

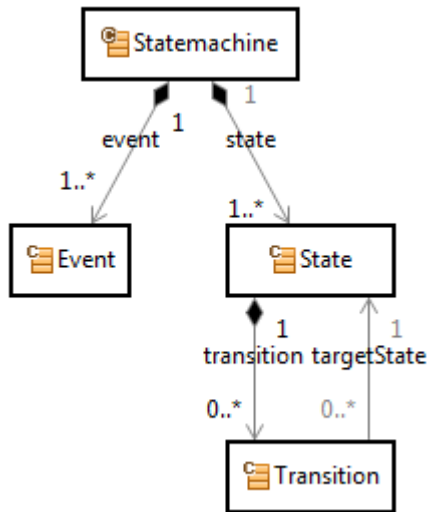


# Tutorial

Code Snippet

Tutorial	Actifsource Tutorial – Code Snippet
Required Time	<ul style="list-style-type: none"> <li>• 120 Minutes</li> </ul>
Prerequisites	<ul style="list-style-type: none"> <li>• Actifsource Tutorial – Installing Actifsource</li> <li>• Actifsource Tutorial – Simple Service</li> <li>• Actifsource Tutorial – Complex Service</li> <li>• Actifsource Tutorial – Statemachine</li> </ul>
Goal	<ul style="list-style-type: none"> <li>• Add conditional expression to a transition</li> <li>• Add code as an action to a transition which will be executed together with the transition</li> </ul>
Topics covered	<ul style="list-style-type: none"> <li>• Code Snippet Editor</li> <li>• Code templates to generate code from code snippets</li> </ul>
Notation	<ul style="list-style-type: none"> <li>☞ To do</li> <li>ⓘ Information</li> <li>• <b>Bold</b>: Terms from actifsource or other technologies and tools</li> <li>• <b><u>Bold underlined</u></b>: actifsource Resources</li> <li>• <u>Underlined</u>: User Resources</li> <li>• <u><i>UnderlinedItalics</i></u>: Resource Functions</li> <li>• Monospaced: User input</li> <li>• <i>Italics</i>: Important terms in current situation</li> </ul>
Disclaimer	<p>The authors do not accept any liability arising out of the application or use of any information or equipment described herein. The information contained within this document is by its very nature incomplete. Therefore, the authors accept no responsibility for the precise accuracy of the documentation contained herein. It should be used rather as a guide and starting point.</p>
Contact	<p><b>actifsource GmbH</b>  Täfernstrasse 37  5405 Baden-Dättwil  Switzerland  <a href="http://www.actifsource.com">www.actifsource.com</a></p>
Trademark	<p><b>actifsource</b> is a registered trademark of <b>actifsource GmbH</b> in Switzerland, the EU, USA, and China. Other names appearing on the site may be trademarks of their respective owners.</p>
Compatibility	<p>Created with actifsource Version 5.8.5</p>

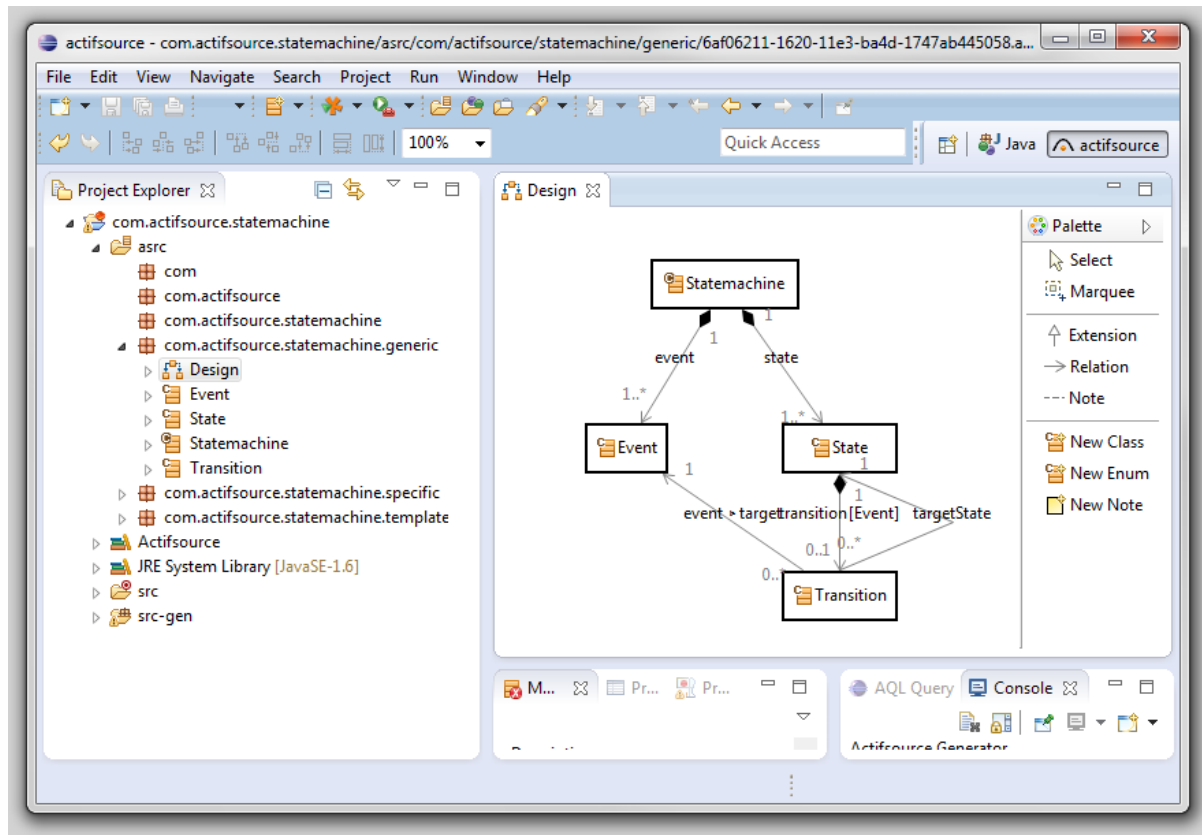
- Create a meta-model for Statemachine, instantiate Statemachine and write a code template to generate a simple implementation of your Statemachines in C++ as seen in the Actifsource Tutorial – Statemachine



- Add a Code Snippets with conditional expressions to Transition
- Create a Domain Diagram for an instance of Statemachine and show the conditions for Transitions in the diagram
- Generate code for Statemachines that checks the associated condition before executing a transition
- Add a Code Snippet with an action to Transition and generate code that executes the action together with the state transition

# Part I: Preparation

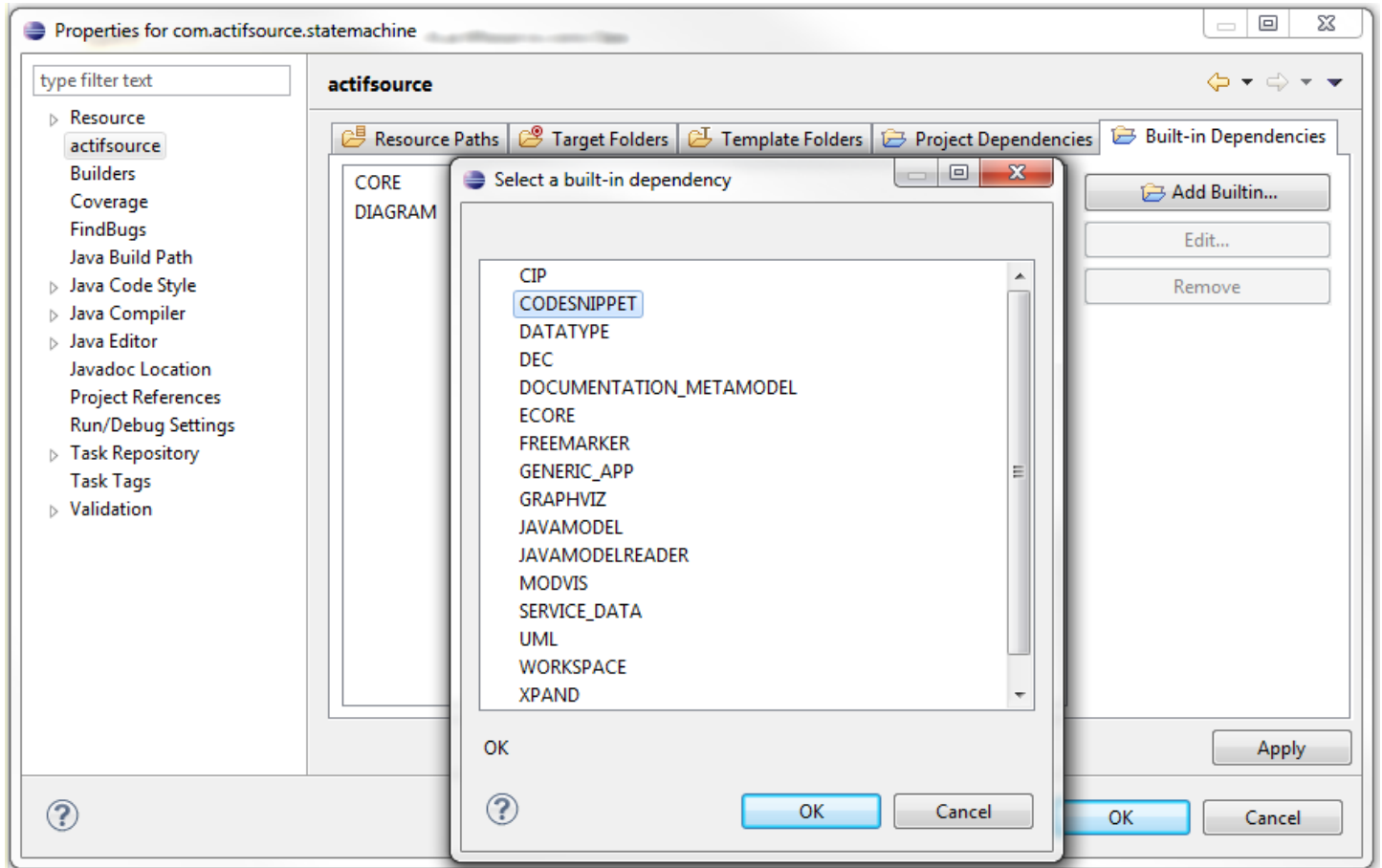
4



Setup a project `com.actifsource.statemachine` with a meta-model for Statemachine, create two instances of Statemachine and implement a code template for Statemachine as seen in the *Actifsource Tutorial – Statemachine*:

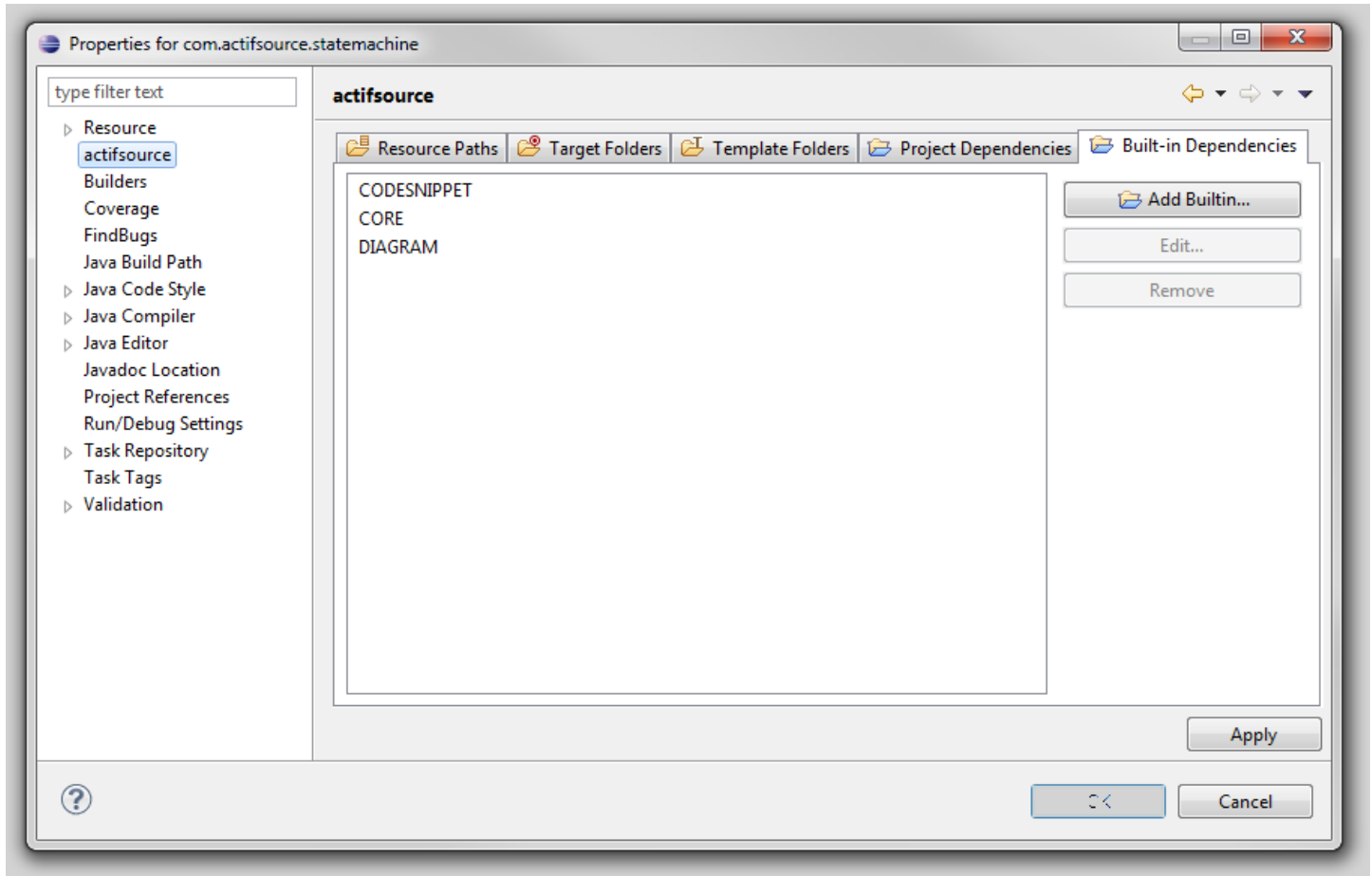
- ↪ Prepare a new **Actifsource Project** `com.actifsource.statemachine`
- ↪ Create a meta-model for Statemachine
- ↪ Create two instances of type Statemachine, namely Statemachine1 and Statemachine2
- ↪ Write a code template StatemachineImpl for the type Statemachine
- ↪ Use the package structure shown above

## Implement conditional transitions

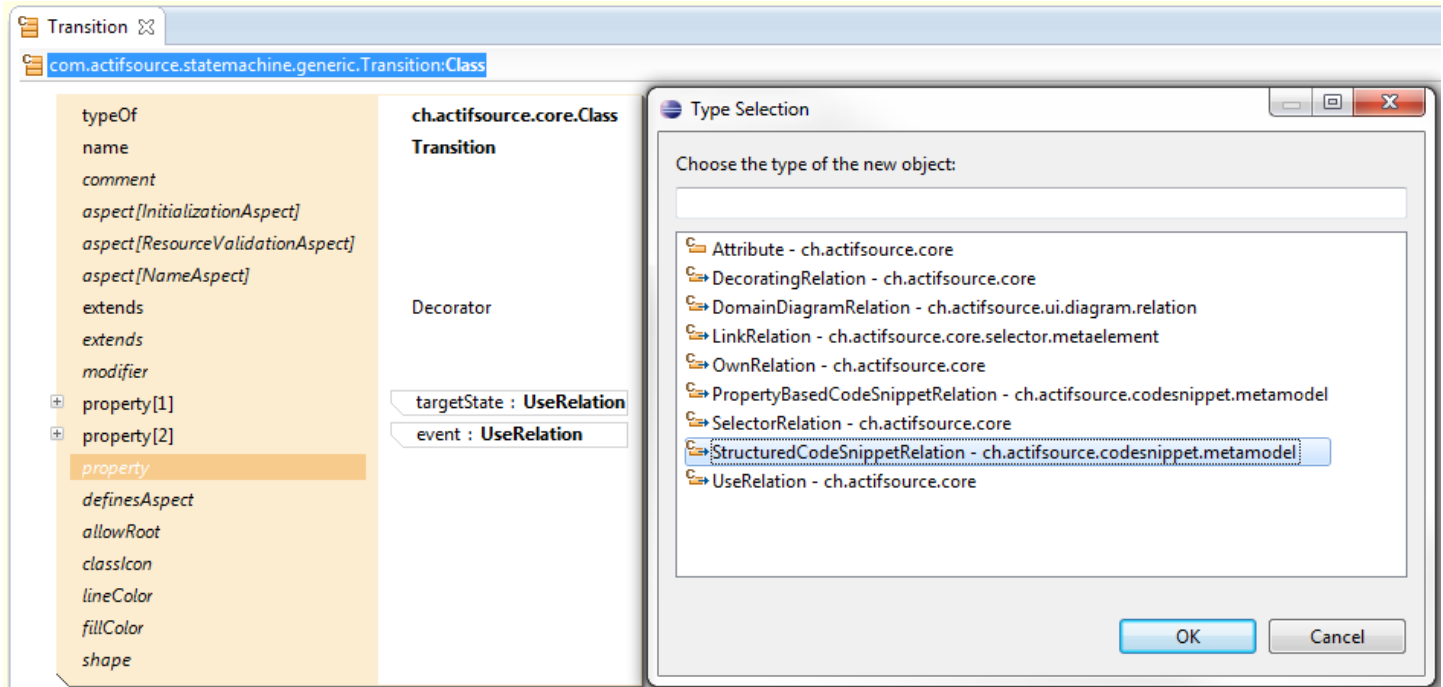


First, we import all the resources needed to enable the support of Code Snippets in our project:

- ↗ Select the project `com.actifsource.statemachine` and choose **Project->Properties** from the main menu
- ↗ In the properties dialog choose **actifsource** and go the **Built-in Dependencies** tab
- ↗ Click on **Add Builtin**
- ↗ In the dialog **Select a built-in dependency** choose **CODESNIPPET** and click on **OK**



↵ Close the Properties dialog by clicking on **OK**



We add a Code Snippet relation to class Transition:

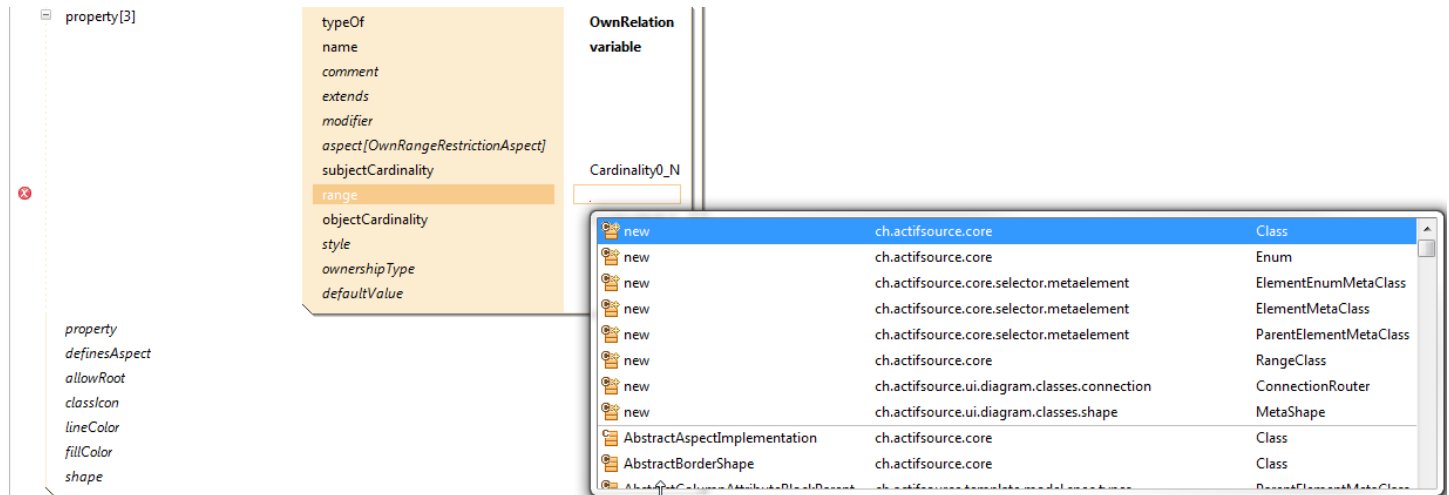
- ↩ Open the class Transition in the Resource Editor
- ↩ Add a (third) property to the class
- ↩ Choose StructuredCodeSnippetRelation in the dialog **Type Selection**

The screenshot shows the configuration of a property named `property[3]`. The configuration is divided into several sections:

- property[3]**: A list of aspects including `OwnRangeRestrictionAspect`, `CodeSnippetRelationAspect`, `CodeSnippetRelationAspect`, `subjectCardinality`, `range`, `objectCardinality`, `style`, `ownershipType`, `defaultValue`, `language` (highlighted), and `token`.
- StructuredCodeSnippetRelation**: A structured relation configuration with the following fields:
  - `condition`: (empty)
  - `implements`: `CodeSnippetRelationAspect`
  - `className`: `ch.actifsource.codesnippet.metamodel.aspect.impl.StructuredCodeSnippetRelationAspect`
  - `subjectCardinality`: `Cardinality0_1`
  - `objectCardinality`: `Cardinality0_1`
- Content Assist Menu**: A dropdown menu for the `language` property showing the following options:
  - `new`: `ch.actifsource.codesnippet.metamodel.language CodeSnippetLanguage`
  - `CMinus`: `ch.actifsource.codesnippet.language CodeSnippetLanguage`
  - `CMinusCondition`: `ch.actifsource.codesnippet.language CodeSnippetLanguage`** (highlighted)
  - `Text`: `ch.actifsource.codesnippet.language CodeSnippetLanguage`

- ↪ Insert `condition` as name of the relation
- ↪ Create a new `CodeSnippetRelationAspect` as shown above
- ↪ Choose `Cardinality0_1` as `subjectCardinality` and as `objectCardinality`
- ↪ Create a new language statement by calling the Content Assist and choosing `CMinusCondition`





We add a new OwnRelation called variable to the class Statemachine:

- ↵ Open Transition in the ResourceEditor
- ↵ Add a new property and choose OwnRelation in the **Type Selection** dialog
- ↵ Insert `variable` as the name of the relation
- ↵ Choose `new ch.actifsource.core.Class` as the range with the support of the Content Assist

The image shows two side-by-side software editors. The left editor is titled 'Statemachine' and shows the definition of a class 'com.actifsource.statemachine.generic.StateMachine:Class'. The right editor is titled 'Variable' and shows the definition of a class 'com.actifsource.statemachine.generic.Variable:Class'.

**Statemachine Editor:**

- Left Panel (Properties):** Includes 'name', 'comment', 'aspect' (with values [InitializationAspect], [ResourceValidationAspect], [NameAspect]), 'extends', 'modifier', and 'property' (with values [1], [2], [3]).
- Right Panel (Class Definition):** Shows 'ch.actifsource.core.Class Statemachine' and 'ch.actifsource.core.NamedResource'.
- Bottom Panel (OwnRelation):** Shows 'event : OwnRelation' and 'state : OwnRelation'. Below this, a table lists properties:
 

typeOf	OwnRelation
name	variable
comment	
extends	
modifier	
aspect [OwnRangeRestrictionAspect]	
subjectCardinality	Cardinality0_N
range	com.actifsource.statemachine.generic.Variable
objectCardinality	Cardinality0_1

**Variable Editor:**

- Left Panel (Properties):** Includes 'name', 'comment', 'aspect' (with values [InitializationAspect], [ResourceValidationAspect], [NameAspect]), 'extends', 'modifier', 'property', 'definesAspect', 'allowRoot', 'classIcon', 'lineColor', 'fillColor', and 'shape'.
- Right Panel (Class Definition):** Shows 'ch.actifsource.core.Class Variable' and 'ch.actifsource.core.NamedResource'.

➤ Insert Variable as name of the newly created class, which is opened automatically in the Resource Editor

property[3]

typeOf  
 name  
 comment  
 extends  
 modifier  
 aspect[OwnRangeRestrictionAspect]  
 aspect[CodeSnippetRelationAspect]  
 aspect[CodeSnippetRelationAspect]  
 subjectCardinality  
 range  
 objectCardinality  
 style  
 ownershipType  
 defaultValue  
 language  
 token  
 token

**StructuredCodeSnippetRelation**

**condition**

CodeSnippetRelationAspect [1] : JavaAspectImplementation

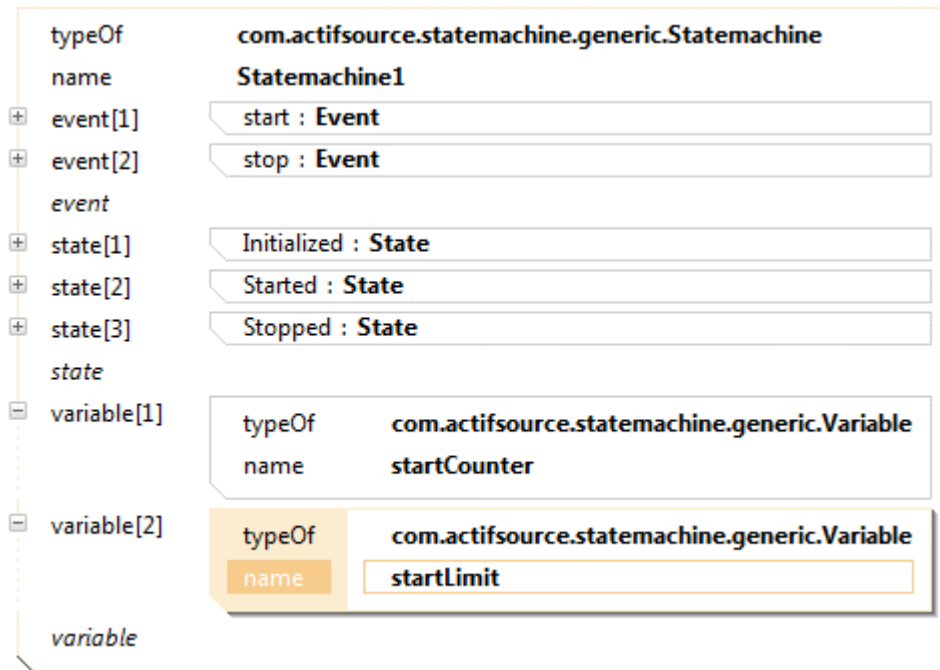
Cardinality0\_1  
 ch.actifsource.codesnippet.metamodel.element.CodeSnippet  
 Cardinality0\_1

CMinusCondition

typeOf **ch.actifsource.codesnippet.metamodel.RelationTokenProvider**  
 selector Transition.-transition.-state.variable  
 tokenType ch.actifsource.codesnippet.metamodel.TokenType.Variable  
 subtoken

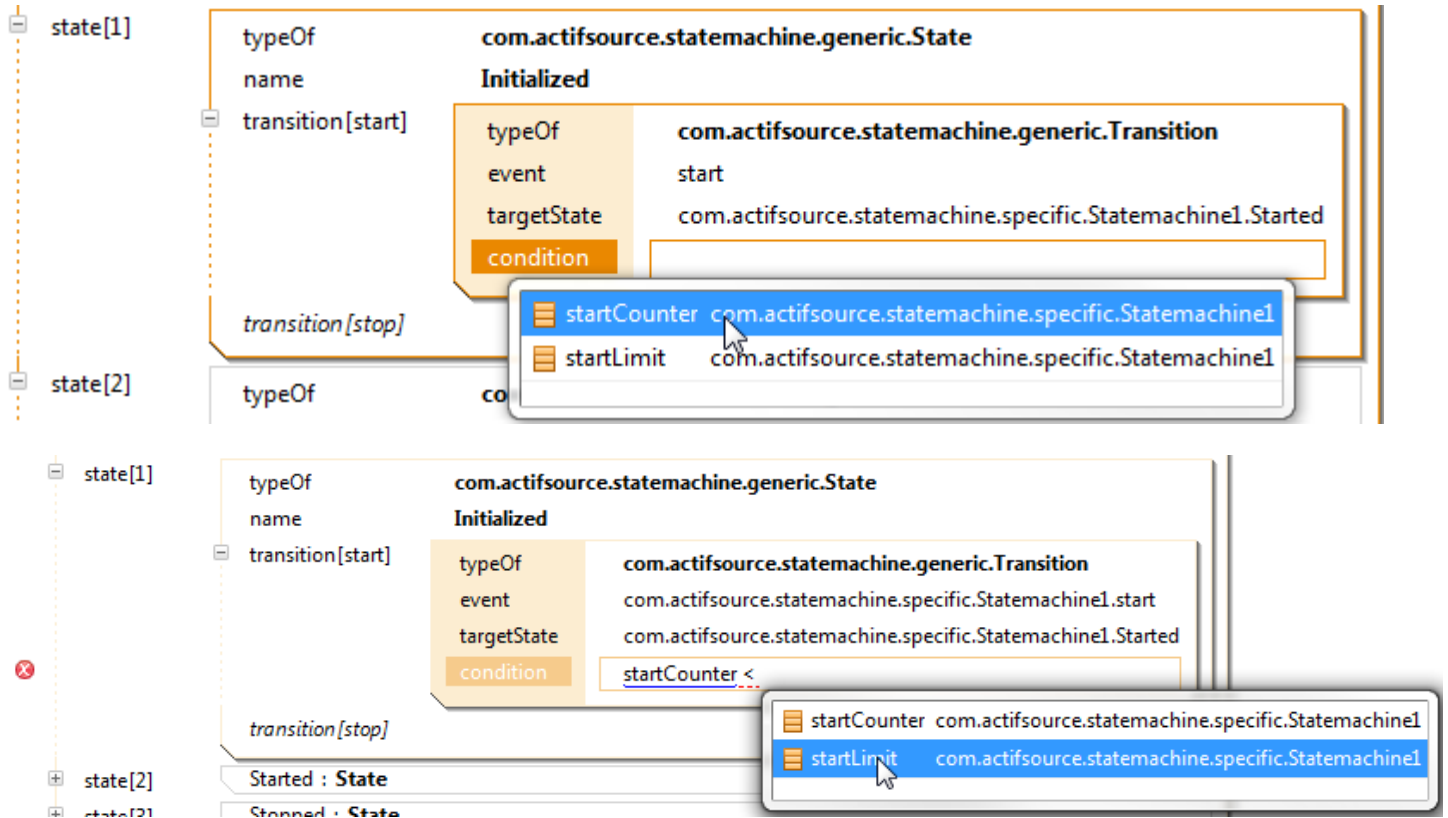
A conditional expression for a Transition should be able to use all Variables that are owned by the Statemachine of the Transition as variables:

- ↪ Insert a RelationTokenProvider
- ↪ Define the Selector Transition.-transition.-state.counter which selects all Variables that are owned by the Statemachine of the Transition
- ↪ Choose ch.actifsource.codesnippet.metamodel.TokenType.Variable as tokenType



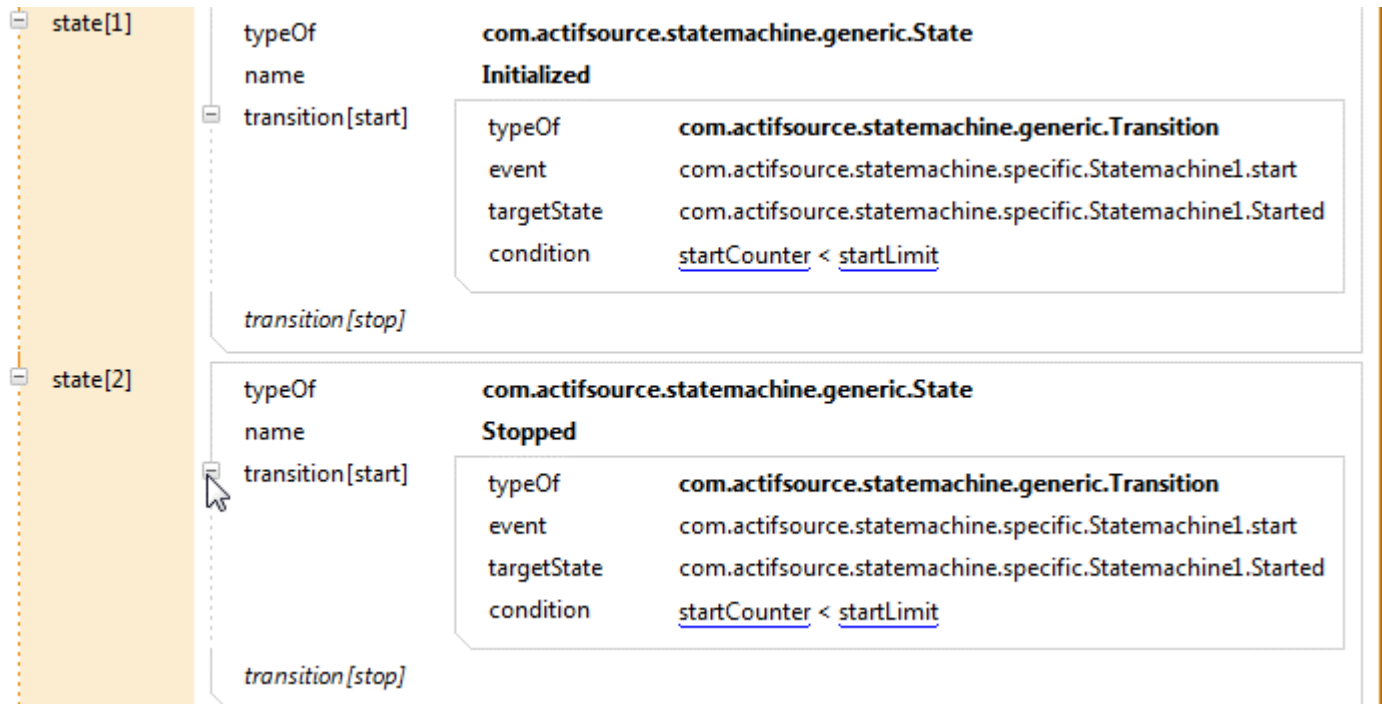
We add a condition to the state transition triggered by the event start that restricts the number of times the StateMachine can switch to the State Started:

- ↪ Open the instance StateMachine1 in the ResourceEditor
- ↪ Add two statements for variable to StateMachine1
- ↪ Choose `startCounter` as the name of the first and `startLimit` as the name of the second Variable



We add a conditional expression to the state transition triggered by the Event `start` in the State `Initialized`:

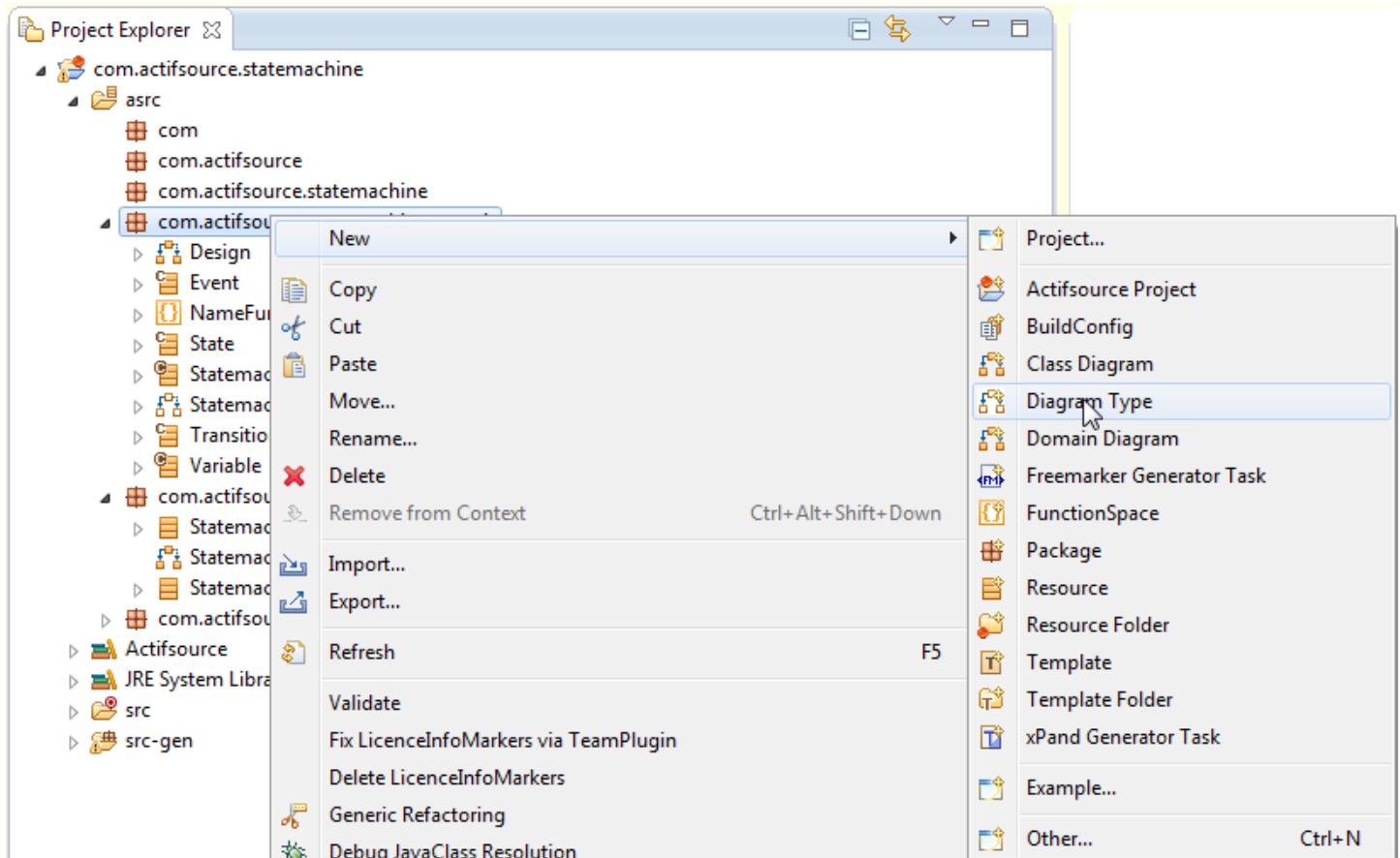
- ↵ Open the state `Initialized` in the Resource Editor and then open its Transition `start`
- ↵ Open the Code Snippet Editor by selecting the relation `condition` and pressing **Enter**
- ↵ Call the **Content Assist** in the Code Snippet Editor and choose `startCounter`
- ↵ Enter ' < ' in the Code Snippet Editor and call the Content Assist again
- ↵ Choose `startLimit` in the **Content Assist**



