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**.

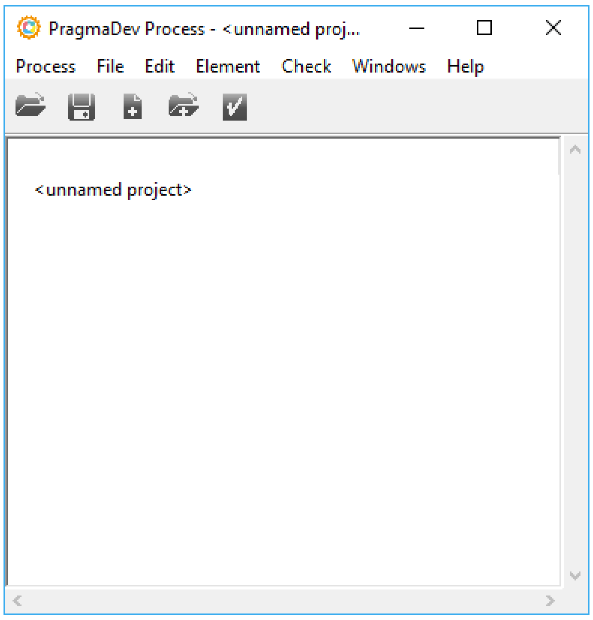
2 The model

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*.

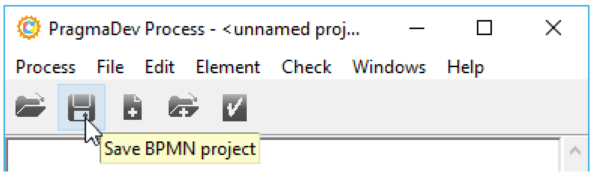
The BPMN model of the pizza delivery used in this tutorial is found in `$PRAGMADEV_PROCESS_HOME/examples/Pizza`.

2.1 Organization

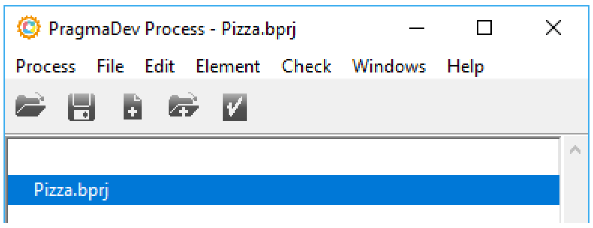
Let's get our hands on the tool! Start PragmaDev Process. The window that appears is called the Project Manager:


PragmaDev Process - <unnamed proj...
Process File Edit Element Check Windows Help
<unnamed project>

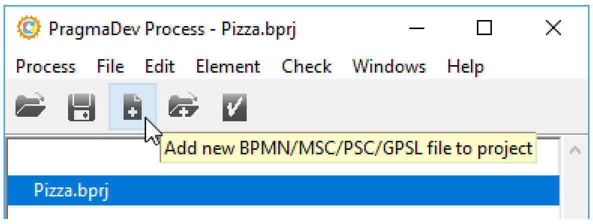
Save the project via the button in the toolbar:


PragmaDev Process - <unnamed proj...
Process File Edit Element Check Windows Help
Save BPMN project

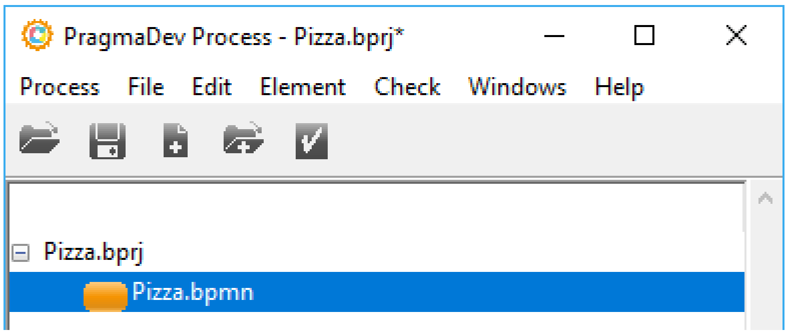
Name the project `Pizza`:


PragmaDev Process - Pizza.bprj
Process File Edit Element Check Windows Help
Pizza.bprj

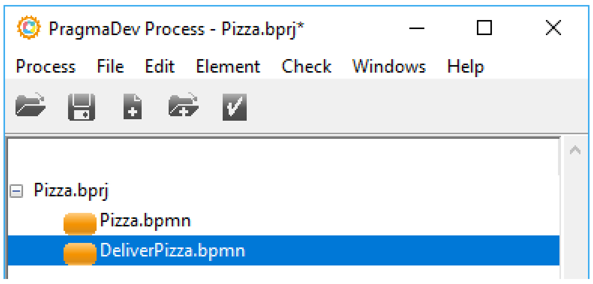
Create and add a new BPMN file to the project via the button in the toolbar:


PragmaDev Process - Pizza.bprj
Process File Edit Element Check Windows Help
Add new BPMN/MSC/PSC/GPSL file to project
Pizza.bprj

Name it `Pizza`:

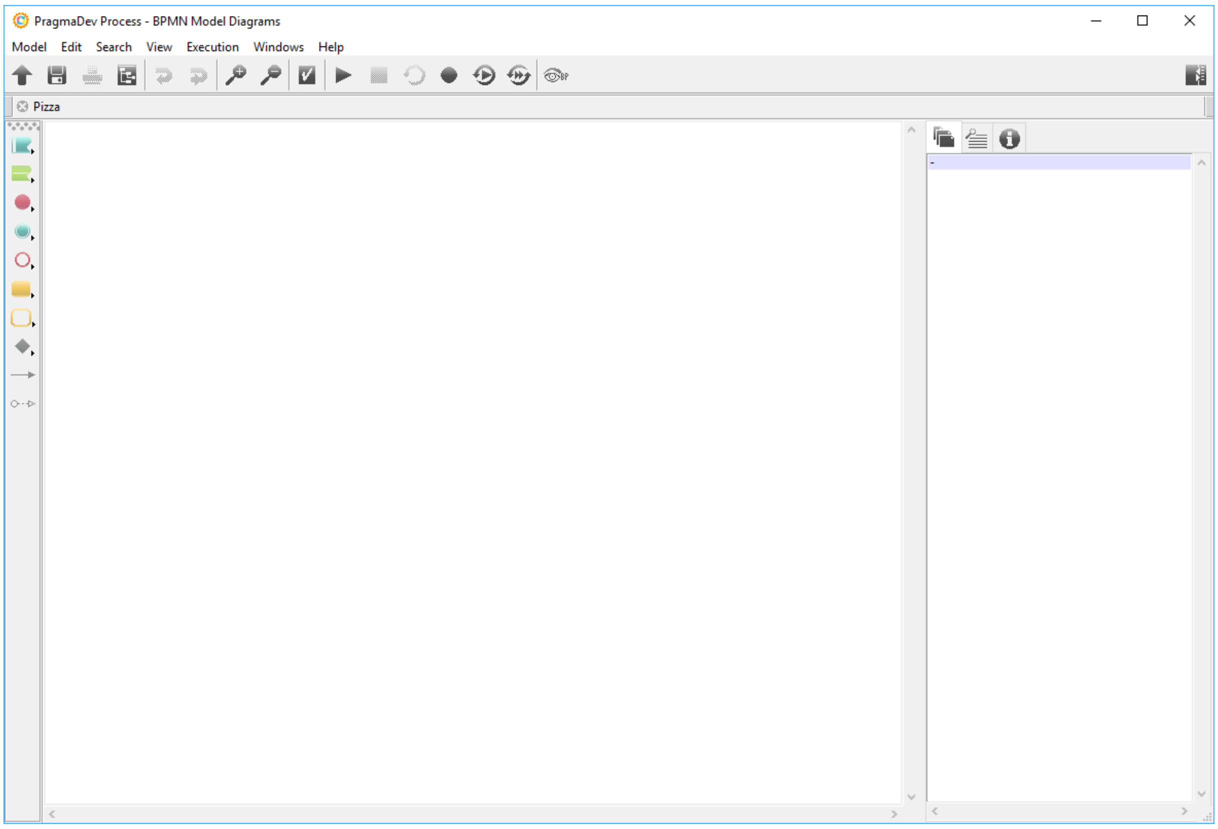

PragmaDev Process - Pizza.bprj*
Process File Edit Element Check Windows Help
Pizza.bprj
Pizza.bpmn

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

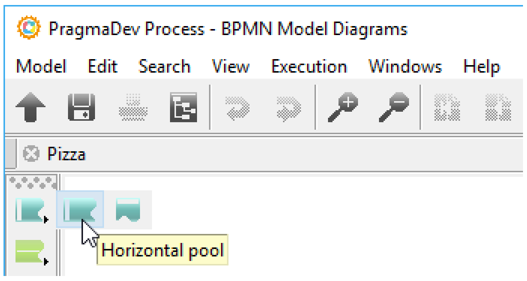

PragmaDev Process - Pizza.bprj*
Process File Edit Element Check Windows Help
Pizza.bprj
Pizza.bpmn
DeliverPizza.bpmn

2.2 Pools and lanes

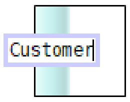
Double click `Pizza.bpmn` in the Project Manager to open it in the BPMN Editor:


PragmaDev Process - BPMN Model Diagrams
Model Edit Search View Execution Windows Help
Pizza

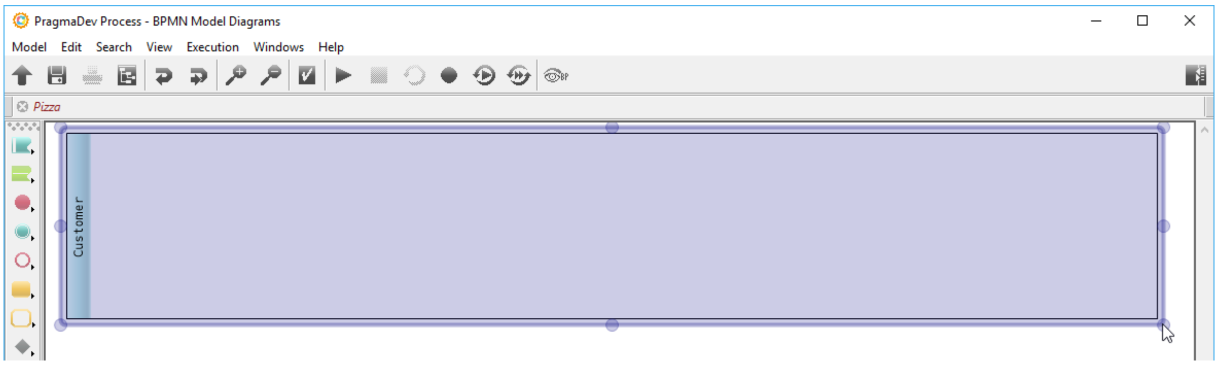
Create a horizontal pool via the tool button:


PragmaDev Process - BPMN Model Diagrams
Model Edit Search View Execution Windows Help
Pizza
Horizontal pool

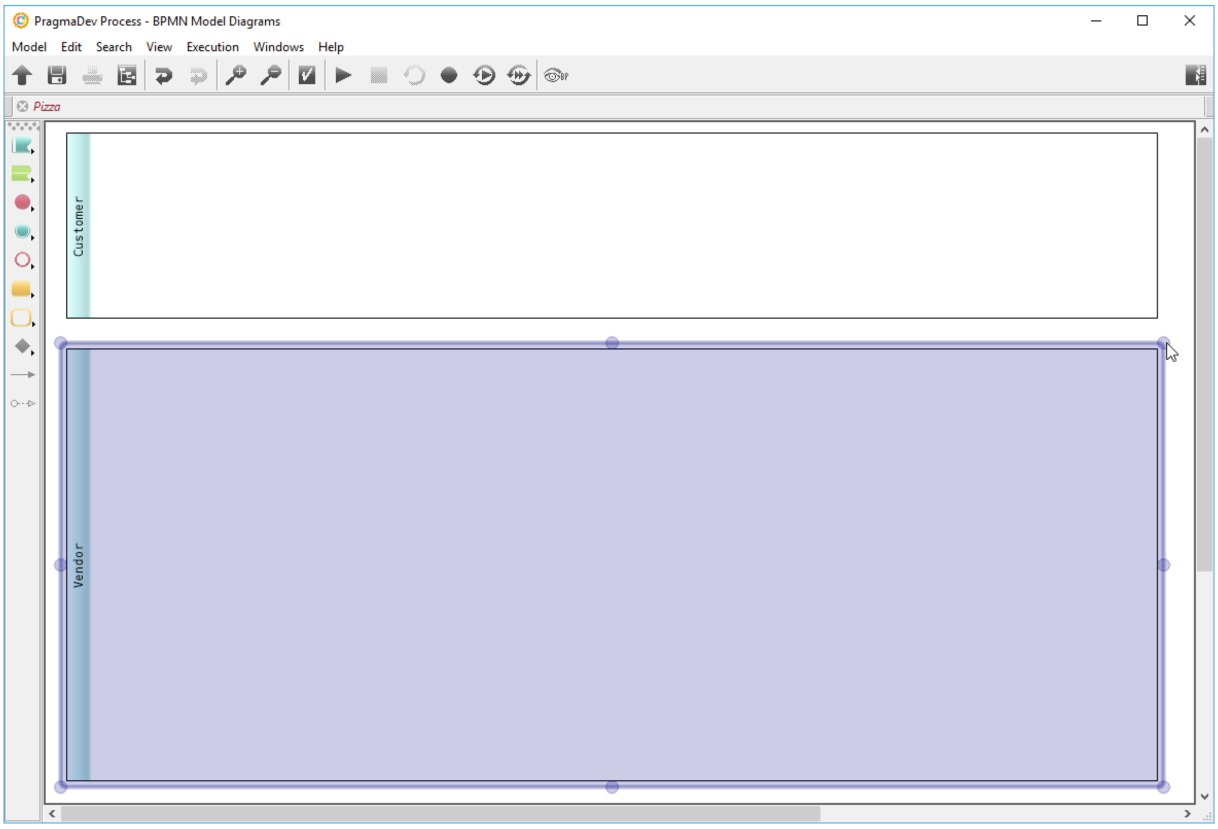
and name it `Customer`:



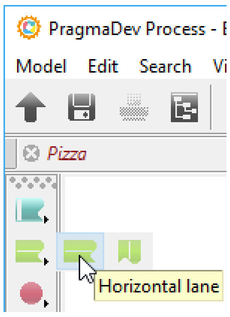
Resize the pool using the points to create space for other elements:



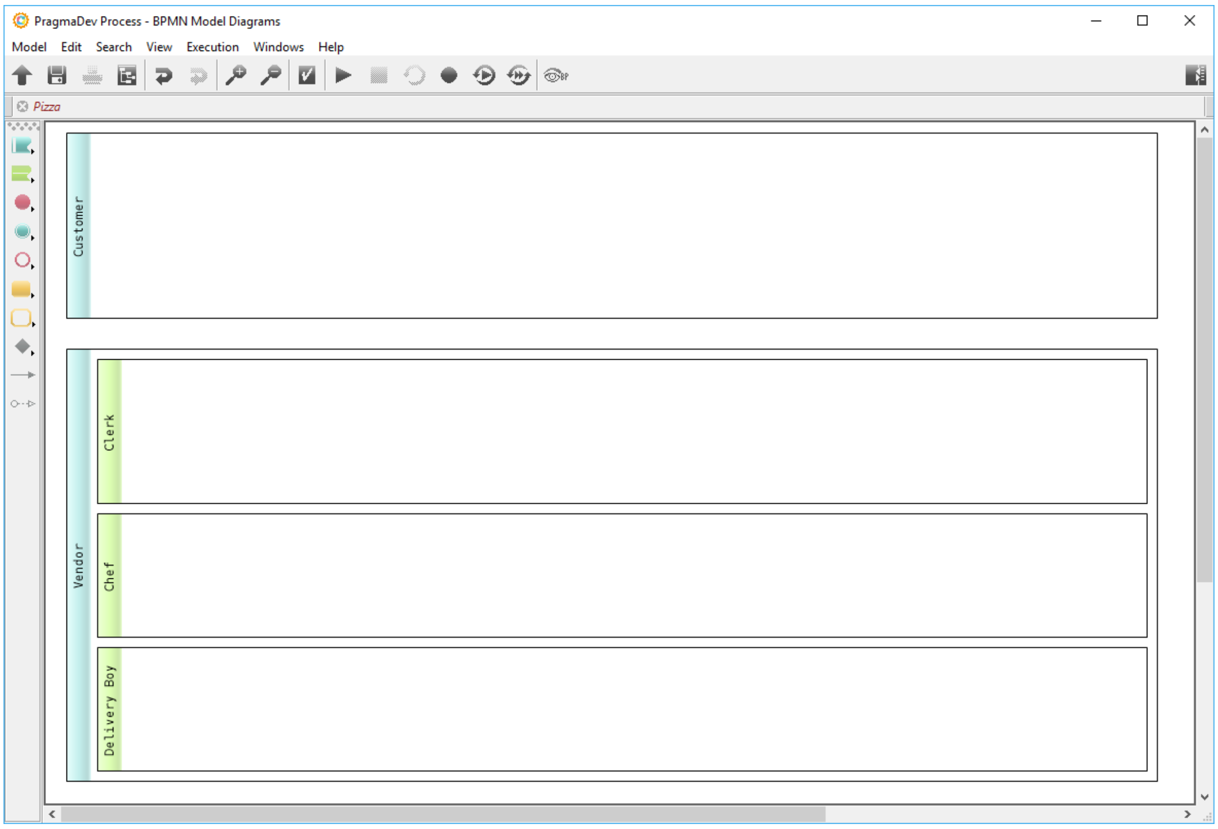
Create another pool named `Vendor` and resize it:



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

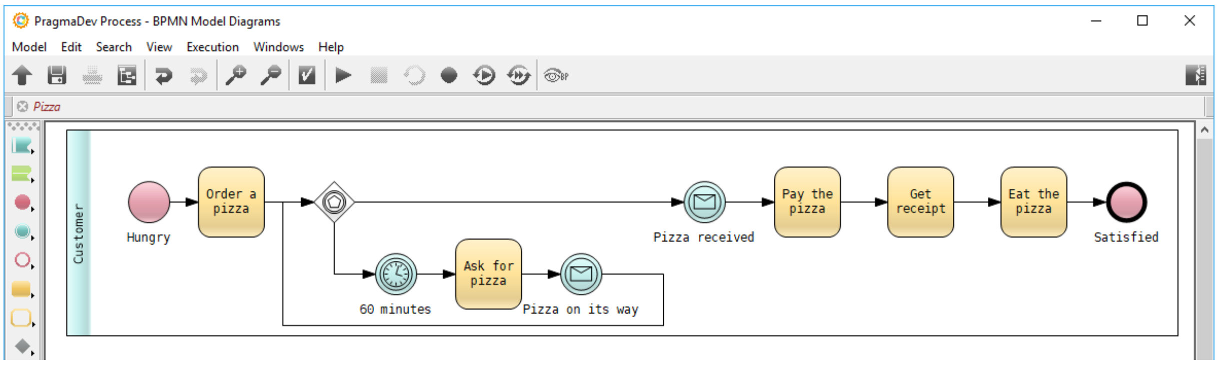


Name them `Clerk`, `Chef`, `Delivery Boy`, and resize them as shown:


PragmaDev Process - BPMN Model Diagrams
Model Edit Search View Execution Windows Help
Pizza
Customer
Vendor
Clerk
Chef
Delivery Boy

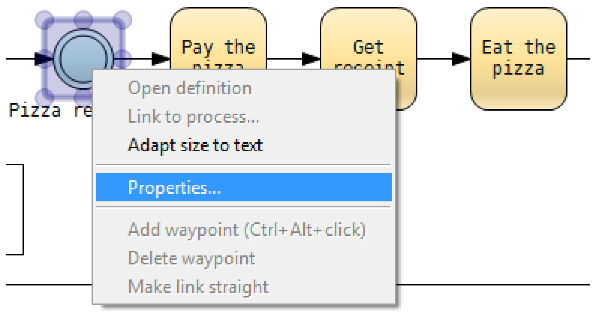
2.3 Processes

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

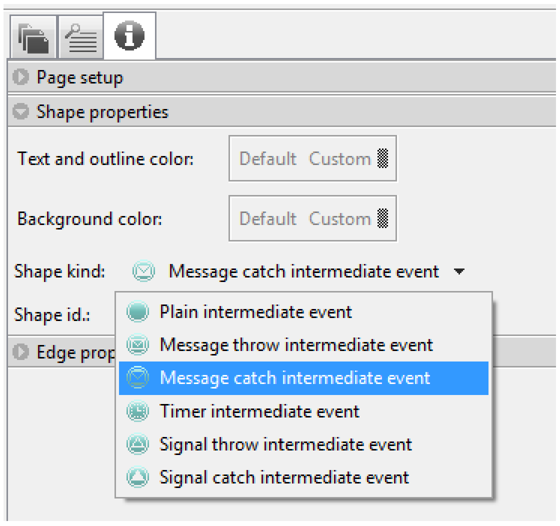


Symbols can either be created with the tools in the toolbar, or via keyboard shortcuts. The insertion shortcuts are all introduced by Control + Space (or Command + Control + Space on macOS), followed by a letter giving the type of symbol to create. For example, to create a start event, press Control + 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:


Page setup
Shape properties
Edge prop
Plain intermediate event
Message throw intermediate event
Message catch intermediate event
Timer intermediate event
Signal throw intermediate event
Signal catch intermediate event

Text and outline color: Default Custom

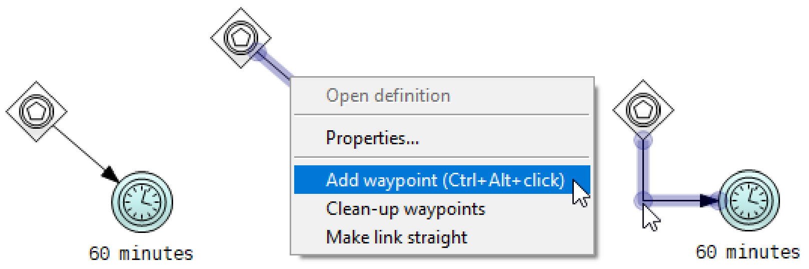
Background color: Default Custom

Shape kind: Message catch intermediate event

Shape id.:

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:

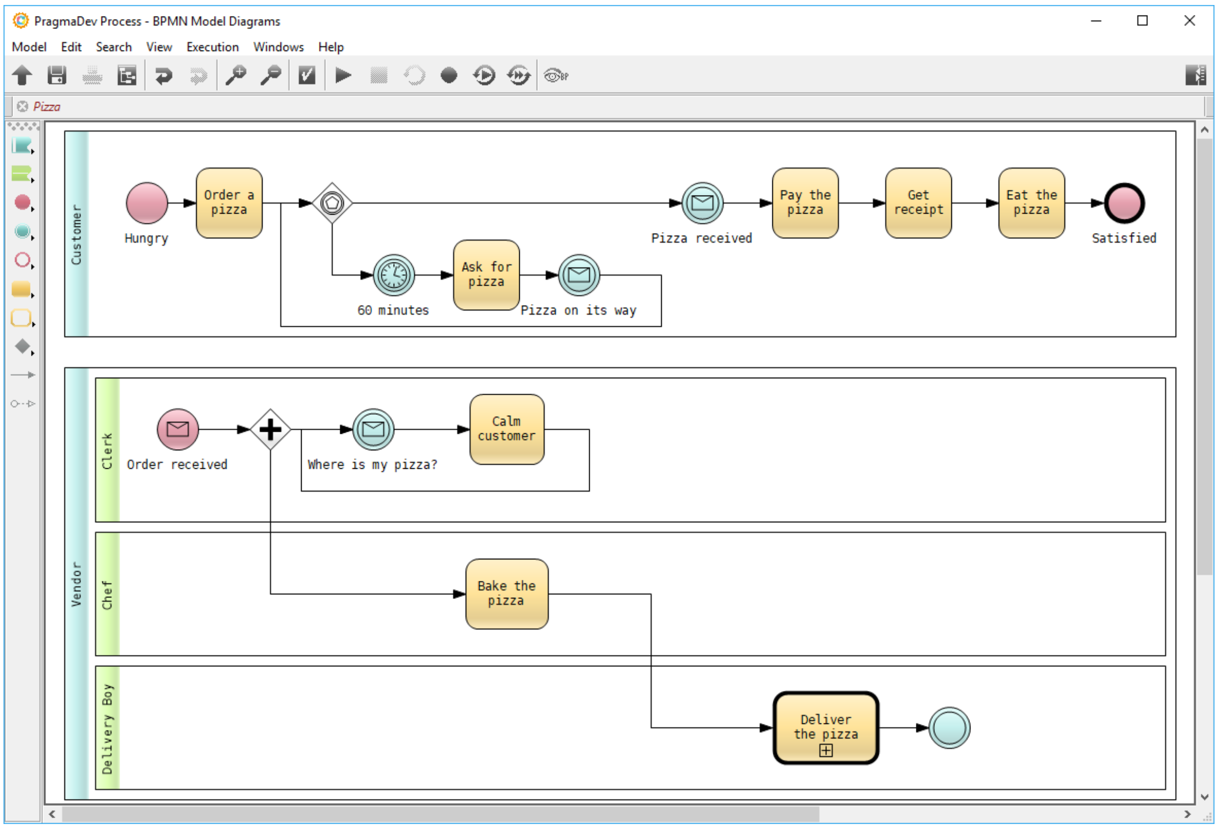


Non-linear paths can be created also by holding *Shift* and clicking to place waypoints while drawing the path from the source to the target symbol.

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:



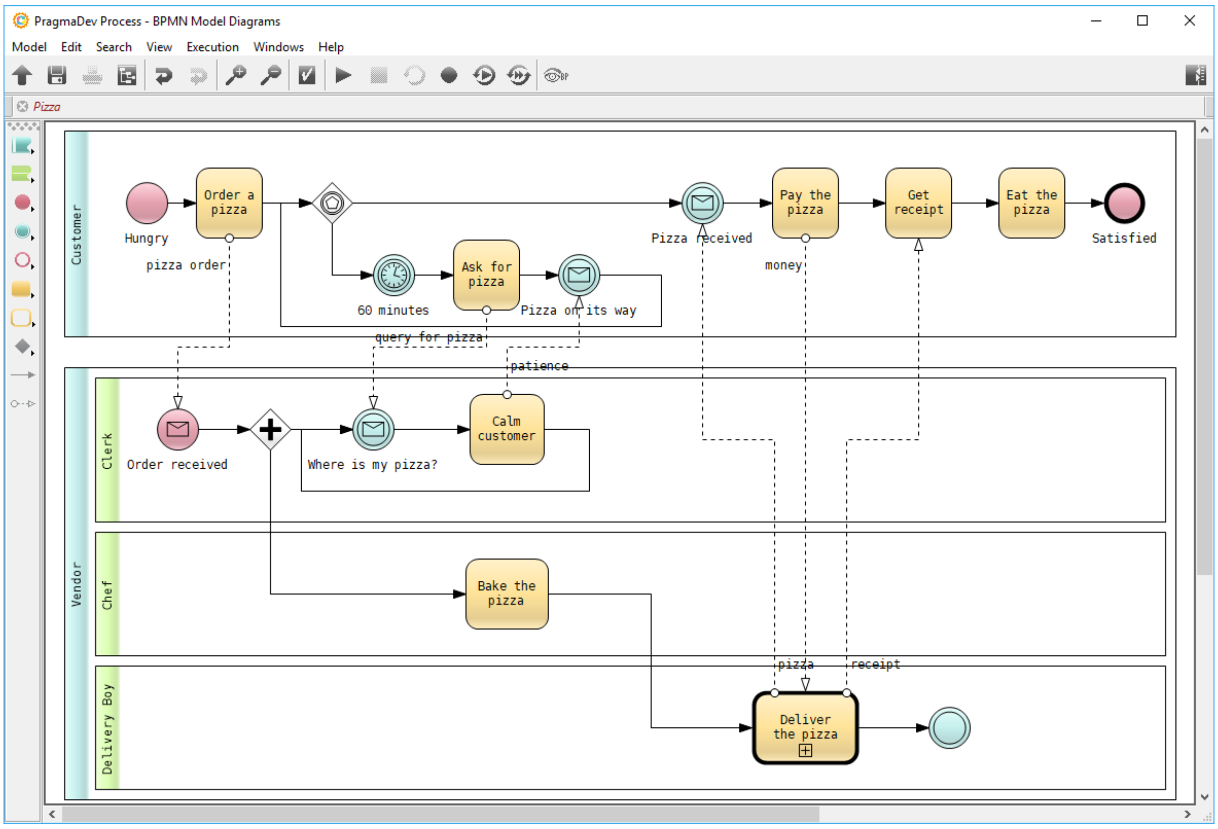
The `Vendor` process says the following:

- 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`.

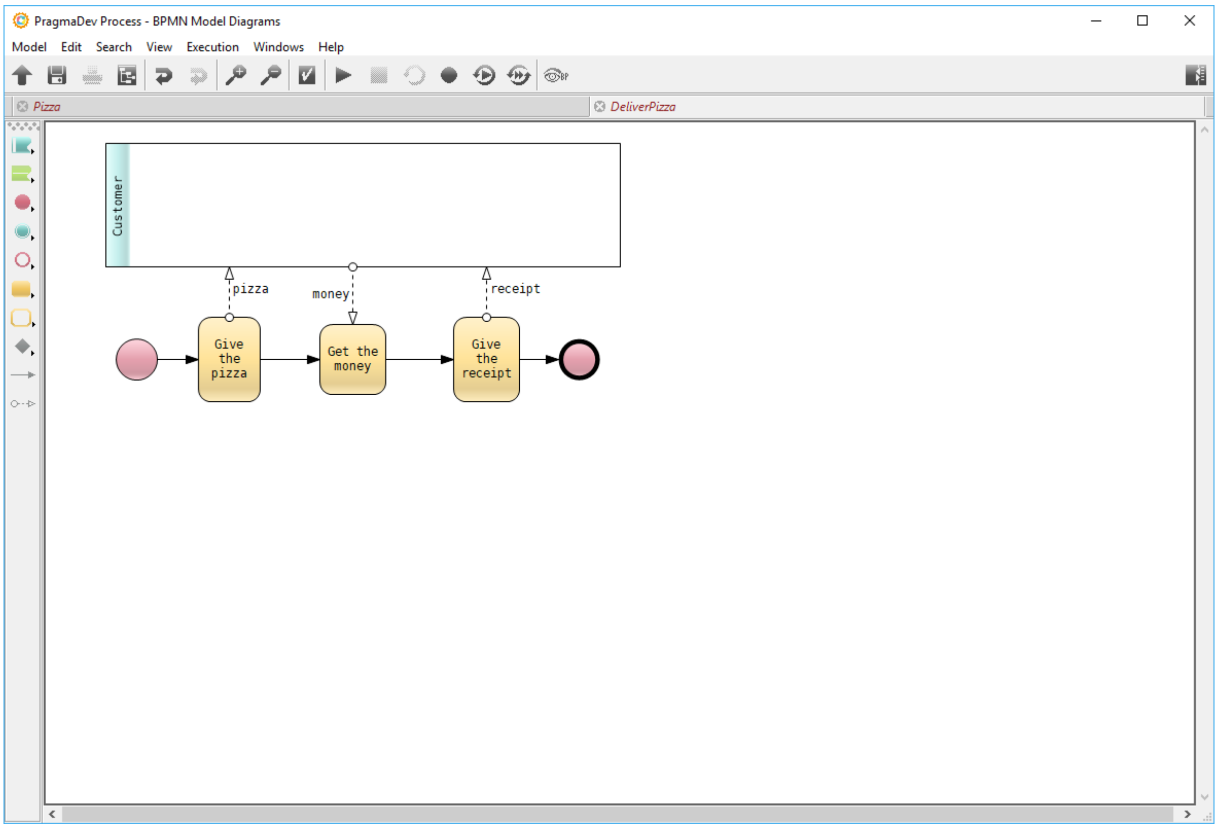
Notes:

- 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:

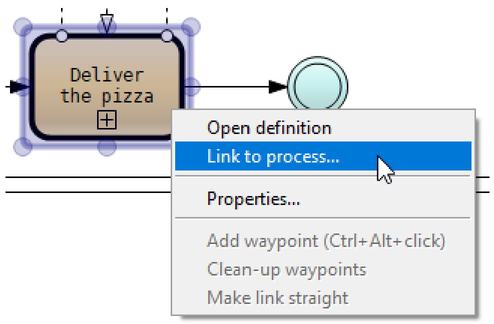


Let's define now the call-activity `Deliver the pizza`. In the Project Manager double click `DeliverPizza.bpmn` to open it in the editor. Draw the following:

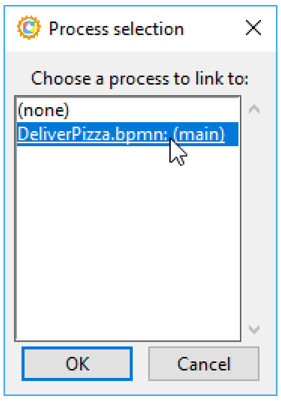


When creating the message flows be sure to draw starting from / ending to either the border or header of the pool `Customer`.

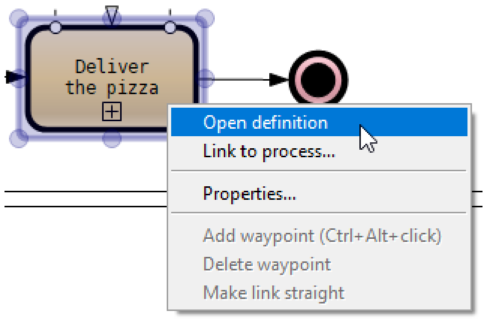
Save the diagram and go back to the `Pizza.bpmn`. Right click the call-activity and then *Link to process...*:



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.

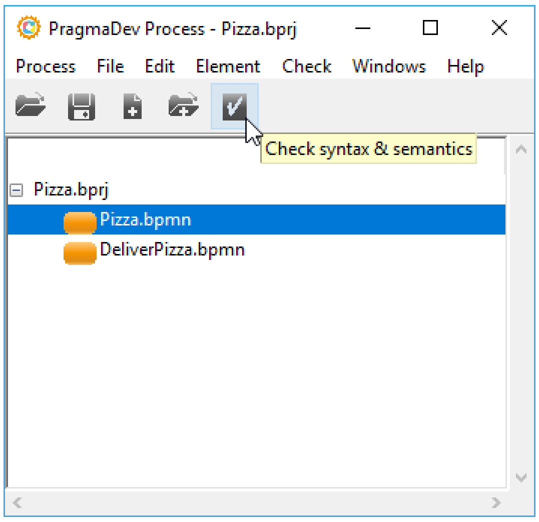
3 Execution

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

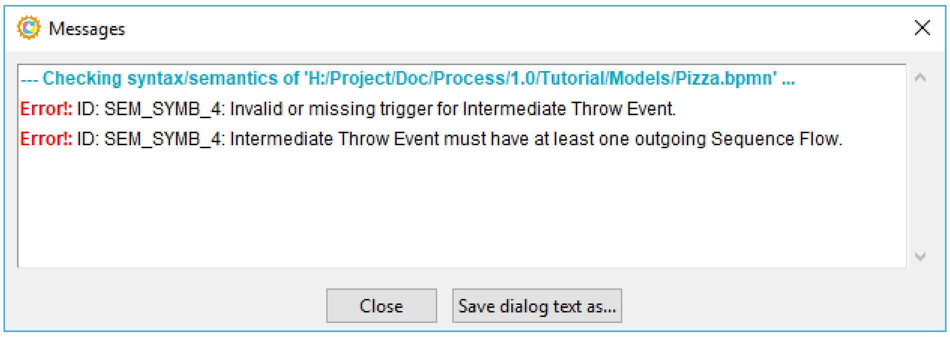
- *Interactive*: allows the user to have full control over the execution. The user interacts with the model at every step of its execution.
- *Automatic*: enables model execution without user input. The execution is guided by MSC traces.

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:

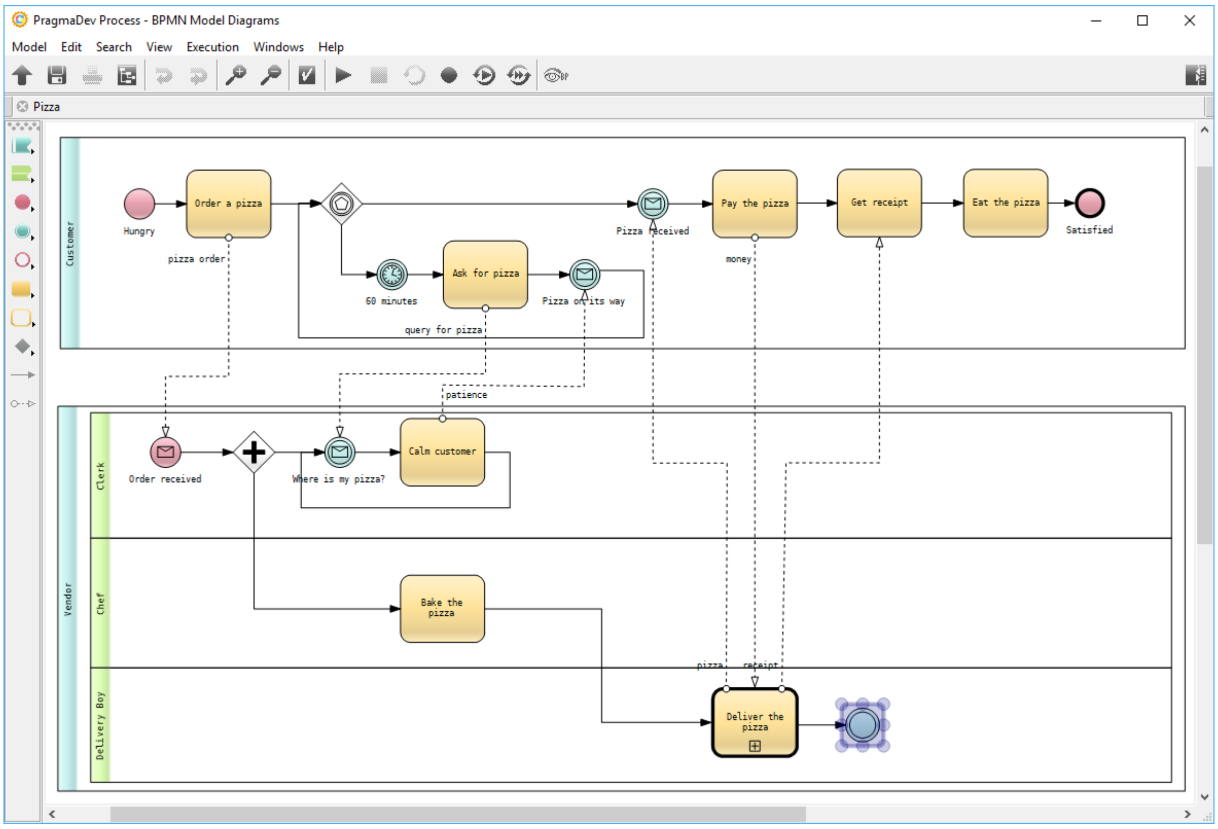

PragmaDev Process - Pizza.bprj
Process File Edit Element Check Windows Help
Check syntax & semantics
Pizza.bprj
Pizza.bpmn
DeliverPizza.bpmn

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

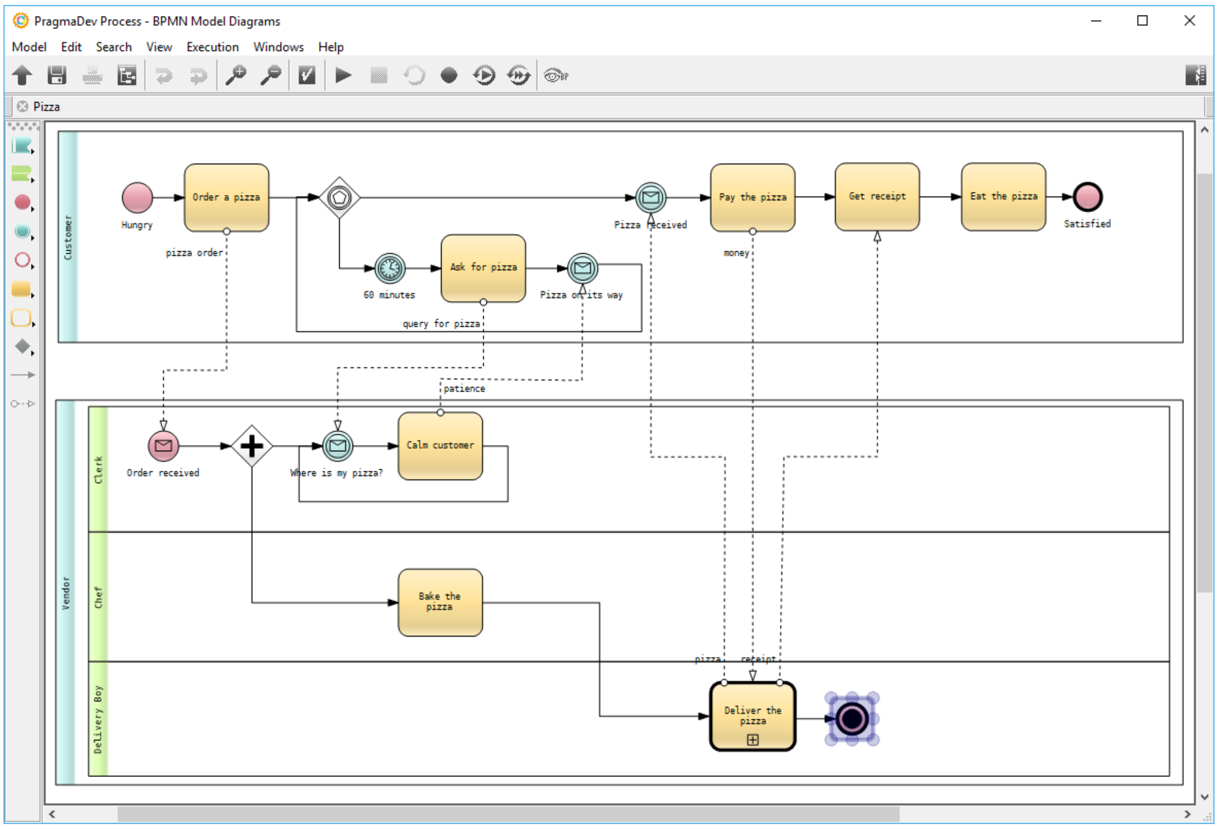


```
--- Checking syntax/semantics of 'H:/Project/Doc/Process/1.0/Tutorial/Models/Pizza.bpmn' ...
Error!: ID: SEM_SYMB_4: Invalid or missing trigger for Intermediate Throw Event.
Error!: ID: SEM_SYMB_4: Intermediate Throw Event must have at least one outgoing Sequence Flow.
```

