Iterations

Another sometimes more elegant way of looping over arrays is using iterations. One can use iterations in the following scenarios:

- Iterating Over Action Scripts
- Iterating Over Adapters
- Iterating Over Class Operations
- Iterating Over Call Behavior Action

Also, you have access to the index of the iterations as described at Accessing the Iteration Index.

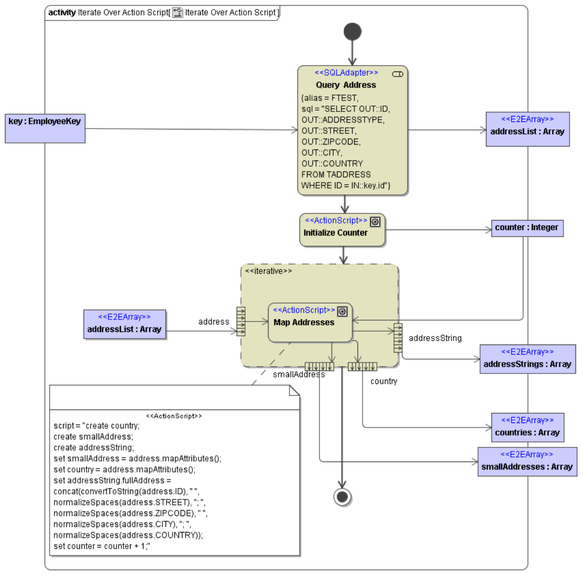
On this Page:

Example File (Builder project Basic Modeling/Flows):

	<your example path>\Basic Modeling\Flows\uml\iteration.xml

Iterating Over Action Scripts

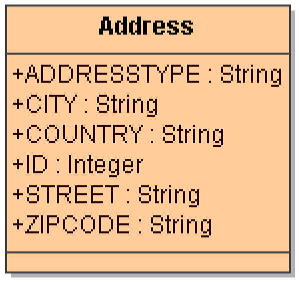
Figure: Iterating Over Action Scripts



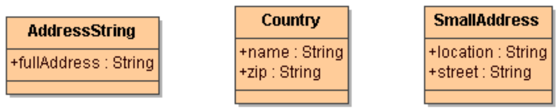
```
{alias = FTEST,
sql = "SELECT OUT::ID,
OUT::ADDRESSTYPE,
OUT::STREET,
OUT::ZIPCODE,
OUT::CITY,
OUT::COUNTRY
FROM TADDRESS
WHERE ID = IN::key.id"}
```

```
script = "create country;
create smallAddress;
create addressString;
set smallAddress = address.mapAttributes();
set country = address.mapAttributes();
set addressString.fullAddress =
concat(convertToString(address.ID), " ",
normalizeSpaces(address.STREET), "; ",
normalizeSpaces(address.ZIPCODE), " ",
normalizeSpaces(address.CITY), "; ",
normalizeSpaces(address.COUNTRY));
set counter = counter + 1;"
```

The example above iterates over the Array **addressList**. This Array has a complex type **Address** as arrayItem. The type **Address** is a class with the following definitions:

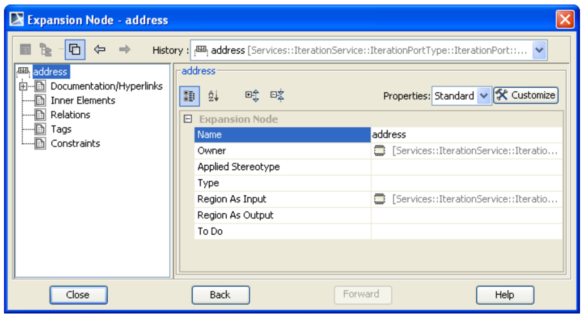


Within the action script the following output items of complex type are created:



At the end of each iteration the output Items are appended to the related output Arrrays.
To iterate over an action script, draw an Expansion Region with stereotype <<iterative>> as shown in the above example.
Within the Expansion Region draw an action with the stereotype <<Action Script>>. This action contains the action script. This script will be processed for each Array Element of the input Array. For each Iteration a new item will be appended to each output Array. Within the action script the Array Items have the given Attribute name defined in the Expansion Node. The Expansion Node therefore creates the 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 expansion node. Each expansion region can have only one input Expansion Node and several output Expansion Nodes.