We enforce that the State can only switch a maximum of startLimit times to the State Started by adding the same conditional expression also to the Transition triggered by the Event start in the State Stopped:

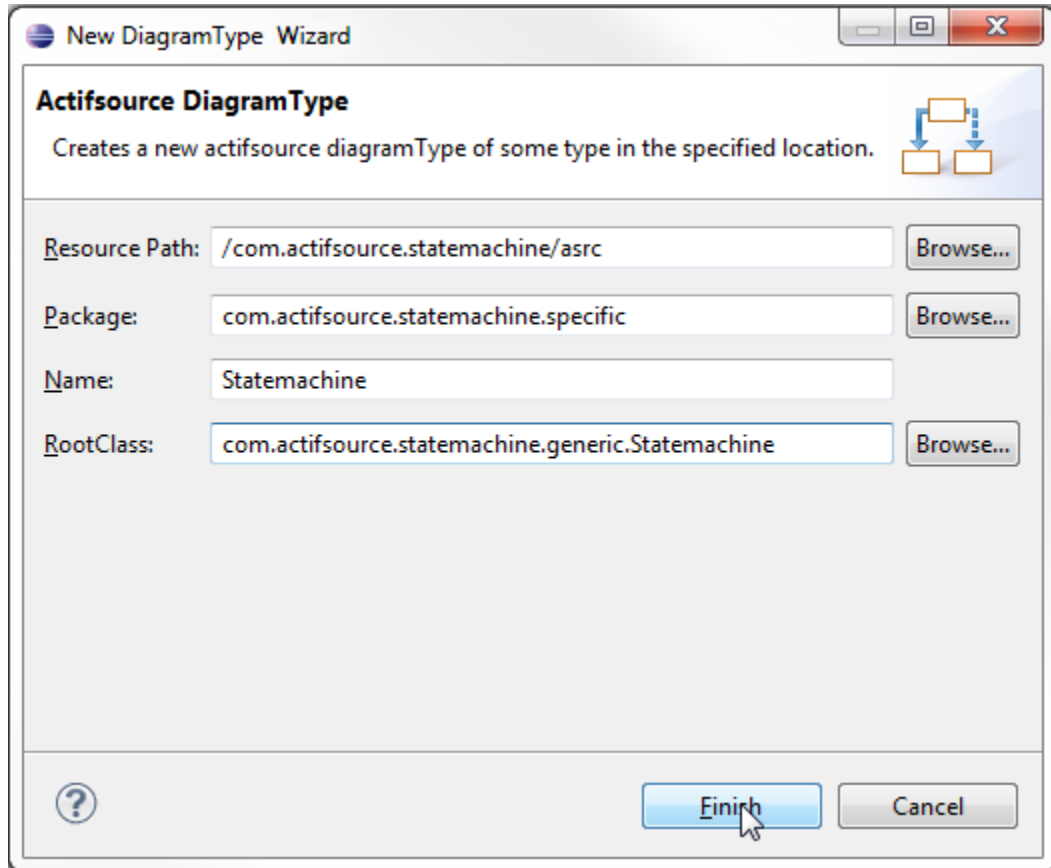
↩ Enter the conditional expression in the same way as on the previous page

# Create a Domain Diagram



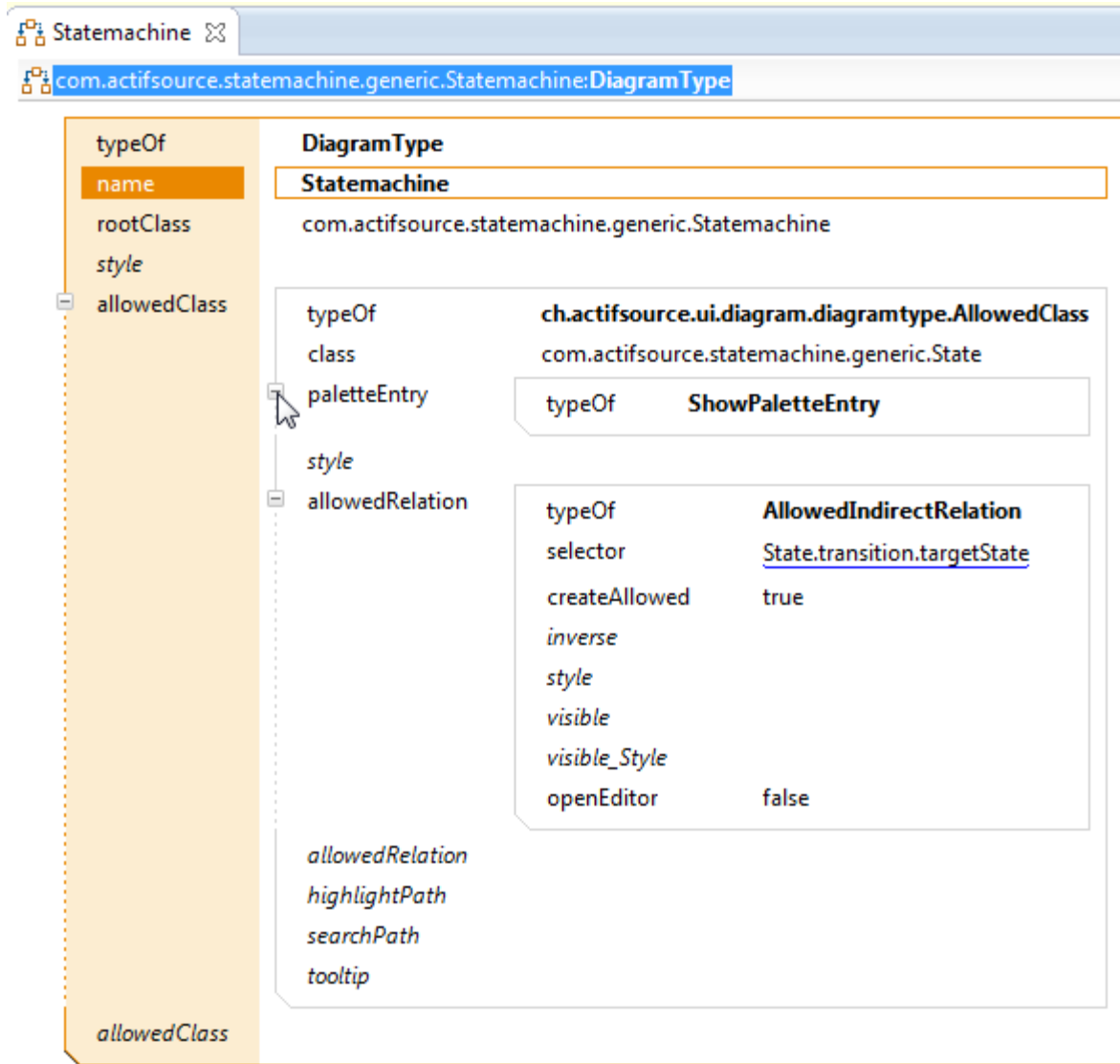
We create a new Diagram Type called Statemachine in order to define properties of Domain Diagrams of Statemachines:

- ↖ Select the package `com.actifsource.statemachine.generic` and choose **New -> Diagram Type**

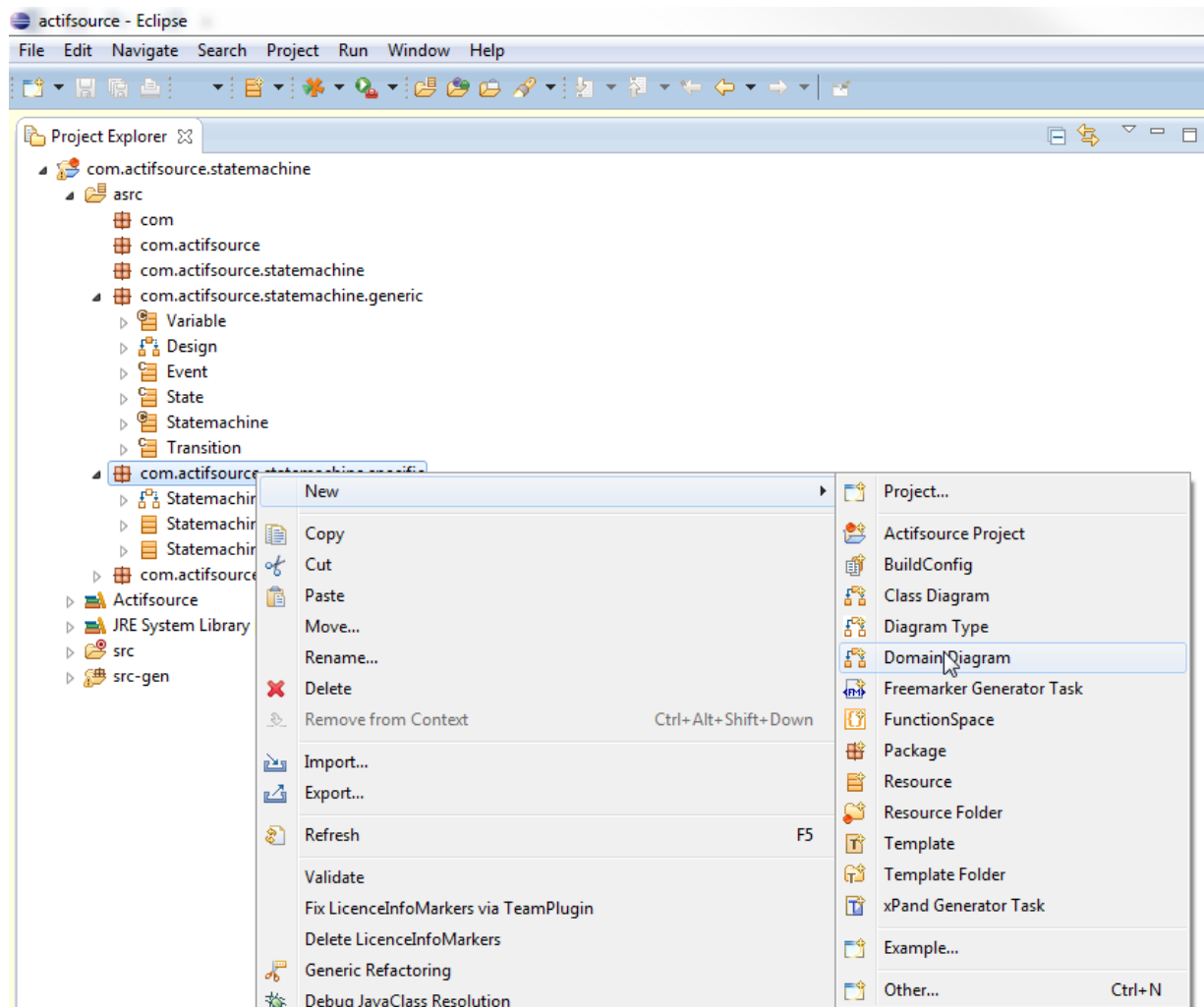


- ↩ Enter `Statemachine` as name for the newly created DiagramType in the **New DiagramType Wizard**
- ↩ Choose `com.actifsource.statemachine.generic.Statemachine` as RootClass



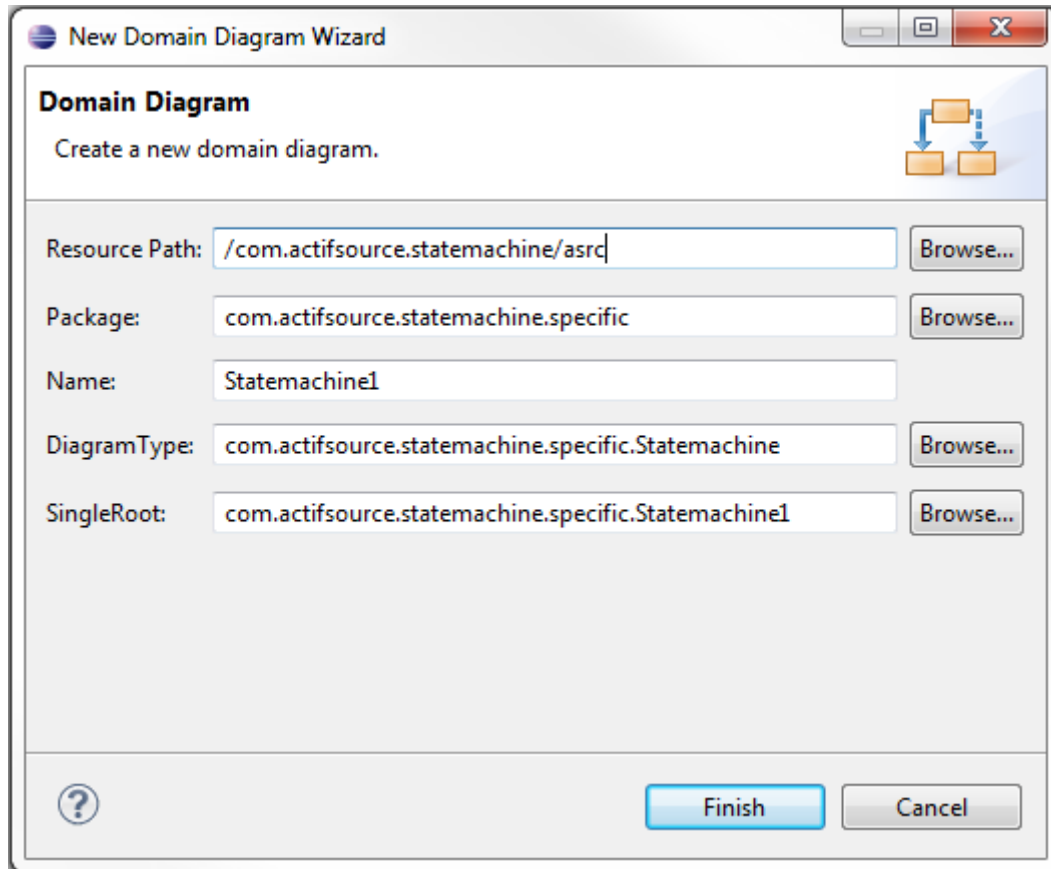


- ↖ Define `allowedClass` as `com.actifsource.statemachine.generic.State`
- ↖ Choose `ShowPaletteEntry` as `paletteEntry`
- ↖ Insert an `allowedRelation` of type `AllowedIndirectRelation` with selector `State.transition.targetState`

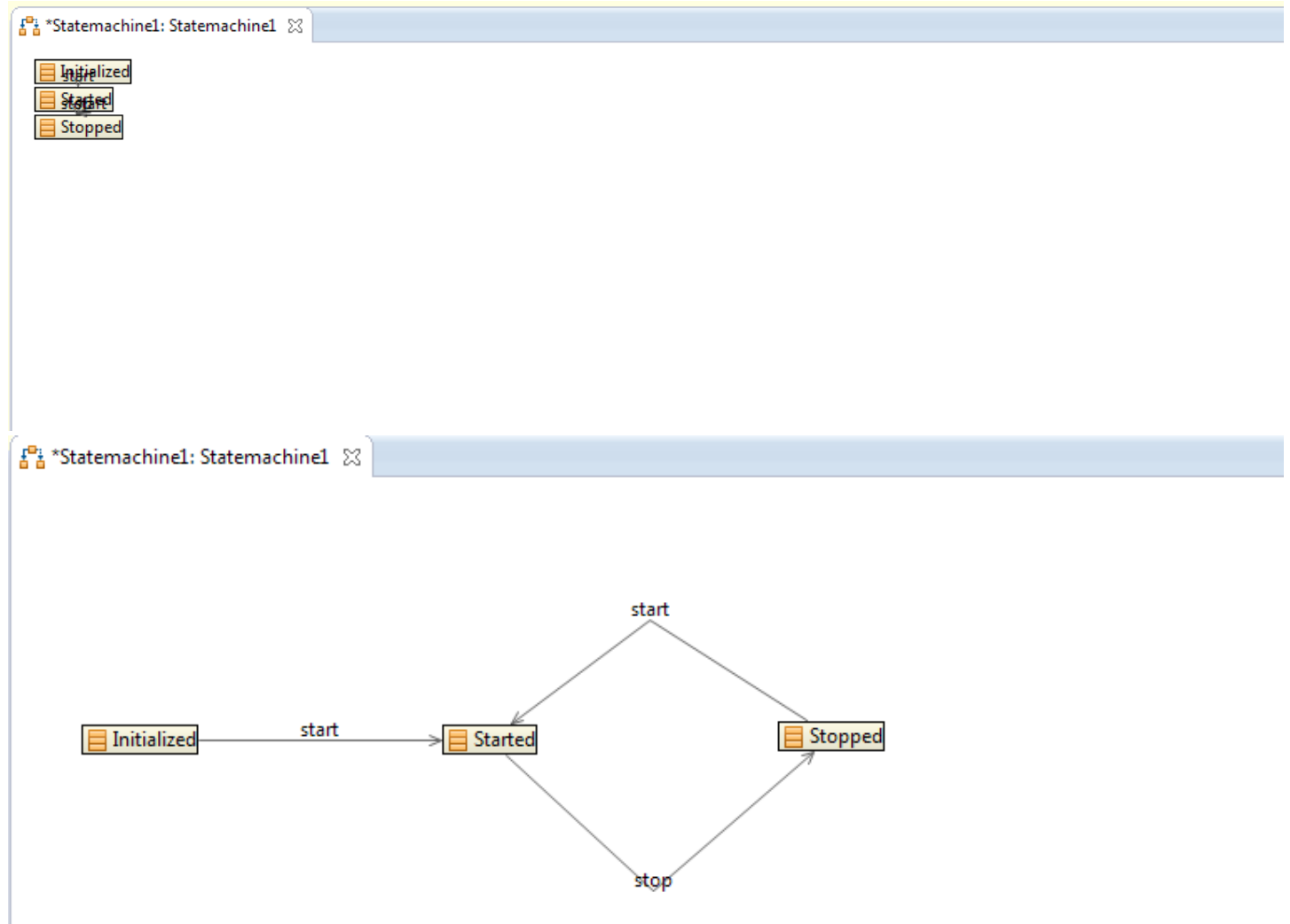


Now we create a new Domain Diagram for Statemachine1 based on the newly created Diagram Type:

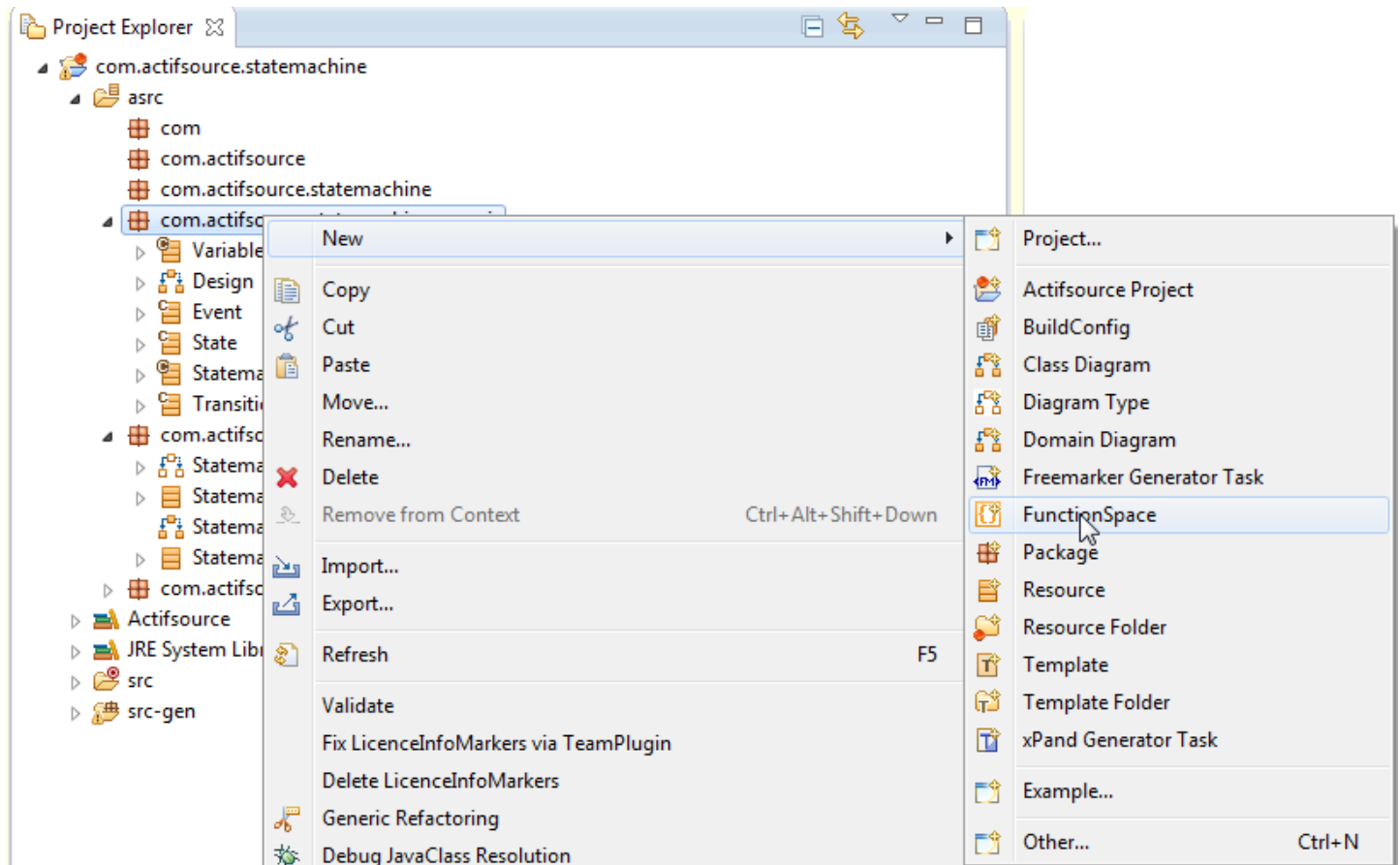
↖ Select `com.actifsource.statemachine.specific` and choose **New -> Domain Diagram**



↩ Check the chosen settings in the open dialog and click **Finish**

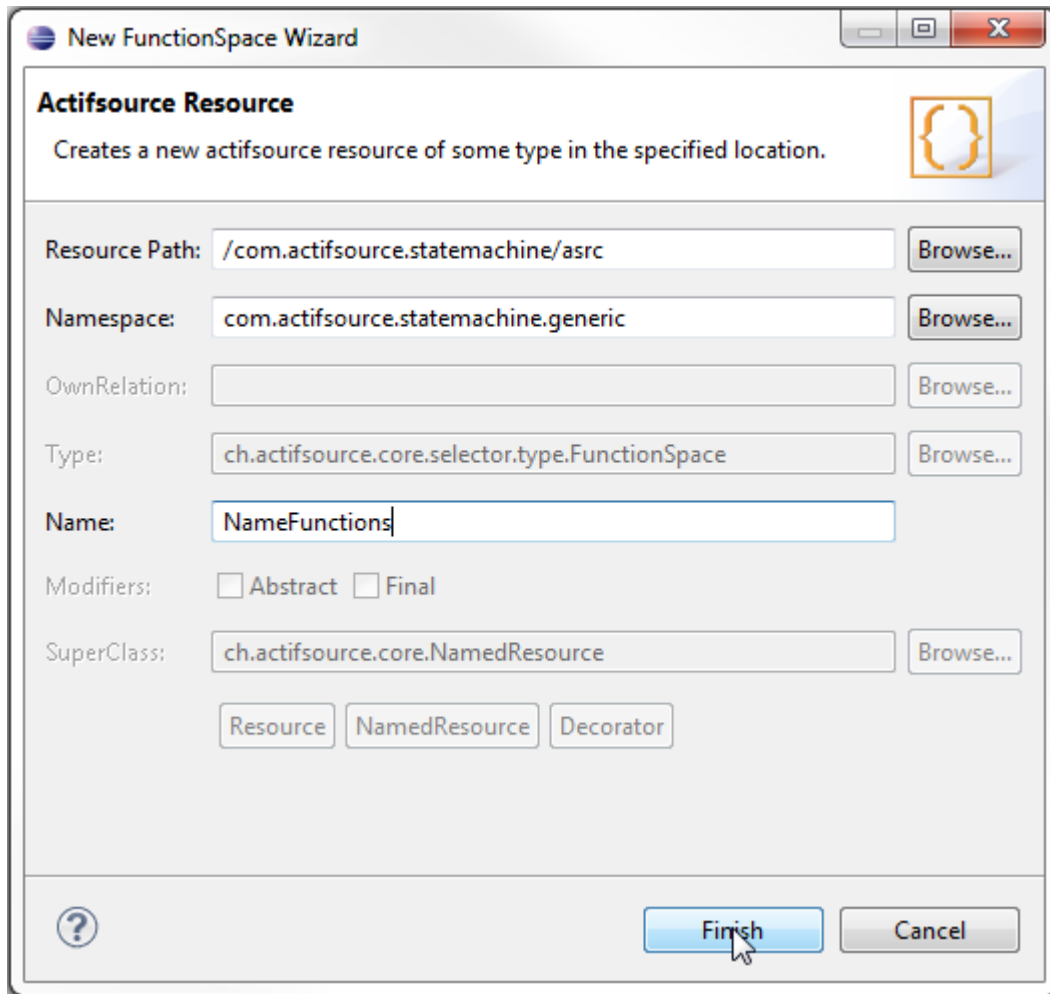


☞ Choose **Select** in the **Palette** of the Diagram Editor and arrange the States such that all transitions are visible

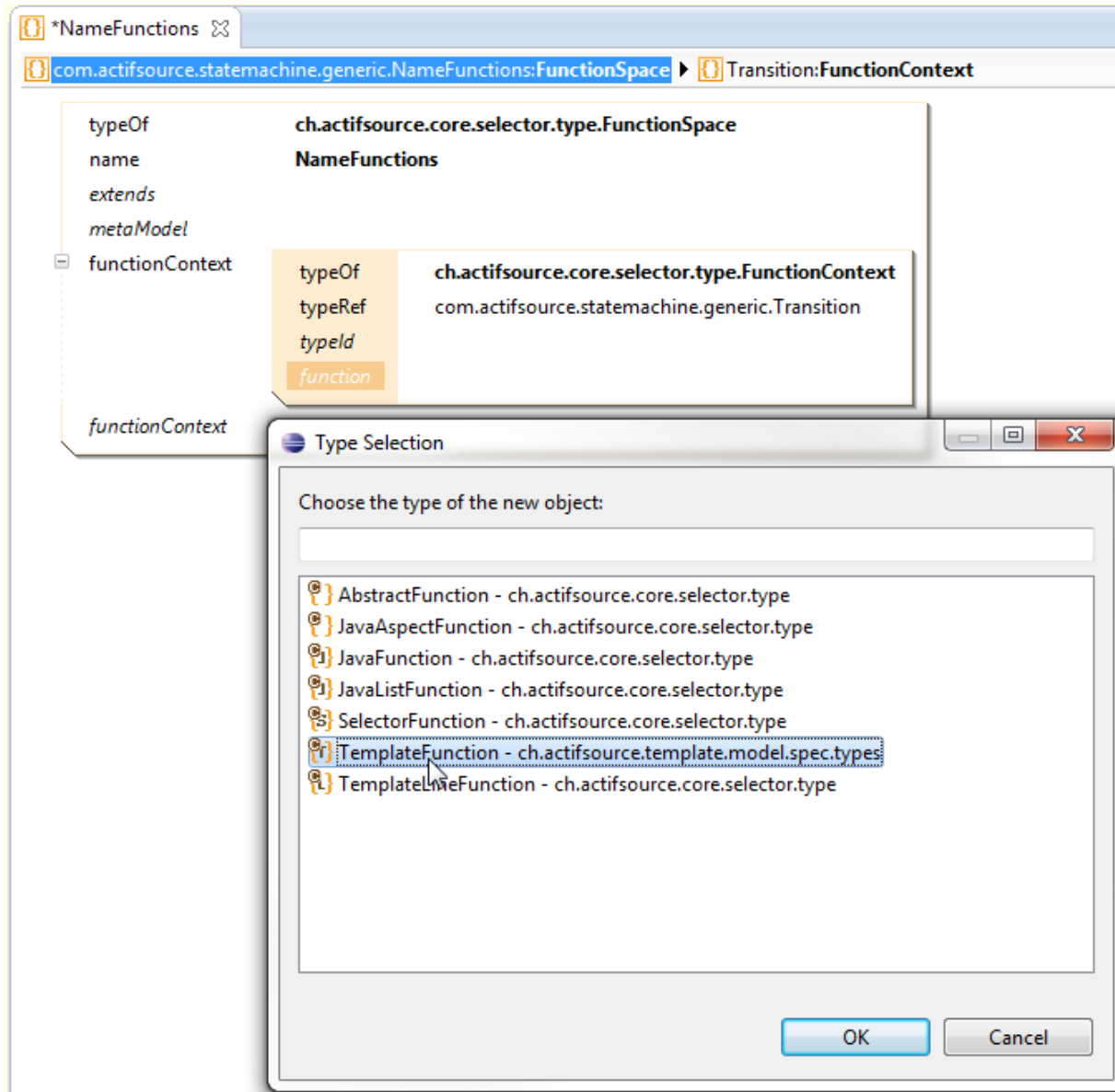


Next, we will see how to display the conditional expressions associated with a Transition in the Domain Diagram. First, we create a function that generates the displayed text from the conditional expression:

- Select the package `com.actifsource.statemachine.generic` and choose **New -> FunctionSpace**



↩ Enter NameFunctions as name of the FunctionSpace



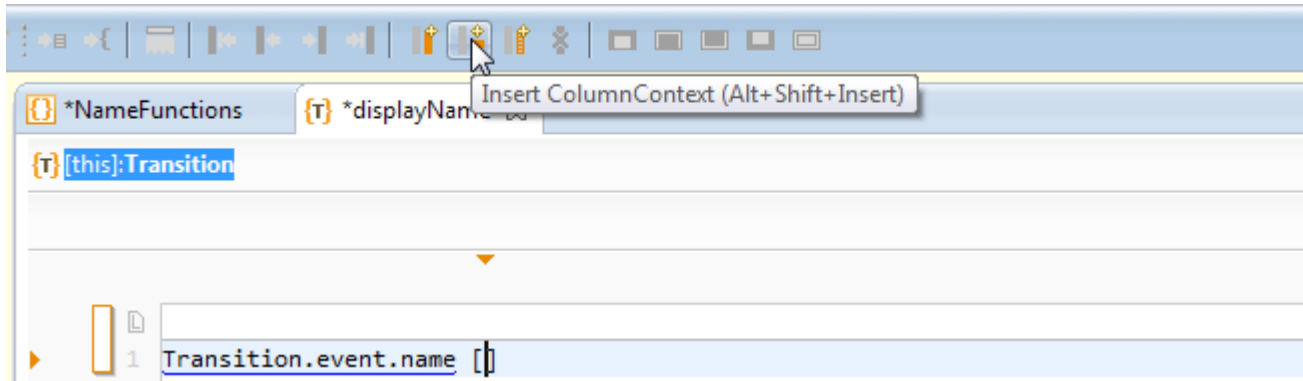
- ↪ Create a new `FunctionContext` with `typeRef com.actifsource.statemachine.generic.Transition`
- ↪ Create a new function in the `FunctionContext` and choose `TemplateFunction` in the **Type Selection** dialog

The screenshot shows an IDE with two main windows. On the left is the 'Project Explorer' showing a package structure for 'com.actifsource.statemachine'. The package 'com.actifsource.statemachine.generic' is expanded, showing sub-packages 'Design', 'Event', 'NameFunctions', and 'Variable'. Under 'NameFunctions', there is a 'functionContext' package containing a 'Transition' package, which in turn contains a 'function' package. The 'function' package contains a class 'displayName'. A mouse cursor is hovering over the 'displayName' class.

On the right is the 'Domain Diagram Editor' showing a class hierarchy. The root class is 'ch.actifsource.core.selector.type.FunctionSpace' with attributes 'name', 'extends', and 'metaModel'. Below it is 'ch.actifsource.core.selector.type.FunctionContext' with attributes 'typeOf', 'typeRef', and 'typed'. Below that is a 'function' class with attributes 'typeOf', 'name', 'comment', 'param', 'children', and 'language'. The 'children' attribute is expanded to show a 'TemplateFunction' class with a 'displayName' attribute. The 'function' class is also expanded to show a 'function' attribute with a value of ': Line'.

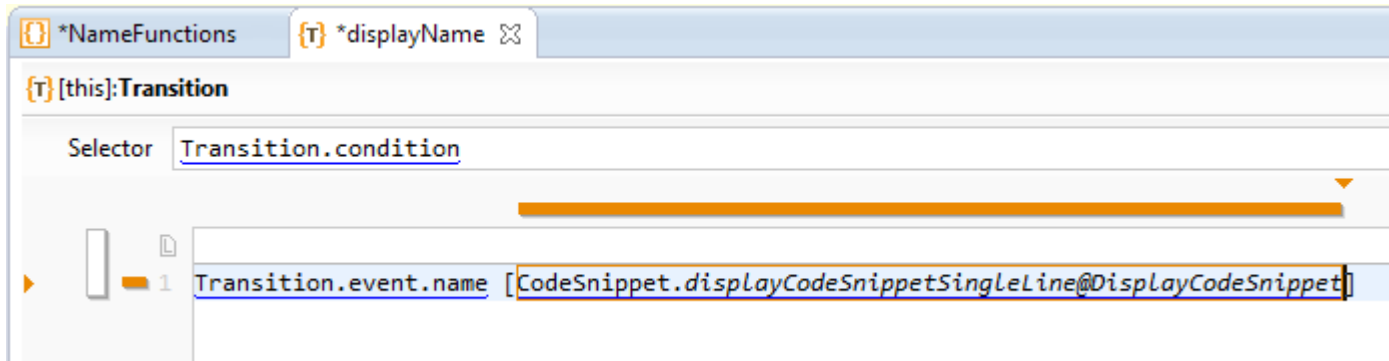
- ↵ Enter `displayName` as name of the TemplateFunction
- ↵ Open the TemplateFunction in the Template Function Editor by double-clicking on displayName in the Project Explorer



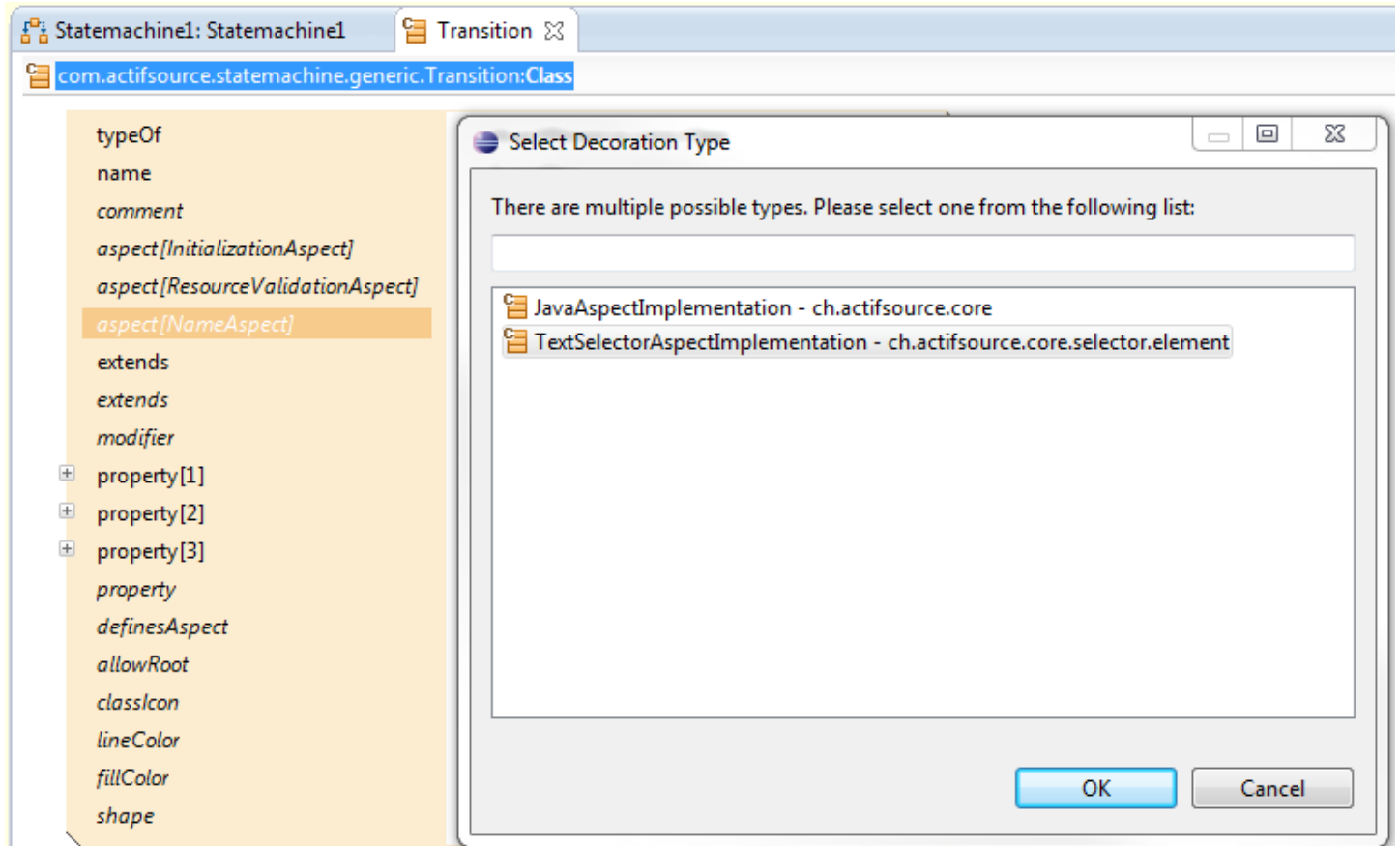


The function should display the name of the event triggering the Transition and in brackets the conditional expression if there is one:

- ↵ Enter Transition.event.name in the TemplateEditor followed by '[']
- ↵ Place the cursor inside the brackets and choose Insert ColumnContext from the menu

