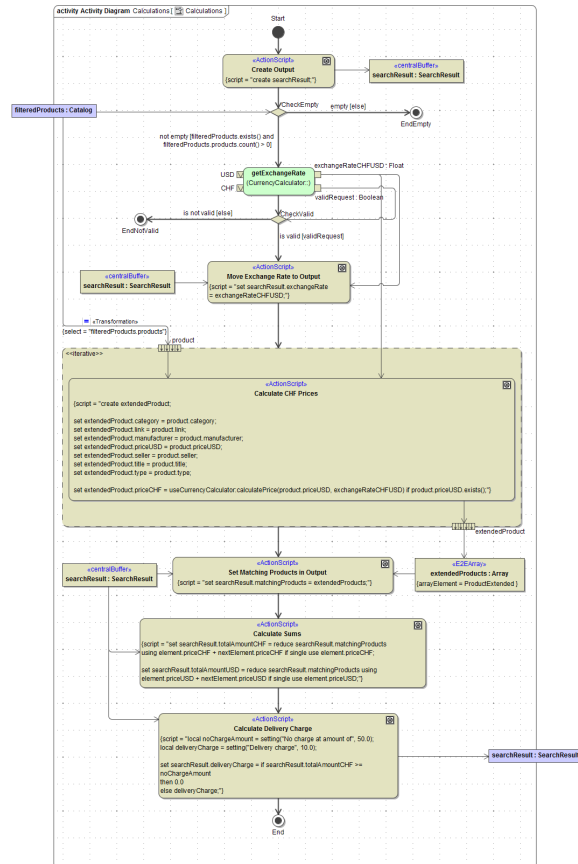


# Implementing the Currency Conversion

The next step is to implement the behavior of the third action **Calculate Total and Currencies**. You will start with the implementation of the currency conversion.

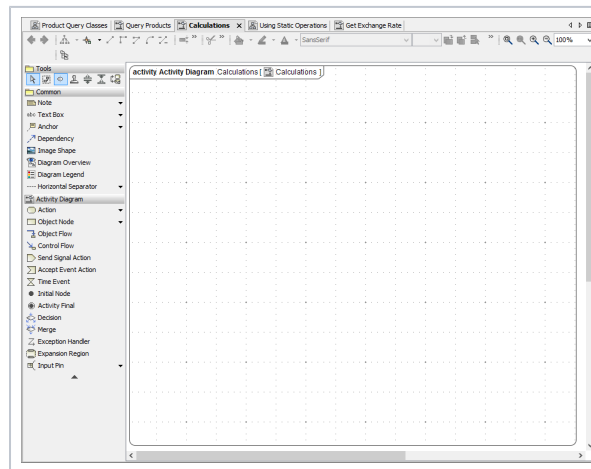
The activity diagram **Calculations** will contain all activities to perform a currency calculation for each product record found in the filtered XML data and to calculate a total of all product prices as shown in the picture below.



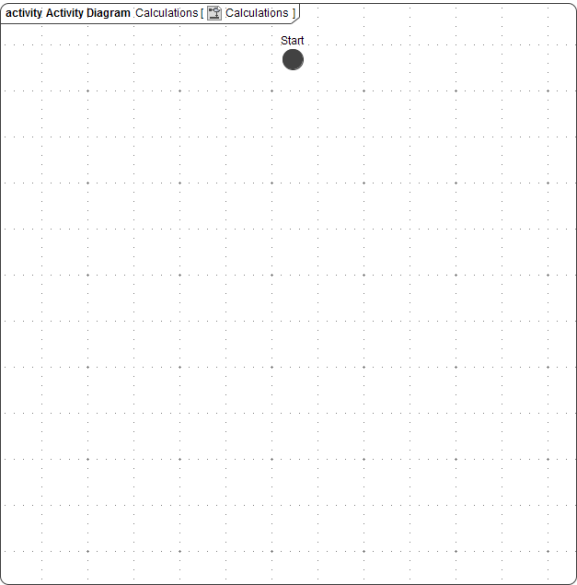
Implementing Additional Calculations

On this Page:

- Calling the SOAP Service
- Checking the Search Result
- Converting the Price from USD to CHF

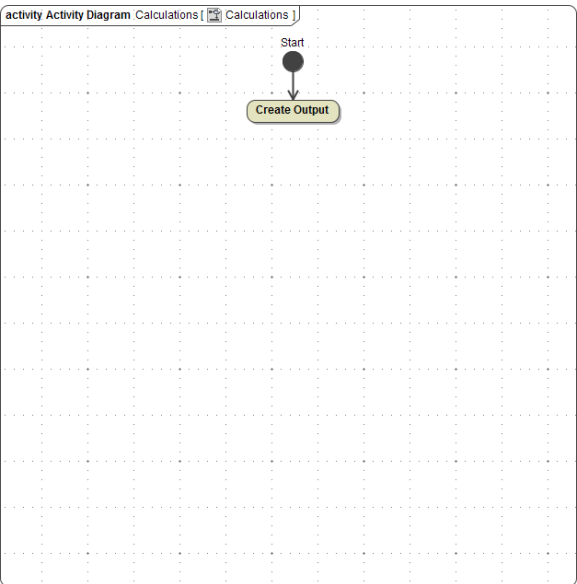


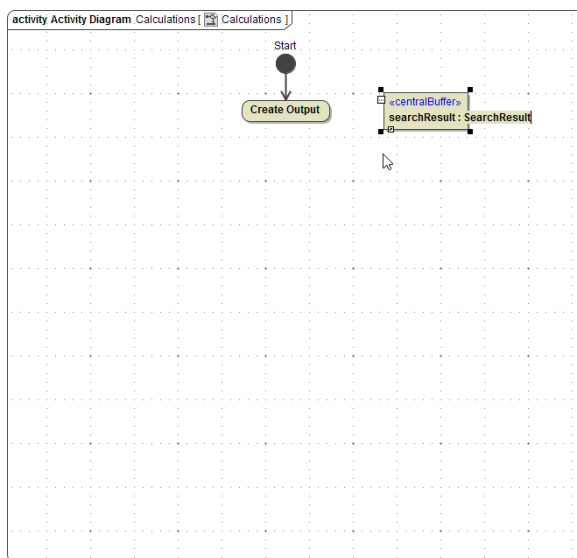
Open the activity diagram **Calculations** in the diagram pane.

<div>activity Activity Diagram Calculations [ Calculations ]</div>  <p>The diagram shows a single 'Start' node (a black circle) at the top left of a grid. The label 'Start' is positioned above the node.</p>	<p>Draw an initial node and assign the name <b>Start</b>.</p>
---	---

## Calling the SOAP Service

Within this activity, you are going to iterate over all **filteredProducts** collected in the prior activities and process each product. Before implementing this, you have to check whether any matching products were found at all within the filtering activities. In the case that no product from file **catalog.xml** matched the entered keywords, the activity **Calculations** has to return an empty result and no further calculations will be done.

<div>activity Activity Diagram Calculations [ Calculations ]</div>  <p>The diagram shows a 'Start' node (a black circle) at the top left. Below it, connected by a downward arrow, is a yellow rounded rectangle labeled 'Create Output'.</p>	<p>First, create an action below the activity start and assign the name <b>Create Output</b>.</p> <p>Connect the action node with the initial node.</p>
---	---

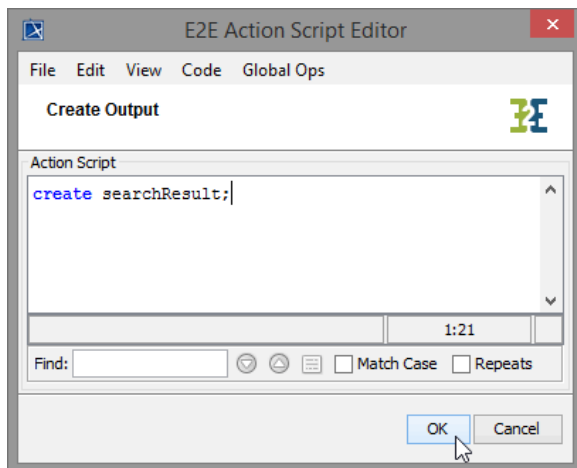


The output created will be of type **SearchResult**. Draw a central buffer node next to action node **Create Output**.

Click into the buffer node and type `searchResult : SearchResult`.

This assigns the name **searchResult** and the type **SearchResult** to the buffer node. The **searchResult** will be buffered until it is complemented within this activity.

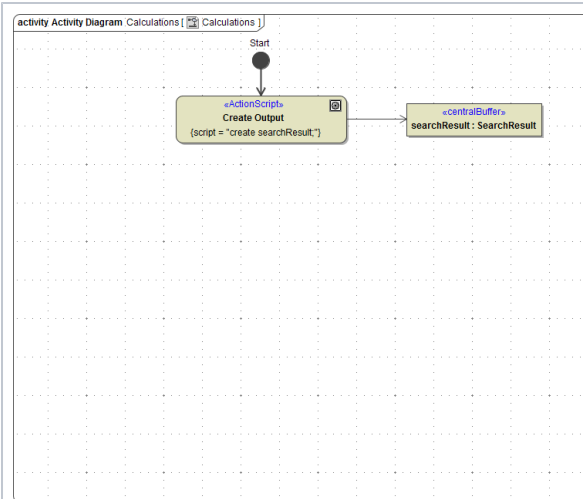
Finish the definition by pressing **Enter**.