Figure: Iterating Over Action Scripts Expansion Node



Expansion Node

Name	address
Owner	[Services::IterationService::Iteratio...
Applied Stereotype	
Type	
Region As Input	[Services::IterationService::Iteratio...
Region As Output	
To Do	

For each Array Item of the input Array a new temporary object is created. This object gets the name from the Expansion Node and the type taken from the tag "**arrayElement**" of the input Array. You can use this variables into the action script.
Output expansions nodes can be drawn by selecting the symbol **Output Expansion Node** from the **Obje ct Node** pull-down menu in the diagram toolbar. The output objects need to be created within the action script. The name is given by the name of the output Expansion Node. The type is taken from the tag "**arra yElement**" on the related output Array Element. For each iteration a new Array Item will be appended to the result Array.

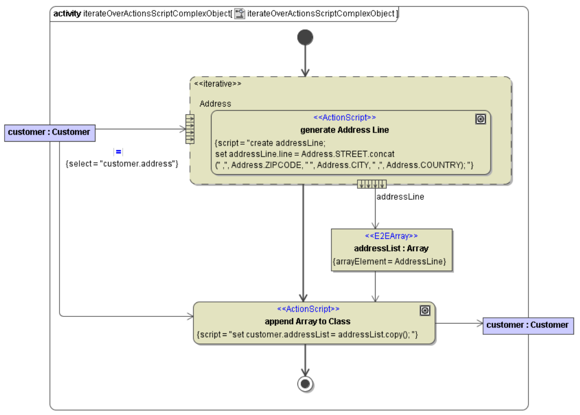
The result objects like **smallAddress**, **country**, and **addressString** need to be created first with the create statement.

To use an object static for all iterations, draw an Object Flow directly to the Action Script within the Expansion Region. See as an example the object "**counter**" in the action script of the figure above. The **c ounter** is static and will be added by 1 for each iteration ("set counter = counter + 1". The object contains at the end the number of processed addresses.

Iterating over Action Script with Complex Objects

Instead of an Array it is also possible to iterate over a dependency in a complex object with a multiplicity.

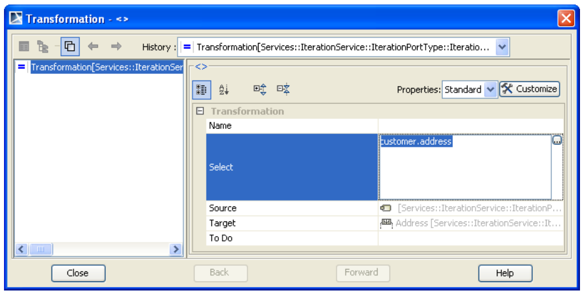
Figure: Iteration over Action Script of Complex Objects


activity iterateOverActionsScriptComplexObject[iterateOverActionsScriptComplexObject]
<<iterative>>
Address
<<ActionScript>>
generate Address Line
{script = "create addressLine;
set addressLine.line = Address.STREET.concat
(" ,", Address.ZIPCODE, " ", Address.CITY, " ,", Address.COUNTRY); "}
customer : Customer
=
{select = "customer.address"}
addressLine
<<E2EArray>>
addressList : Array
{arrayElement = AddressLine}
<<ActionScript>>
append Array to Class
{script = "set customer.addressList = addressList.copy(); "}
customer : Customer

To declare the multiplicity where the iteration should loop over, define a **Select** statement on the ObjectFlow. The ObjectFlow from Complex Object to the Expansion Node needs the stereotype Transfor

mation. If the Stereotype Transformation is set, the symbol is shown in your activity diagram.
Enter the menu by double click on the symbol or selecting the **Specification** menu out of the context menu.

