

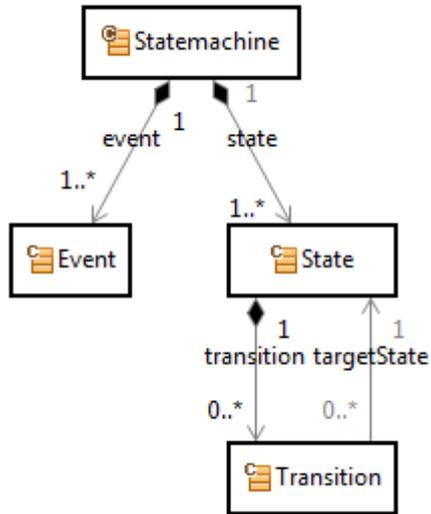


Tutorial

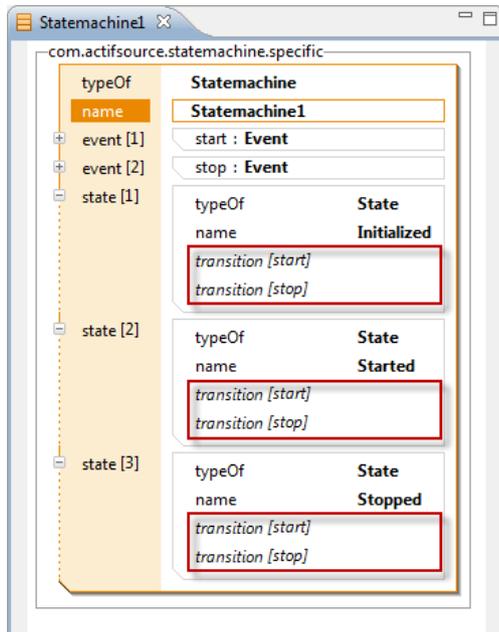
State Machine

Tutorial	Actifsource Tutorial – State Machine
Required Time	<ul style="list-style-type: none"> • 40 Minutes
Prerequisites	<ul style="list-style-type: none"> • Actifsource Tutorial – Installing Actifsource • Actifsource Tutorial – Simple Service • Actifsource Tutorial – Complex Service
Goal	<ul style="list-style-type: none"> • Developing an easy to use state machine model • Show possible events in every transition • Restrict transition target to state instances of the own state machine
Topics covered	<ul style="list-style-type: none"> • Decorating Relation Aspect • Range Restriction Aspect • Selector (forward and reverse selection)
Notation	<ul style="list-style-type: none"> •  To do •  Information • Bold: Terms from actifsource or other technologies and tools • <u>Bold underlined</u>: actifsource Resources • <u>Underlined</u>: User Resources • <u><i>UnderlinedItalics</i></u>: Resource Functions • Monospaced: User input • <i>Italics</i>: Important terms in current situation
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>actifsource GmbH Täfernstrasse 37 5405 Baden-Dättwil Switzerland www.actifsource.com</p>
Trademark	<p>actifsource is a registered trademark of actifsource GmbH 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>

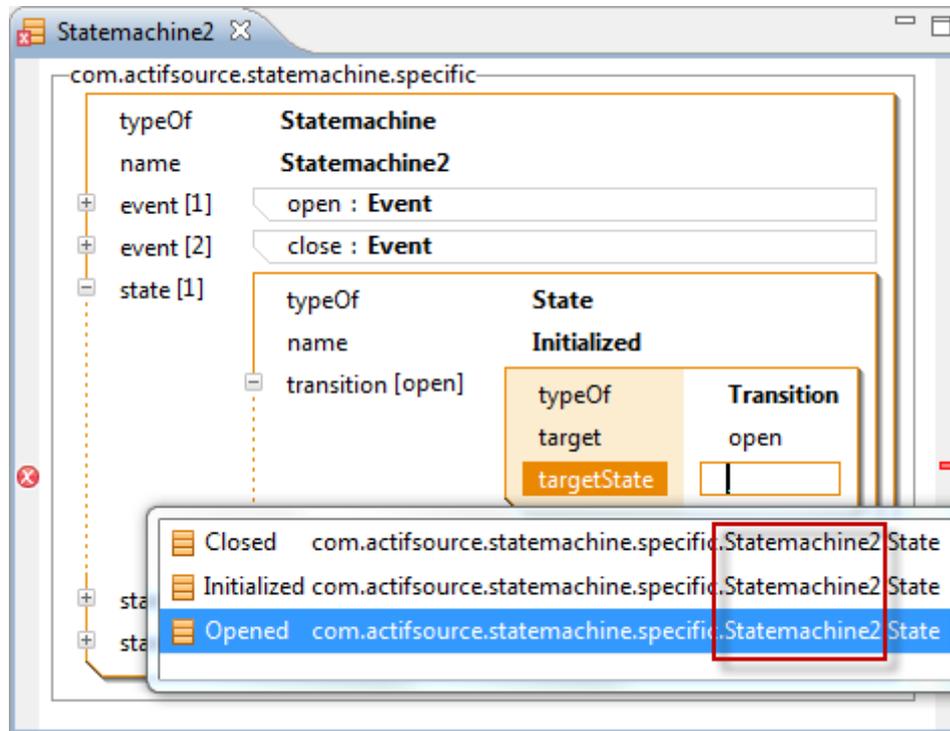
- Create a simple state machine



- Show possible events in every transition



- Restrict transition target to state instances of the own state machine



-

- Write a code template to generate code for a statemachine

```
StateMachine1Impl.hpp
class StateMachine1Impl
{
private:
    enum{
        Initialized,
        Started,
        Stopped
    } m_aState;

public:
    void start()
    {
        switch(m_aState)
        {
            case Initialized:
                m_aState = Started;
                break;
            case Stopped:
                m_aState = Started;
                break;
        }
    }

    void stop()
    {
        switch(m_aState)
        {
            case Started:
                m_aState = Stopped;
                break;
        }
    }
};
/* Actifsource ID=[0b6aff85-85f9-11e4-a105-d1f
```

```
StateMachine2Impl.hpp
class StateMachine2Impl
{
private:
    enum{
        Initialized,
        Opened,
        Closed
    } m_aState;

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

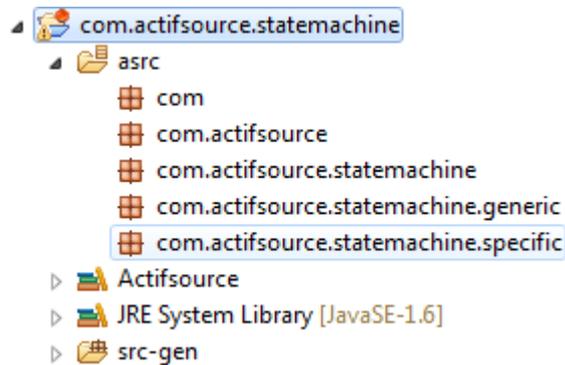
    void close()
    {
        switch(m_aState)
        {
            case Opened:
                m_aState = Closed;
                break;
        }
    }
};
/* Actifsource ID=[0b6aff85-85f9-11e4-a105-c
```

Part I:

Preparation

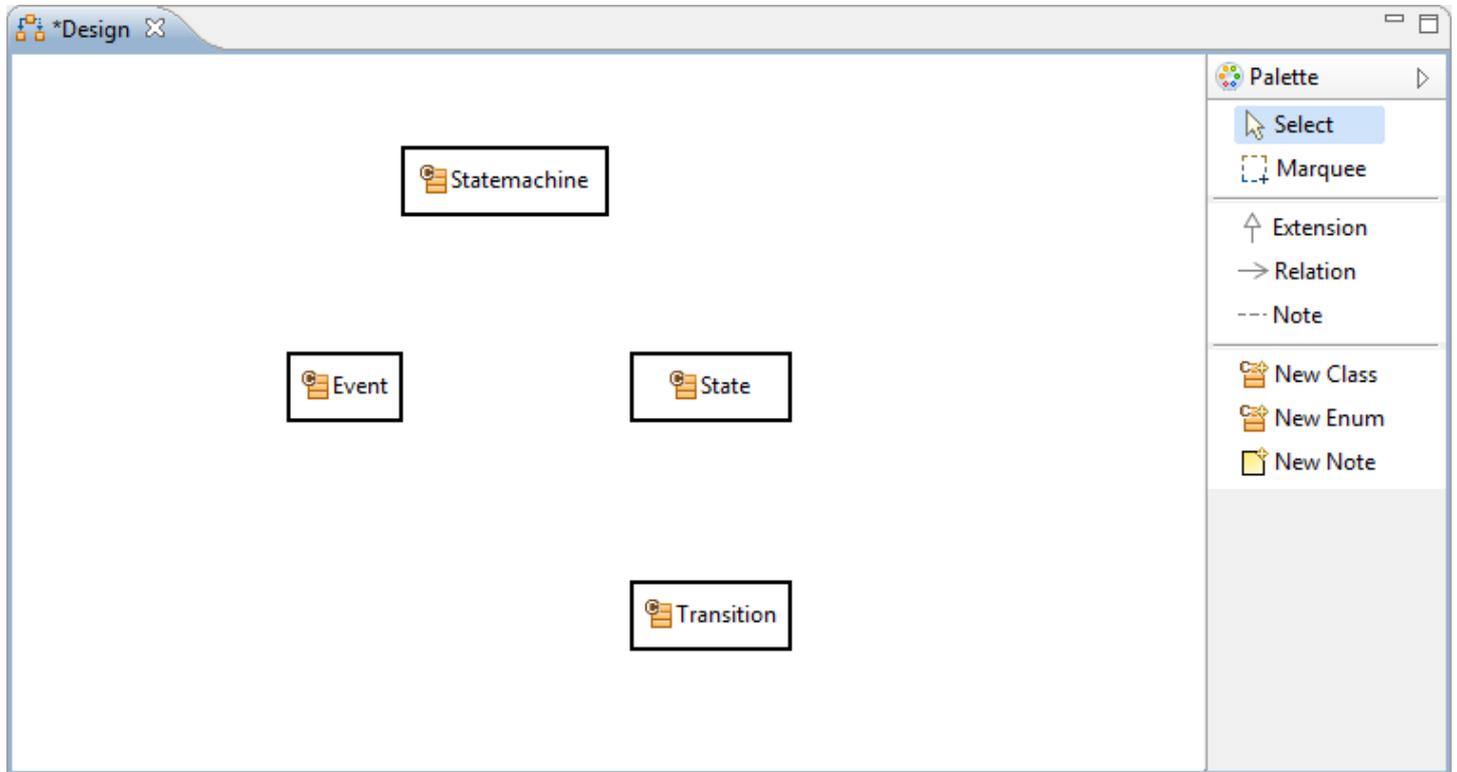
6

- ↪ Prepare a new **actifsource Project** named `com.actifsource.statemachine` as seen in the *Actifsource Tutorial – Simple Service*
- ↪ Use the following package structure

