PRAGMADEV PROCESS

TUTORIAL
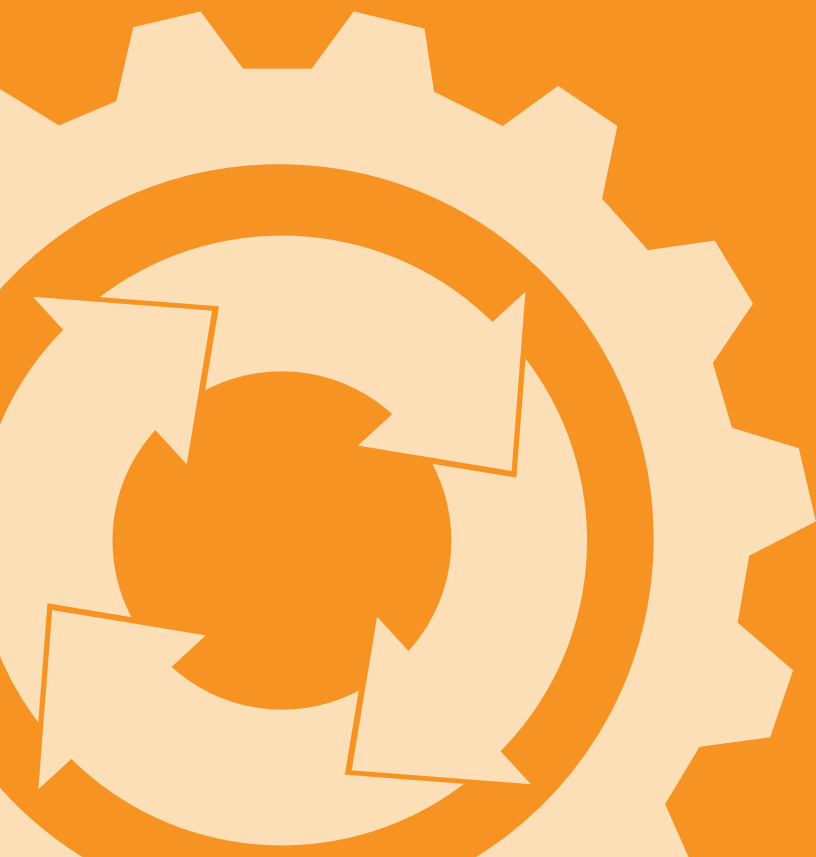
PRAGMADEV
modeling and testing tools

Contents

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*.
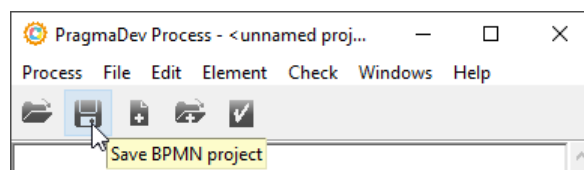
The BPMN model of the pizza delivery used in this tutorial is found in `$PRAGMADEV_-PROCESS_HOME/examples/Pizza`.

2.1 Organization

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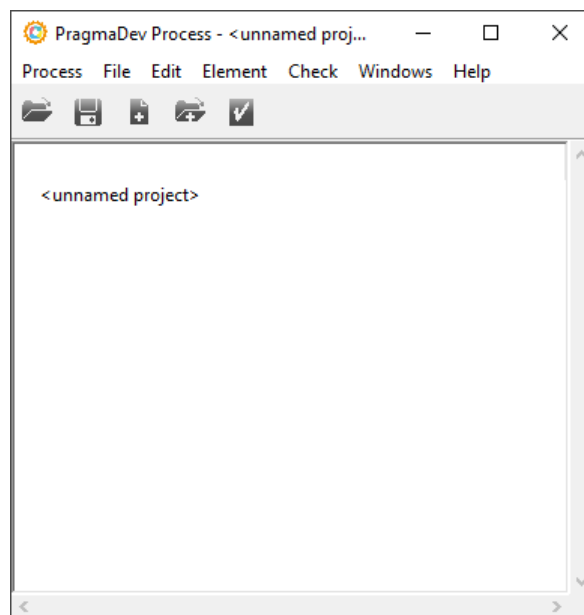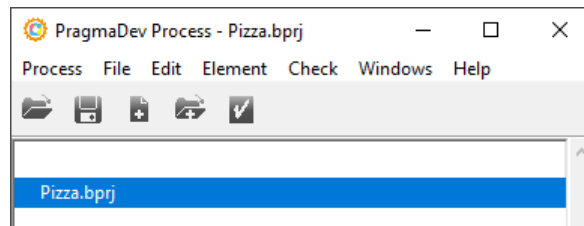

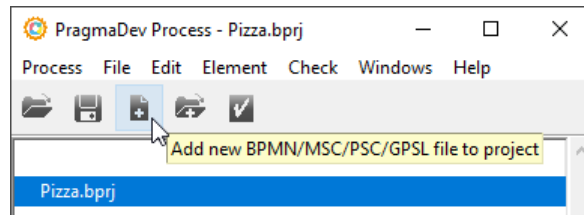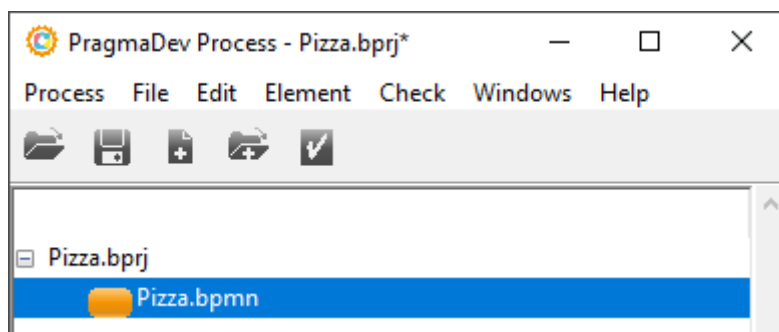Save the project via the button in the toolbar:

Name the project `Pizza`:

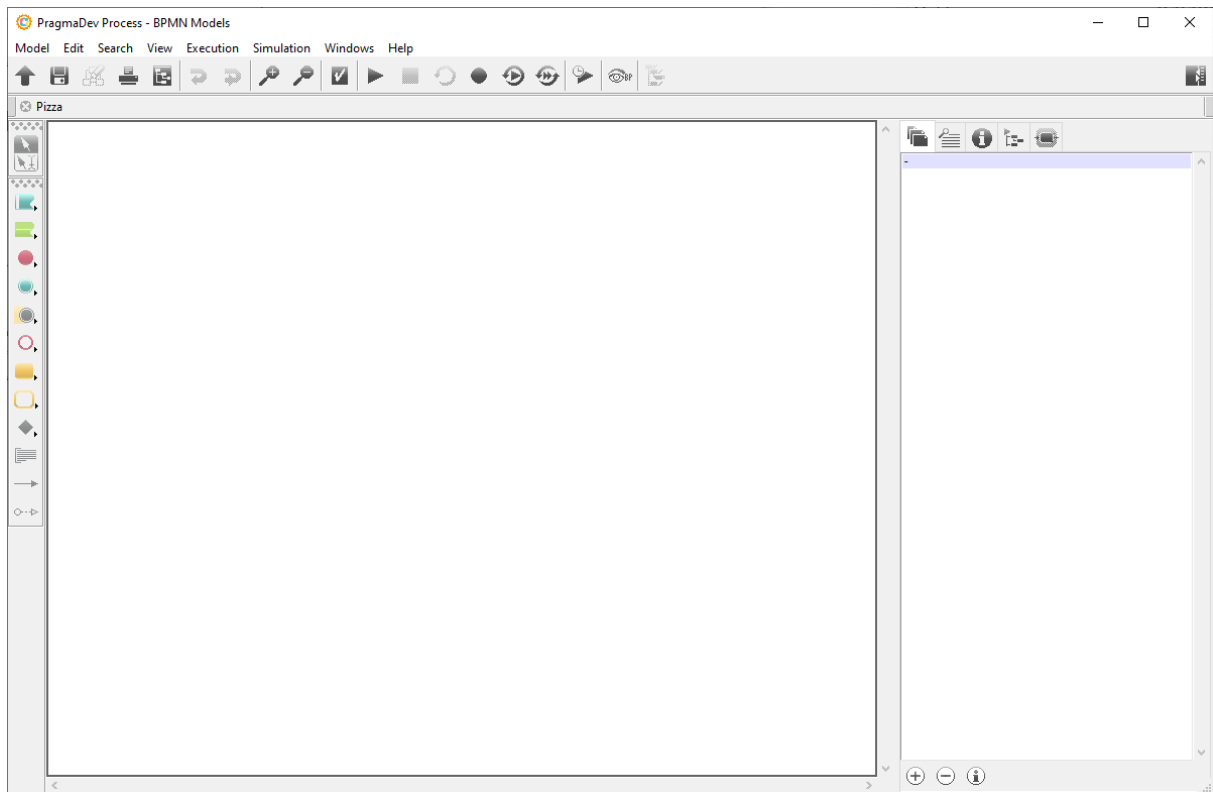Create and add a new BPMN file to the project via the button in the toolbar:
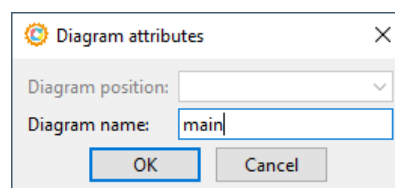

Name it `Pizza`:

2.2 Pools and lanes

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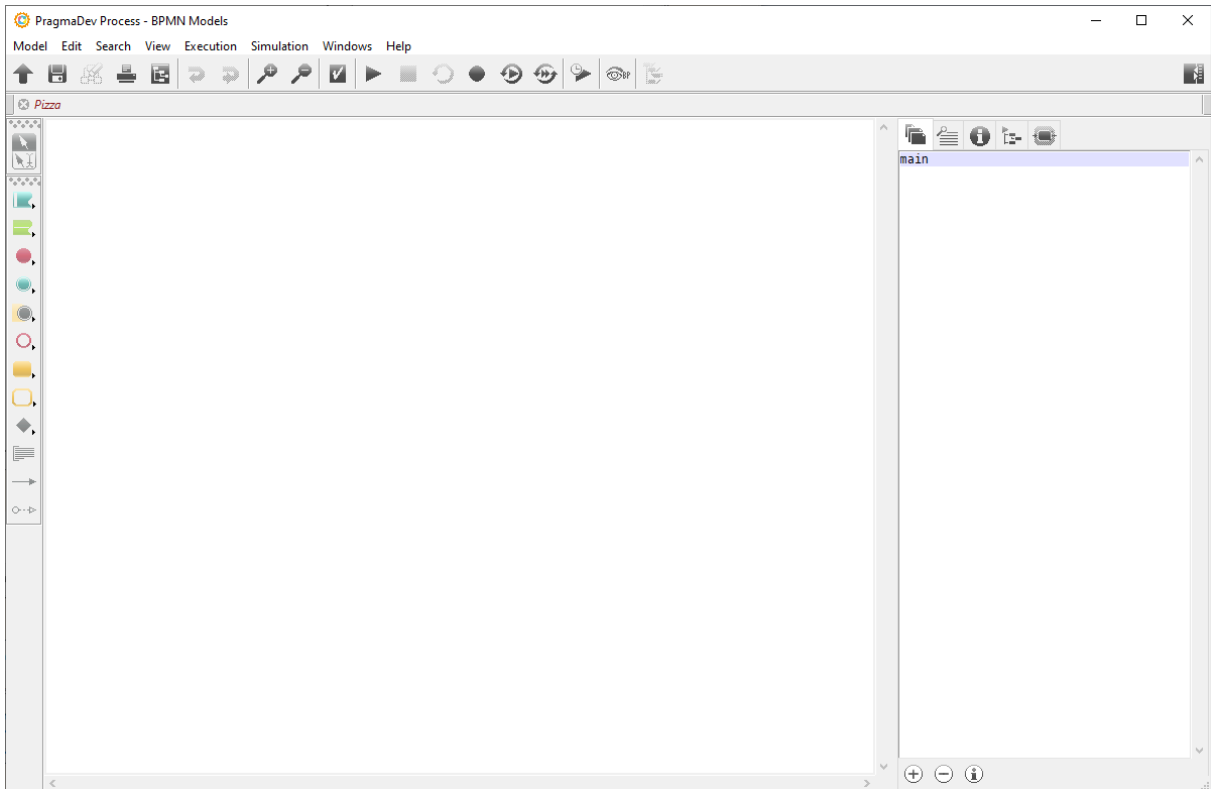

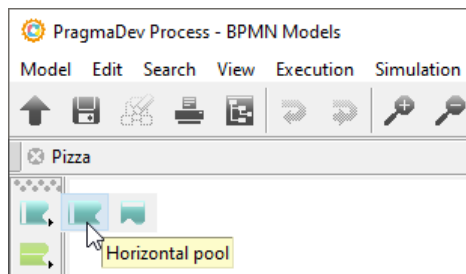To change the name of the current diagram use the button:

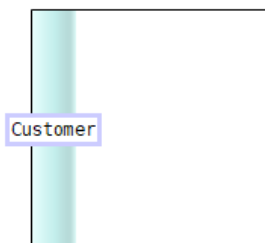Name it `main` and hit "OK":

Notice the change in the diagram browser:

Create a horizontal pool via the tool button, or via the keyboard shortcut control+space, then p:
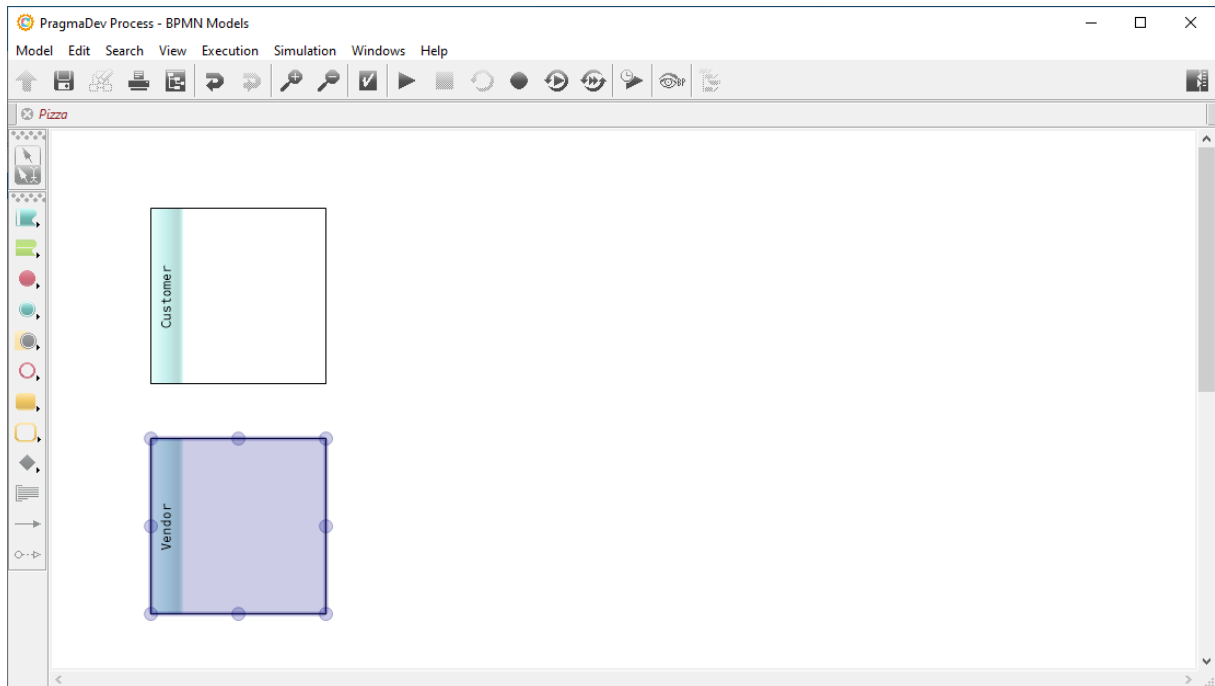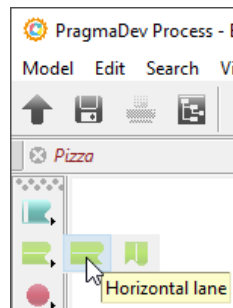

and name it `Customer`:

You can leave the pool at its default size, as it will grow automatically when the process for the customer is created in it. We'll clean-up things later to make everything look
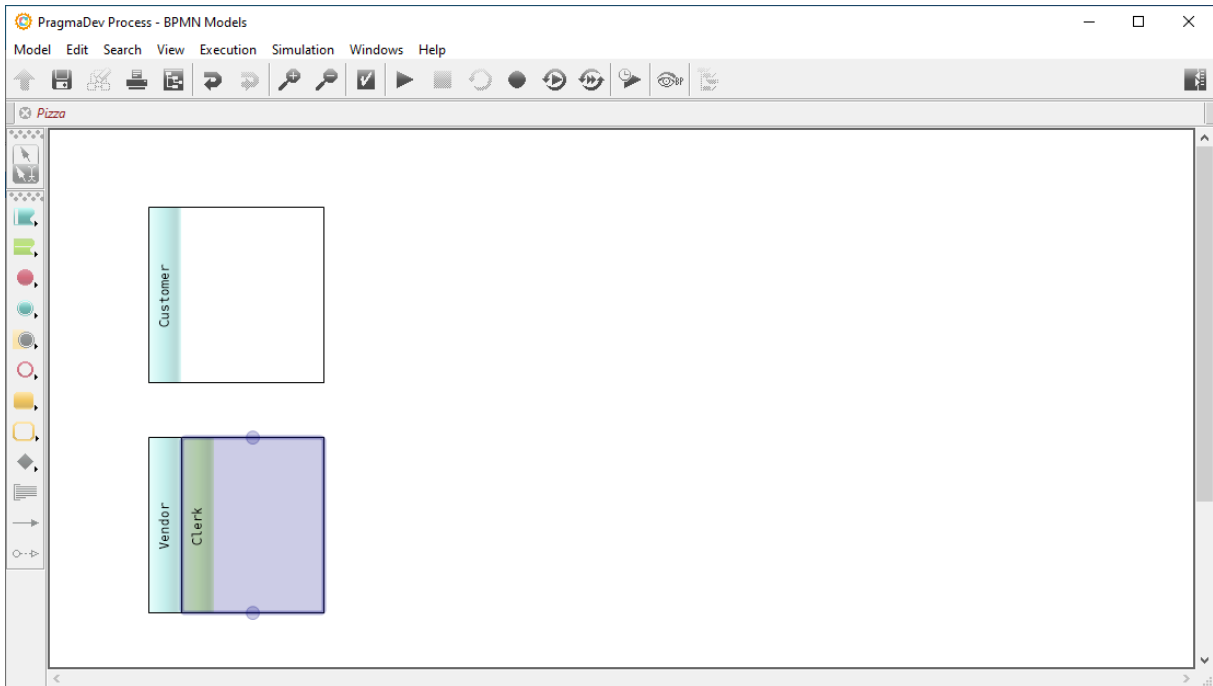
good.

Create another pool named `Vendor`:

Create a horizontal lane inside the `Vendor` using the tool button, or via the keyboard shortcut control+space, then `l`:

Name it Clerk; it should automatically resize itself to the parent pool:

Now repeat the operation twice, taking care of dropping the new lane in the lower half of the last one in the pool to add it at last position. Name the two new lanes `Chef` and `Delivery boy`:
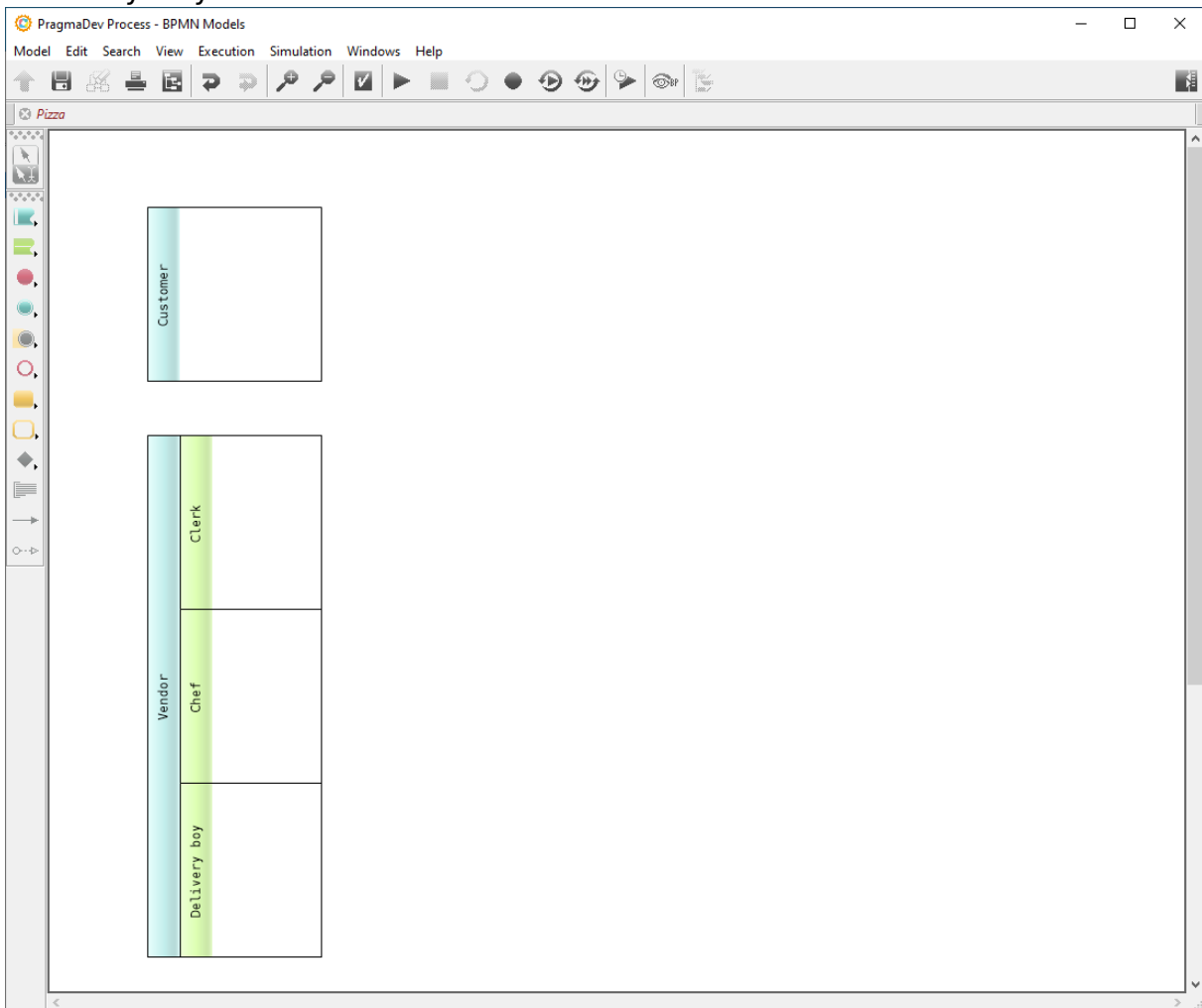
NB: to add lanes to a pool, you can also right-click on it to display its contextual menu and select "Add lane at first position" or "Add lane at last position", or right-click on a lane inside it and select "Add lane before" or "Add lane after" in its contextual menu.

2.3 Processes

Use the tool buttons to draw the `Customer` process as shown:
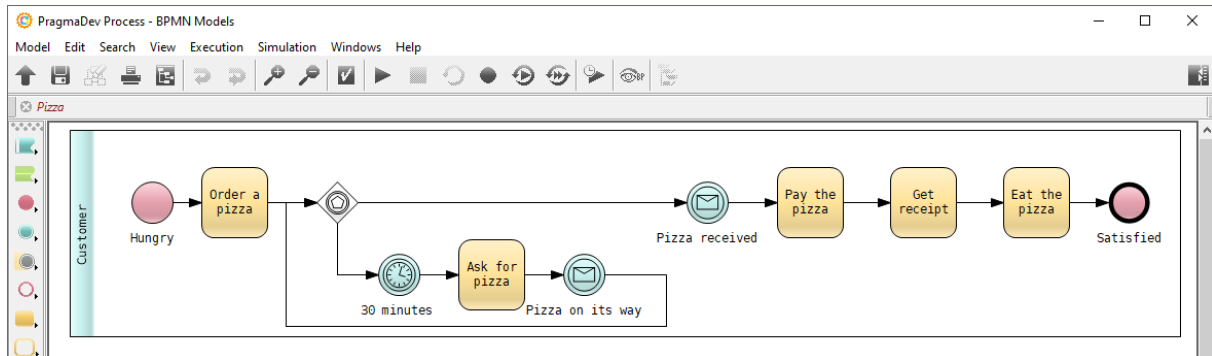
Symbols and links can either be created:
- with the tools in the toolbar;
- via keyboard shortcuts;
- by using the hover buttons appearing when the mouse pointer is over a symbol.

The very first symbol (here the start symbol named "`Hungry`") have to be created using one of the first 2 methods. For the second method, the insertion keyboard 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 "?".

For all symbols other than the first, the most efficient method is to use the hover buttons. For example, once the start symbol has been created, putting the mouse pointer over it will display 3 buttons beside it:
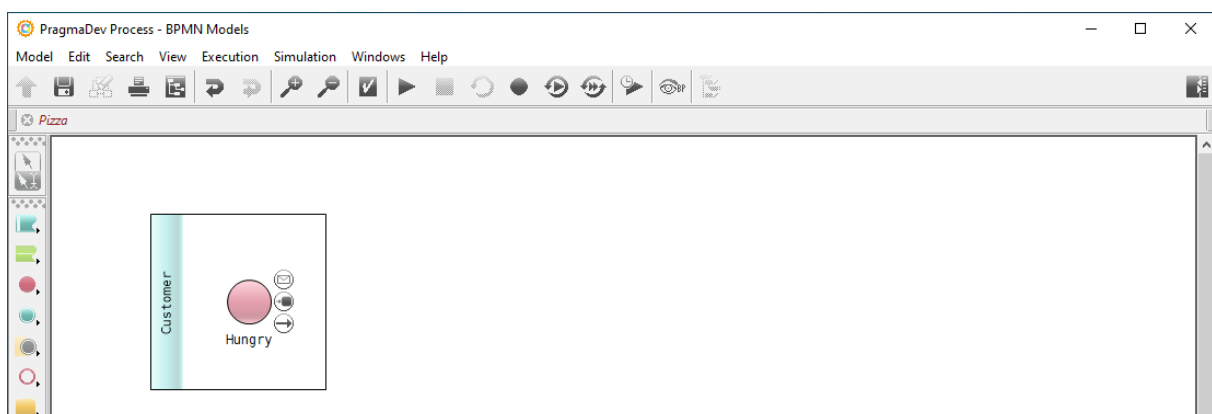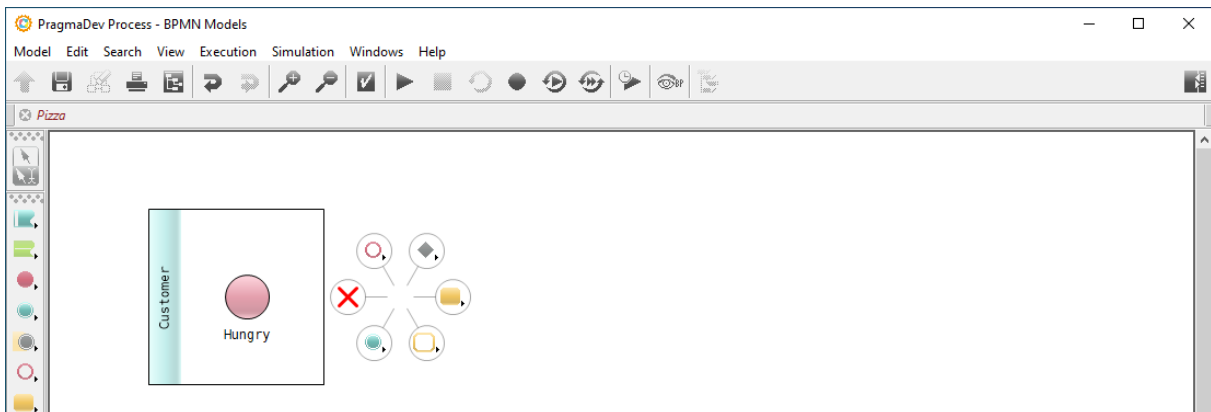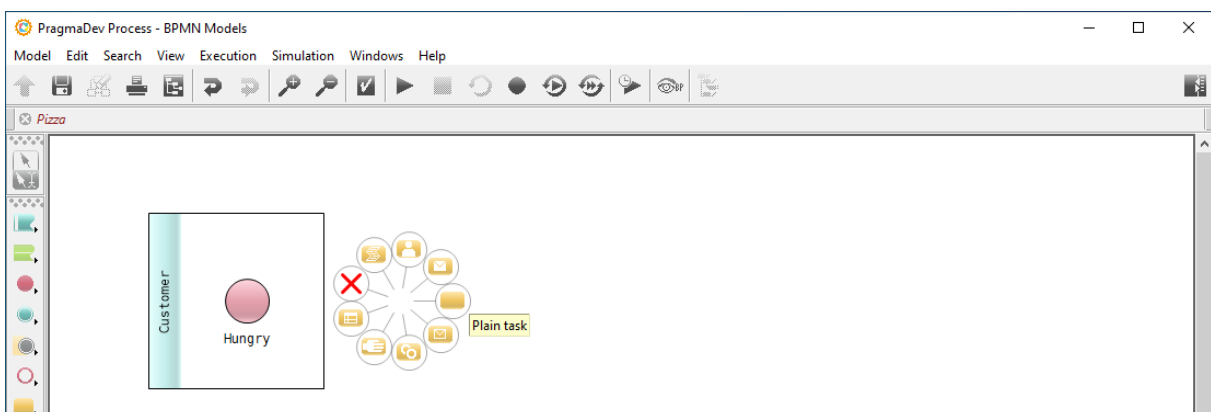
Adding a successor is done by pressing the mouse button inside the middle one, then drag to where we want to put it:
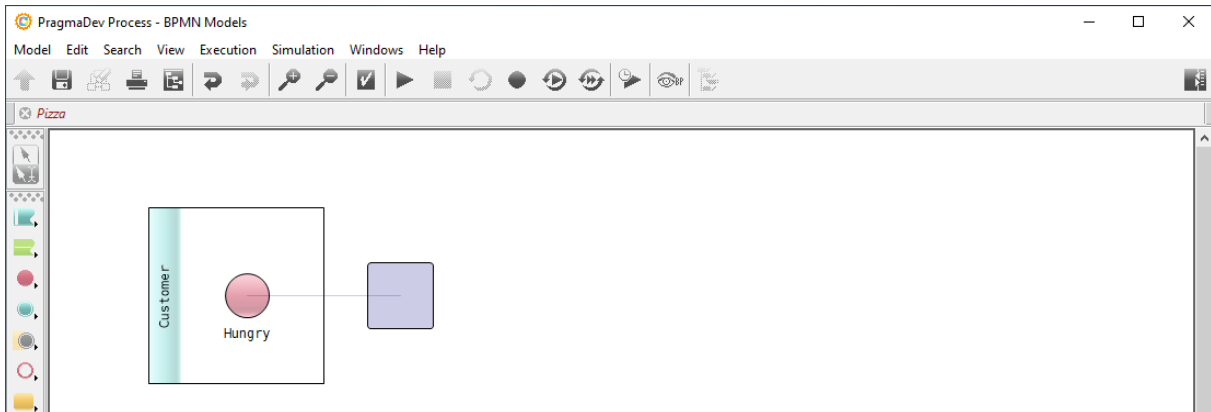
When the button is released, a menu is displayed to select the actual type for the successor:
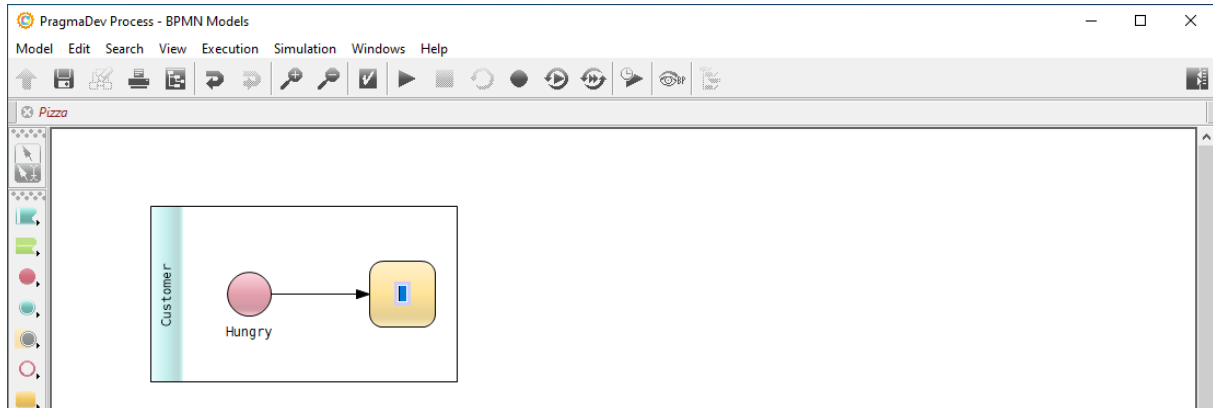

