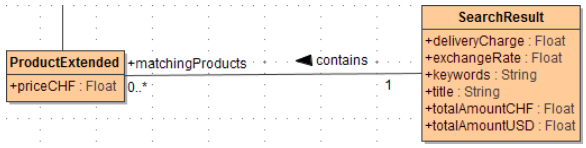
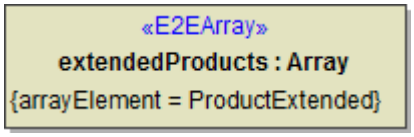
Implementing Additional Calculations MD18

In the last step, you are going to summarize the prices to form a price total and calculate the delivery charge, assuming an order quantity of one.



This additional information you defined to be stored in class **SearchResult**.

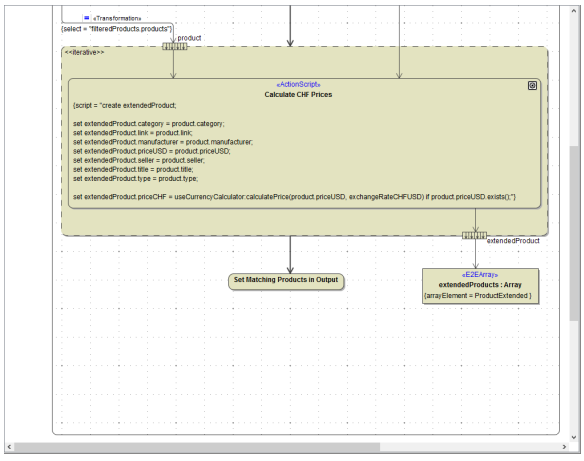
Objects of class **SearchResult** also contain zero to infinite objects of class **ProductExtended** in array **matchingProducts** defined on the association end.



As an output of the iteration you implemented in the last chapter, you got the array **extendedProducts** containing objects of type **ProductsExtended**.

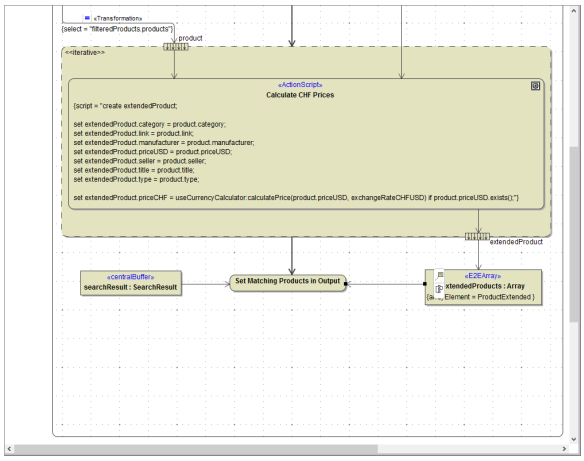
All filtered products with converted prices are stored in this array.

Now you are going to set this array into the output object.



Below **Calculate CHF Prices** draw an action node **Set Matching Products in Output** and connect it to the control flow.

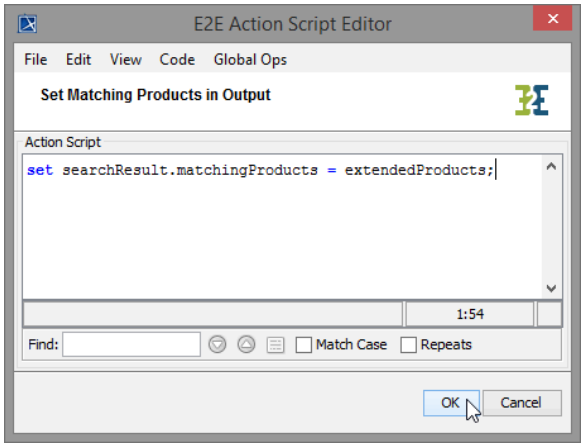
Components



Before calculating the prices, you already defined a buffer node **searchResult** of type **SearchResult** and stored the exchangeRate in this object. You could connect this buffer node to **Set Matching Products in Output** now. But for reasons of comprehensibility, you will copy the buffer node again.

Select the buffer node **searchResult** in the containment tree and drop near the left diagram border.

Draw object flows from the buffer node and the array **extendedProducts** to the action node.