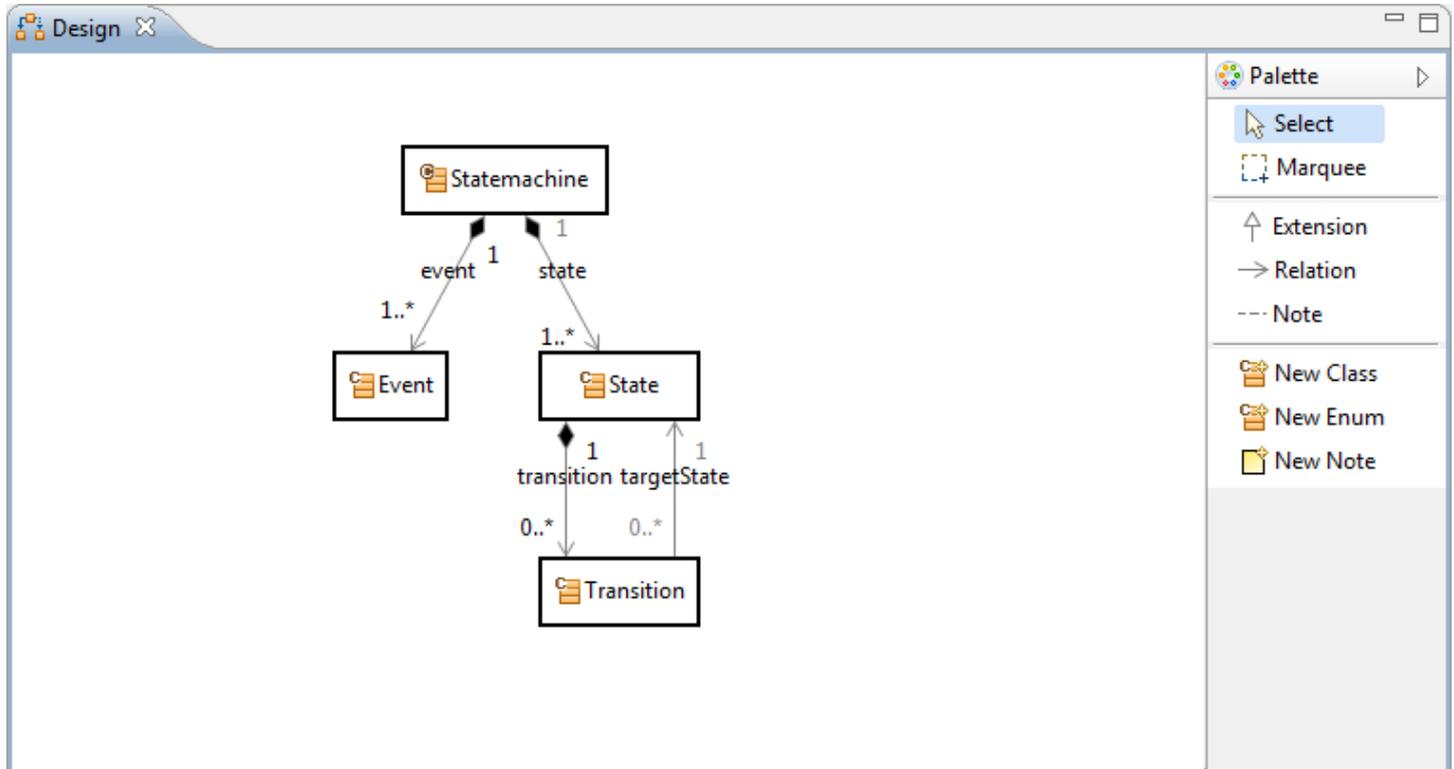


Create a State Machine

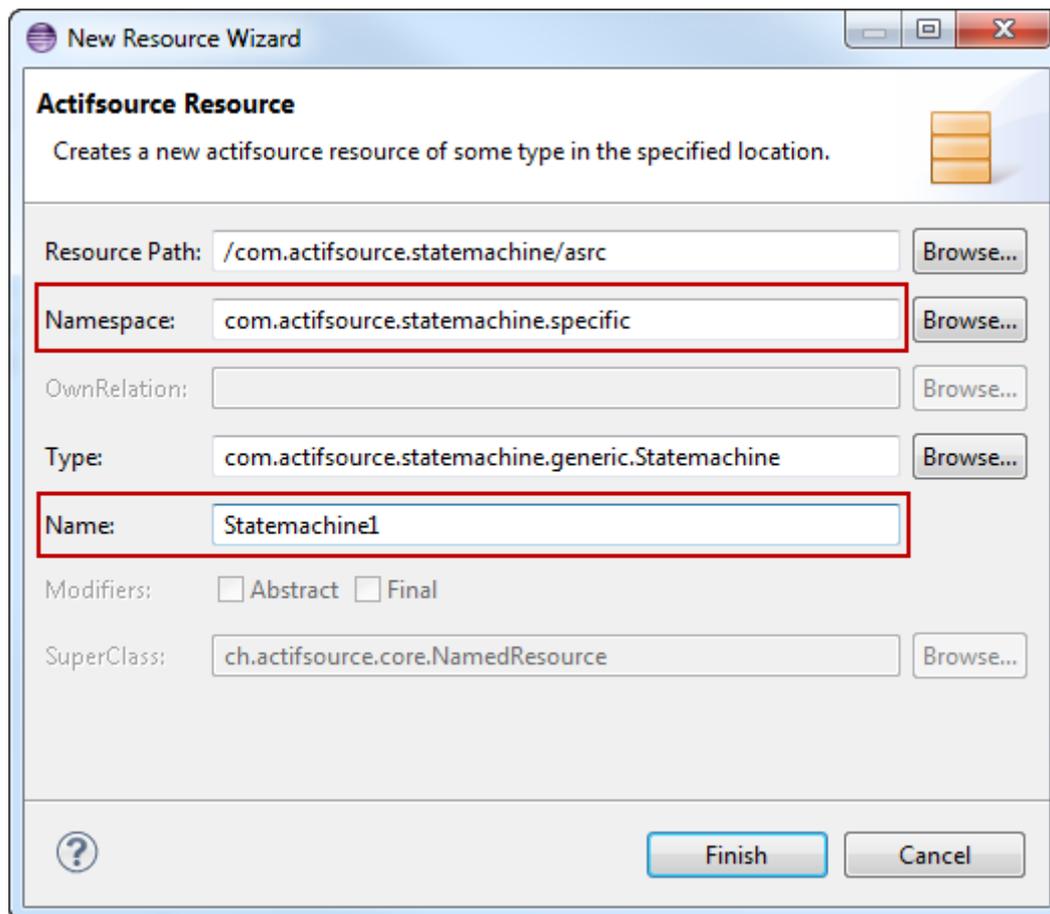
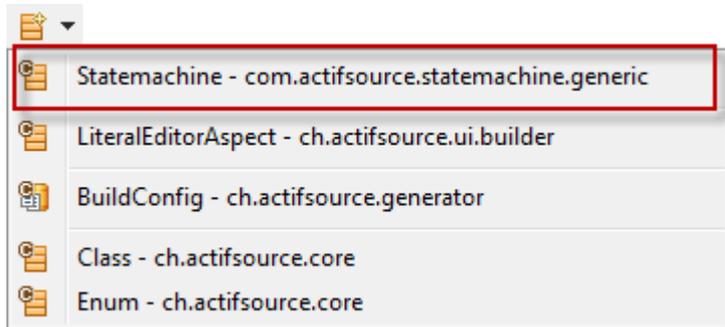
- ① Create a simple state machine
- ① Instantiate the state machine and see its deficits



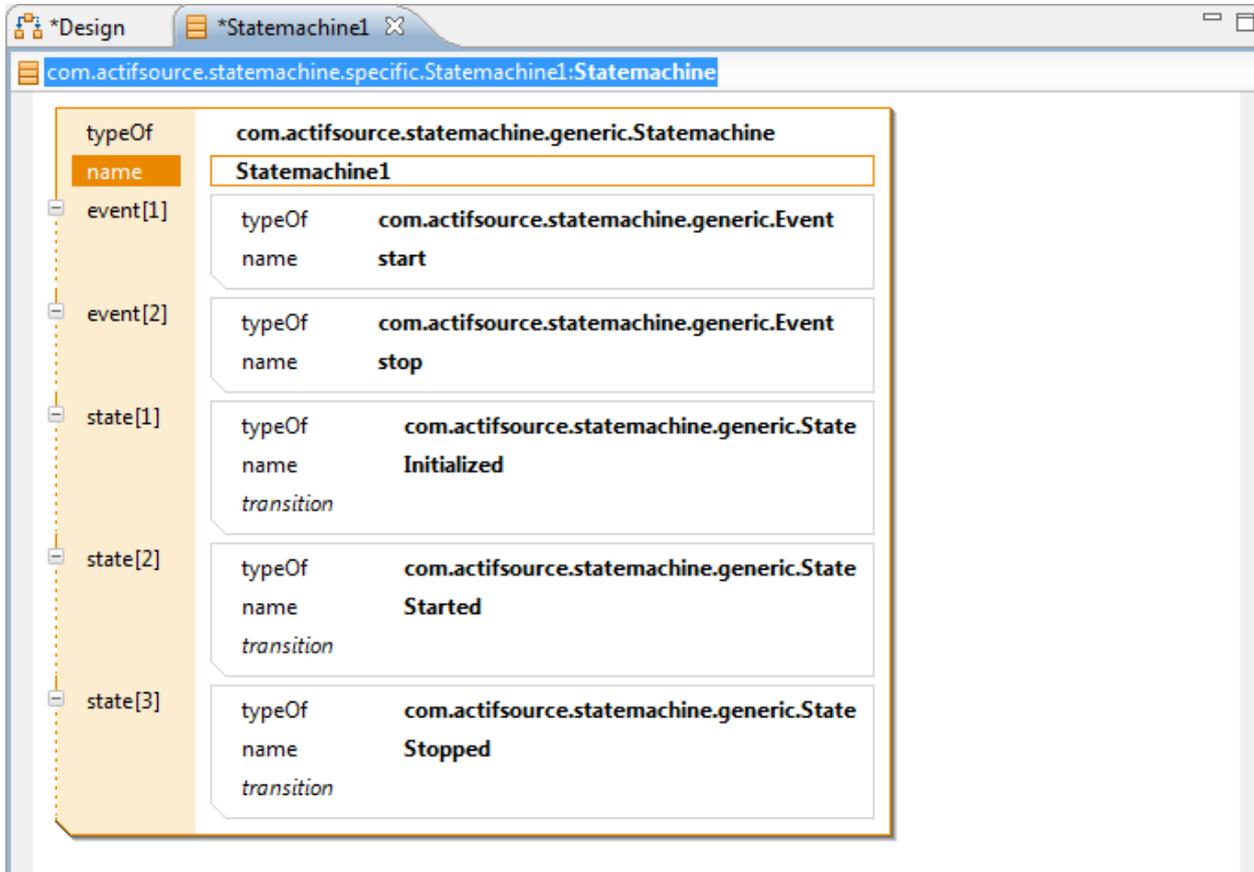
- ↩ Create a **Generic Domain Model** named *Design* in the **Package generic** using the **DiagramEditor**
- ↩ The Design shall contain the following **Domain Classes**
 - Statemachine, Event, State, Transition



- ↗ Insert a **OwnRelation** between
 - State and Event
 - State and State
- ↗ Insert a **DecoratingRelation** between
 - State and Transition
- ↗ Insert a **UseRelation** between
 - Transition and State
- ↗ Adjust the **Cardinalities** as shown above
- ⓘ Warning: The layout for the relations transition and targetState might differ in your editor



↪ Create a Statemachine named Statemachine1 in the **Package specific**



- ↪ Add the Events `start` and `stop`
- ↪ Add the States `Initialized`, `Started` and `Stopped` as shown above

Decorating Relation Aspect

- ① Learn how to decorate a relation with a list of resources in order to prevent the mixing of instances from different Statemachines

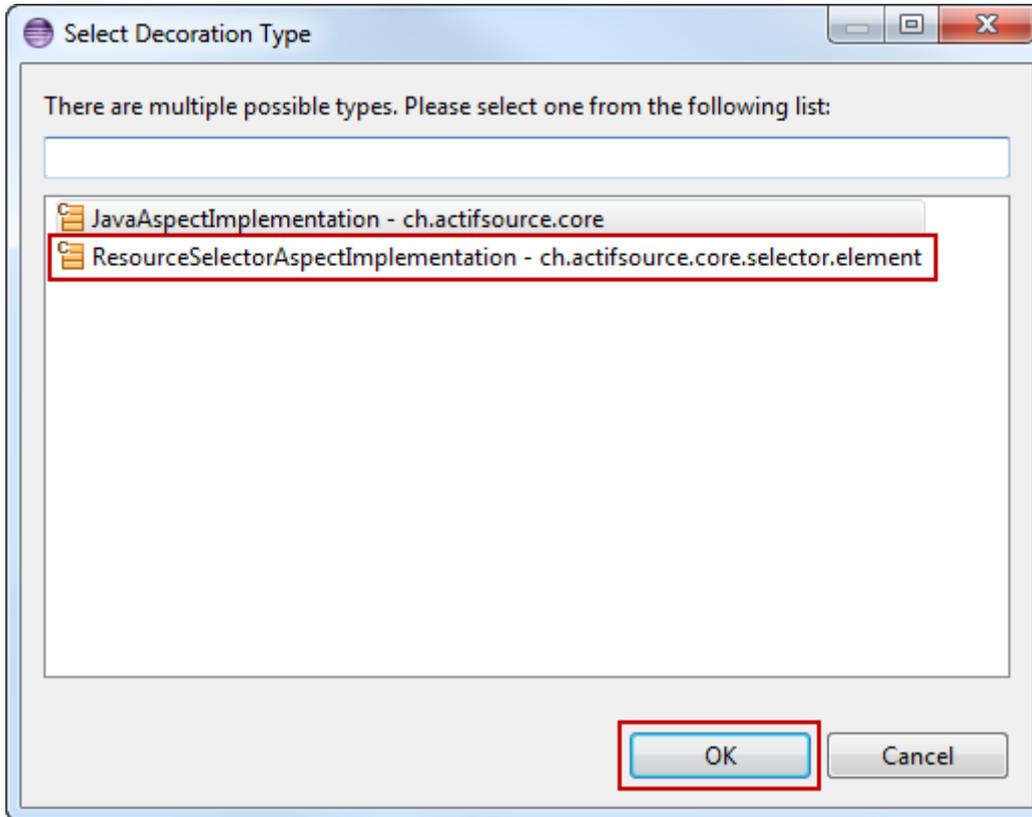
The screenshot shows the IBM Design Center interface. The breadcrumb path is `com.actifsource.statemachine.generic.State:Class` and the selected element is `transition:DecoratingRelation`. The main area displays the properties of the `DecoratingRelation` transition, which is highlighted in a yellow box. The properties are:

<code>typeOf</code>	<code>ch.actifsource.core.Class</code>
<code>name</code>	<code>State</code>
<code>comment</code>	
<code>aspect[InitializationAspect]</code>	
<code>aspect[ResourceValidationAspect]</code>	
<code>aspect[NameAspect]</code>	
<code>extends</code>	<code>ch.actifsource.core.NamedResource</code>
<code>modifier</code>	
<code>property</code>	
<code>definesAspect</code>	
<code>allowRoot</code>	

The `DecoratingRelation` transition properties are:

<code>typeOf</code>	<code>DecoratingRelation</code>
<code>name</code>	<code>transition</code>
<code>comment</code>	
<code>subjectCardinality</code>	<code>Cardinality0_1</code>
<code>aspect[OwnRangeRestrictionAspect]</code>	
<code>aspect[DecoratingRelationAspect]</code>	
<code>modifier</code>	
<code>objectCardinality</code>	<code>Cardinality1_1</code>
<code>relationMode</code>	
<code>style</code>	
<code>defaultValue</code>	
<code>range</code>	<code>com.actifsource.statemachine.generic.Transition</code>

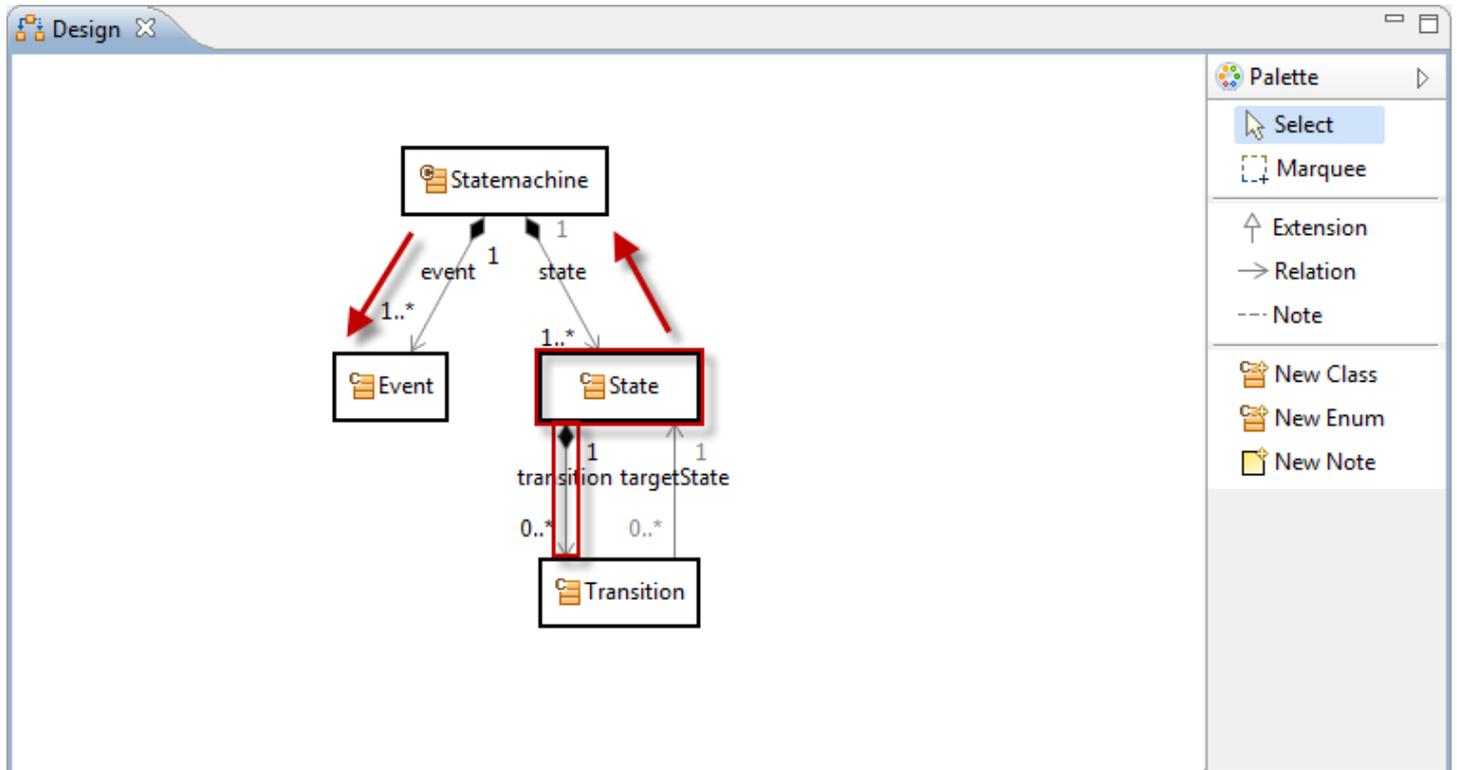
- ↖ In State open the **DecoratingRelation** transition
- ↖ Press Enter on `aspect[DecoratingRelationAspect]`