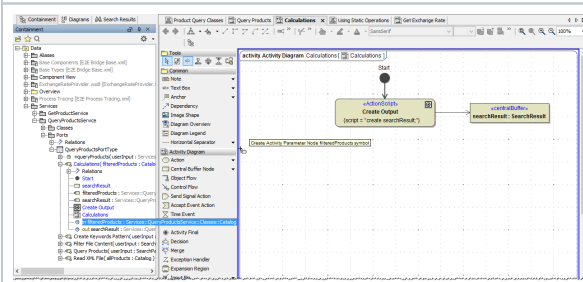
Select the action node **Create Output** again and press **Ctrl - Enter** to open the Action Script Editor.

Create the object **searchResult** as shown in the screenshot on the left.

Click **OK** or press **Ctrl - Enter** again to close the Action Script Editor.

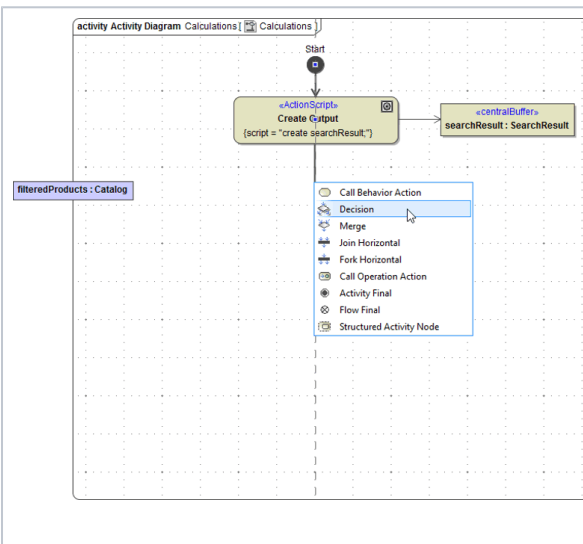


Draw an object flow from **Create Output** to the buffer node **searchResult**.



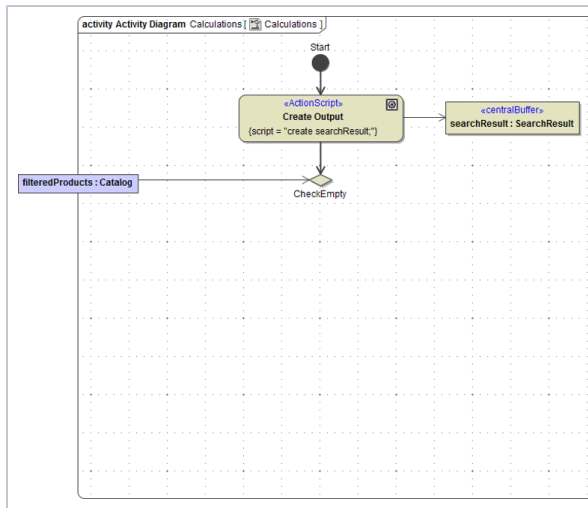
Drag and drop the parameter **filter redProducts** on the diagram border of **Calculations**.

You are going to check now, whether there is any input data to process.



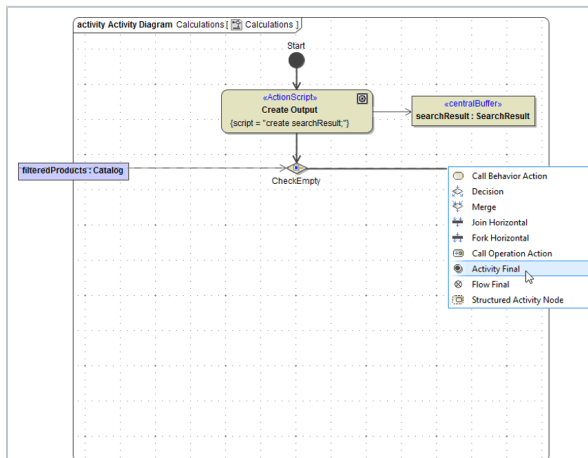
Select the action node **Create Output** and choose the control flow icon from the smart manipulation bar.

Instead of left-clicking to position the action node, click the right mouse button. Select **Decision** from the context menu to insert a decision node.

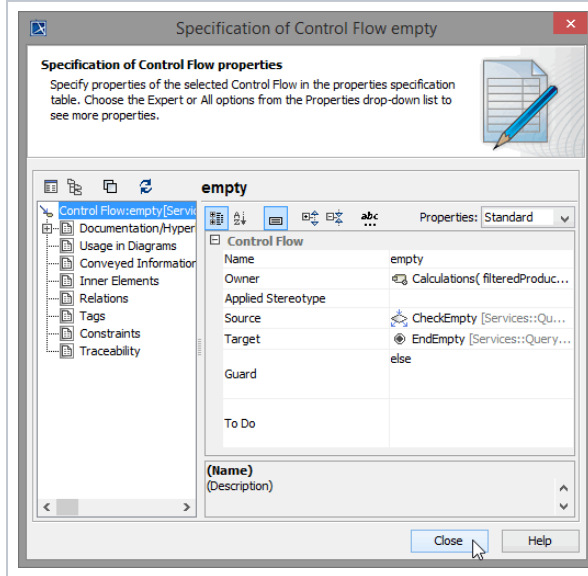


Assign the name **CheckEmpty** and draw an object flow from the input parameter **filteredProducts** to **CheckEmpty**.

If the input parameter **filteredProducts** does not contain valid product information, the calculations will not be proceeded, but the control flow will directly come to an end.



Choose **Control Flow** from the decision nodes smart manipulation toolbar. Below the buffer node insert an activity final node (right-click) and assign the name **EndEmpty**.

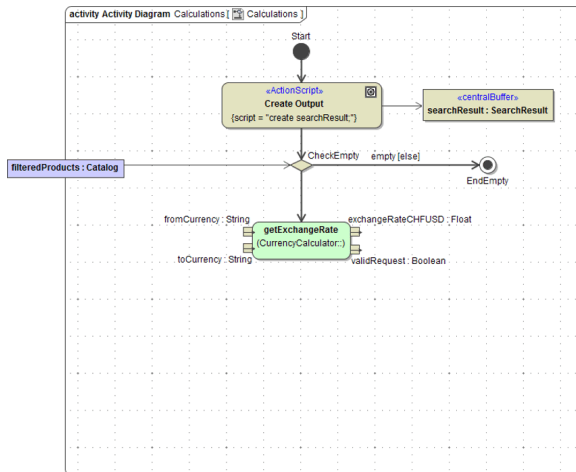
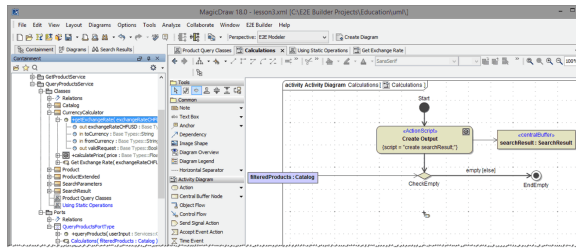


Double-click the control flow coming from the decision **CheckEmpty** and enter the name **empty** in the **Specification** dialog.

Insert **else** as a guard expression.

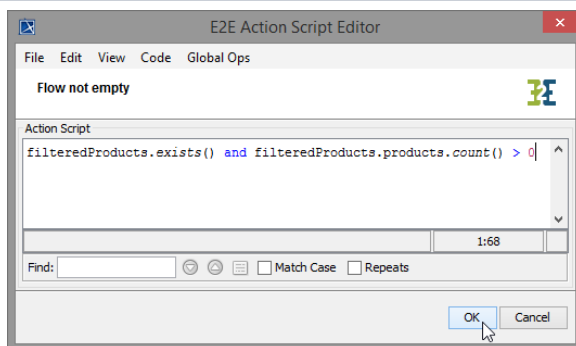
The effective guard expression will be added to the continuing control flow. Keep this in mind, as first you are now going to reuse the previously copied operation from class **CurrencyCalculator**.

Drag and drop the operation **getExchangeRate** from class **CurrencyCalculator** onto the diagram pane.



This will draw the action node together with all necessary input and output pins.

Delete the unnecessary target pin, rearrange the input and output pins and draw a control flow from **CheckEmpty** to **getExchangeRate**.



Now remember to enter the guard expression on this control flow. Select it and start typing the name **not empty**.