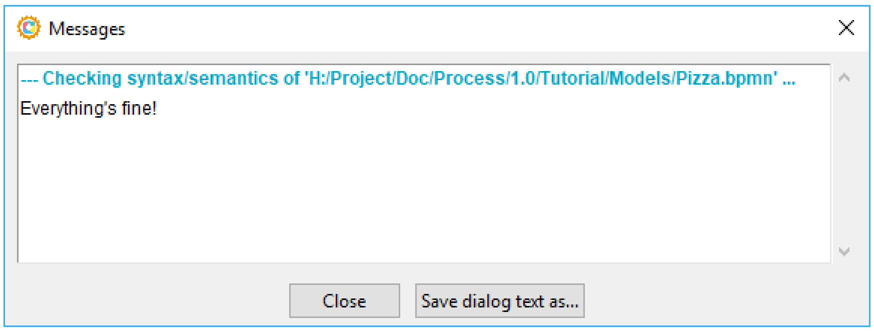
Double-clicking either of the listed errors will open the model in the editor and the concerned element will be selected:



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:

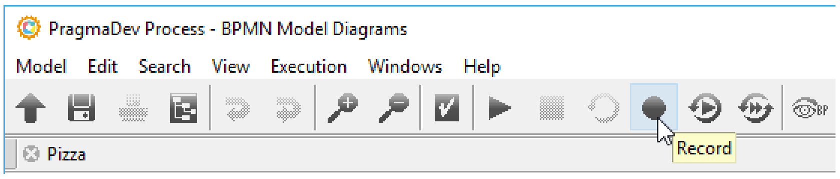

Messages

--- Checking syntax/semantics of 'H:/Project/Doc/Process/1.0/Tutorial/Models/Pizza.bpmn' ...
Everything's fine!

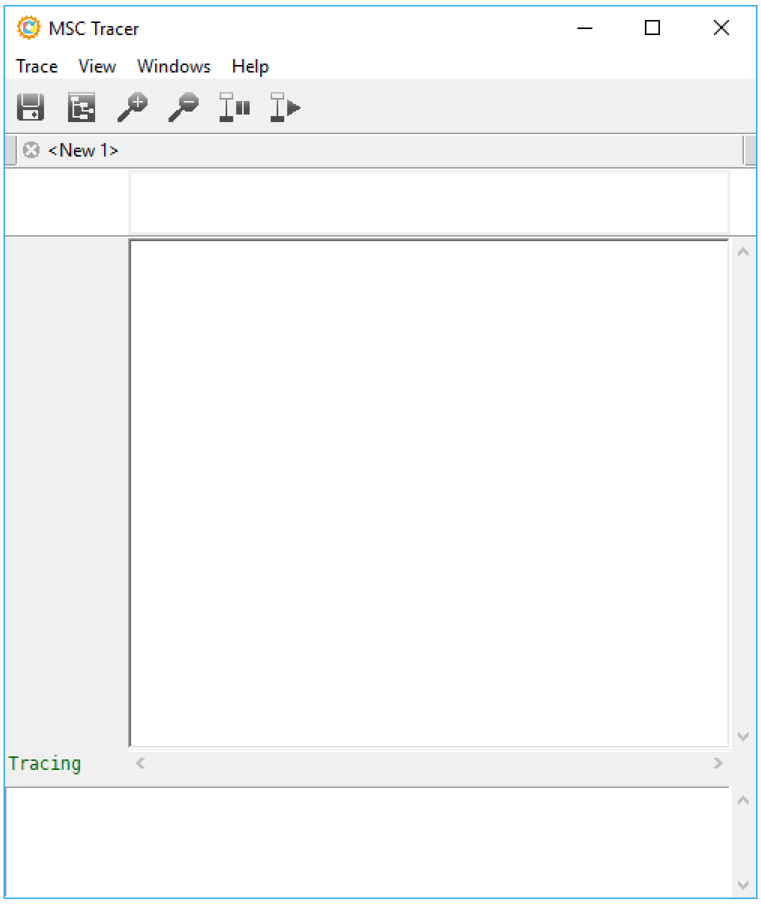
Close Save dialog text as...

3.2 Interactive execution

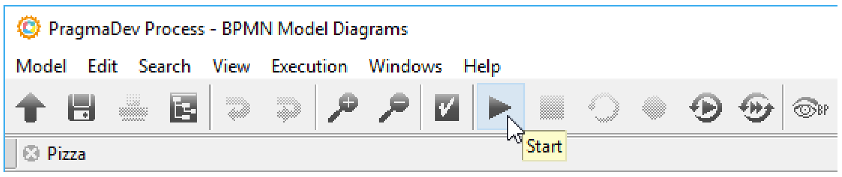
With `Pizza.bpmn` opened in the editor (double-click on it in the project manager), click the *Record* button:


PragmaDev Process - BPMN Model Diagrams
Model Edit Search View Execution Windows Help
Record
Pizza

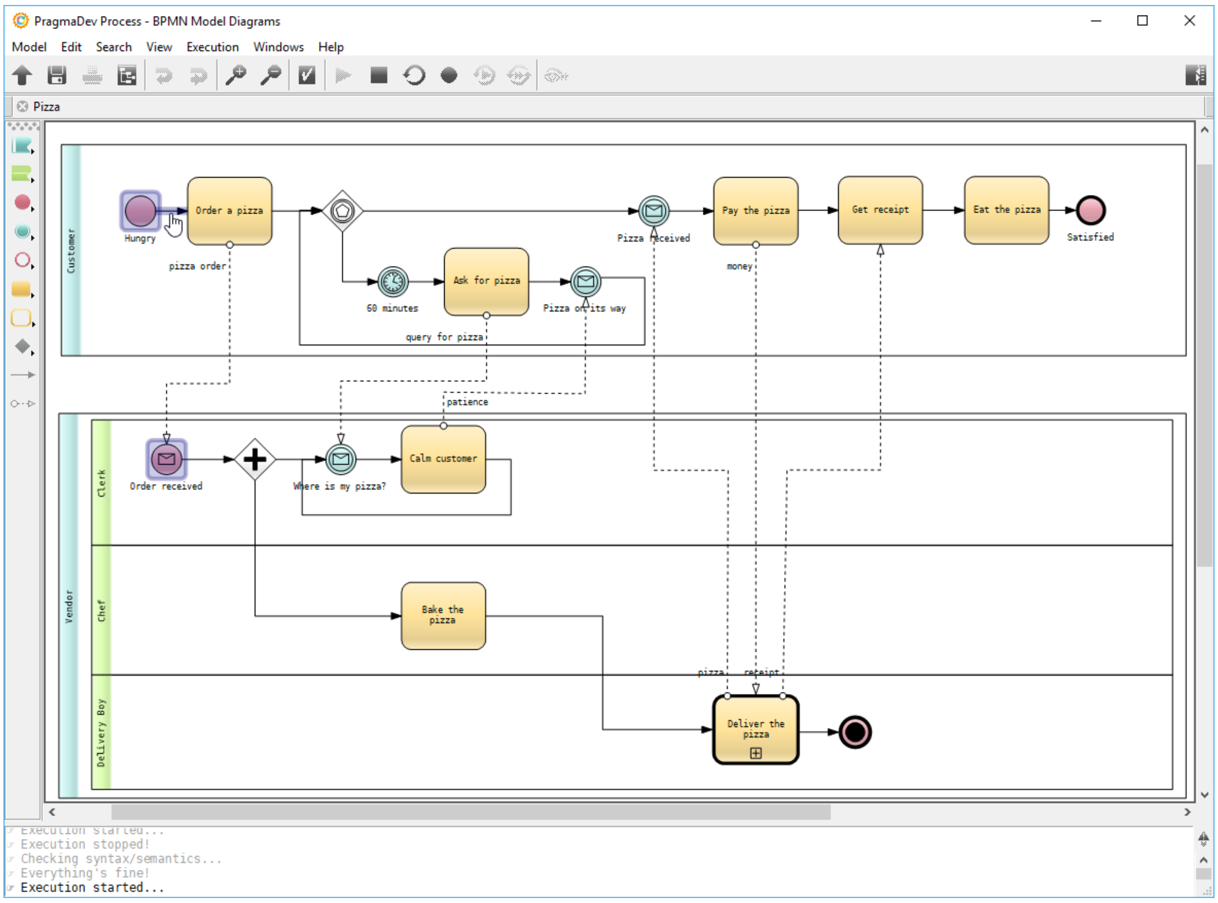
This will allow us to record the execution in the *MSC Tracer*:



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

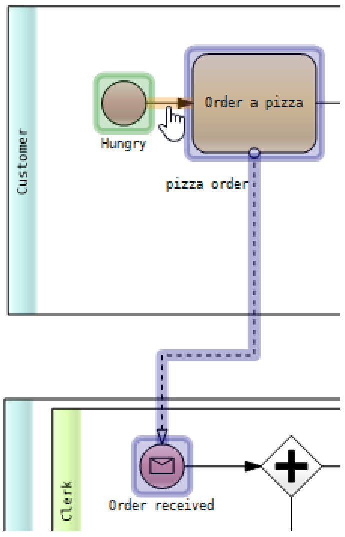

PragmaDev Process - BPMN Model Diagrams
Model Edit Search View Execution Windows Help
Start
Pizza

Start events and executable flows will be marked in blue:

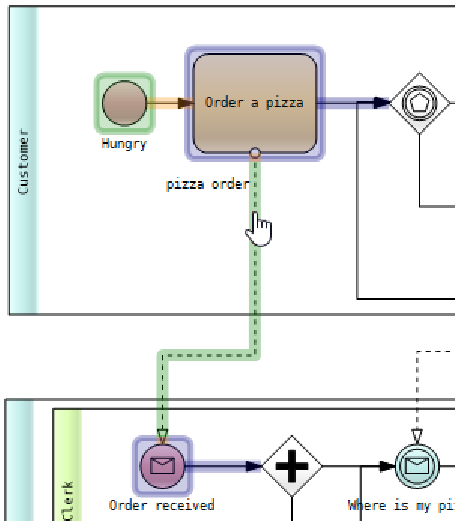


```
Execution started...
Execution stopped!
Checking syntax/semantics...
Everything's fine!
Execution started...
```

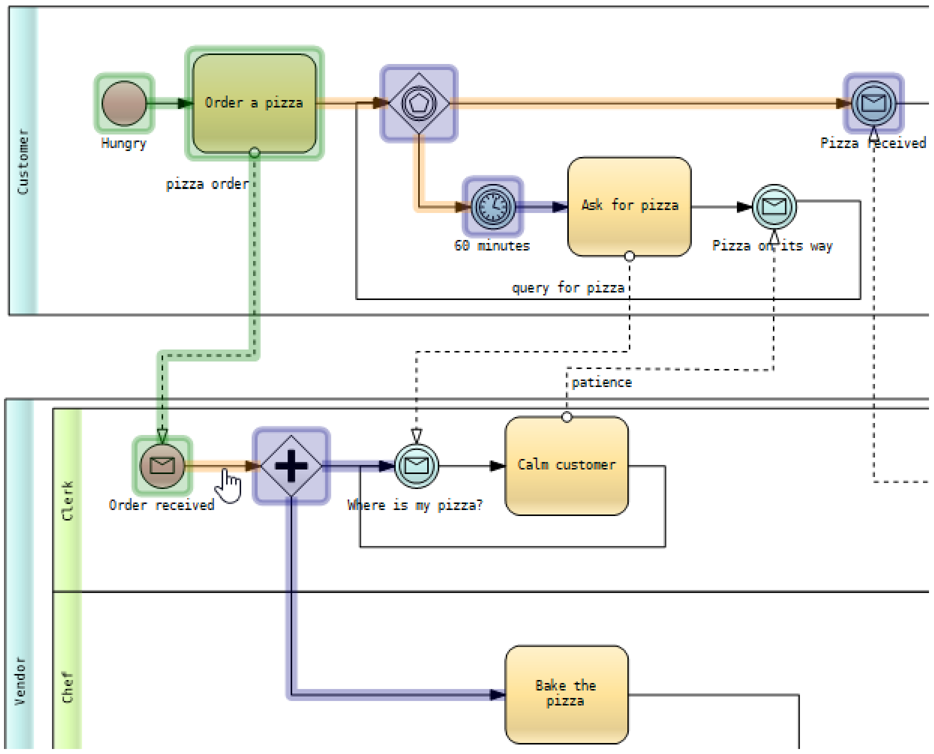
Please note that generally speaking only sequence flows, message flows, or call-activities can be clicked. No other symbol can be clicked even though they appear in blue. In our example there is a single sequence flow that can be executed. Clicking on it will advance execution, i.e., make the `Customer`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

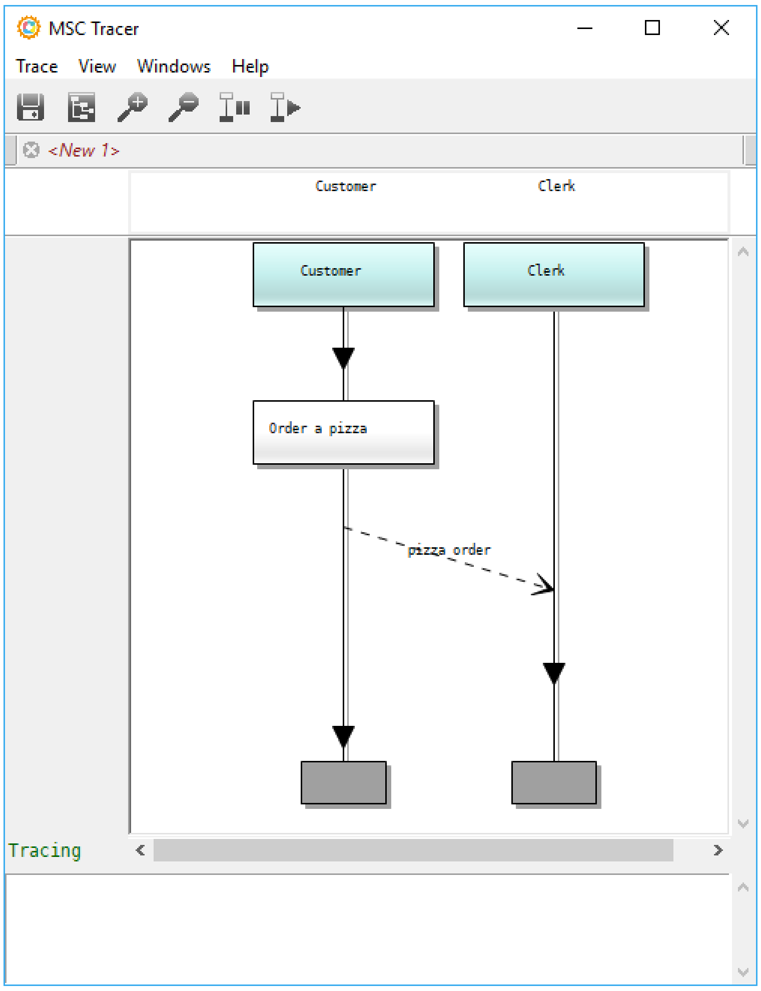


There are two independent sequence flows that can be executed. Clicking both flows will result in:

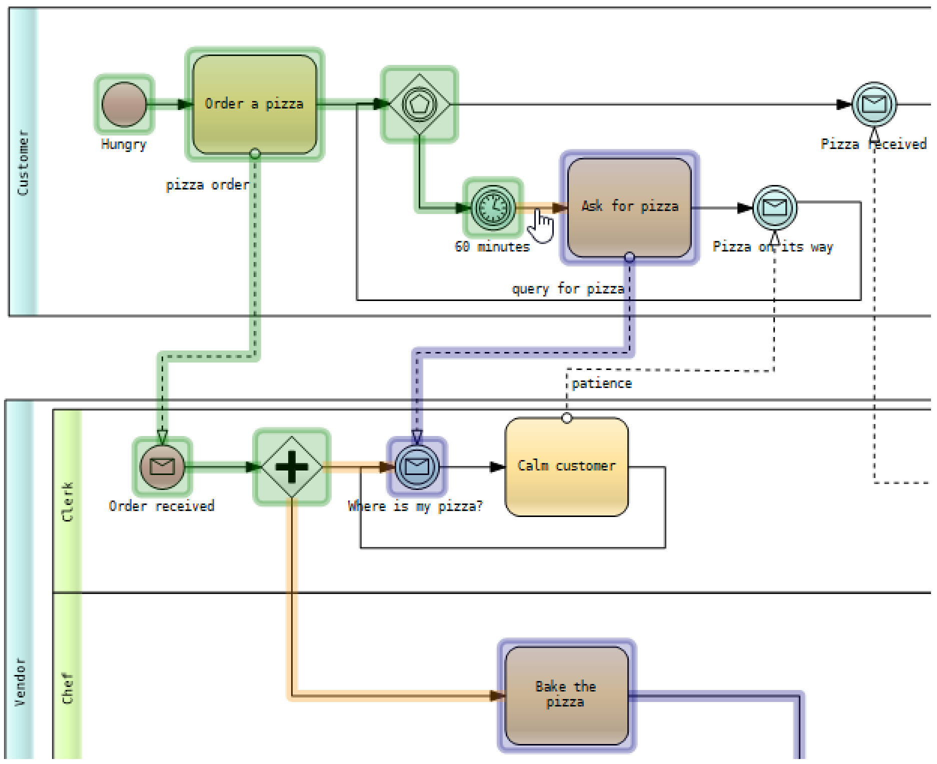


By entering the event gateway, the `Customer` will wait for its pizza to be delivered (i.e., the `Pizza received` event) and start a `60 minutes` timer. If the timer fires before the pizza is delivered, the `Customer` will ask the `Clerk` for the pizza. On the other hand, the parallel gateway in the `Vendor` process will allow the `Clerk` to listen to any `Customer` queries, while the `Chef` starts baking the pizza.