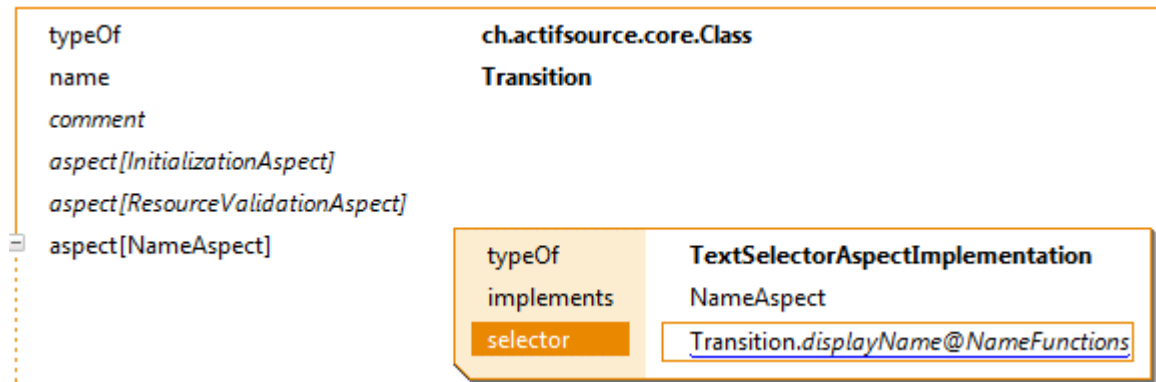


- ↪ Choose Transition.condition as selector for the column context
- ↪ Enter CodeSnippet.displayCodeSnippetSingleLine@DisplayCodeSnippet in the column context. This function outputs a string that represents the expression in the Code Snippet as entered by the user, i.e., without parsing or processing it

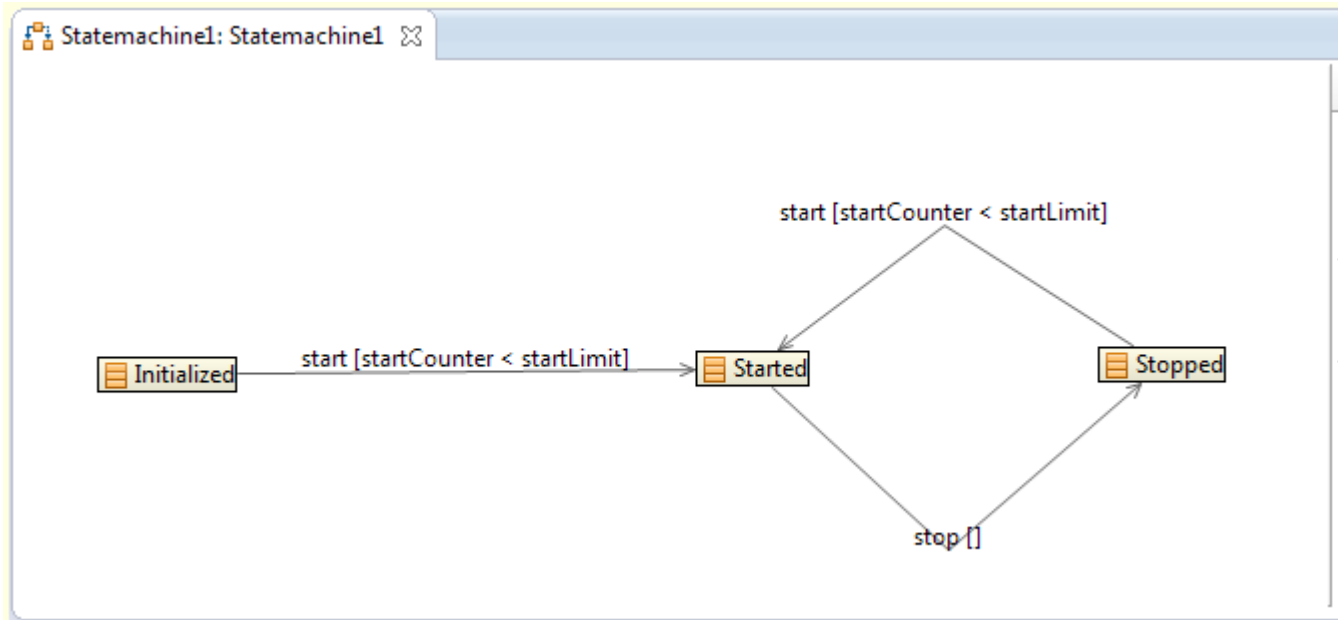


We will use the newly created function `displayName` to define a `NameAspect` for `Transitions`:

- ↪ Create a `NameAspect` with the Content Assist and choose `TextSelectorAspectImplementation` in the **Select Decoration Type** dialog

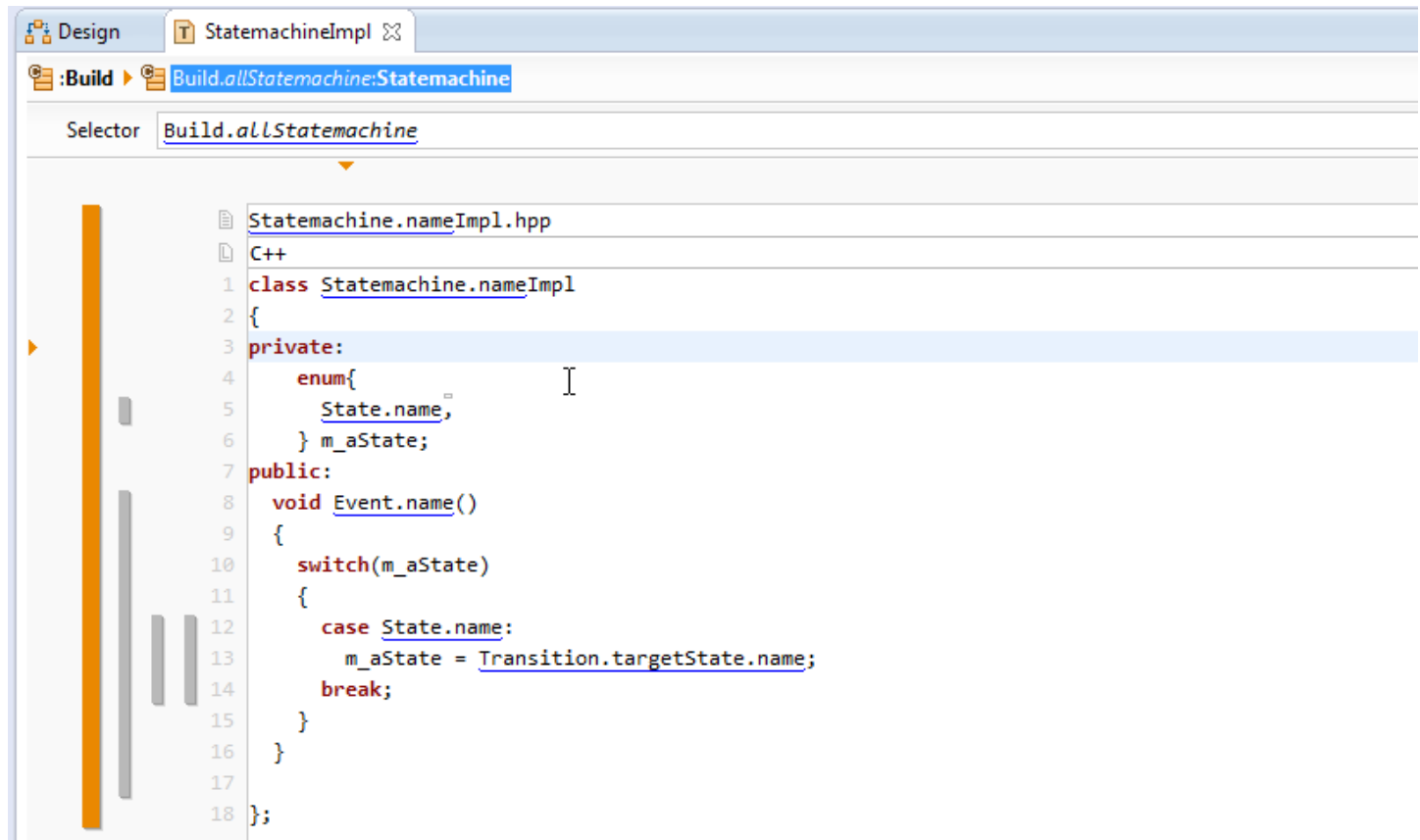


- ↪ Enter `Transition.displayName@NameFunction` as selector of the NameAspect, i.e., the name of the Transition will be the output of the function we have defined before



- ↪ Open the Statemachine1 Domain Diagram again and check that the Transitions are now displayed with their name including the conditional expressions as specified above

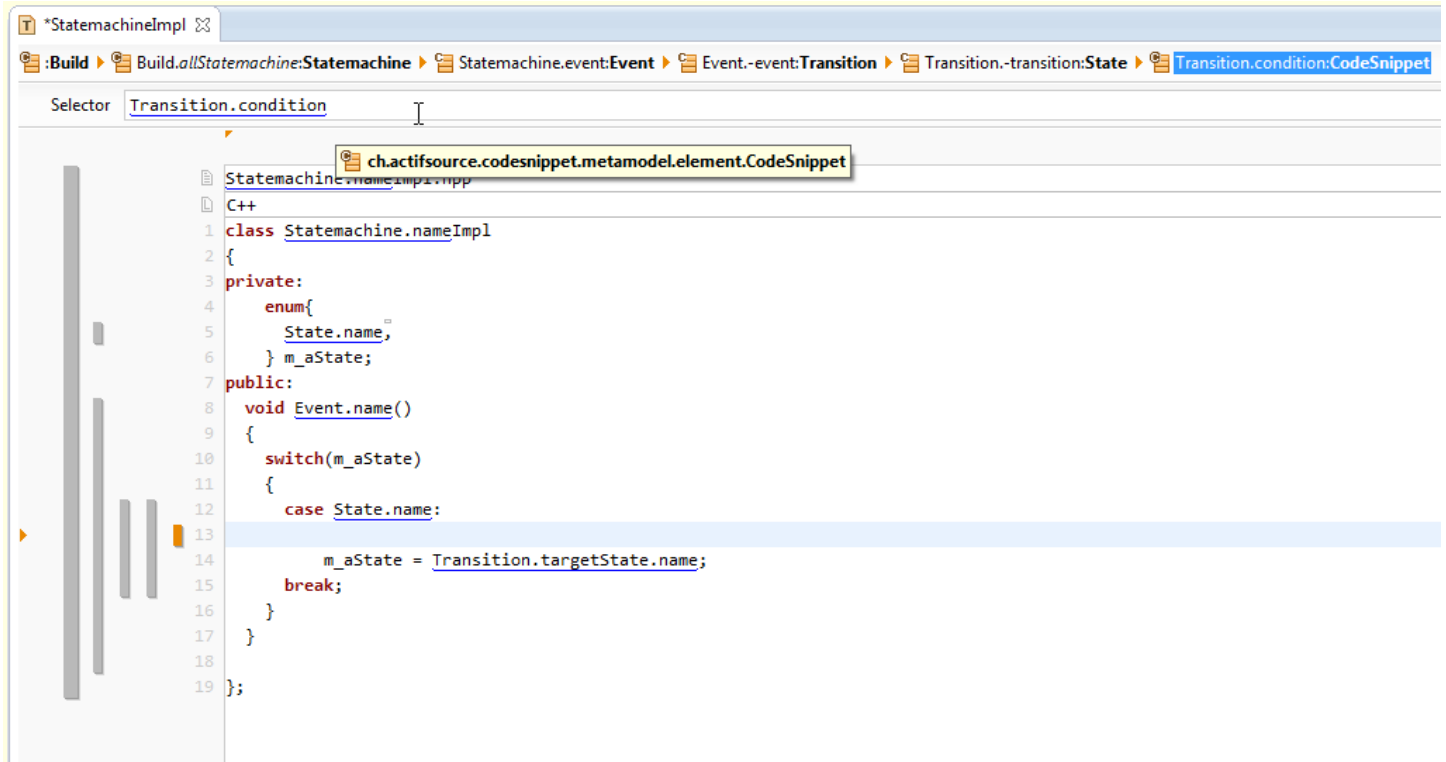
## Generate code for conditional transitions



```
Statemachine.nameImpl.hpp
C++
1 class Statemachine.nameImpl
2 {
3 private:
4     enum{
5         State.name,
6     } m_aState;
7 public:
8     void Event.name()
9     {
10        switch(m_aState)
11        {
12            case State.name:
13                m_aState = Transition.targetState.name;
14            break;
15        }
16    }
17
18 };
```

In this part, we will extend the template `StatemachineImpl` as implemented in the Actifsource Tutorial – State machine such that the state machine executes a state transition only if the associated condition is fulfilled:

↪ Open the template `StatemachineImpl` in the Template Editor

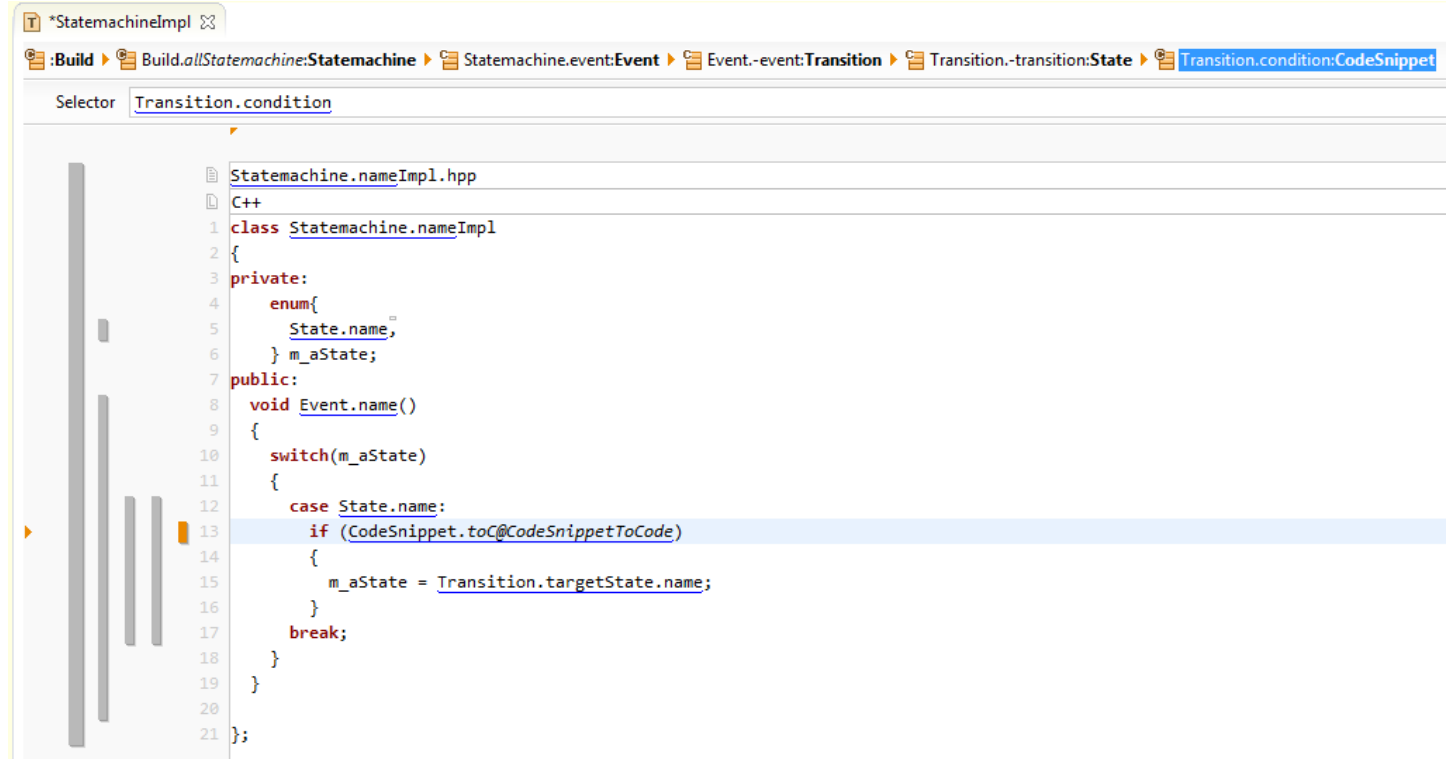


The screenshot shows an IDE window titled '\*StateMachineImpl'. The breadcrumb trail is: Build > Build.allStateMachine:StateMachine > StateMachine.event:Event > Event.-event:Transition > Transition.-transition:State > Transition.condition:CodeSnippet. The 'Selector' field contains 'Transition.condition'. A tooltip for 'ch.actifsource.codesnippet.metamodel.element.CodeSnippet' is visible. The code editor shows the following C++ code:

```
1 class StateMachine.nameImpl
2 {
3 private:
4     enum{
5         State.name,
6     } m_aState;
7 public:
8     void Event.name()
9     {
10        switch(m_aState)
11        {
12            case State.name:
13                m_aState = Transition.targetState.name;
14            break;
15        }
16    }
17 }
18 };
```

We write an if-statement with the conditional expression associated with a Transition as condition:

- ↪ Insert a new Line Context and write the selector Transition.condition for the newly created context



```
StateMachine.nameImpl.cpp
C++
1 class StateMachine.nameImpl
2 {
3 private:
4     enum{
5         State.name,
6     } m_aState;
7 public:
8     void Event.name()
9     {
10        switch(m_aState)
11        {
12            case State.name:
13                if (CodeSnippet.toC@CodeSnippetToCode)
14                {
15                    m_aState = Transition.targetState.name;
16                }
17                break;
18        }
19    }
20 };
21
```

- ↪ Write an if-statement in the new line context and insert a call to the function toC@CodeSnippetToCode on the CodeSnippet available in the this context as condition of the if-statement
- ↪ Add opening and closing braces around the existing assignment expression





```
Statemachine1Impl.hpp
void start()
{
    switch(m_aState)
    {
        case Initialized:
            if (startCounter < startLimit)
            {
                m_aState = Started;
            }
            break;
        case Stopped:
            if (startCounter < startLimit)
            {
                m_aState = Started;
            }
            break;
    }
}

void stop()
{
    switch(m_aState)
    {
        case Started:
            {
                m_aState = Stopped;
            }
            break;
    }
}

Statemachine2Impl.hpp
void open()
{
    switch(m_aState)
    {
        case Initialized:
            {
                m_aState = Opened;
            }
            break;
        case Closed:
            {
                m_aState = Opened;
            }
            break;
    }
}

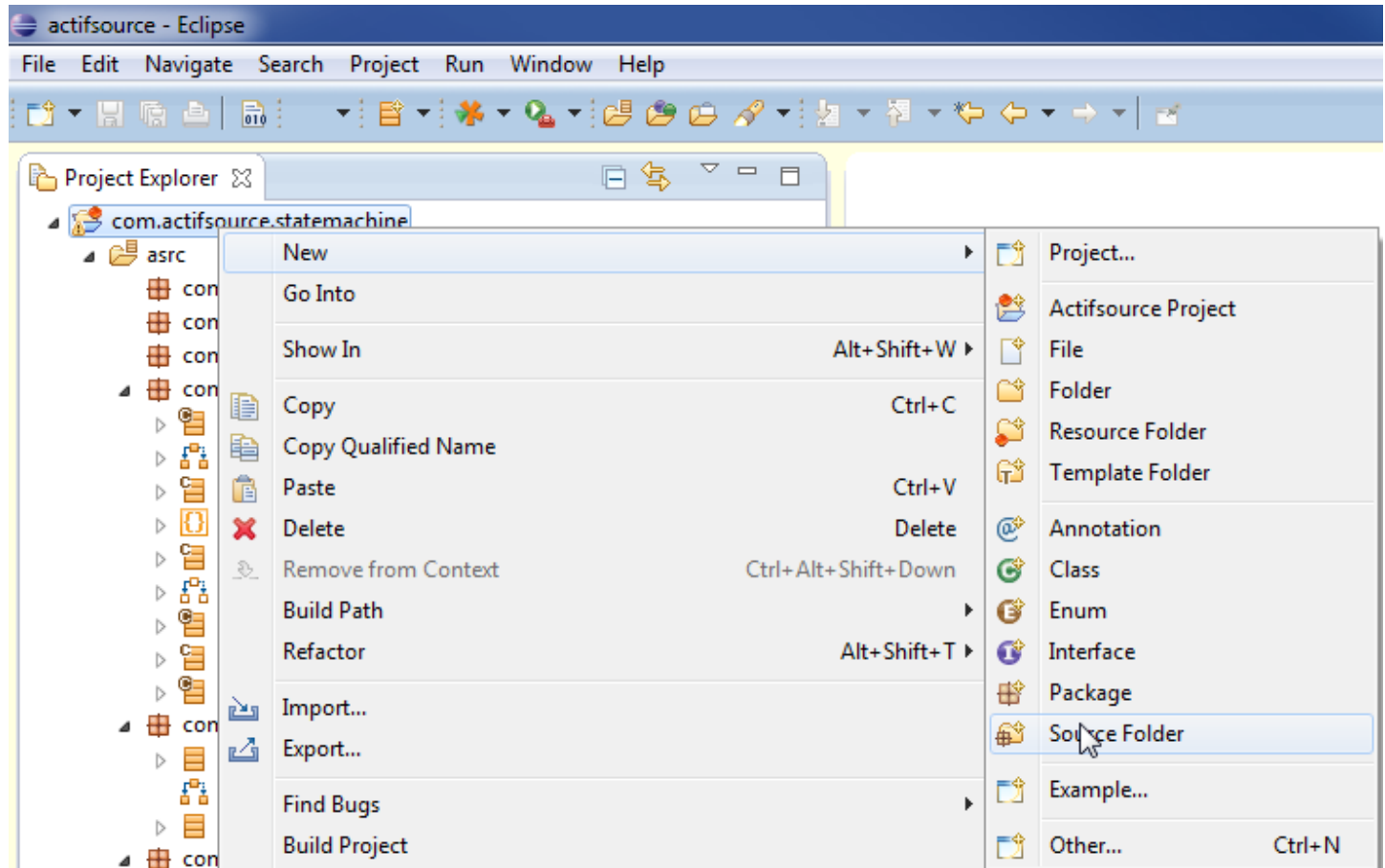
void close()
{
    switch(m_aState)
    {
        case Opened:
            {
                m_aState = Closed;
            }
            break;
    }
}
};
```

- Open the generated files Statemachine1Impl.hpp (overwritten) and Statemachine2Impl.hpp (unchanged) and inspect the changes.

