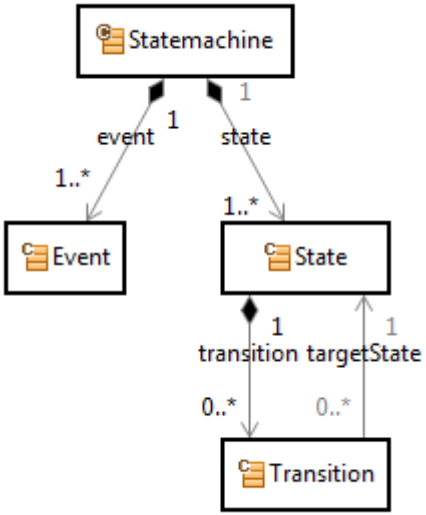
Tutorial

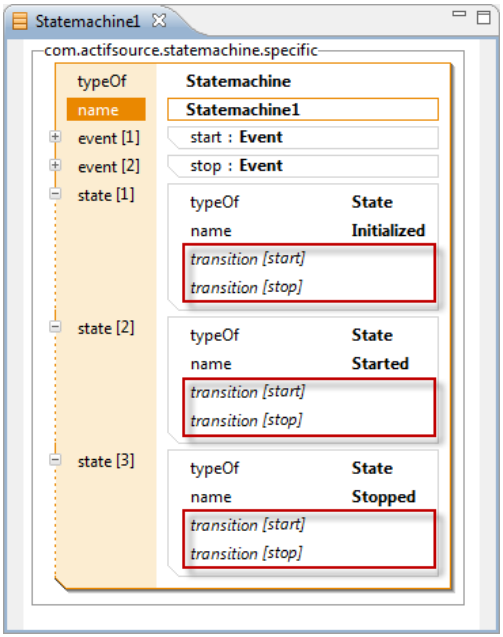
State Machine

Tutorial	Actifsource Tutorial – State Machine
Required Time	• 40 Minutes
Prerequisites	• Actifsource Tutorial – Installing Actifsource • Actifsource Tutorial – Simple Service • Actifsource Tutorial – Complex Service
Goal	• 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	• Decorating Relation Aspect • Range Restriction Aspect • Selector (forward and reverse selection)
Notation	↳ To do ⓘ Information • **Bold**: Terms from actifsource or other technologies and tools • **Bold underlined**: actifsource Resources • Underlined: User Resources • *UnderlinedItalics*: Resource Functions • `Monospaced`: User input • *Italics*: Important terms in current situation
Disclaimer	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.
Contact	**actifsource GmbH** Täfernstrasse 37 5405 Baden-Dättwil Switzerland www.actifsource.com
Trademark	**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.

