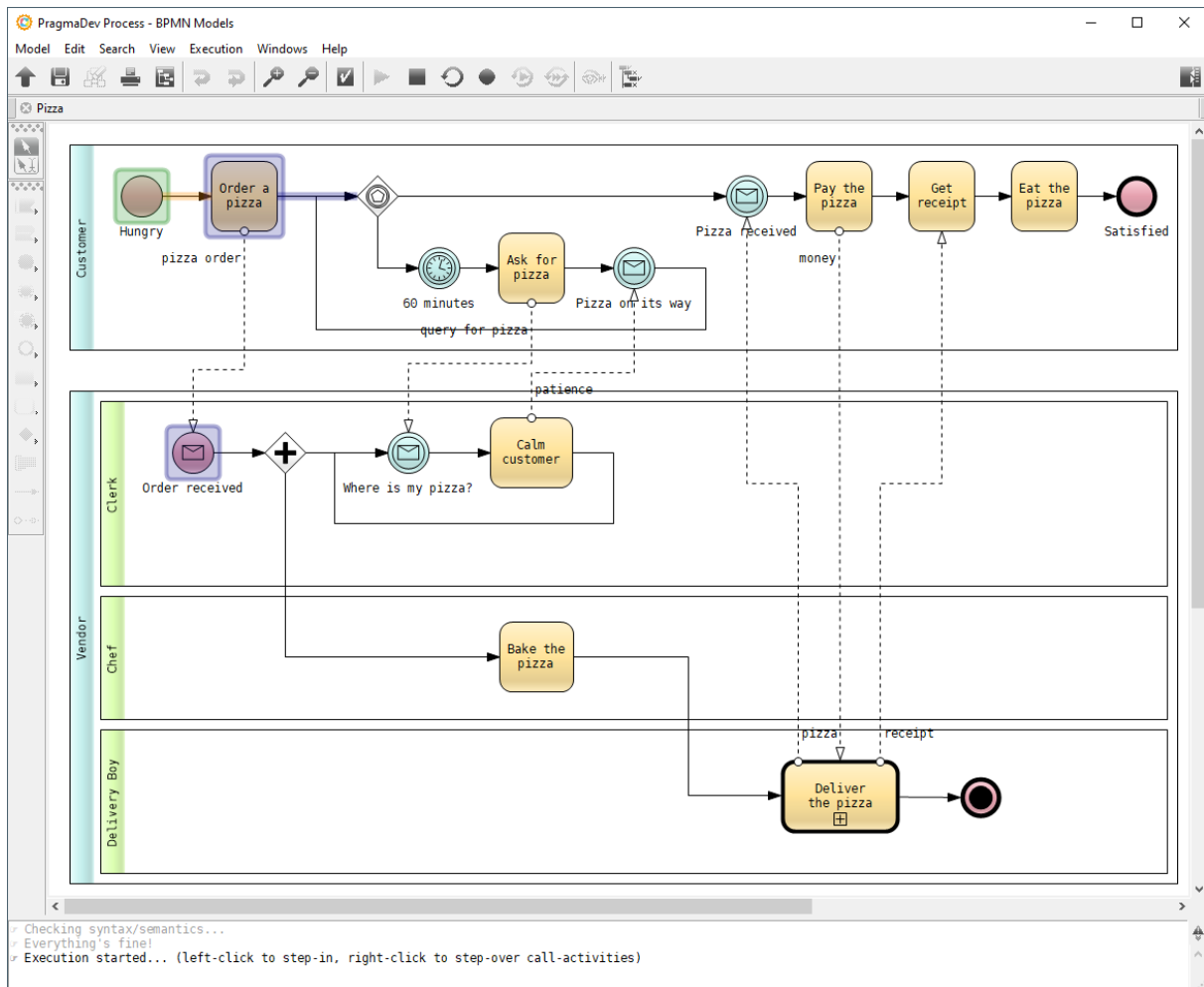
Click the *Step* button in the MSC editor:



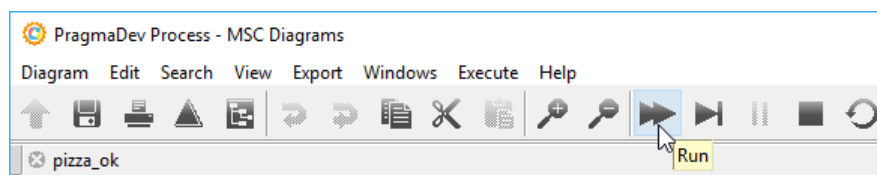
The first sequence flow will be selected in the MSC editor:



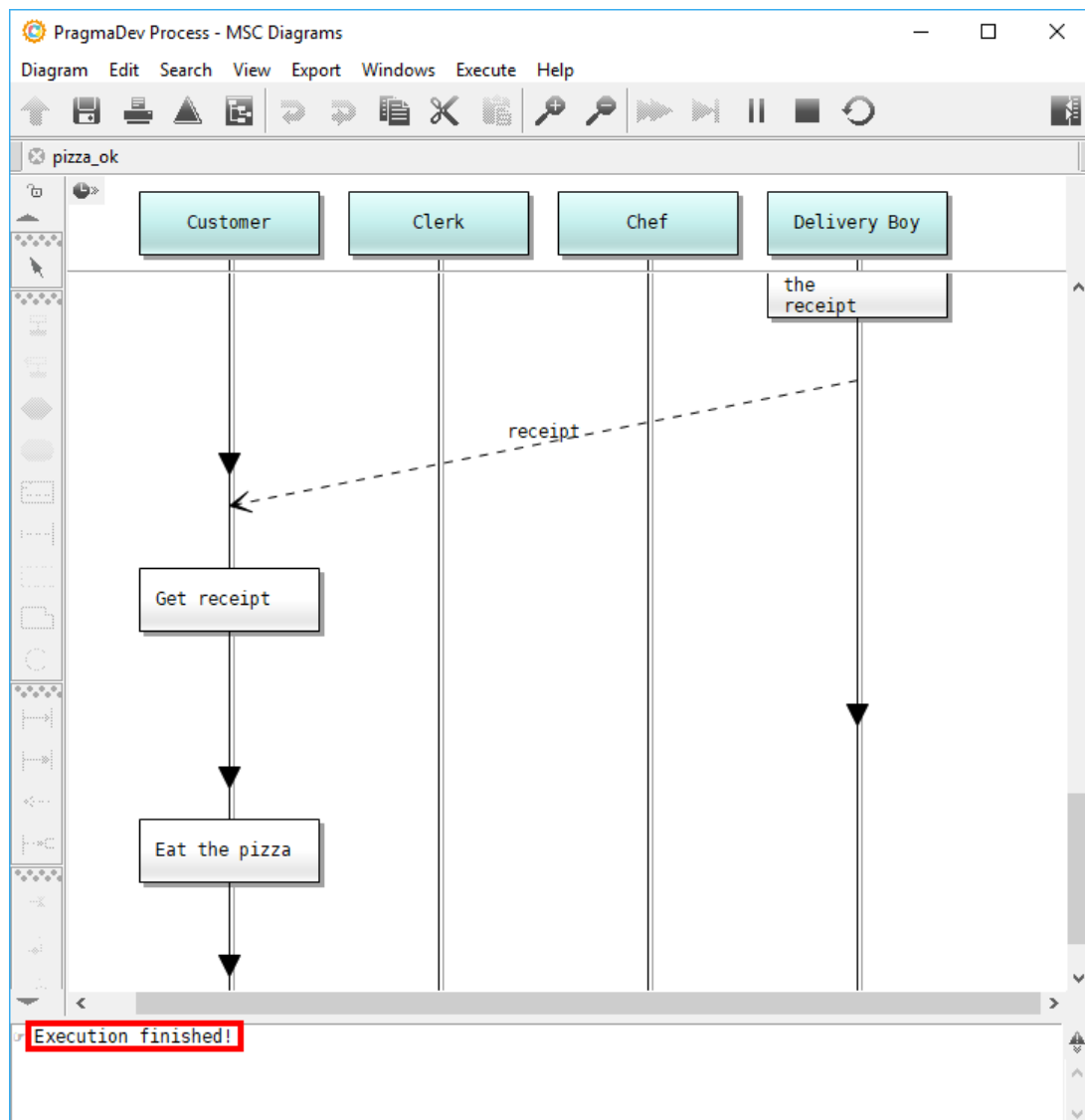
and executed in the BPMN editor:



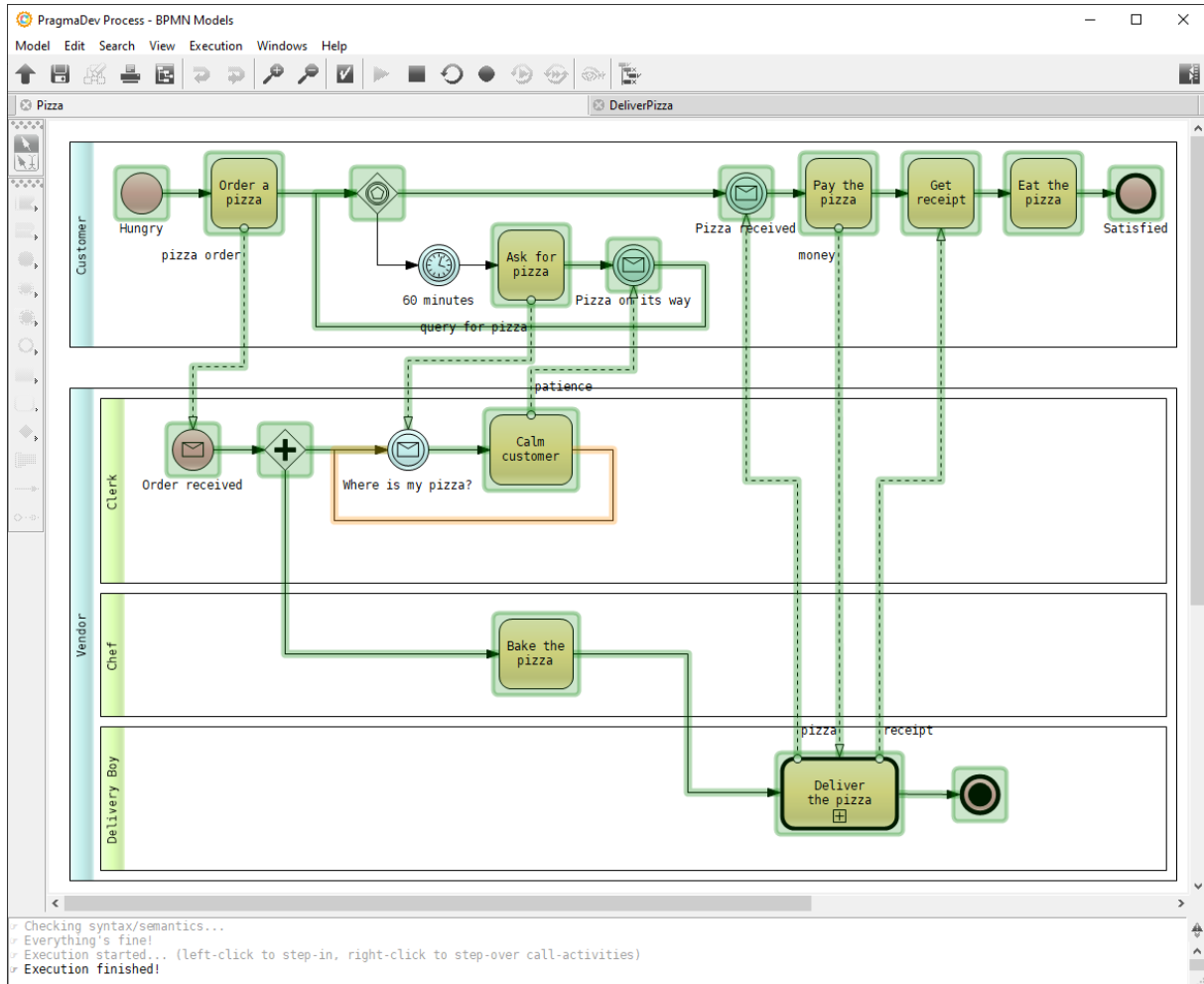
Try the *Step* button a couple of times and observe both (MSC trace and BPMN model) as execution advances in steps. Now Click the *Run* button:



Execution will continue until no more steps are left:

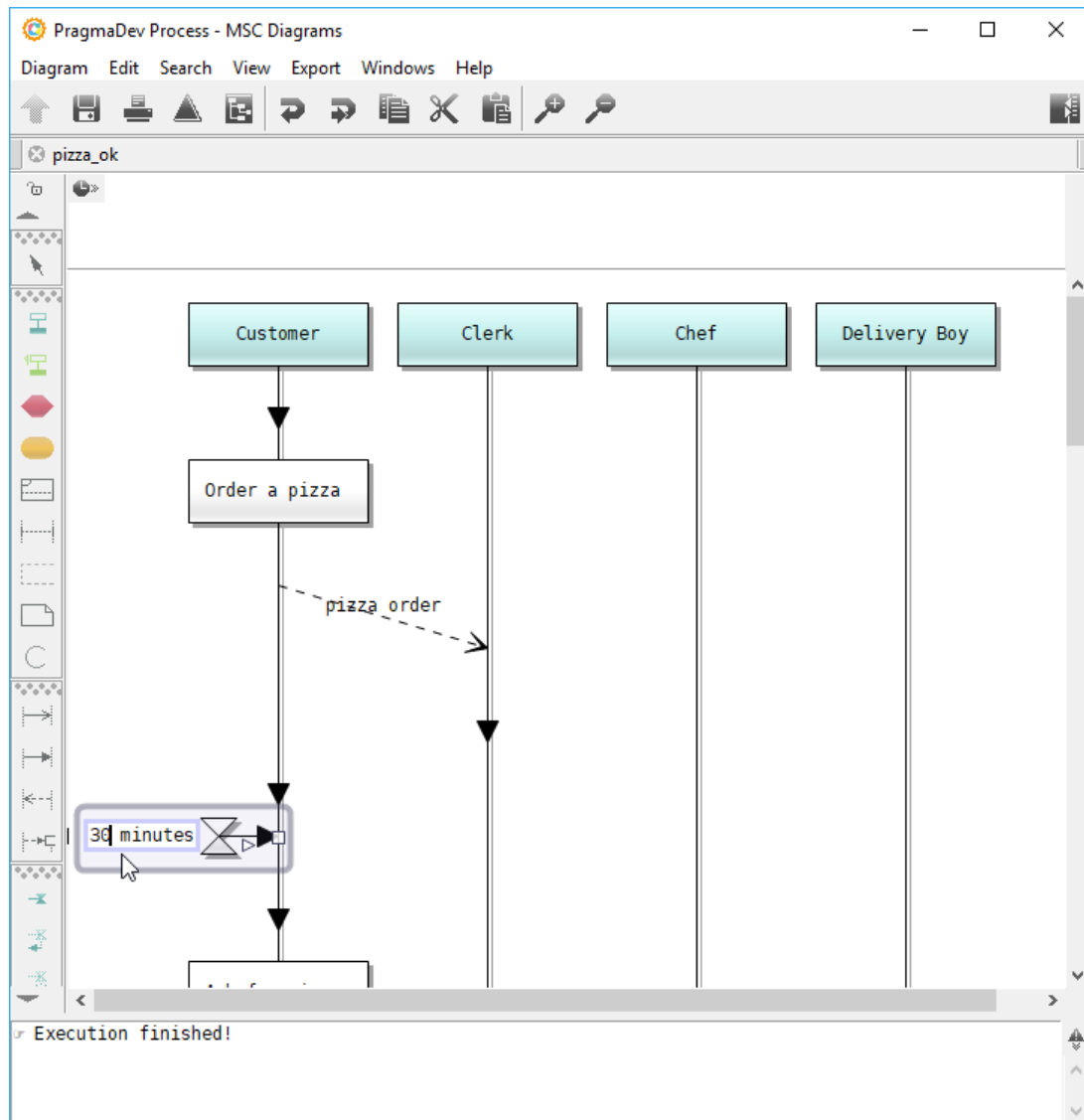


Check also the BPMN editor:

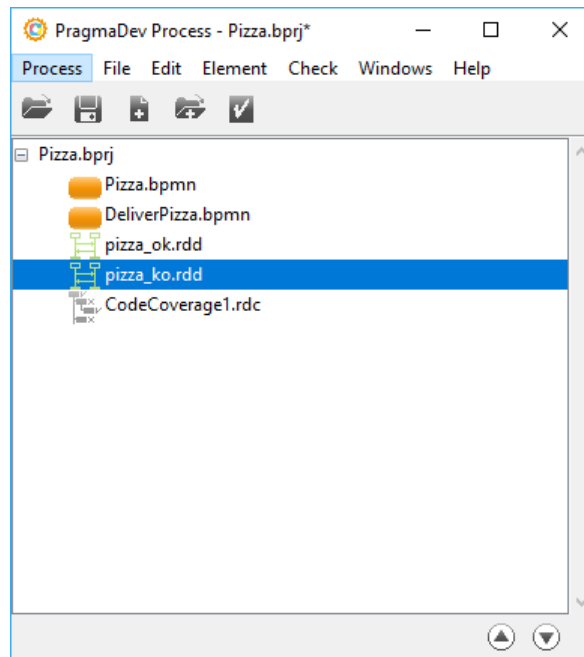


Use the *Stop* button to terminate execution.

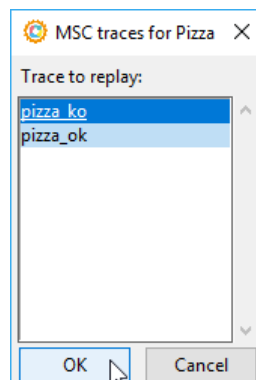
With the `pizza_ok` opened in the editor, find the 60 minutes timer and change it to 30 minutes:



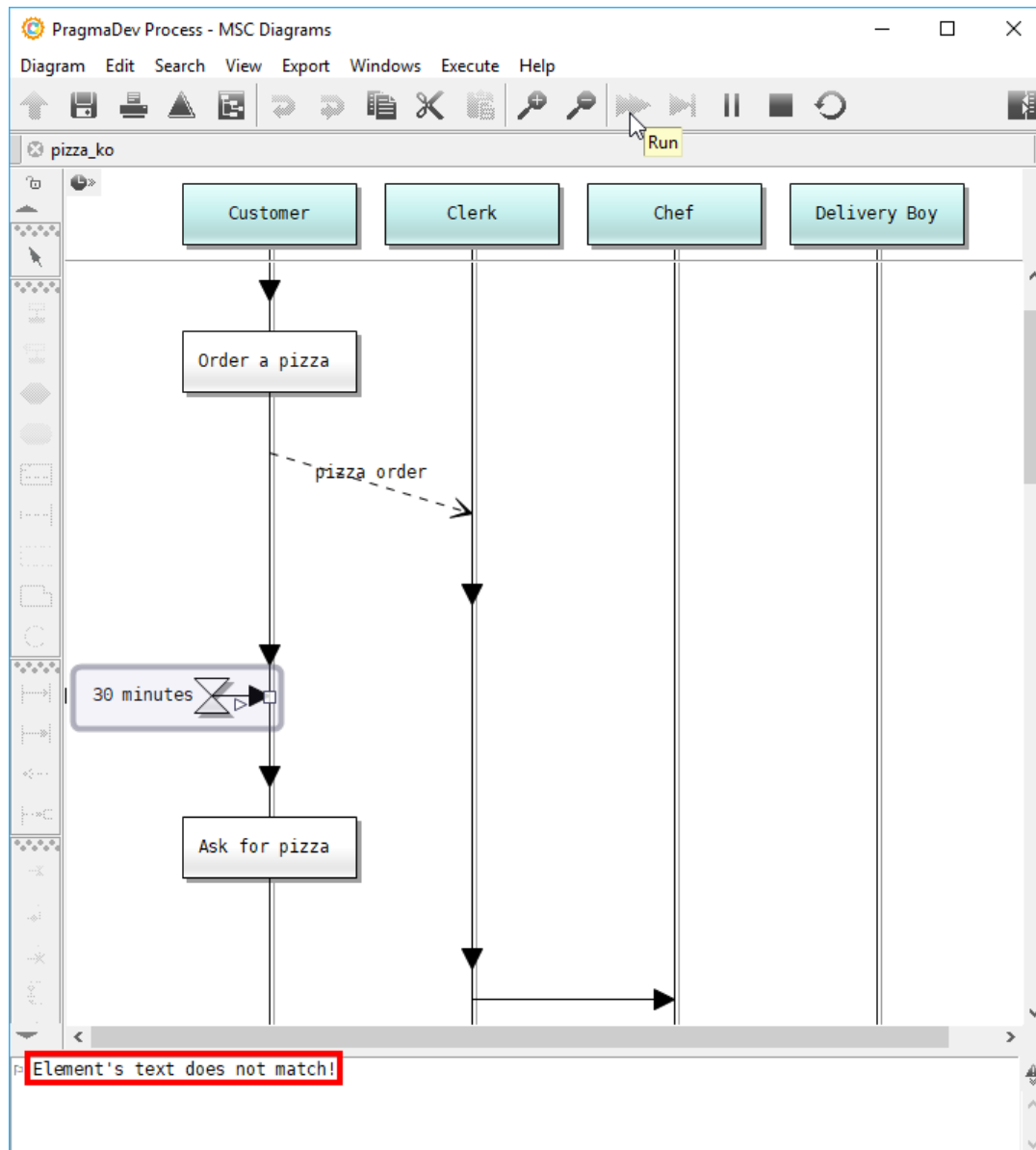
Save the modified MSC trace as `pizza_ko` via the menu *"Diagram / Save As..."*; it will be added to the project:



With `Pizza.bpmn` still opened in the editor, click the *Replay* button, and select `pizza_ko` for execution:

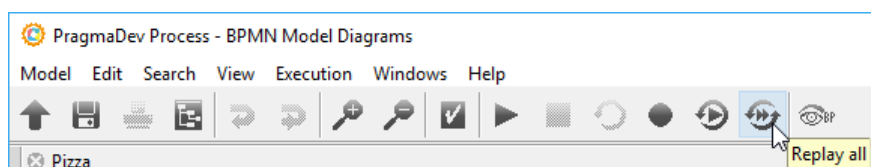


In the MSC editor click the *Run* button. Execution will stop at the timer symbol complaining about "text mismatch":