Here, we want a task, so we click on the rightmost item, showing a task. A secondary menu is displayed, allowing to choose the actual type for the task to create:

We want a plain task, so again, we click on the rightmost item, and the successor task is created:


Notice that the parent pool has automatically adjusted its size to contain the new symbol. This is because pools always contain whole processes, so since the new symbol is a successor of the start, it is in the same process, so the parent pool must include both. We can now set the actual name for the task:
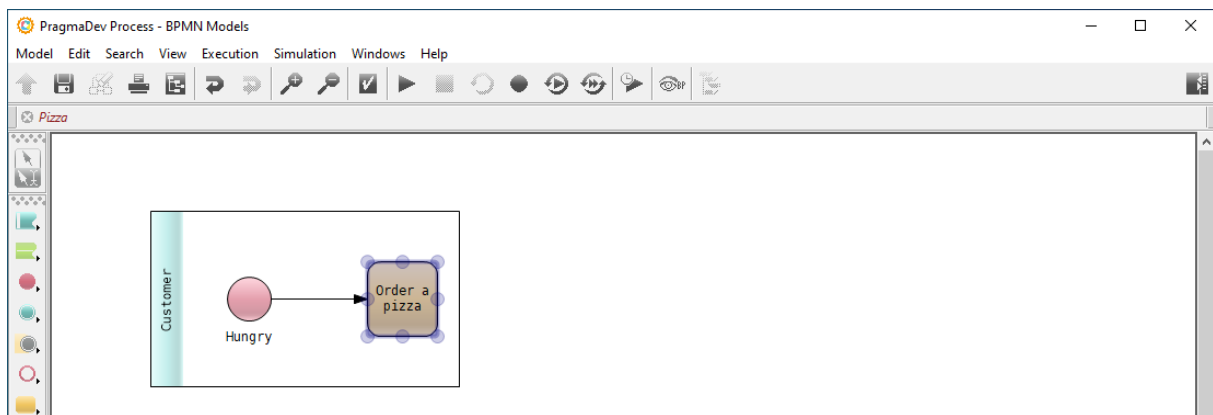

And we're done. We can now continue adding successors via the hover buttons until the whole process is created:
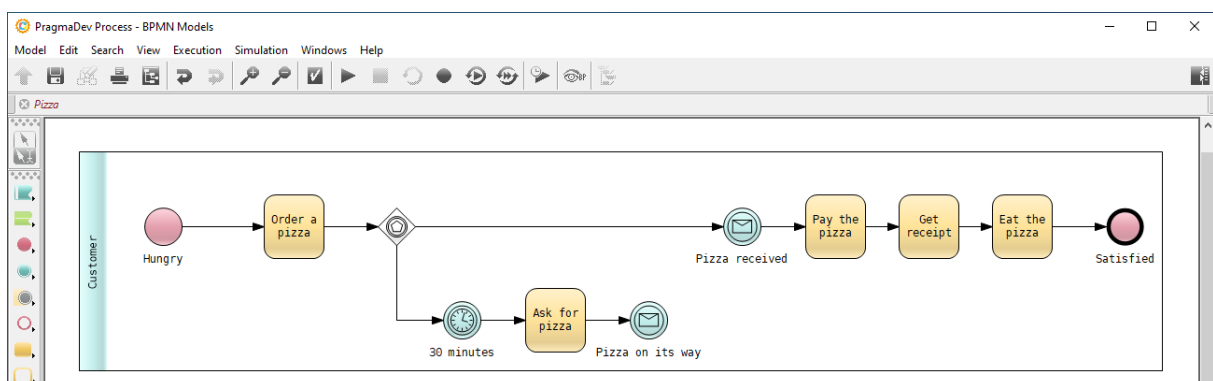
Notice that we left out something in the process: the sequence flow going from the "Pizza on its way" message catch event back to the event-based gateways is missing. This is because it cannot be inserted the same way, since the successor already exists. But there's another hover button that we can use:
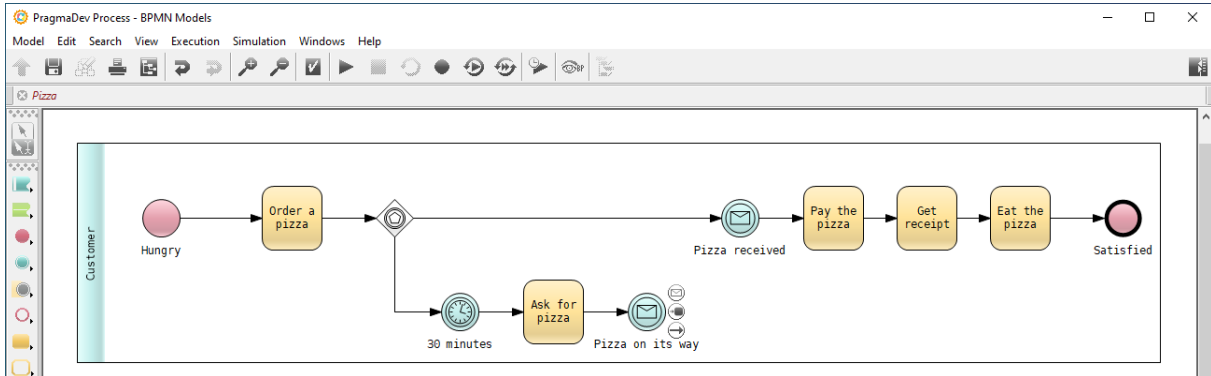
The hover button at the bottom allows to quickly draw a sequence flow from the symbol to any other one. Clicking and dragging to the target symbol would create the flow with the default path, but we don't want that. So we'll click on the hover button, release the mouse button, then shift+click on every corner we want for our sequence flow:
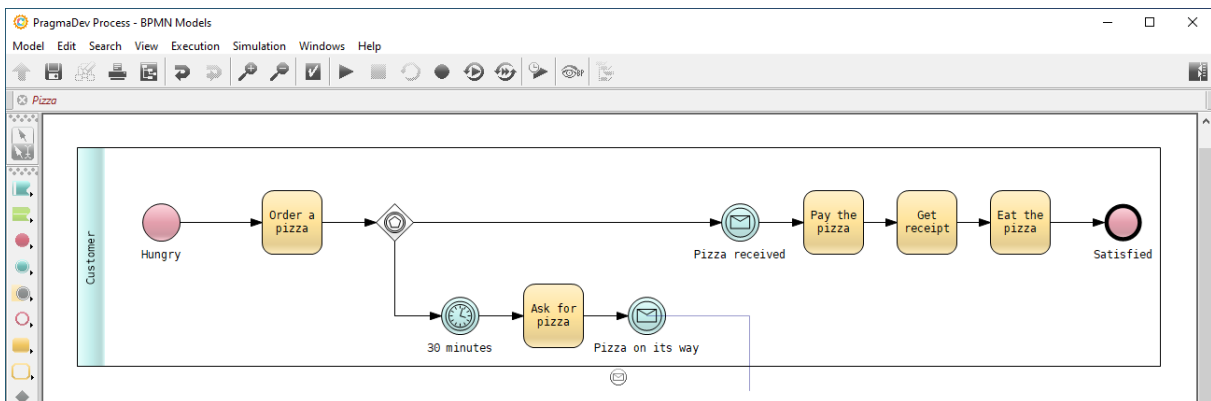
Once the event-based gateway is clicked, the link is created just as we want it, and the parent pool is automatically resized to include everything:
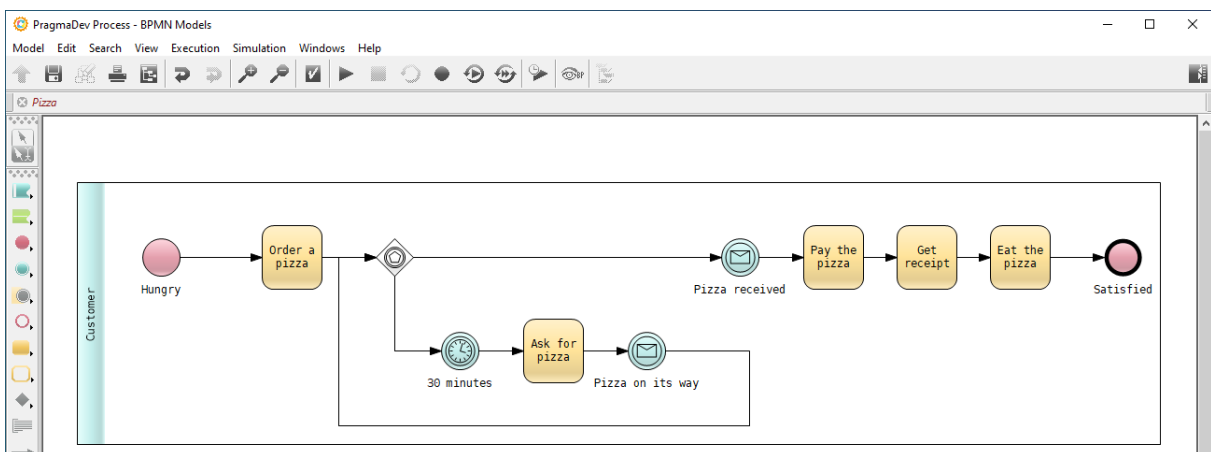
Note: when creating symbols via the keyboard shortcuts, the secondary circular menu allowing to select the actual type for the symbol to insert will also be displayed.

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 sequence of horizontal and vertical segments by default. Should you need to change their path, you can either move the points at the corners (called "waypoints") by selecting the link, then move the circular handle that will appear at the corner:
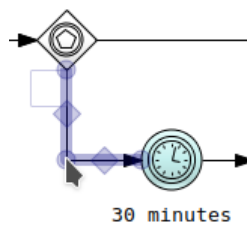
or move any of its horizontal or vertical segments by using the diamond-shaped handle in the middle of the segment:
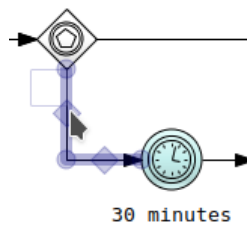
Waypoints can also be added by right clicking a flow and select "Add waypoint" in the contextual menu, then move the waypoint to the desired position:
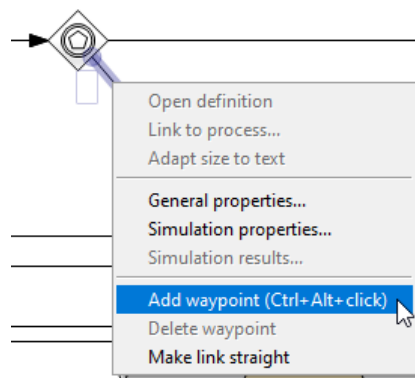
Custom non-linear paths can be created also by holding "Shift" and clicking to place waypoints while drawing the path from the source to the target symbol.
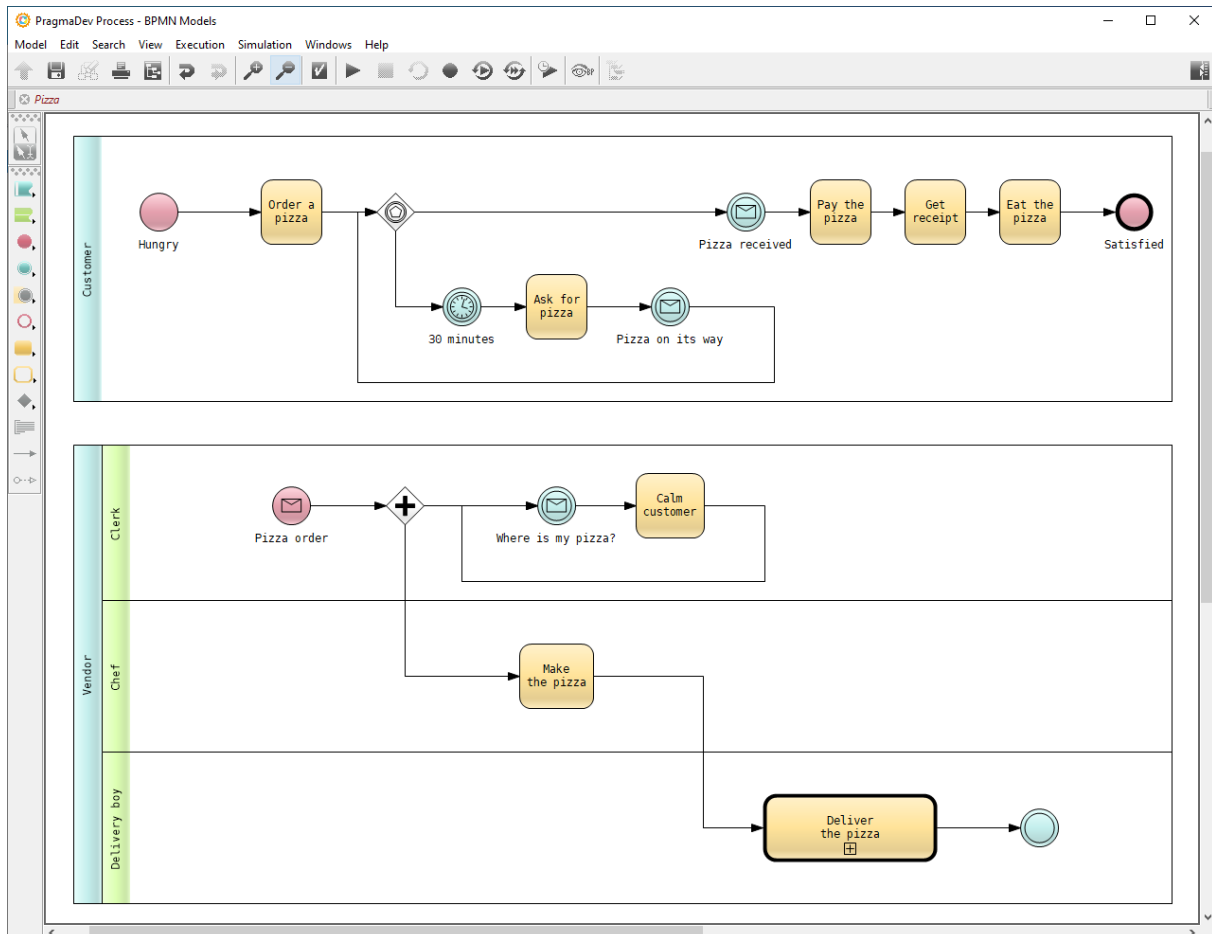
The Customer process says the following:

- 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 `30 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 the `Clerk` will try to `Calm customer`.
- The `Clerk` will also forward the received order to the `Chef` to `Make the pizza`.
- When ready, the `Chef` will hand over the pizza to the `Delivery Boy` to `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, for example using the 'message flow' hover button (the first one) which is used the same way as the 'sequence flow' hover button:
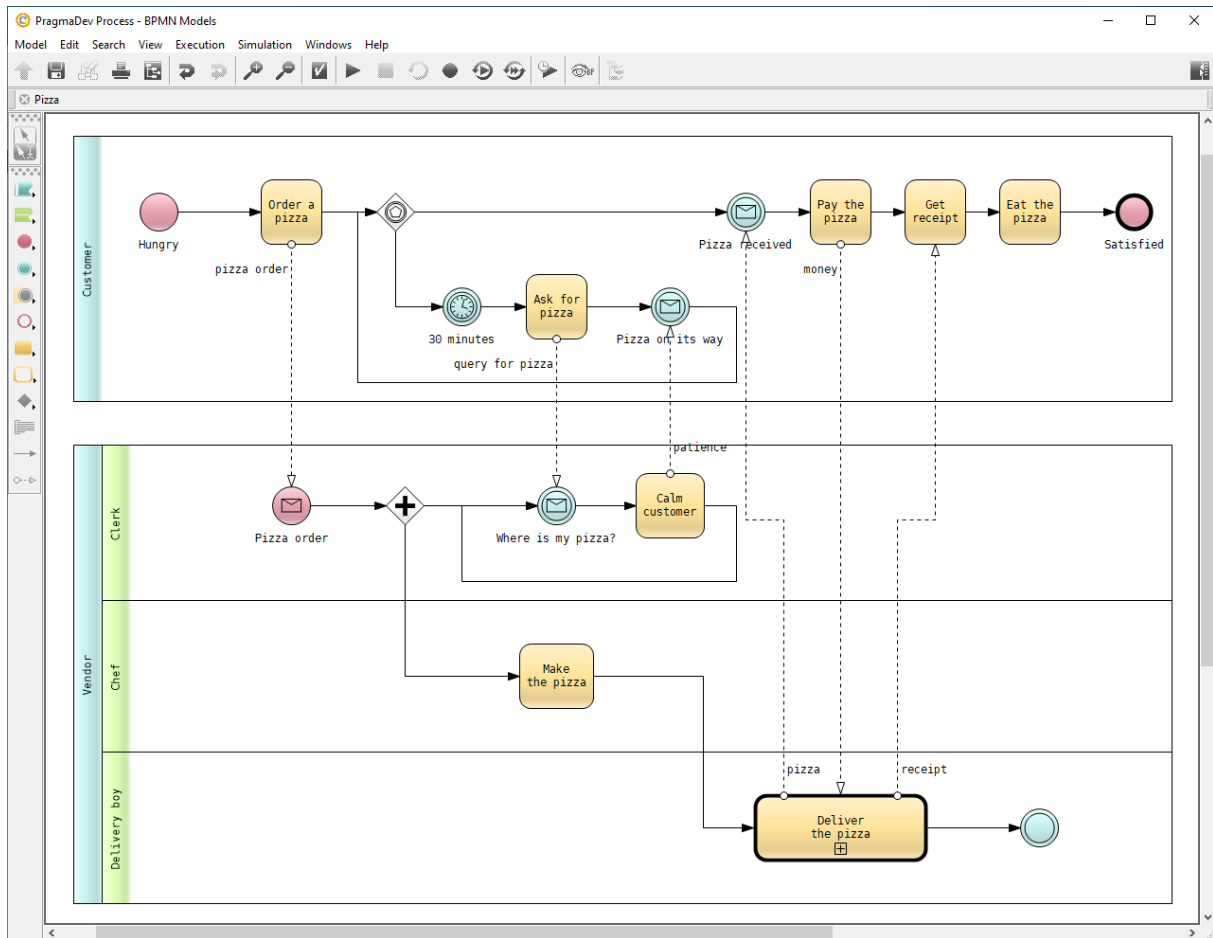
Let's define now the call-activity `Deliver the pizza`. Add a new diagram via the button at the bottom of the diagram browser:


Name it `delivery` and hit "OK":
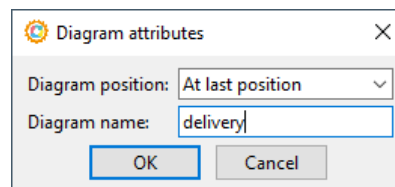
The new empty diagram will be selected. Draw the following:
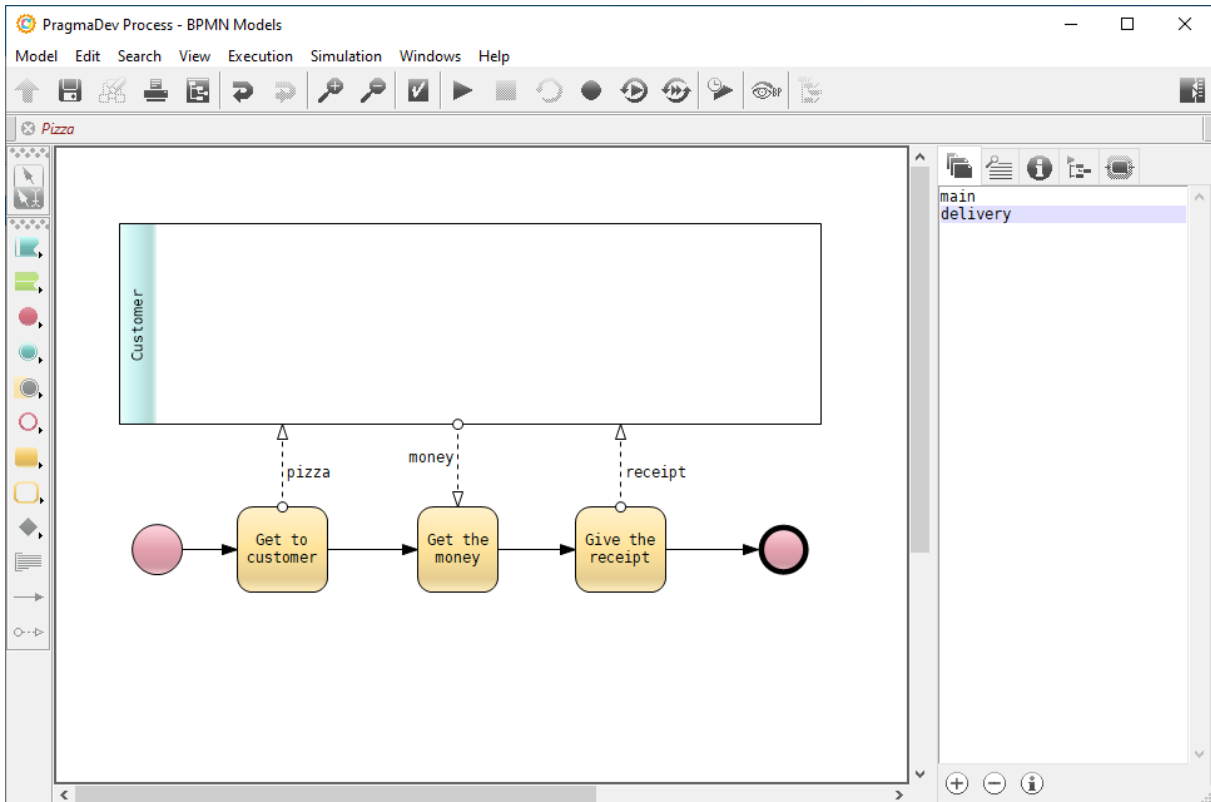
When creating the message flows be sure to draw starting from / ending to either the border or header of the pool Customer. Note that pools also have a hover button to create message flows from them, so you can use that to create the one named money.

Save the changes and go to the main diagram. Right click the call-activity and then "Link to process...":
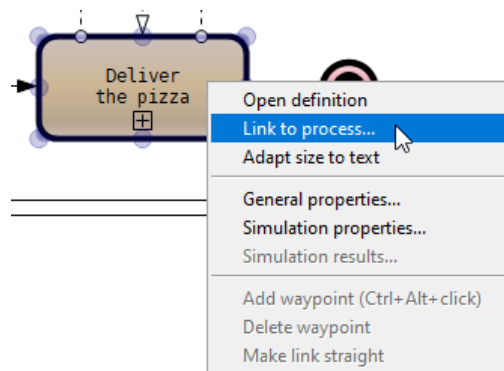
Select the single available option (except *none*) in the dialog and hit "OK":
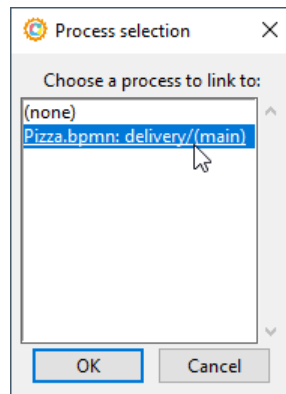
Right click the call-activity and then "Open definition":
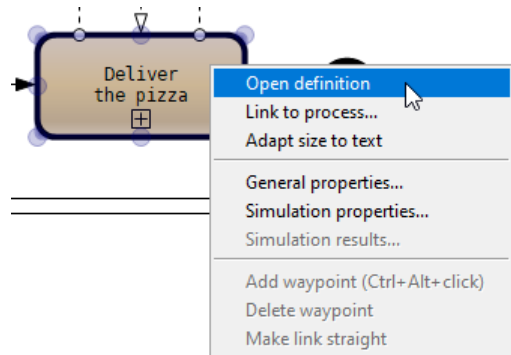
This should display the `delivery` diagram in the editor.

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:
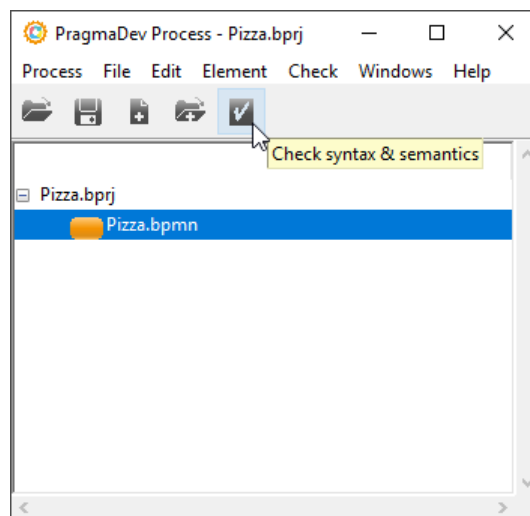
A window will pop-up showing the result of the semantic check on the model:
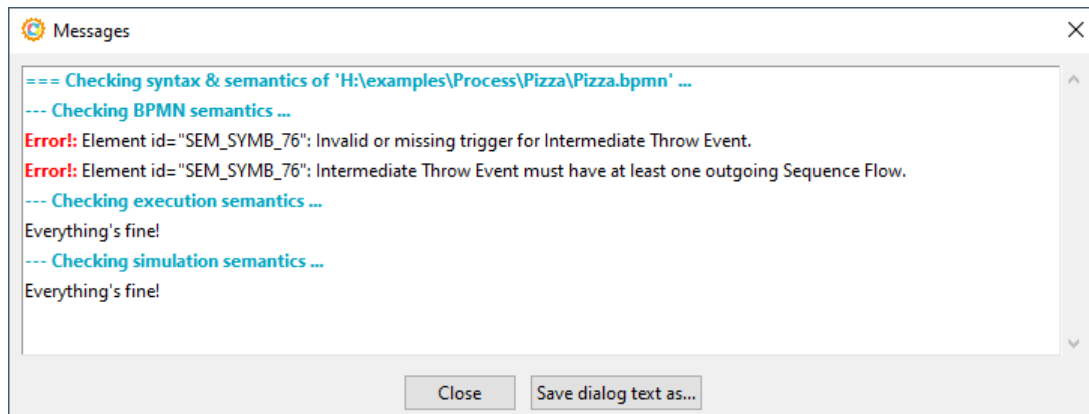
Double-clicking either of the listed errors will open the model in the editor and the concerned element will be selected:
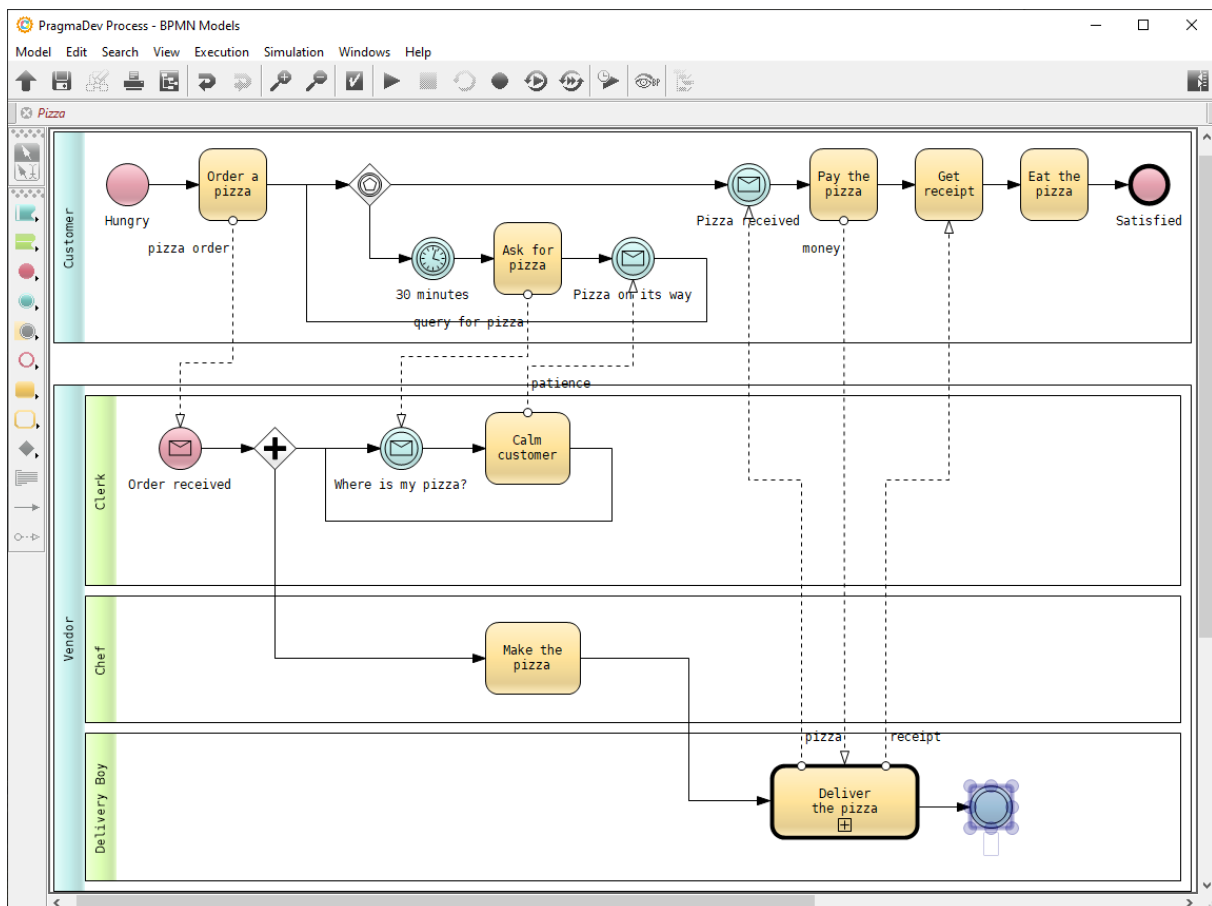
Replace the intermediate event with a terminate end event to correct the errors:
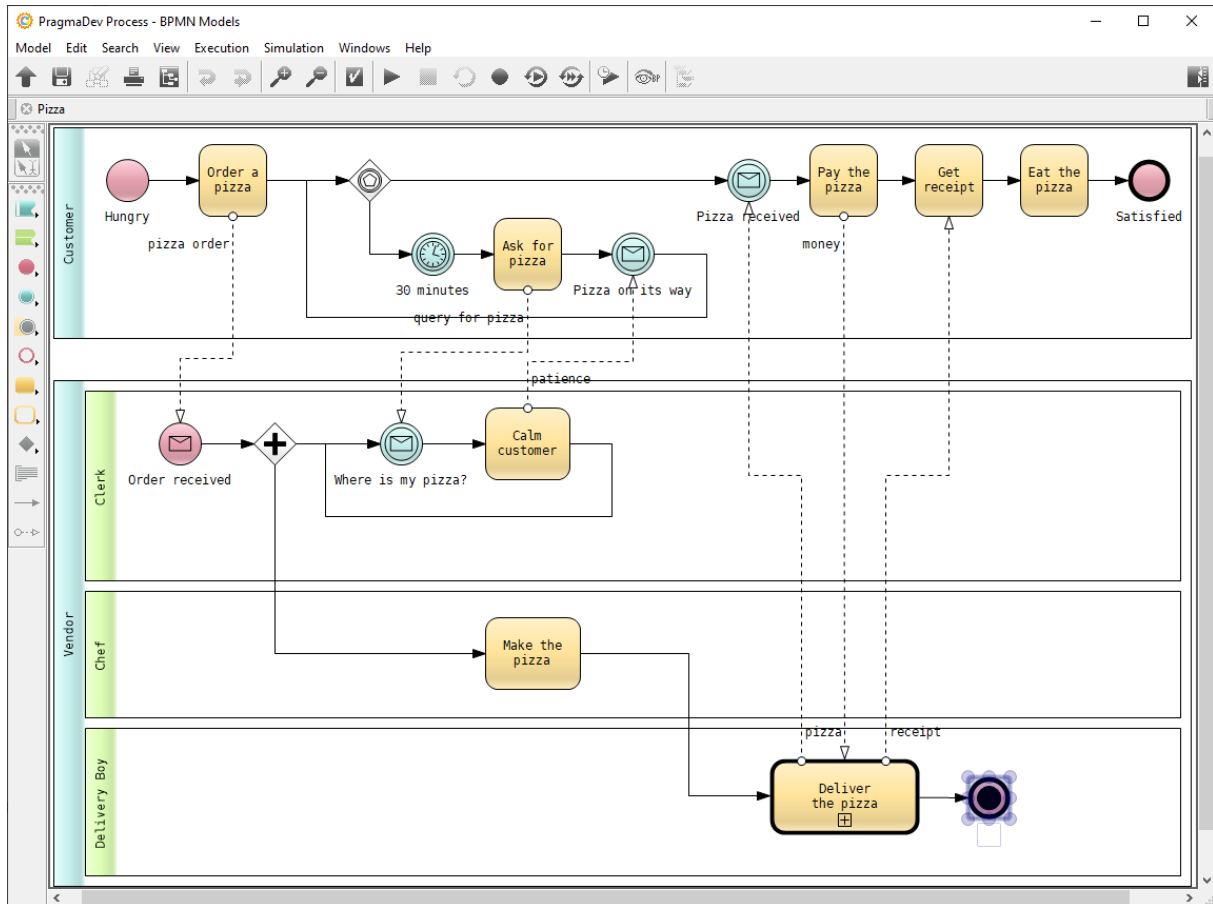
Re-running a semantic check on the model should not list any errors:
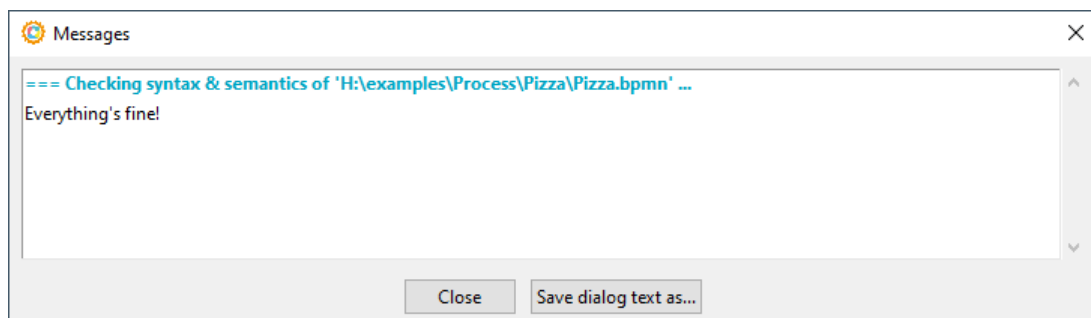
3.2 Interactive execution