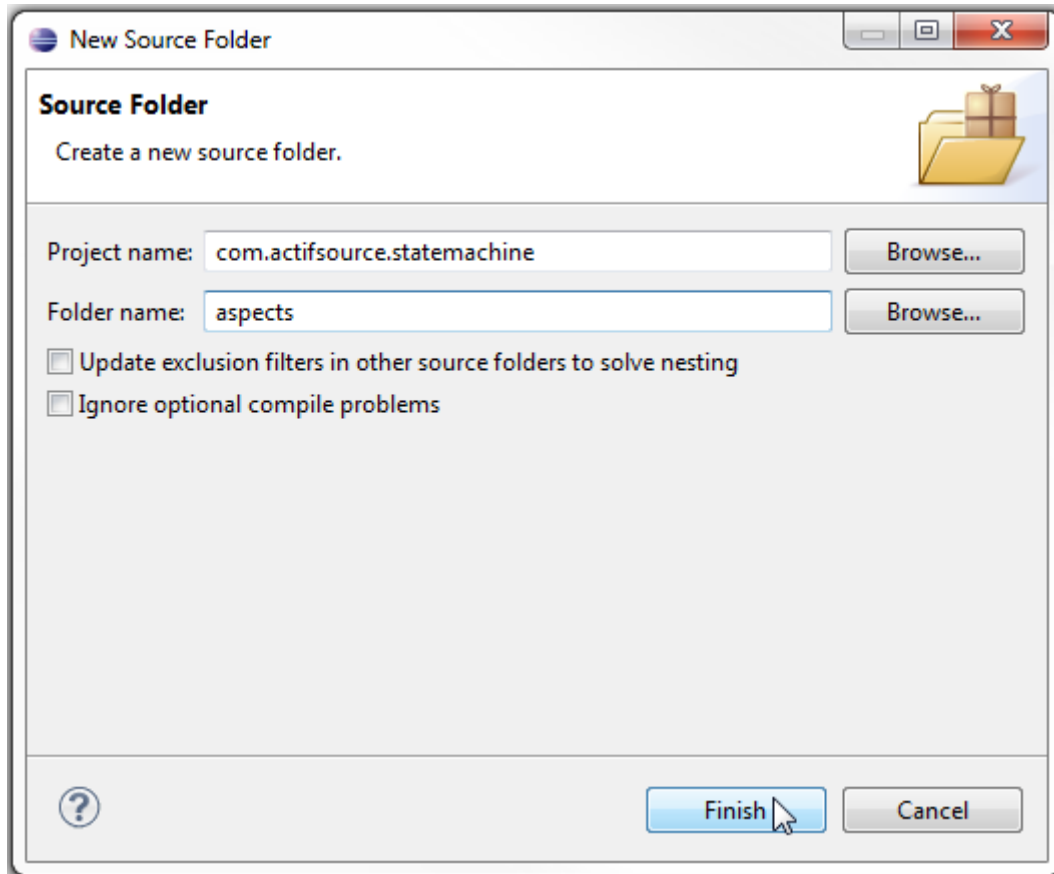
## Customized variable names

```
public:
void start()
{
    switch(m_aState)
    {
        case Initialized:
            if (m_startCounter < m_startLimit)
            {
                m_aState = Started;
            }
            break;
        case Stopped:
            if (m_startCounter < m_startLimit)
            {
                m_aState = Started;
            }
            break;
    }
}
```

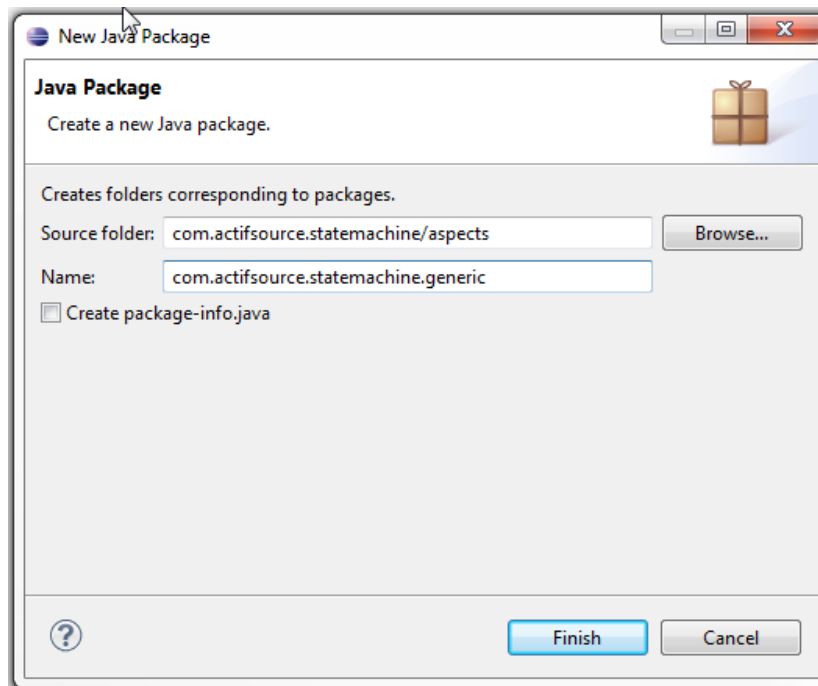
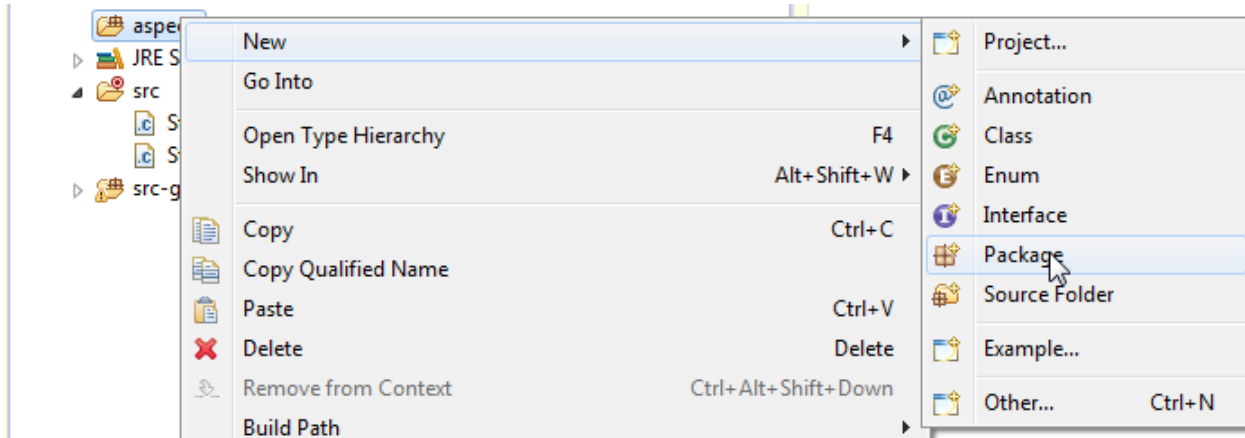
- ① The built-in template functions that we used so far generate the name of variables (and also functions) by calling simple name on the corresponding resource
- ① In this part we will change the generated variable names by adding a prefix to the variable names according to our naming convention



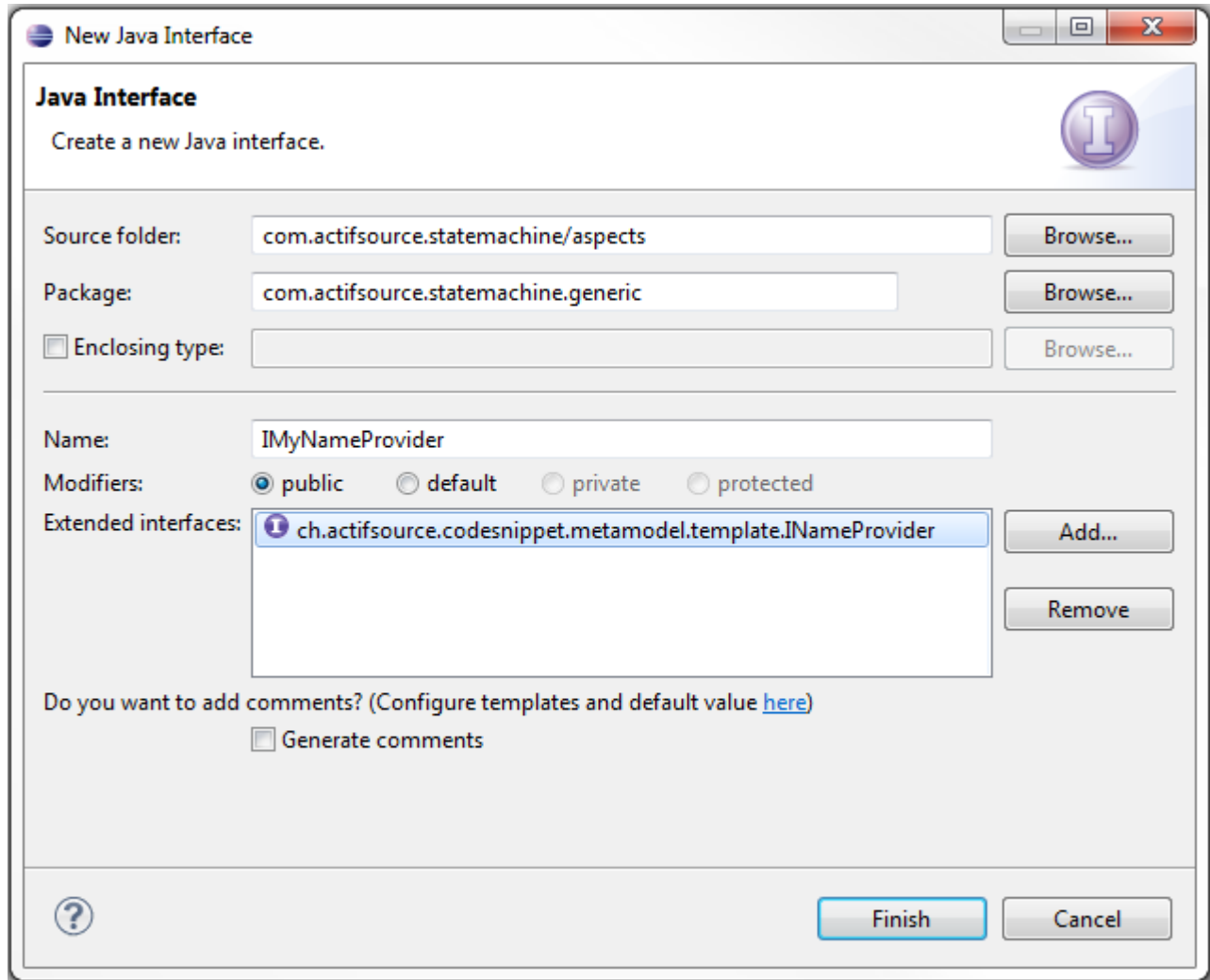
- ↖ Create a new Java source folder: select the project `com.actifsource.statemachine` and choose **New -> Source Folder** from the menu



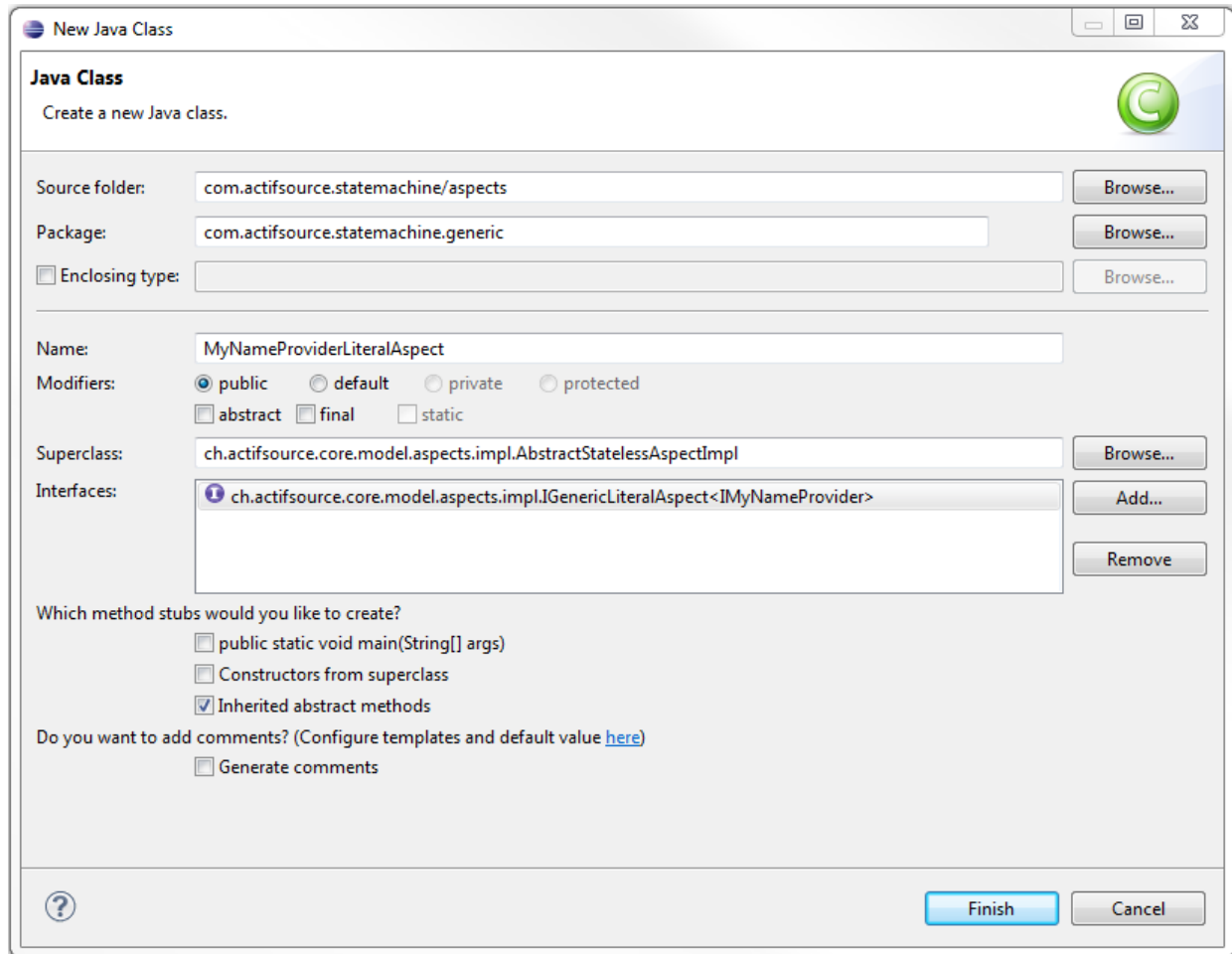
↪ Insert `aspects` as name of the new folder



↩ Create a new package com.actifsource.statemachine.generic in the aspects folder: select the aspects folder and choose **New -> Package** from the menu



- ↵ Select the package `com.actifsource.statemachine.generic` in the folder `aspects` and choose **New -> Interface** from the menu
- ↵ Write `IMyNameProvider` as name of the new interface
- ↵ Choose `ch.actifsource.codesnippet.metamodel.template.INameProvider` as **Extended interface**



- ↵ Select the package `com.actifsource.statemachine.generic` in the folder `aspects` and choose **New -> Class** from the menu
- ↵ Write `MyNameProviderLiteralAspect` as name of the new class
- ↵ Choose `ch.actifsource.core.model.aspects.impl.AbstractStatelessAspectImpl` as **Superclass**
- ↵ Choose `ch.actifsource.core.model.aspects.impl.IGenericLiteralAspect<IMyNameProvider>` as **Interface**
- ↵ Click **Finish**

```
IMyNameProvider.java | MyNameProviderLiteralAspect.java ✕
package com.actifsource.statemachine.generic;

import ch.actifsource.core.INode;
import ch.actifsource.core.Literal;
import ch.actifsource.core.job.IReadJobExecutor;
import ch.actifsource.core.model.aspects.impl.AbstractStatelessAspectImpl;
import ch.actifsource.core.model.aspects.impl.IGenericLiteralAspect;
import ch.actifsource.core.scope.IResourceScope;

public class MyNameProviderLiteralAspect extends AbstractStatelessAspectImpl
    implements IGenericLiteralAspect<IMyNameProvider> {

    @Override
    public boolean allowMultiline() {
        return false;
    }

    @Override
    public String isValid(IReadJobExecutor arg0, IResourceScope arg1,
        String arg2) {
        return null;
    }

    @Override
    public Literal create(IMyNameProvider arg0) {
        return new Literal(arg0.toString());
    }

    @Override
    public String getJavaConstructionExpression(IReadJobExecutor arg0,
        IResourceScope arg1, INode arg2) {
        return null;
    }

    @Override
    public IMyNameProvider getValue(IReadJobExecutor arg0, IResourceScope arg1,
        INode arg2) {
        return null;
    }

    @Override
    public Class<IMyNameProvider> getValueType() {
        return IMyNameProvider.class;
    }
}
```