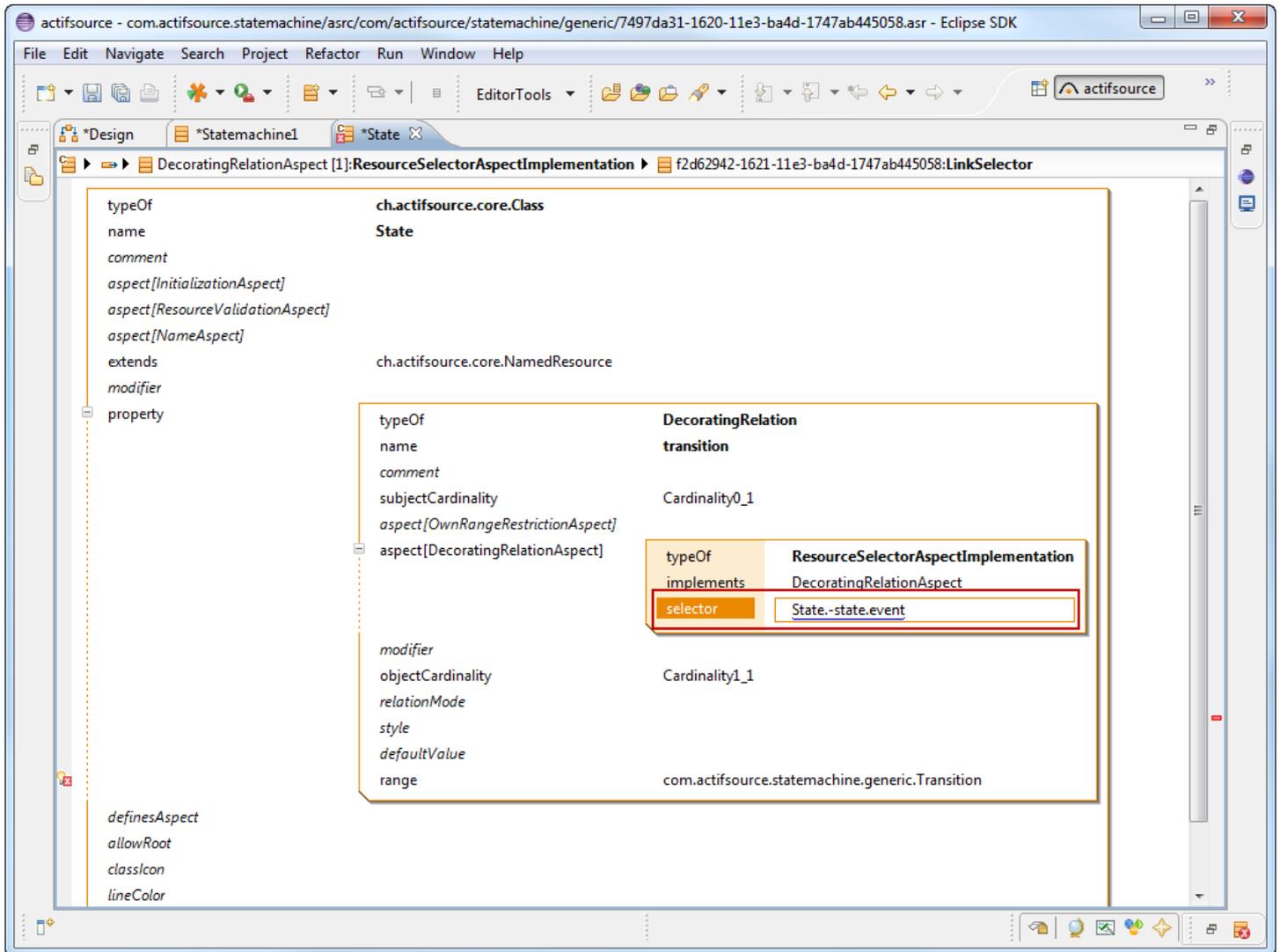
- ① Note that you can choose between a *JavaAspectImplementation* and a *SelectorAspectImplementation*
 - Selecting the *JavaAspectImplementation* allows you to write Java Code for complex operations
 - Selecting the *ResourceSelectorAspectImplementation* allows you to use the easy Selector syntax

↵ Select *ResourceSelectorAspectImplementation*

↵ Click *OK*



- ① Let's look at a possible Transition for every Event
- ① The **DecoratingRelation** transition is found in State
- ① We have to navigate from State to Event
 - Navigate backwards from State via state to Statemachine
 - Navigate forward from Statemachine via event to Event



↩ Enter the Selector `State.-state.event` using Content Assist (Ctrl+Space)

ⓘ Note that State.-state navigates backwards from State to Statemachine

The screenshot shows the IDE interface with the following details:

- Class Definition:**
 - `ch.actifsource.core.Class`
 - `name State`
 - `comment`
 - `aspect [InitializationAspect]`
 - `aspect [ResourceValidationAspect]`
 - `aspect [NameAspect]`
 - `extends ch.actifsource.core.NamedResource`
 - `modifier`
 - `property`
- DecoratingRelation Class:**
 - `typeOf DecoratingRelation`
 - `name transition`
 - `comment`
 - `subjectCardinality Cardinality0_1`
 - `aspect [OwnRangeRestrictionAspect]`
 - `aspect [DecoratingRelationAspect]`
 - `modifier`
 - `objectCardinality Cardinality1_1`
 - `relationMode`
 - `style`
 - `defaultValue`
 - `range com.actifsource.statemachine.generic.Transition`
- ResourceSelectorAspectImplementation Class:**
 - `typeOf ResourceSelectorAspectImplementation`
 - `implements DecoratingRelationAspect`
 - `selector State.-state.event`
- Quick Assist:**
 - Message: "The class used as range of a DecoratingRelation must extend ch.actifsource.core.Decorator!"
 - Action: "Quick Assist available (Ctrl+1)"

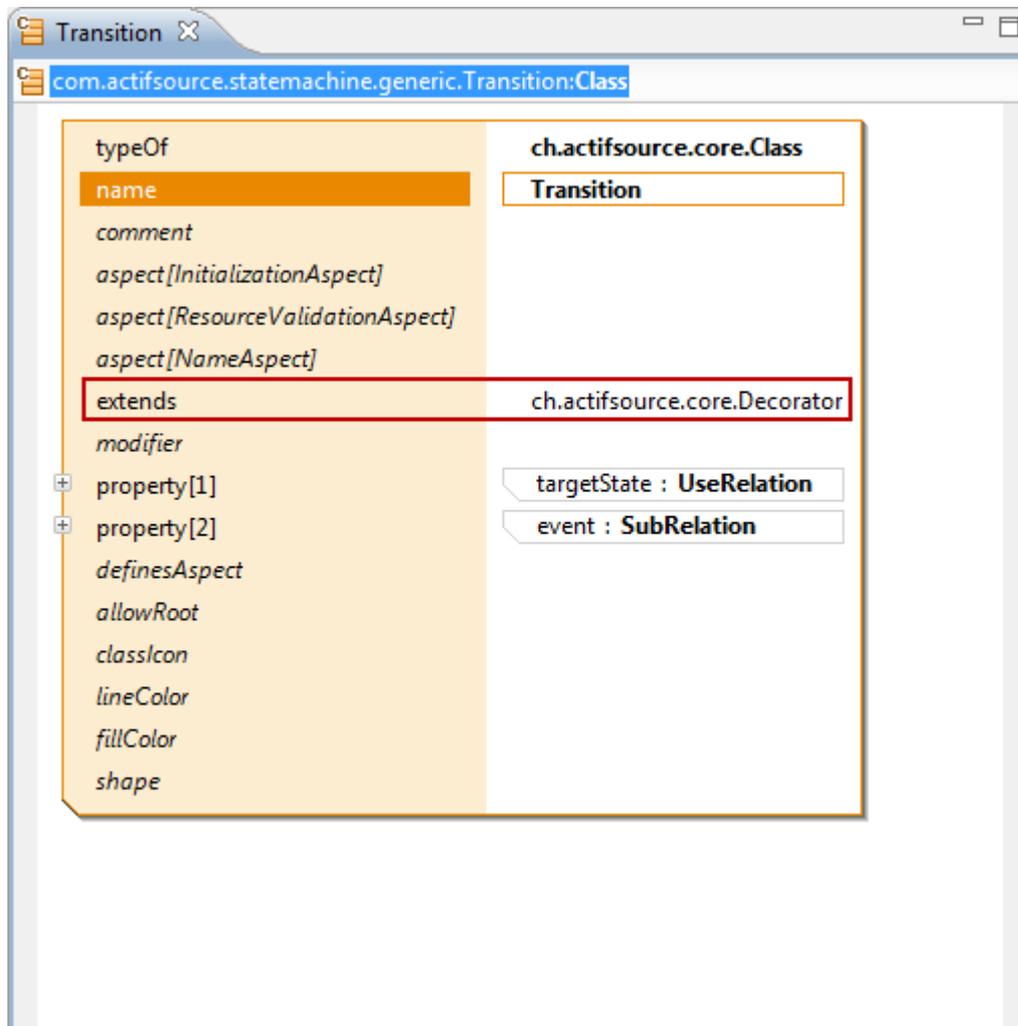
- ① Implementing a *DecoratingRelationAspect* asks for a subclass of *Decorator*
- ① *Decorator* has a **useRelation target** which is used to store the specific decorating Resource
 - Shown as: `decoratingRelation[target]`
- ↪ Open **Quick Assist** by clicking the light bulb or press **Ctrl+1**

The screenshot shows an IDE window with the following content:

- Class: ch.actifsource.core.State**
 - typeOf: ch.actifsource.core.Class
 - name: State
 - comment:
 - aspect [InitializationAspect]
 - aspect [ResourceValidationAspect]
 - aspect [NameAspect]
 - extends: ch.actifsource.core.NamedResource
 - modifier:
 - property:
- Class: DecoratingRelation**
 - typeOf: DecoratingRelation
 - name: transition
 - comment:
 - subjectCardinality: Cardinality0_1
 - aspect [OwnRangeRestrictionAspect]
 - aspect [DecoratingRelationAspect]
 - modifier:
 - objectCardinality: Cardinality1_1
 - relationMode:
 - style:
 - defaultValue:
 - range: com.actifsource.statemachine.generic.Transition
- Class: ResourceSelectorAspectImplementation**
 - typeOf: ResourceSelectorAspectImplementation
 - implements: DecoratingRelationAspect
 - selector: State.-state.event

A Quick Assist tooltip is visible over the 'range' property of 'DecoratingRelation', suggesting: **Let Transition extend Decorator.**

Use **Quick Assist** to let Transition extend Decorator



The screenshot shows the Eclipse IDE interface for the class `Transition` (package `com.actifsource.statemachine.generic`). The class is shown extending `ch.actifsource.core.Decorator`. The `extends` statement is highlighted with a red box. The class also has several aspects and properties.

Property	Value
<code>typeOf</code>	<code>ch.actifsource.core.Class</code>
<code>name</code>	<code>Transition</code>
<code>comment</code>	
<code>aspect[InitializationAspect]</code>	
<code>aspect[ResourceValidationAspect]</code>	
<code>aspect[NameAspect]</code>	
<code>extends</code>	<code>ch.actifsource.core.Decorator</code>
<code>modifier</code>	
<code>property[1]</code>	<code>targetState : UseRelation</code>
<code>property[2]</code>	<code>event : SubRelation</code>
<code>definesAspect</code>	
<code>allowRoot</code>	
<code>classIcon</code>	
<code>lineColor</code>	
<code>fillColor</code>	
<code>shape</code>	

- ↗ Open Transition
- ⓘ By default a **Class** extends **NamedResource**
- ↗ The **Quick Assist Action** changed the **extends** statement from **NamedResource** to **Decorator**

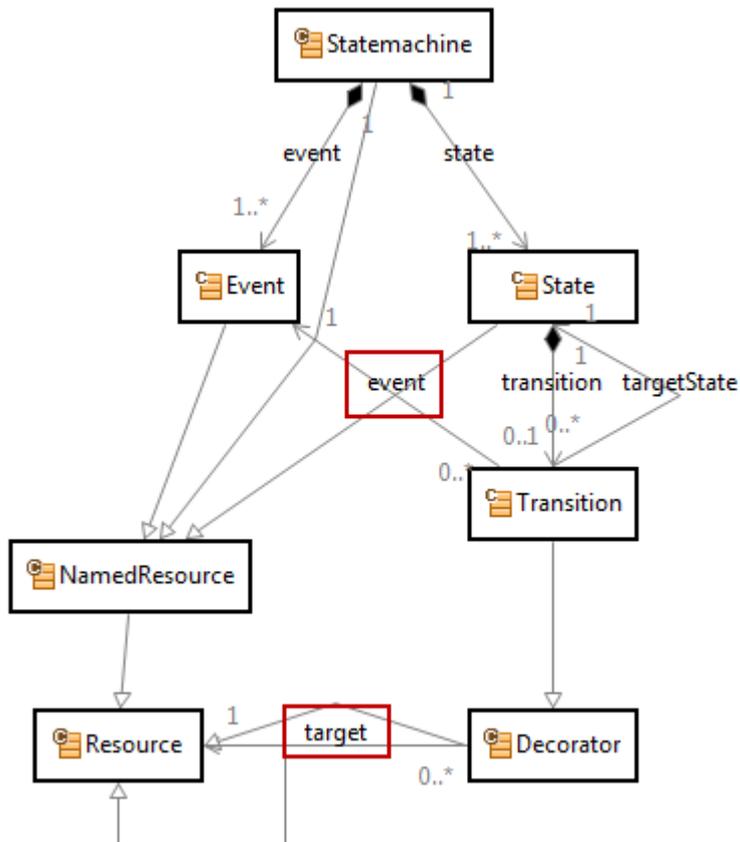
The screenshot shows the Eclipse IDE with the class `com.actifsource.statemachine.generic.Transition` selected. The class hierarchy is displayed as follows:

- `Transition` (highlighted)
 - inherits from `ch.actifsource.core.Decorator`
 - inherits from `ch.actifsource.core.Class`
- `ch.actifsource.core.Decorator`
 - inherits from `ch.actifsource.core.Class`
 - has a subrelation `SubRelation` (highlighted)
- `ch.actifsource.core.Decorator.target`
 - inherits from `ch.actifsource.core.Decorator.target`