With `Pizza.bpmn` opened in the editor (double-click on it in the project manager), click the "Record" button:
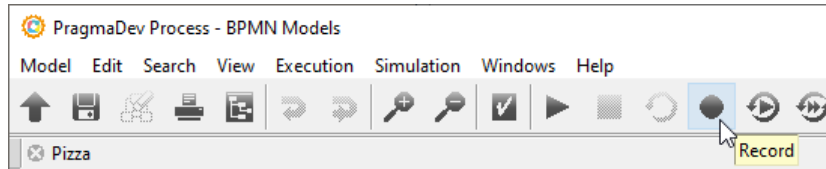
This will allow us to record the execution in the *MSC Tracer*:
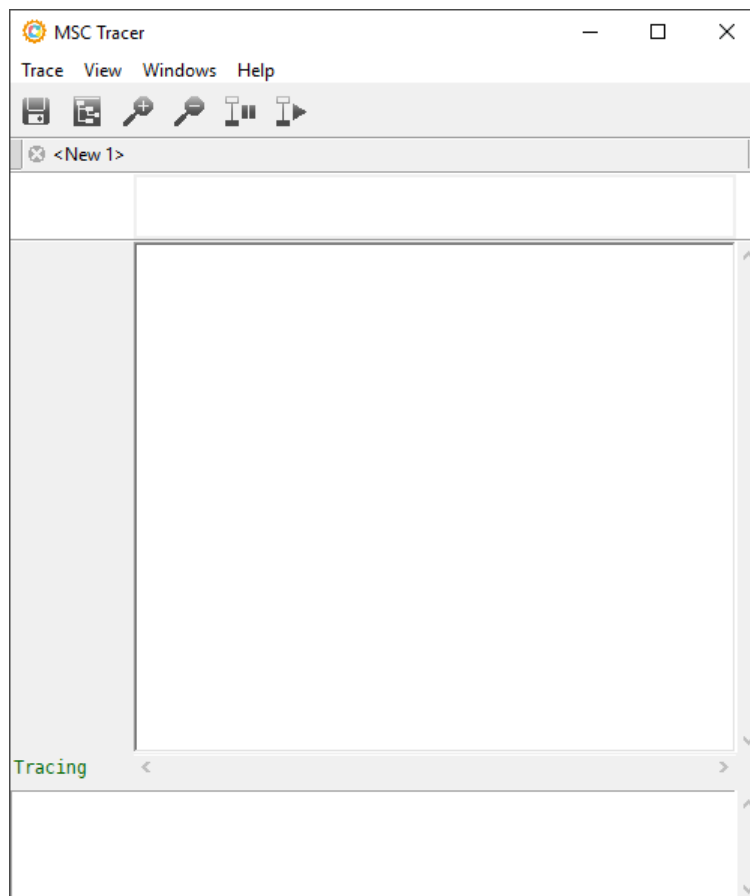
We can now start the execution via the "Start" button:
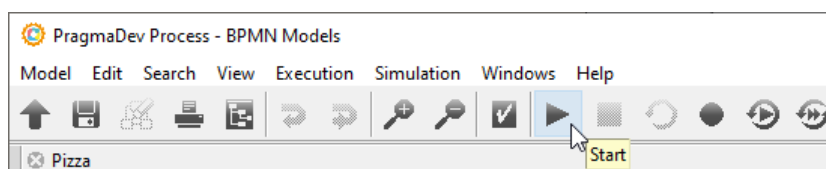
Start events and executable flows will be marked in blue:
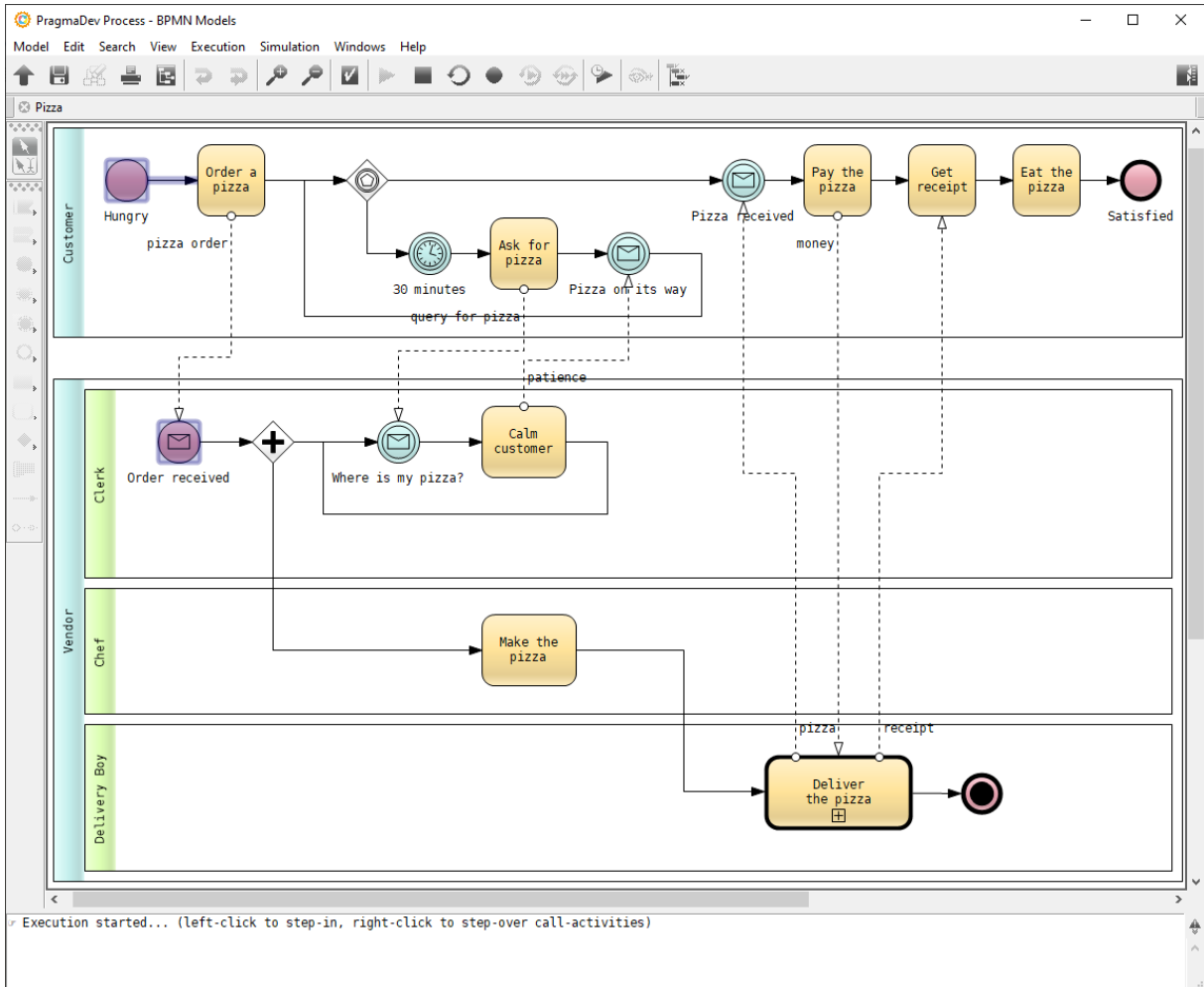
Please note that generally speaking only sequence flows, some message flows, or call-activities can be clicked. No other symbol can be clicked even though they appear in blue. In our example there is a single sequence flow that can be executed. Clicking on it will advance execution, i.e., make the `Customer` order a pizza:
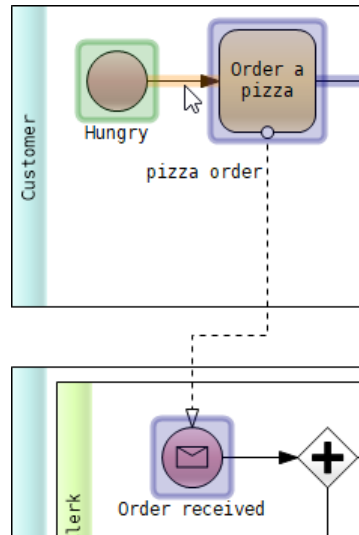
The next (and only) step will be for the `Customer` to send the `pizza order` to the `Clerk`; to continue click the sequence flow:
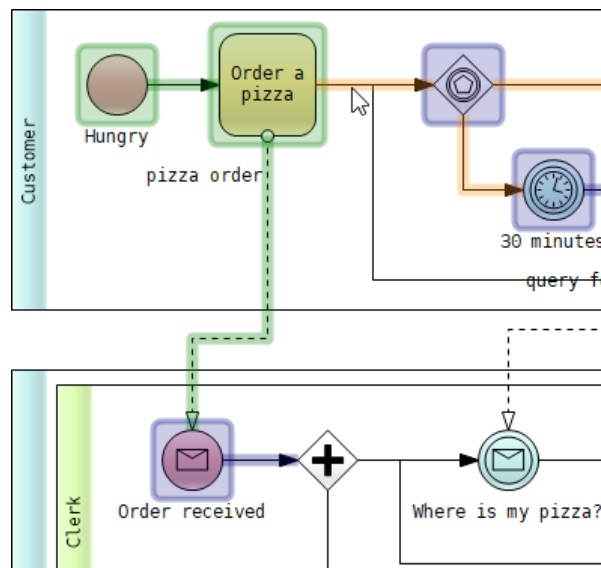
The `pizza order` will start the `Vendor` process. Clicking the sequence flow will result in:
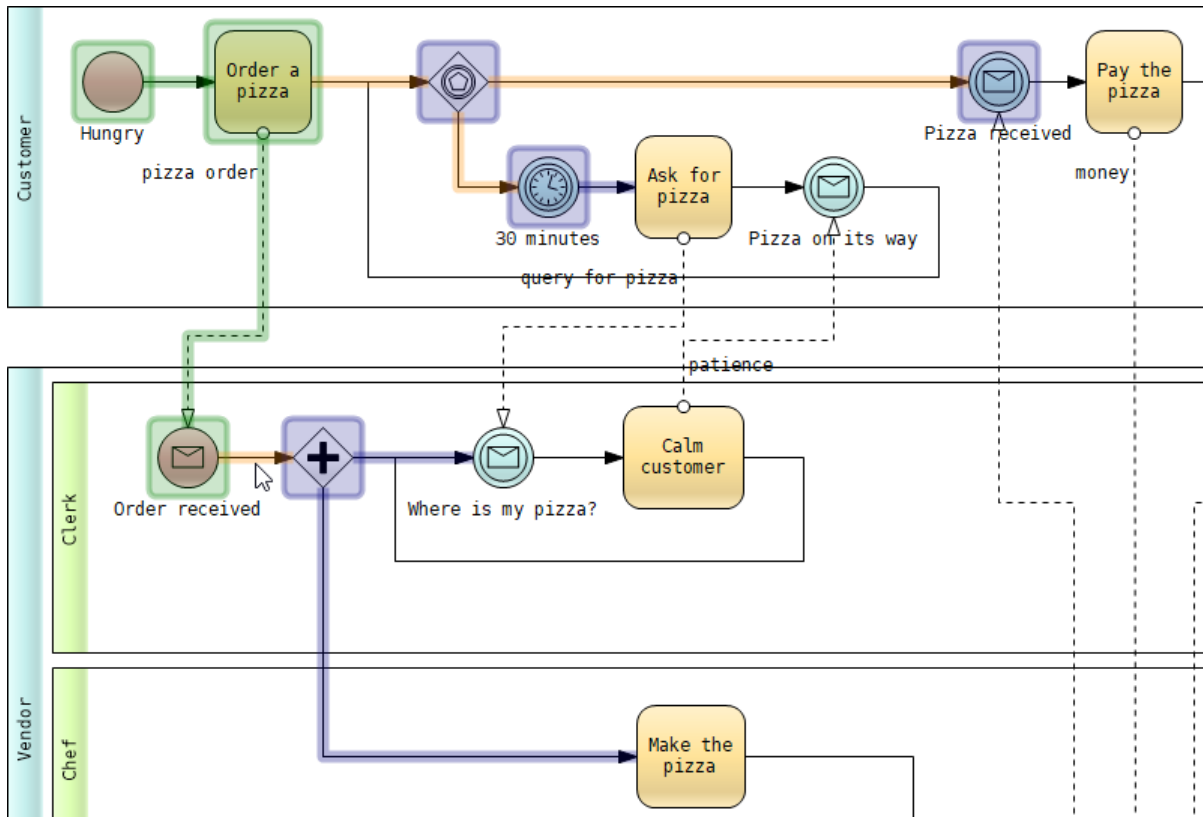
By entering the event gateway, the `Customer` will wait for its pizza to be delivered (i.e., the `Pizza received` event) and start a `30 minutes` timer. If the timer fires before the pizza is delivered, the `Customer` will ask the `Clerk` for the pizza. On the other hand, the parallel gateway in the `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*:
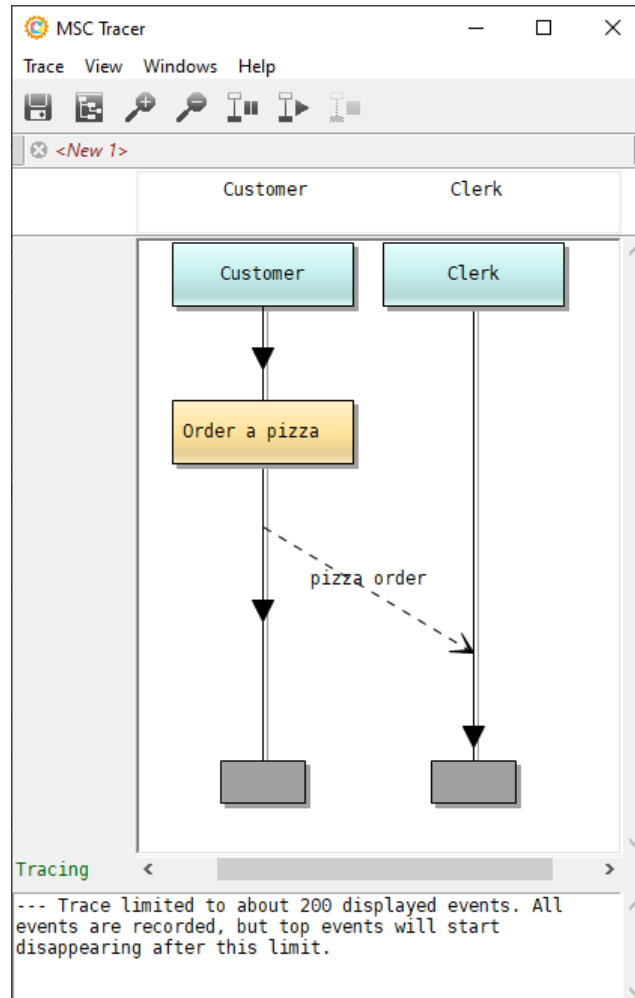
Let's make the timer fire, the `Clerk` listen for `Customer` queries, and the `Chef` start baking the pizza by clicking all three sequence flows:
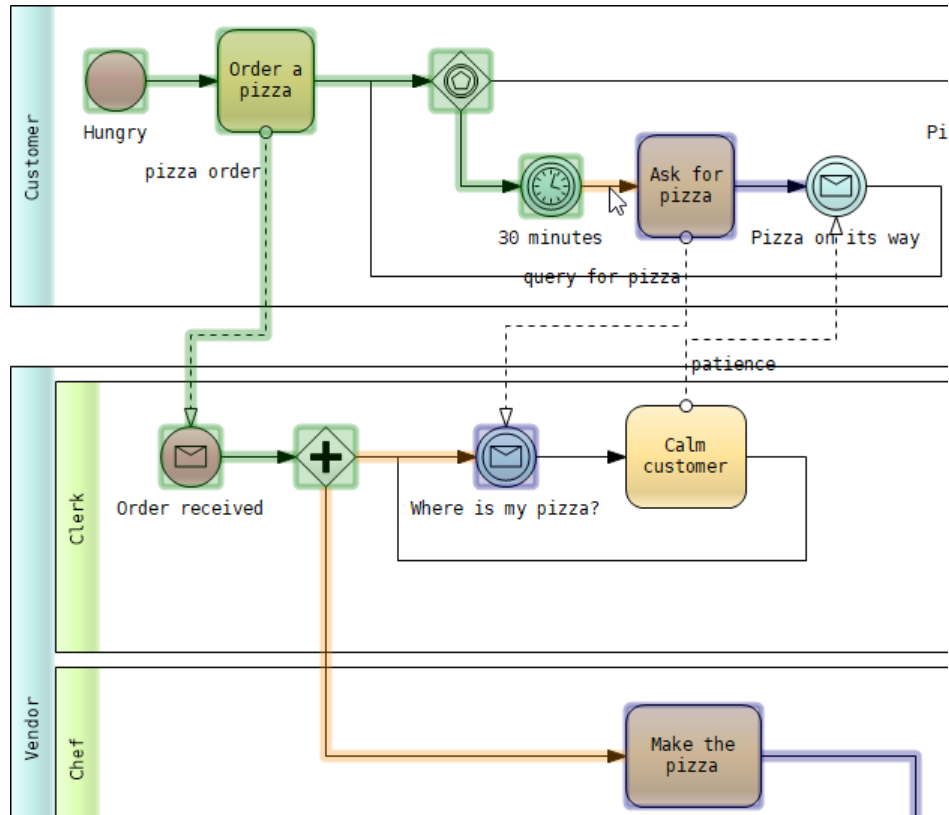
Notice that the `Pizza received` event is no longer active because the `30 minutes` event was triggered. In this situation the `Customer` is asking the `Clerk` for the pizza, while the pizza is being prepared by the `Chef`. If `query for pizza` is sent:
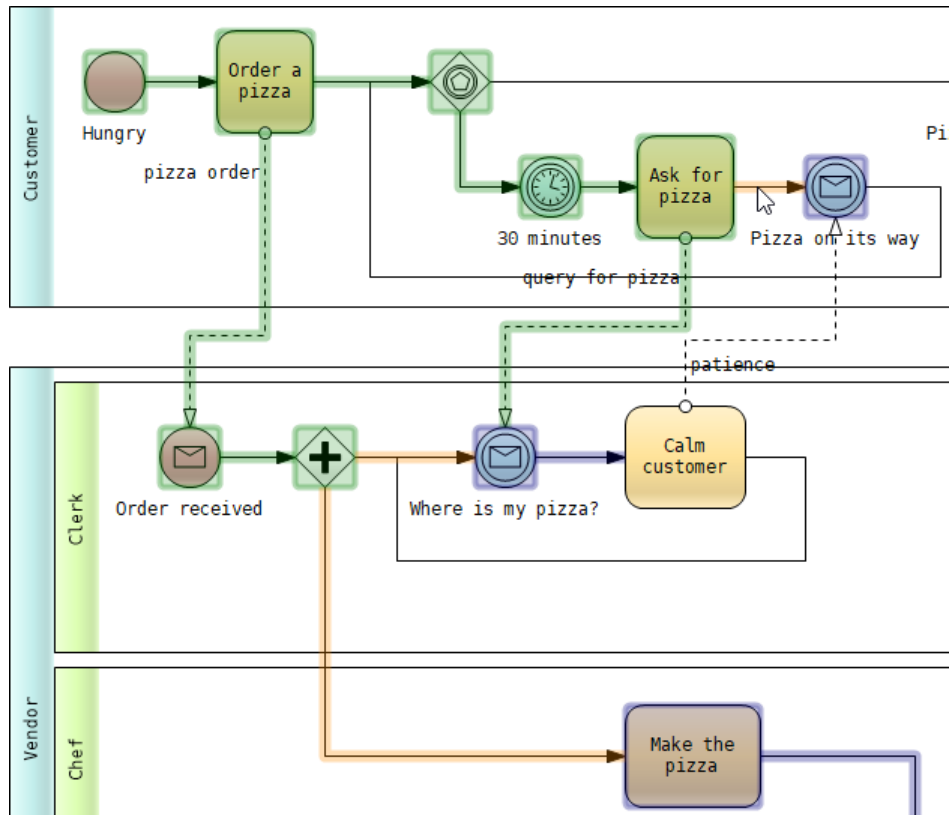
By clicking the sequence flow, the `Clerk` will `Calm customer` while the later will be waiting for a feedback (i.e., the `Pizza on its way` event):
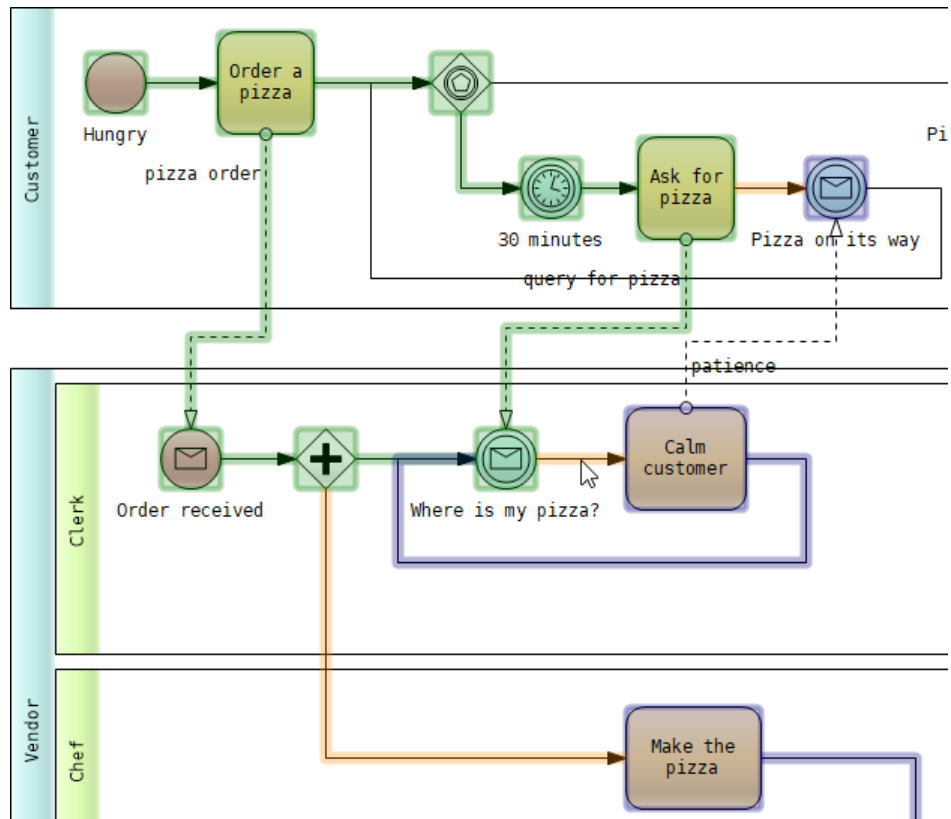
Make the `Clerk` respond to the `Customer`'s query by clicking the sequence flow which will automatically send the `patience` message:
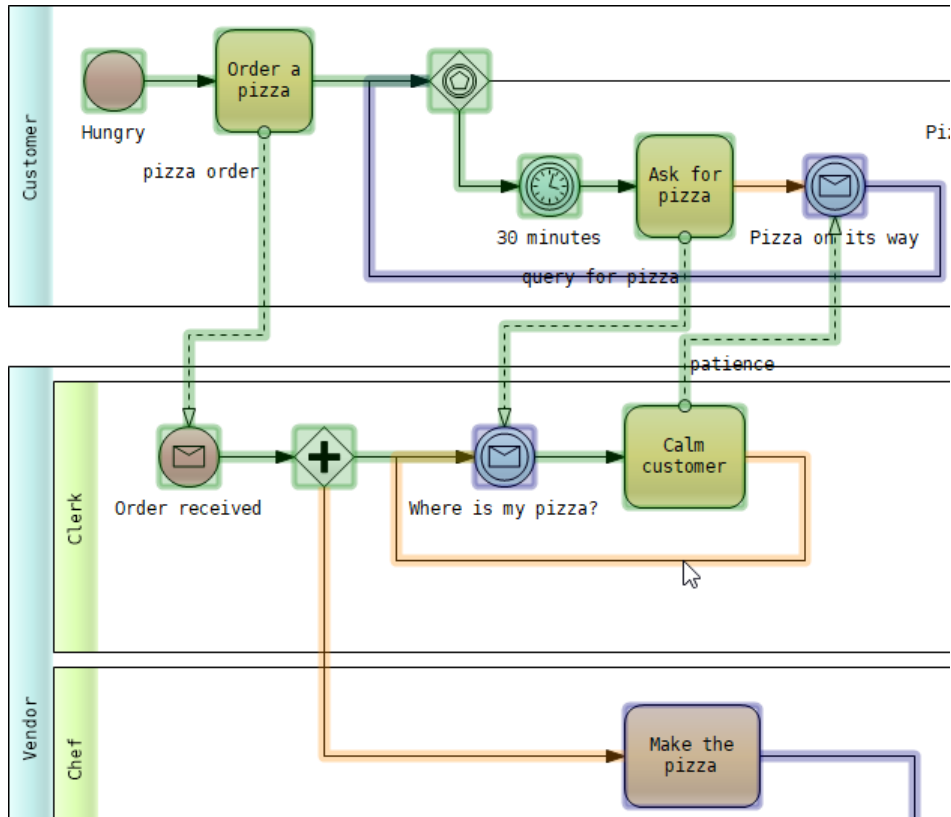
Continue by clicking the looping sequence flow in the `Customer` process:
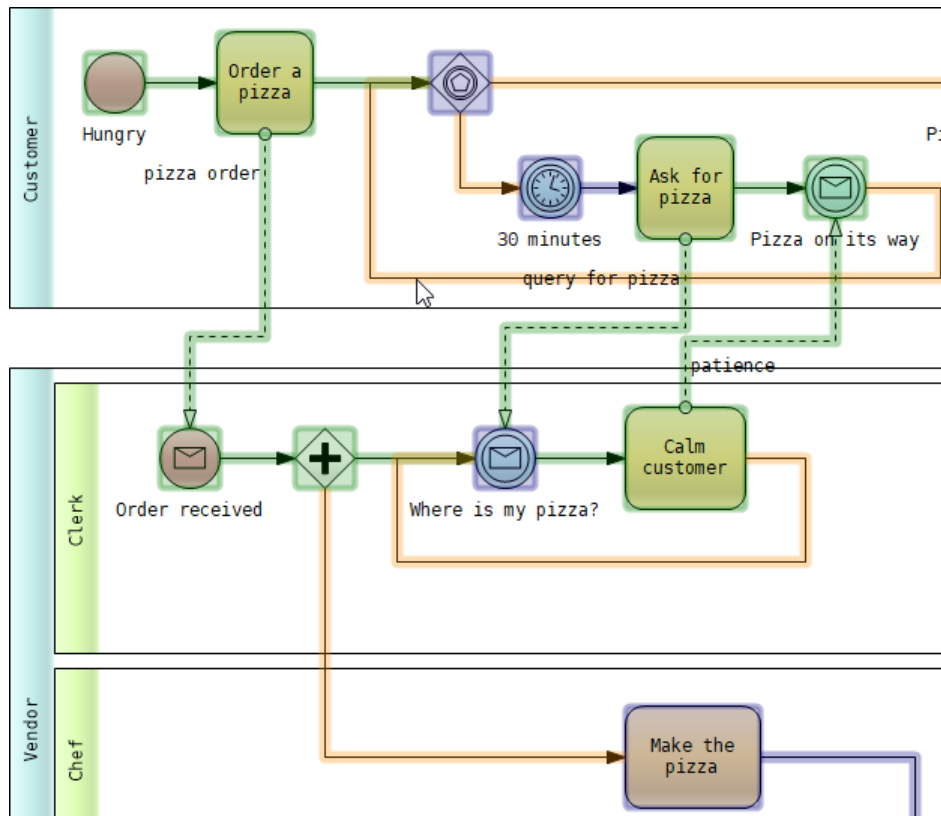
The `Customer` will get back to waiting for the pizza or for another `30 minutes` to pass before querying the `Clerk` again. We will not let the `Customer` wait any longer, so let's make sure the pizza is ready by clicking the sequence flow outgoing `Make the pizza`:
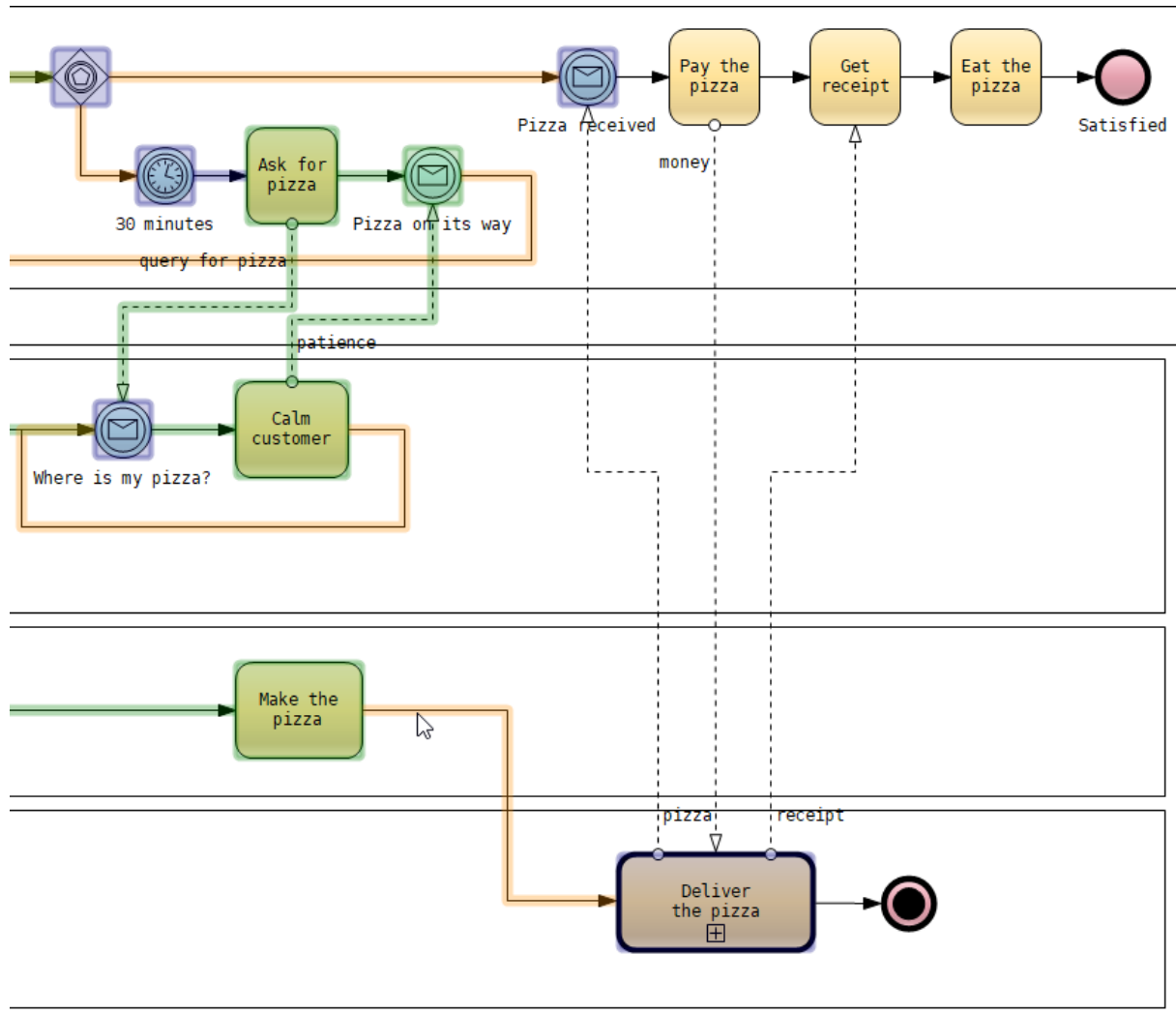
Now it's time to `Deliver the pizza`. This is a BPMN call-activity which calls the process defined in the `delivery` diagram. We can either step-over (right-click) or step-in (left-click) the call-activity. Stepping over the call-activity will not execute the called process, but will consider the activity 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 show the `delivery` diagram in the editor and start the called process:
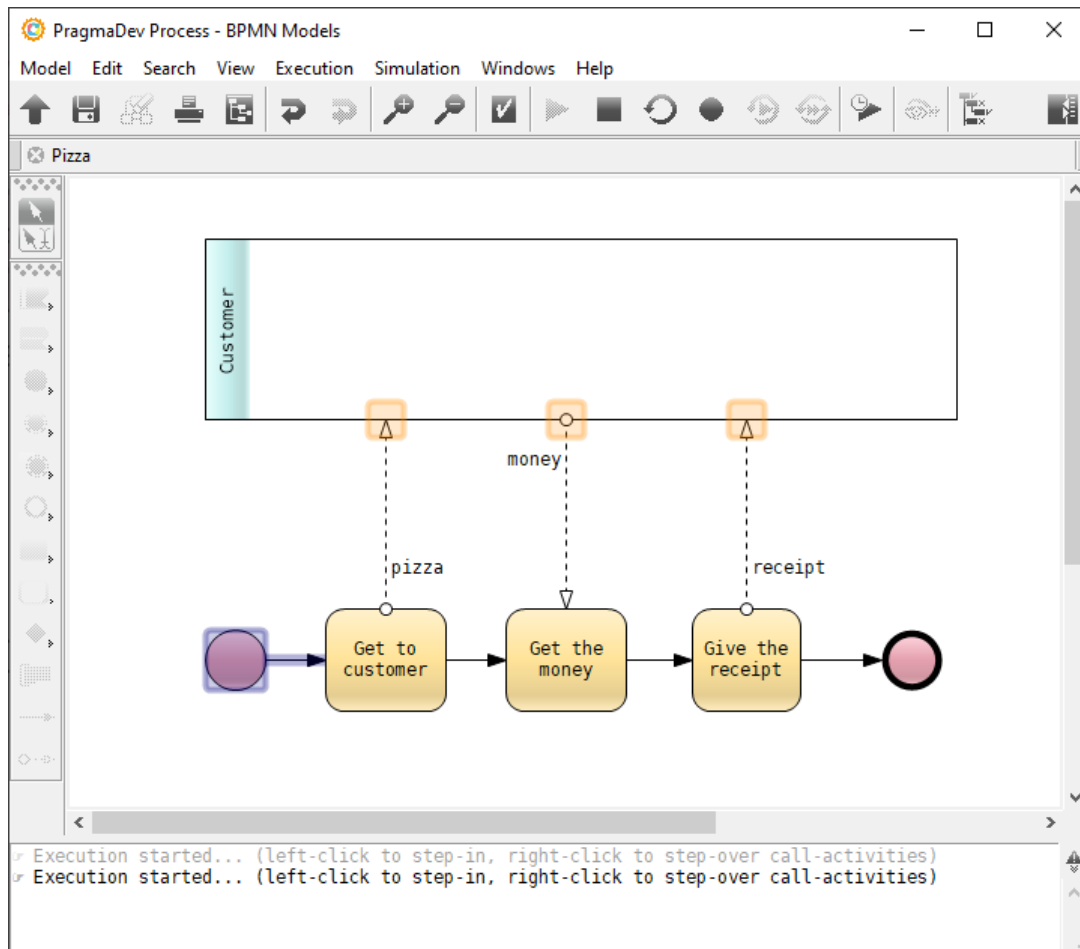
We can `Give the pizza` to the `Customer` by clicking the sequence flow:
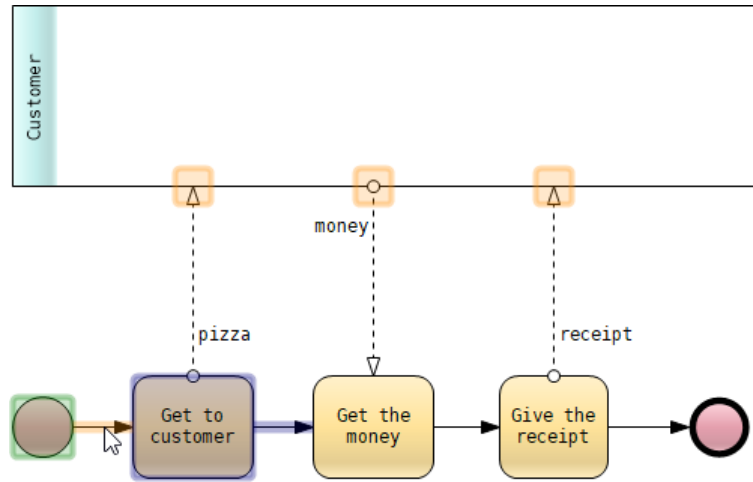
and then the following sequence flow which will automatically send the `pizza` message:
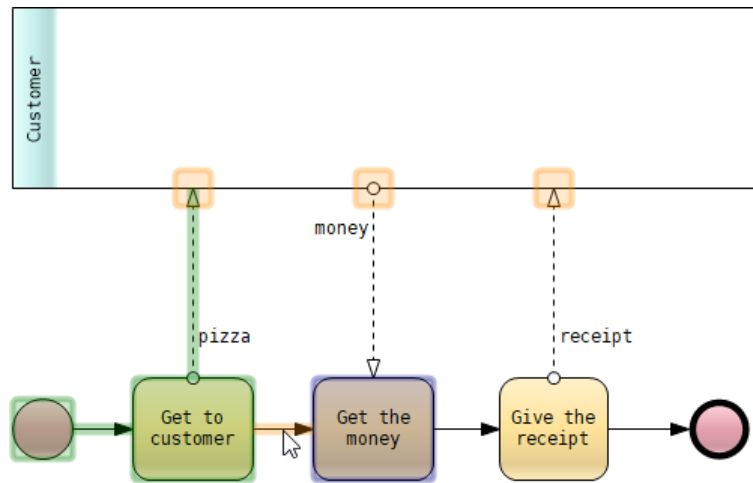
If we switch to the `main` diagram in the editor we can see that the `Customer` has indeed received the pizza and can proceed with the payment:
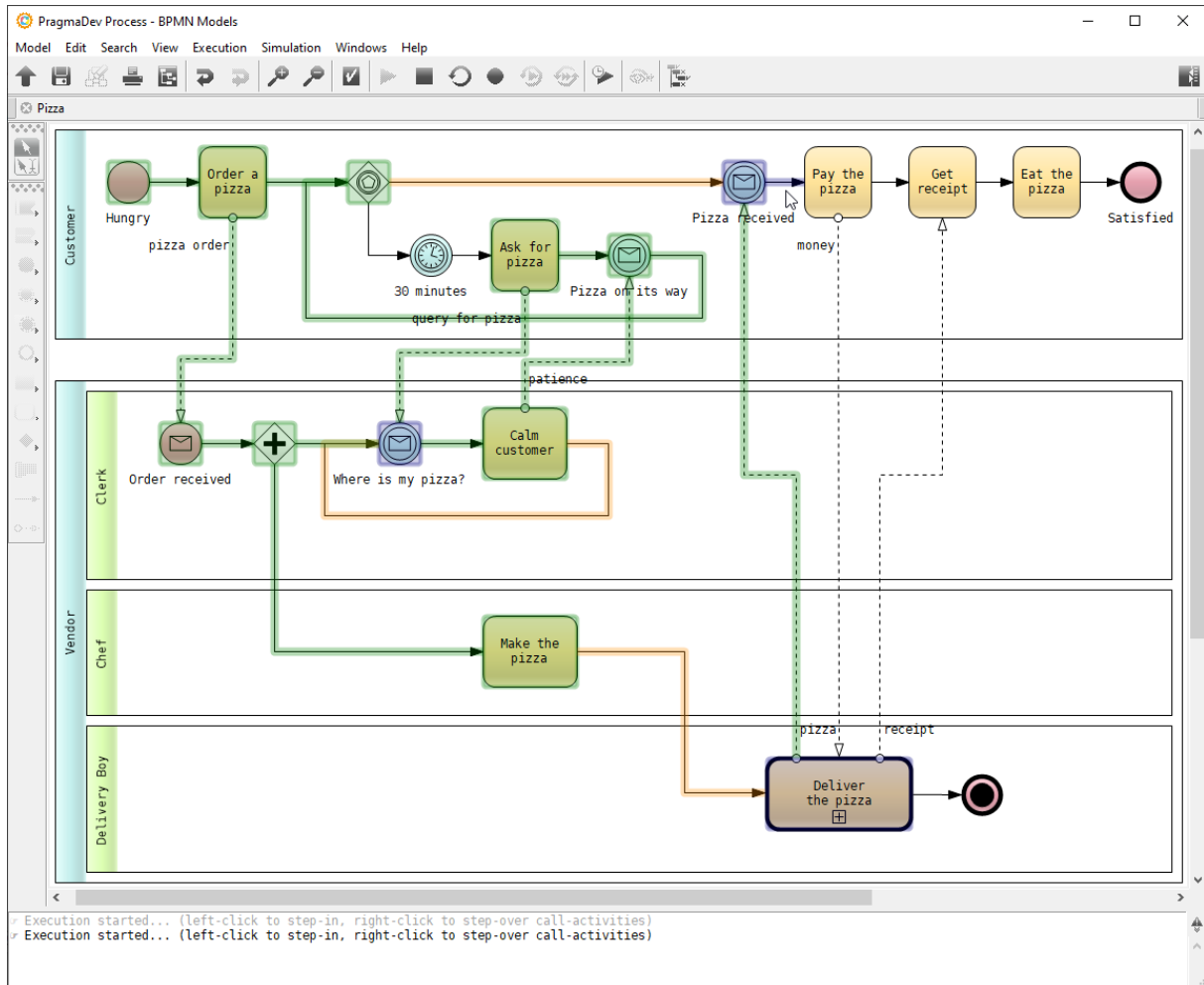
Click the next two sequence flows to give the money to the Delivery Boy:
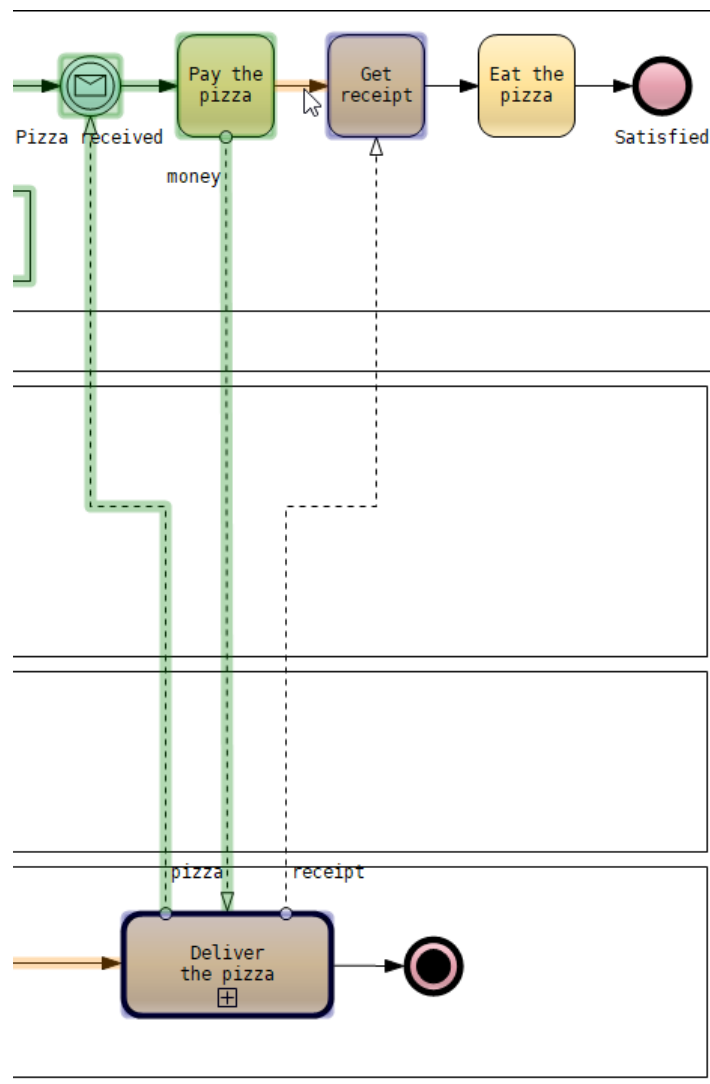
Continue by clicking all possible flows in both diagrams until there is nothing left to do, i.e., execution is finished:
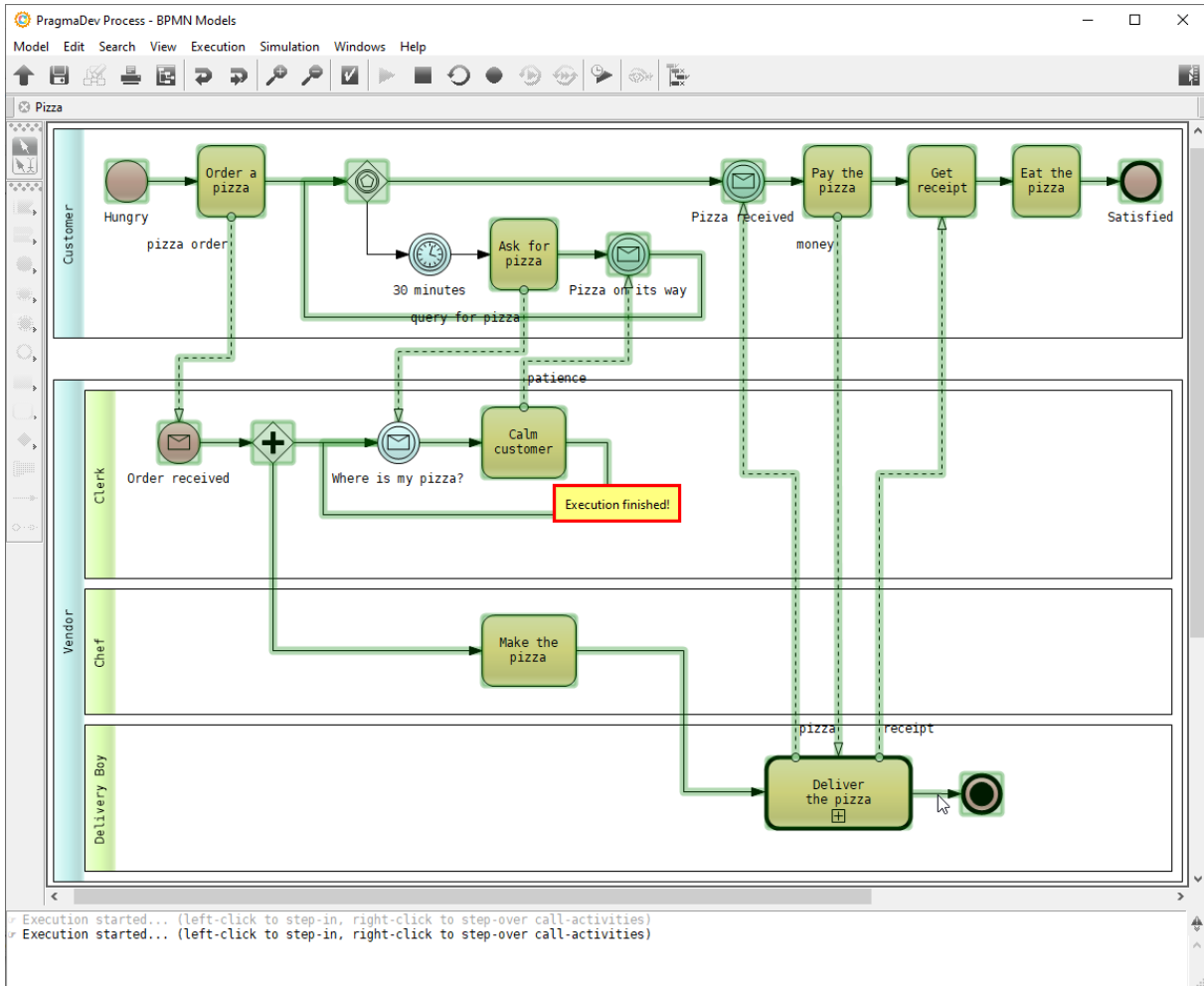
Note that in the MSC trace, activity starts and ends might be traced separately. This happens for example if you click the incoming sequence flow for the activity "Eat the pizza" for the Customer, then click the outgoing sequence flow for the "Deliver the pizza" call activity going to the terminate end event, and only after that click the ougoing sequence flow for "Eat the pizza". In this case, something is happening between the begining and the end of "Eat the pizza", so they will be traced separately:
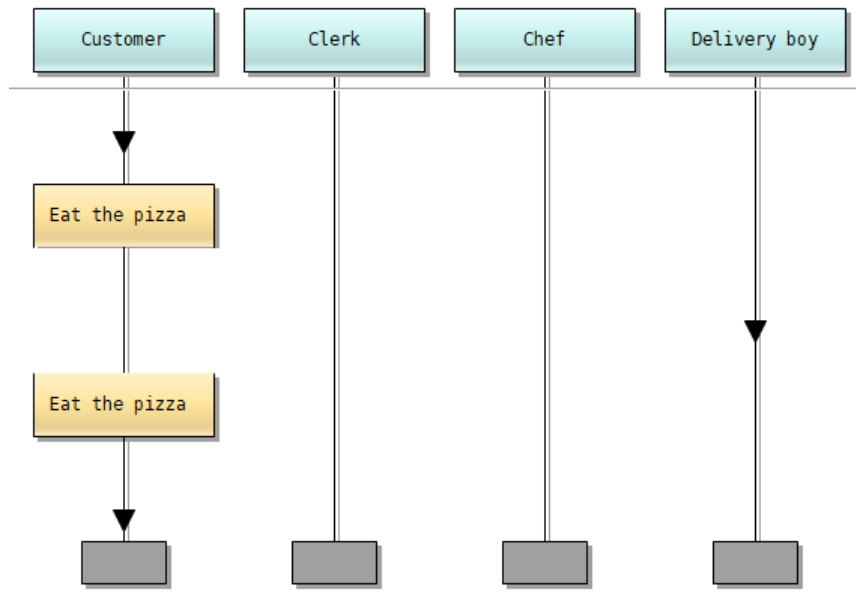
To generate model coverage click the "Generate coverage" button:
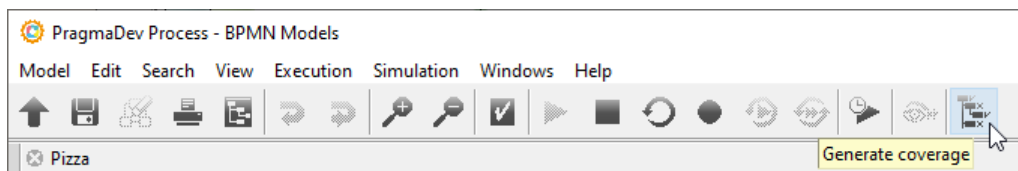
Coverage information will be shown as follows:

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:
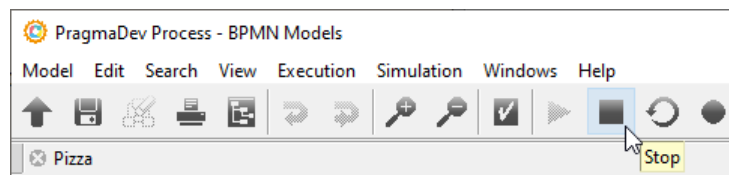
Save the recording via the "Save trace" button in the *MSC Tracer*:
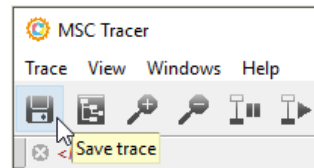
Give it a name (e.g., `pizza_ok`), and it will be added automatically to the project:
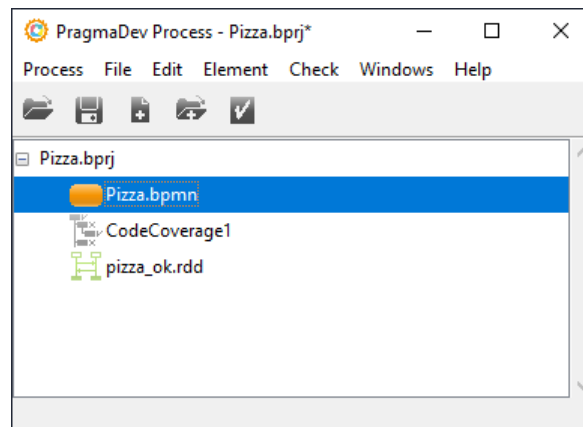
3.3 Automatic execution

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:
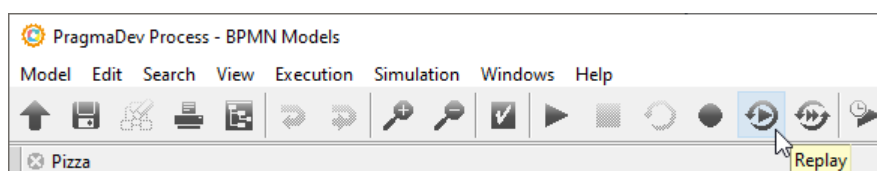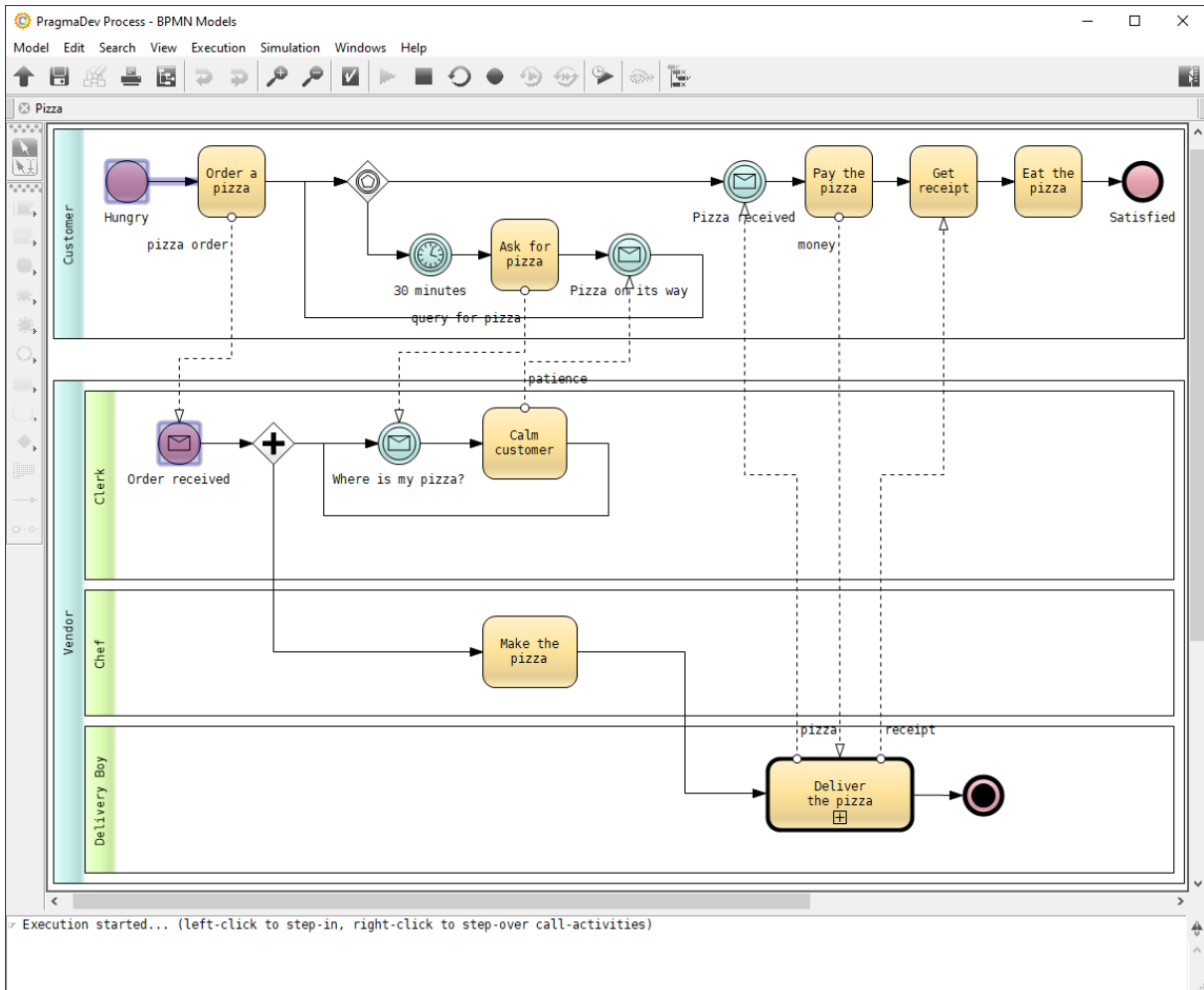
All recordings (MSC traces) found in the project will be listed; select `pizza_ok` and click the "OK" button:
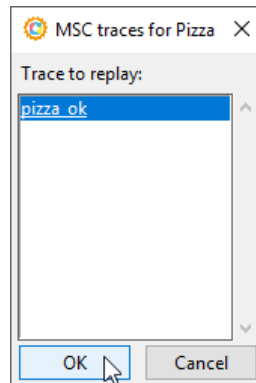
The `pizza_ok` will be opened in the MSC editor:
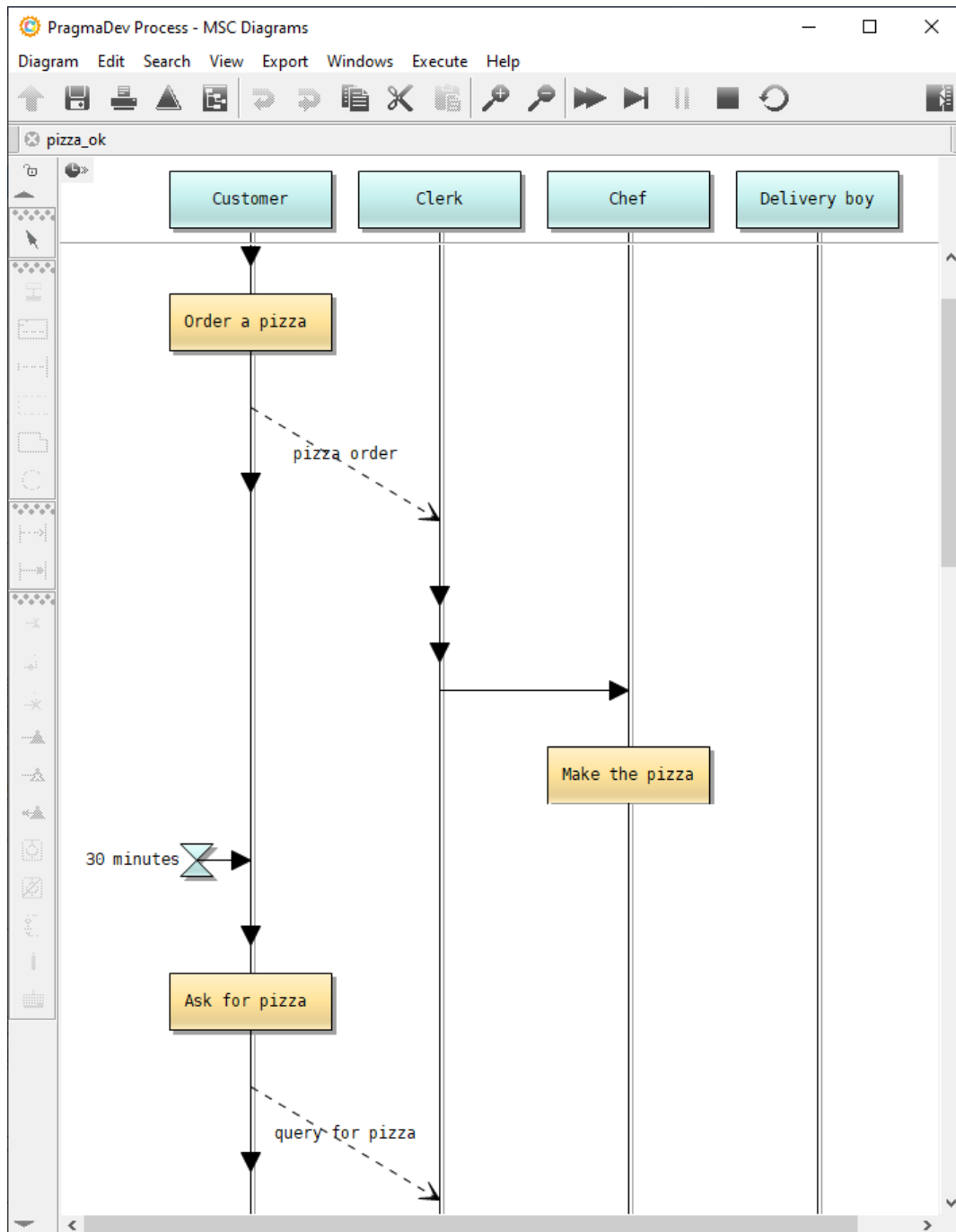
while the execution will start automatically in the BPMN editor:

Click the "Step" button in the MSC editor:

The first sequence flow will be selected in the MSC editor:
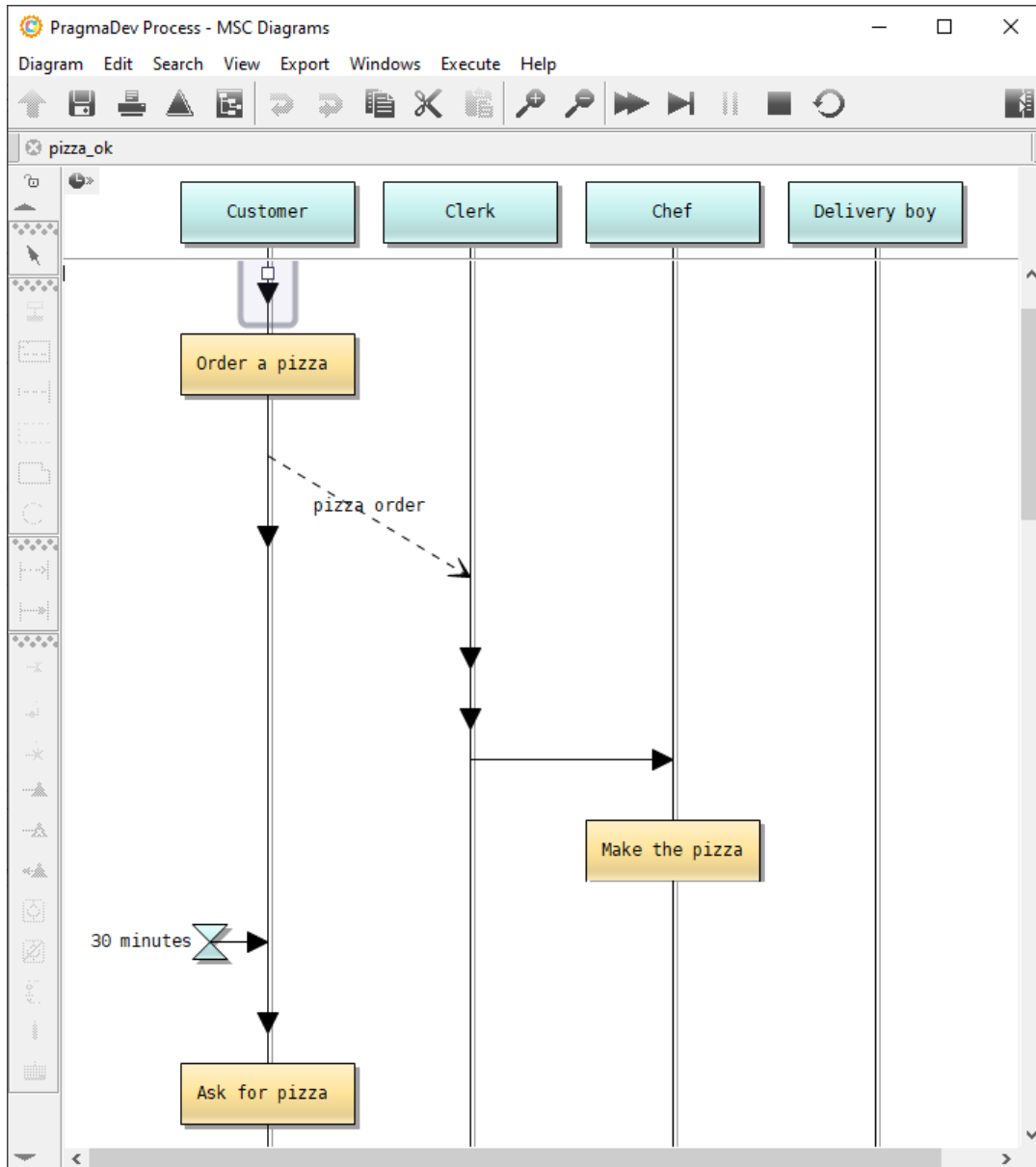
and executed in the BPMN editor:
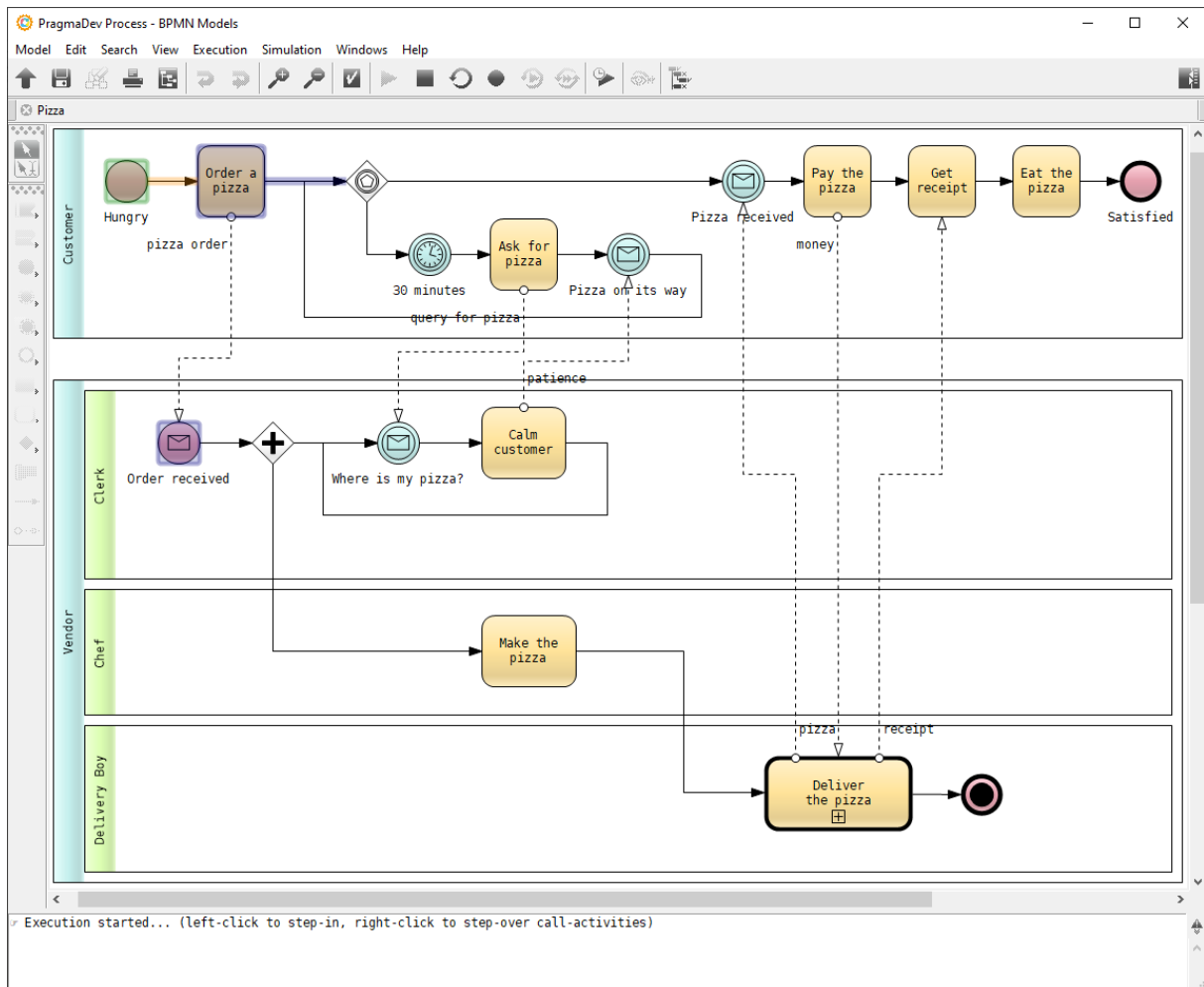
Try the "Step" button a couple of times and observe both (MSC trace and BPMN model) as execution advances in steps. Now click the "Run" button:

Execution will continue until no more steps are left:
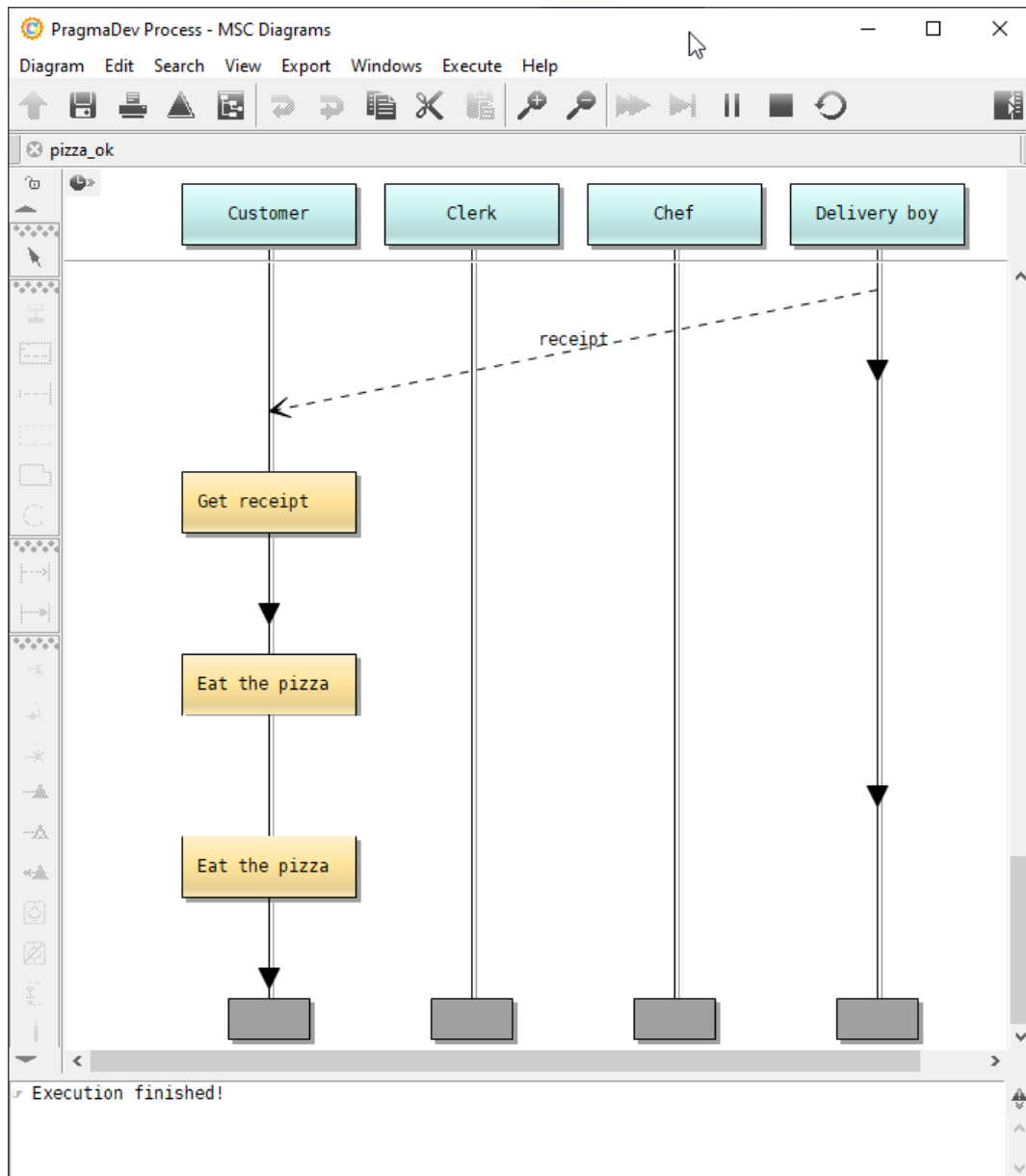
Check also the BPMN editor:
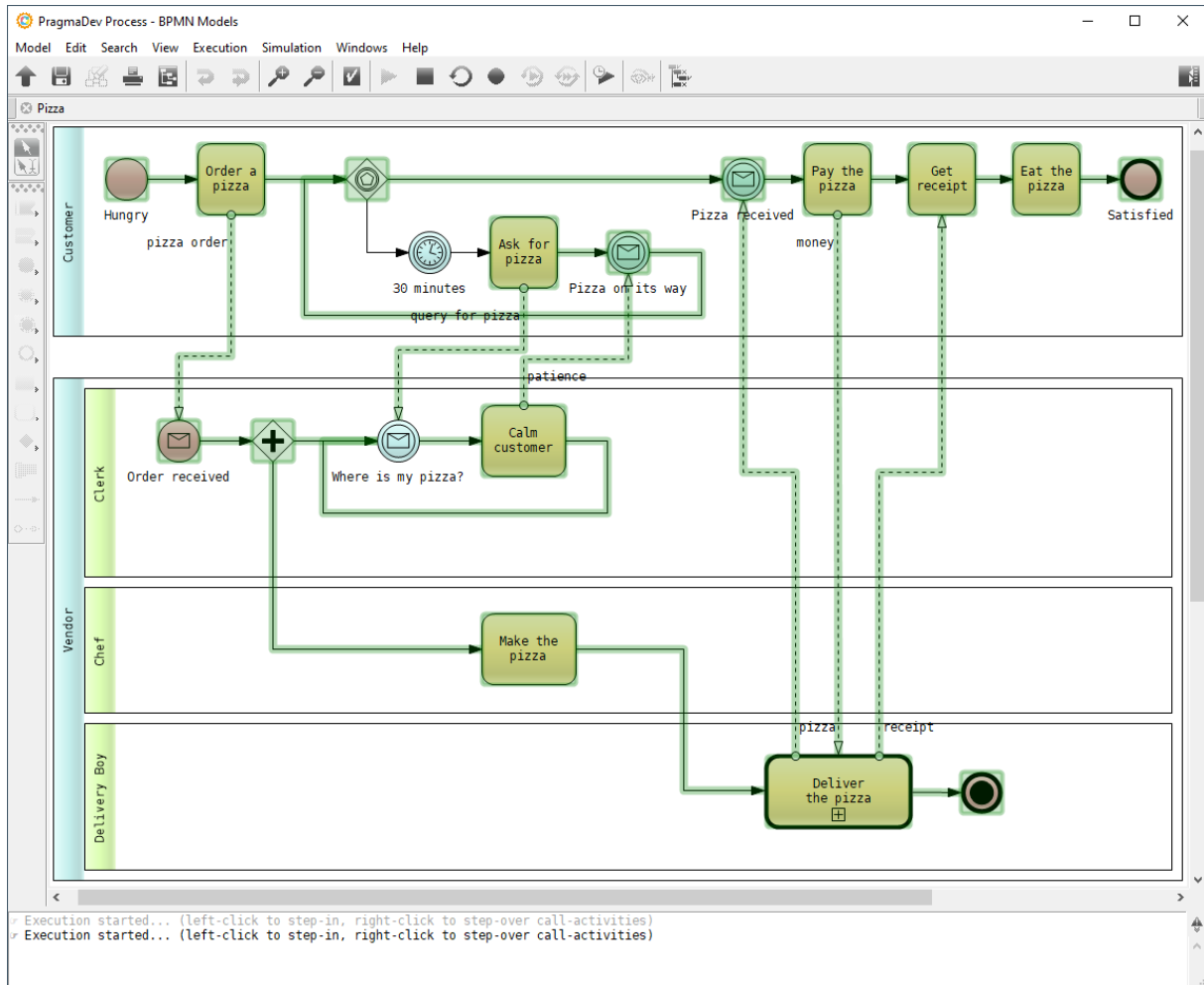
Use the "Stop" button to terminate execution.

With the `pizza_ok` opened in the editor, find the `30 minutes` timer and change it to `10 minutes`:
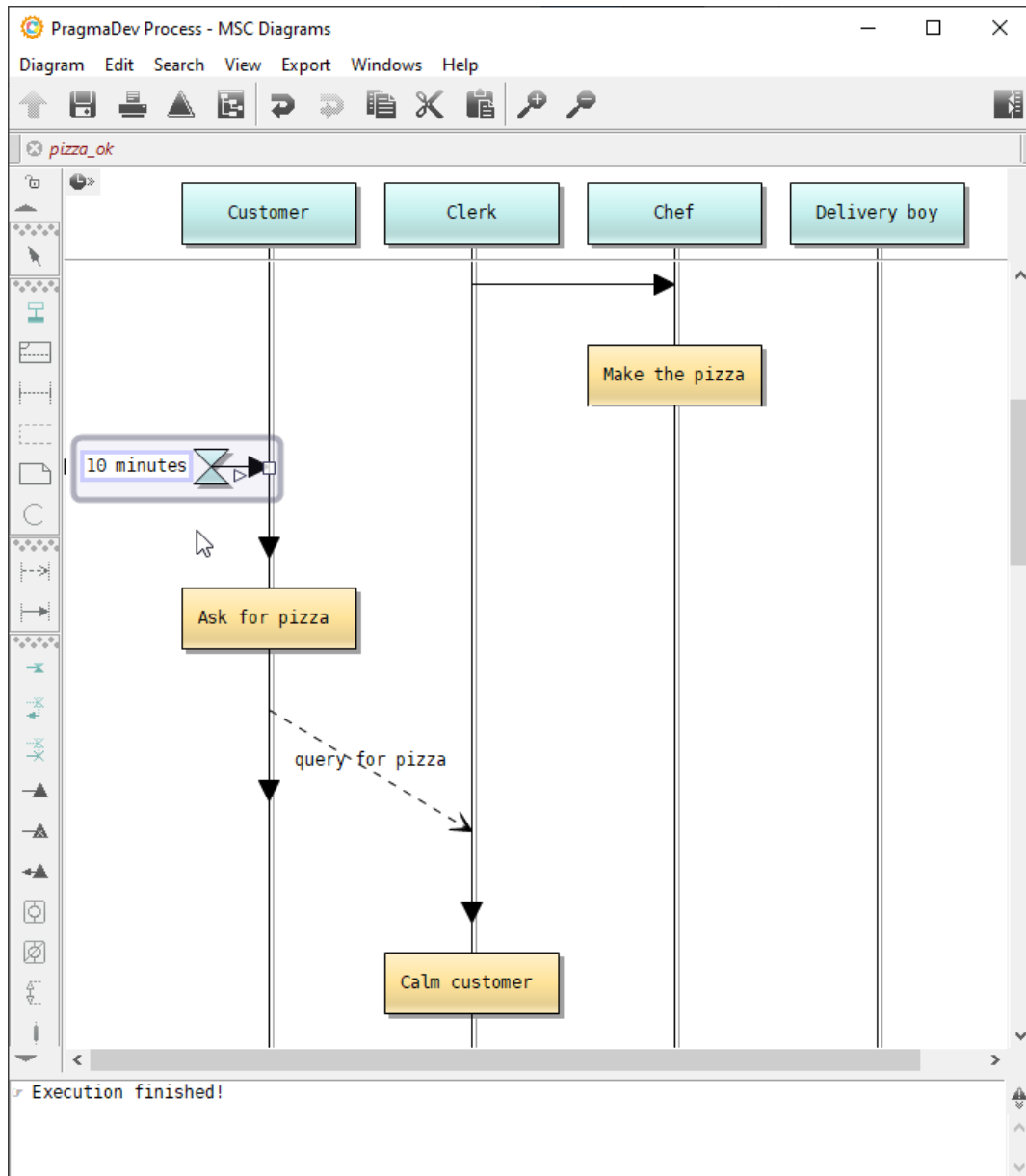
Save the modified MSC trace as `pizza_ko` via the menu "Diagram / Save as..."; it will be added to the project:
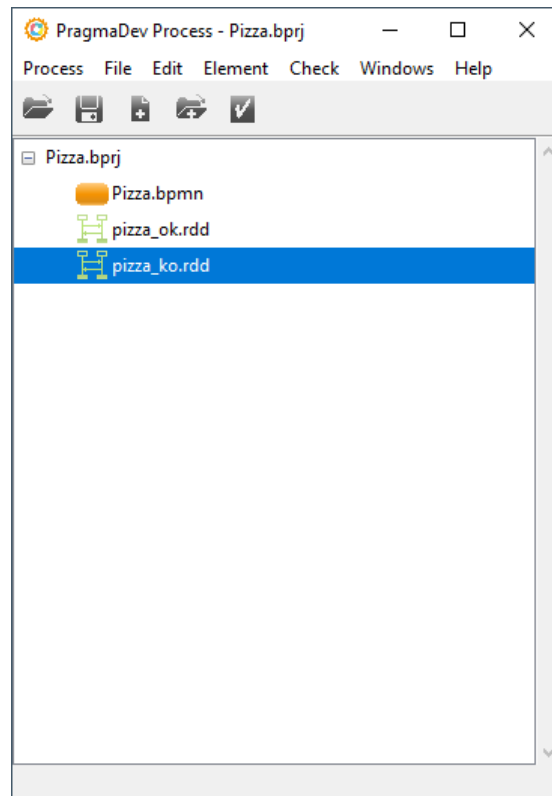
With `Pizza.bpmn` still opened in the editor, click the "Replay" button, and select `pizza_ko` for execution:
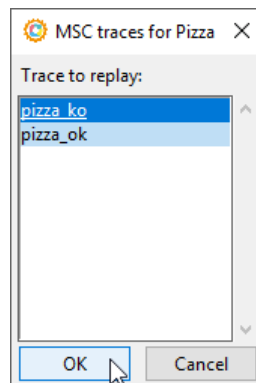
In the MSC editor click the "Run" button. Execution will stop at the timer symbol complaining about text mismatch:
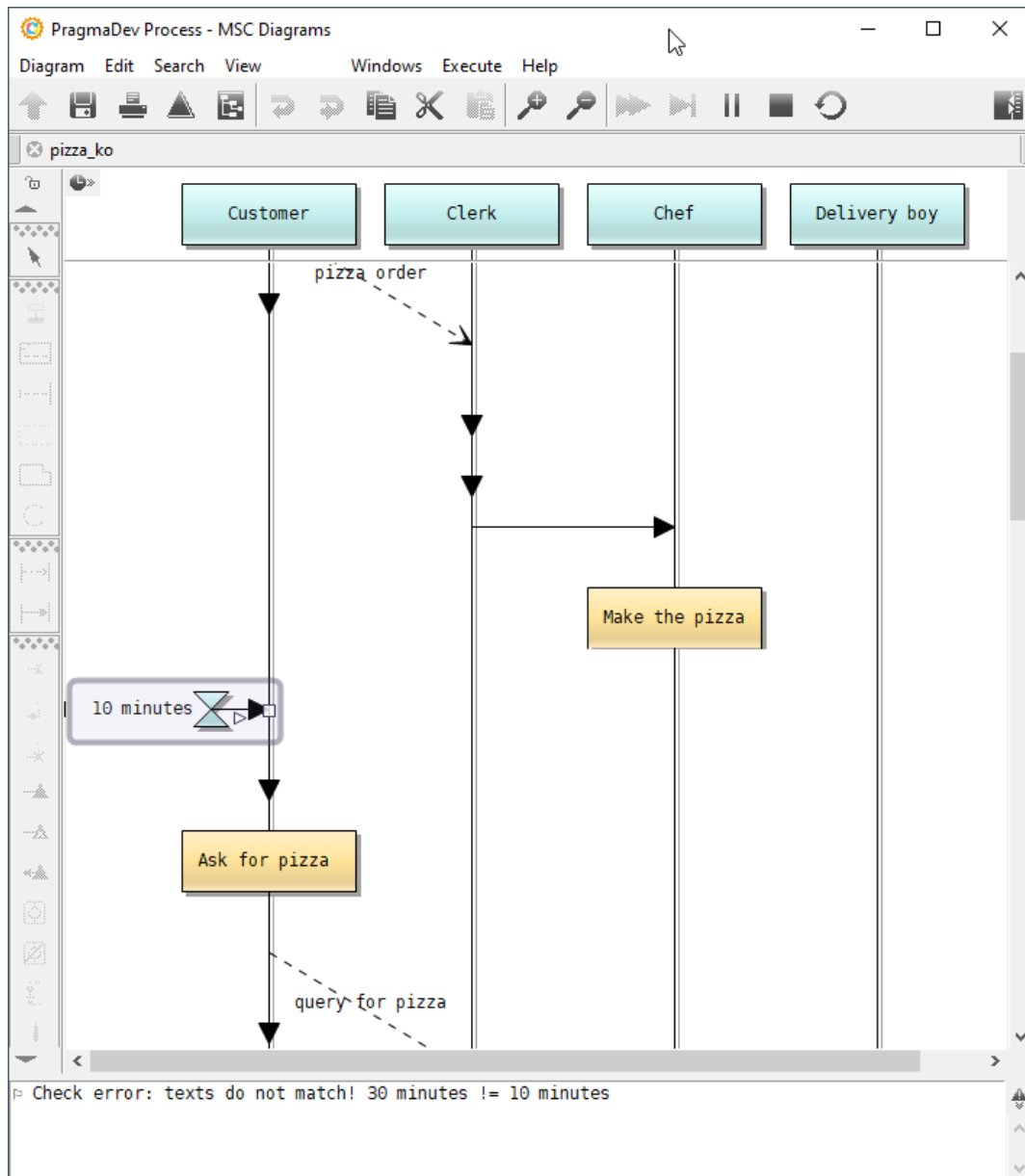
3.3.2 Multi-trace execution

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:
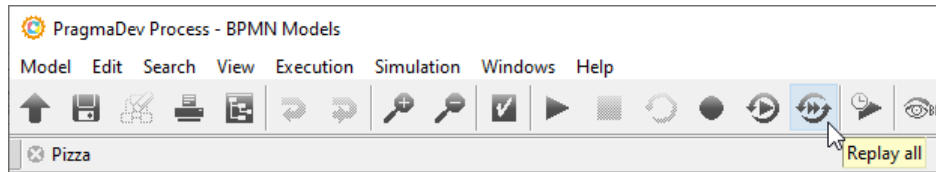
Select `pizza_ok` and `pizza_ko` by clicking on them while holding "Ctrl". With both traces selected click the "Resume" button:
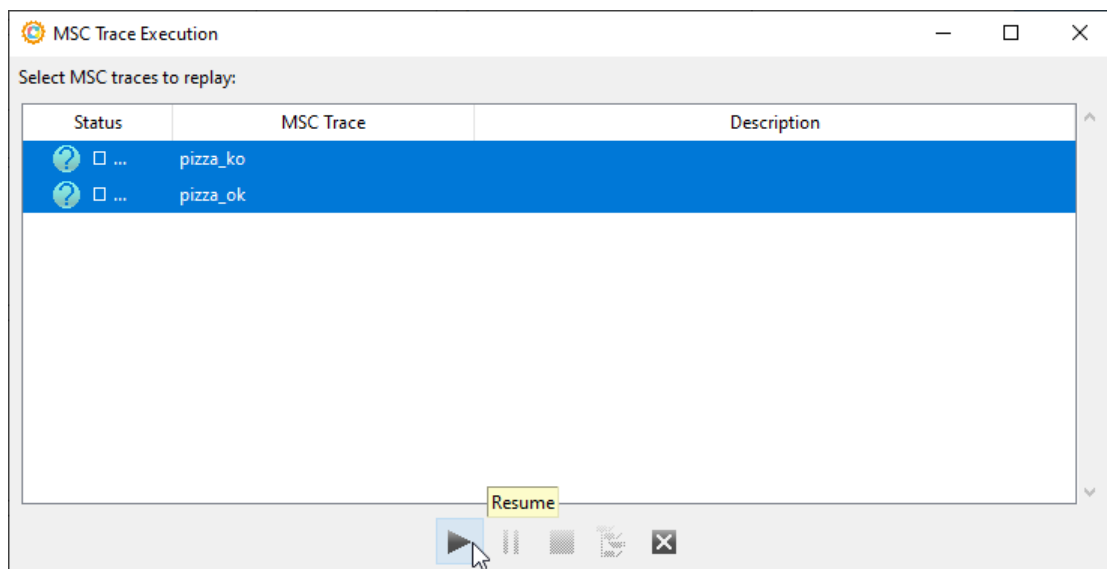
The traces will be executed in the background, and the result of such execution will be shown in the "Status" column:
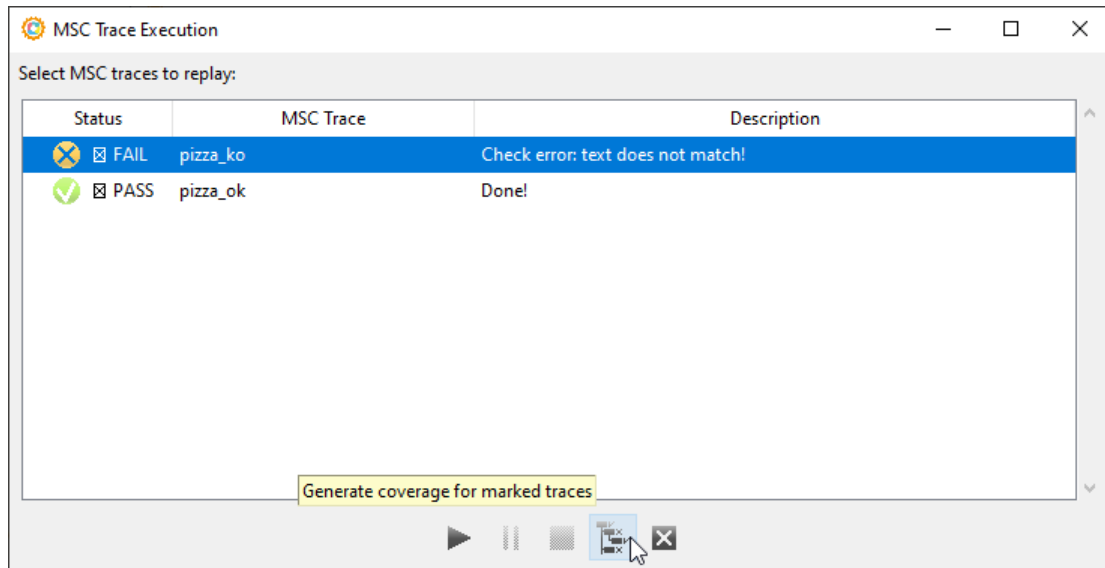
Note that model coverage can be generated via the "Generate coverage for marked traces" button.