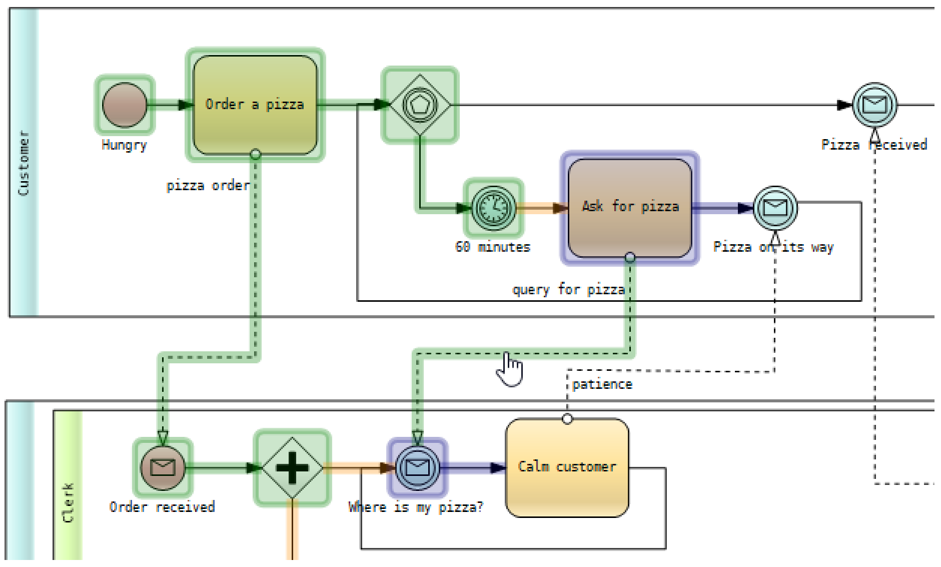
While execution advances in the editor, all steps are also being recorded in the *MSC Tracer*:



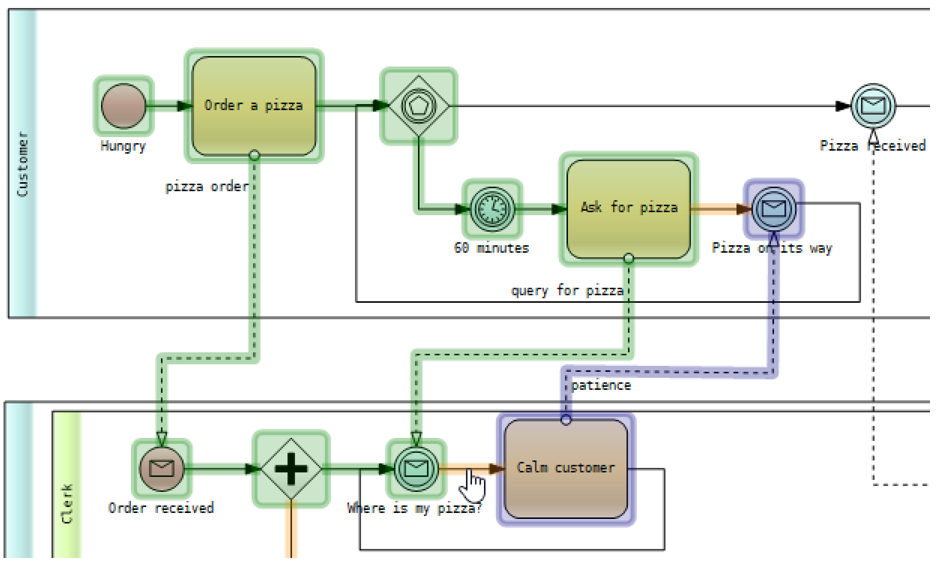
Let’s make the timer fire, the `Clerk` listen for `Customer` queries, and the `Chef` start baking the pizza by clicking all three sequence flows:



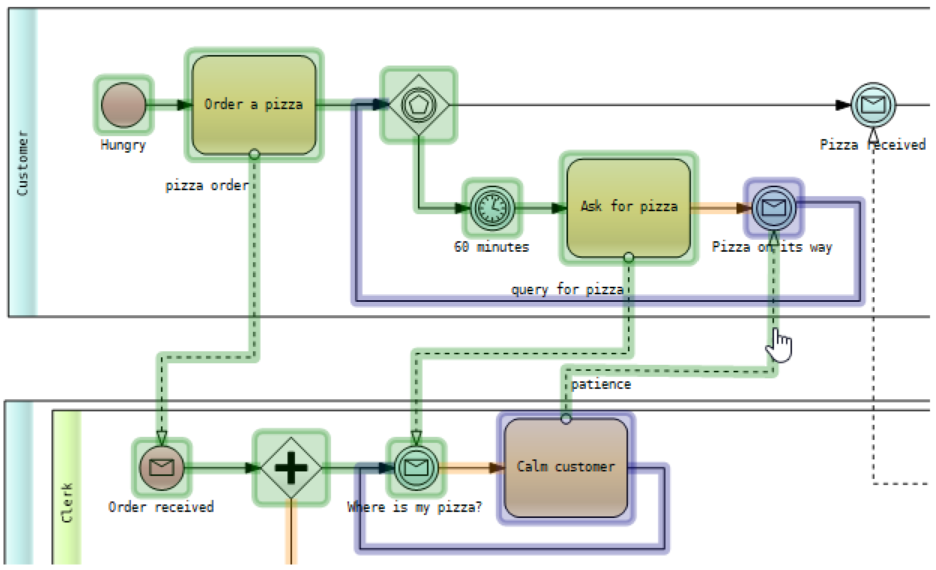
Notice that the `Pizza received` event is no longer active because the `60 minutes` event was triggered. In this situation the `Customer` is asking the `Clerk` for his/her pizza, while the pizza is being baked by the `Chef`. If `query for pizza` is sent:



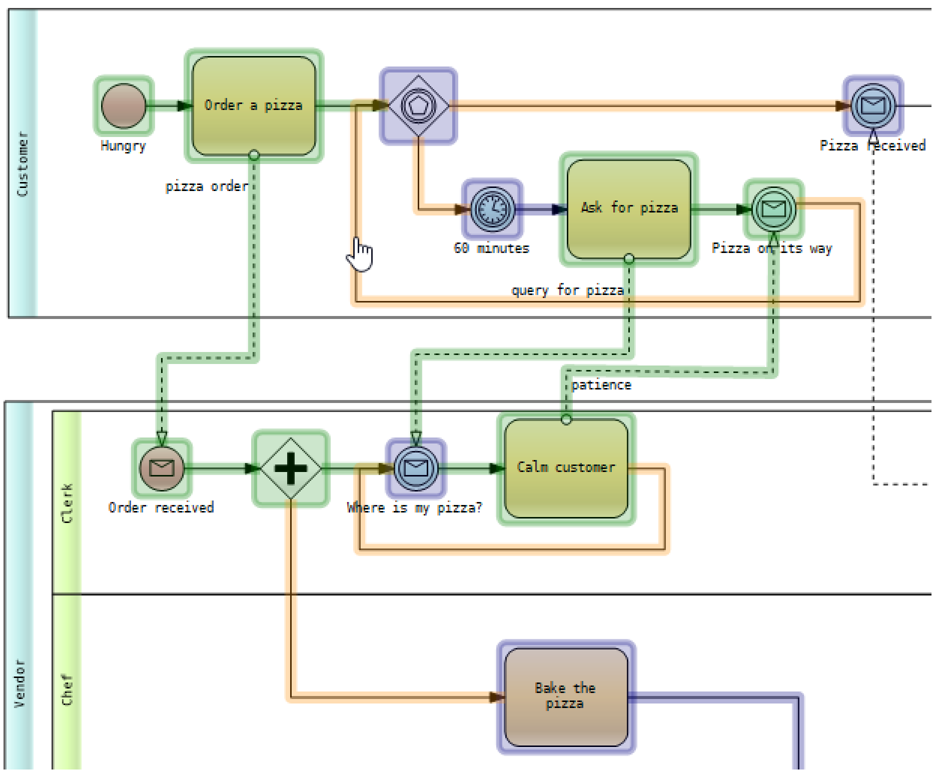
By clicking the two sequence flows, the `Clerk` will `Calm customer` while the later will be waiting for a feedback for his/her query (i.e., the `Pizza on its way` event):



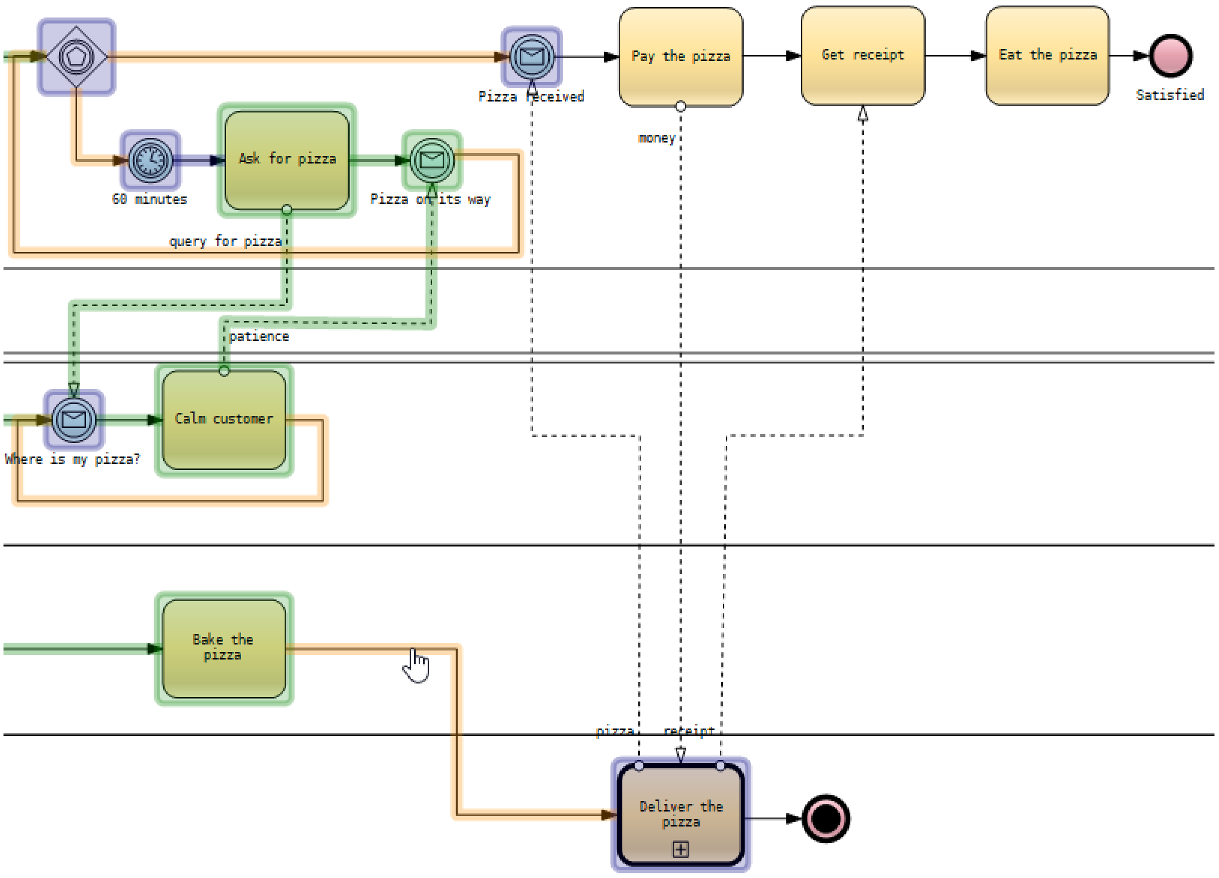
Make the `Clerk` respond to the `Customer`'s query by clicking the `patience` message flow:



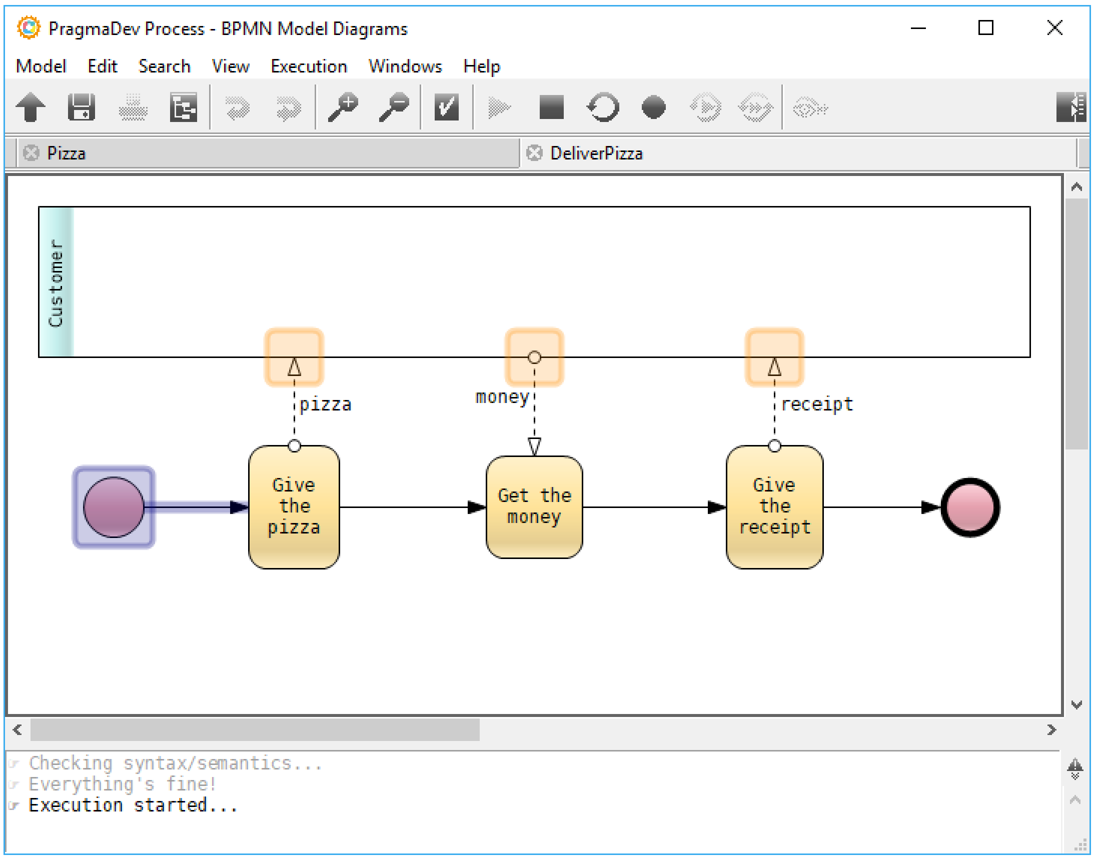
Continue by clicking the two looping sequence flows:



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`:

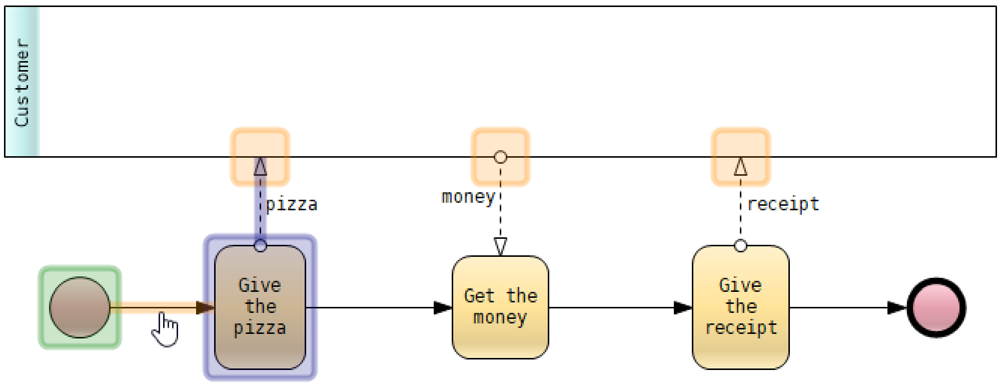


Now its time to `Deliver the pizza`. This is a BPMN call-activity which calls the process defined in `DeliverPizza.bpmn`. We can either step-over (right-click) or step-in (left-click) the call-activity. Stepping over the call-activity will not execute the called process, but will consider the activity as being a simple BPMN task. In this case the execution will block (deadlock) 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:

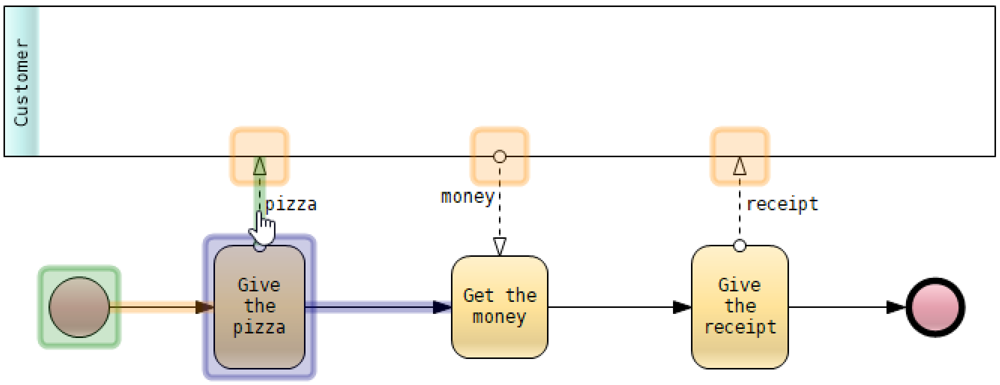


```
Checking syntax/semantics...
Everything's fine!
Execution started...
```

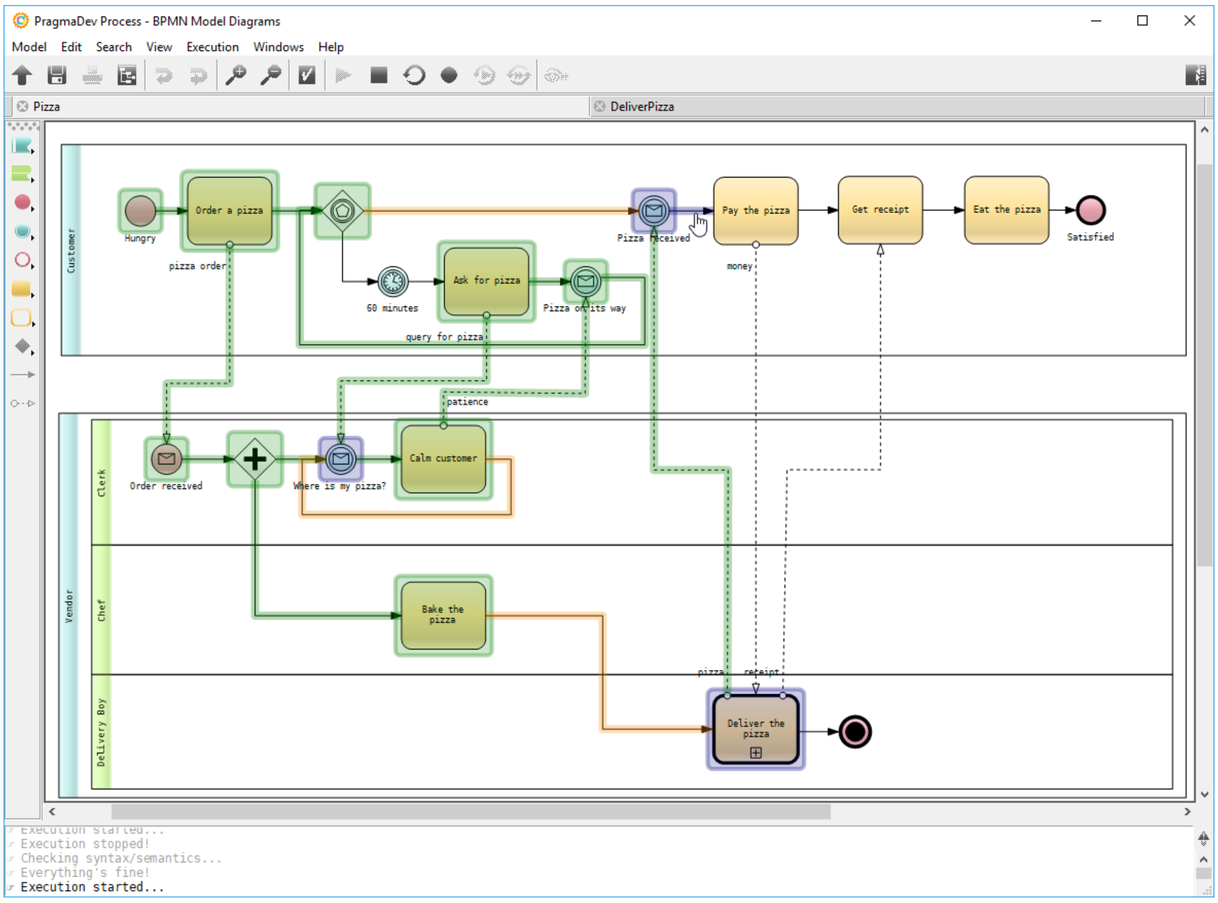
We can `Give the pizza` to the `Customer` by clicking the sequence flow:



and then the message flow of the `pizza`:

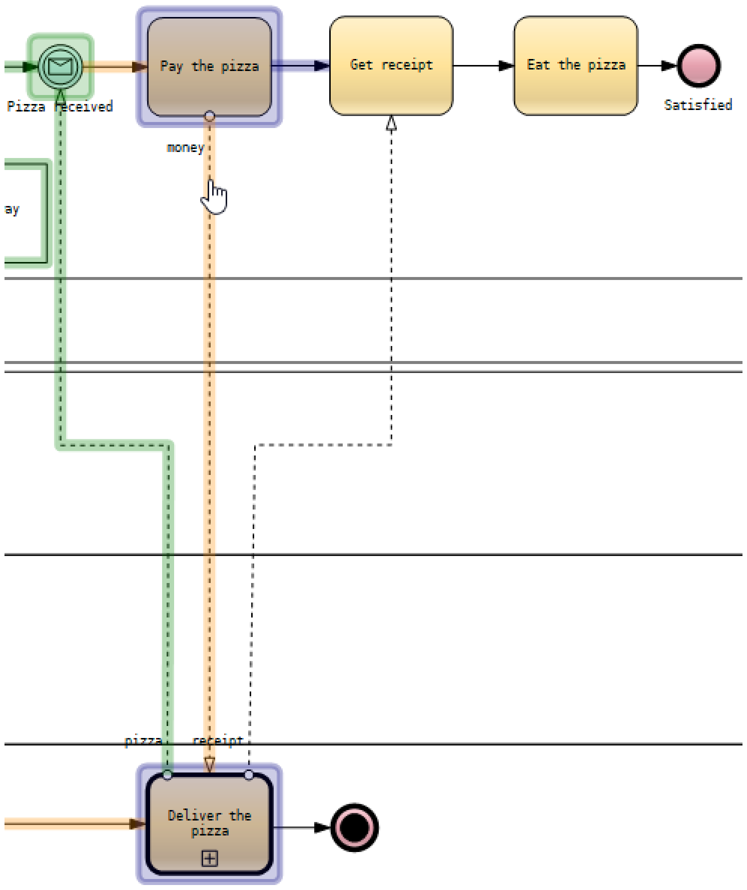


If we switch to the `Pizza.bpmn` in the editor we can see that the `Customer` has indeed received the pizza and can proceed with the payment:

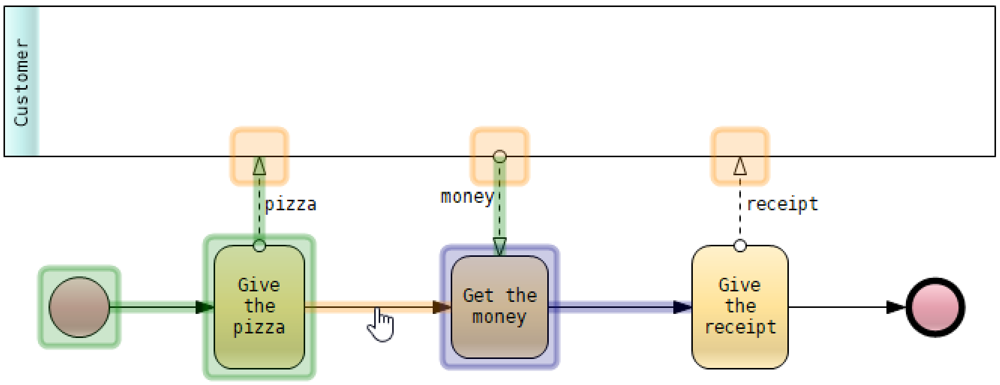


```
Execution started...
Execution stopped!
Checking syntax/semantics...
Everything's fine!
Execution started...
```

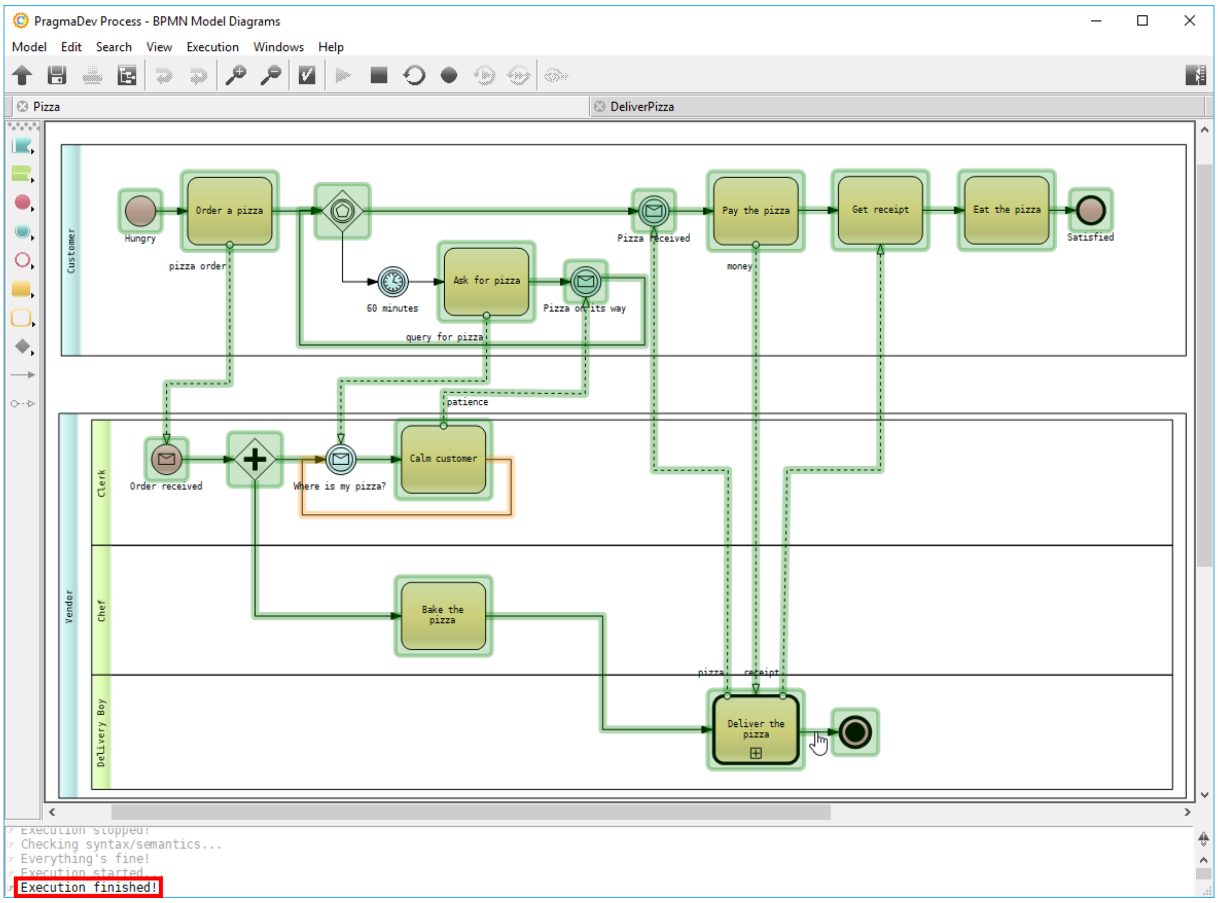
Click the sequence flow and then the message flow for giving the `money` to the `Delivery boy`:



Switch again to `DeliverPizza.bpmn` and click the sequence flow to `Get the money`:

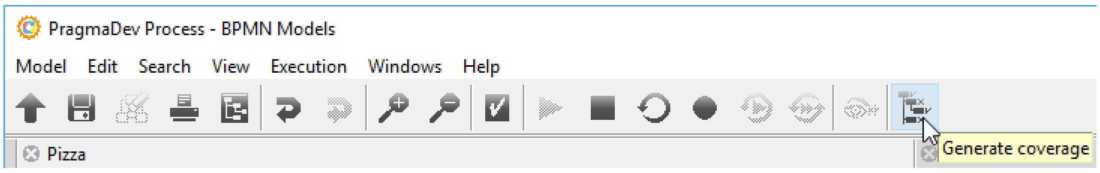


Continue by clicking all possible flows until there is nothing left to do, i.e., execution is finished:

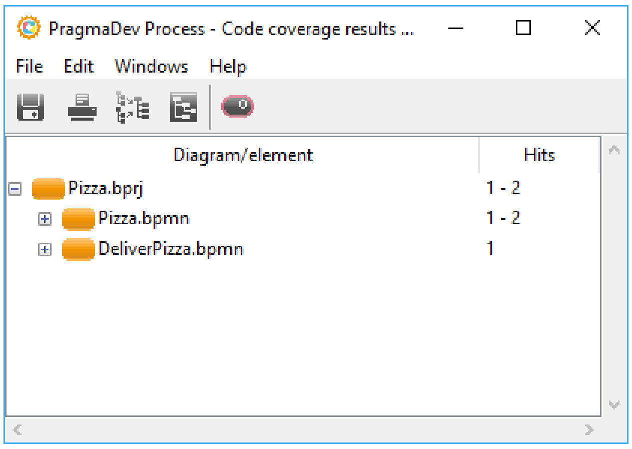


```
Execution stopped!
Checking syntax/semantics...
Everything's fine!
Execution started
Execution finished!
```

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


PragmaDev Process - BPMN Models
Model Edit Search View Execution Windows Help
Pizza
Generate coverage

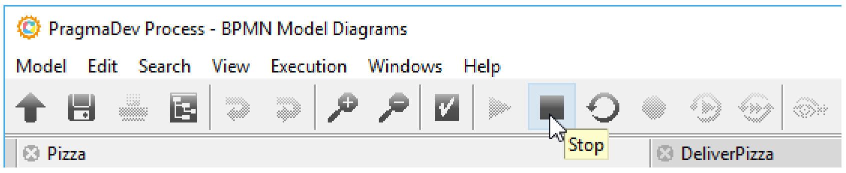
Coverage information will be shown as follows:



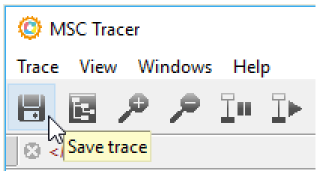
Diagram/element	Hits
Pizza.bprj	1 - 2
Pizza.bpmn	1 - 2
DeliverPizza.bpmn	1

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.

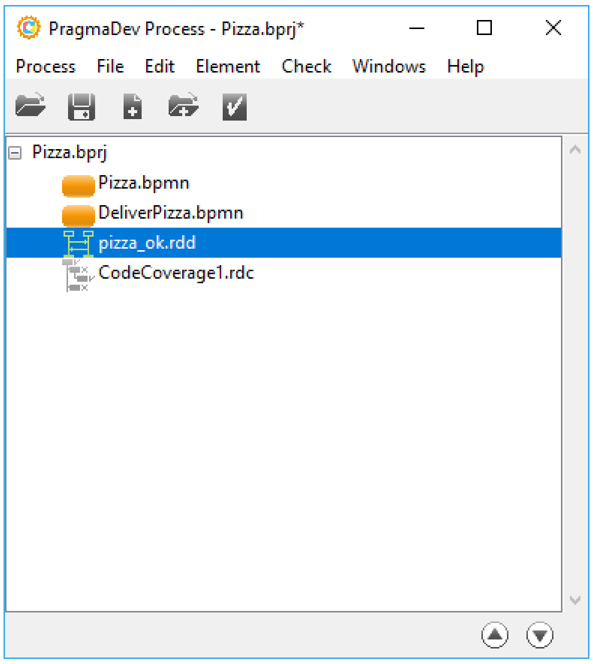
Close the coverage window and stop the execution via the *Stop* button:


PragmaDev Process - BPMN Model Diagrams
Model Edit Search View Execution Windows Help
Stop
Pizza
DeliverPizza

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



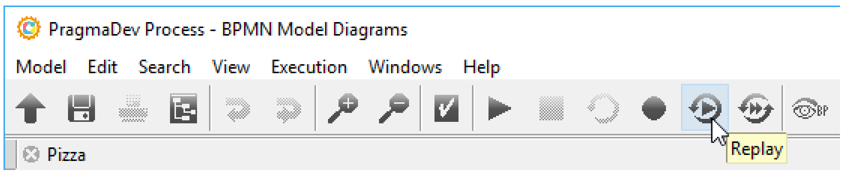
Give it a name (e.g., `pizza_ok`), and it will be added automatically to the project:


PragmaDev Process - Pizza.bprj*
Process File Edit Element Check Windows Help
Pizza.bprj
Pizza.bpmn
DeliverPizza.bpmn
pizza_ok.rdd
CodeCoverage1.rdc

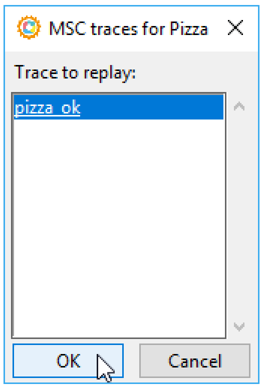
3.3 Automatic execution

3.3.1 Single-trace execution

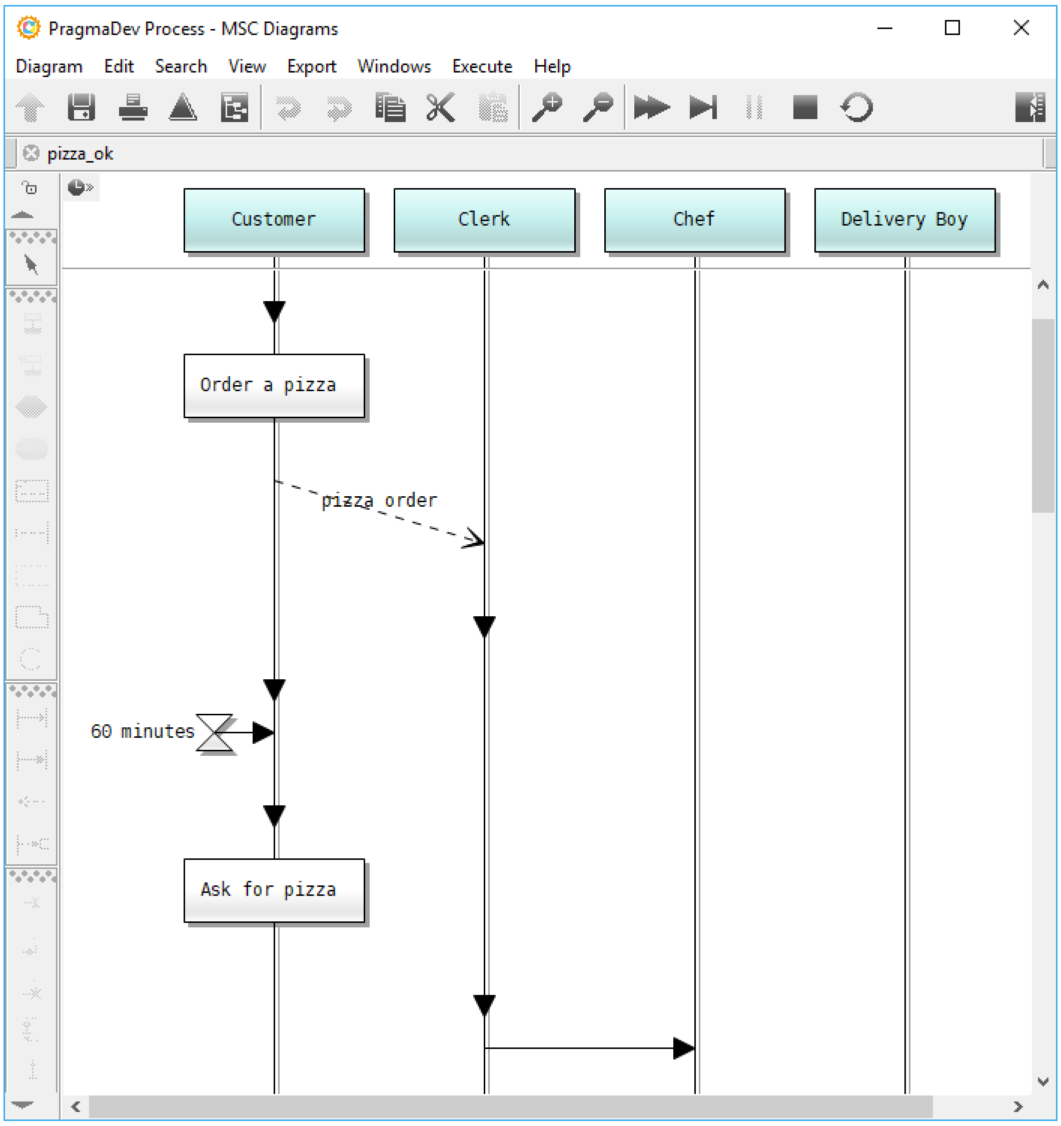
It is possible to replay a scenario. With `Pizza.bpmn` opened in the editor (double-click on it in the project manager), click the *Replay* button:


PragmaDev Process - BPMN Model Diagrams
Model Edit Search View Execution Windows Help
Replay
Pizza

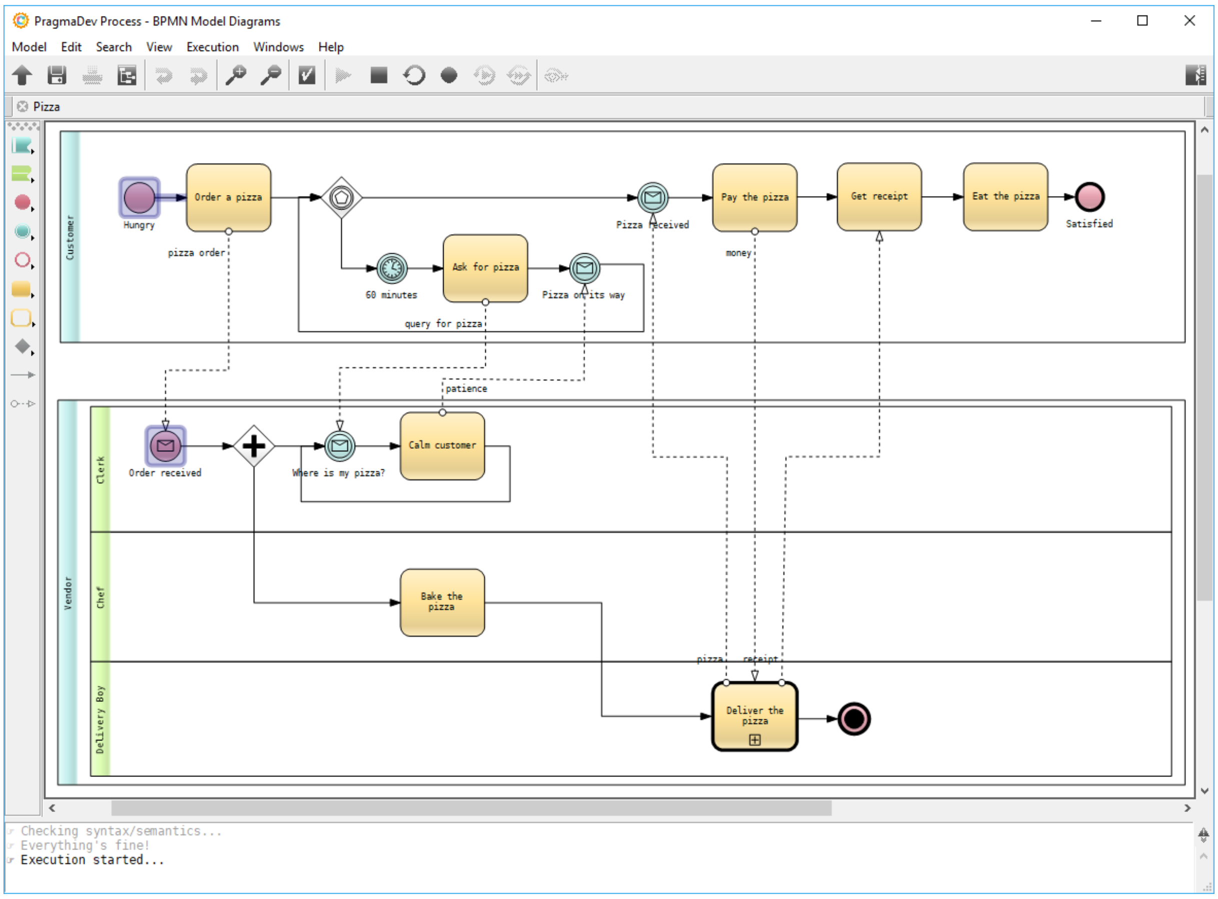
All recordings (MSC traces) found in the project will be listed; select `pizza_ok` and click the *OK* button:



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

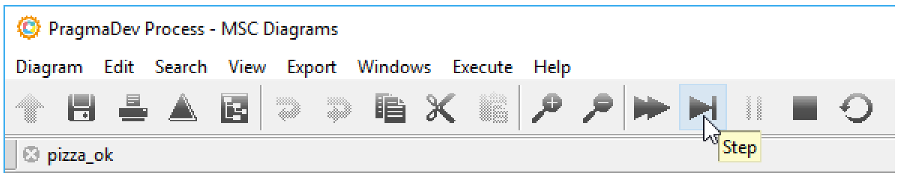


while the execution will start automatically in the BPMN editor:

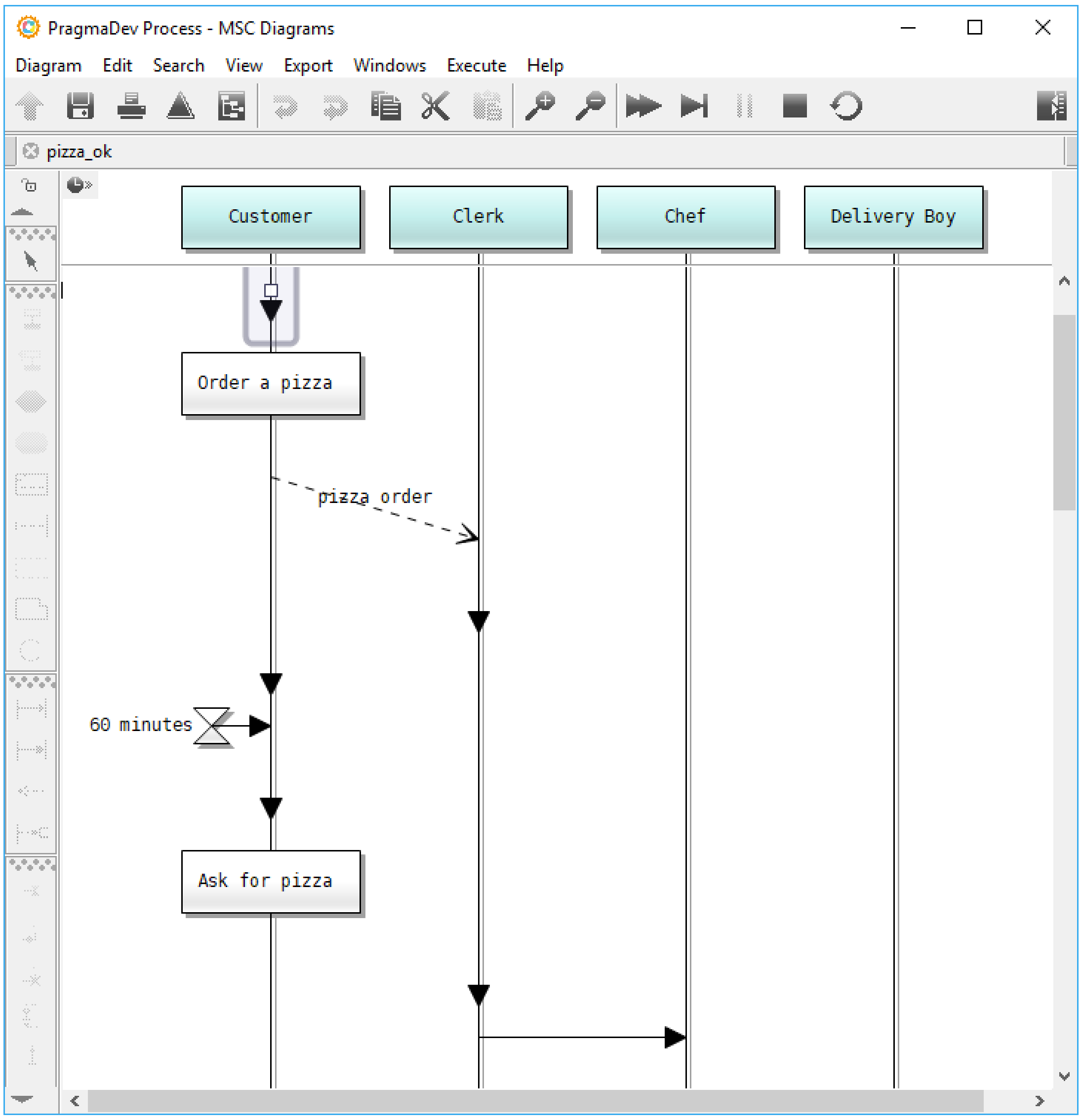


```
Checking syntax/semantics...
Everything's fine!
Execution started...
```

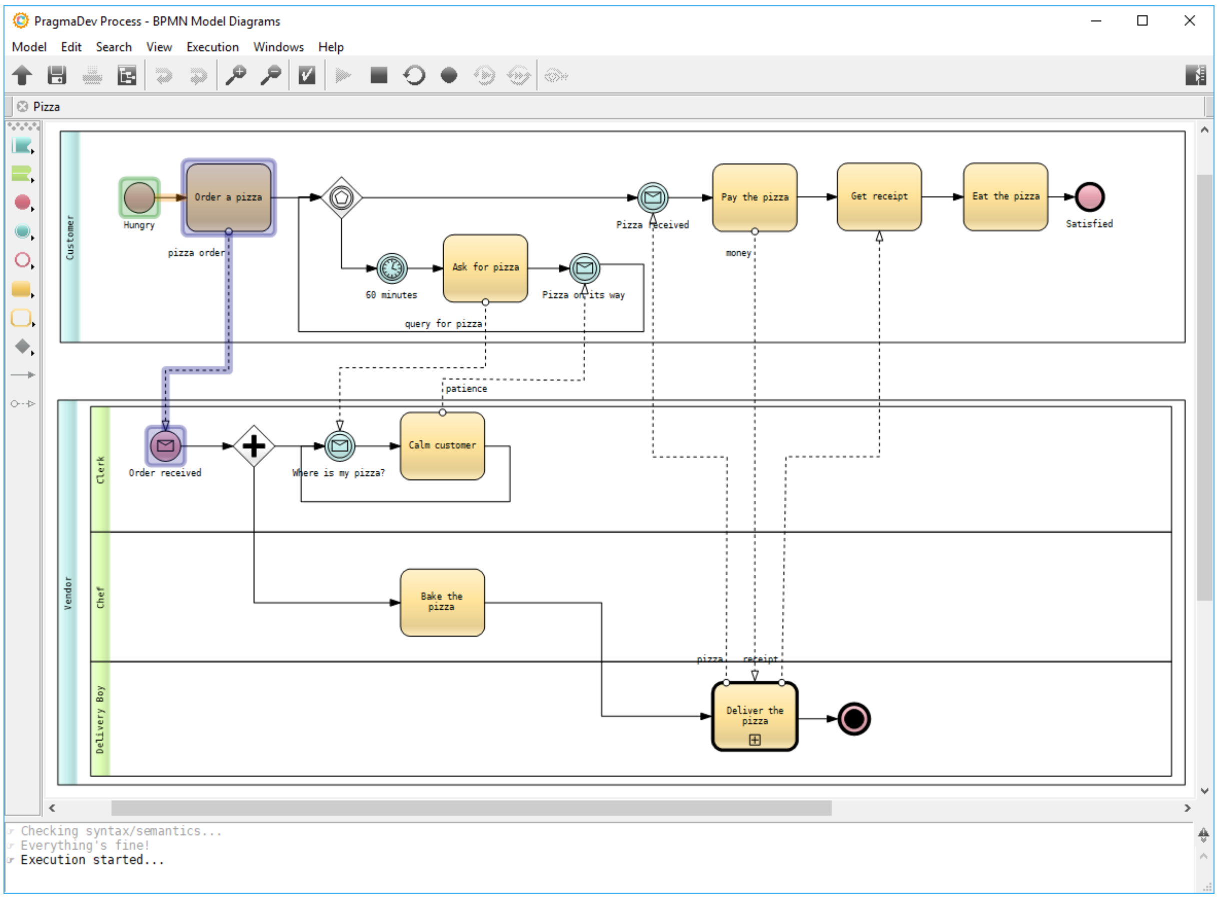
Click the *Step* button in the MSC editor:


PragmaDev Process - MSC Diagrams
Diagram Edit Search View Export Windows Execute Help
Step
pizza_ok

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

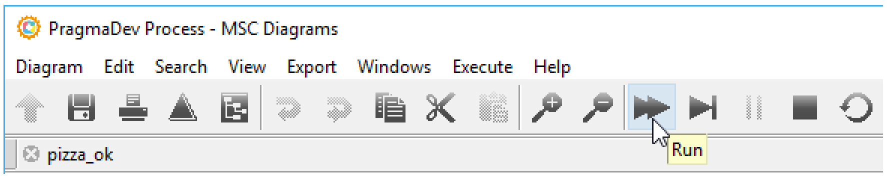

PragmaDev Process - MSC Diagrams
Diagram Edit Search View Export Windows Execute Help
pizza_ok
Customer
Clerk
Chef
Delivery Boy
Order a pizza
pizza_order
60 minutes
Ask for pizza

and executed in the BPMN editor:

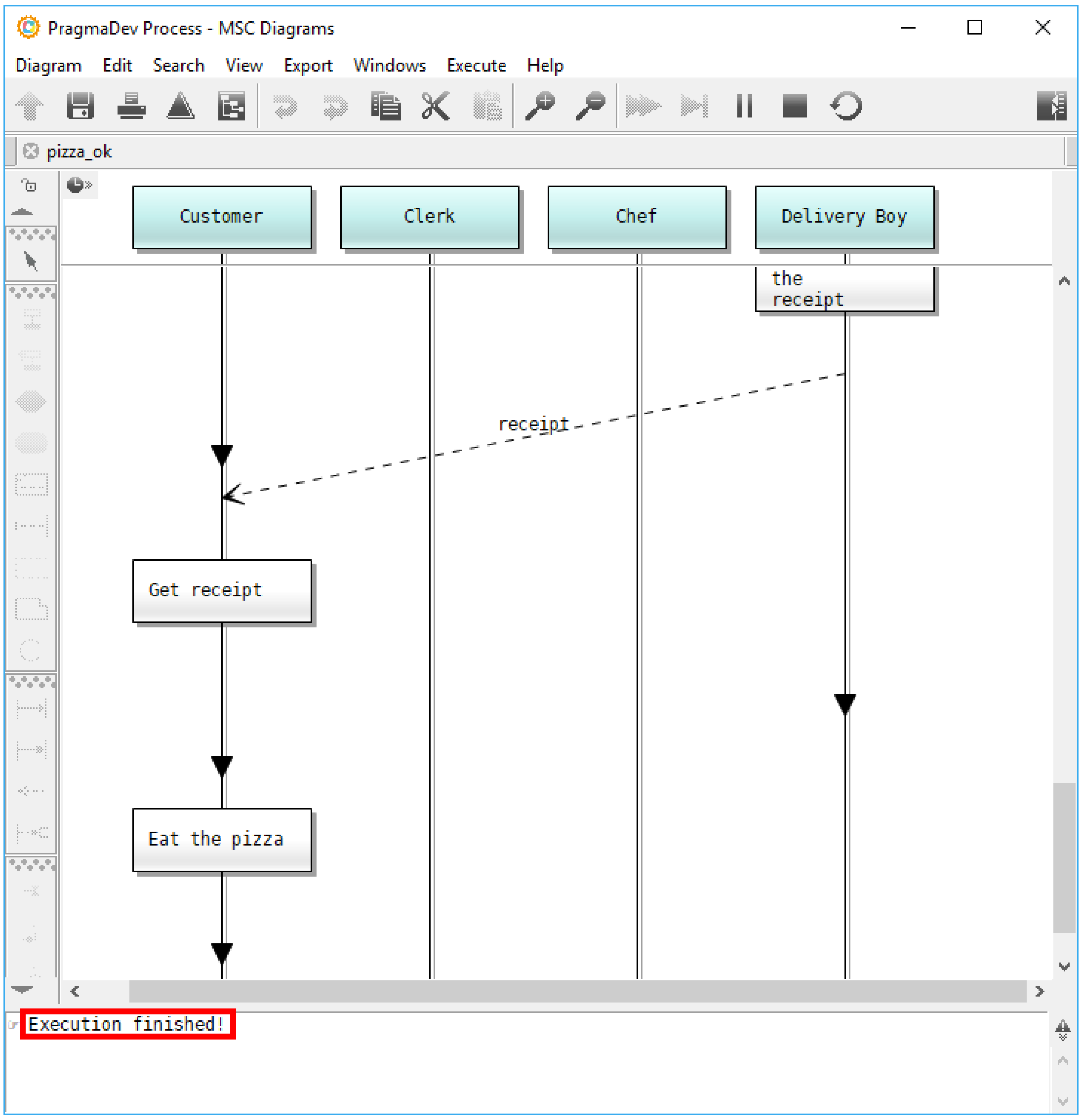


```
Checking syntax/semantics...
Everything's fine!
Execution started...
```

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:

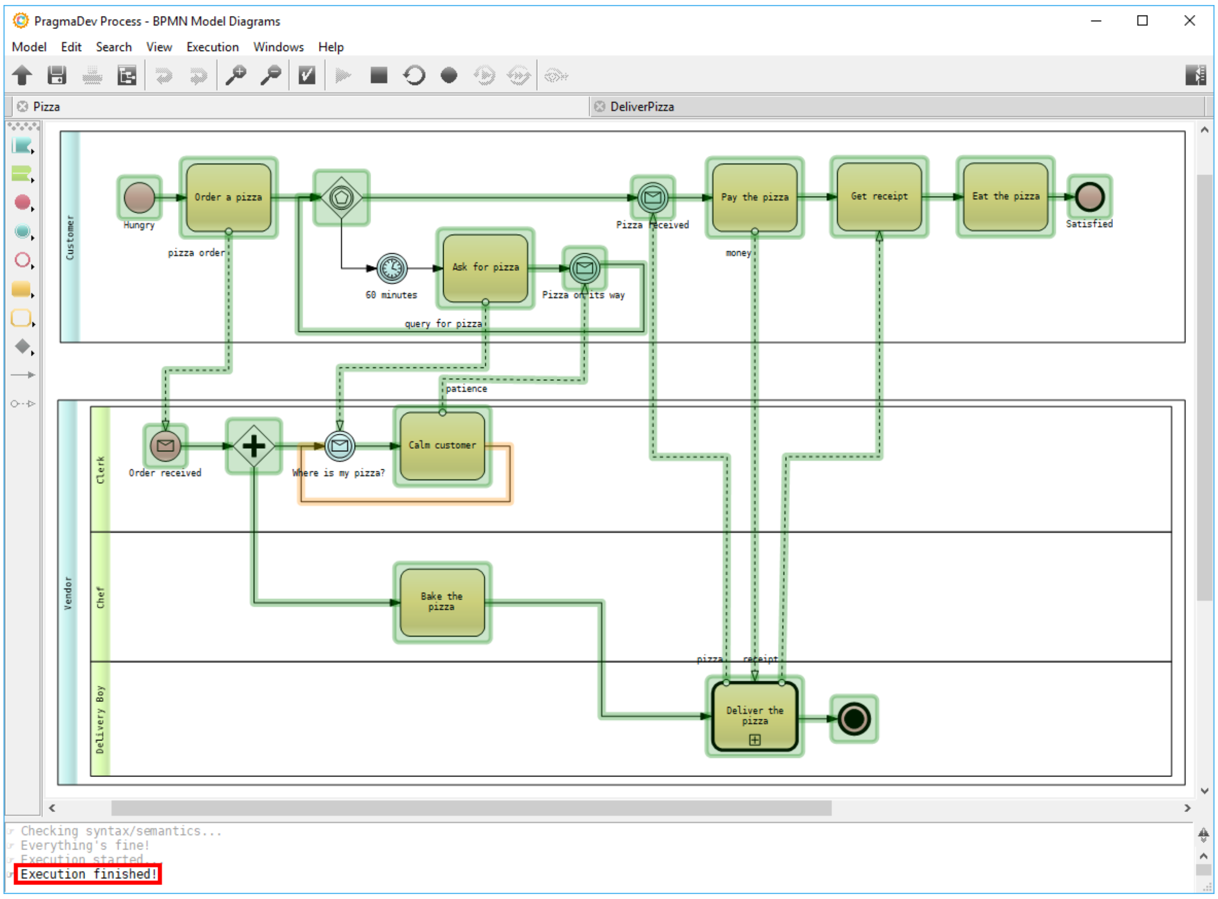

PragmaDev Process - MSC Diagrams
Diagram Edit Search View Export Windows Execute Help
pizza_ok
Run

Execution will continue until no more steps are left:



```
Execution finished!
```

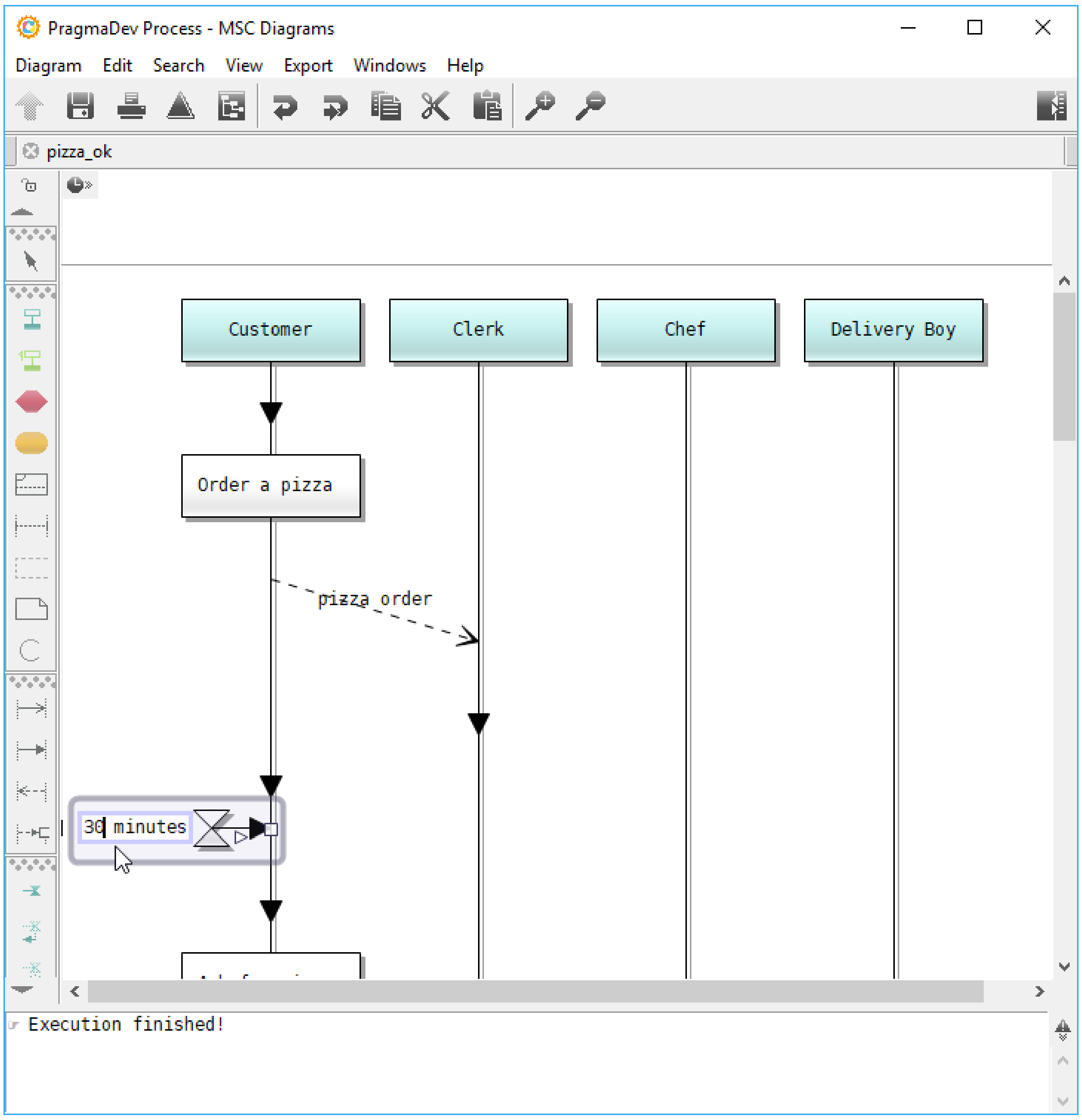
Check also the BPMN editor:



```
Checking syntax/semantics...
Everything's fine!
Execution started...
Execution finished!
```

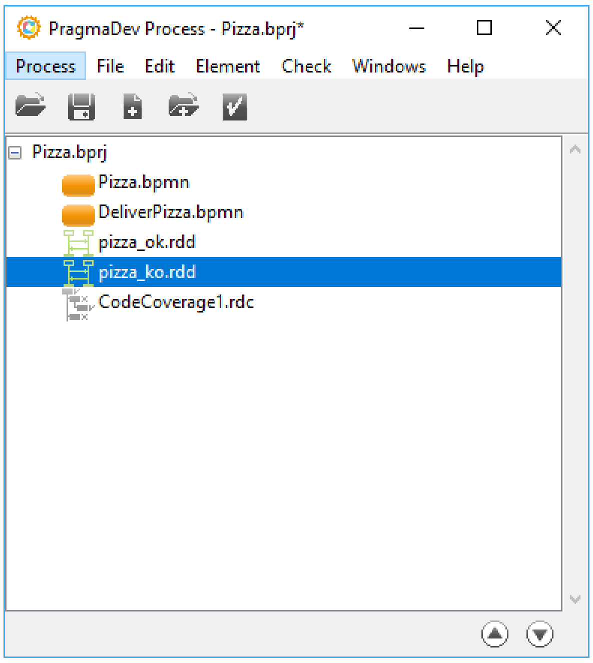
Use the *Stop* button to terminate execution.