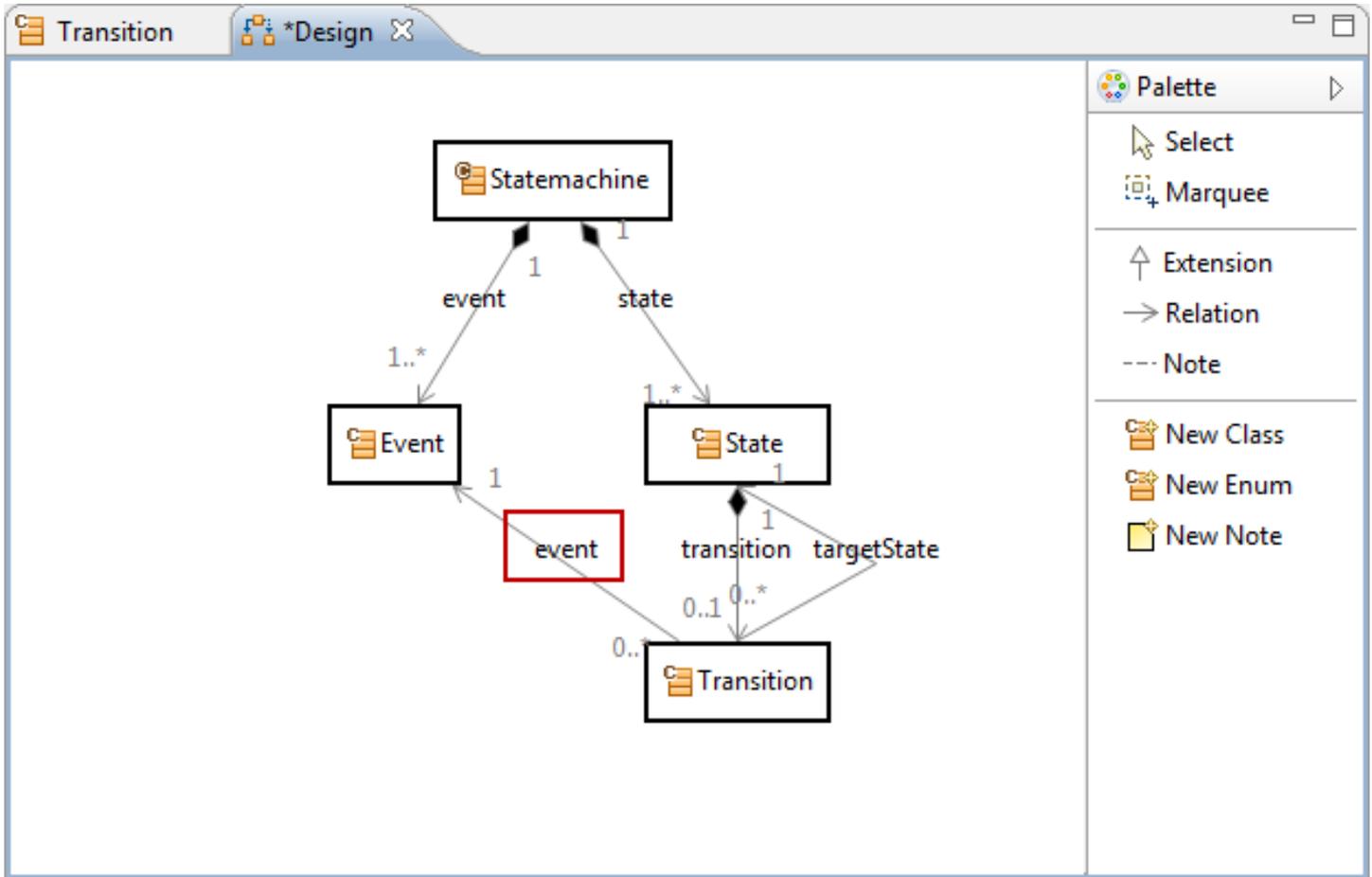
The `SubRelation` class has the following attributes:

- `typeOf`: `SubRelation`
- `name`: `event`
- `comment`
- `subjectCardinality`: `Cardinality1_1`
- `aspect`: `[UseRangeRestrictionAspect]`
- `aspect`: `[OwnRangeRestrictionAspect]`
- `modifier`
- `objectCardinality`: `Cardinality0_N`
- `relationMode`
- `style`
- `range`: `com.actifsource.statemachine.generic.Event`
- `extends`: `ch.actifsource.core.Decorator.target`
- `displayStrategy`

- ① Quick Assist has done the following
- Added **extend Decorator**
 - Added **SubRelation event**



- ① The **range** of `Decorator.target` is **Resource** and therefore untyped in the context of your domain
- ① The new **SubRelation** `target` extends `Decorator.target` but with `Event` as its **range**
- ① When writing template code, you are able to access `Transition.event` typed as `Event`



① Note that the SubRelation target has been added in the *Design Diagram* automatically

①

The screenshot shows the class hierarchy for `com.actifsource.statemachine.specific.StateMachine1`. The class is a subclass of `com.actifsource.statemachine.generic.StateMachine`. The hierarchy is as follows:

- `com.actifsource.statemachine.specific.StateMachine1` (name: `StateMachine1`)
 - `event[1]` (typeOf: `com.actifsource.statemachine.generic.Event`, name: `start`)
 - `event[2]` (typeOf: `com.actifsource.statemachine.generic.Event`, name: `stop`)
 - `state[1]` (typeOf: `com.actifsource.statemachine.generic.State`, name: `Initialized`)
 - `transition[start]`
 - `transition[stop]`
 - `state[2]` (typeOf: `com.actifsource.statemachine.generic.State`, name: `Started`)
 - `transition[start]`
 - `transition[stop]`
 - `state[3]` (typeOf: `com.actifsource.statemachine.generic.State`, name: `Stopped`)
 - `transition[start]`
 - `transition[stop]`

- Open the specific `StateMachine StateMachine1`
- ① Note there is a **decoratingRelation** transition for every `Event`
- ① Add new `Events` and observe the **decoratingRelation** transition

The screenshot shows a software development environment with a state machine configuration. The main window displays a tree view of a state machine with states and transitions. A specific state 'Initialized' is selected, and a new transition is being created. A dropdown menu is open, showing a list of states to select as the target state. The 'Started' state is highlighted in blue.

The tree view shows the following structure:

- com.actifsource.statemachine.generic.Statemachine1
 - event[1] start : Event
 - event[2] stop : Event
 - state[1]
 - com.actifsource.statemachine.generic.State
 - name Initialized
 - transition[start]
 - com.actifsource.statemachine.generic.Transition
 - targetState
 - event
 - transition[stop]
 - state[2] Started : State
 - state[3] Stopped : State

The dropdown menu shows the following options:

- new com.actifsource.statemachine.generic State
- Initialized com.actifsource.statemachine.specific.Statemachine1 State
- Started com.actifsource.statemachine.specific.Statemachine1 State
- Stopped com.actifsource.statemachine.specific.Statemachine1 State

- ↪ In the State Initialized create a new Transition for transition[start]
- ↪ Select Started as targetState
- ⓘ Note that the relation target has been completed automatically with the specific decorating Event start

The screenshot shows the configuration of a State machine in an IDE. The main window is titled '*Statemachine1' and shows the class 'com.actifsource.statemachine.specific.Statemachine1:Statemachine'. The left sidebar shows a tree view with 'typeOf', 'name', 'event[1]', 'event[2]', 'state[1]', 'state[2]', and 'state[3]'. The main area displays the configuration for 'Statemachine1' with events 'start', 'stop', and 'Initialized'. Two states are shown: 'Started' and 'Stopped', each with transitions to 'Stopped' and 'Started' respectively. The 'Started' state transition to 'Stopped' has an event 'com.actifsource.statemachine.specific.Statemachine1.stop'. The 'Stopped' state transition to 'Started' has an event 'com.actifsource.statemachine.specific.Statemachine1.start'.

↩ Configure the State instances Started and Stopped as shown above

Range Restriction Aspect

- ① Content Assist (Ctrl+Sapce) in actifsource shows all instances of a desired type; It is often useful to restrict this selection
- ① Learn how to apply range restrictions to filter instances for a given type

The image shows two screenshots of a UML class browser. The top screenshot shows a class named `com.actifsource.statemachine.specific.Statemachine1:Statemachine`. It has a `typeOf` property set to `com.actifsource.statemachine.generic.Statemachine` and a `name` property set to `Statemachine1`. It contains three event instances: `start : Event`, `stop : Event`, and `Initialized : State`. It also contains three state instances: `Started : State` and `Stopped : State`. The bottom screenshot shows a class named `*Statemachine2` with a `typeOf` property set to `com.actifsource.statemachine.generic.Statemachine` and a `name` property set to `Statemachine2`. It contains two event instances: `open : Event` and `close : Event`. It also contains three state instances: `Initialized : State`, `Opened : State`, and `Closed : State`. The bottom screenshot is highlighted with a red border.

- ① Let's discover the needs for a range restriction aspect
- ↪ Create a `Statemachine` named `Statemachine2` in the **Package specific**
- ↪ Add the `Event` instances `open` and `close`
- ↪ Add the `States` instances `Initialize`, `Opened` and `Closed`

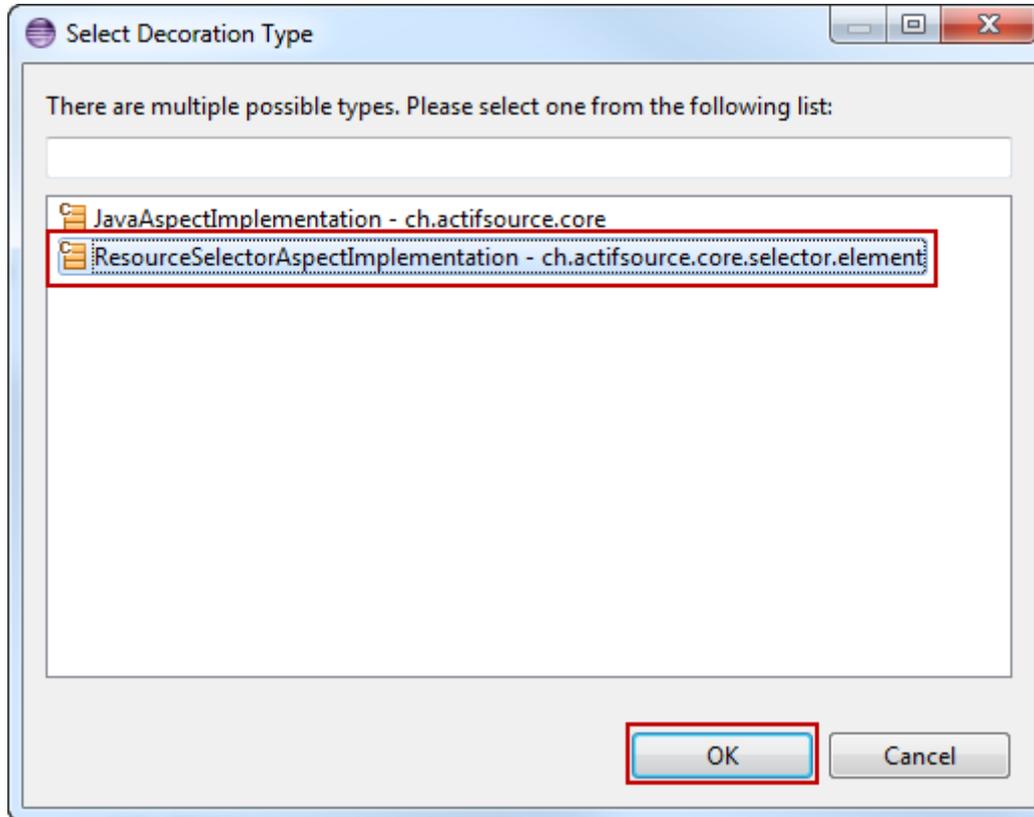
The screenshot shows a state machine diagram for `StateMachine2`. The diagram is structured as follows:

- StateMachine2** (typeOf: `com.actifsource.statemachine.generic.StateMachine`)
 - name: `StateMachine2`
 - event[1]: `open : Event`
 - event[2]: `close : Event`
 - state[1]: `Initialized : State`
 - transition[open]: `Transition` (typeOf: `com.actifsource.statemachine.generic.Transition`)
 - targetState:
 - event:
 - transition[close]: `Transition`
 - state[2]: `Opened : State`
 - state[3]: `Closed : State`

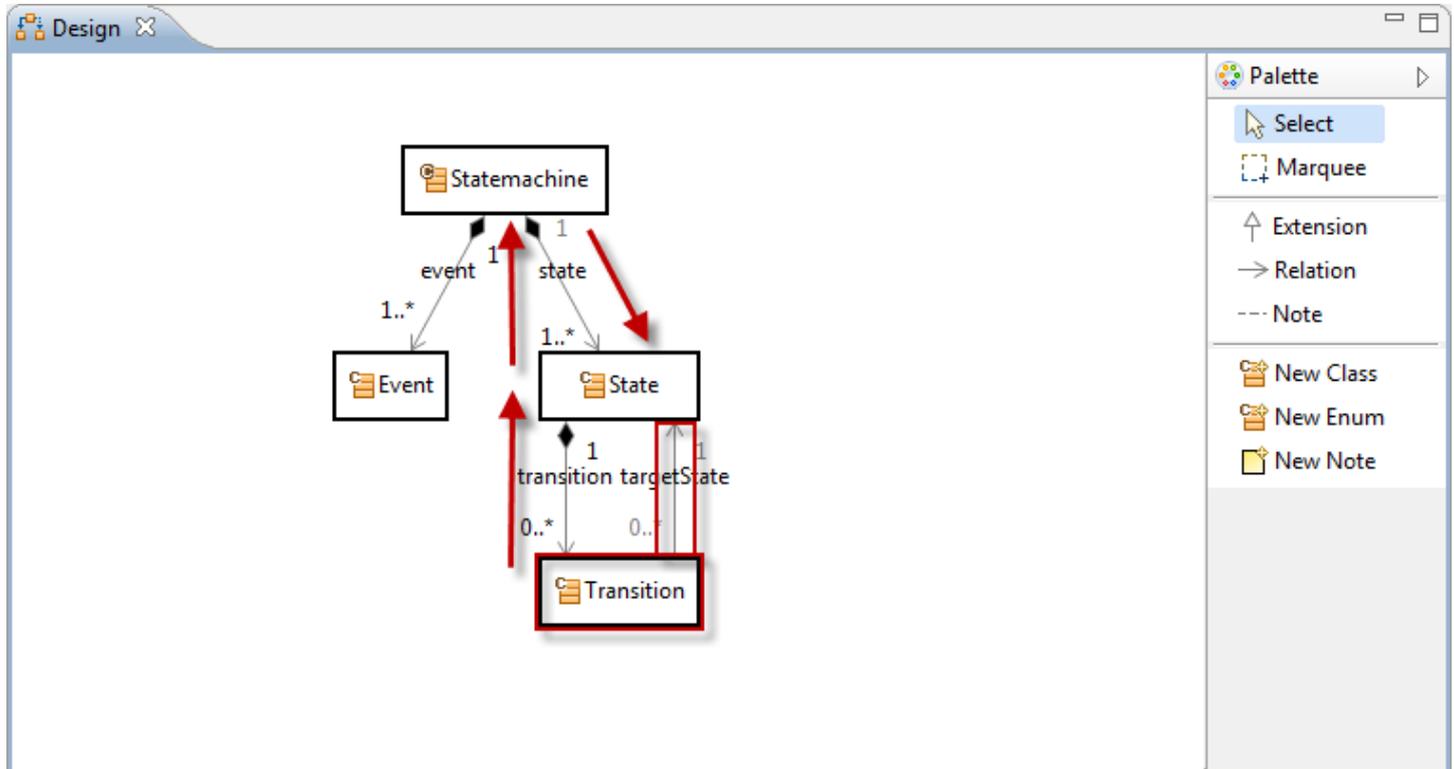
A dropdown menu is open for the `targetState` field of the `open` transition. The list of options is:

- Closed com.actifsource.statemachine.specific.StateMachine2 State
- Initialized com.actifsource.statemachine.specific.StateMachine1 State
- Initialized com.actifsource.statemachine.specific.StateMachine2 State
- Opened com.actifsource.statemachine.specific.StateMachine2 State**
- Started com.actifsource.statemachine.specific.StateMachine1 State
- Stopped com.actifsource.statemachine.specific.StateMachine1 State

- ↪ Create any new Transition
- ↪ Use **Content Assist** (Ctrl+Space) to add a targetState of type State
- ⓘ Note that all instances of State are listed instead of just the ones from StateMachine2



- ① Note that you can choose between a *JavaAspectImplementation* and a *SelectorAspectImplementation*
 - Selecting the *JavaAspectImplementation* allows you to write Java Code for complex operations
 - Selecting the *ResourceSelectorAspectImplementation* allows you to use the easy Selector syntax
- ↩ Select *ResourceSelectorAspectImplementation*
- ↩ Click OK



- ① Let's restrict the **range** of `targetState` to instances of States owned by the own State Machine
- ① The **useRelation** `targetState` is found in Transition
- ① We have to navigate from Transition to all States of the State Machine
 - Navigate backwards from Transition via `transition` to State
 - Navigate backwards from State via `state` to State Machine
 - Navigate forward from State Machine via `state` to State

The screenshot shows an IDE window titled "Statemachine2" with a tab for "*targetState". The breadcrumb path is "com.actifsource.statemachine.generic.Transition:Class" and the selected element is "targetState:UseRelation".