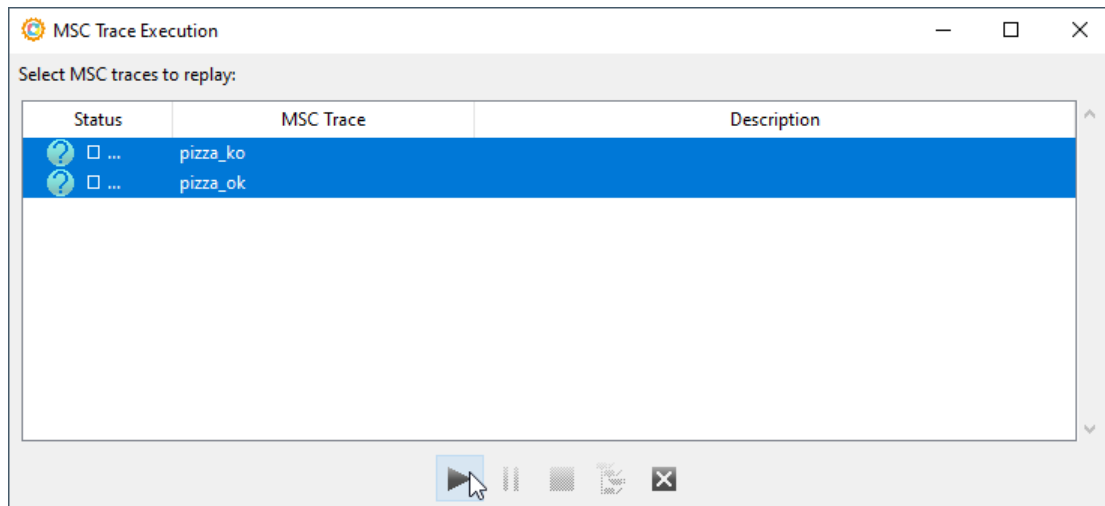


3.3.2 Multi-trace execution

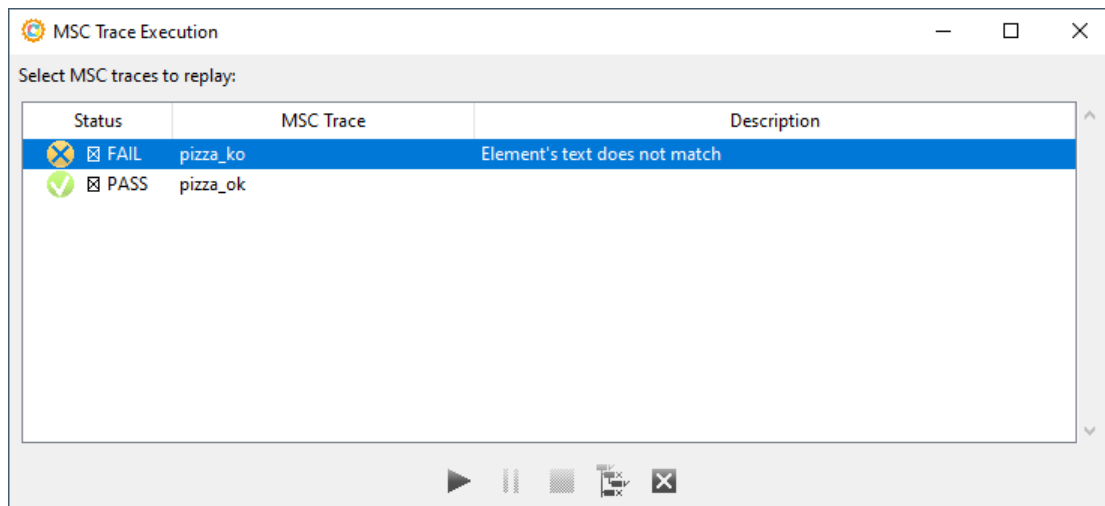
The tool can replay automatically several scenarios in a row to make sure they are still valid. Stop execution (if still running), and with `Pizza.bpmn` opened in the editor, click the *Replay all* button:



Select `pizza_ok` and `pizza_ko` by clicking on them while holding *Ctrl*. With both traces selected click the *Resume* button:



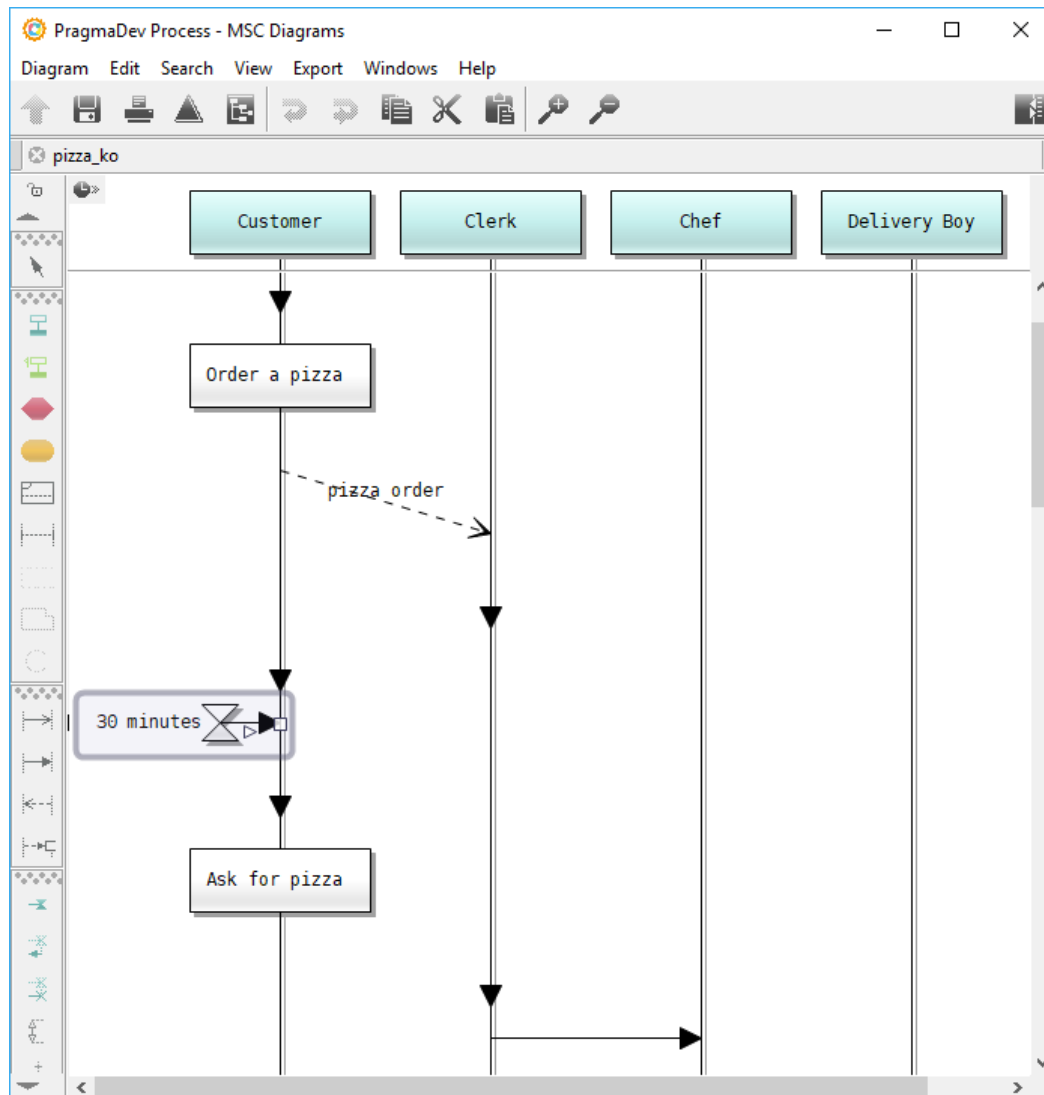
The traces will be executed in the background, and the result of such execution will be shown in the *Status* column:



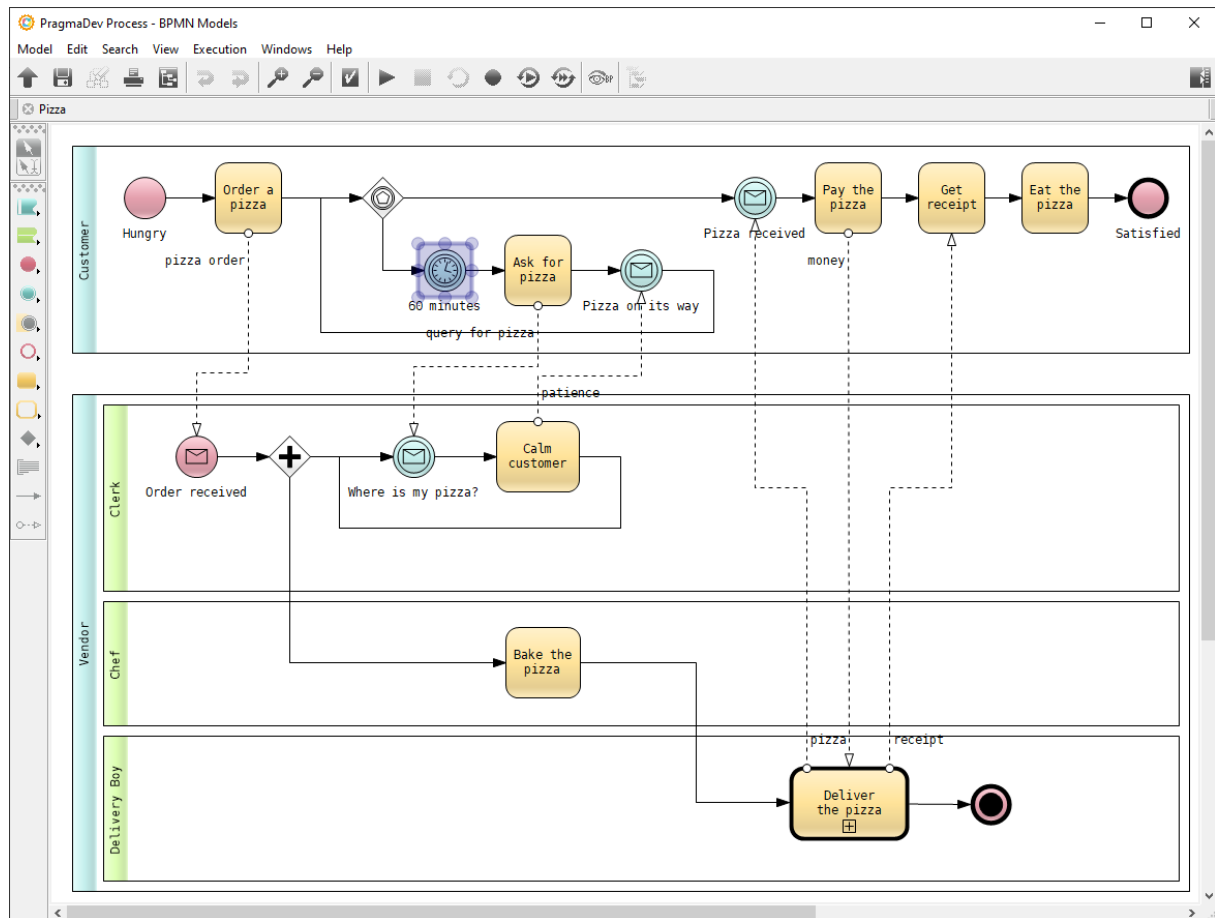
Note that model coverage can be generated via the *Generate coverage for marked traces* button.

Double-clicking a failed trace will:

- Open the trace in the MSC editor and select the concerned element:



- Select the corresponding element (if possible) in the BPMN editor:



4 Exploration

Exploration of BPMN models in PragmaDev Process is done via OBP (see User Manual). PragmaDev Process exploration feature can be used to check:

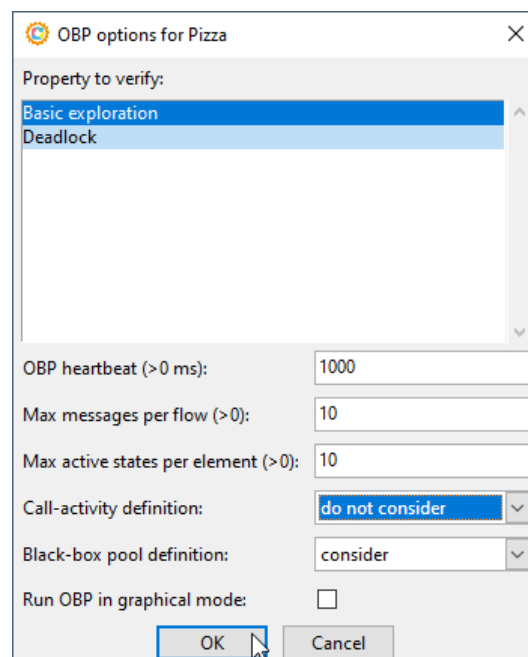
- *Complexity*: through full exploration of the model.
- *Reachability*: identification of unreachable paths.
- *Property*: verification of a property expressed in either PSC (Property Sequence Chart) or GPSL (Generic Property Specification Language).

4.1 Complexity check

The tool can automatically execute any possible combination of flows to evaluate the complexity and the reachability of the model. With `Pizza.bpmn` opened in the editor (double-click on it in the project manager), click the *Explore with OBP* button:

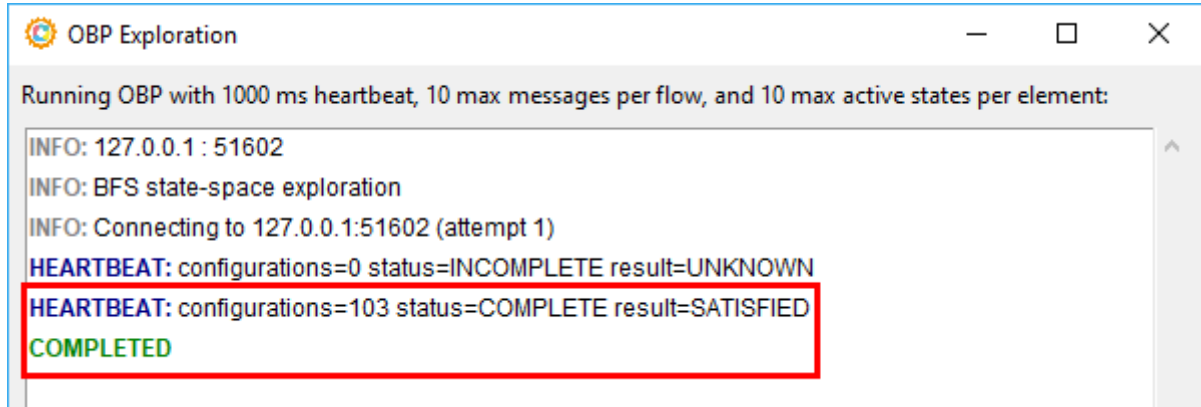


Select *Basic exploration*, set *Call-activity definition* to *do not consider*, and click the *OK* button:



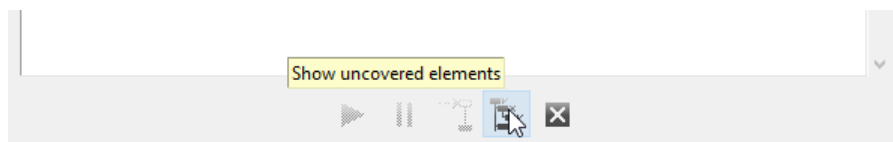
The *do not consider* option will step-over call-activities during exploration.

Observe the increasing number of configurations¹ while waiting for the exploration to complete. The number of configuration can be compared to the model complexity to find out if that result is too high or normal (see User Manual).