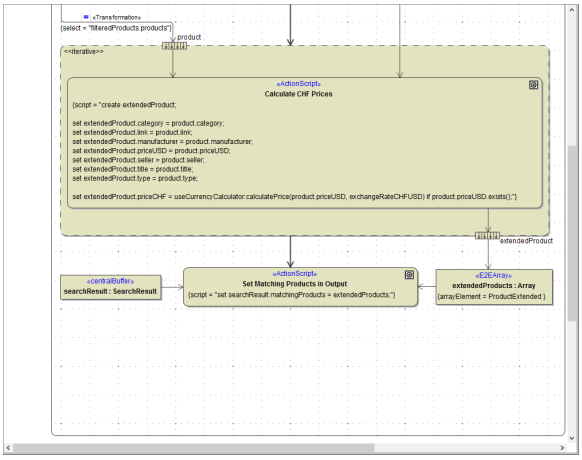


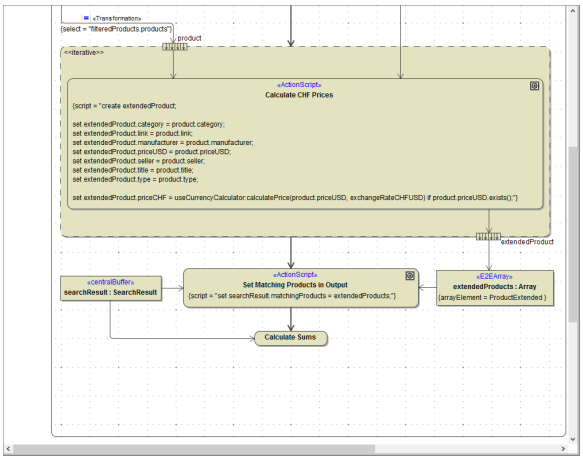
Open the Action Script Editor of **Set Matching Products to Output** and enter the following statement:

```
set searchResult.matchingProducts = extendedProducts;
```

The array **matchingProducts** you defined in the class diagram on the association end belonging to class **SearchResult** receives the content of array **extendedProducts**, containing all filtered products with converted prices.

Click **OK**.



	Draw an action node with control flow below **Set Matching Products in Output** and assign the name **Calculate Sums**. Draw an object flow from **searchResult** to the newly created action node, as you need it as input.

Open the Action Script editor and insert the two following set statements:

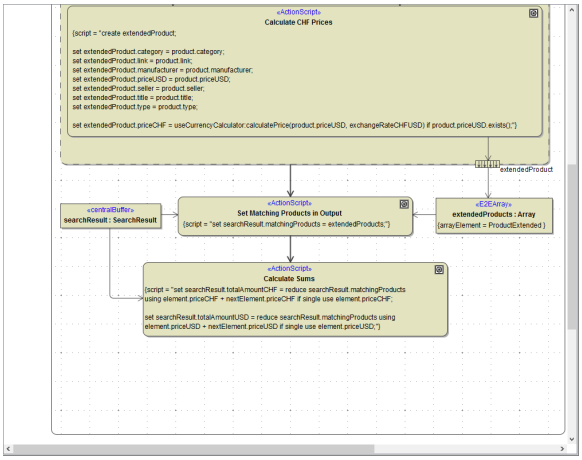
```
set searchResult.totalAmountCHF = reduce searchResult.matchingProducts
using element.priceCHF + nextElement.priceCHF if single use element.
priceCHF;
set searchResult.totalAmountUSD = reduce searchResult.matchingProducts
using element.priceUSD + nextElement.priceUSD if single use element.
priceUSD;
```

In this action script, you combine a `set` statement with a `reduce` operation. The `reduce` operation applies to arrays and allows you to reduce the array having elements of complex types to a scalar value. Therefore, an expression is recursively applied to each array element (`element`) and its next element (`nextElement`).