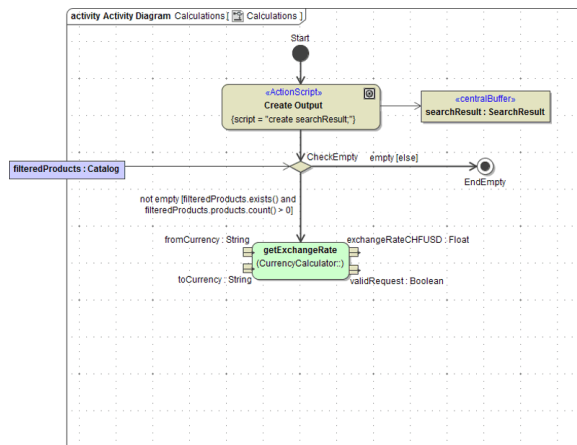
As you are going to enter a complex guard expression, press **Ctrl - Enter** to use the support of the Action Script Editor and enter the following expression:

```
filteredProducts.exists() and
filteredProducts.products.count() > 0
```

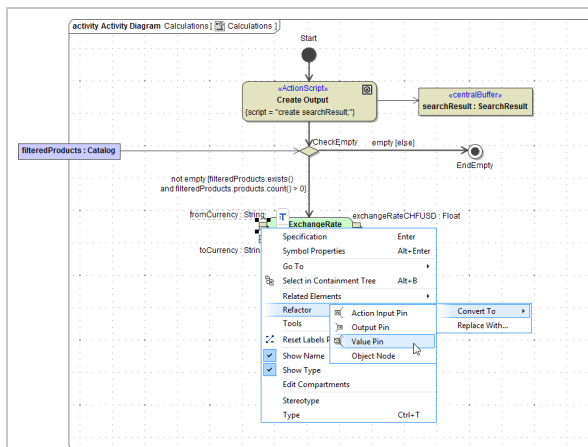
If the parameter **filteredProducts** is existing and contains a number of products greater than 0, this path of the decision will be followed.

Remember, that the **else** path has been defined as to end in an activity final.

Your activity diagram now should look as follows.

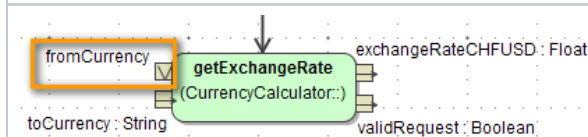


All product prices are in currency USD and are to be converted into CHF. Because of that fact, the exchange rate only has to be get once and you are going to replace the input pins by value pins. A value pin is an input pin that provides a value to an action that does not come from an incoming object flow edge.

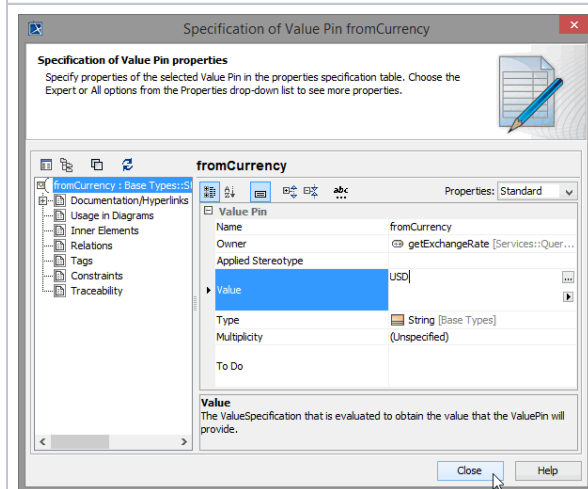


Right-click the input pin **fromCurrency**.

Choose **Refactor > Convert To > Value Pin** from the context menu.

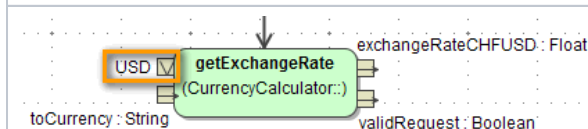


Note that the symbol of the pin **fromCurrency** on the diagram pane has changed.



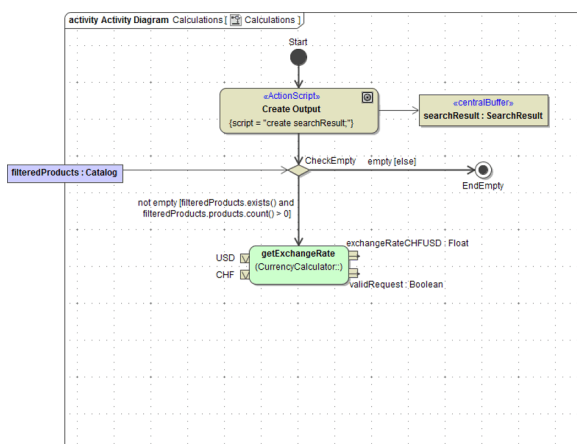
Open the **Specification** dialog of the value pin.


In the **Value** field enter **USD** and click **Close**.



Now the value is displayed on the diagram pane instead of the pin's name.

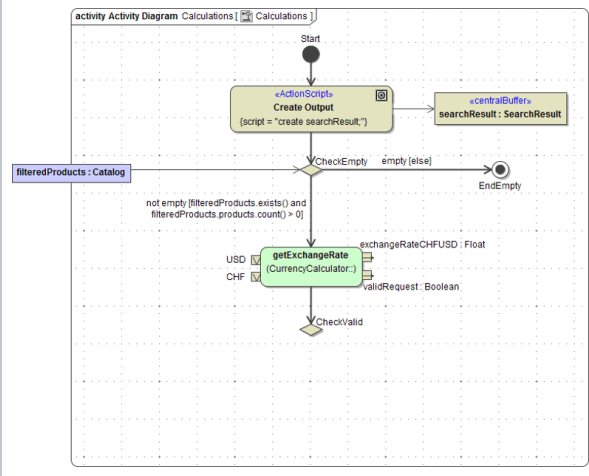
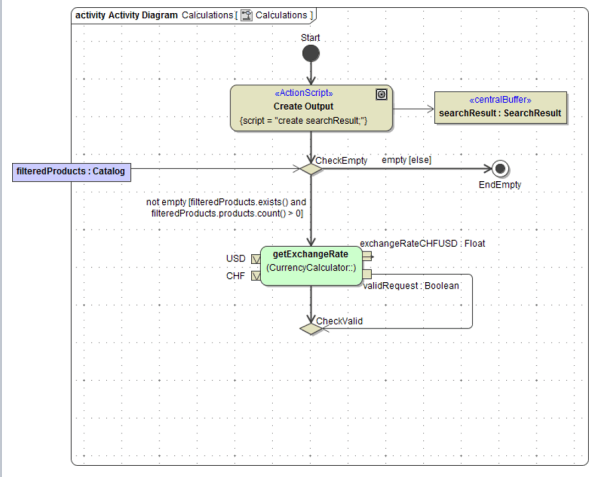
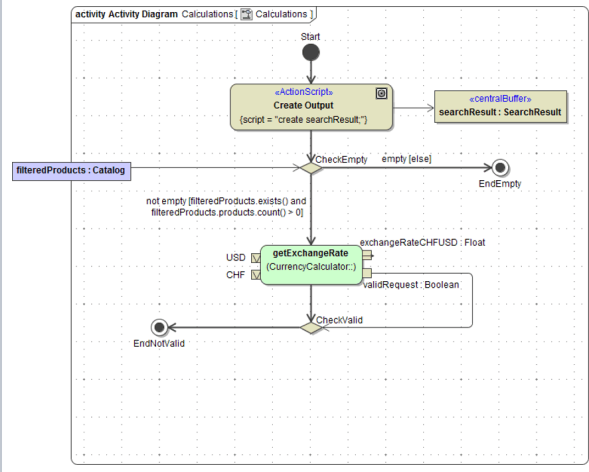
Now convert the second input pin **toCurrency** into a value pin and assign the value **CHF**.



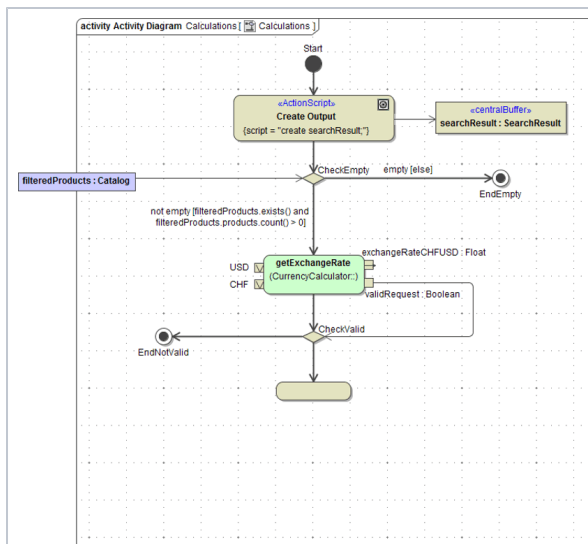
Save  the UML model.

## Checking the Search Result

After calling the SOAP adapter, you have to decide, whether you got a valid response from the Exchange Rate Provider.

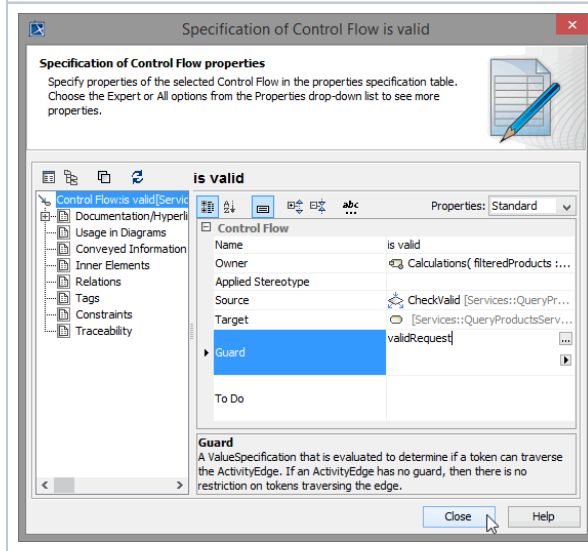
	<p>Insert a <b>Decision on Node</b> and assign the name <b>CheckValid</b>.</p>
	<p>As an input, the decision node needs the flag <b>validRequest</b>.</p> <p>Draw the necessary object flow.</p>
	<p>When the SOAP request was not valid, you can not continue with the calculations.</p> <p>Draw an <b>Activity Final</b> named <b>EndNotValid</b>.</p>





When the SOAP request was valid, you will transfer the exchange rate to the output parameters and continue with the calculations.