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

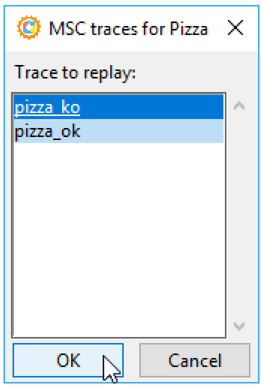

PragmaDev Process - MSC Diagrams
Diagram Edit Search View Export Windows Help
pizza_ok
Customer
Clerk
Chef
Delivery Boy
Order a pizza
pizza_order
30 minutes

```
Execution finished!
```

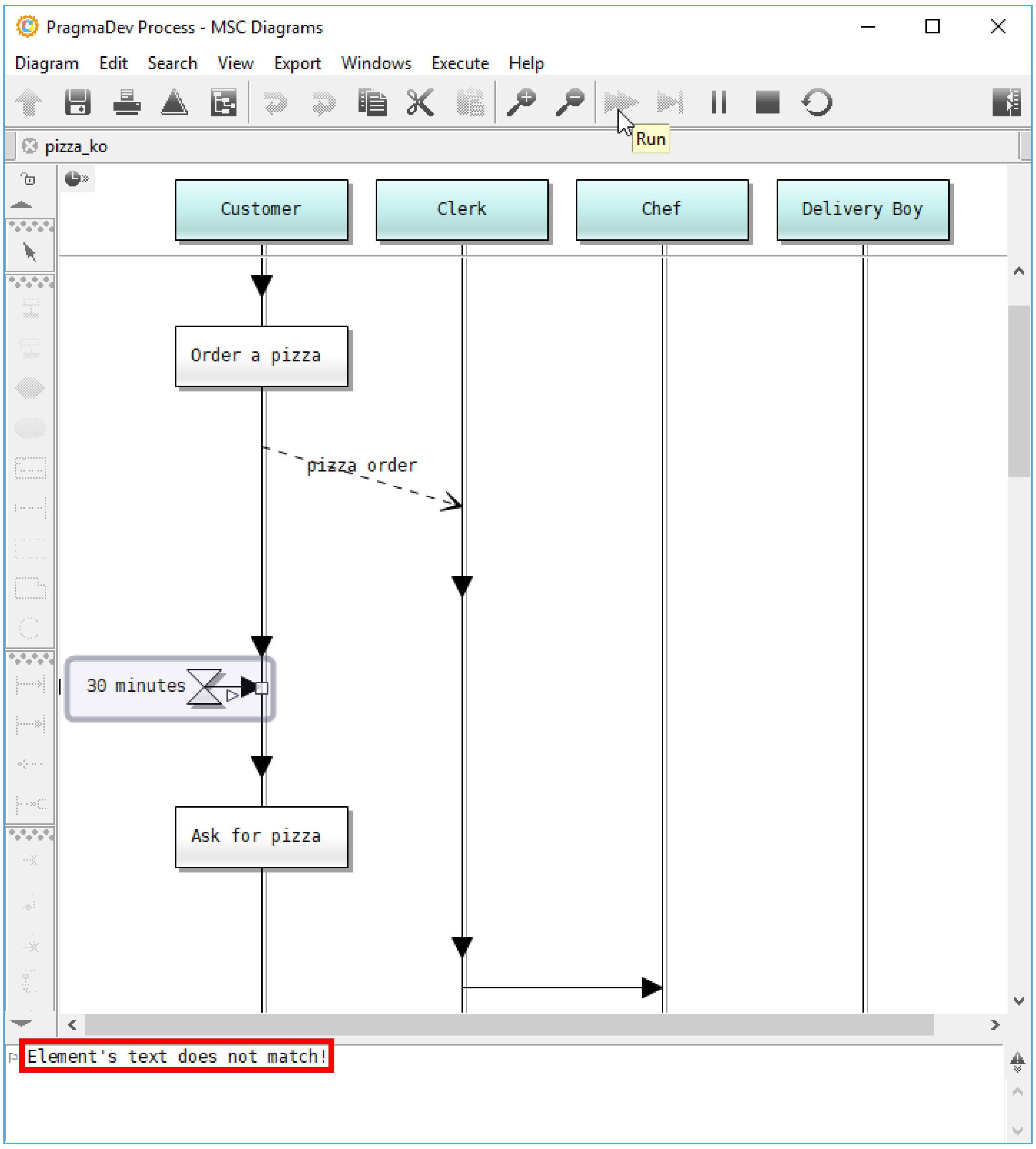
Save the modified MSC trace as `pizza_ko` via the menu *"Diagram / Save As..."*; it will be added to the project:



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



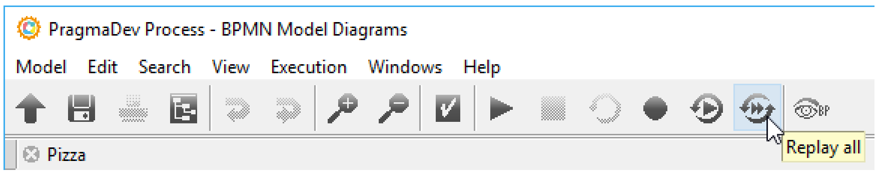
In the MSC editor click the *Run* button. Execution will stop at the timer symbol complaining about "text mismatch":



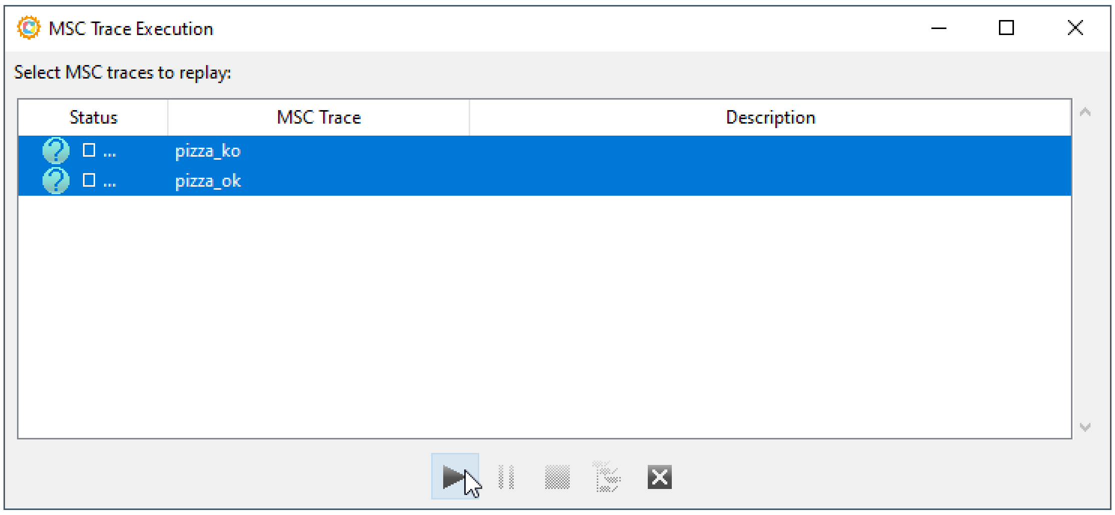
```
Element's text does not match!
```

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:

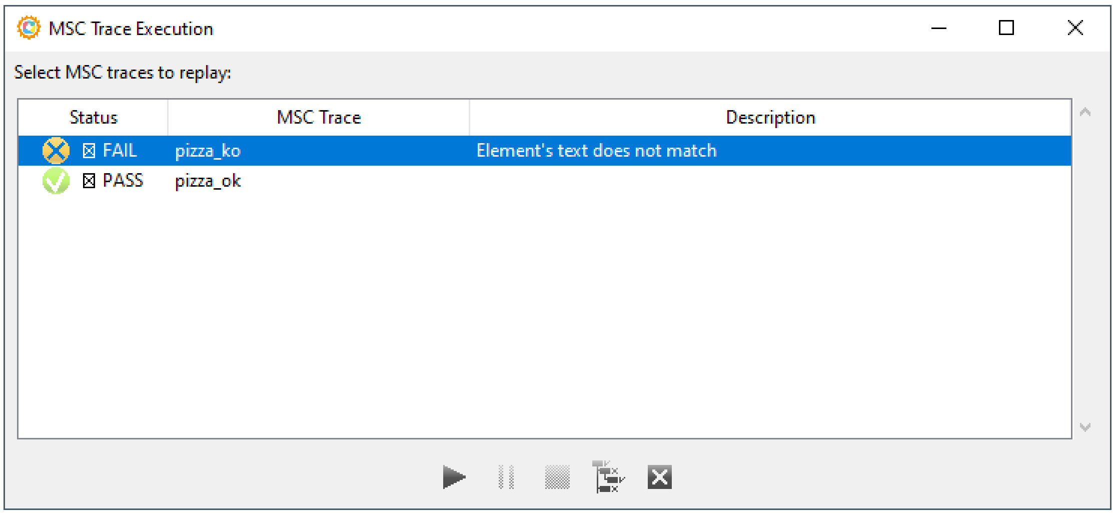

PragmaDev Process - BPMN Model Diagrams
Model Edit Search View Execution Windows Help
Replay all
Pizza

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



Status	MSC Trace	Description
☐ ...	pizza_ko	
☐ ...	pizza_ok	

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

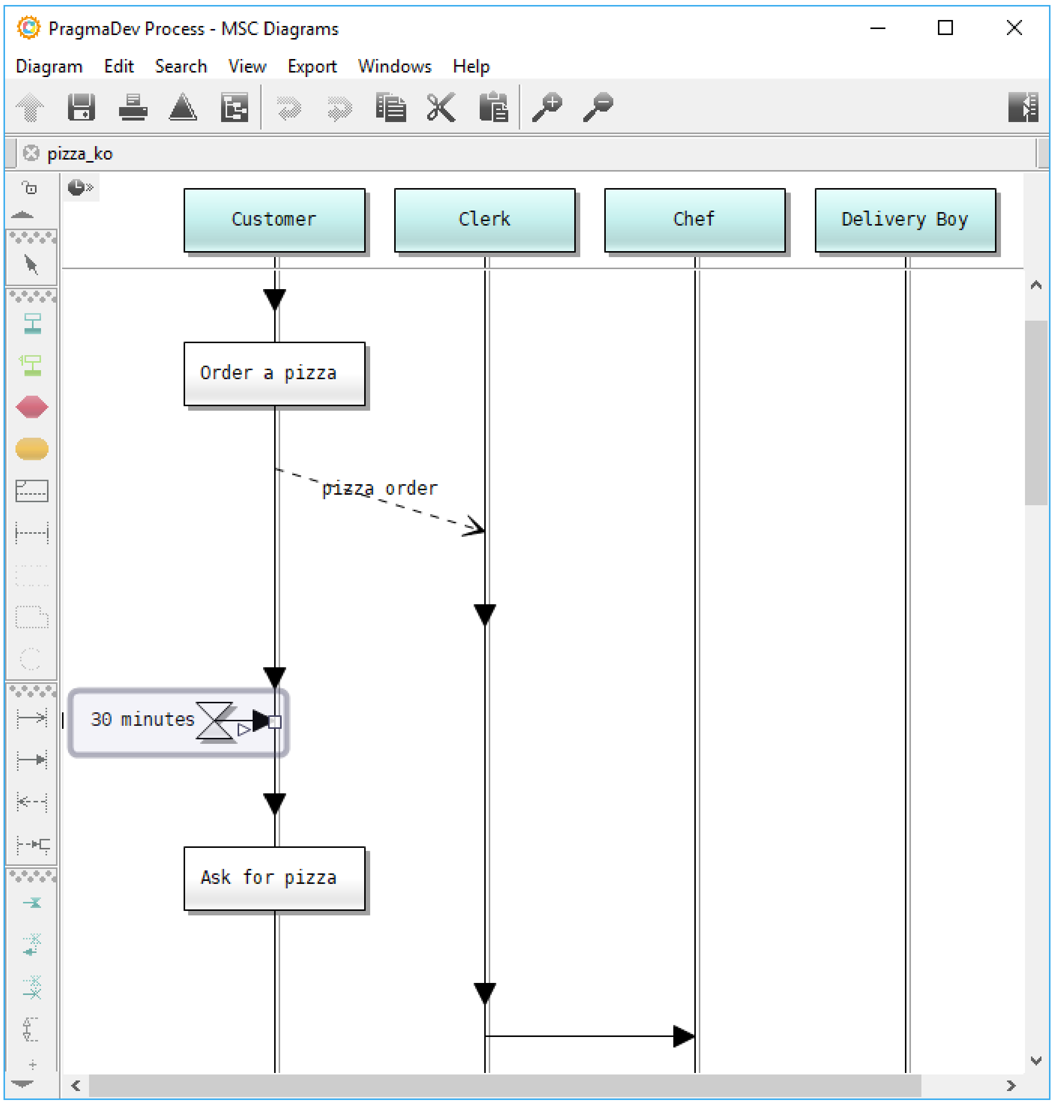


Status	MSC Trace	Description
☒ FAIL	pizza_ko	Element's text does not match
☒ PASS	pizza_ok	

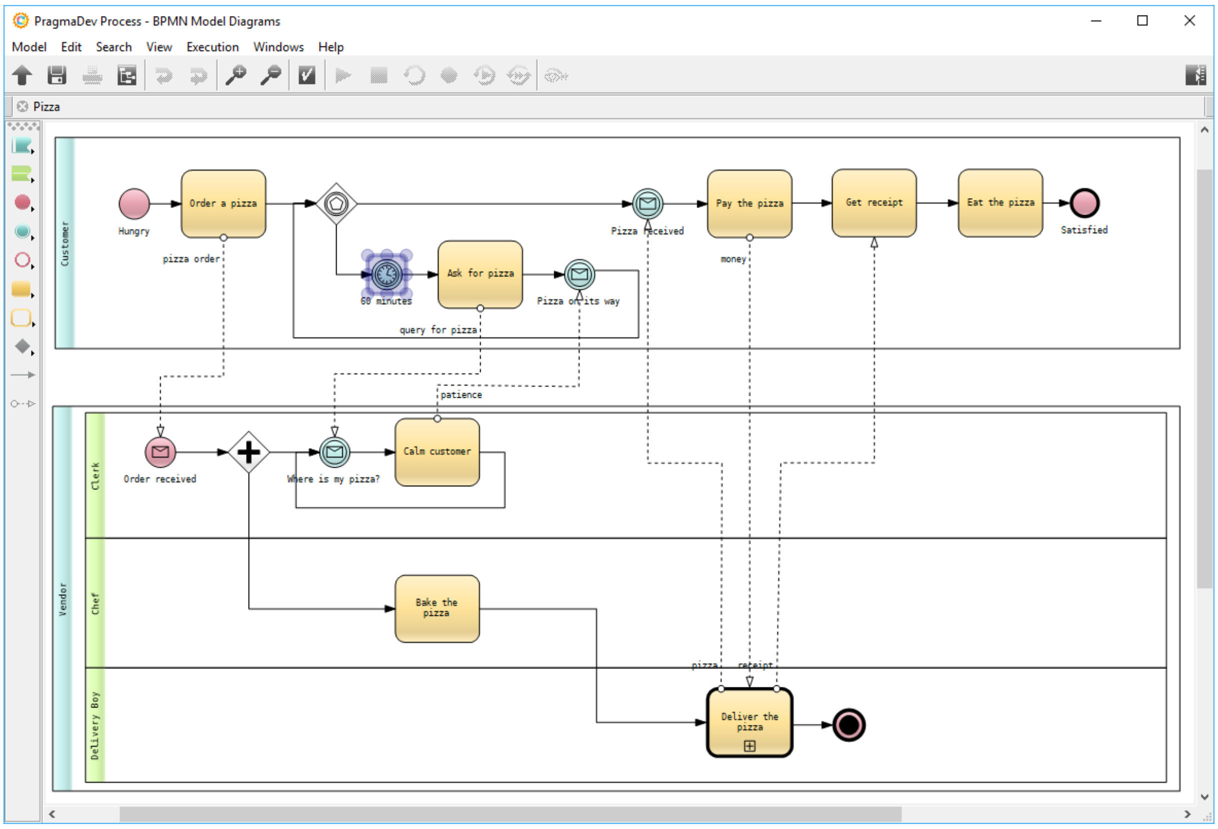
Note that model coverage can be generated via the *Generate coverage for marked traces* button.

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:


PragmaDev Process - BPMN Model Diagrams
Model Edit Search View Execution Windows Help
Pizza
Customer
Hungry
Order a pizza
pizza order
60 minutes
Ask for pizza
query for pizza
Pizza on its way
Pizza received
Pay the pizza
money
Get receipt
Eat the pizza
Satisfied
patience
Vendor
Clerk
Order received
Where is my pizza?
Calm customer
Chef
Bake the pizza
Delivery Boy
pizza
receipt
Deliver the pizza

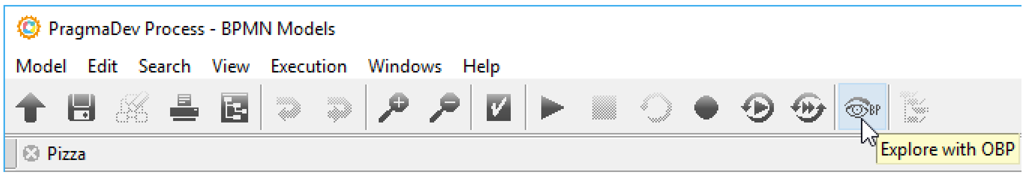
4 Exploration

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

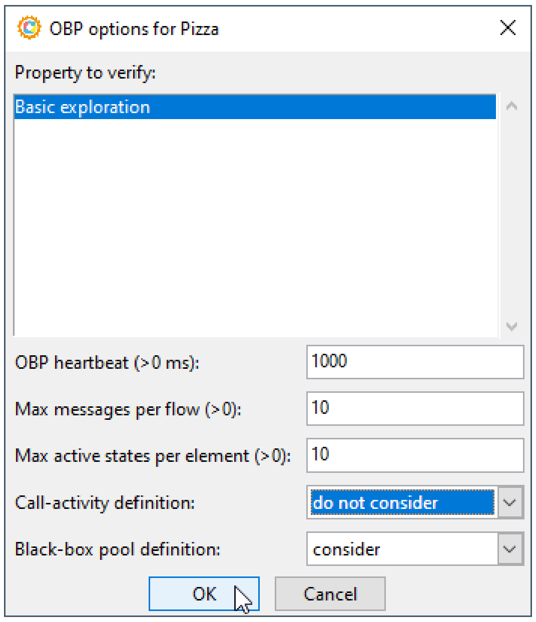
- *Complexity*: through full exploration of the model.
- *Reachability*: identification of unreachable paths.
- *Property*: verification of a property expressed in either PSC (Property Sequence Chart) or GPSL (Generic Property Specification Language).