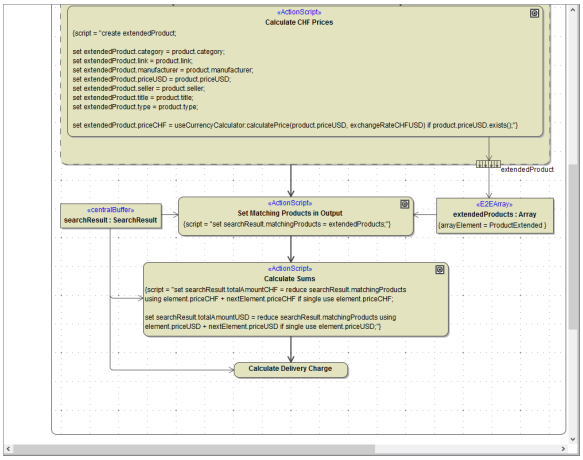
`set searchResult.totalAmountCHF`	The total of the reduce operation is stored in **searchResult.totalAmountCHF**.
`reduce searchResult.matchingProducts`	The reduce operation is applied to the array **searchResult.matchingProducts**.
`using element.priceCHF + nextElement.priceCHF`	For calculating the sum, you must use the currently evaluated array element (**element**) and its next neighbor (**nextElement**). The attribute **priceCHF** of **element** and **nextElement** are added up and the result is stored in **searchResult.totalAmountCHF**.

`if single use element.priceCHF;`	This is a mandatory extension and allows to define the action for an array containing only one single element. If there is only one matching product, the total amount equals to the price of this product.

For more information about the reduce operation refer to the xUML Services Reference Guide.



Finally, you are going to calculate the delivery charge.

	Draw an action node with control flow below **Calculate Sums** and assign the name **Calculate Delivery Charge**. Draw an object flow from **searchResult** to the newly created action node, as you need it as input.

Open the Action Script Editor and insert the following statements:

```
local noChargeAmount = setting("No charge at amount of", 50.0);
local deliveryCharge = setting("Delivery charge", 10.0);

set searchResult.deliveryCharge = if searchResult.totalAmountCHF >=
noChargeAmount then 0.0 else deliveryCharge;
```

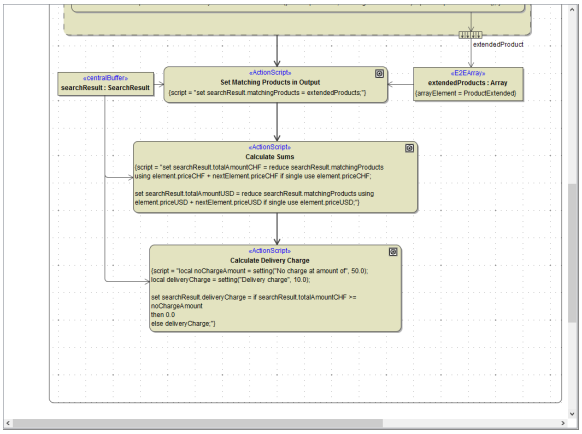
For the delivery charge calculations, you need the delivery charge amount itself (**deliveryCharge**), the amount starting from which the delivery is free (**noChargeAmount**) and the total of the order you just calculated (**searchResult.totalAmountCHF**).

`local noChargeAmount = setting("No charge at amount of", 50.0);` `local deliveryCharge = setting("Delivery charge", 10.0);`	**noChargeAmount** and **deliveryCharge** are defined as local variables within the action script.

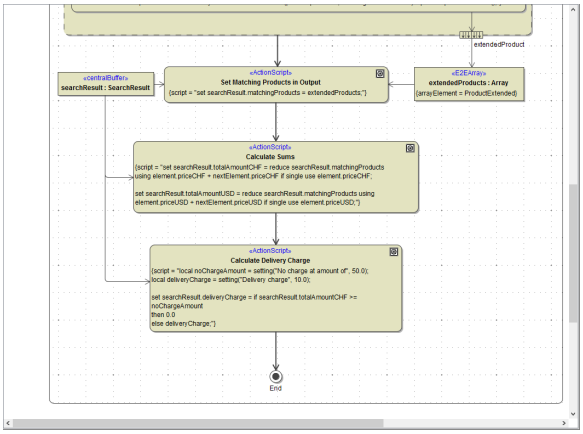
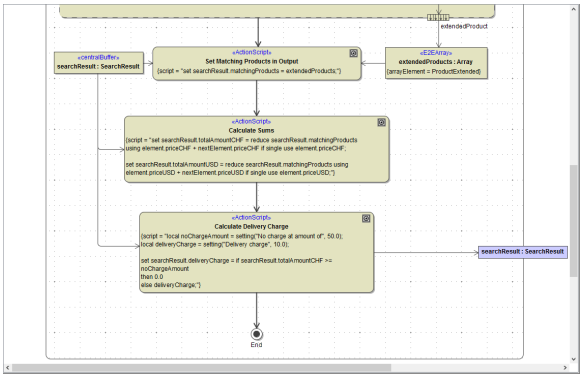
Remember that in lesson 2 you already learned how to use the setting macro function. With the setting macro literals can be stored global to an xUML service. In the Bridge context, it is possible to define name / value pairs that are configurable from the E2E Embedded Runtime and the E2E Bridge.