Draw a control flow that ends in an action node.

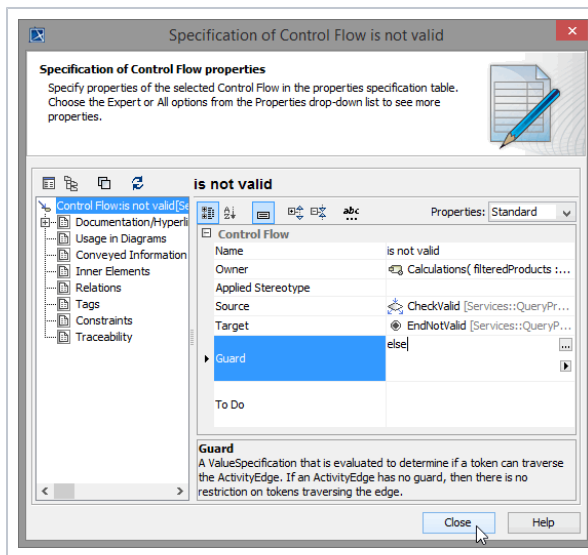


Now, you have to define the guard expressions.

Double-click the control flow that leads down to open the **Specification** dialog. Assign the name **is valid** and enter the guard expression **validRequest**.

Click **Close**.

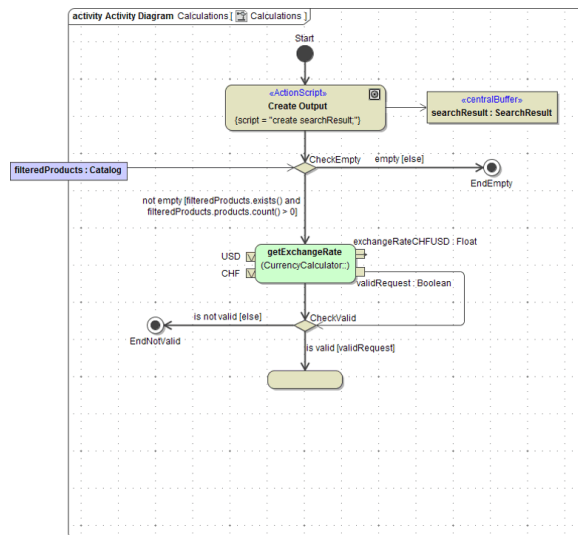
As it concerns a variable of type boolean, the expression **validRequest** is equal to the expression **validRequest = true**.



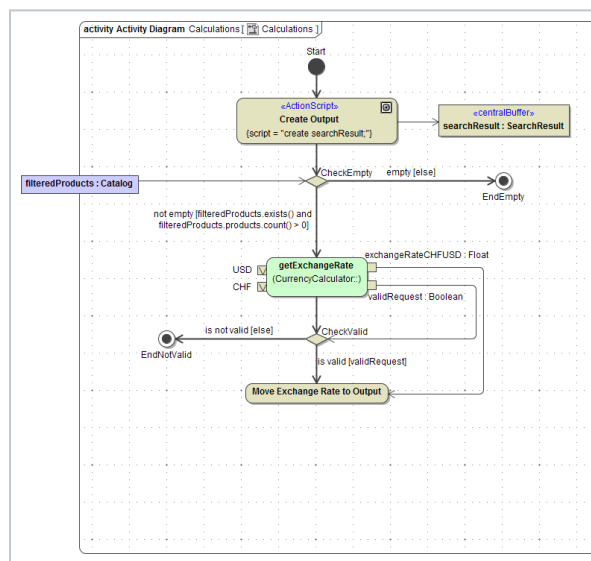
Double-click the other control flow which leads to **EndNotValid**. Assign the name **is not valid** and enter the guard expression **else**.

Click **Close**.

If the request of the external SOAP service was valid, the calculations are proceeded. If not, all further calculations are skipped.



Now, continue with the calculations. To the unnamed action node assign the name **Move Exchange Rate to Output**.



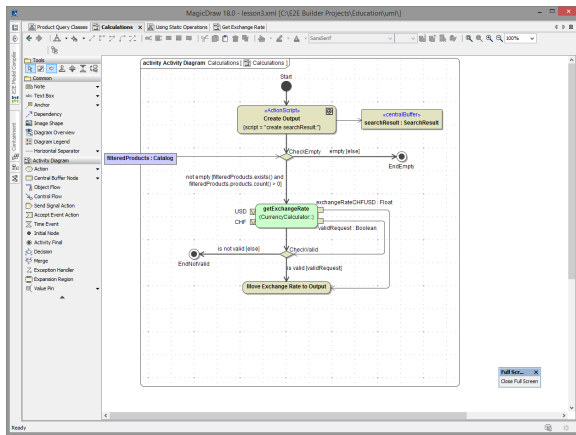
As the exchange rate will be moved to output, you need the exchange rate as an input to the action node.

Draw the appropriate object flow.

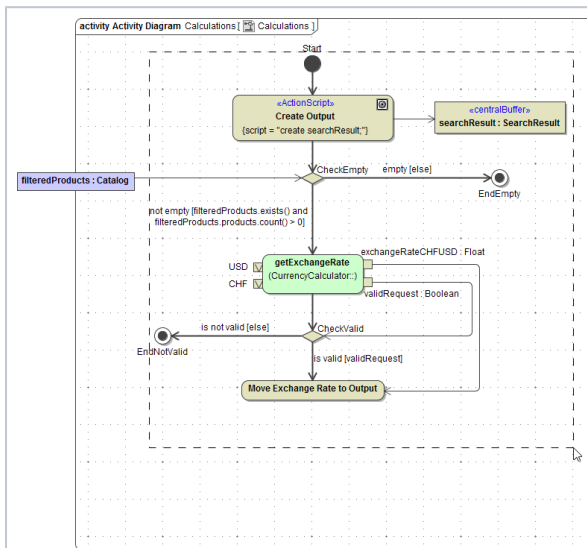
The object **searchResult** will be used within this action node and is therefore needed as an input. To avoid crossing control and object flows which reduce comprehensibility of the diagram, you will not directly connect the buffer node **searchResult** created in **Creating Output** to the action node **Move Exchange Rate to Output**.

You are going to copy the buffer node **searchResult** on the left of action node **Move Exchange Rate to Output**.

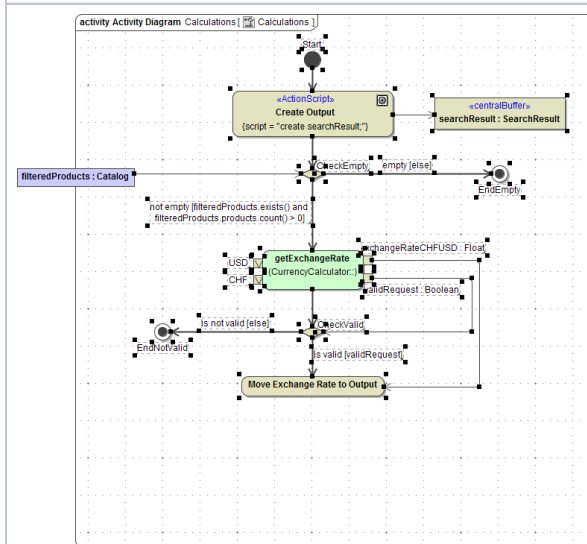
But first, make some room in your diagram. Press **F11** to switch to full screen mode.



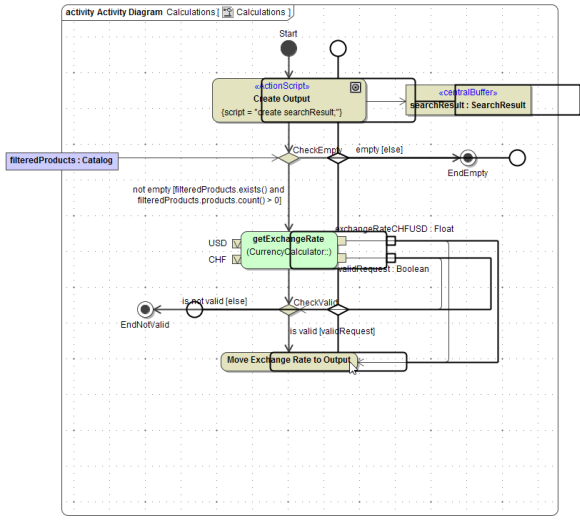
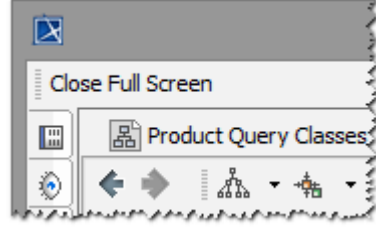
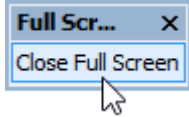
The MagicDraw menu bar disappears, the containment tree and the Compiler window are toggled to auto-hide mode to the benefit of the diagram pane, which expands to the full screen.