Double-clicking a failed trace will:
- Open the trace in the MSC editor and select the concerned element:
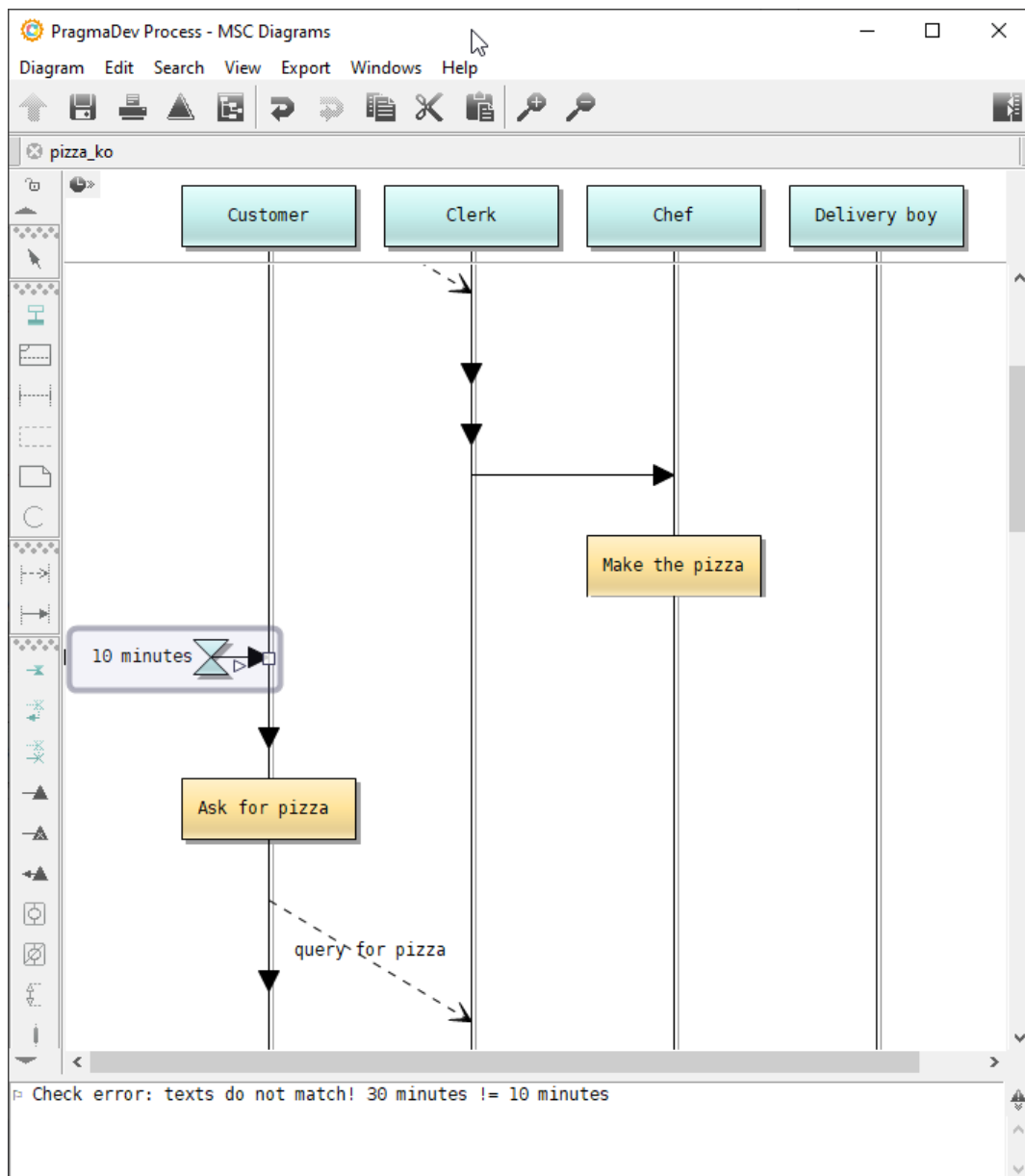
- Select the corresponding element (if possible) in the BPMN editor:
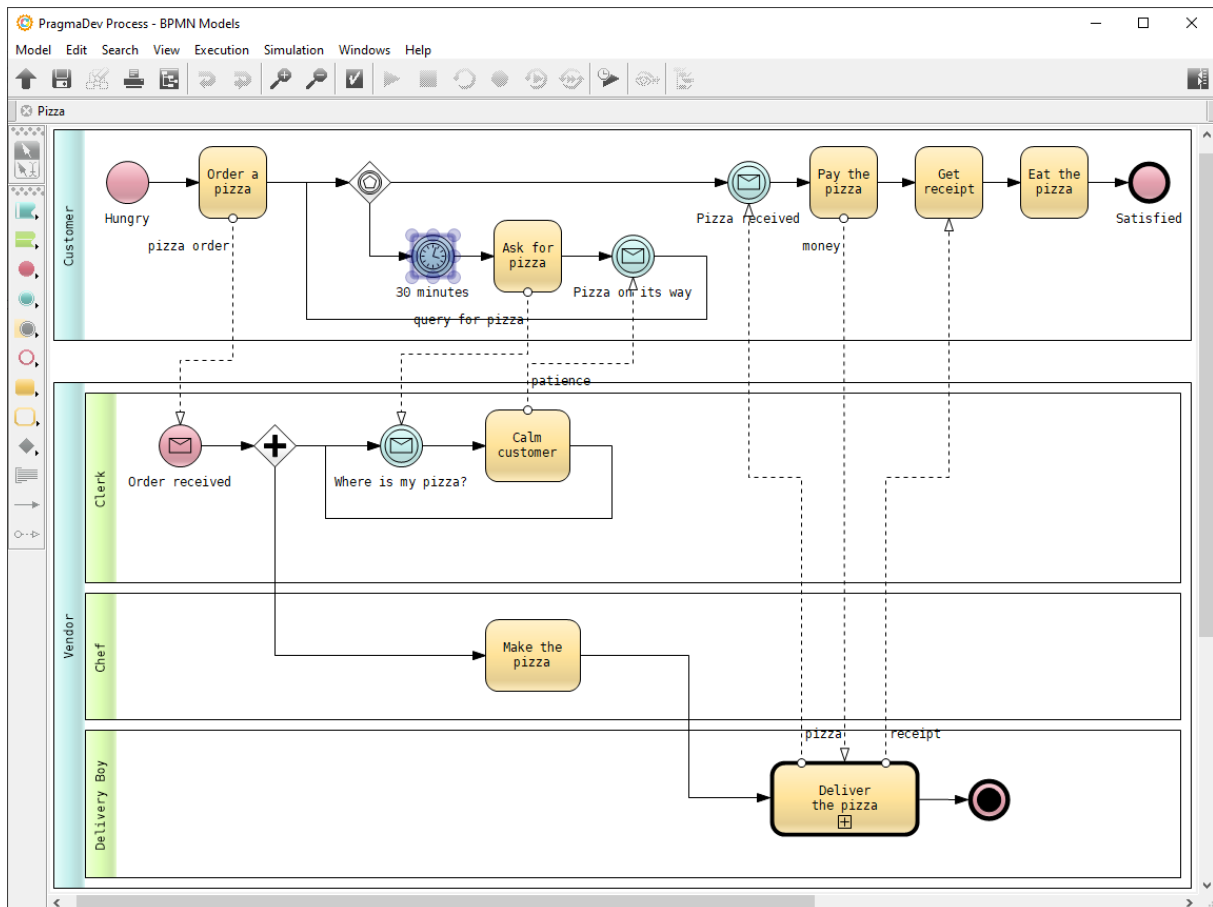
4 Exploration

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.
- *Deadlock*: identification of blocking scenarios.
- *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:
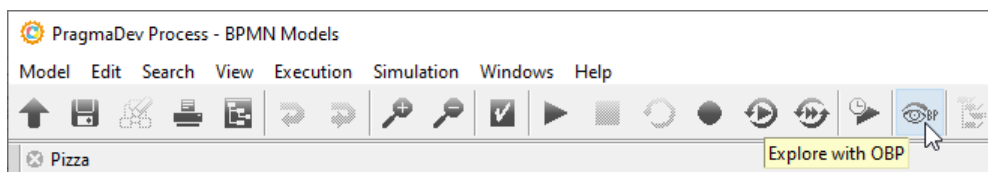
Select "Basic exploration", set "Call-activity definition" to "do not consider", and click the "OK" button:
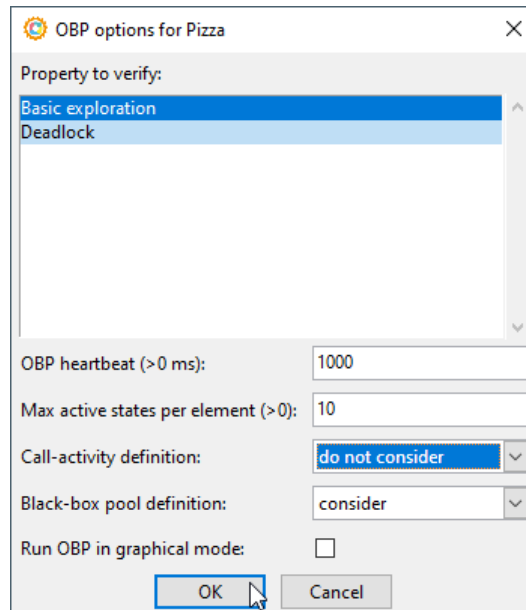
The "do not consider" option will step-over call-activities during exploration.

Observe the increasing number of fired transitions and of configurations[a] while waiting for the exploration to complete. The number of fired transitions shows the progress of the exploration. The number of configurations is the number of different states of the model as a whole figured out so far; it can be compared to the model complexity to find out if that result is too high or normal (see User Manual).
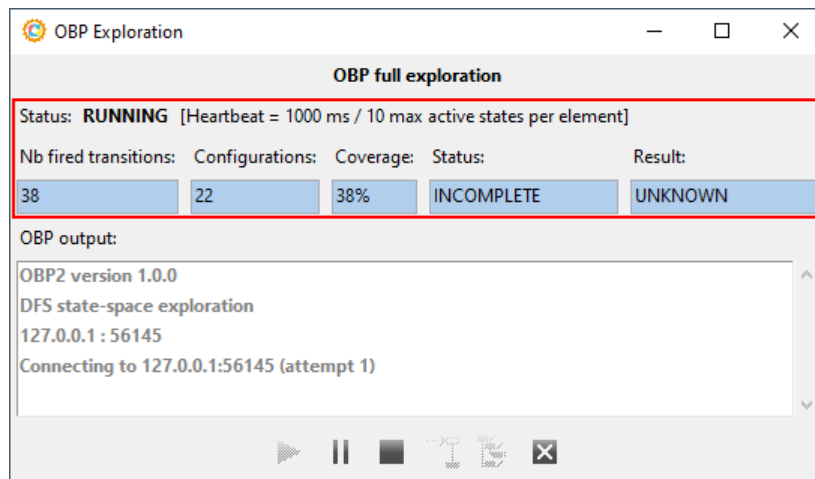
[a]The exploration may be interrupted before completion when using the free version of PragmaDev Process.

4.2 Reachability

A basic exploration will find all reachable paths in the process. If some symbols are not reachable, there is probably a problem in the model. The coverage rate is displayed in the exploration progress dialog; here, a coverage rate of 61% is displayed, so some symbols were unreachable. To identify them, after the exploration is completed, click the "Show uncovered elements" button:
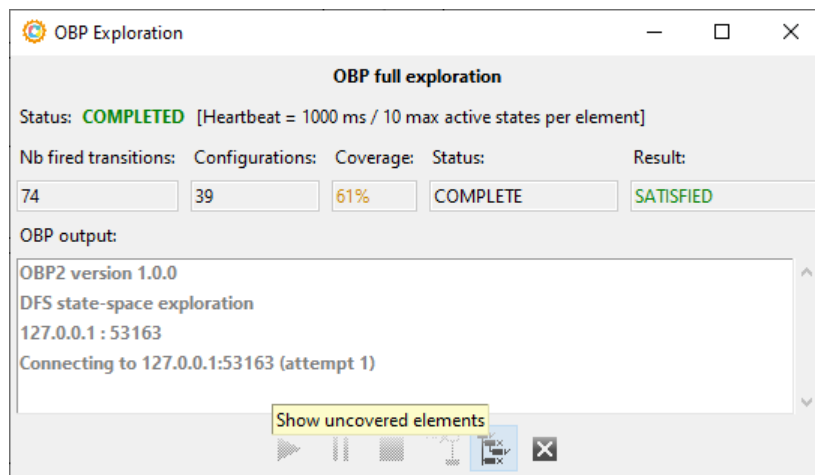
Coverage information will be shown as follows:
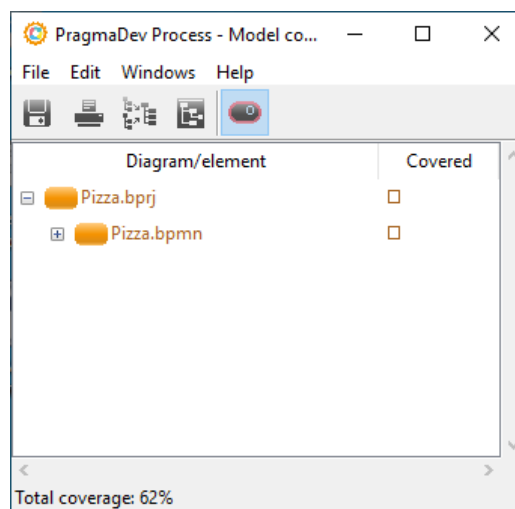
The element tree can be expanded via the entries in the "Edit" menu to show which symbols were uncovered:
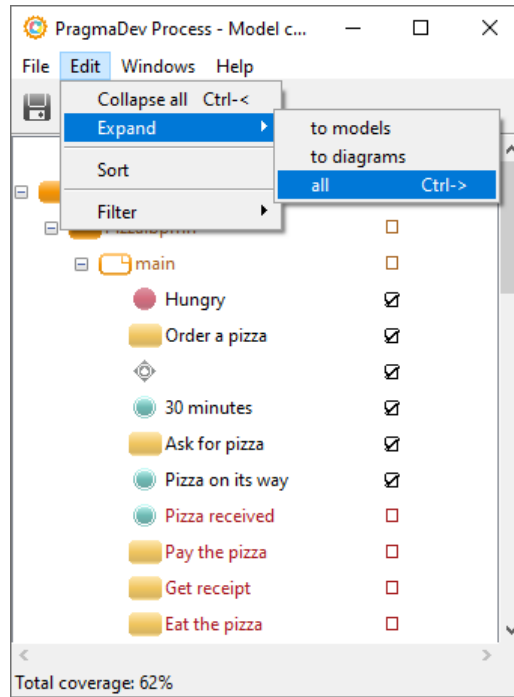
All non covered elements during exploration will also be marked in red in the editor:
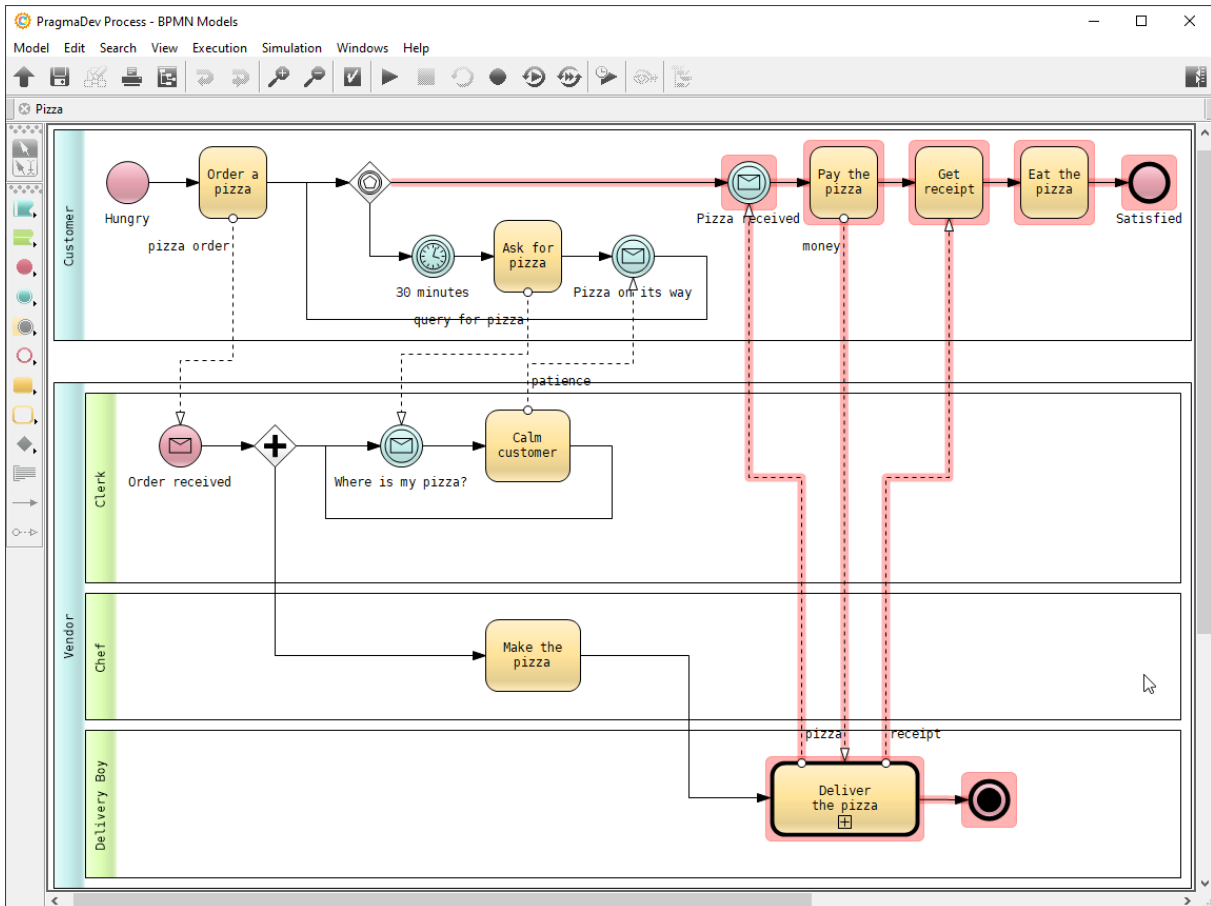
In our example, stepping over the `Deliver the pizza` call-activity resulted in being stuck during exploration. This is because the behavior of a stepped over activity is to wait for all incoming message flows before executing all outgoing message flows. Outgoing sequence flows can be executed only when 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.

Close the coverage information window, and "Quit" OBP.
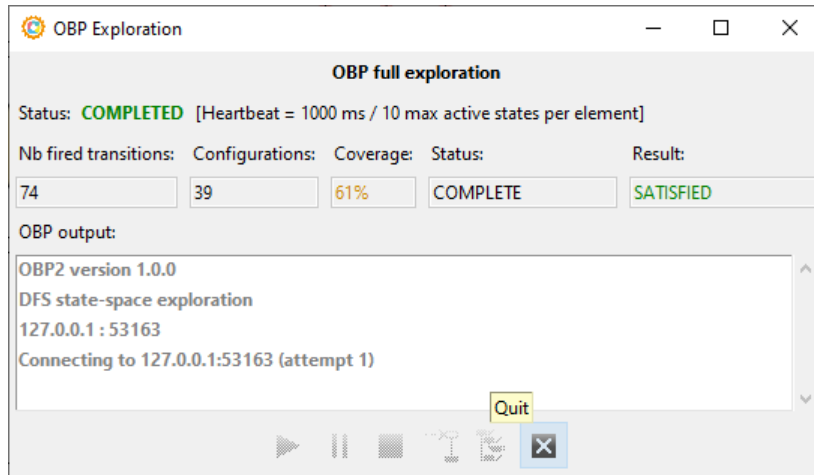
4.3 Deadlock check

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:
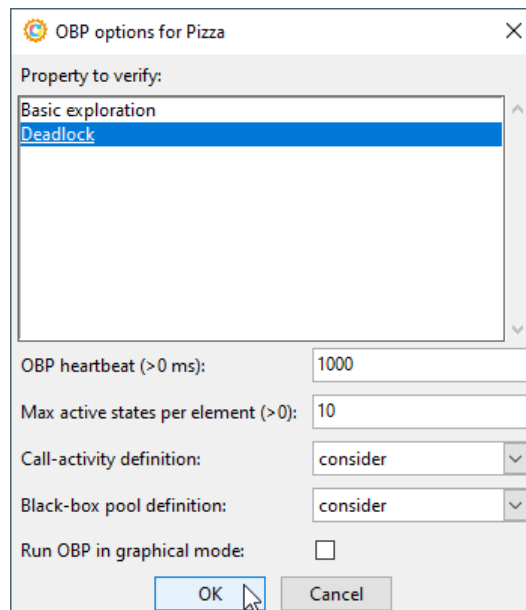
Observe the result returned by OBP saying that the property was satisfied:
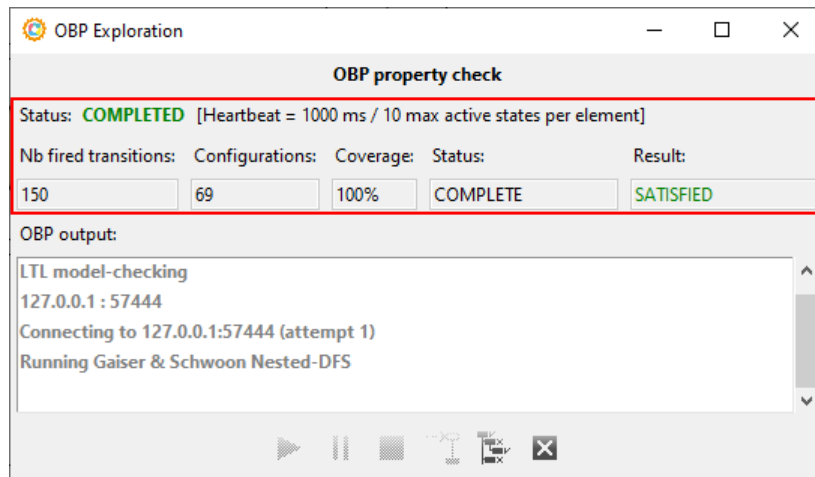
This means that there are no deadlocks in the model.

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:
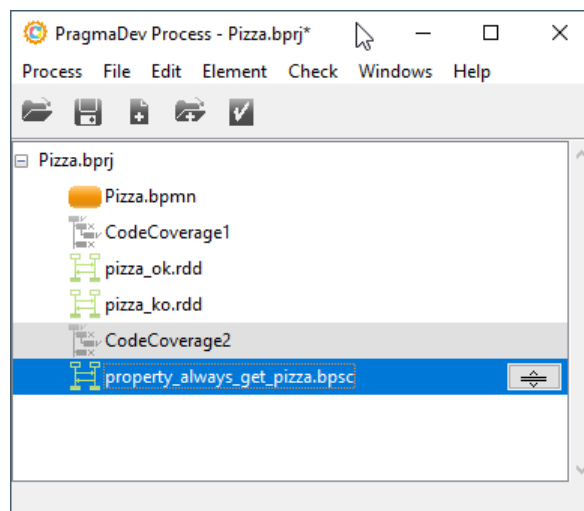
Double click on it to open the MSC/PSC Editor, and draw the following property (more information about the PSC can be found in the User manual):
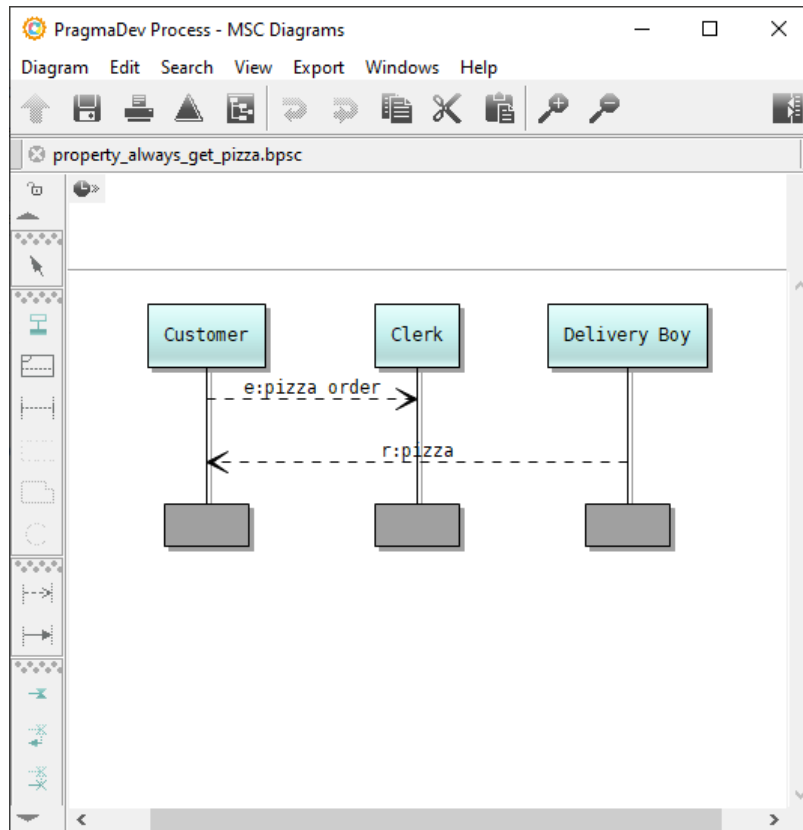
The PSC property says the following:
- If the `Customer` has sent a `pizza order` to the `Clerk`, then
- the `Delivery Boy` *must* hand over the pizza to the `Customer`.

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":
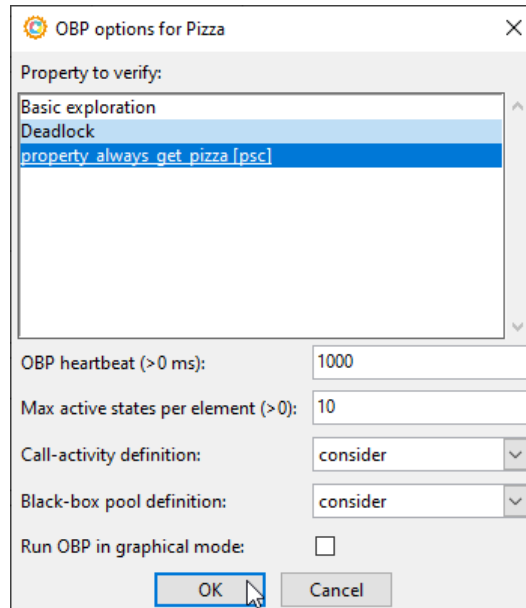
Observe the result returned by OBP saying that the property was violated.[1] Property violation will activate the "Display counter example" button, click it:
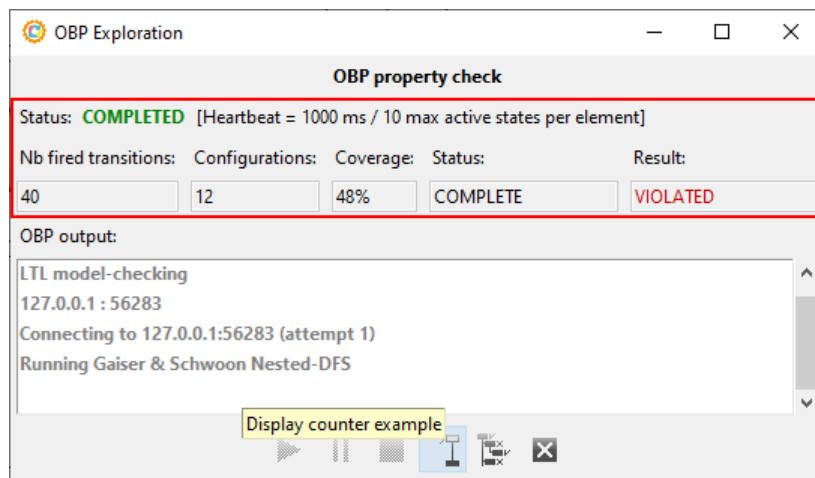
When asked to save the MSC, name it `pizza_not_delivered` and save it. The counter example will be added to the project:
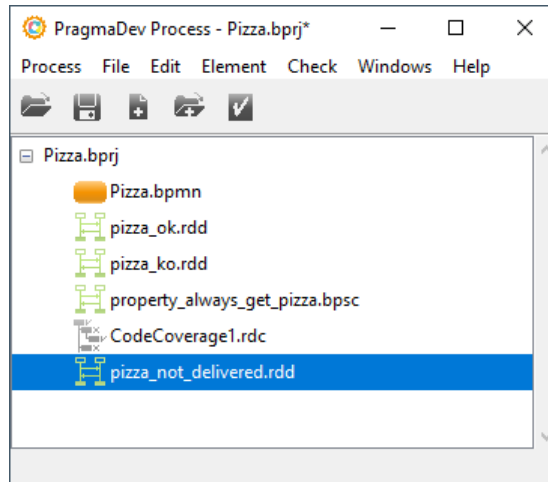
and displayed in the MSC Editor:
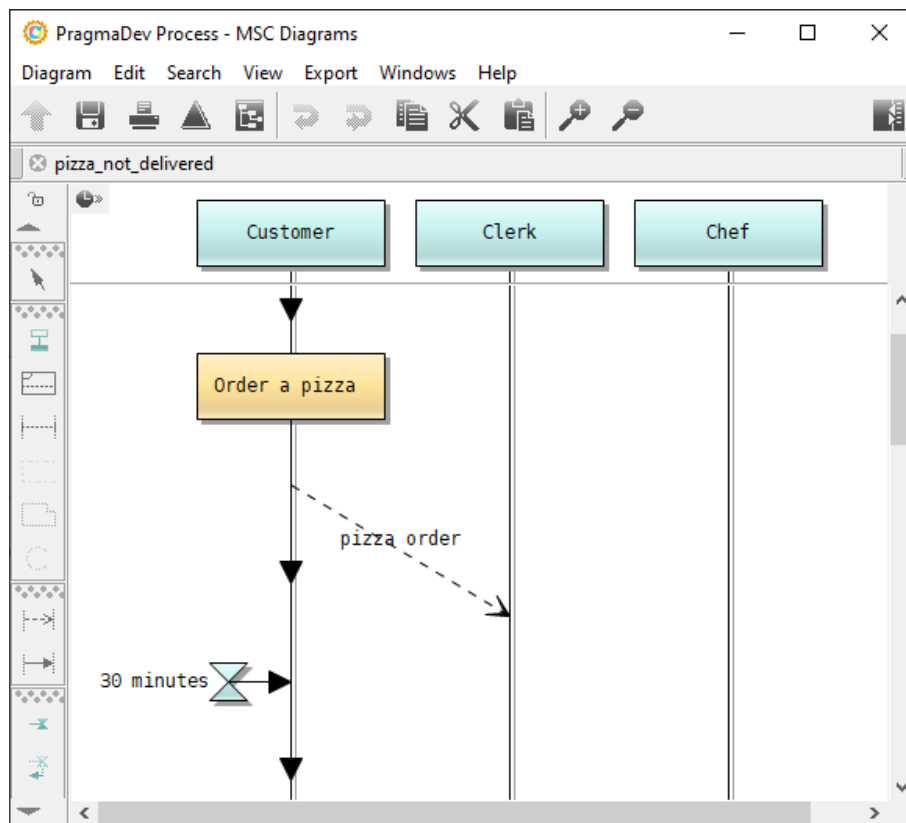
Quit OBP and hit the "Replay" button in the BPMN Editor:
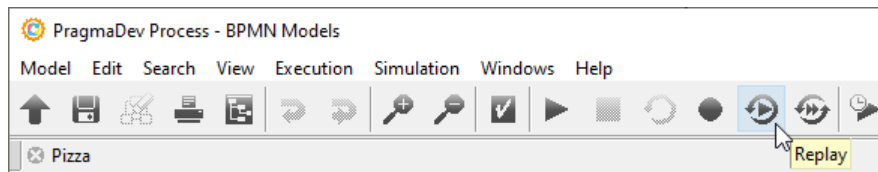
Select `pizza_not_delivered` and hit "OK":
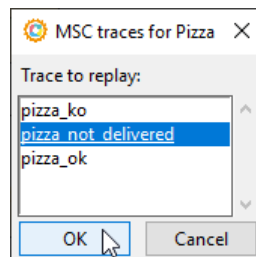
Hit the "Run" button in the MSC Editor:
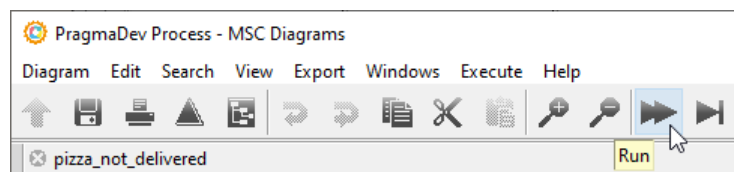
Observe the counter example being replayed in the BPMN editor until the end:
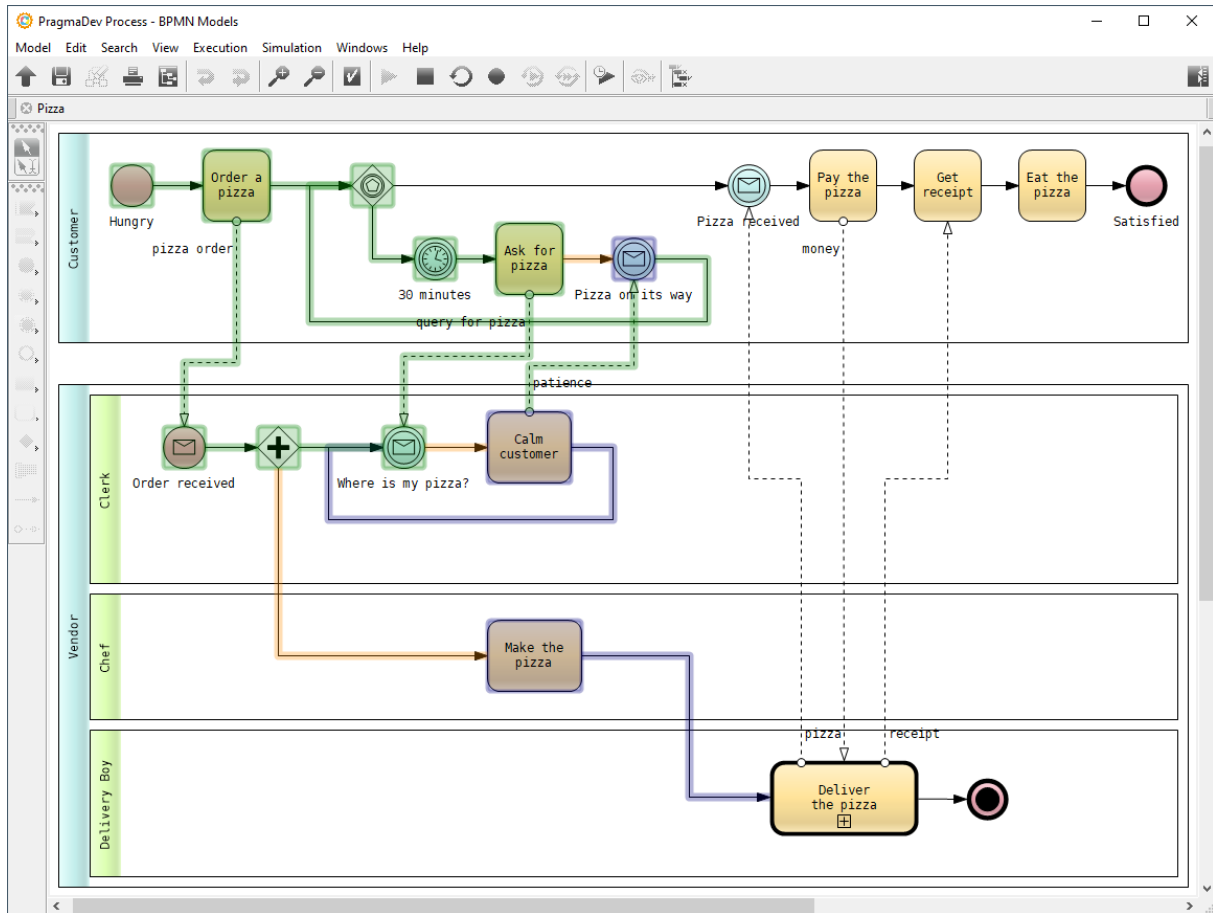
The property was violated because there exists a case (as shown above) where the Cus-tomer is stuck in an infinite loop always asking for his/her pizza, and thus never receiving it.

5 Simulation

The objective of a simulation is to evaluate the time needed to perform a given process, as well as its cost. As an example, simulation can help in choosing the best method of delivery for the pizza, e.g., bike vs car.