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:


PragmaDev Process - BPMN Models
Model Edit Search View Execution Windows Help
Pizza
Explore with OBP

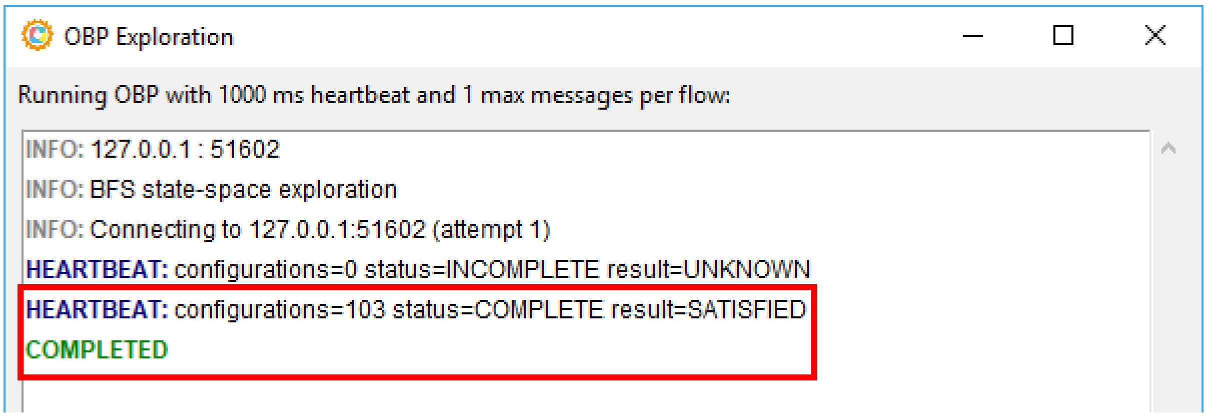
Select *Basic exploration*, set *Call-activity definition* to *do not consider*, and click the *OK* button:


OBP options for Pizza

Property to verify:
Basic exploration

OBP heartbeat (>0 ms): 1000
Max messages per flow (>0): 10
Max active states per element (>0): 10
Call-activity definition: do not consider
Black-box pool definition: consider

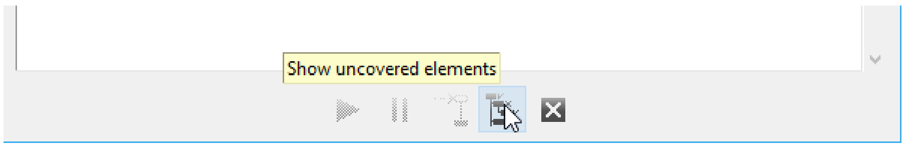
The *do not consider* option will step-over call-activities during exploration.
Observe the increasing number of configurations[1] while waiting for the exploration to complete. The number of configuration can be compared to the model complexity to find out if that result is too high or normal (see User Manual).



```
Running OBP with 1000 ms heartbeat and 1 max messages per flow:
INFO: 127.0.0.1 : 51602
INFO: BFS state-space exploration
INFO: Connecting to 127.0.0.1:51602 (attempt 1)
HEARTBEAT: configurations=0 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=103 status=COMPLETE result=SATISFIED
COMPLETED
```

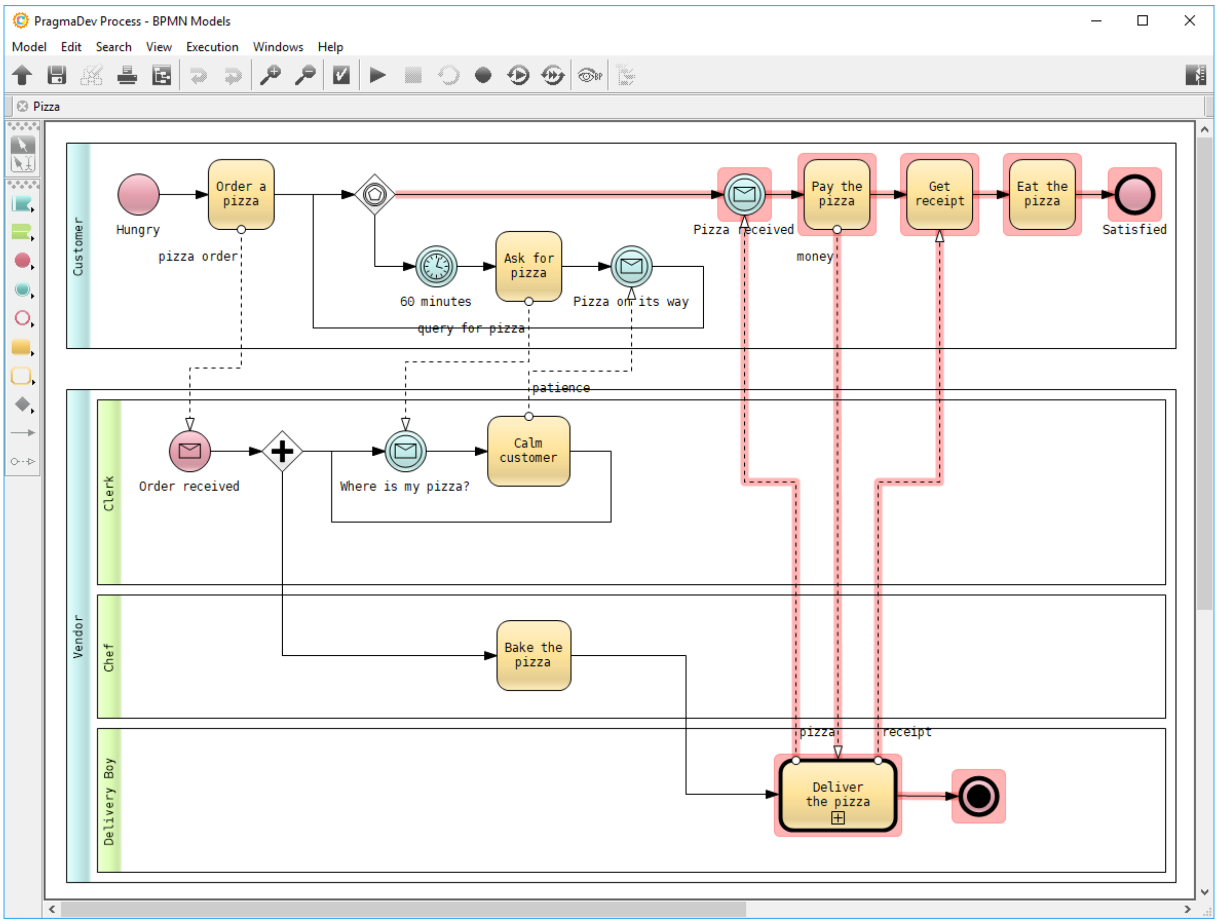
4.2 Reachability

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:



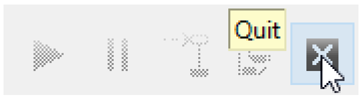
All non covered elements during exploration will be marked in red in the editor:

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



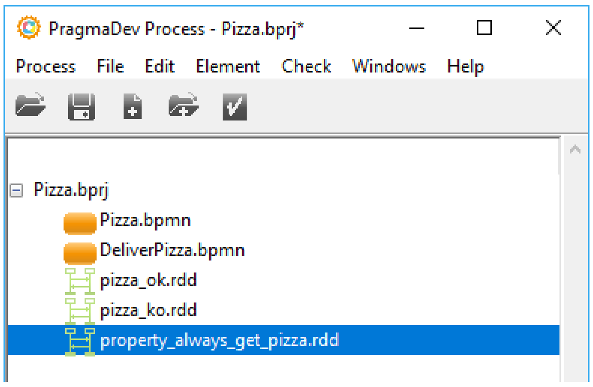
In our example, stepping over the *Deliver the pizza* call-activity resulted in a deadlock during exploration. This is because the behavior of a stepped over activity is to wait for all incoming message flows before executing all outgoing message flows. Outgoing sequence flows can be executed only when 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, and hence the deadlock.

Quit OBP.

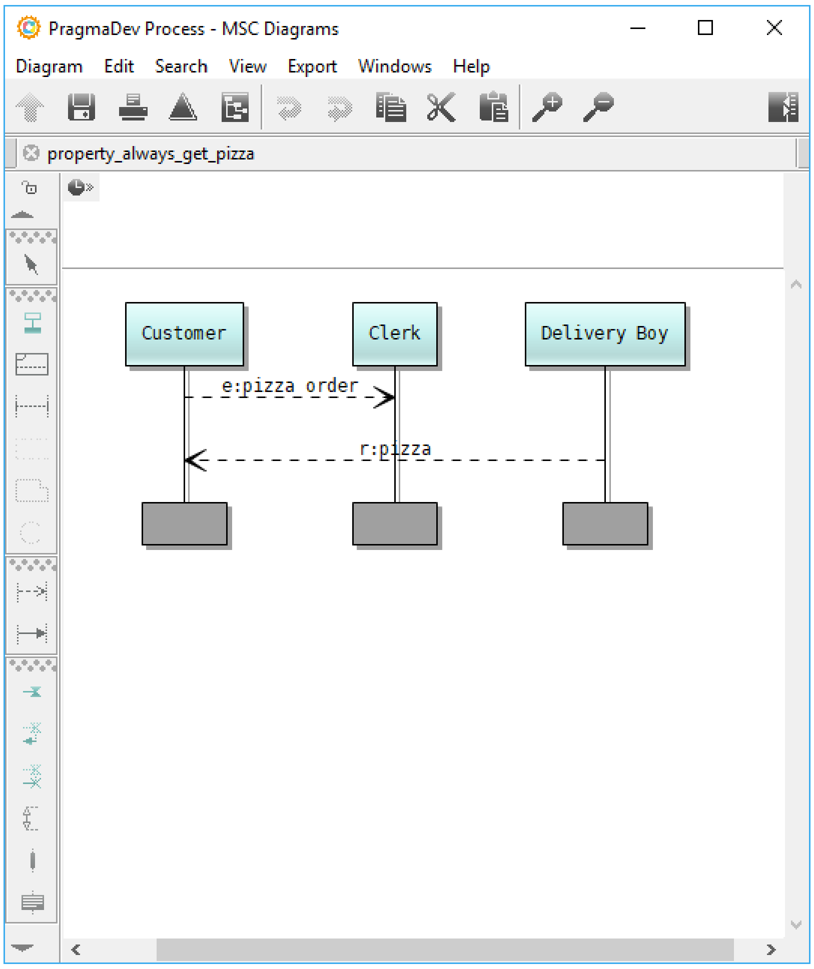


4.3 Property verification

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`:


PragmaDev Process - Pizza.bprj*
Process File Edit Element Check Windows Help
Pizza.bprj
Pizza.bpmn
DeliverPizza.bpmn
pizza_ok.rdd
pizza_ko.rdd
property_always_get_pizza.rdd

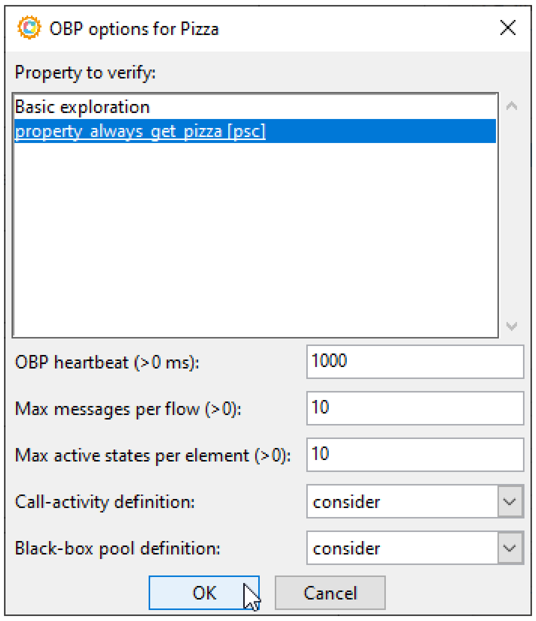
Double click on it to open the MSC/PSC Editor, and draw the following property:


PragmaDev Process - MSC Diagrams
Diagram Edit Search View Export Windows Help
property_always_get_pizza
Customer
Clerk
Delivery Boy
e:pizza_order
r:pizza

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*:



OBP options for Pizza

Property to verify:

```
Basic exploration
property_always_get_pizza [psc]
```

OBP heartbeat (>0 ms): 1000

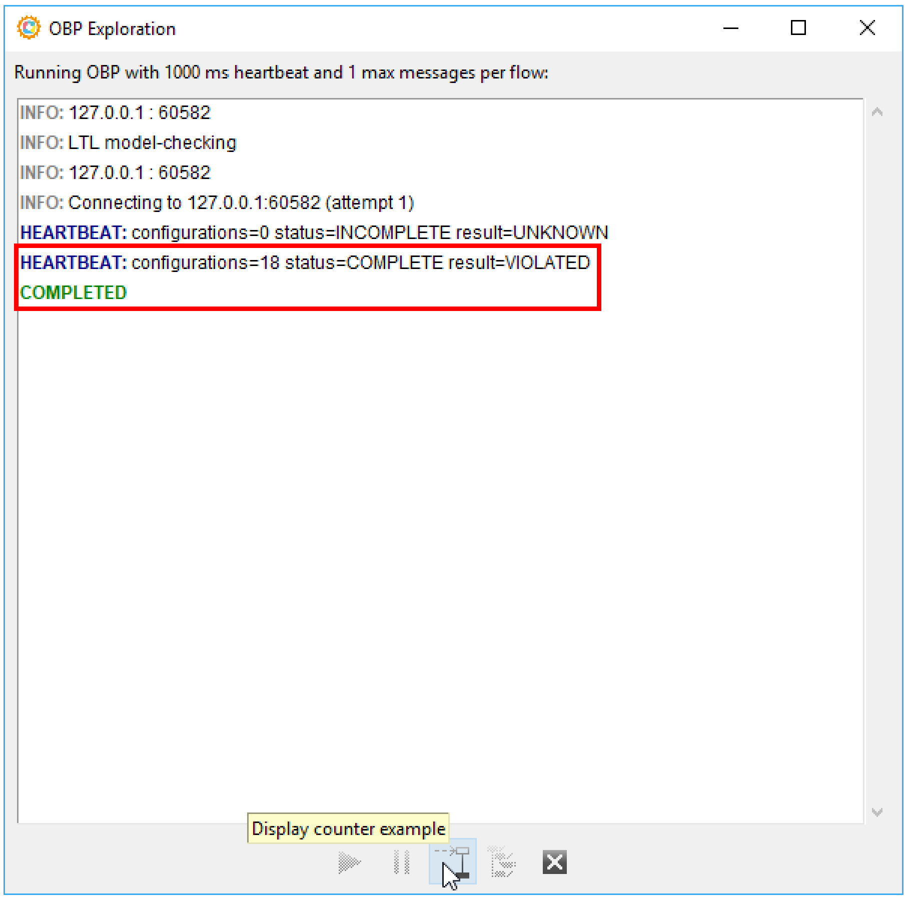
Max messages per flow (>0): 10

Max active states per element (>0): 10

Call-activity definition: consider

Black-box pool definition: consider

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



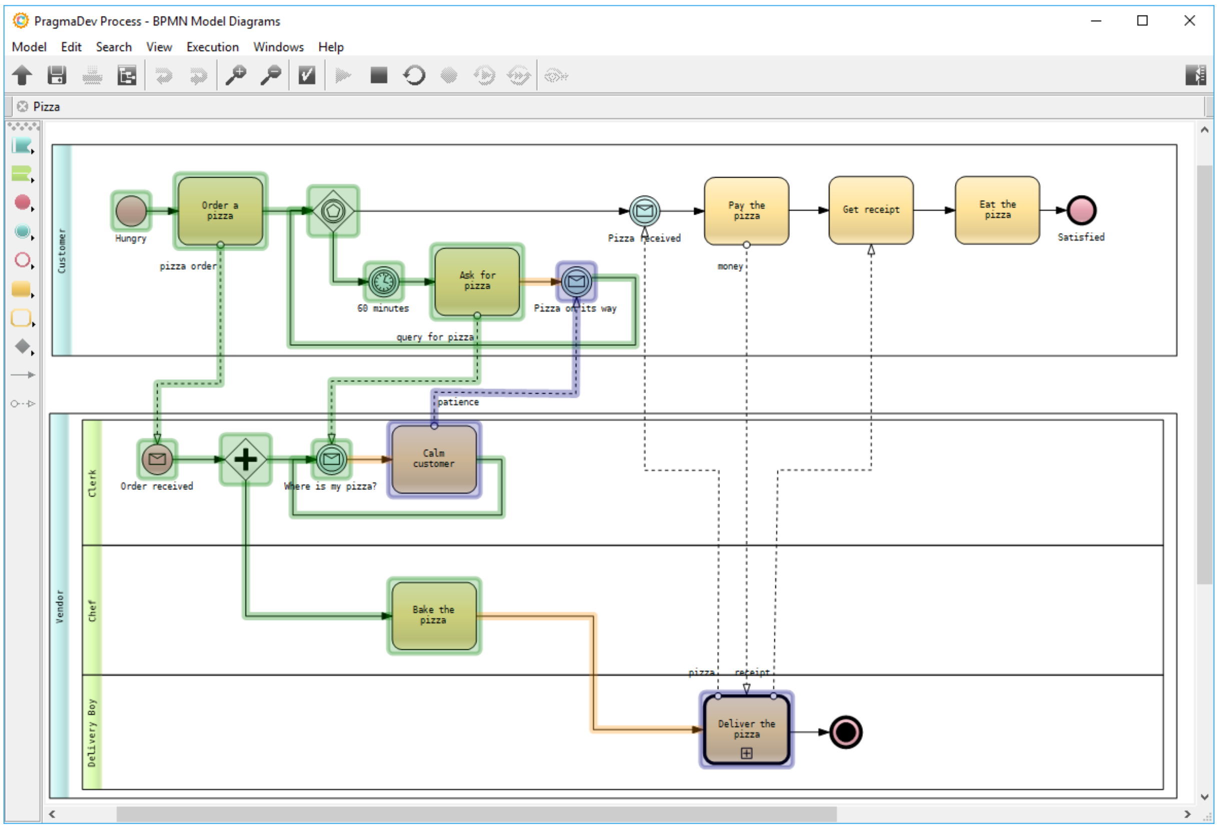
OBP Exploration

Running OBP with 1000 ms heartbeat and 1 max messages per flow:

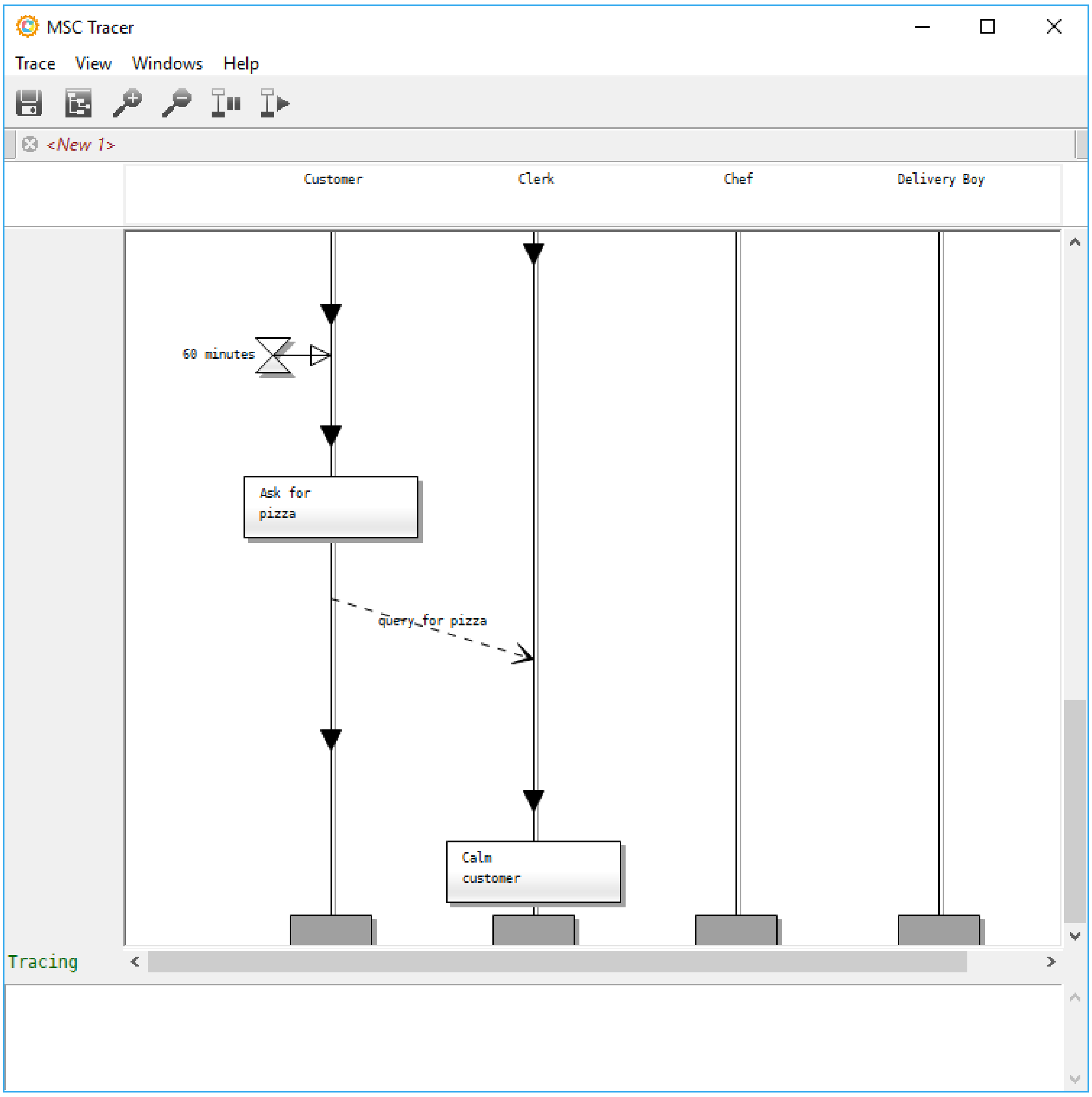
```
INFO: 127.0.0.1 : 60582
INFO: LTL model-checking
INFO: 127.0.0.1 : 60582
INFO: Connecting to 127.0.0.1:60582 (attempt 1)
HEARTBEAT: configurations=0 status=INCOMPLETE result=UNKNOWN
HEARTBEAT: configurations=18 status=COMPLETE result=VIOLATED
COMPLETED
```

[2]The 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 BMPN models.