The left sidebar shows a list of properties for the selected element, with "name" highlighted. The main area displays the configuration for the "UseRelation" aspect:

- targetState** (text field)
- Cardinality1_1
- ResourceSelectorAspectImplementation** (typeOf)
- ch.actifsource.core.UseRelation.UseRangeRestrictionAspect (implements)
- Transition.-transition.-state.state** (selector, highlighted with a red box)
- Cardinality0_N
- com.actifsource.statemachine.generic.State (range)

↩ Enter the Selector `Transition.-transition.-state.state` using Content Assist (Ctrl+Space)

The screenshot shows the IDE's state machine editor for `StateMachine2`. The breadcrumb path is `com.actifsource.statemachine.specific.StateMachine2:StateMachine2` > `Initialized:State` > `open:Transition`. The diagram shows a state machine with the following structure:

- `StateMachine2` (typeOf `com.actifsource.statemachine.generic.StateMachine`)
 - name: `StateMachine2`
 - event[1]: `open : Event`
 - event[2]: `close : Event`
 - state[1]:
 - typeOf: `com.actifsource.statemachine.generic.State`
 - name: `Initialized`
 - transition[open]:
 - typeOf: `com.actifsource.statemachine.generic.Transition`
 - targetState: (property being edited)
 - event: (property)
 - transition[close]: (property)
 - state[2]: `Opened : State`
 - state[3]: `Closed : State`

The content assist popup shows the following list of `State` instances:

new	com.actifsource.statemachine.generic	State
Closed	com.actifsource.statemachine.specific.StateMachine2	State
Initialized	com.actifsource.statemachine.specific.StateMachine2	State
Opened	com.actifsource.statemachine.specific.StateMachine2	State

- ↩ Use **Content Assist** (Ctrl+Space) again to add the `targetState` `Opened` of type `State`
- ⓘ Note that only instances of `State` from `StateMachine2` are listed

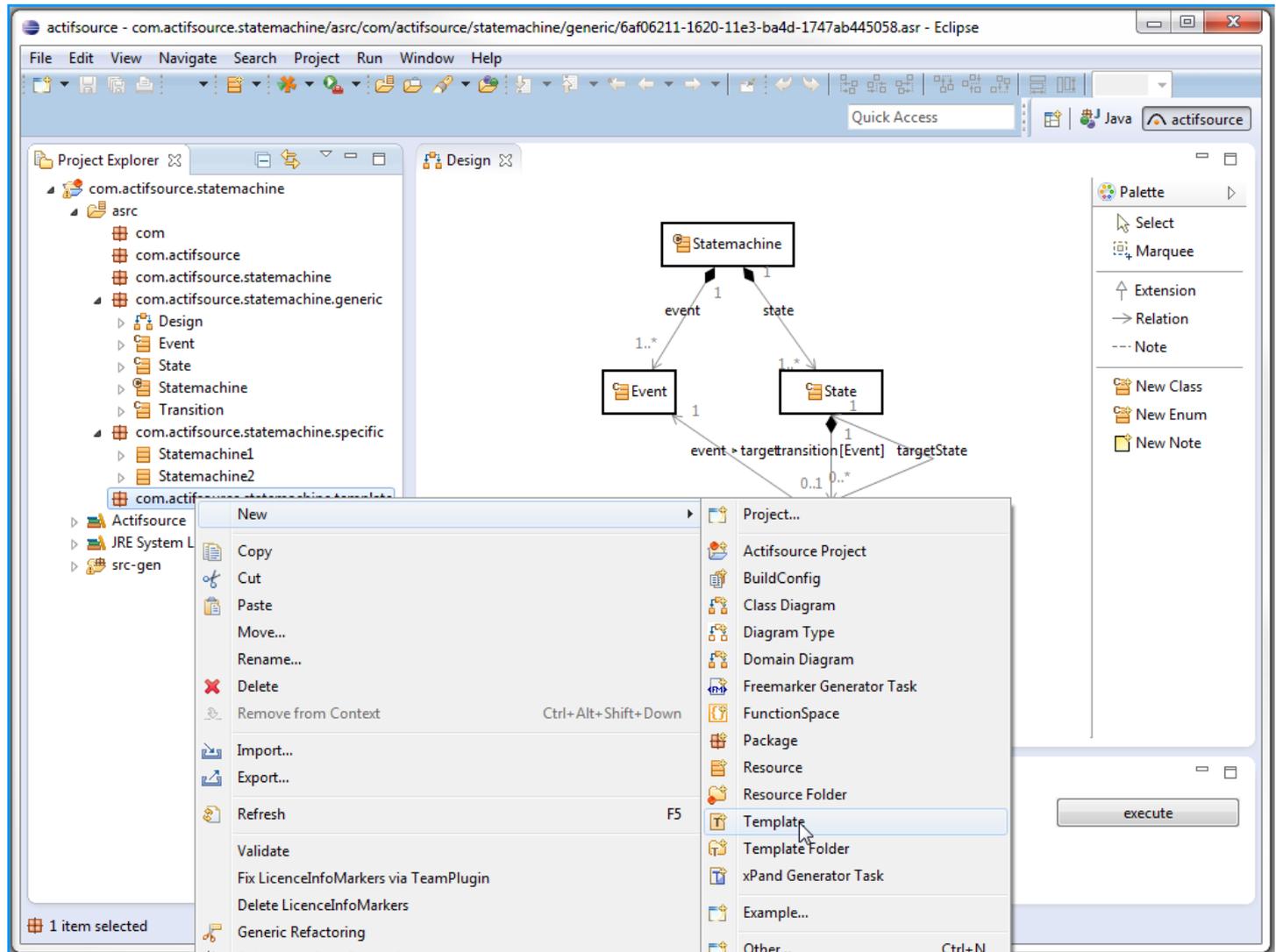
The screenshot shows the IDE interface for the class `com.actifsource.statemachine.specific.StateMachine2:StateMachine`. The class is expanded to show its state transitions. The `state[1]` field is highlighted in orange, and the `state[2]` and `state[3]` fields are highlighted in red. The `state[2]` field is expanded to show its transition to `state[3]`.

Field	Value																		
<code>typeOf</code>	<code>com.actifsource.statemachine.generic.StateMachine</code>																		
<code>name</code>	<code>StateMachine2</code>																		
<code>event[1]</code>	<code>open : Event</code>																		
<code>event[2]</code>	<code>close : Event</code>																		
<code>state[1]</code>	<code>Initialized : State</code>																		
<code>state[2]</code>	<table border="1"> <thead> <tr> <th>Field</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><code>typeOf</code></td> <td><code>com.actifsource.statemachine.generic.State</code></td> </tr> <tr> <td><code>name</code></td> <td><code>Opened</code></td> </tr> <tr> <td><code>transition[open]</code></td> <td></td> </tr> <tr> <td><code>transition[close]</code></td> <td> <table border="1"> <thead> <tr> <th>Field</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><code>typeOf</code></td> <td><code>com.actifsource.statemachine.generic.Transition</code></td> </tr> <tr> <td><code>targetState</code></td> <td><code>Closed</code></td> </tr> <tr> <td><code>event</code></td> <td><code>com.actifsource.statemachine.specific.StateMachine2.close</code></td> </tr> </tbody> </table> </td> </tr> </tbody> </table>	Field	Value	<code>typeOf</code>	<code>com.actifsource.statemachine.generic.State</code>	<code>name</code>	<code>Opened</code>	<code>transition[open]</code>		<code>transition[close]</code>	<table border="1"> <thead> <tr> <th>Field</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><code>typeOf</code></td> <td><code>com.actifsource.statemachine.generic.Transition</code></td> </tr> <tr> <td><code>targetState</code></td> <td><code>Closed</code></td> </tr> <tr> <td><code>event</code></td> <td><code>com.actifsource.statemachine.specific.StateMachine2.close</code></td> </tr> </tbody> </table>	Field	Value	<code>typeOf</code>	<code>com.actifsource.statemachine.generic.Transition</code>	<code>targetState</code>	<code>Closed</code>	<code>event</code>	<code>com.actifsource.statemachine.specific.StateMachine2.close</code>
Field	Value																		
<code>typeOf</code>	<code>com.actifsource.statemachine.generic.State</code>																		
<code>name</code>	<code>Opened</code>																		
<code>transition[open]</code>																			
<code>transition[close]</code>	<table border="1"> <thead> <tr> <th>Field</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><code>typeOf</code></td> <td><code>com.actifsource.statemachine.generic.Transition</code></td> </tr> <tr> <td><code>targetState</code></td> <td><code>Closed</code></td> </tr> <tr> <td><code>event</code></td> <td><code>com.actifsource.statemachine.specific.StateMachine2.close</code></td> </tr> </tbody> </table>	Field	Value	<code>typeOf</code>	<code>com.actifsource.statemachine.generic.Transition</code>	<code>targetState</code>	<code>Closed</code>	<code>event</code>	<code>com.actifsource.statemachine.specific.StateMachine2.close</code>										
Field	Value																		
<code>typeOf</code>	<code>com.actifsource.statemachine.generic.Transition</code>																		
<code>targetState</code>	<code>Closed</code>																		
<code>event</code>	<code>com.actifsource.statemachine.specific.StateMachine2.close</code>																		
<code>state[3]</code>	<table border="1"> <thead> <tr> <th>Field</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><code>typeOf</code></td> <td><code>com.actifsource.statemachine.generic.State</code></td> </tr> <tr> <td><code>name</code></td> <td><code>Closed</code></td> </tr> <tr> <td><code>transition[open]</code></td> <td> <table border="1"> <thead> <tr> <th>Field</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><code>typeOf</code></td> <td><code>com.actifsource.statemachine.generic.Transition</code></td> </tr> <tr> <td><code>targetState</code></td> <td><code>Opened</code></td> </tr> <tr> <td><code>event</code></td> <td><code>com.actifsource.statemachine.specific.StateMachine2.open</code></td> </tr> </tbody> </table> </td> </tr> <tr> <td><code>transition[close]</code></td> <td></td> </tr> </tbody> </table>	Field	Value	<code>typeOf</code>	<code>com.actifsource.statemachine.generic.State</code>	<code>name</code>	<code>Closed</code>	<code>transition[open]</code>	<table border="1"> <thead> <tr> <th>Field</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><code>typeOf</code></td> <td><code>com.actifsource.statemachine.generic.Transition</code></td> </tr> <tr> <td><code>targetState</code></td> <td><code>Opened</code></td> </tr> <tr> <td><code>event</code></td> <td><code>com.actifsource.statemachine.specific.StateMachine2.open</code></td> </tr> </tbody> </table>	Field	Value	<code>typeOf</code>	<code>com.actifsource.statemachine.generic.Transition</code>	<code>targetState</code>	<code>Opened</code>	<code>event</code>	<code>com.actifsource.statemachine.specific.StateMachine2.open</code>	<code>transition[close]</code>	
Field	Value																		
<code>typeOf</code>	<code>com.actifsource.statemachine.generic.State</code>																		
<code>name</code>	<code>Closed</code>																		
<code>transition[open]</code>	<table border="1"> <thead> <tr> <th>Field</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><code>typeOf</code></td> <td><code>com.actifsource.statemachine.generic.Transition</code></td> </tr> <tr> <td><code>targetState</code></td> <td><code>Opened</code></td> </tr> <tr> <td><code>event</code></td> <td><code>com.actifsource.statemachine.specific.StateMachine2.open</code></td> </tr> </tbody> </table>	Field	Value	<code>typeOf</code>	<code>com.actifsource.statemachine.generic.Transition</code>	<code>targetState</code>	<code>Opened</code>	<code>event</code>	<code>com.actifsource.statemachine.specific.StateMachine2.open</code>										
Field	Value																		
<code>typeOf</code>	<code>com.actifsource.statemachine.generic.Transition</code>																		
<code>targetState</code>	<code>Opened</code>																		
<code>event</code>	<code>com.actifsource.statemachine.specific.StateMachine2.open</code>																		
<code>transition[close]</code>																			

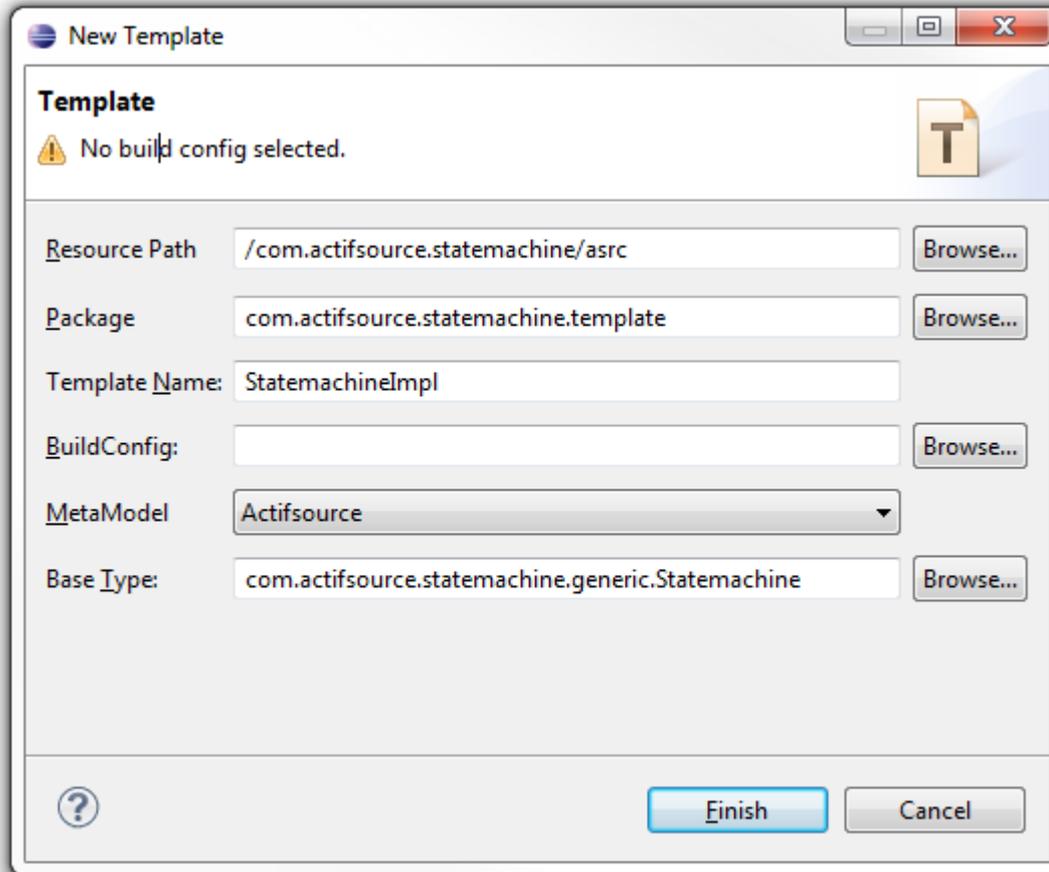
- ① Get familiar with **Decorating Relations** and **Range Restrictions**
- ① Write an **actifsource Code Template** to generate a state machine