5.1 Resources

An activity might require a resource to be achieved. We will use our simple example to demonstrate that with human resources. Add a new BPSim resource file (named `Team`) to the project:

Double click `Team.brsc` in the Project Manager to open it in the editor, and create two human resources: a `Cook` and another guy that has no particular qualification (named `Unqualified`):
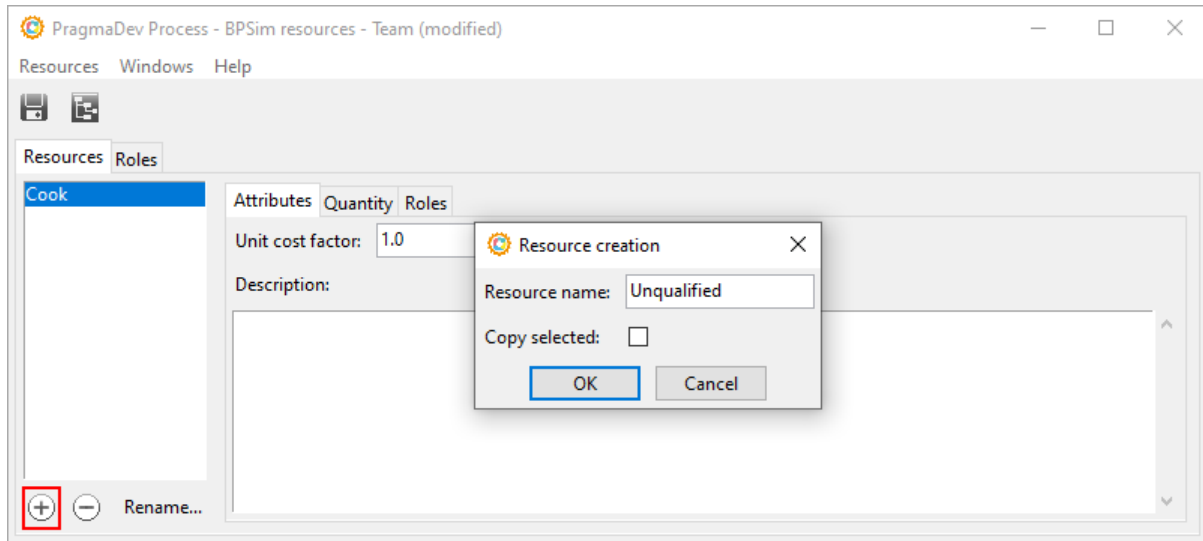
In the "Quantity" tab switch to editing mode via the "Edit" button:
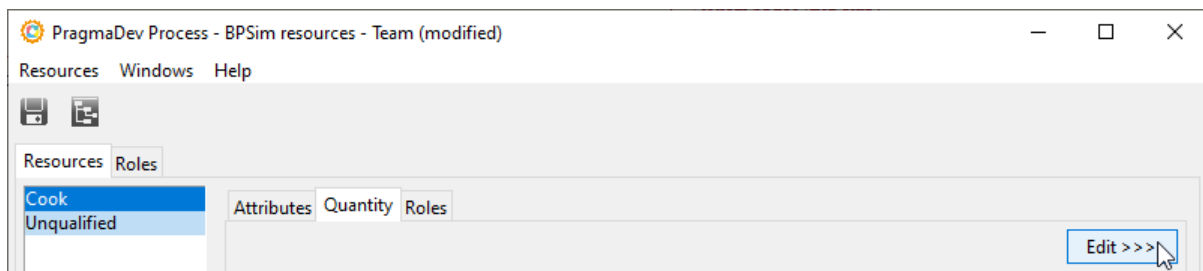
The quantity of the resource can be edited by double-clicking the "Default" entry in the table. To keep it simple, set the quantity to 1 for both resources while keeping all other options unchanged:
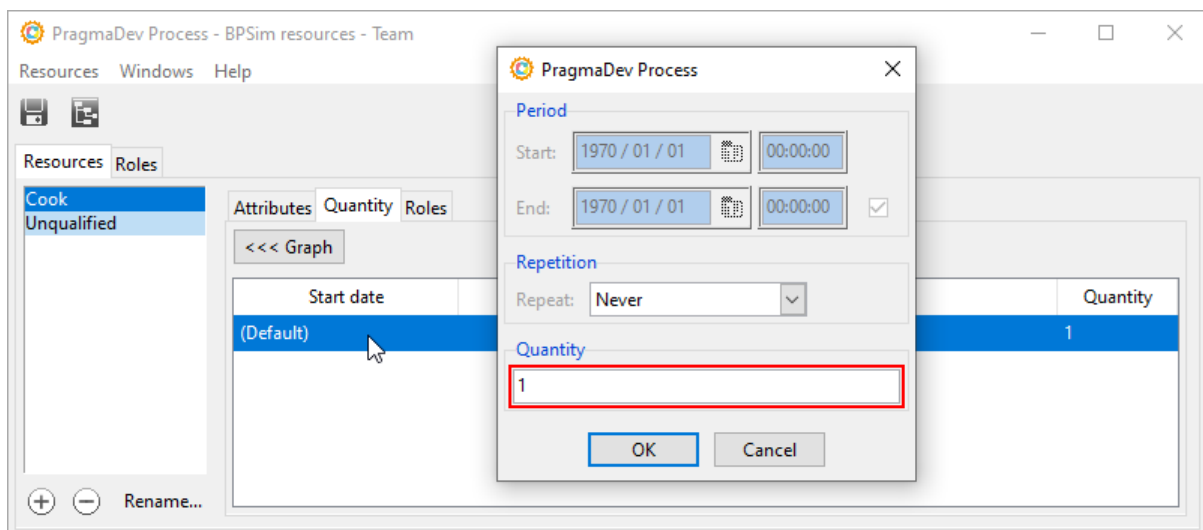
In the "Roles" tab define three roles: Chef, Clerk, and Delivery:
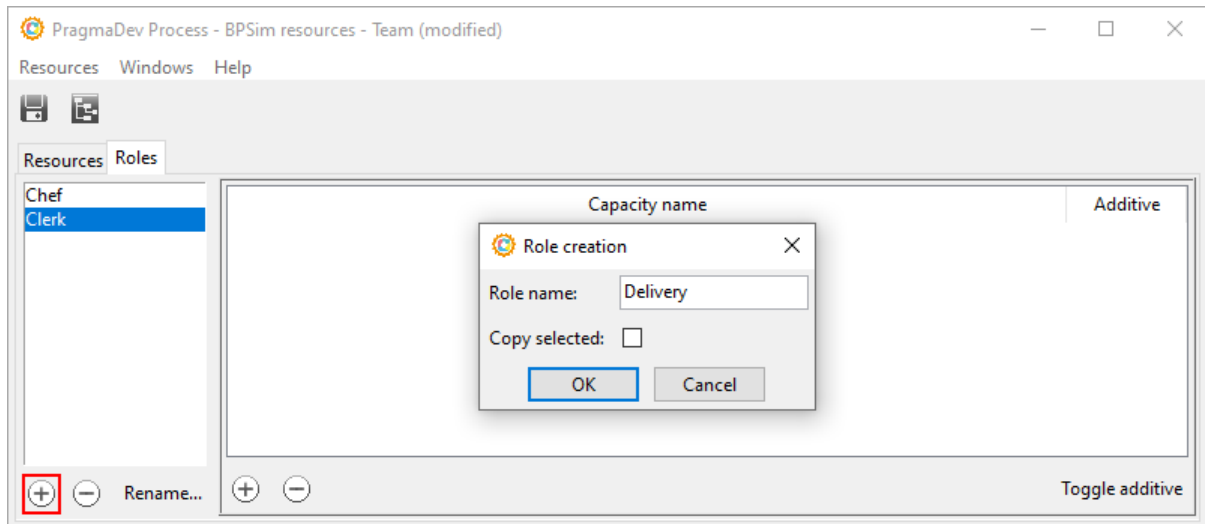
Let's go back to the "Resources" tab and click on the "Role" sub-tab. We will create a situation where there are only two persons working in the pizza shop. The employee with no particular qualification can answer the phone and deliver the pizza but he can not cook. Add Clerk and Delivery to the roles of the resource Unqualified:
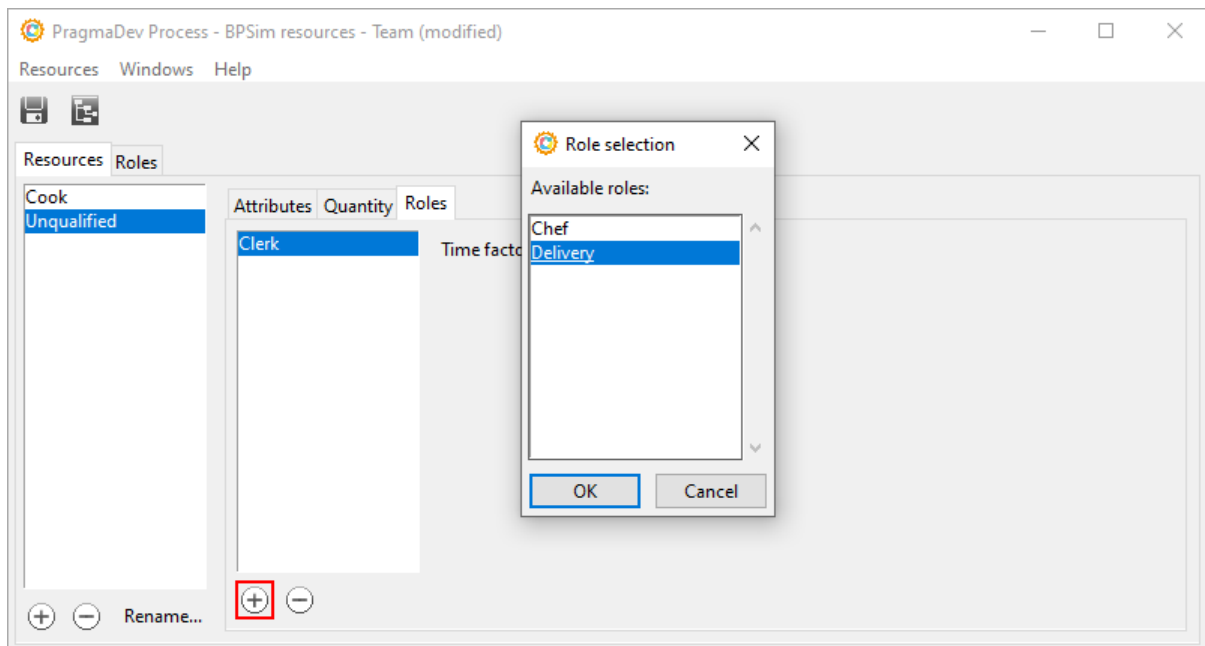
The `Cook` can cook the pizza and answer the phone but he can not deliver the pizza. Add `Chef` and `Clerk` to the roles of the resource `Cook`:
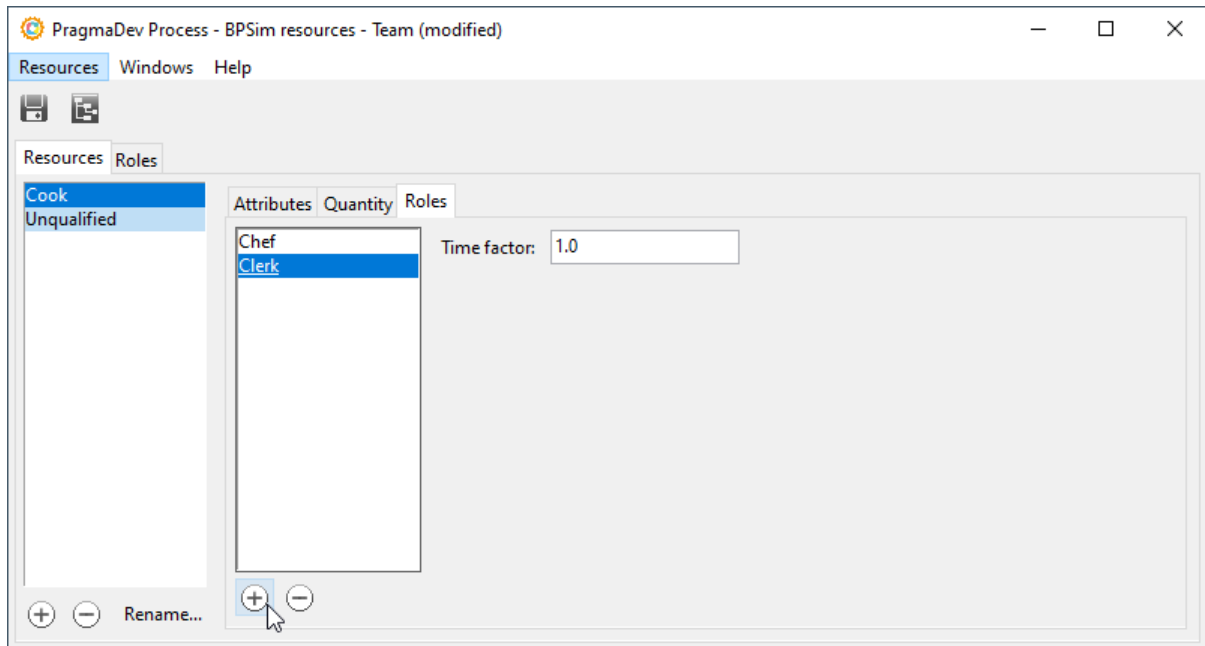
We will use these resources later in the simulation parameters.

5.2 Simulation parameters

5.2.1 Bike delivery

5.2.1.1 Scenario parameters

With the `Pizza.bpmn` opened in the BPMN editor, select "Simulation scenarios..." in the "Simulation" menu:
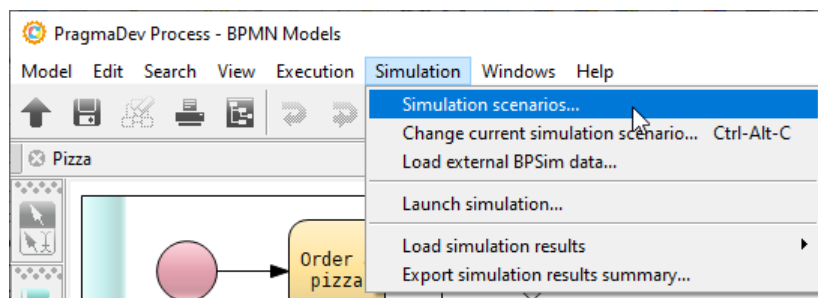
Add a new simulation scenario via the "New / copy scenario" button:
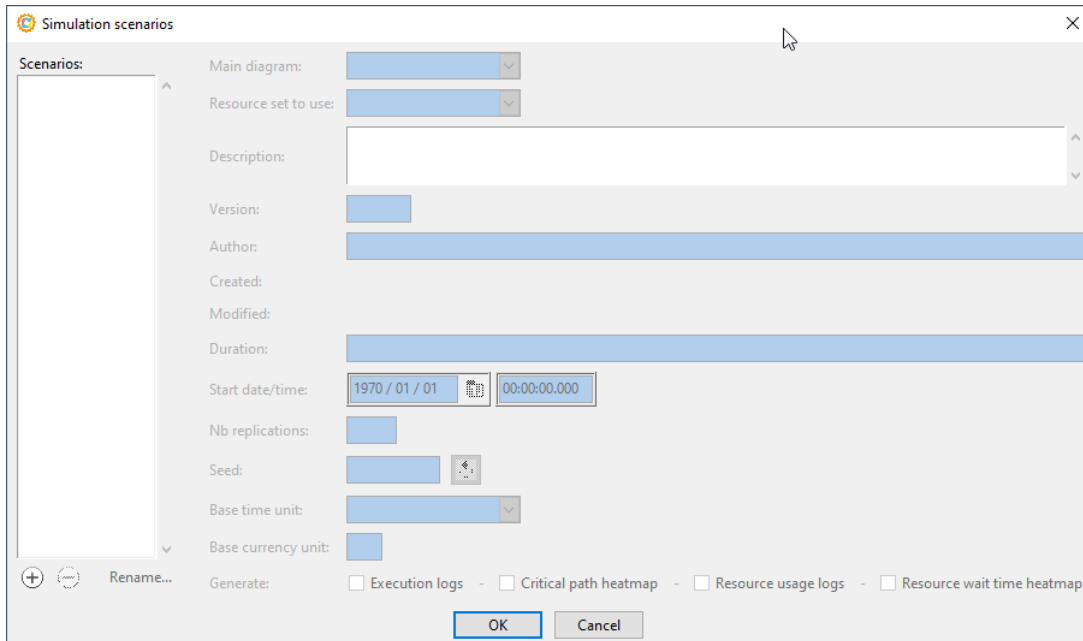
Name it bike_delivery and hit "Create":
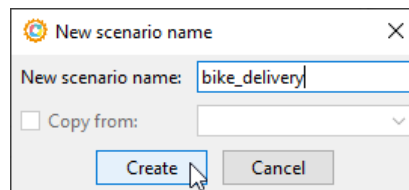
Fill out the scenario attributes as follows and hit "OK":
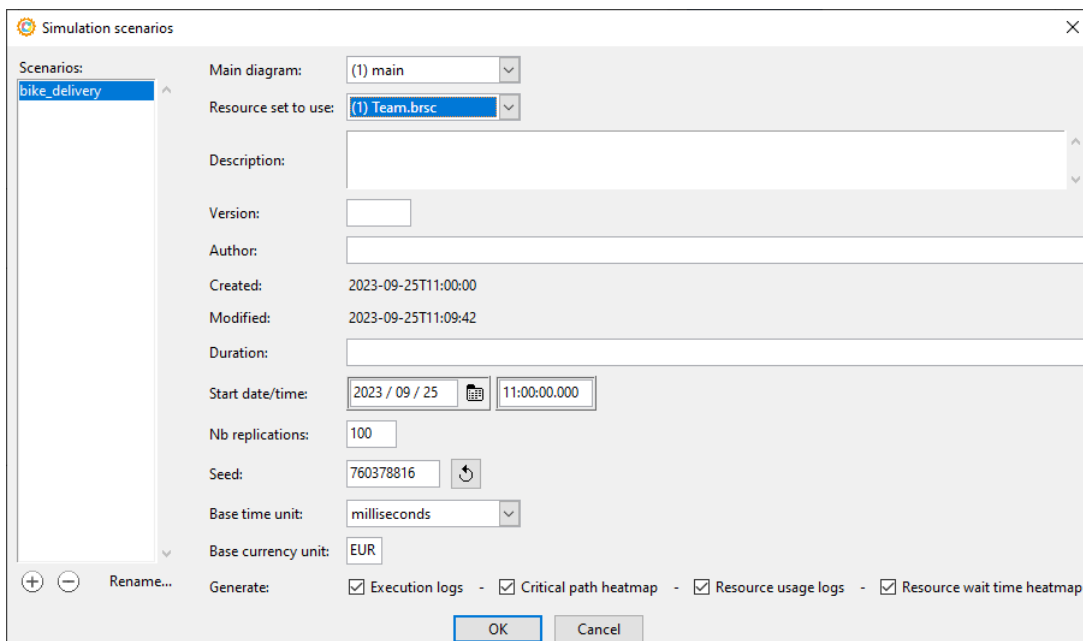
The scenario is set to start in diagram `main` and to use the resources in `Team`; its number of replications (i.e. runs) is set to 100, the time will be measured in minutes, and cost will be measured in EUR.

5.2.1.2 Element parameters

Time, cost, resources, and control parameters

Simulation properties for every BPMN element in the model can be accessed via the contextual menu by selecting "Simulation properties...":

Notice that the current scenario is the `bike_delivery` we created earlier. Start by configuring the parameters of the `Hungry` start event. Select "Control parameters: trigger" and fill the parameters as follows:

The values mean that every simulation run (i.e. replication) will handle 10 pizza orders with a delay of 10 minutes from one order to the other. Let's continue with the simulation parameters for the task `Order a pizza`:

This task is set to take from 1 to 5 minutes with a mean of 3 minutes (following a triangular distribution). Next, set the timer to fire after 30 minutes:

For the vendor activities, in addition to the "Time parameters", the "Resource parameters" have to be set too. To `Make the pizza` a resource with the role `Chef` is needed:

Set the parameters for all the tasks in the `main` diagram with the following values (using a triangular distribution for the time parameters):

Task	Time parameters			Resource parameters	
	mode	min	max	type	quantity
Ask for pizza	3	2	5	None	0
Pay the pizza	1	1	2	None	0
Get receipt	1	1	2	None	0
Eat the pizza	20	15	30	None	0
Calm customer	2	1	5	Role Clerk	1
Make the pizza	10	8	15	Role Chef	1

NB: for activities "Pay the pizza" and "Get receipt", since they have the exact same parameters, you can select them both and enter the parameters for both at the same time.

Switch to the `delivery` diagram and set the values of the task `Get to customer` as follows:
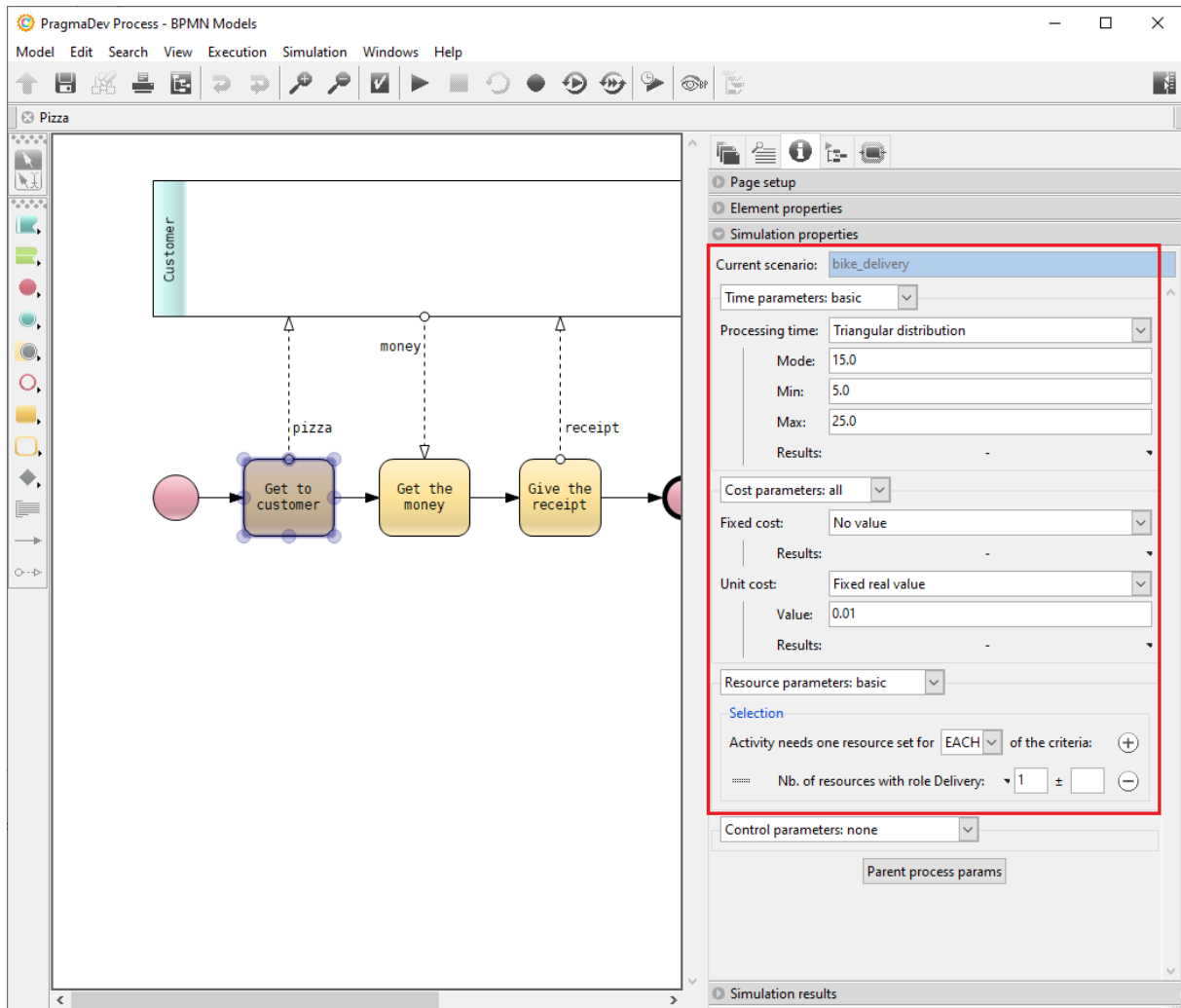
Note that for this task the "Unit cost" is set to 0.01 EUR/minute. This value represents a rough estimation of the cost of using a bike for the delivery. Complete the remaining tasks of the delivery diagram with the following values:

Task	Time parameters			Resource parameters	
	mode	min	max	type	quantity
Get the money	1	1	2	Role Delivery	1
Give the receipt	1	1	2	Role Delivery	1

As above, the parameters for these two activities can be entered only once if you select both activities.
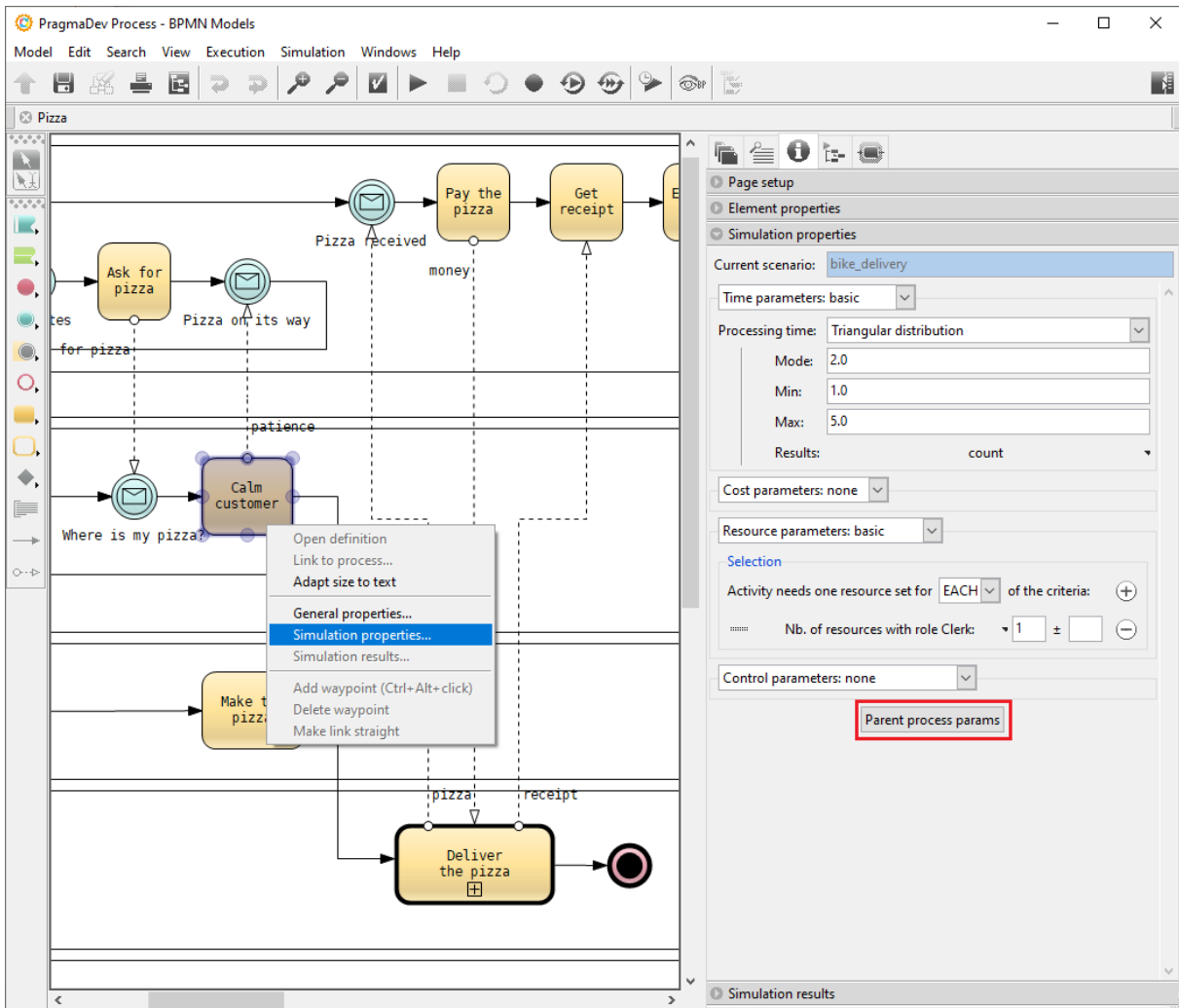
Result requests

In the element parameters, in addition to simulation inputs (e.g., time, cost, control), we can set also the desired outputs (or results) of the simulation. The outputs can be requested on a single BPMN element and/or on a BPMN process. Let's request some results from the simulation. Return to the `main` diagram and go to the simulation properties of the task `Calm customer`. Select "count" for the "Results" as shown:
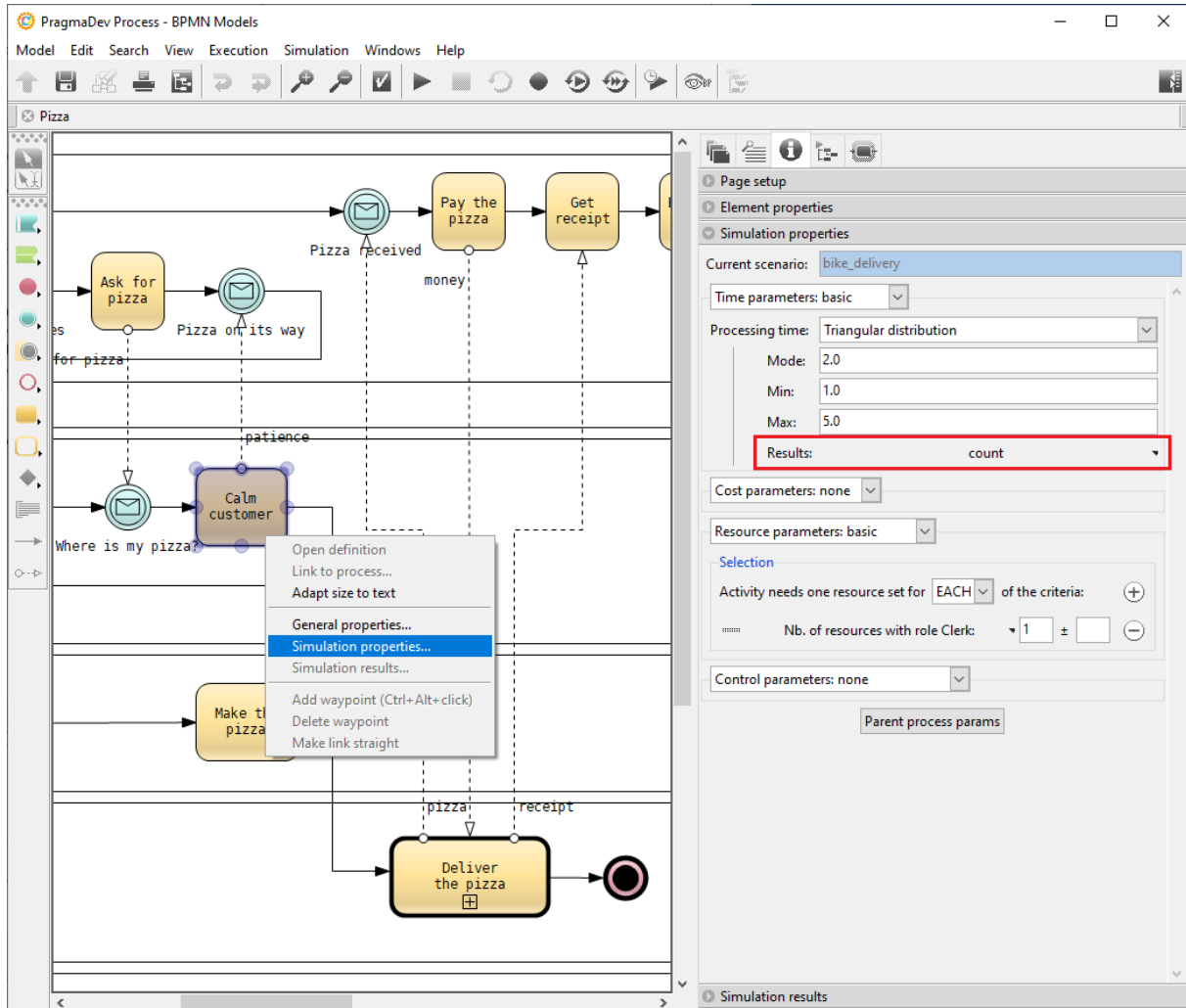

This will request the number of times the `Clerk` had to deal with complaints from `Clients`. The value will give us an indicator for the service quality, i.e., less complaints means less unsatisfied clients, which is better for the business.