- ↵ In the newly created class remove all TODO comments
- ↵ Write the statement `return MyNameProvider.class;` in the method `getValueType()`
- ↵ Write the statement `return new Literal(arg0.toString());`



**New Resource Wizard**

**Actifsource Resource**  
Creates a new actifsource resource of some type in the specified location.

Resource Path:

Namespace:

OwnRelation:

Type:

Name:

Modifiers:  Abstract  Final

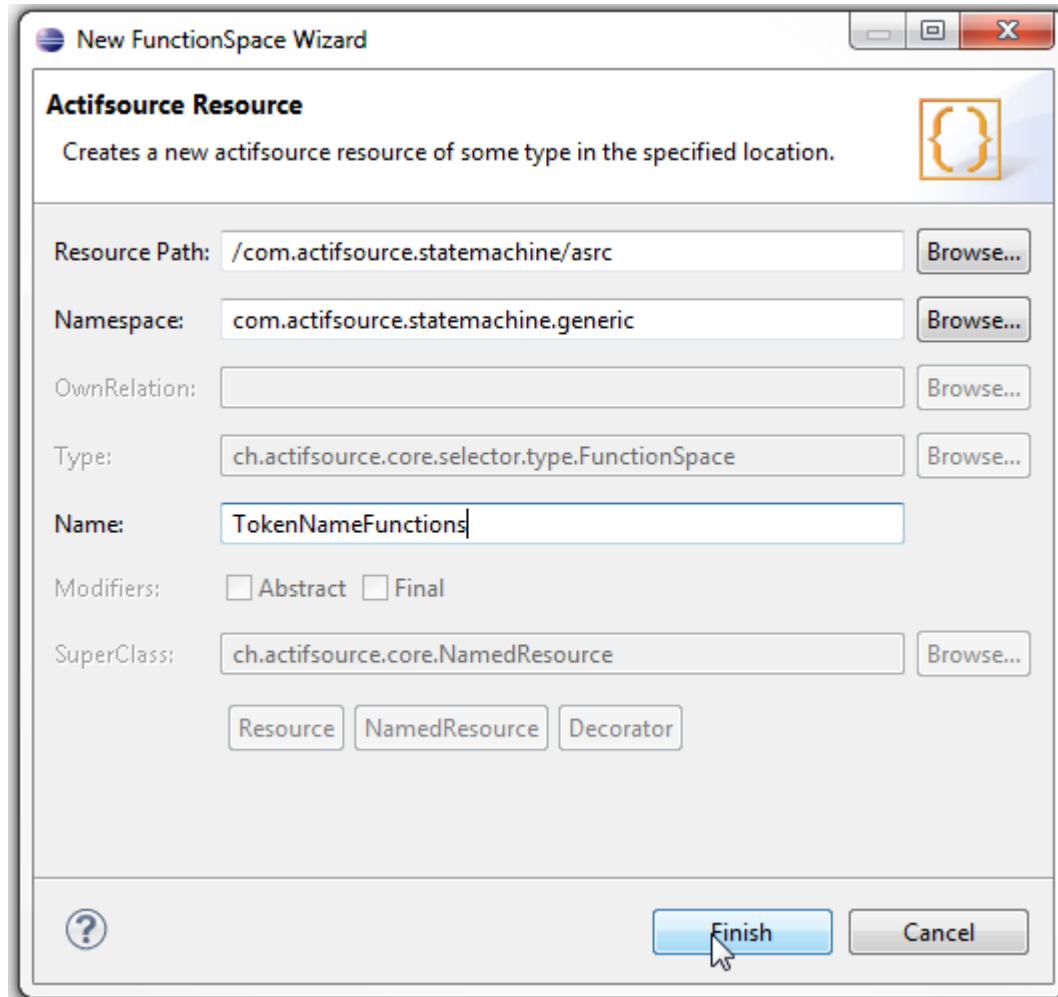
SuperClass:

- ↵ Create a new Literal type: select the package `com.actifsource.statemachine.generic` in the folder `asrc` and choose **New -> Resource** from the menu
- ↵ Choose the type `ch.actifsource.core.Literal` as **Type** of the new Resource
- ↵ Write `MyNameProviderLiteral` as **Name** of the new Resource

◆ `com.actifsource.statemachine.generic.MyNameProviderLiteral:Literal`

typeOf	<b>ch.actifsource.core.Literal</b>						
name	<b>MyNameProviderLiteral</b>						
comment							
aspect[LiteralAspect]	<table border="1"> <tr> <td>typeOf</td> <td><b>JavaAspectImplementation</b></td> </tr> <tr> <td>implements</td> <td>LiteralAspect</td> </tr> <tr> <td>className</td> <td>com.actifsource.statemachine.generic.MyNameProviderLiteralAspect</td> </tr> </table>	typeOf	<b>JavaAspectImplementation</b>	implements	LiteralAspect	className	com.actifsource.statemachine.generic.MyNameProviderLiteralAspect
typeOf	<b>JavaAspectImplementation</b>						
implements	LiteralAspect						
className	com.actifsource.statemachine.generic.MyNameProviderLiteralAspect						
aspect[LiteralAspect]							
extends	ch.actifsource.codesnippet.metamodel.parsetree.template.NameProvider						
extends							
modifier							

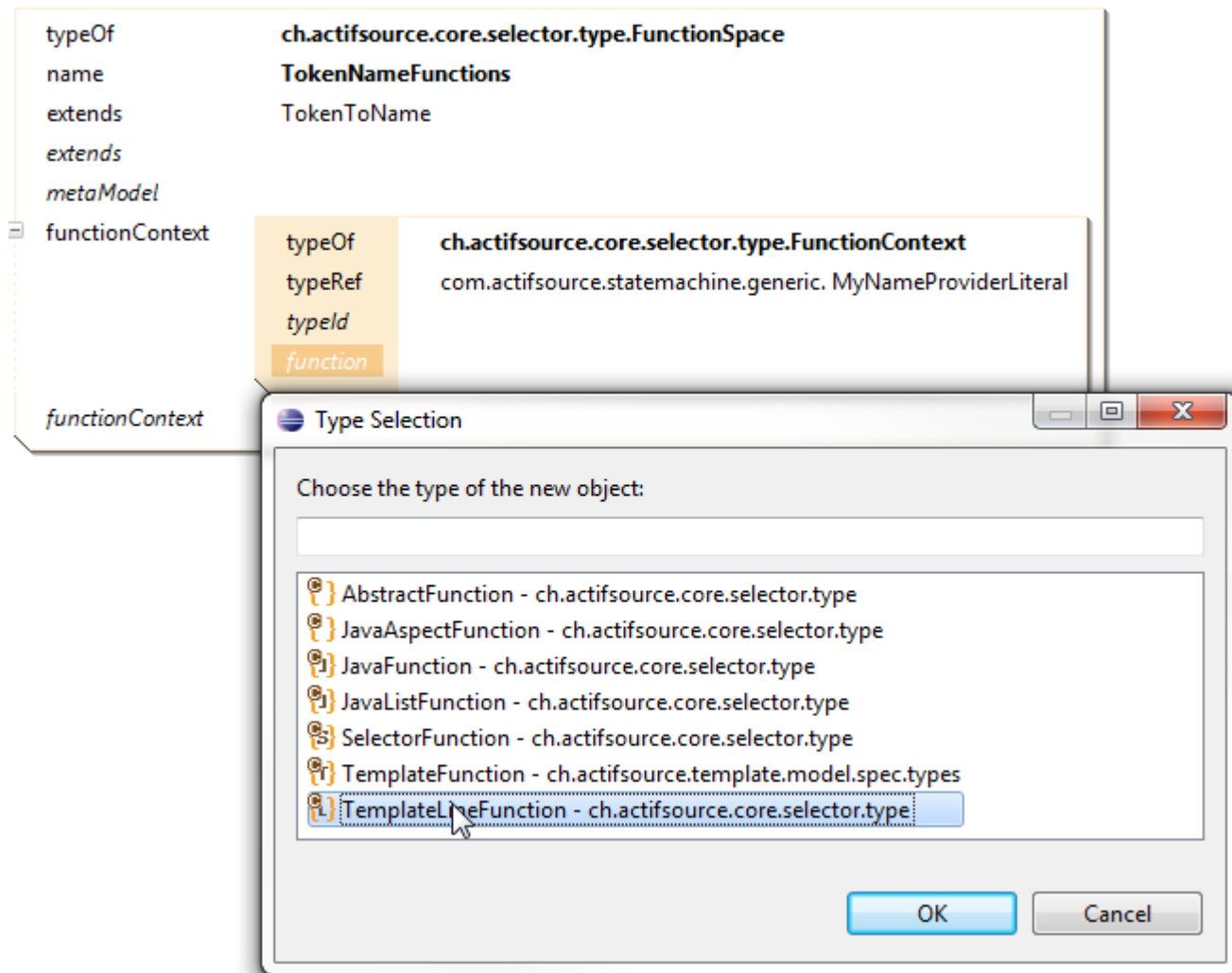
- ↪ Open the new Literal in the Resource Editor
- ↪ Create a LiteralAspect with type JavaAspectImplementation and choose the class `com.actifsource.statemachine.generic.MyNameProviderLiteralAspect` as className
- ↪ Let the MyNameProviderLiteral extend `ch.actifsource.codesnippet.metamodel.parsetree.template.NameProvider`, which is the default NameProvider and generates names by calling `simpleName@BuiltIn`



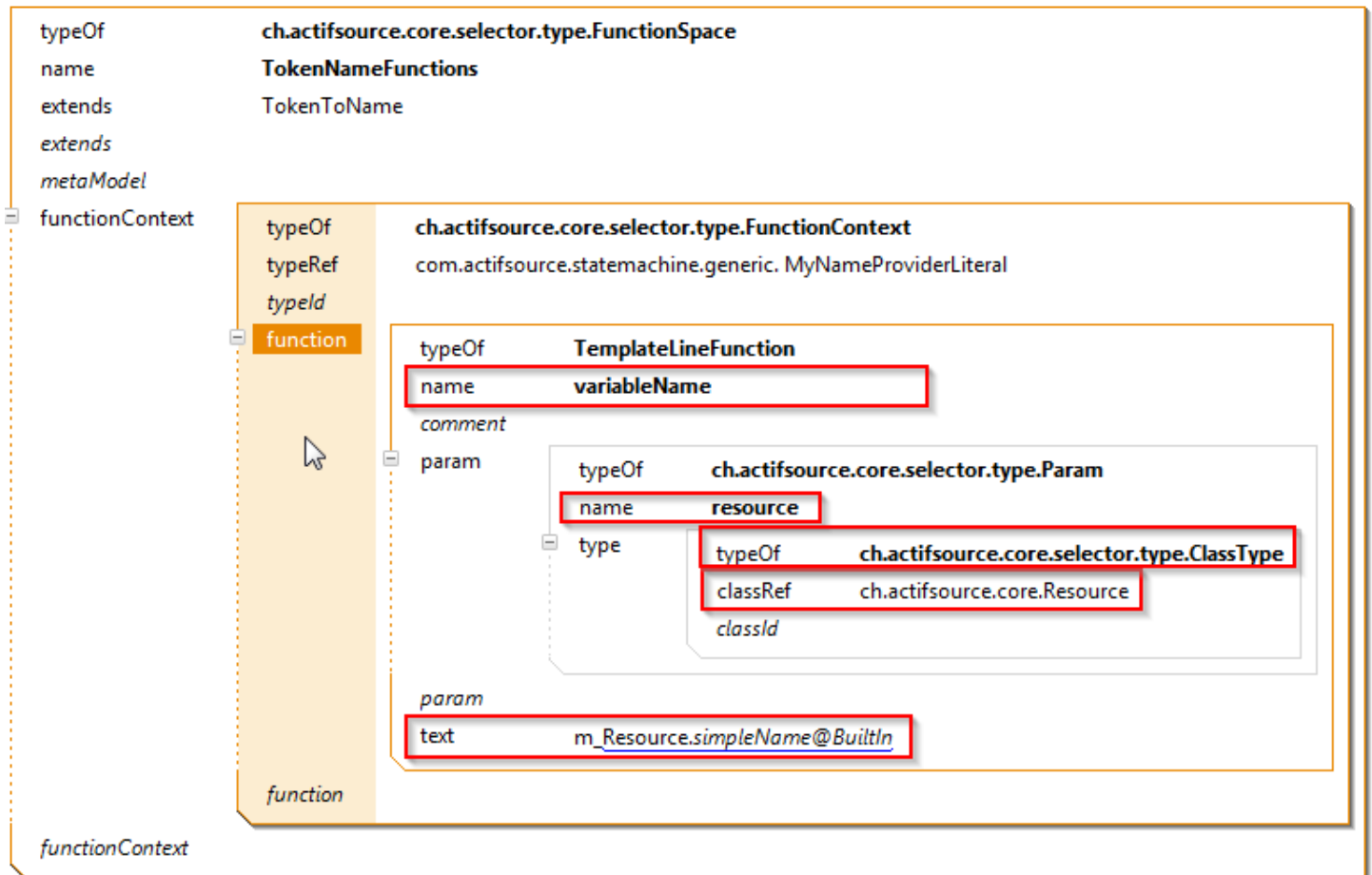
- ↪ Create a new FunctionSpace: Select the package `com.actifsource.statemachine.generic` in the `asrc` folder and choose **New -> FunctionSpace**
- ↪ Insert `TokenNameFunctions` as the name of the FunctionSpace and click **Finish**

typeOf	<b>ch.actifsource.core.selector.type.FunctionSpace</b>
name	<b>TokenNameFunctions</b>
<b>extends</b>	<input type="text" value="TokenToName"/>
<i>extends</i>	
<i>metaModel</i>	
<i>functionContext</i>	

↩ Let the new FunctionSpace TokenNameFunctions extend from TokenToName



- ↵ Open the FunctionSpace `TokenNameFunctions` in the Resource Editor
- ↵ Create a new FunctionContext with typeRef `MyNameProviderLiteral`
- ↵ Create a new function and choose TemplateLineFunction from the **Type Selection** dialog



- ↩ Insert `variableName` as the name of the new `TemplateLineFunction`
- ↩ Create a parameter `Param` of type `ClassType` and with `classRef ch.actifsource.core.Resource`
- ↩ Insert `m_Resource.simpleName@BuiltIn` as text, i.e., the function appends the prefix "m\_" to the output of [Resource.simpleName@BuiltIn](#)
- ⓘ Warning: Please choose the exact function name for `variableName` or `functionName` since these functions are overwritten

The image shows a screenshot of an IDE interface. On the left, a class hierarchy is visible for `functionContext`. The hierarchy includes:

- `functionContext` (expanded)
  - `functionContext[1]`
  - `functionContext[2]` (expanded)
    - `functionContext` (expanded)
      - `typeOf`: `ch.actifsource.core.selector.type.FunctionSpace`
      - `name`: `TokenNameFunctions`
      - `extends`: `TokenToName`
      - `extends`: `metaModel`

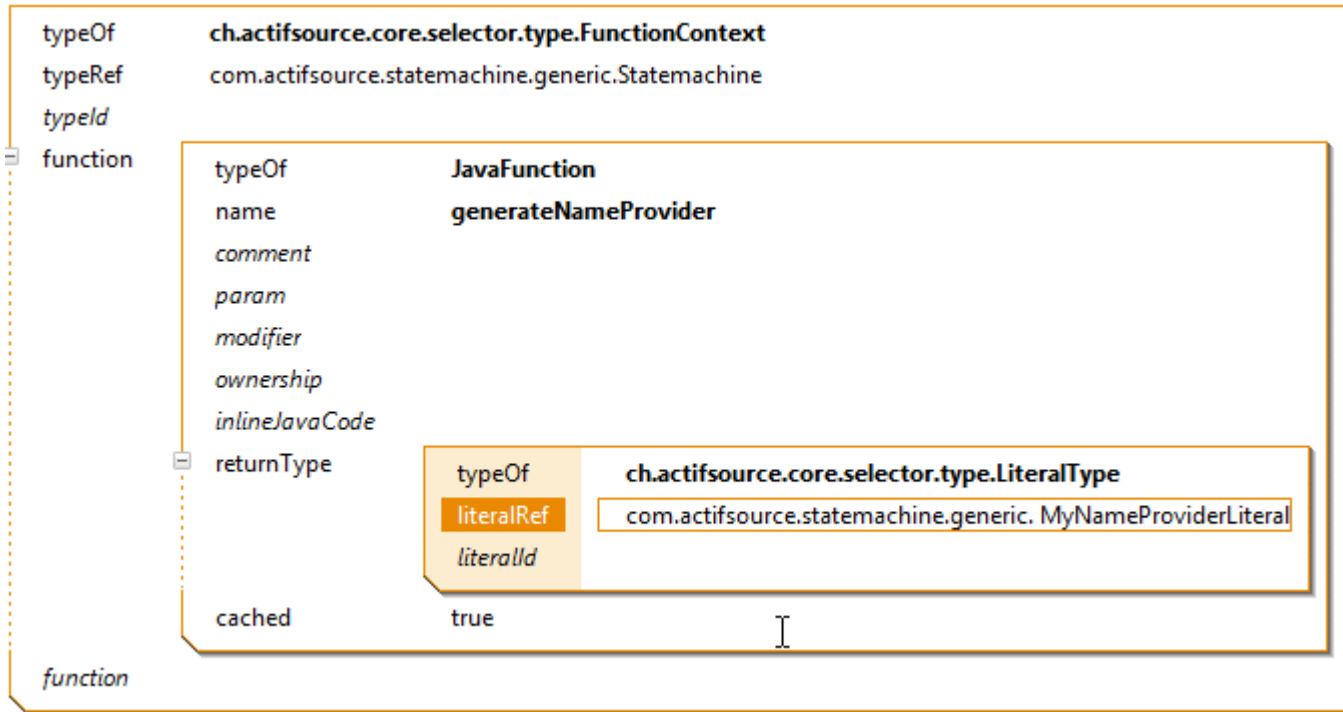
Below the hierarchy, a `functionContext` object is shown with a `MyNameProviderLiteral` property set to `FunctionContext`. A `function` property is also visible, with its `typeOf` set to `ch.actifsource.core.selector.type.FunctionContext` and `typeRef` set to `com.actifsource.statemachine.generic.StateMachine`.