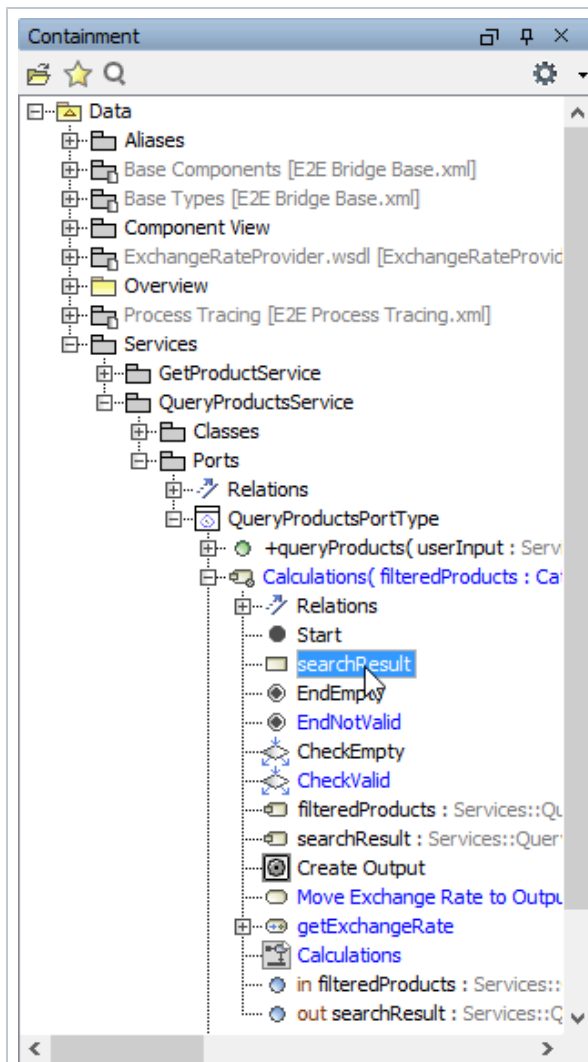


Select all diagram elements beside the parameter **filteredProducts** by drawing a selection area.

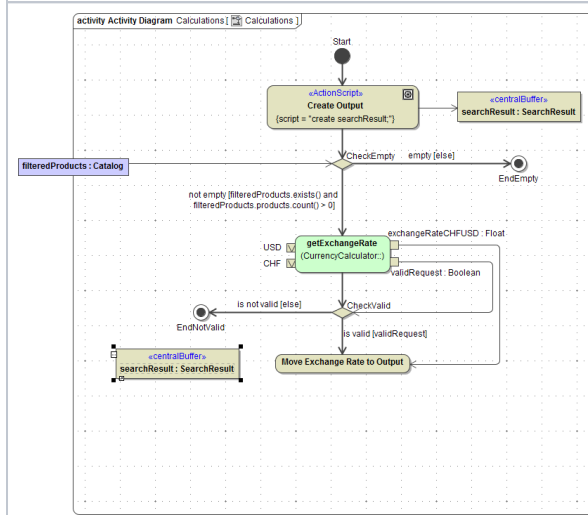


All selected elements are marked with black rectangles.

 <p>The diagram shows an activity for calculations. It starts with a 'Start' node leading to an 'ActionScript: Create Output' node with the script 'create searchResult'. This leads to a 'SearchResult: SearchResult' object. A decision node 'CheckEmpty' follows. If empty, it goes to 'EndEmpty'. If not empty, it leads to 'getExchangeRate (CurrencyCalculator:)' which takes 'USD' and 'CHF' as inputs and returns 'exchangeRateCHFUSD: Float'. This leads to a 'CheckValid' decision. If not valid, it goes to 'EndNotValid'. If valid, it goes to 'Move Exchange Rate to Output'.</p>	<p>Move the selection a little to the right.</p> <p>If you drag the selection area over the diagram border, the diagram border will move accordingly.</p>
 <p>A screenshot of a software interface showing a 'Close Full Screen' button in the upper left corner of a window titled 'Product Query Classes'.</p>	<p>Close Full Screen mode.</p> <p>You can do this by clicking the appropriate button in the upper left corner.</p>
 <p>A screenshot of a separate window titled 'Full Scr...' with a 'Close Full Screen' button. A mouse cursor is pointing at the button.</p>	<p>The button may be displayed in a separate window also.</p> <p>The third possibility to close Full Screen mode is pressing <b>F11</b> again.</p>

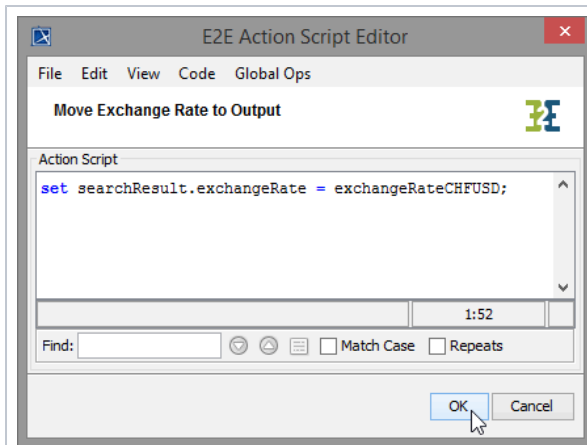


In the containment tree, select the buffer node **searchResult**.



Drop it next to the action **Move Exchange Rate to Output**.

Connect the buffer node to the action node.

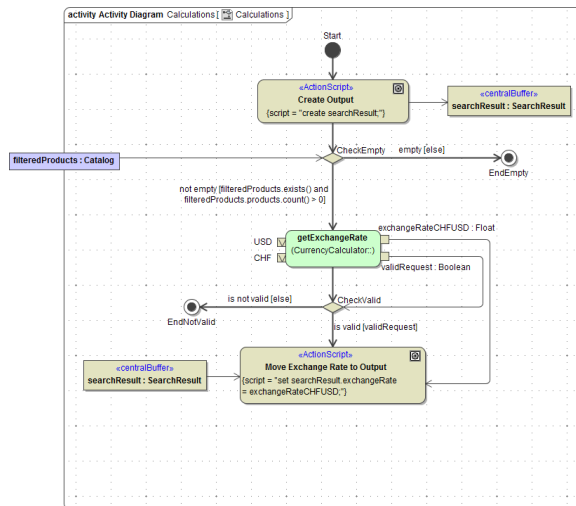



Select the action node **Move Exchange Rate to Output** again and press **Ctrl - Enter** to open the **Action Script Editor**.

Enter the statement shown in the screenshot on the left. The value of the output parameter **exchangeRateCHFUSD** is copied to the attribute **exchangeRate** of the object **searchResult**.

Close the dialog.

The action node is completed.

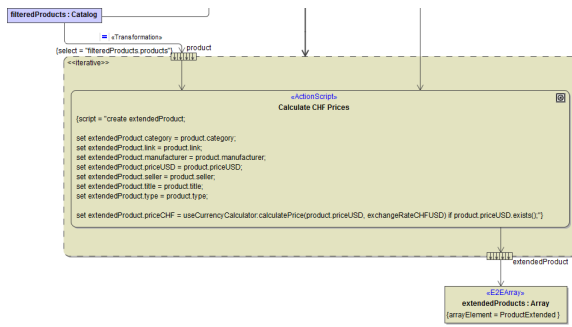


Save  the UML model.

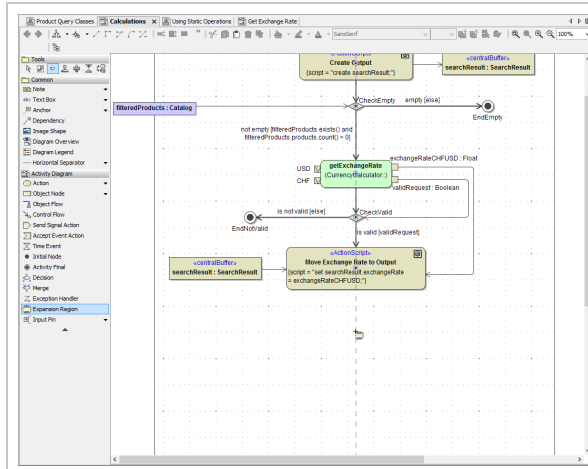
## Converting the Price from USD to CHF

In the next step, the price conversion from USD to CHF is done for each product record found in the filtered data. The filtered products are stored in object **filteredProducts**, which you need as an input parameter. You will define an UML iteration to iterate each product record and execute specific actions.

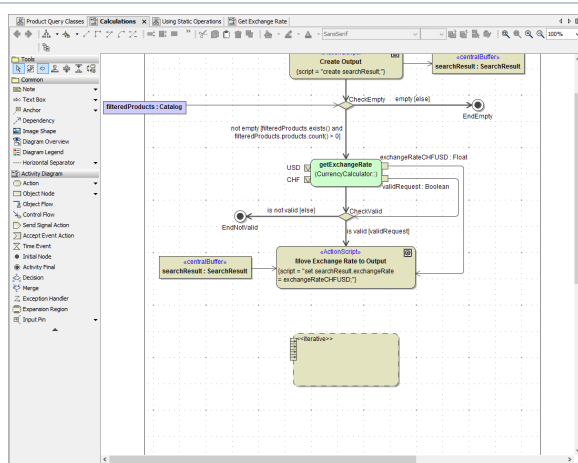
E2E Builder provides functionality to iterate over action script, an adapter, a class operation or over a call behavior action. In this lesson, you will learn how to iterate over an action script as shown in the picture below.