Figure: Transformation Menu


Transformation - <>
History : Transformation[Services::IterationService::IterationPortType::Iteratio...
Transformation[Services::IterationSer
<>
Properties: Standard
Customize
Close
Back
Forward
Help

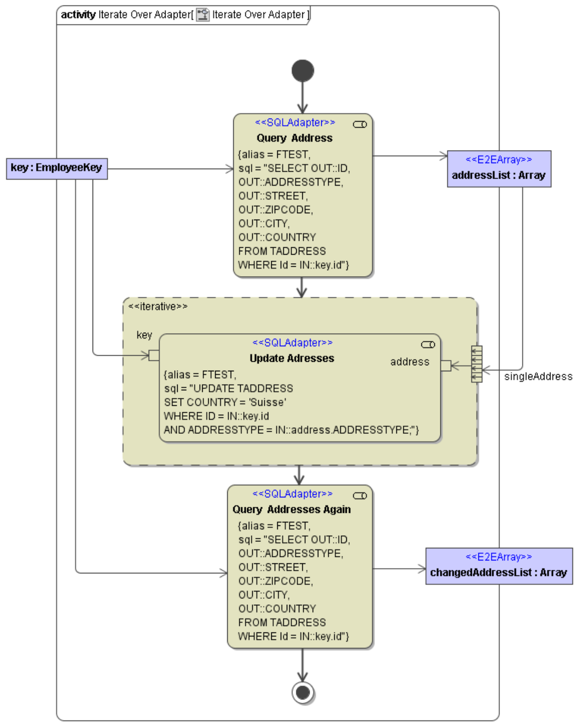
Transformation	
Name	
Select	customer.address
Source	[Services::IterationService::IterationP...
Target	Address [Services::IterationService::It...
To Do	

Insert the name of the association in the field **Select**.
To append the output of an iteration to a complex type object, define an Array Element and append it to the complex type object by an additional ActionScript. See Example above. The Array **addressList** is copied to the class **Customer** by the Action Script.

Iterating Over Adapters

Iterations over adapters (in the following example an SQL adapter) can be executed with the Expansion Region like iterations over Action Script. Inside the Expansion Region just draw an adapter instead of the Action Script.

Figure: Iterating over Adapter



```
{alias = FTEST,
sql = "SELECT OUT::ID,
OUT::ADDRESSTYPE,
OUT::STREET,
OUT::ZIPCODE,
OUT::CITY,
OUT::COUNTRY
FROM TADDRESS
WHERE Id = IN::key.id"}
```

```
{alias = FTEST,
sql = "UPDATE TADDRESS
SET COUNTRY = 'Suisse'
WHERE ID = IN::key.id
AND ADDRESSTYPE = IN::address.ADDRESSTYPE;"}
```

```
{alias = FTEST,
sql = "SELECT OUT::ID,
OUT::ADDRESSTYPE,
OUT::STREET,
OUT::ZIPCODE,
OUT::CITY,
OUT::COUNTRY
FROM TADDRESS
WHERE Id = IN::key.id"}
```