In the foreground, a `Type Selection` dialog box is open. It prompts the user to "Choose the type of the new object:" and displays a list of available types:

- `AbstractFunction - ch.actifsource.core.selector.type`
- `JavaAspectFunction - ch.actifsource.core.selector.type`
- `JavaFunction - ch.actifsource.core.selector.type` (highlighted)
- `JavaListFunction - ch.actifsource.core.selector.type`
- `SelectorFunction - ch.actifsource.core.selector.type`
- `TemplateFunction - ch.actifsource.template.model.spec.types`
- `TemplateLineFunction - ch.actifsource.core.selector.type`

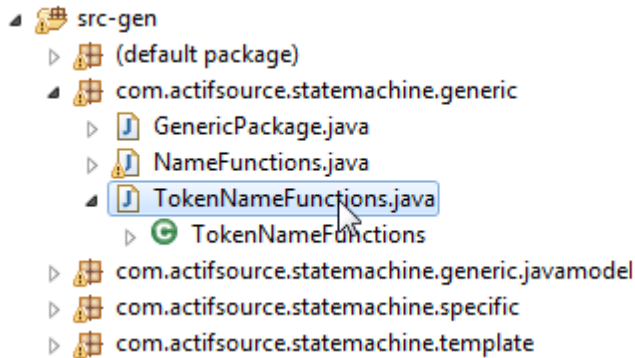
The dialog box has `OK` and `Cancel` buttons at the bottom.

- ↗ Create a new `FunctionContext` with `typeRef` `StateMachine`
- ↗ Create a new function and choose `JavaFunction` from the **Type Selection** dialog



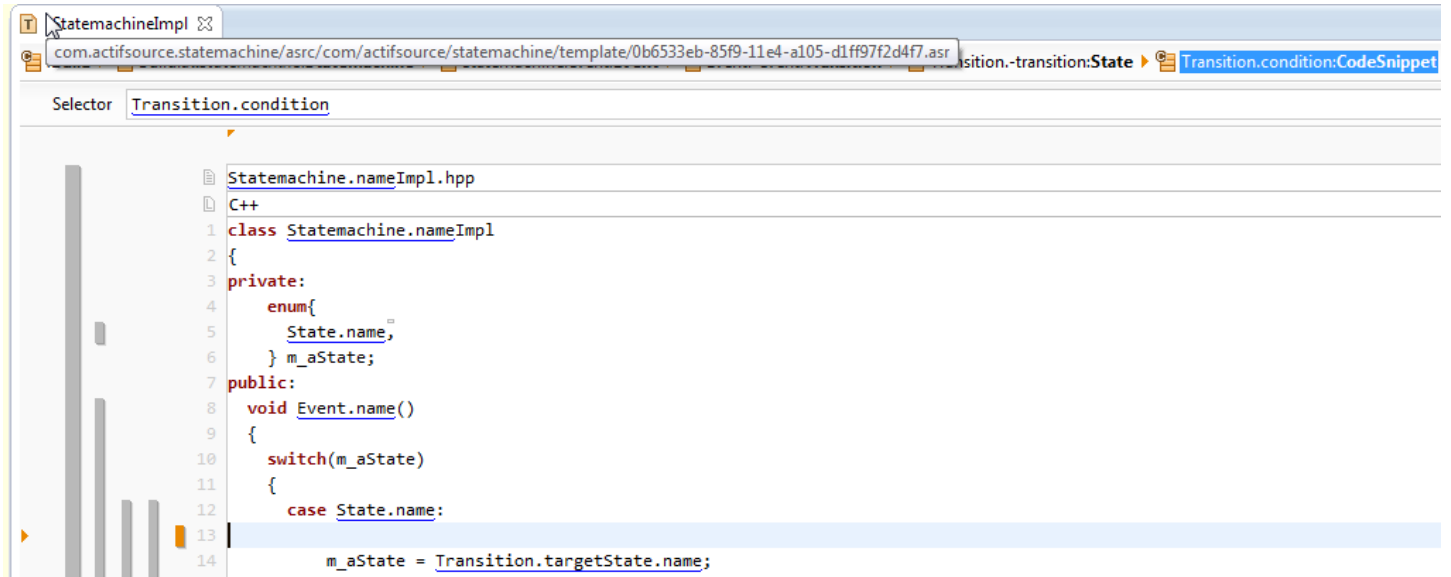
- ↪ Insert `generateNameProvider` as name of the function
- ↪ Create a statement `returnType` of type `LiteralType` with `literalRef MyNameProviderLiteral`





```
public com.actifsource.statemachine.generic.IMyNameProvider generateNameProvider(final com.actifsource.statemachine.generic.javamodel.IStateMachine statemachine) {  
    /* Begin Protected Region [[dc239b05-877d-11e4-a98e-6dd1214d65d1]] */  
    return new IMyNameProvider() {  
    };  
    /* End Protected Region [[dc239b05-877d-11e4-a98e-6dd1214d65d1]] */  
}
```

- ↵ Save the FunctionSpace TokenNameFunctions
- ↵ Open the newly generated file `TokenNameFunctions.java` in the **Java Editor**
- ↵ Write the statement `return new IMyNameProvider() {}` inside the protected region in the method body of the method `generateNameProvider`
- ↵ Save the file



```
StatemachineImpl
com.actifsource.statemachine/asrc/com/actifsource/statemachine/template/0b6533eb-85f9-11e4-a105-d1ff97f2d4f7.asr
Transition.-transition:State Transition.condition:CodeSnippet
Selector Transition.condition
Statemachine.nameImpl.hpp
C++
1 class Statemachine.nameImpl
2 {
3 private:
4     enum{
5         State.name,
6     } m_aState;
7 public:
8     void Event.name()
9     {
10        switch(m_aState)
11        {
12            case State.name:
13                m_aState = Transition.targetState.name;
14        }
15    }
16 }
```

- ↪ Open the template StatemachineImpl in the Template Editor
- ↪ Insert a new LineContext on the line after the case expressions
- ↪ Choose Transition.condition as the selector of the new context

\*StatemachineImpl

Build Build.allStatemachine:Statemachine Statemachine.event:Event Event.-event:Transition Transition.-transition:State Transition.co

Selector Statemachine.generateNameProvider@TokenNameFunctions:NameProvider

```

Statemachine.nameImpl.hpp
C++
1 class Statemachine.nameImpl
2 {
3 private:
4     enum{
5         State.name,
6     } m_aState;
7 public:
8     void Event.name()
9     {
10        switch(m_aState)
11        {
12            case State.name:
13                if (CodeSnippet.toCwithNameProvider@CodeSnippetToCode)
14                    m_aState = Transition.targetState.name;
15                break;
16        }
17    }
18 }
19 };

```

- ↵ Insert a new LineContext on the same line
- ↵ Choose Statemachine.generateNameProvider@TokenNameFunctions:NameProvider as selector of the new context
- ↵ Write an if-statement with CodeSnippet.toCwithNameProvider@CodeSnippetToCode
- ↵ Note that there is an error on the edited line because the parameter to the function cannot be resolved

Selector CodeSnippet:CodeSnippet

```
StateMachine.nameImpl.hpp
C++
1 class StateMachine.nameImpl
2 {
3 private:
4     enum{
5         State.name,
6     } m_aState;
7 public:
8     void Event.name()
9     {
10        switch(m_aState)
11        {
12            case State.name:
13                if (CodeSnippet.toCwithNameProvider@CodeSnippetToCode)
14                    {
15                        m_aState = Transition.targetState.name;
16                    }
17                break;
18        }
19    }
20
21};
```

- ↶ Insert a new line context on the same line and choose CodeSnippet:CodeSnippet (This dummy context allows Actifsource to correctly and automatically resolve the parameter to the function from the contexts)
- ↶ Save the template and make sure that the code is generated

```
StateMachine1Impl.hpp
void start()
{
    switch(m_aState)
    {
        case Initialized:
            if (m_startCounter < m_startLimit)
            {
                m_aState = Started;
            }
            break;
        case Stopped:
            if (m_startCounter < m_startLimit)
            {
                m_aState = Started;
            }
            break;
    }
}
```

- ↪ Open the file StateMachine1Impl.hpp and check that the variable names have been generated with the defined prefix "m\_"

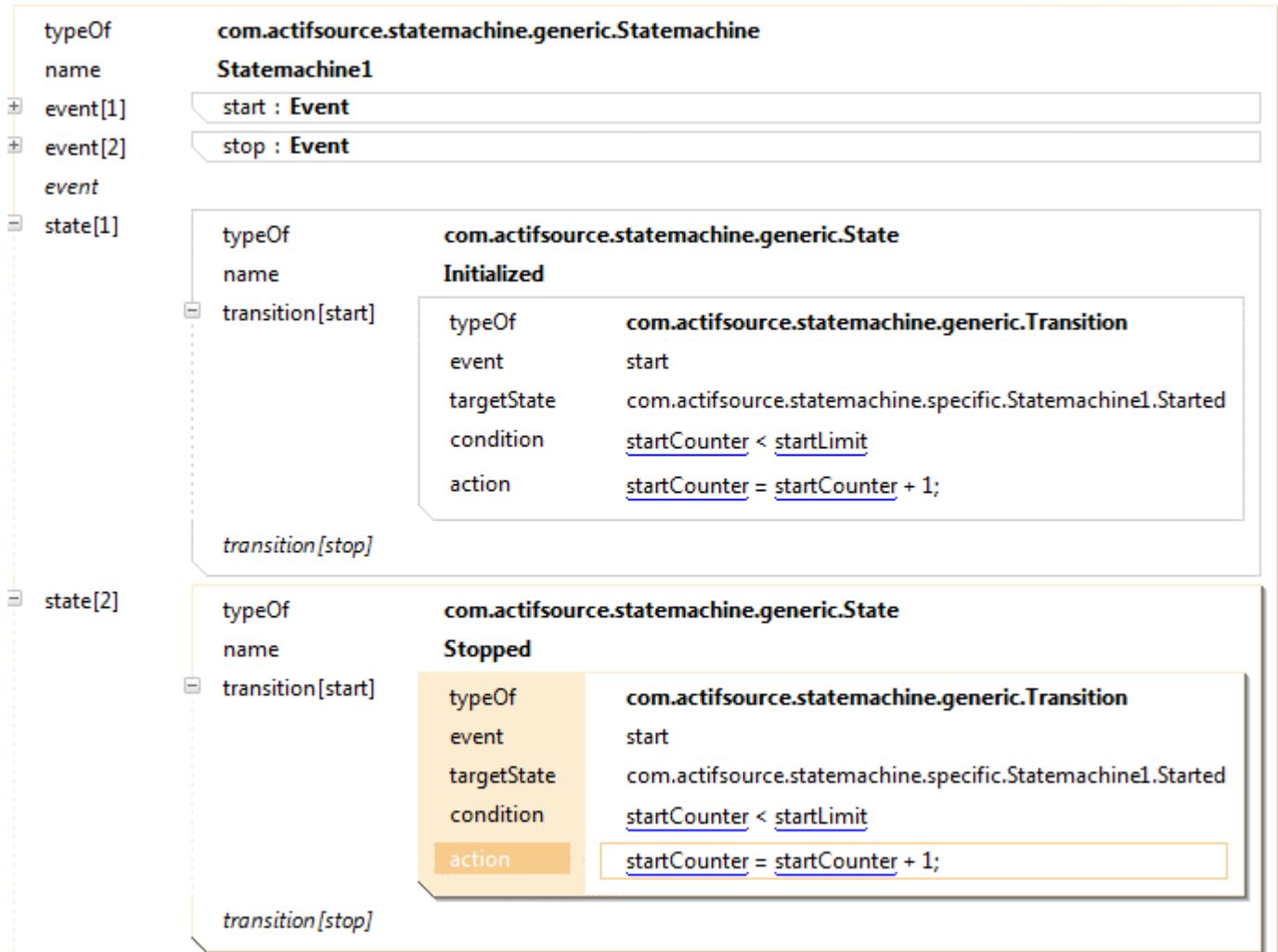
# Implement actions for transitions

The screenshot shows a resource editor for a transition. On the left, a list of properties is visible: property[1], property[2], property[3], and property[4]. The main editor area displays the following details:

- targetState** : UseRelation
- event** : UseRelation
- condition** : StructuredCodeSnippetRelation
- StructuredCodeSnippetRelation**
  - action**
  - CodeSnippetRelationAspect [1] : JavaAspectImplementation**
  - Cardinality0\_1**
  - ch.actifsource.codesnippet.metamodel.element.CodeSnippet
  - Cardinality0\_1**
- CMinus**
  - typeof** : ch.actifsource.codesnippet.metamodel.RelationTokenProvider
  - selector** : [Transition.-transition.-state.variable](#)
  - tokenType** : ch.actifsource.codesnippet.metamodel.TokenType.Variable
  - subtoken**

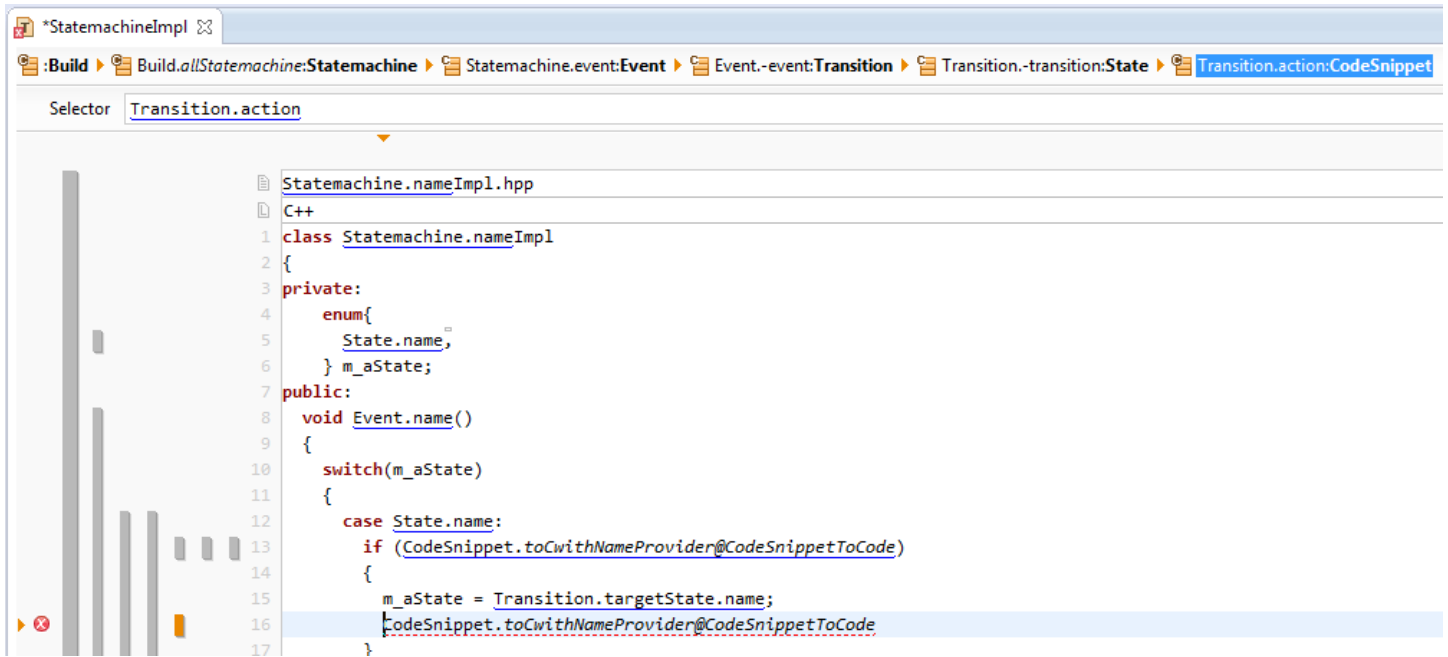
In this part, we will see how to add an action to a transition. We add the code corresponding to the action as a Code Snippet to the model. This code will be executed together with the transition:

- ↗ Open the class [Transition](#) in the Resource Editor and add a Code Snippet relation as already seen in Part II
- ↗ Insert action as name of the [StructuredCodesnippetRelation](#), choose [subjectCardinality](#) and [objectCardinality](#) [Cardinality0\\_1](#)
- ↗ Create a [RelationTokenProvider](#) with selector [Transition.-transition.-state.variable](#) and [tokenType](#) [Variable](#)
- ↗ Choose the [language](#) [CMinus](#)



We will now increment the `startCounter` each time that we switch to the State Started:

- ↪ Open State Machine1 in the Resource Editor
- ↪ Add an action to the State Initialized and insert the code `startCounter = startCounter + 1;` into the Code Snippet\_Editor
- ↪ Add an action to the State Stopped and insert the code `startCounter = startCounter + 1;` into the Code Snippet\_Editor



The screenshot shows a code editor window titled '\*StatemachineImpl'. The breadcrumb navigation at the top indicates the current context: ':Build > Build.allStatemachine:Statemachine > C Statemachine.event:Event > Event.-event:Transition > C Transition.-transition:State > Transition.action:CodeSnippet'. The 'Selector' dropdown is set to 'Transition.action'. The code editor displays the following C++ code:

```
Statemachine.nameImpl.hpp
C++
1 class Statemachine.nameImpl
2 {
3 private:
4     enum{
5         State.name,
6     } m_aState;
7 public:
8     void Event.name()
9     {
10        switch(m_aState)
11        {
12            case State.name:
13                if (CodeSnippet.toCwithNameProvider@CodeSnippetToCode)
14                {
15                    m_aState = Transition.targetState.name;
16                    CodeSnippet.toCwithNameProvider@CodeSnippetToCode
17                }
18        }
19    }
20 }
```

- ↪ Open the template StatemachineImpl in the Template Editor
- ↪ Add a new line context after the assignment statement that updates the state
- ↪ Choose Transition.action as the selector of the state (to insert code for Transitions which have an action defined)
- ↪ Call the function toCwithNameProvider@CodeSnippetToCode on the CodeSnippet in the new line context
- ↪ Note that there is an inconsistency because we have not yet added a NameProvider to our contexts which can be used as the parameter to the function



The screenshot shows an IDE window titled '\*StatemachineImpl'. The breadcrumb navigation at the top indicates the current context: Event.-event:Transition > Transition.-transition:State > Transition.action:CodeSnippet > Statemachine.generateNameProvider@TokenNameFunctions. The 'Selector' field contains the text 'Statemachine.generateNameProvider@TokenNameFunctions:NameProvider'. The code editor displays the following C++ code:

```

Statemachine.nameImpl.hpp
C++
1 class Statemachine.nameImpl
2 {
3 private:
4     enum{
5         State.name,
6     } m_aState;
7 public:
8     void Event.name()
9     {
10        switch(m_aState)
11        {
12            case State.name:
13                if (CodeSnippet.toCwithNameProvider@CodeSnippetToCode)
14                {
15                    m_aState = Transition.targetState.name;
16                    CodeSnippet.toCwithNameProvider@CodeSnippetToCode
17                }
18            }
19        }

```

A context menu is open over line 16, showing the selector 'CodeSnippet:CodeSnippet:CodeSnippet'.

- ↪ Add a new line context on the same line as before and choose Statemachine.generateNameProvider@TokenNameFunctions:NameProvider as selector of the context
- ↪ Add another line context on the same line and choose the selector CodeSnippet:CodeSnippet (needed for the parameter matching)
- ↪ Save the template and make sure that the code in Statemachine1Impl.hpp is generated and overwritten

Statemachine1Impl.hpp

```
void start()
{
    switch(m_aState)
    {
        case Initialized:
            if (m_startCounter < m_startLimit)
            {
                m_aState = Started;
                m_startCounter = m_startCounter + 1;
            }
            break;
        case Stopped:
            if (m_startCounter < m_startLimit)
            {
                m_aState = Started;
                m_startCounter = m_startCounter + 1;
            }
            break;
    }
}
```

- Open the file Statemachine1Impl.hpp and check that the code that increments the counter has been correctly inserted
- ① Complete the generated classes by adding a member function initialize() and the missing variables (and probably adding the cases for the missing enumeration values to make the compiler happy).