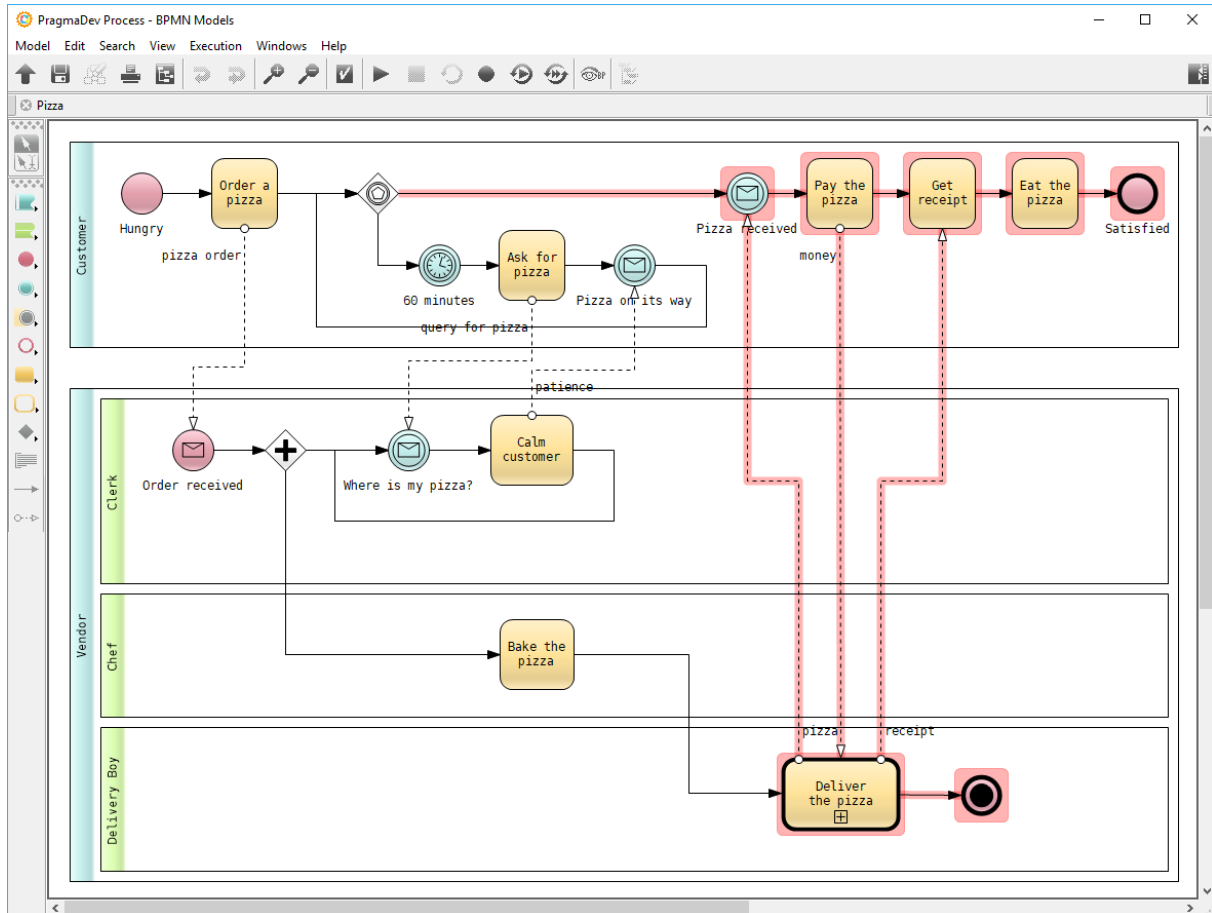
4.2 Reachability

An automatic exploration will find any reachable paths in the process. If some symbols are not reachable there is probably a problem in the model. To identify unreachable symbols, after an exploration is completed, click the *Show uncovered elements* button at the bottom of the exploration window:



All non covered elements during exploration will be marked in **red** in the editor:

¹The exploration may be interrupted before completion when using the free version of PragmaDev Process.



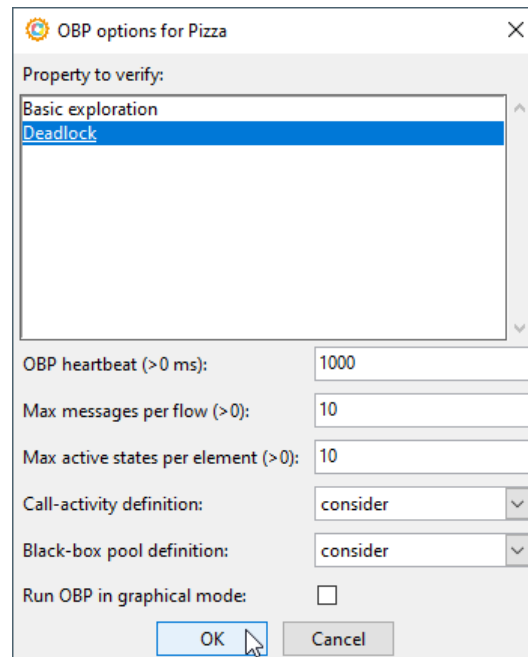
In our example, stepping over the *Deliver the pizza* call-activity resulted in being stuck during exploration. This is because the behavior of a stepped over activity is to wait for all incoming message flows before executing all outgoing message flows. Outgoing sequence flows can be executed only when all message flows have been treated. The *Deliver the pizza* call-activity cannot send the *pizza* message before receiving the *money* message, which cannot happen.

Quit OBP.

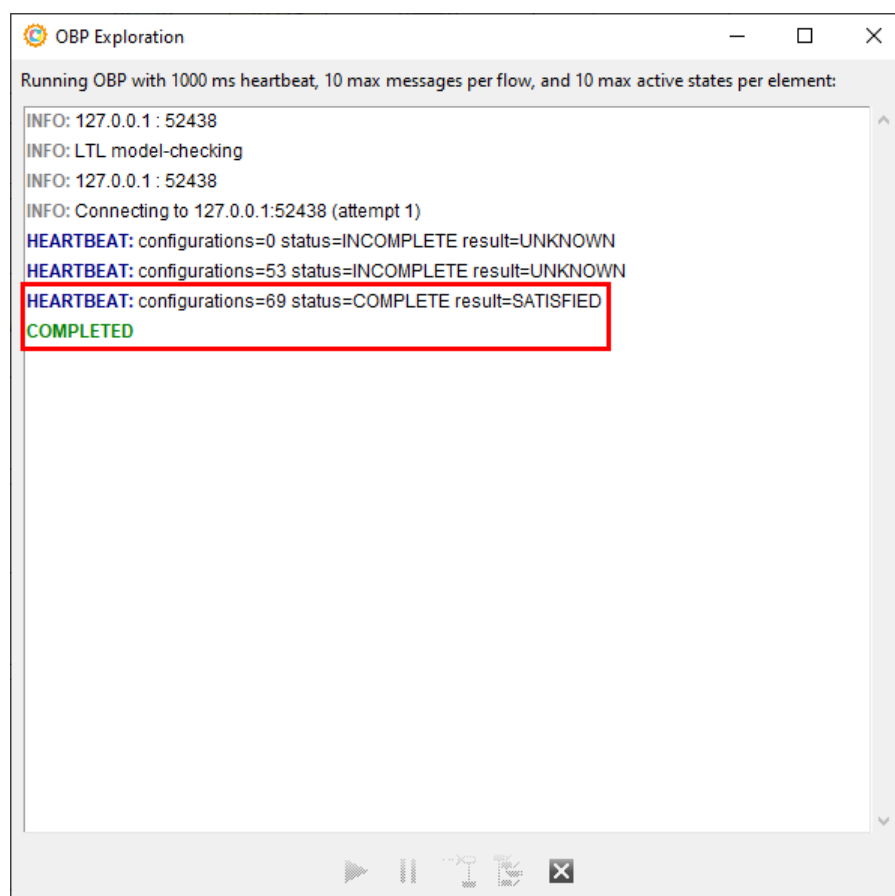


4.3 Deadlock check

Now let's check the model for deadlocks. A deadlock is a state where there are no actions to perform but the process is not finished (terminated). Run OBP again with *Deadlock* selected:



Observe the result returned by OBP saying that the property was satisfied:

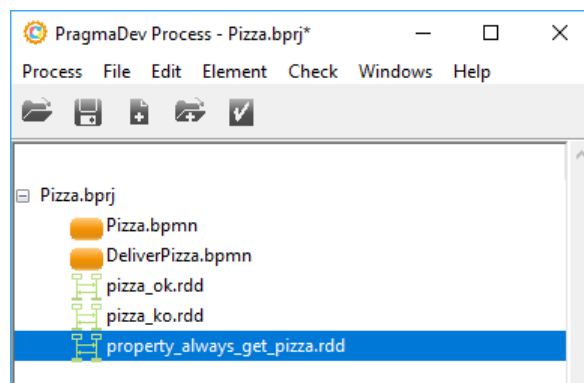


This means that there are no deadlocks in the model.