Code Template for Statemachines

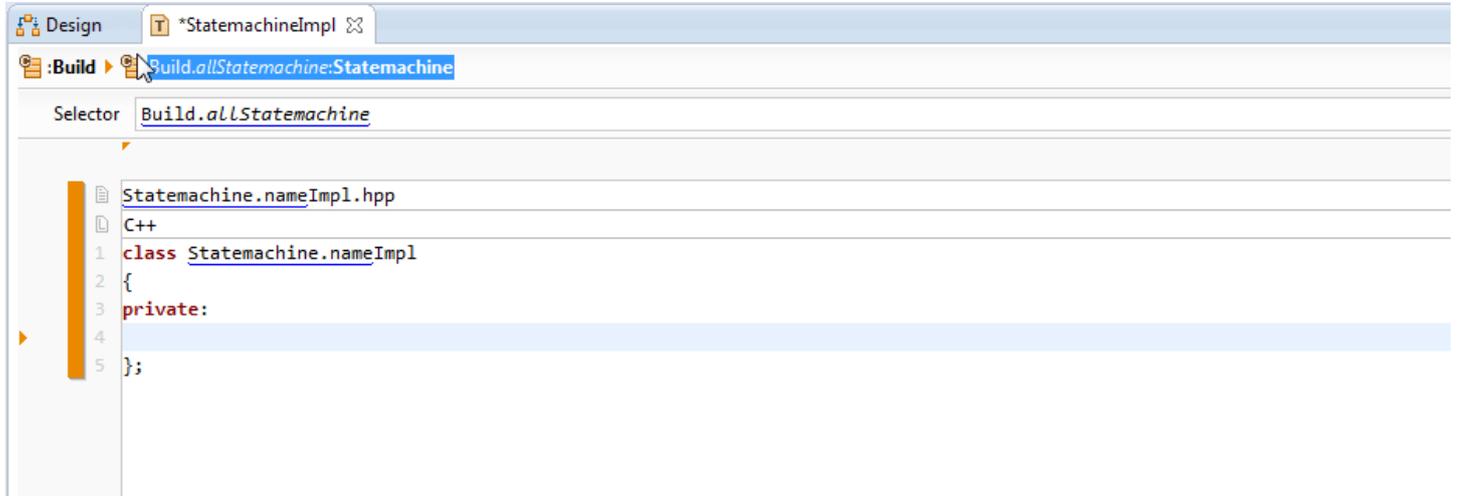
- ① Write a code template for instances of Statemachine



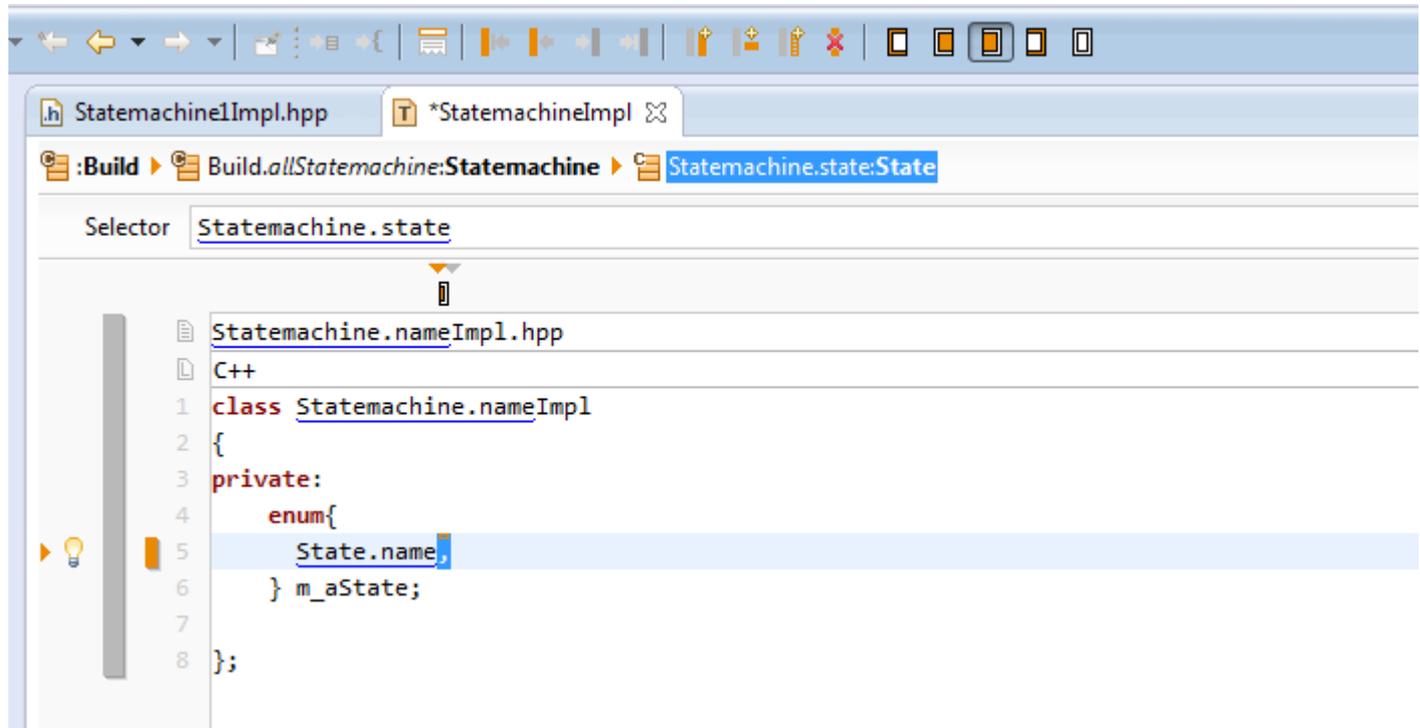
- ↩ Create a package `com.actifsource.statemachine.template`
- ↩ Select the new package and choose **New->Template** from the context menu.



- ↵ Insert `StateMachineImpl` as Template Name
- ↵ Choose the Base Type `com.actifsource.statemachine.generic.StateMachine`
- ↵ Click **Finish**

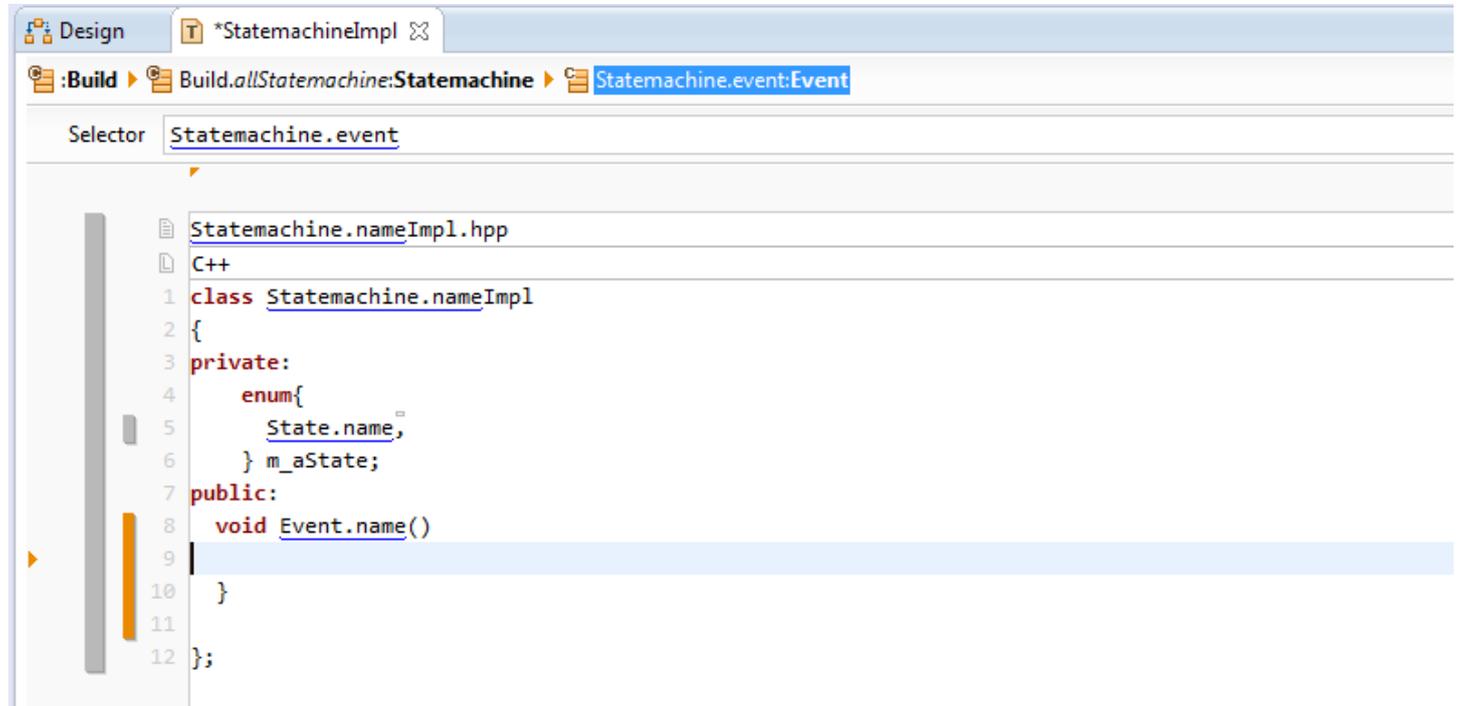


- ↪ Insert State machine.nameImpl.hpp on the Filename Line and make sure that the language (C++) is automatically detected.
- ↪ Write the skeleton for a class State machine.nameImpl



Next, we define an enumeration variable with the all the States of a StateMachine as enumerators. This variable stores the current state of a StateMachine:

- ↪ Write the declaration of enumeration variable *m_aState*
- ↪ Insert a LineContext in the enumeration list and choose the Selector StateMachine.state with the support of the Content Assist
- ↪ Insert State.name in the newly created LineContext. Append a ','. Then mark the ',' and select `NotLast` to make sure that there is no comma after the last entry in the enumeration list.

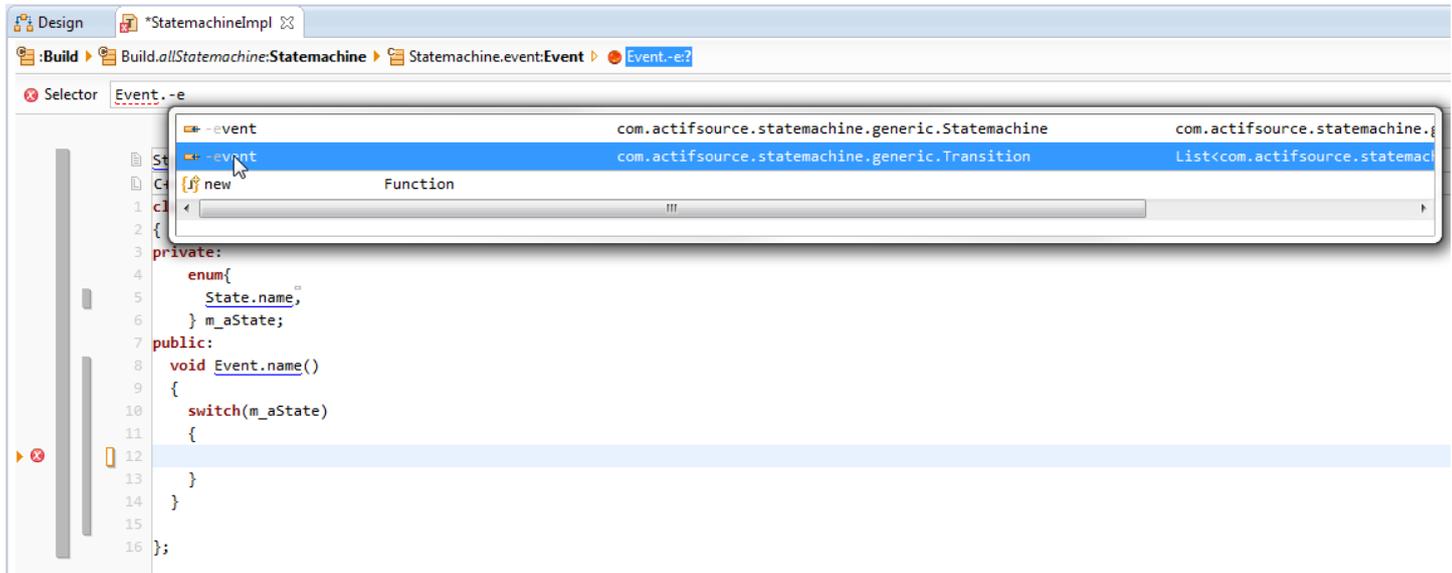


The screenshot shows an IDE window titled '*StateMachineImpl'. The breadcrumb path is ':Build > Build.allStateMachine:StateMachine > StateMachine.event:Event'. The 'Selector' field contains 'StateMachine.event'. The code editor shows the following C++ code:

```
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 }
11
12};
```

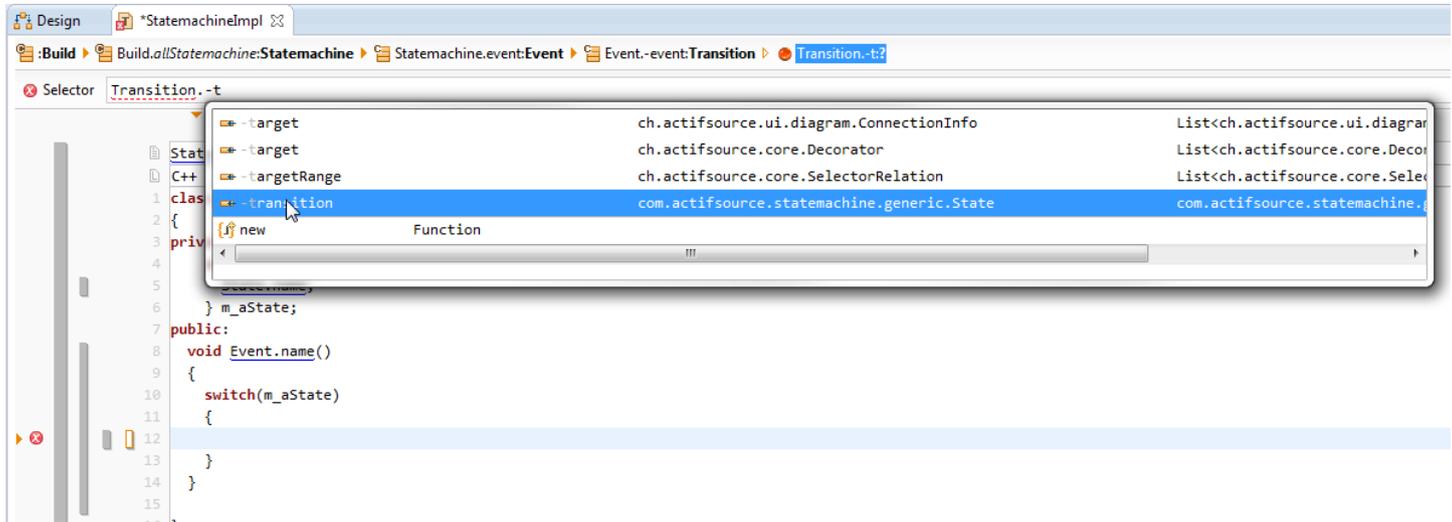
We define a member function for each event of our StateMachine, which will later handle all the possible transitions triggered by the event:

- ↪ Create a new LineContext and choose StateMachine.event as the selector of the line context
- ↪ Write the skeleton of a function returning void named Event.name