The setting macro function also can be combined with a local variable definition.

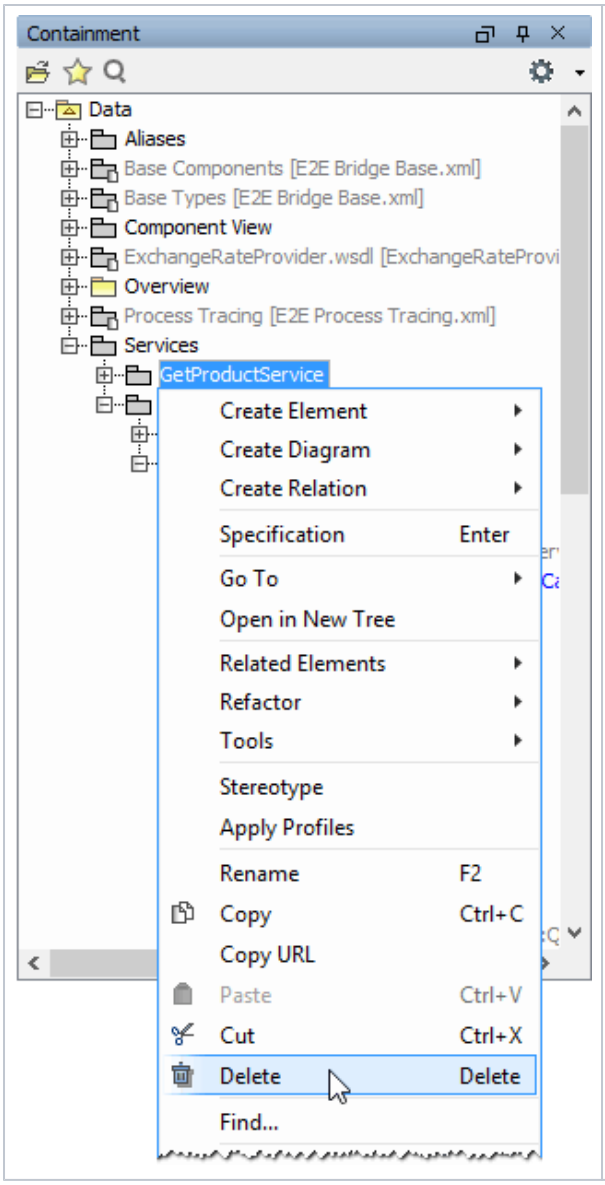
`set searchResult.deliveryCharge =`	The result of the calculation is stored in **searchResult. deliveryCharge**.
`if searchResult.totalAmountCHF >= noChargeAmount`	The set statement can be combined with a condition. If the order value exceeds the limit defined in **noChargeAmount** ...
`then 0.0`	... then no delivery charge is asked ... Remember to use the format **0.0** as **searchResult. deliveryCharge** is of type float.
`else deliveryCharge;`	... in all other cases the delivery charge defined by the setting is assigned.



Complete the diagram by drawing the activity final node and add the parameter **searchResult** on the right border of the diagram pane.

	Assign the name **End** to the activity final node.
	Drag and drop the parameter **searchResult** onto the right diagram border and connect it to **Calculate Delivery Charge**.

Now, all calculations are implemented.



The package **GetProductService** is not needed anymore and can be deleted. Leaving the package would lead to compiler errors as you changed the interface of the **CurrencyCalculator** in the second iteration.

Save the UML model.