Still with the properties of `Calm customer` shown, hit the "Parent process params" button:

Select "min", "max", "mean" for both "Elapsed time" and "Unit cost", and hit "OK":
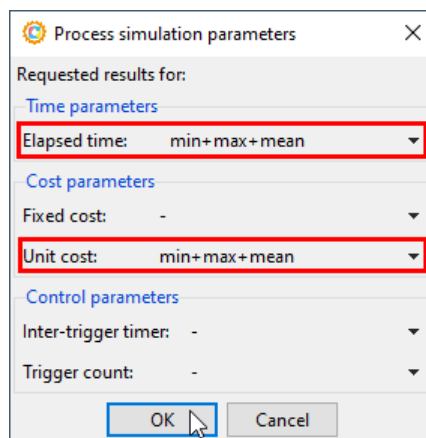

This will request from the possible simulation results the minimum, average, and maximum time and cost needed to perform the `Vendor` process.
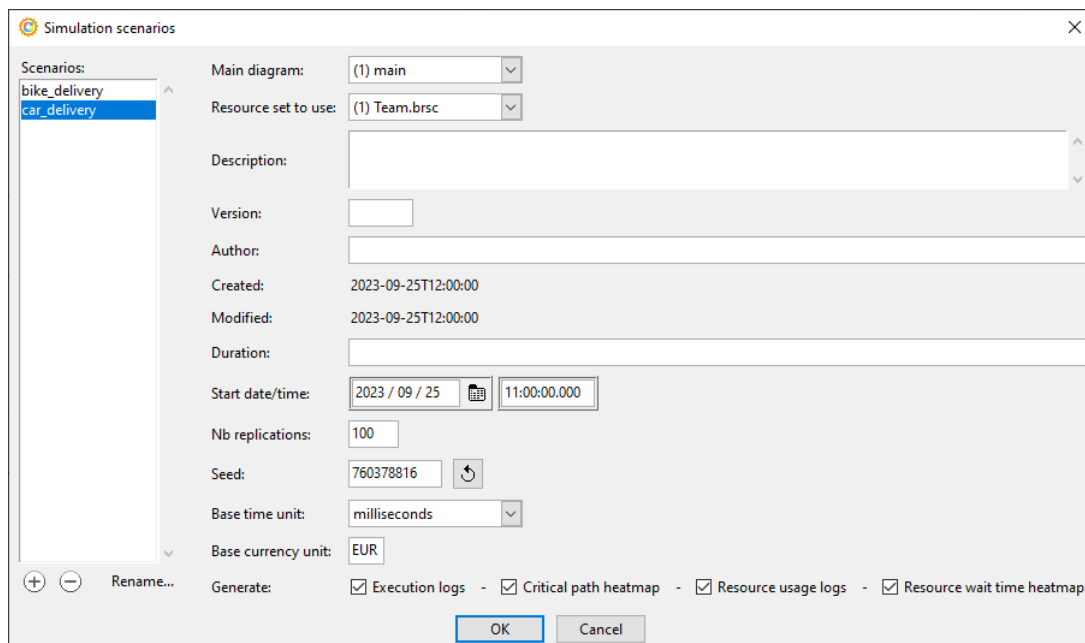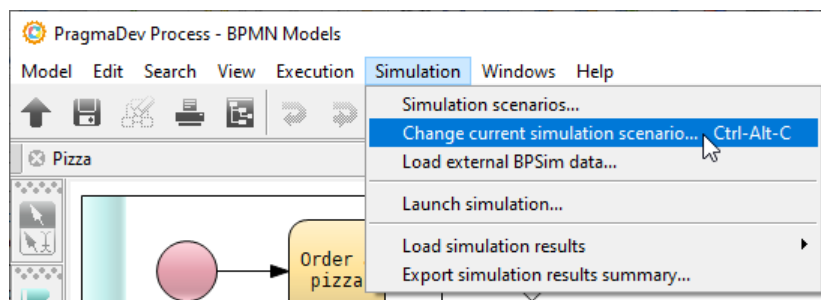
5.2.2 Car delivery

5.2.2.1 Scenario parameters

With the `Pizza.bpmn` opened in the BPMN editor, select "Simulation scenarios..." in the "Simulation" menu. Add a new simulation scenario, name it `car_delivery`, and make sure its initial values are copied from the `bike_delivery` scenario:
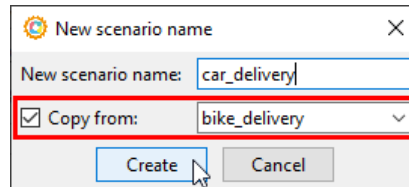
Hit "Create" to add the new scenario, and then "OK" to apply the changes in the scenarios:
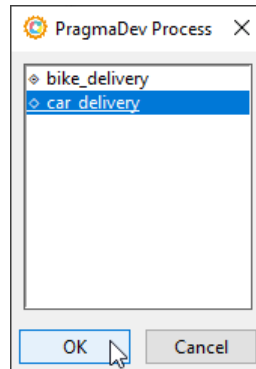
Make sure the current simulation scenario is the new one. For this, in the "Simulation" menu, select "Change current simulation scenario...":

Select `car_delivery` in the list of scenarios and hit "OK":

5.2.2.2 Element parameters

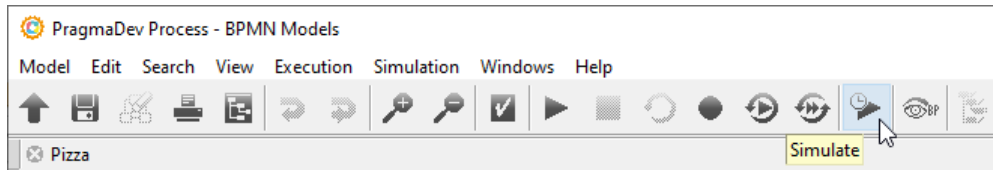In the `delivery` diagram, change the simulation properties of the task `Get to customer` as follows:

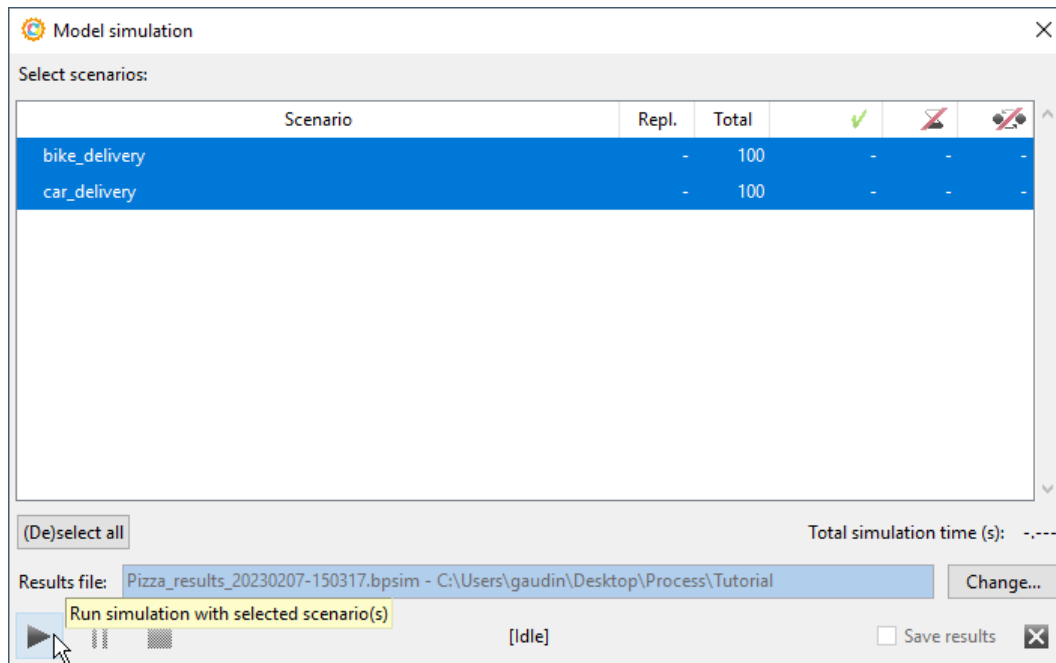Note that the delivery by car takes less time but is more expensive compared to the delivery by bike.

"Save model" to apply all changes.
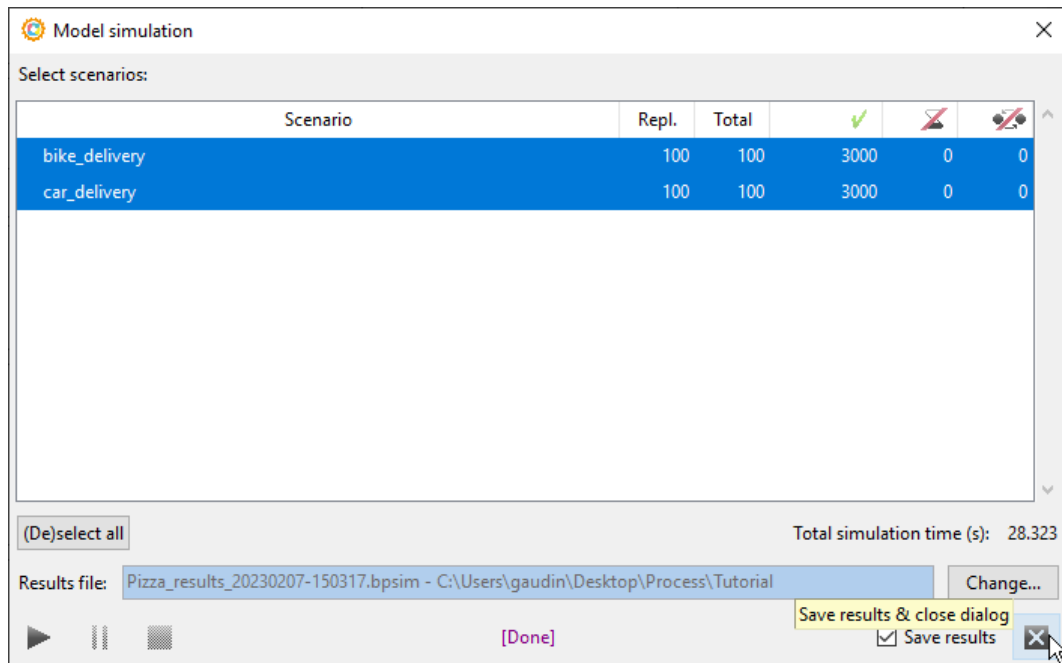
5.3 Running the simulation

With the `Pizza.bpmn` opened in the BPMN editor, and the `main` diagram shown, hit the "Simulate" button:

With both scenarios selected hit the "Run simulation with selected scenario(s)" button:

When done make sure the "Save results" is checked and hit the "Save results & close dialog" button:

A new simulation results file (in BPSim format) will be added into the project under the BPMN file:
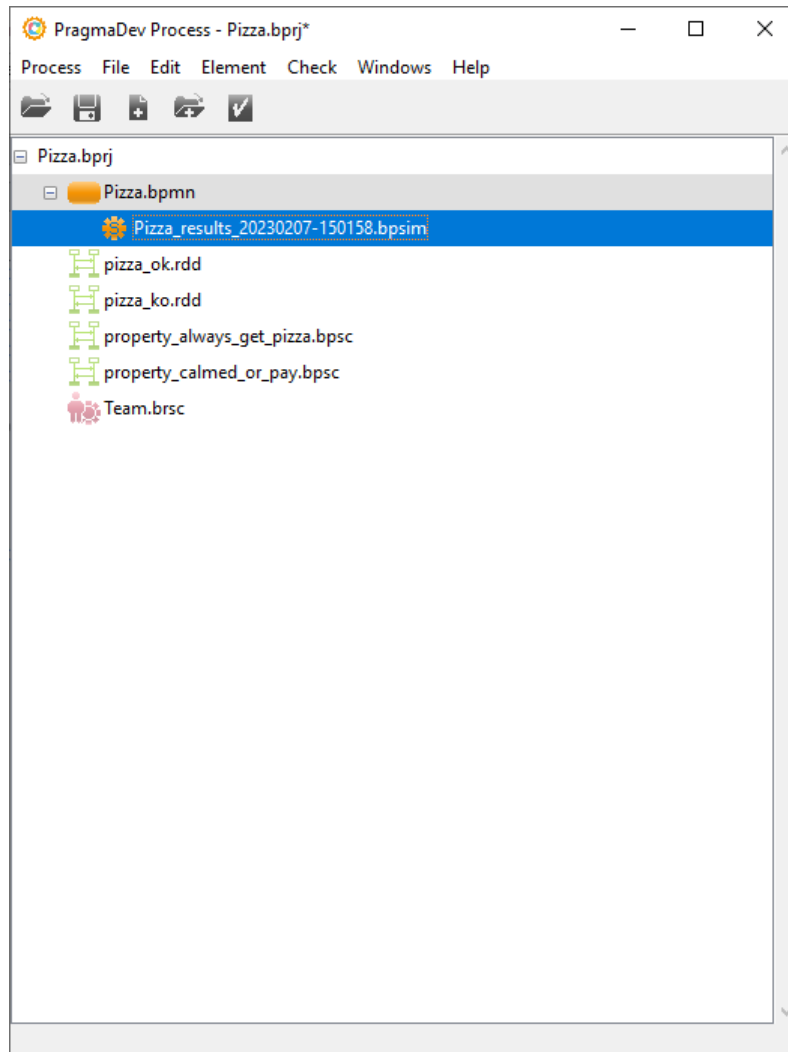
Save the project.

Note that simulation results files are not editable. They are always associated to a BPMN file in the project, and opening a results file will open the corresponding BPMN file in the editor and load the results with it.

5.4 Simulation results

Double-click the simulation results file to load it in the BPMN editor. Right-click the task `Calm customer` and select "Simulation results..." in the contextual menu. Make sure "Show values" is checked to display the exact values[a] in the graph:

---
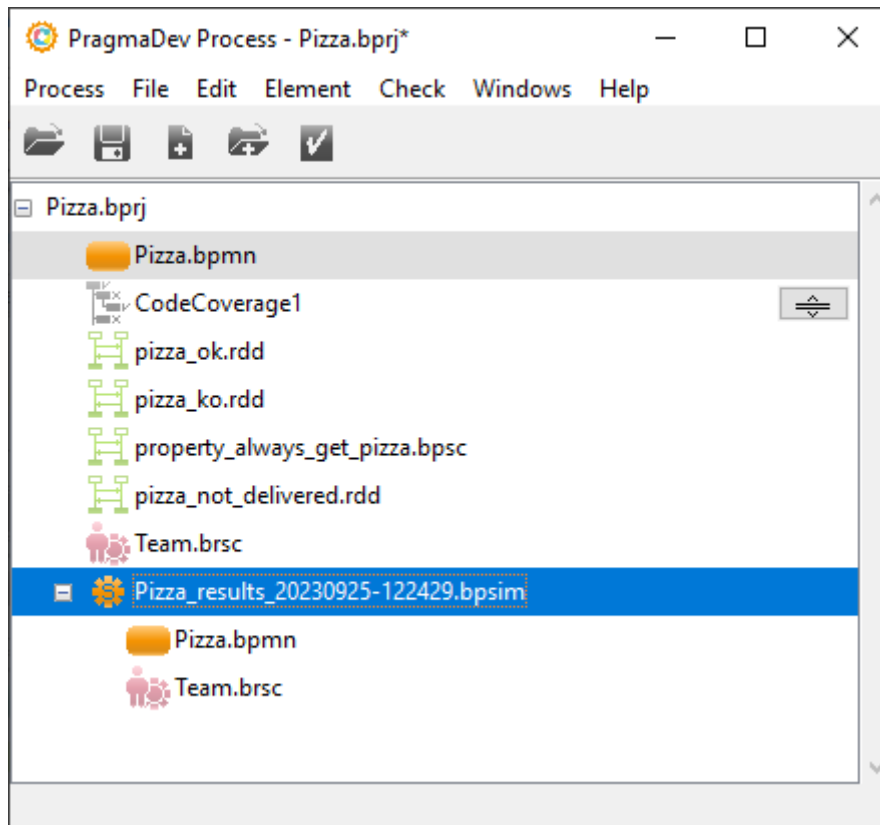
[a]Note that the exact values may differ from the figure.

The bar graph will show the number of times the task has been executed for each scenario. Compared to the delivery by bike, the number of complaints is significantly lower when a car is used.

Still with the results of the task `Calm customer` shown, hit the "Parent process results" button:

Select "Cost: unit" in the "Parameter":

The graph will show the cost for both scenarios, i.e., bike and car delivery. Observe the cost of bike delivery being significantly lower compared to the delivery by car.
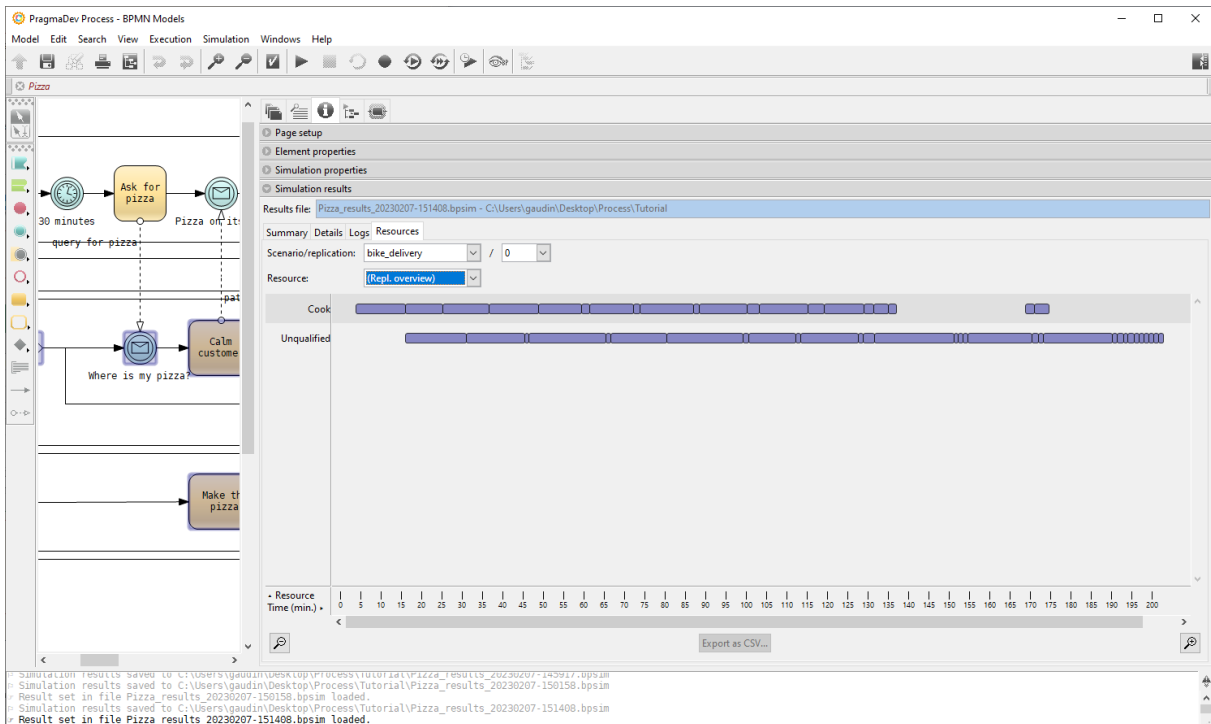
Change the selection in the "Parameter" to "Time: elapsed":

The graph will show the times of delivery for both scenarios.

Select the "Resources" tab to see how occcupied were the resources during the simulation.

One can see that at some point the `Cook` was unoccupied, because there were no more pizzas to make.

A summary of simulation results can be exported in CSV format. For that, in the "Simulation" menu select "Export simulation results summary...":
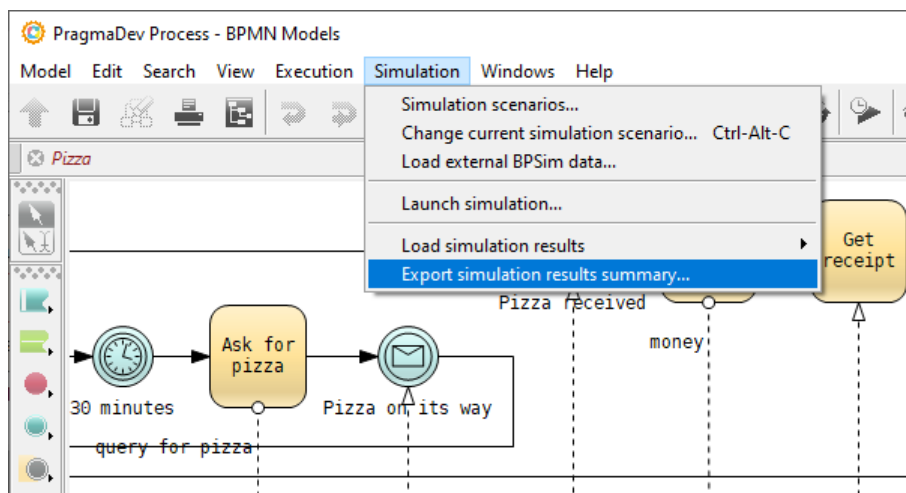
The summary is presented as follows:

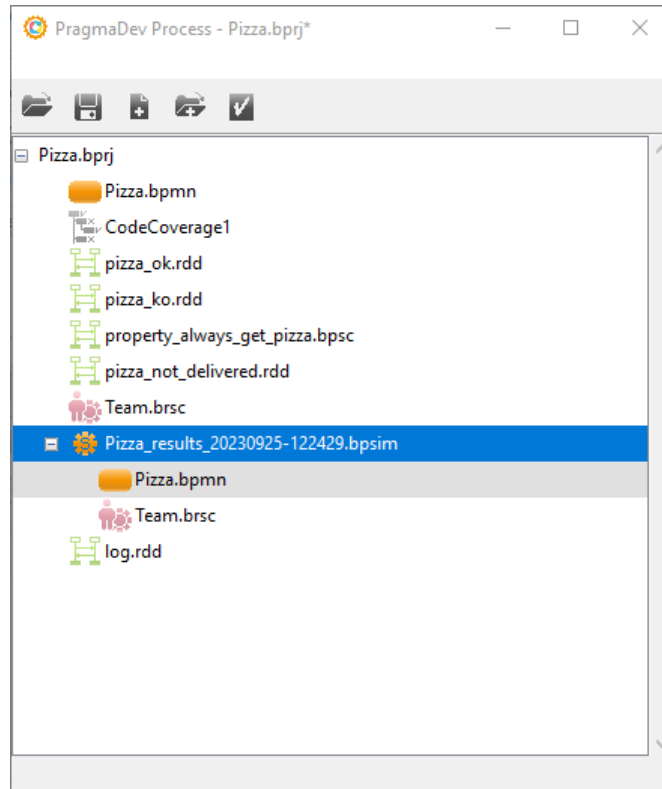| Scenario id.	Scenario name	Element id.	Param. name	min	max	mean	count	sum
SEM_SYMB_74	bike_delivery	SEM_SYMB_40	processing time				997	
SEM_SYMB_74	bike_delivery	SEM_SYMB_40	lag time			14.4044283258		
SEM_SYMB_74	bike_delivery	SEM_SYMB_4_f04a6565a0e766464843044cedf61f06	elapsed time	21.5529981079	181.512284158	117.026818542		
SEM_SYMB_74	bike_delivery	SEM_SYMB_4_f04a6565a0e766464843044cedf61f06	unit cost	0.0525929304574	0.248221128912	0.147681130701		
SEM_SYMB_75	car_delivery	SEM_SYMB_40	processing time				700	
SEM_SYMB_75	car_delivery	SEM_SYMB_4_f04a6565a0e766464843044cedf61f06	elapsed time	22.5566684311	143.873354224	80.9784614735		
SEM_SYMB_75	car_delivery	SEM_SYMB_4_f04a6565a0e766464843044cedf61f06	unit cost	0.519761483762	1.96603417837	1.16008888002		

5.5 Simulation log

With the simulation results shown, go to the "Log" tab. Select one the entries and hit the "Save as MSC..." button:

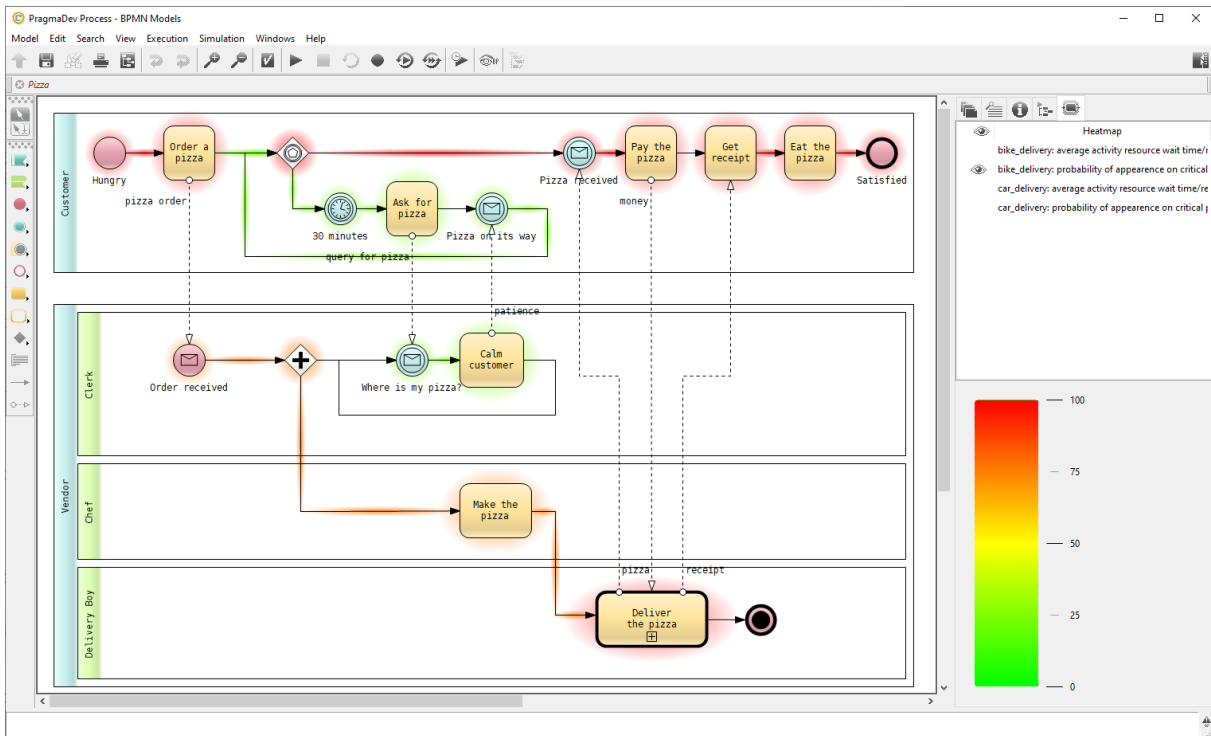Name it `log.rdd` and save it. The trace will be added to the project:
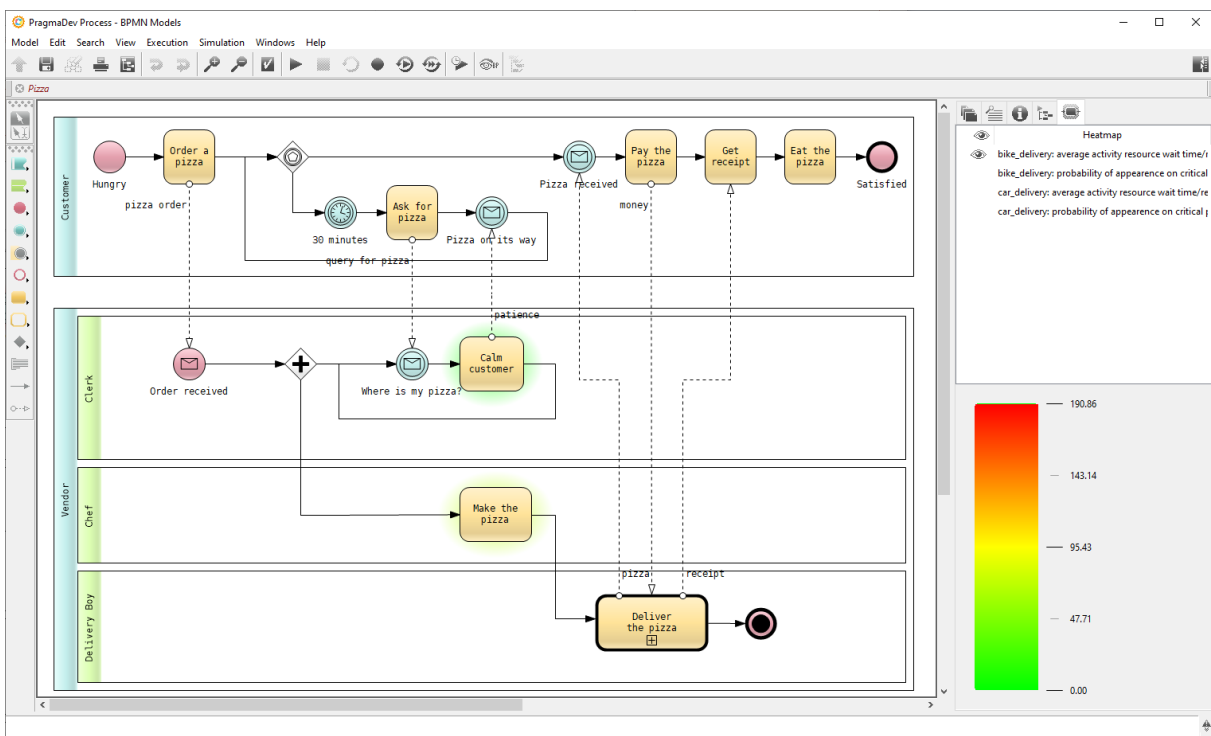
and opened in the MSC editor:
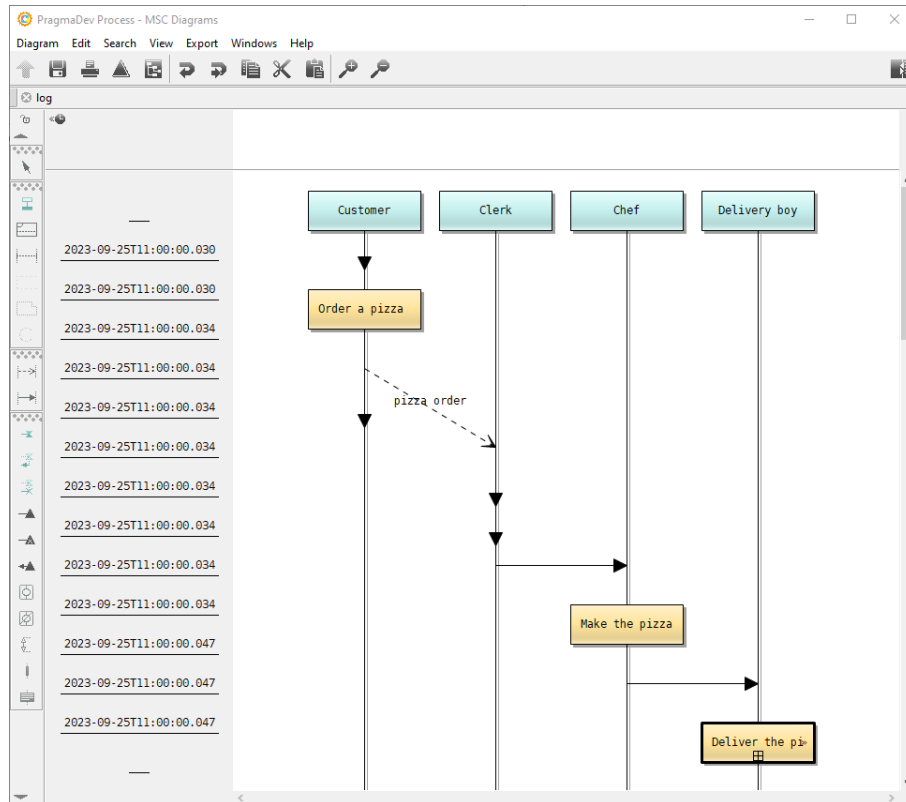
5.6 Critical path

The critical path analysis was checked in the simulation option window. In order to view graphically the critical path, select the Heatmaps tab in the right panel of the editor and one of the scenario.

The one for the bike should look like this:

The waiting time for a resource looks like:

As a first quick analysis, it is missing one unqualified employee so that, one employee stays in the shop to take the calls, while the other one is delivering the pizzas.

6 Conclusion

During this tutorial we have been through:

- BPMN,
- Project Manager,
- BPMN Editor,
- BPMN semantic check,
- BPMN Executor,
    - Interactive,
    - Automatic,
- BPMN Explorer,
    - Complexity,
    - Reachability,
    - Deadlock,
    - Property,
- BPMN Simulator,
    - Resources
    - Time & cost,
    - Scenarios,
    - Log,
    - Optimization & trade-off,
    - Critical path.

PragmaDev Process is a powerful tool that allows creating, editing, checking, executing, exploring, simulating, and optimizing BMPN models.