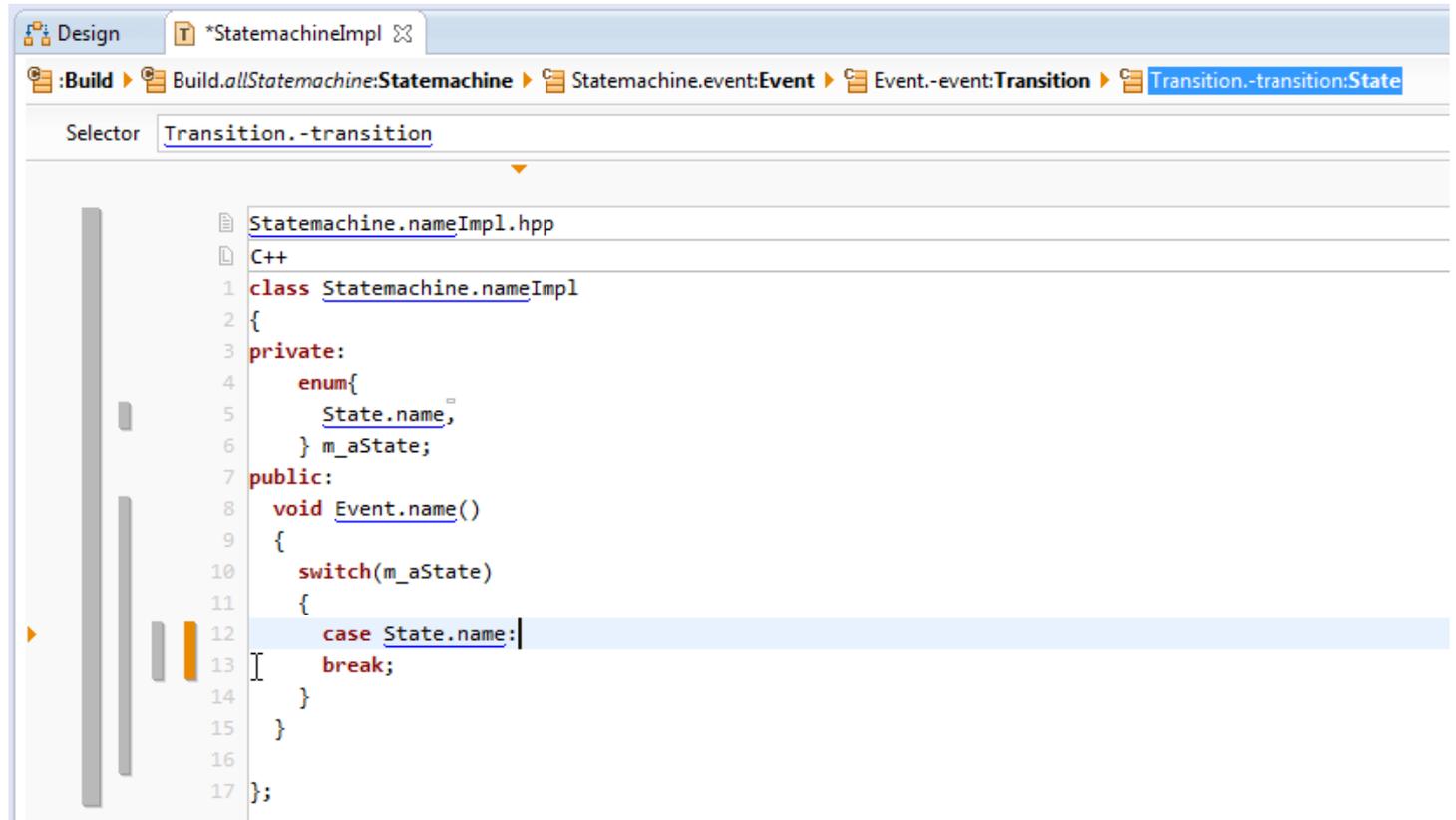
We write a switch-statement with the current state `m_aState` as control variable and define a `LineContext` that iterates over all `Transitions` referring to an `Event` through the relation `Transition.event`:

- ↪ Create a switch-statement with the `m_aState` as control variable
- ↪ Create a `LineContext` inside the switch-statement
- ↪ Choose `Event.-event` as the Selector of the new `LineContext`



We create a LineContext that iterates over all States that refer to a Transition through the relation State.transition:

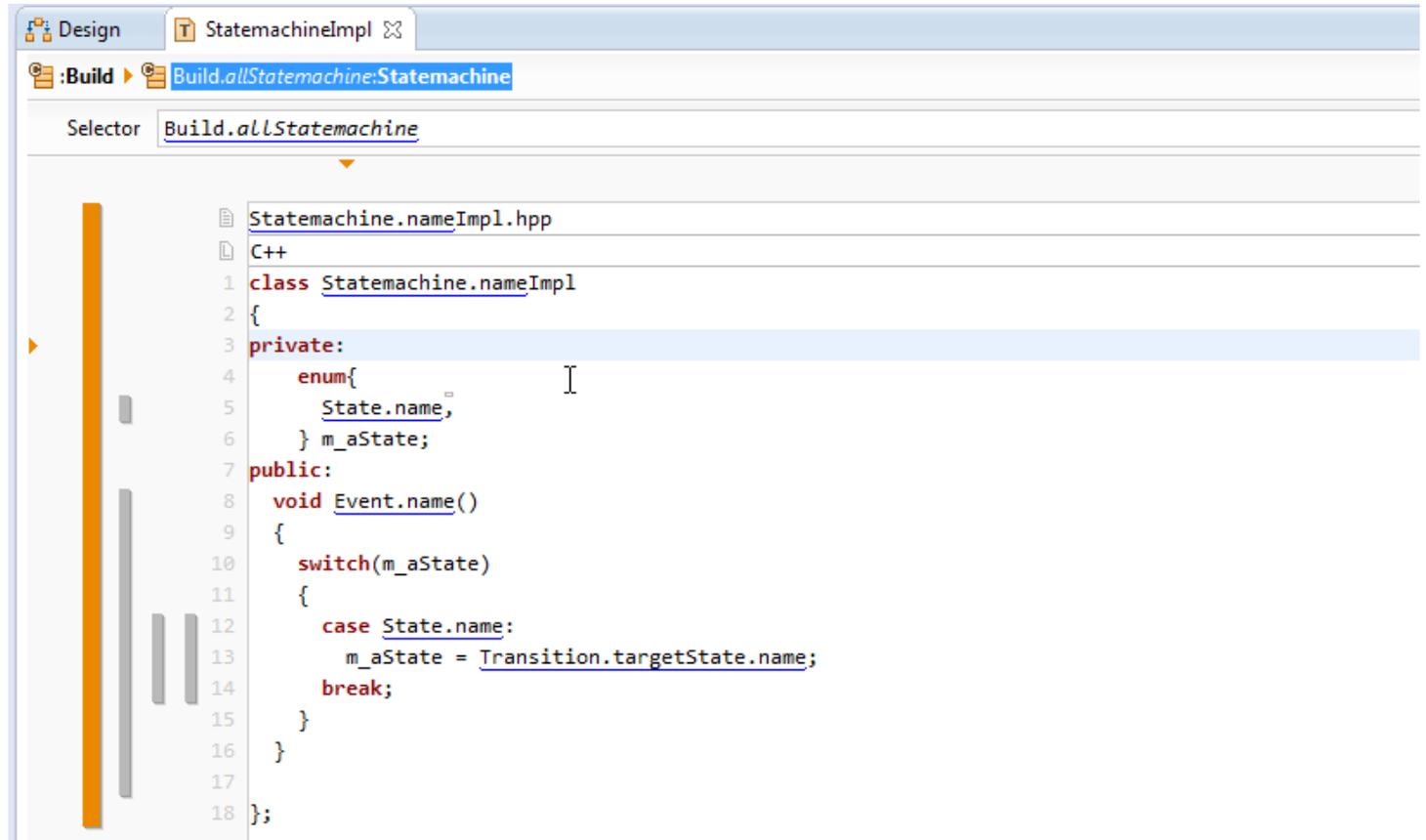
- ↪ Create a LineContext on the same line as LineContext that we have created before
- ↪ Choose Transition.-transition as the Selector of the new LineContext



```
Design | *StateachineImpl x
Build > Build.allStateachine:Stateachine > Stateachine.event:Event > Event.-event:Transition > Transition.-transition:State
Selector: Transition.-transition
Stateachine.nameImpl.hpp
C++
1 class Stateachine.nameImpl
2 {
3 private:
4     enum
5     {
6         State.name,
7     } m_aState;
8 public:
9     void Event.name()
10    {
11        switch(m_aState)
12        {
13            case State.name:
14                break;
15        }
16    }
17};
```

We write a case-statement for each State that is (indirectly) referring to an Event through State.transition.event:

- ↪ Insert a case State.name and add a `break` at the end of the case-statement

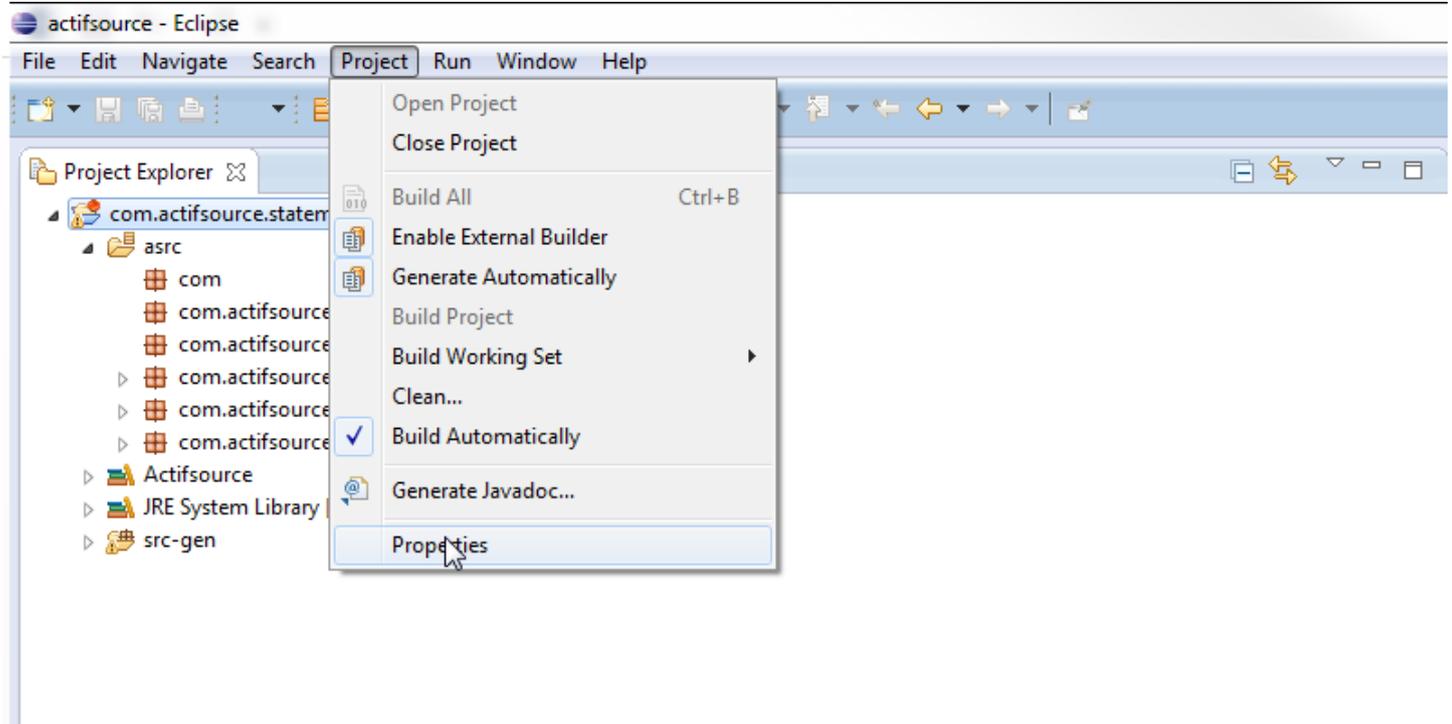


The screenshot shows a C++ IDE with a project named 'StatemachineImpl'. The 'Build' menu is open, and the 'Build.allStatemachine:Statemachine' option is selected. The 'Selector' field contains 'Build.allStatemachine'. The main editor displays the file 'Statemachine.nameImpl.hpp' with the following code:

```
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};
```

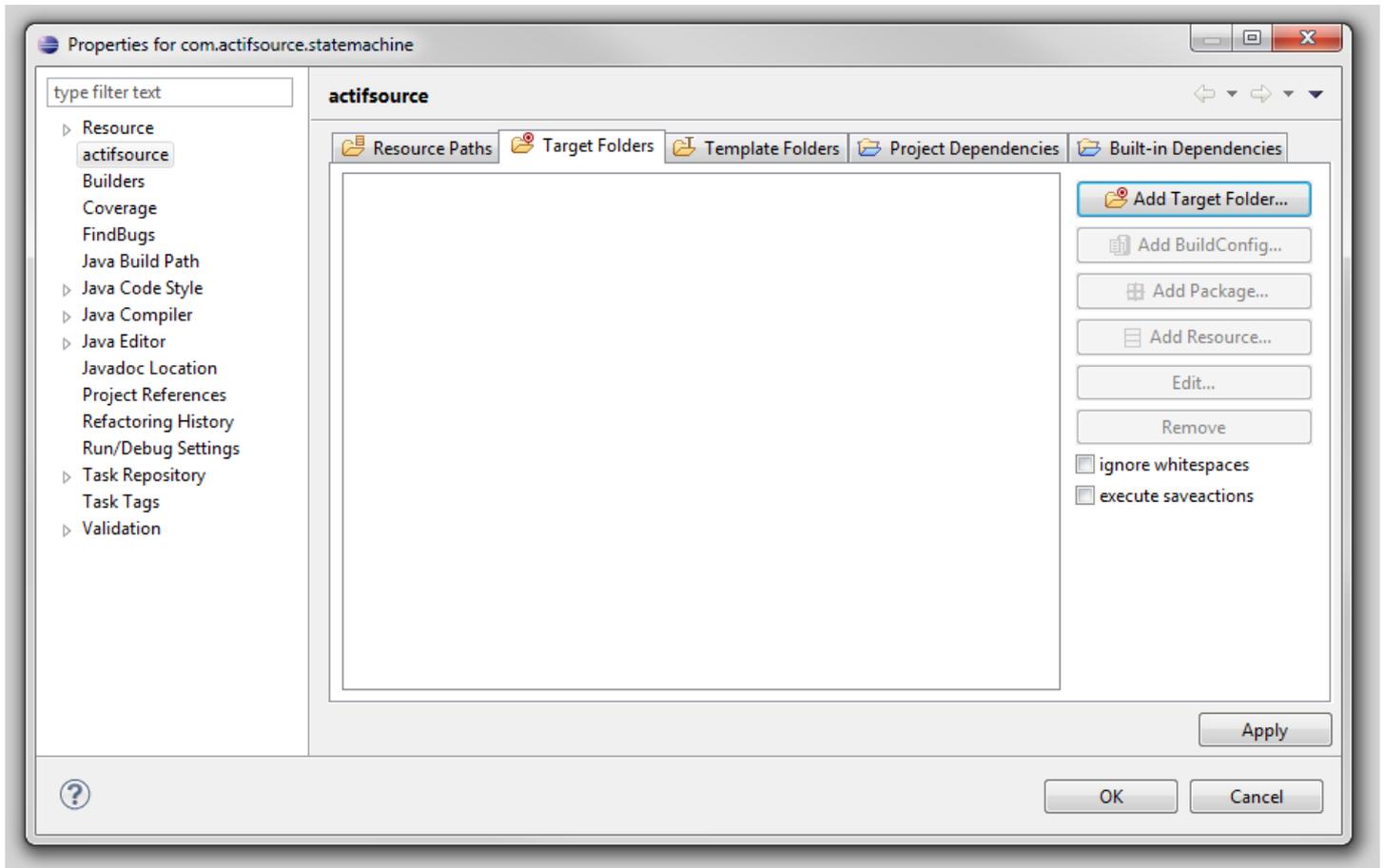
We update the current state as follows: We first select for an Event the Transitions that refer to the Event through Transition.event. For each Transition, we select the States that are connected to Transition by State.transition. For each State, it holds that if the current state m_aState is equal to State, then the new State of the Statemachine is Transition.targetState:

✎ Write code to assign Transition.targetState.name to the variable m_aState

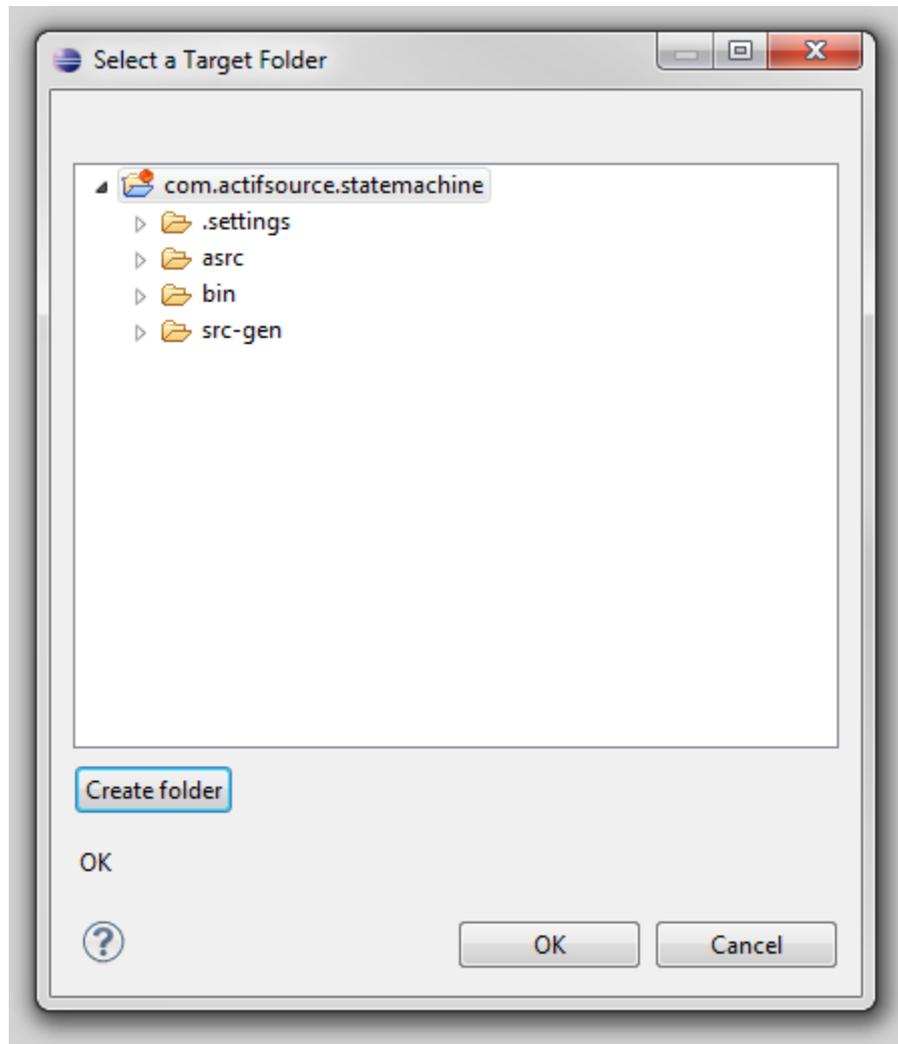


In order to generate code from the template we have implemented before, we setup the project properties for Actifsource:

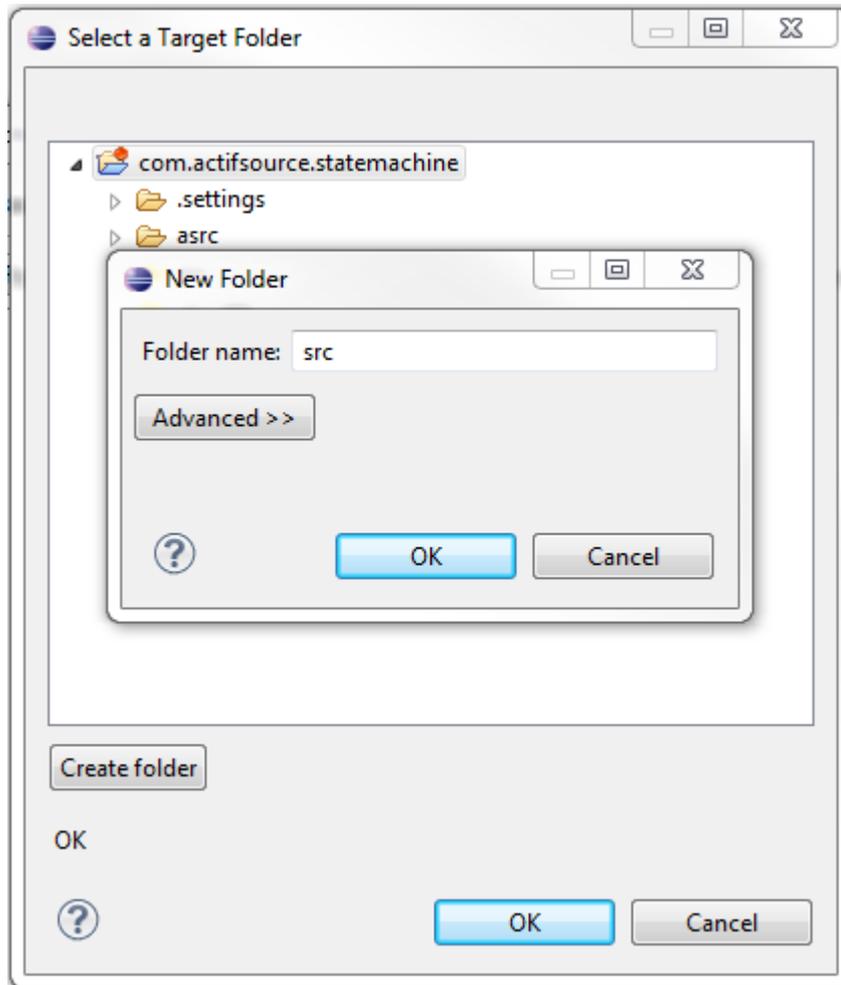
- ✎ Select the project `com.actifsource.statemachine` and choose **Project->Properties** from the main menu



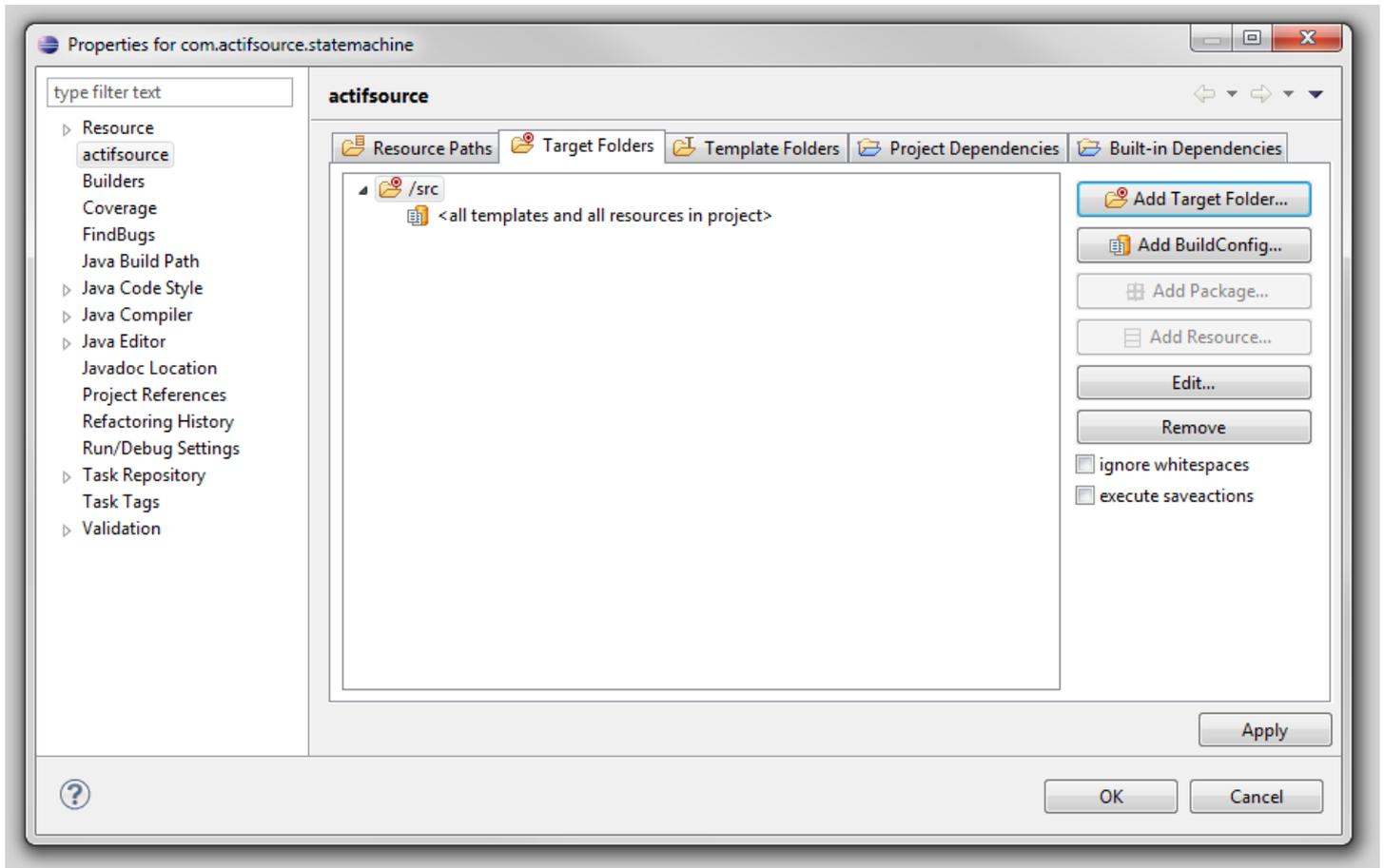
↩ In the Properties dialog choose **actifsource** and select the tab **Target Folders**



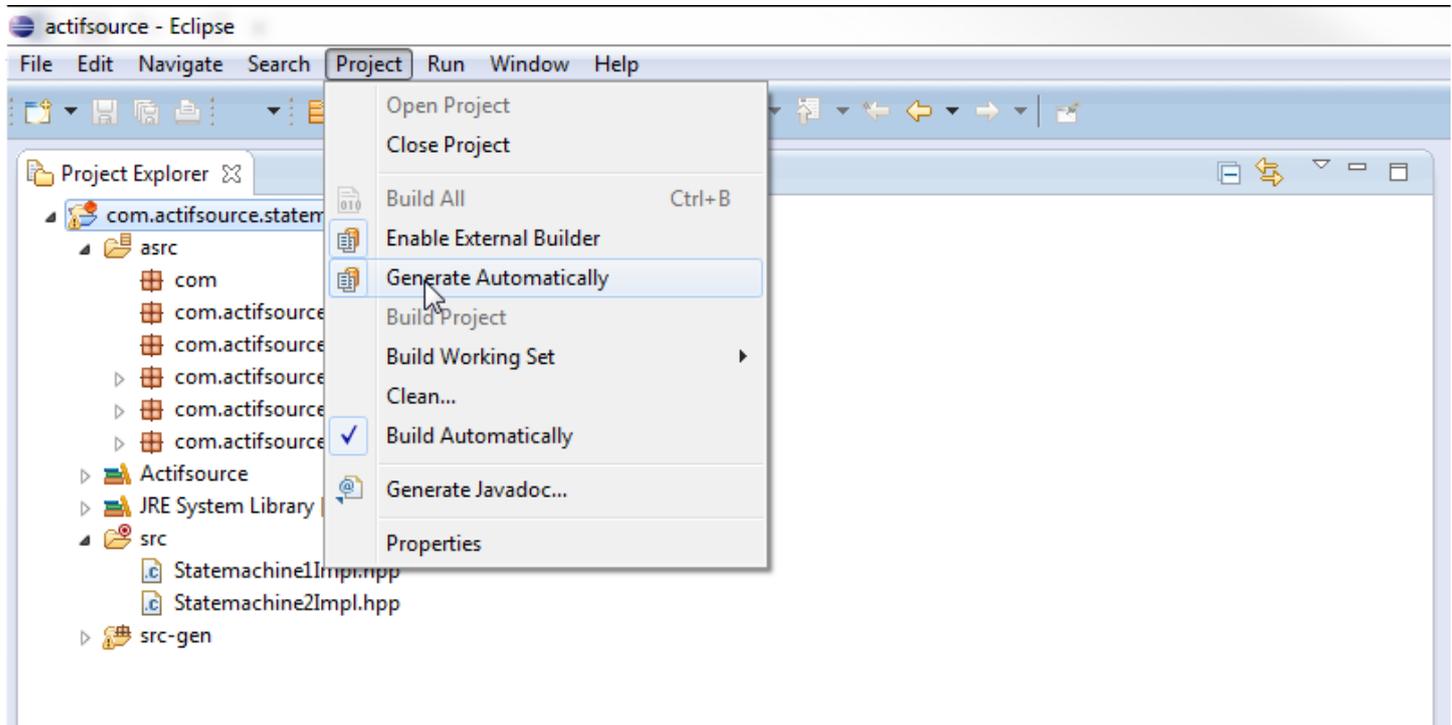
↩ In the dialog **Select Target Folder**, click on the button **Create folder**



- ↵ Enter `src` as Folder Name in the **New Folder** dialog
- ↵ Click on **OK** in the New Folder dialog and then in the **Select Target Folder** dialog



↩ Check the settings on the **Target Folders** tab and close the dialog by clicking on **OK**



The code generator now applies the template `StatemachineImpl` to the two `Statemachine` instances and stores the resulting files to the `src` folder:

- ↪ Open the `src` folder and check that the two files `Statemachine1Impl.hpp` and `Statemachine2Impl.hpp` have been generated
- ↪ If the files have not been generated, make sure that **Generate Automatically** is active under **Project** in the main menu

```

StateMachine1Impl.hpp
class StateMachine1Impl
{
private:
    enum{
        Initialized,
        Started,
        Stopped
    } m_aState;

public:
    void start()
    {
        switch(m_aState)
        {
            case Initialized:
                m_aState = Started;
                break;
            case Stopped:
                m_aState = Started;
                break;
        }
    }

    void stop()
    {
        switch(m_aState)
        {
            case Started:
                m_aState = Stopped;
                break;
        }
    }
};
/* Actifsource ID=[0b6aff85-85f9-11e4-a105-d1f

StateMachine2Impl.hpp
class StateMachine2Impl
{
private:
    enum{
        Initialized,
        Opened,
        Closed
    } m_aState;

public:
    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;
        }
    }
};
/* Actifsource ID=[0b6aff85-85f9-11e4-a105-c

```

- ↪ Open the newly generated files and inspect and compare the code for the two State machines
- ① Learn how to extend the StateMachine by conditional transitions and actions executed together with a transition by working through the Actifsource Tutorial – Code Snippets
- ① Complete the generated classes by adding a member function initialize()