Iterations are defined by the use of expansion regions with stereotype `<<iterative>>`.



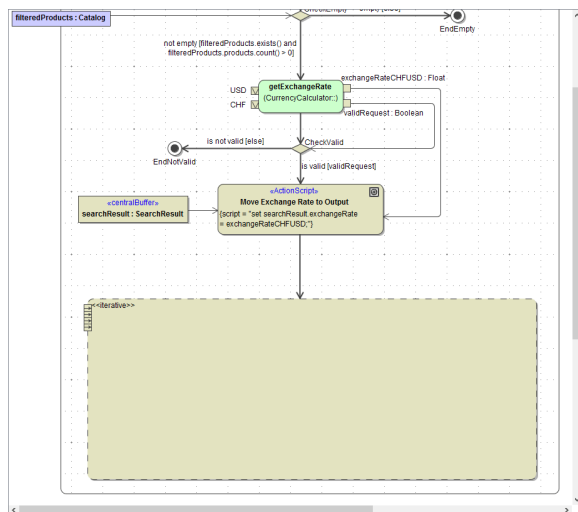
Select an **Expansion Region** from the diagram toolbar and place it on the diagram pane.



The expansion region is drawn with an **Expansion Node** in the upper left corner (  ).

An object flow that arrives at an expansion node contains a collection of objects or data, which are separated by the expansion node before being passed onto elements within the expansion region

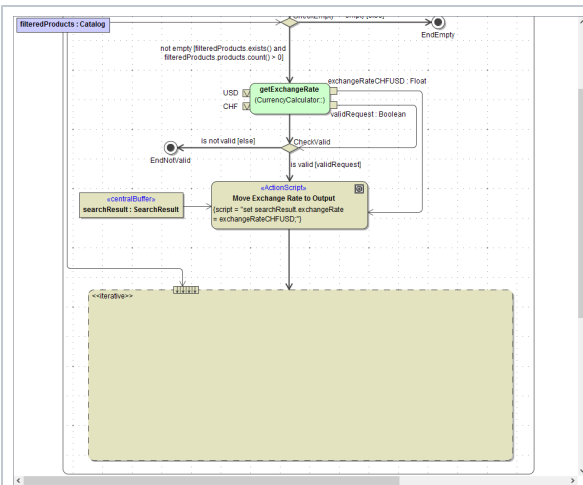
An expansion region must have one or more expansion nodes receiving input. You can iterate over only one of the input nodes. It may have any number of expansion nodes as output including the case of having no output expansion node.



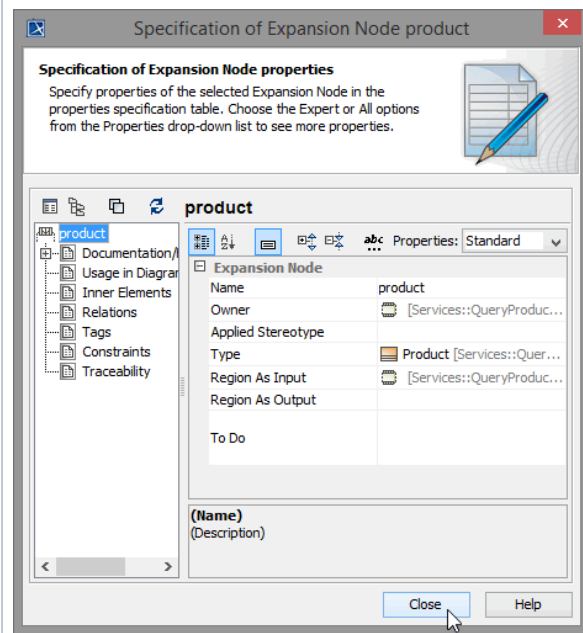
Scroll down if necessary and resize the expansion region. Toggling to Full Screen mode may be helpful as well.

Connect the expansion region to the control flow.





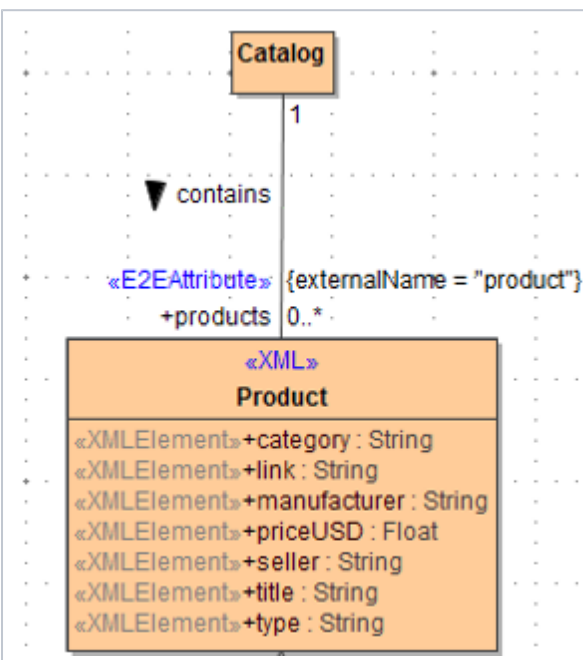
Draw the object flow from the activity parameter node **filteredProducts** to the expansion node as shown in the picture on the left.



Open the **Expansion Node's Specification** dialog and assign the name **product** and the type **Product**.

The element **product** is used as a temporary iteration object. In each iteration, a product record is stored in this object.

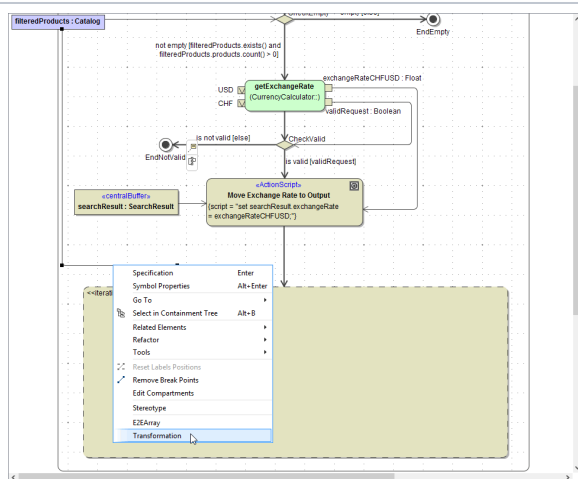
Click **Close**.



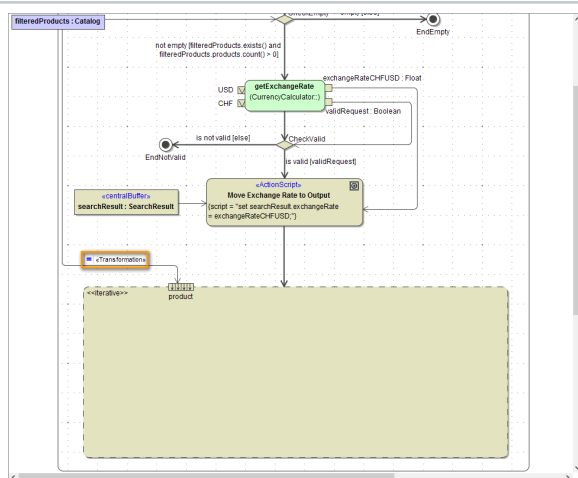
The object **filteredProducts** of type **Catalog** has an array attribute **products** that contains elements of type **Product**.


In each iteration, one element of the array **products** will be assigned to the temporary iteration object **product**.

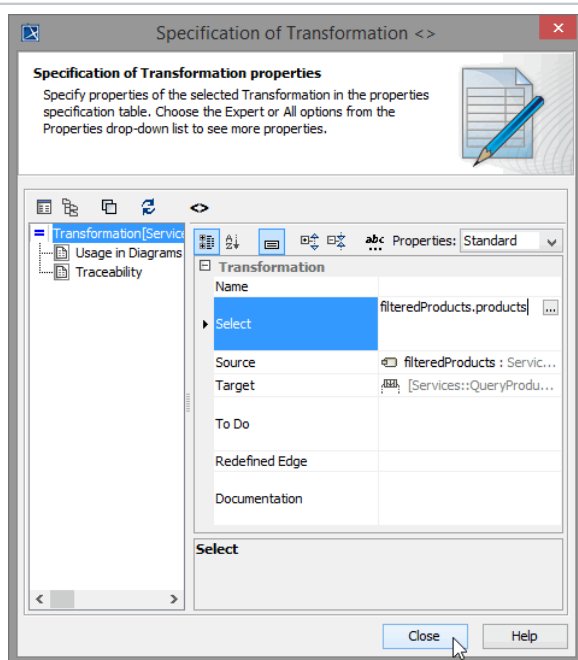
In order to pass an element of the array **products** of **filteredProducts** to the iteration object **product**, you can use a **Transformation**.



Right-click the object flow connecting the activity parameter node with the expansion node and select **Transformation** from the context menu.