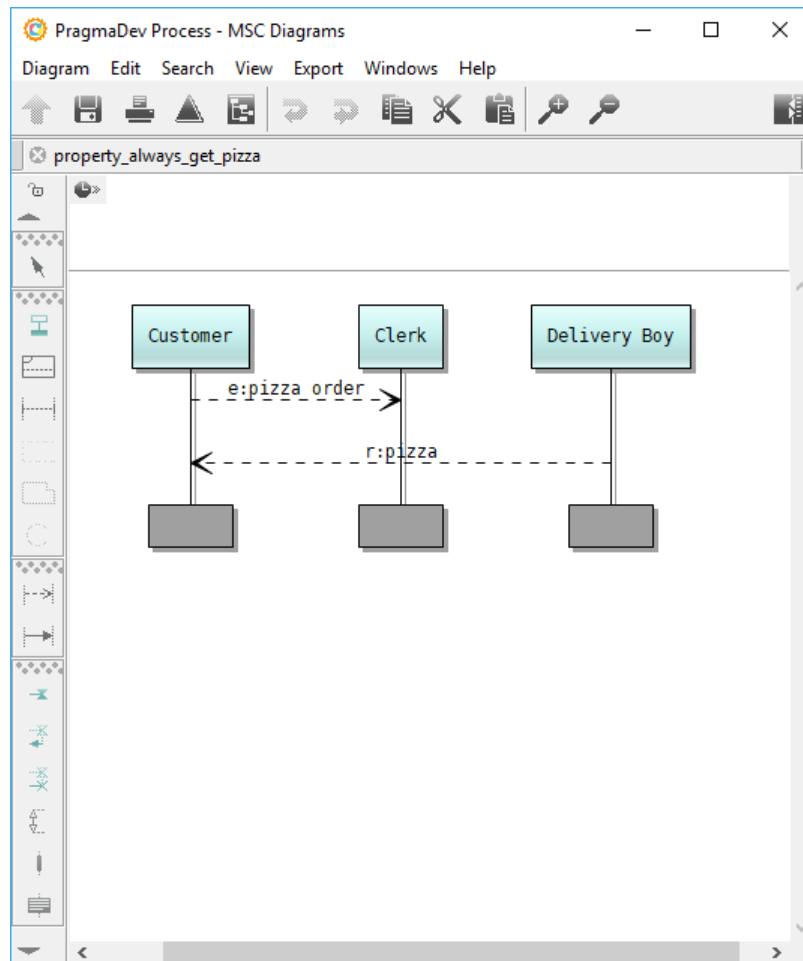
Note that the presence of non covered elements in the model does not mean that there is also a deadlock. An infinite loop is a typical case where the process never terminates, but there is always at least one action to perform. The following section will illustrate this case using a property.

4.4 Property verification

We will first create a property and then verify it on the model. In the Project Manager create and add a new PSC file to the project via the button in the toolbar, and name it `property_always_get_pizza`:



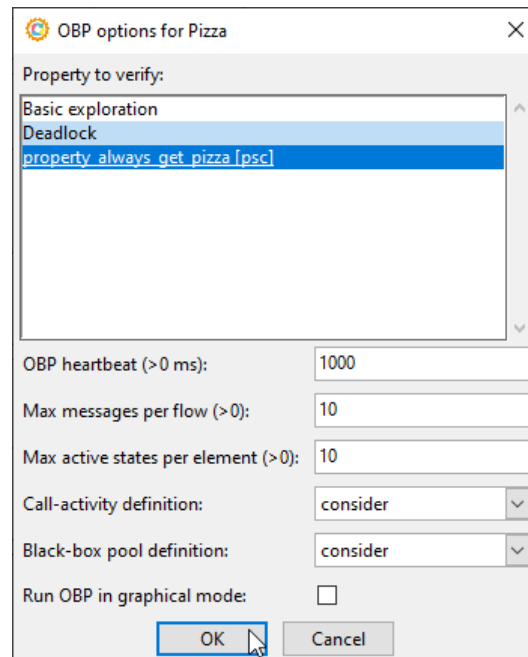
Double click on it to open the MSC/PSC Editor, and draw the following property:



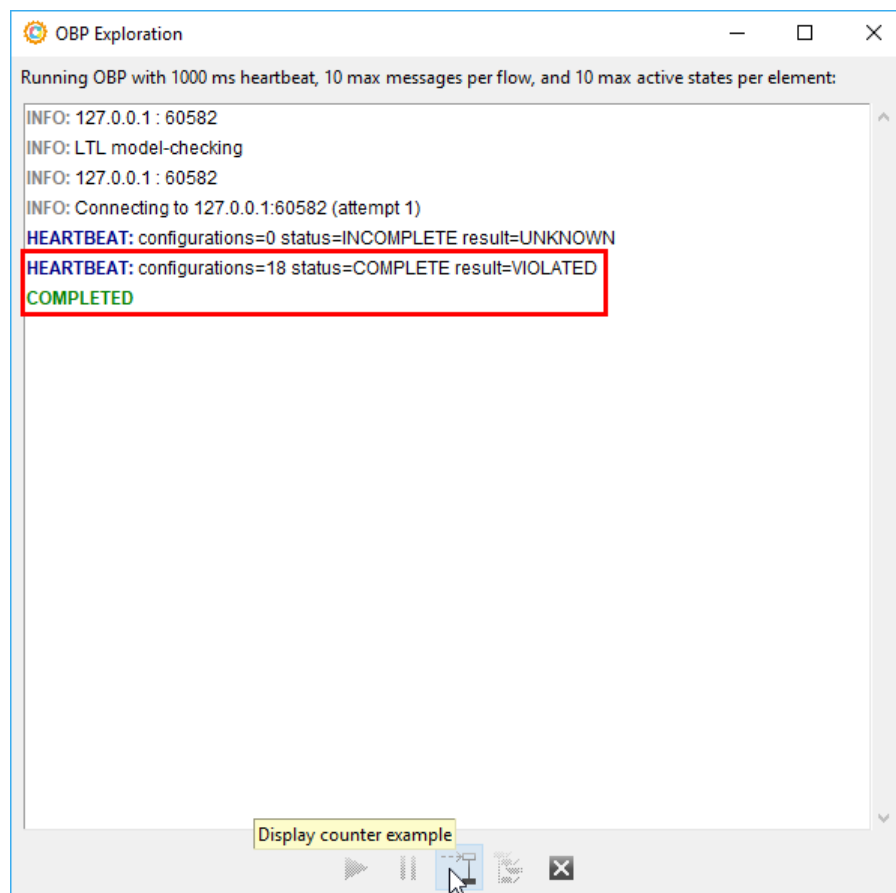
The PSC property says the following:

- If the Customer has sent a `pizza order` to the Clerk, then
- the Delivery Boy *must* hand over the pizza to the Customer.