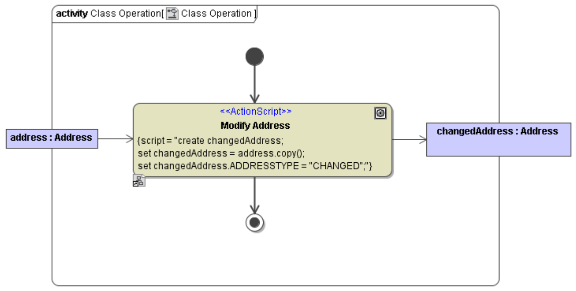
Within the Expansion Region with the stereotype <<iterative>> you can draw any E2E Bridge Adapters, in this Example an **SQL Adapter**. The Expansion Region allows one input Array and several output Arrays. It loops over the input Array and uses internally the temporary variable given in the Expansion Node. If the name of the Expansion Node corresponds to the parameter of the adapter, any Object Flow within the Expansion Regions are optional. The iteration over Adapter also supports input and output Pins on the Adapter. This means you can do a mapping between the temporary variable name given in the Expansion Node and the input Pin of the Adapter (see example above). Any Static Variable that has to be available for all iterations can directly be attached by an Object Flow into the Adapter. For this purpose no Expansion Nodes are used. The Object Flow can be directly connected to the Adapter or by input Pin. Like in the Example "Iterate Over Action Script" any output variables must be created by the adapter. The Output Variable must be named by an output Expansion Node on the frame of the Expansion Region. It will then append an item in the associated Array Item. With the usage of output Pins on the adapter a mapping to a different name of the temporary Array Item is possible.

When defining SQL statements of an action state, the E2E Action Script Editor cannot be used. The MagicDraw action specification dialog has to be used instead.

Iterating Over Class Operations

Iterations over Class Operations are similar to other iterations. The service iterates over an array and executes for each item of the array the defined class operation. In this example it modifies the attribute **A DDRESSTYPE** on each **address** by inserting the value "**CHANGED**".

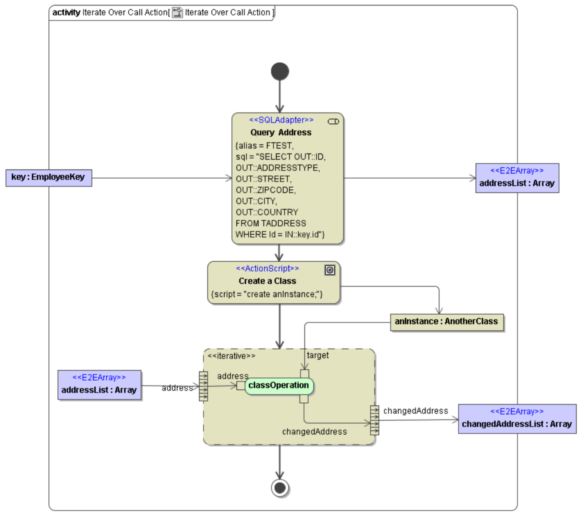
Figure: Used Class Operation



```
{script = "create changedAddress;
set changedAddress = address.copy();
set changedAddress.ADDRESSTYPE = "CHANGED";"}
```

To iterate over a Class Operation, draw a "Call Operation" symbol within the Expansion Region. Therefore, drag and drop the class operation from the Containment Tree into the Expansion Region or draw the symbol for a "Call Operation Action" in the Expansion Region. After you drop the symbol in the expansion region, you have to select the linked operation out of the pop up menu "Select Operation". The name of the selected operation is shown in the symbol.

Figure: Iterate Over Class Operation



```
{alias = FTEST,
sql = "SELECT OUT::ID,
OUT::ADDRESSTYPE,
OUT::STREET,
OUT::ZIPCODE,
OUT::CITY,
OUT::COUNTRY
FROM TADDRESS
WHERE Id = IN::key.id"}
```