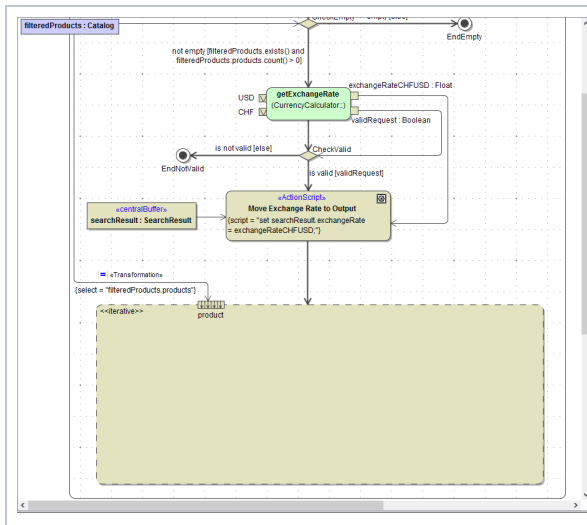
This stereotypes the object flow as transformation and adds the corresponding icon .



Double-click this icon to open the **Specification** dialog of the **Transformation**.

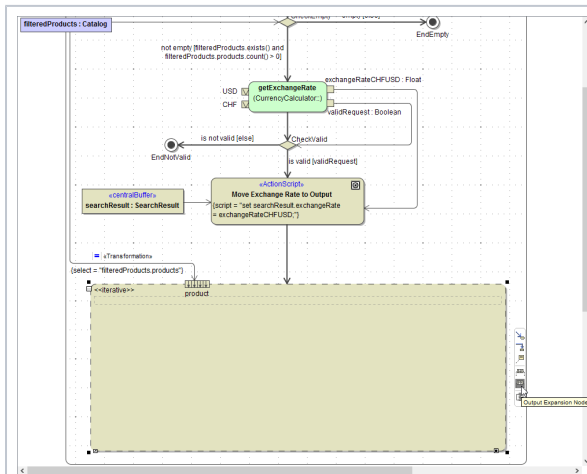
In the **Select** field, enter **filteredProducts.products**.  
products  
and close the dialog.

To avoid typing errors, you could also use the **Action Script Editor** for this.

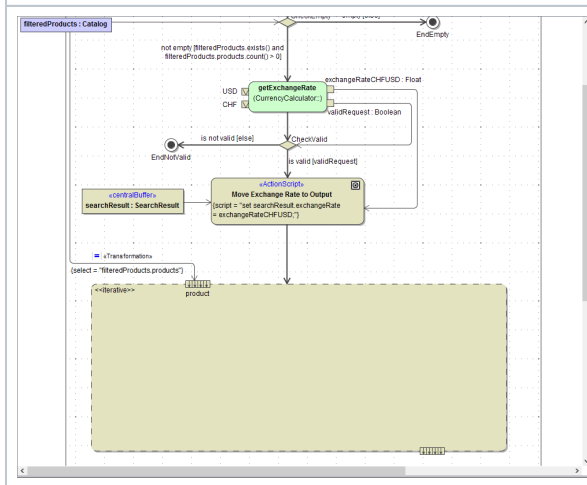


The transformation rule is displayed in the diagram.

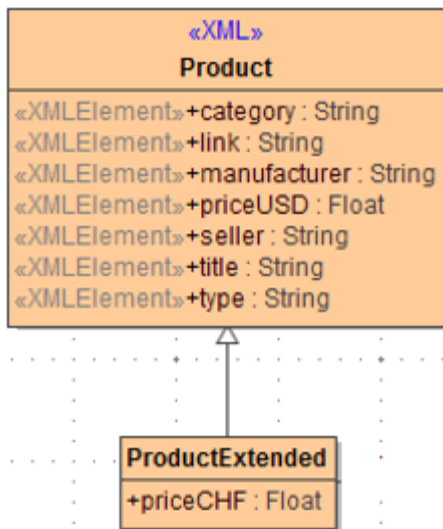
Now, define the result of each iteration step.



Create an **Output Expansion Node** by selecting the corresponding icon from the smart manipulation toolbar of the expansion region.

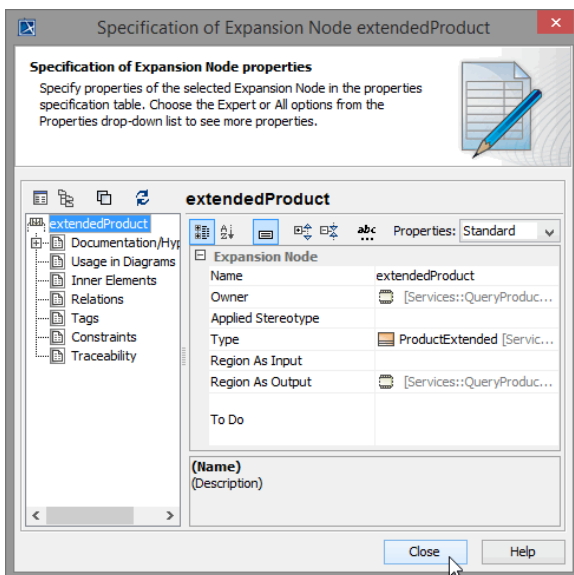


Drag the output expansion node to the bottom of the expansion region.



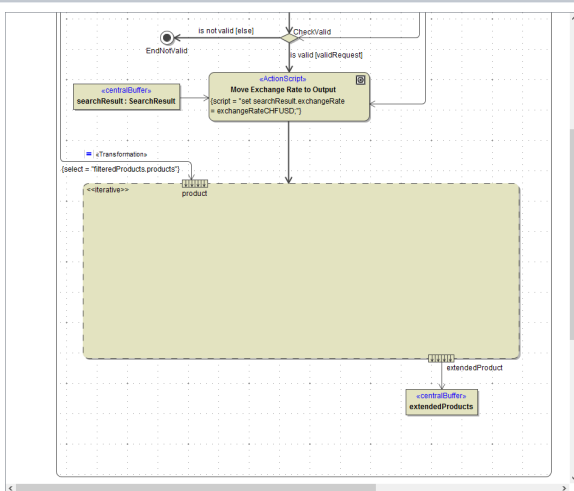
The calculated price in CHF is no attribute of class **Product**, as **Product** describes the structure of the XML file.

For storing the price in CHF, you defined the class **ProductExtended** with all attributes of **Product** and the additional attribute **priceCHF**. This type, you are going to use for the output expansion node.



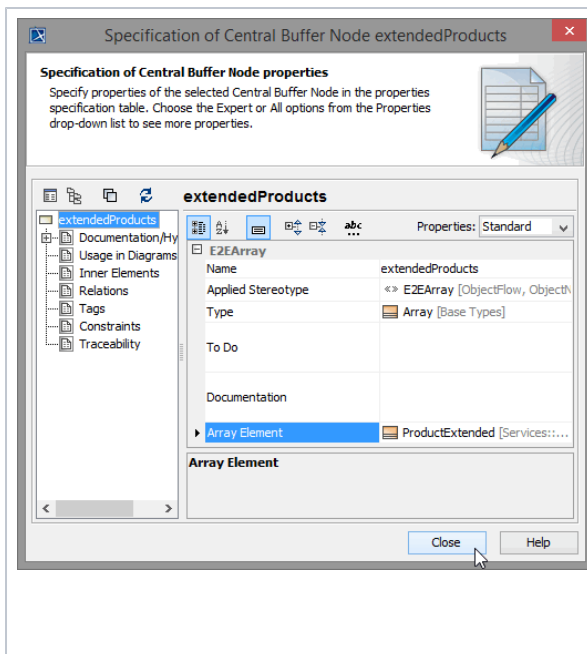
Open the **Specification** dialog of the output expansion node. Assign the name **extendedProduct** and the type **ProductExtended**.

Click **Close**.



The result of each iteration step has to be stored in an array.

Draw an object flow starting at the output expansion node and ending in a central buffer node. Assign the name **extendedProducts**.



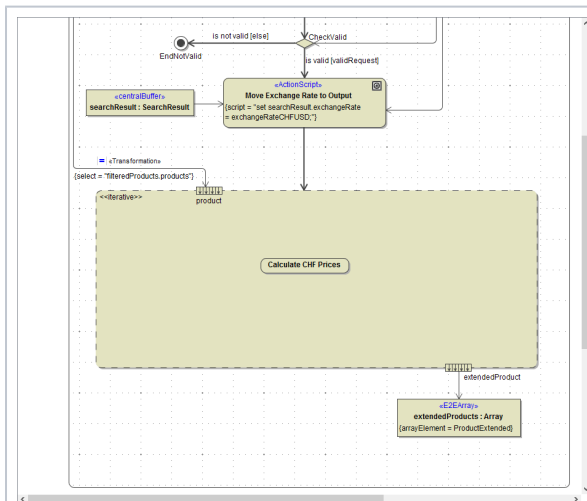
Open the **Specification** dialog of **extendedProducts**.

Apply the stereotype **<< E2EArray>>** and assign the type **Array**.

The array element type is defined in field **Array Element**. Each array element has to be of type **ProductExtended**. Select the corresponding type from the list.