To verify this property, with the `Pizza.bpmn` opened in the editor, click the *Run OBP* button. Select `property_always_get_pizza` and click *OK*:

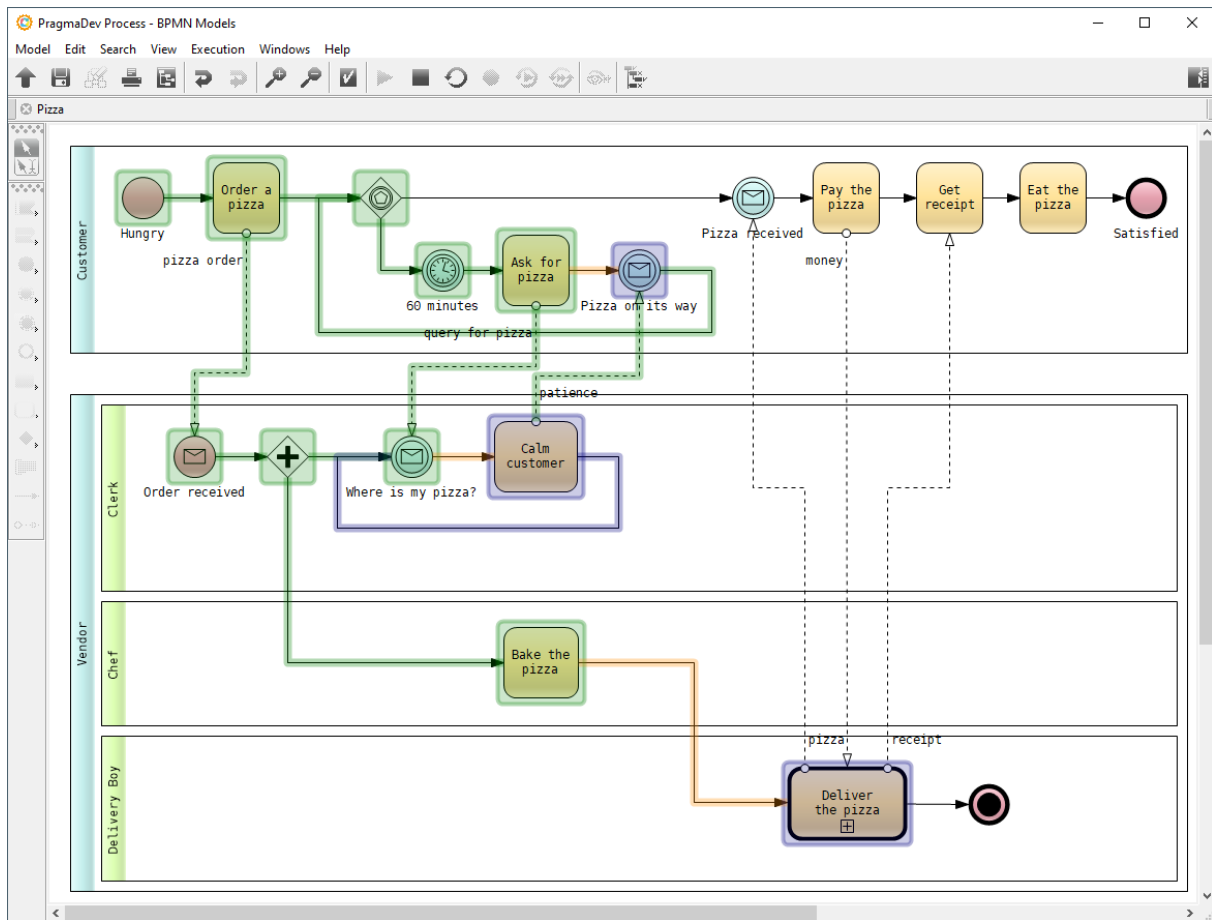


Observe the result returned by OBP saying that the property was violated.² Property violation will activate the *Display counter example* button, click it:

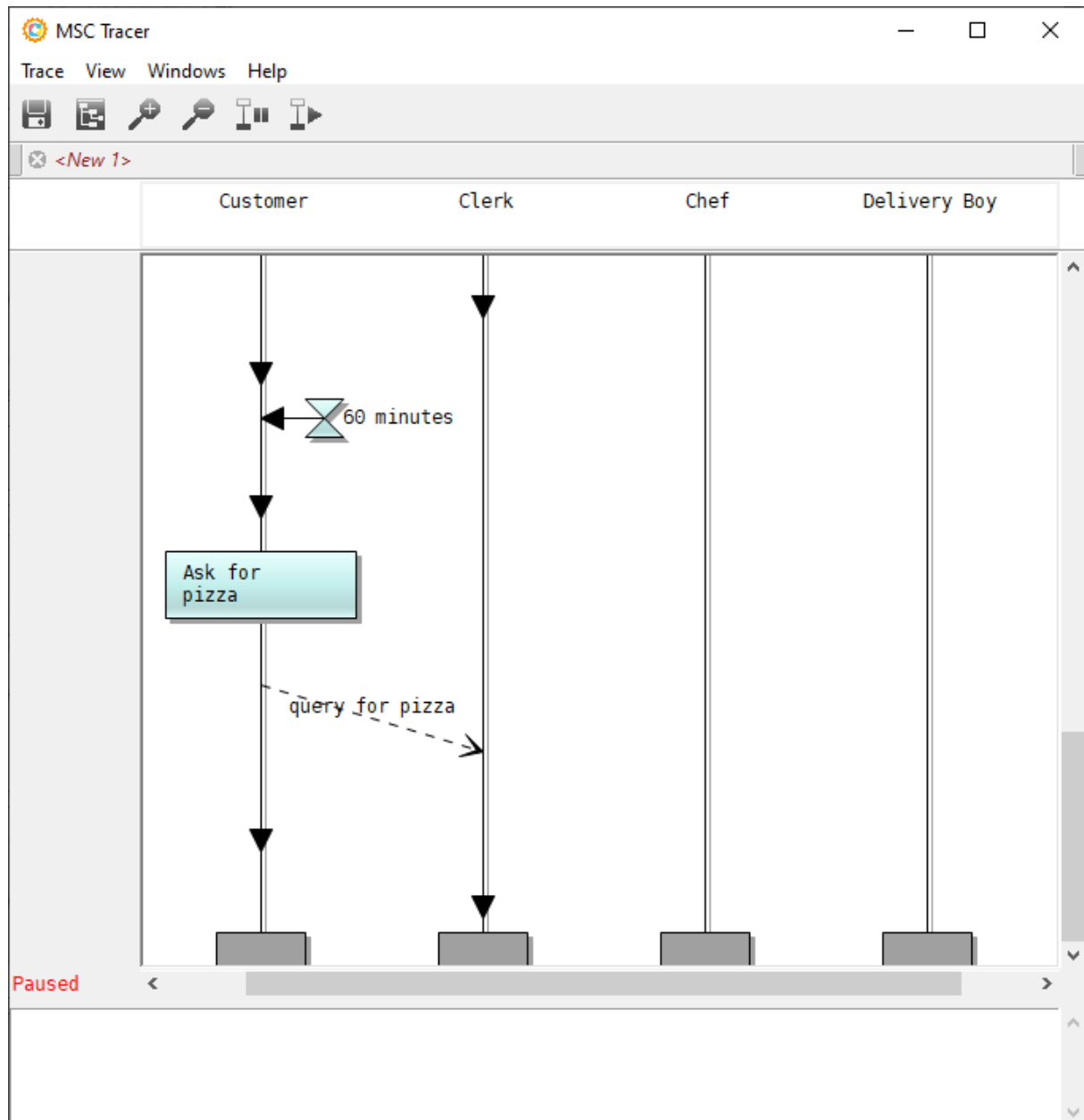


²The exploration will be interrupted before completion when using the free version of PragmaDev Process.

The sequence of events that violated the property will be automatically replayed in the editor:



and recorded in the MSC Tracer:



The property was violated because there exists a case (as shown above) where the Customer is stuck in an infinite loop always asking for his/her pizza, and thus never receiving it.

5 Conclusion

During this tutorial we have been through:

- BPMN,
- Project Manager,
- BPMN Editor,
- BPMN semantic check,
- BPMN Executor,
 - Interactive,
 - Automatic,
- BPMN Explorer,
 - Complexity,
 - Reachability,
 - Property.

PragmaDev Process is a powerful tool that allows creation, editing, checking, executing, and exploring BPMN models.