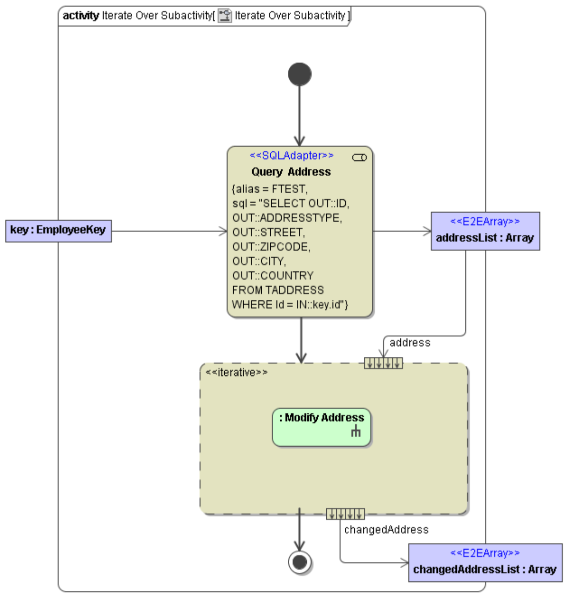
```
{script = "create anInstance;"}
```

To link the Call Action to the right Class Instance define an input Pin on the Call Operation Action with the name "**target**". Draw an Object Flow from the Object Instance in your Activity Diagram to the input Pin "**target**".
For any other input and output parameter of the Class Operation use Expansion Nodes on the Expansion Region. The Expansion Node must have the parameter name for the class Operation or be mapped by an object flow to an input Pin with the parameter name of the class operation.

Iterating Over Call Behavior Action

The following example shows how to extract each single item of the array **addressList**. Each single **address** object is passed as input to the subactivity (activity diagram **Modify Address**). The object **changedAddress** is the output of each cycle and is appended to the array **changedAddressList**.

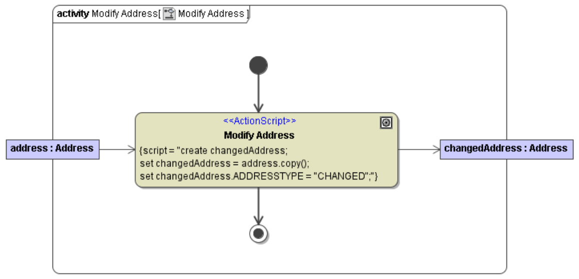
Figure: Iterating over Subactivities



```
{alias = FTEST,
sql = "SELECT OUT::ID,
OUT::ADDRESSTYPE,
OUT::STREET,
OUT::ZIPCODE,
OUT::CITY,
OUT::COUNTRY
FROM TADDRESS
WHERE Id = IN::key.id"}
```

Draw in your activity diagram an Expansion Node. If the Subactivity already exists as Action with an Activity Diagram, you can select the Activity Diagram out of the Containment Tree and drop it in the Expansion Region. For a new Subactivity, draw an "Any Action" symbol with the Action Metaclass **"Call Behaviour Action"**. Select the new symbol and select New Activity Diagram in the context menu. Define the Package for the new diagram and push the button Create Owner in the Create Diagram menu. A list opens. Select **Activity** and give the activity a name. Push **Ok** and the new activity diagrams opens.

Figure: Executed Subactivity of each Iteration



```
{script = "create changedAddress;
set changedAddress = address.copy();
set changedAddress.ADDRESSTYPE = "CHANGED";"}
```

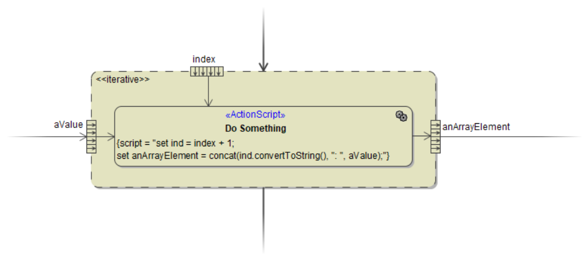
One input Array is allowed and any output Arrays. The parameter for the subactivity must be defined as Expansion Node. The name has to be the used parameter name within the subactivity (**address** in the examples above). Any output Parameter has to be defined as output Expansion Node with the used name out of the subactivity. Like in the other iterations a mapping by input / output Pins is possible.

Accessing the Iteration Index

For all above mentioned usages of iterations, you can access the index value of the iteration. To do that, define an input node to the expansion region that

- is of type Integer
- has no incoming object flows

Such input nodes will be treated as indices by the Runtime. These indices are zero-based, and you can have as many as you need. The sample activity snipped below shows a simple index usage.



```
{script = "set ind = index + 1;
set anArrayElement = concat(ind.convertToString(), ": ", aValue);"}
```