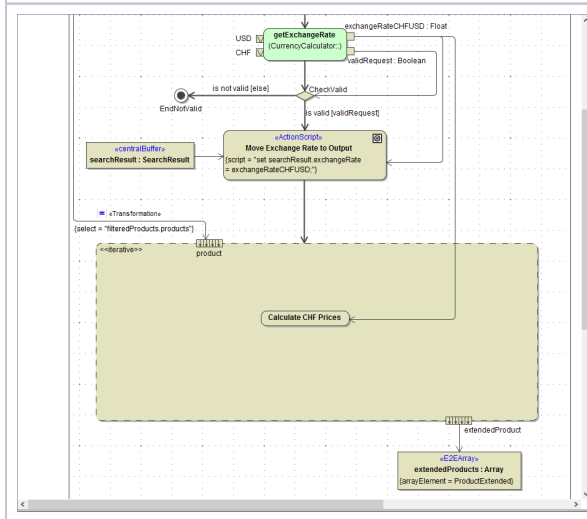
Click **Close**.

Now, you have defined the input and the output of the expansion region. The purpose of this iteration is to iterate over an action, which calculates the CHF price for each **products** item.



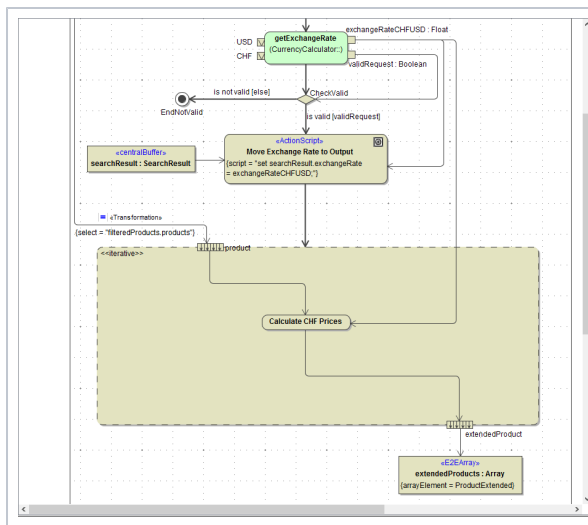
To add an action node to the expansion region, select the **Action** icon from the diagram toolbar and place it within the expansion region.

Assign the name **Calculate CHF Prices**.



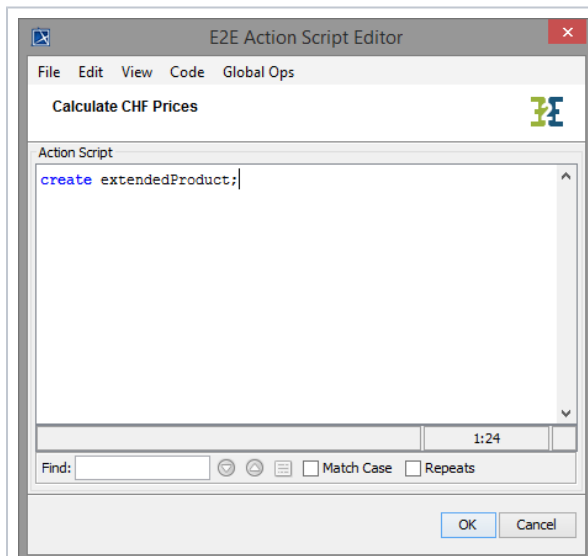
The action node **Calculate CHF Prices** needs the exchange rate as input to do the calculation.

Therefore, draw an object flow from the output pin **exchangeRateCHFUSD** of **getExchangeRate** directly to the action node **Calculate CHF Prices**.



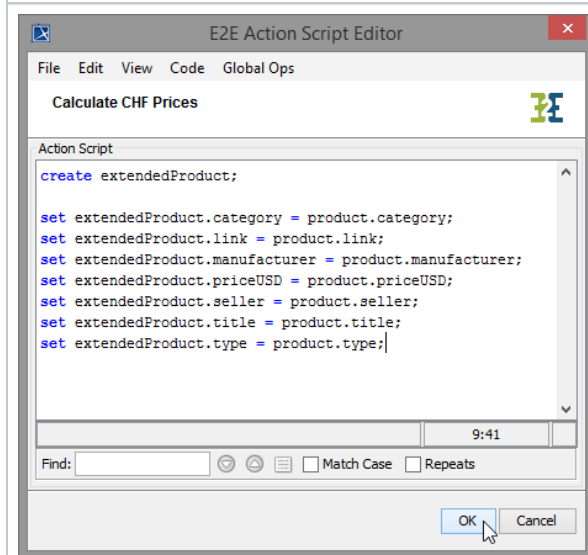
Connect the input and output expansion nodes to the action node as well.

This action will contain the action script being processed for each array element of the input array. As a result of each iteration step, a new item will be appended to the output array. Within the action script, the array items have the attribute name defined in the input expansion node. The expansion node creates temporary input array items. The output objects need to be created by create statements within the action script and are appended to the related output array by the output expansion node.



Select **Calculate CHF Prices** and open the Action Script editor.

First, create object **extendedProduct**, that you need to store the calculated data.



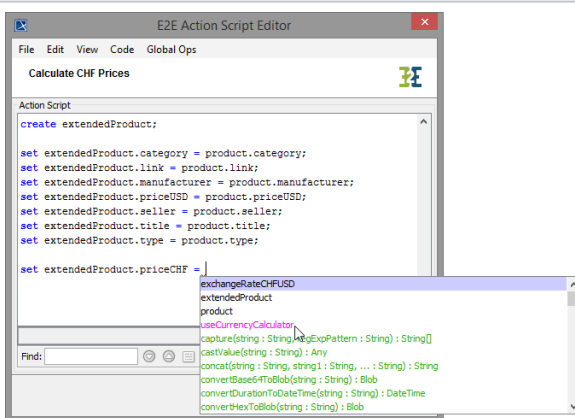
Move all product data from the input object **product** to the output object **productExtended**, using set statements.

```

set extendedProduct.category = product.category;
set extendedProduct.link = product.link;
set extendedProduct.manufacturer = product.manufacturer;
set extendedProduct.priceUSD = product.priceUSD;
set extendedProduct.seller = product.seller;
set extendedProduct.title = product.title;
set extendedProduct.type = product.type;

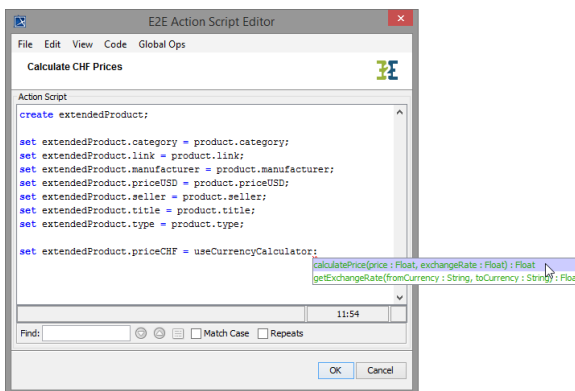
```

Finally, you are going to implement the calculation of the CHF price. You will use the class operation **calculatePrice** you defined in class **CurrencyCalculator**. Remember, that you made it available to be used within action script via a `<<use>>` dependency called **useCurrencyCalculator**.

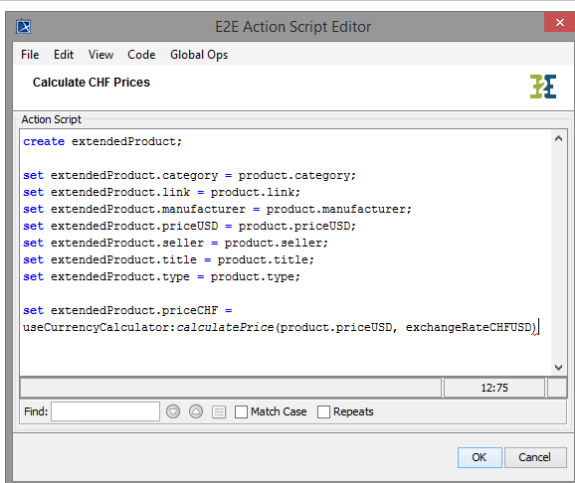


Start typing the set-statement `set extendedProduct.priceCHF =` and press **Ctrl - Space** to display the suggestion list.

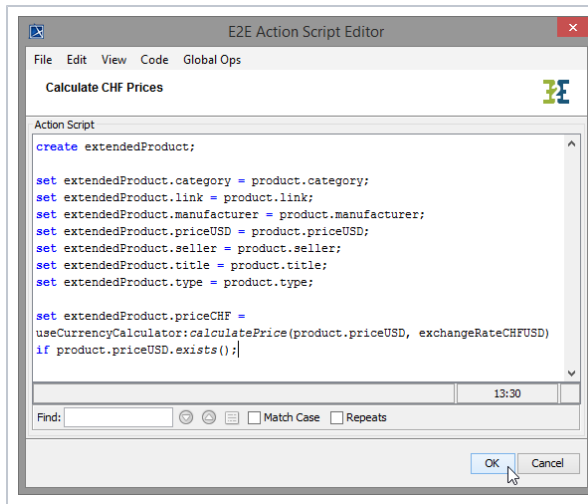
Select the `<<use>>` dependency **useCurrencyCalculator**.



Type `:` and the available methods will be displayed in the suggestion list. Select **calculatePrice**.




Enter the two needed parameters `product.priceUSD` and `exchangeRateCHFUSD` within the brackets.



Finalize the statement by typing `if product.priceUSD.exists();` as the currency calculation only should be executed if a price is assigned to the product.

The last statement in total should read:

```
set extendedProduct.priceCHF = useCurrencyCalculator:calculatePrice  
(product.priceUSD,exchangeRateCHFUSD)  
    if product.priceUSD.exists();
```

Save  the UML model.