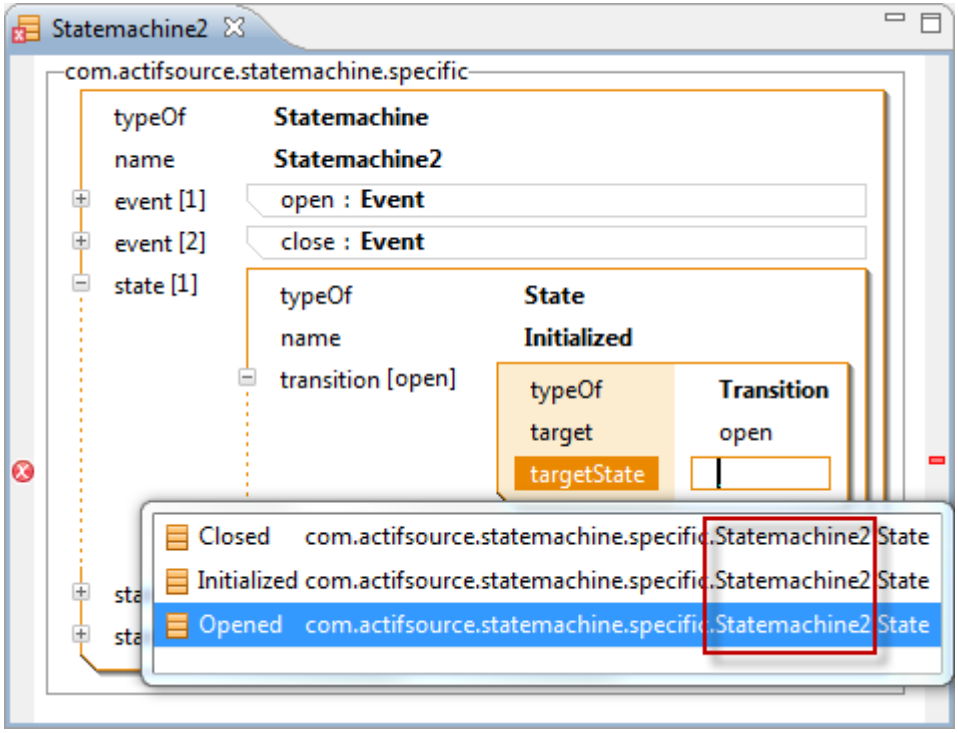
- Create a simple state machine



- Show possible events in every transition

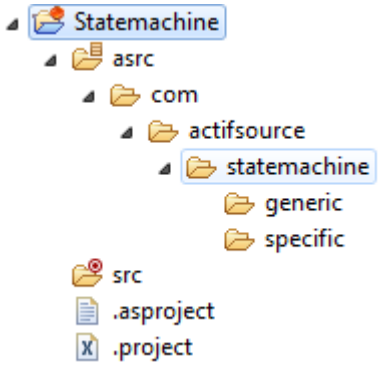


- Restrict transition target to state instances of the own state machine



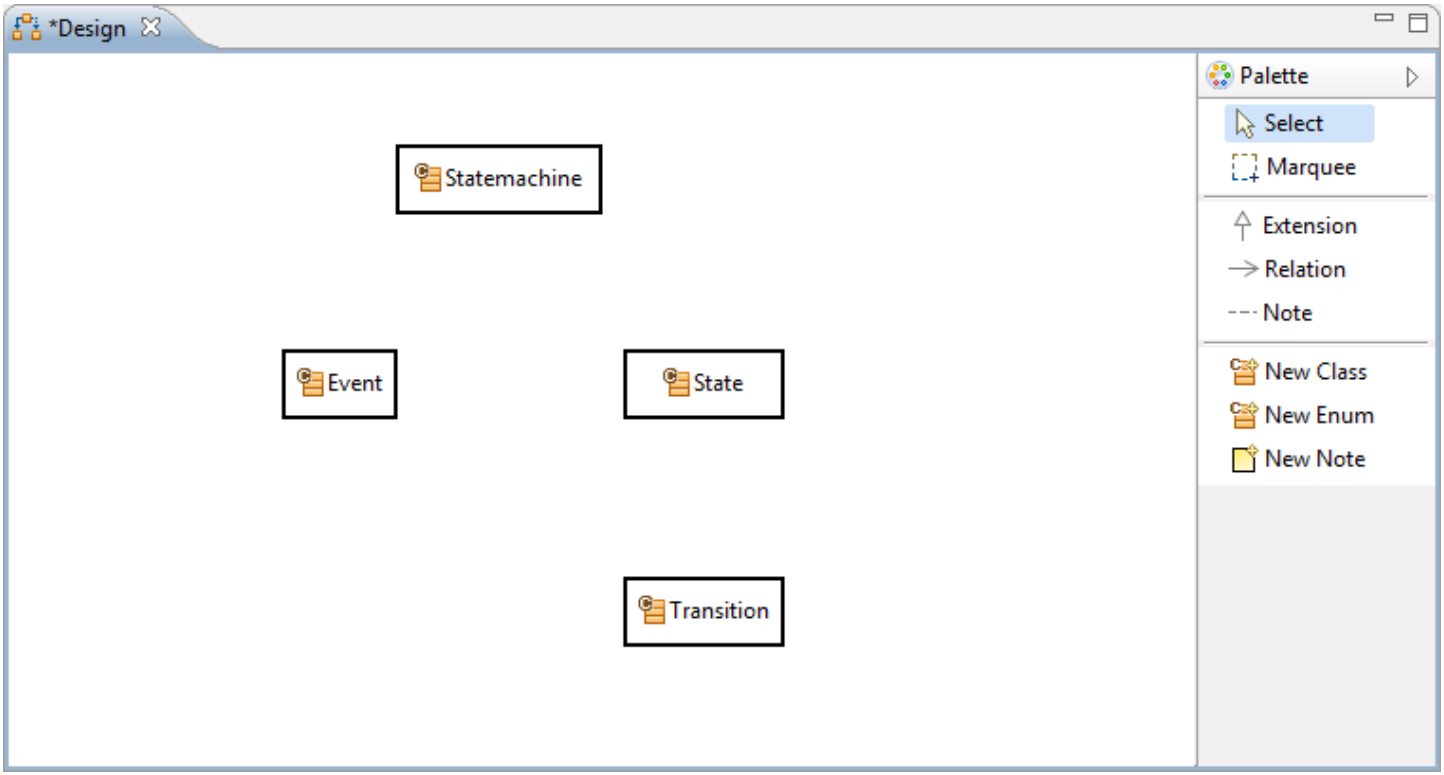
Part I: Preparation

- Prepare a new **actifsource Project** named Statemachine as seen in the *Actifsource Tutorial – Simple Service*
  - Setup the Target Folder *src*
- Use the following package structure

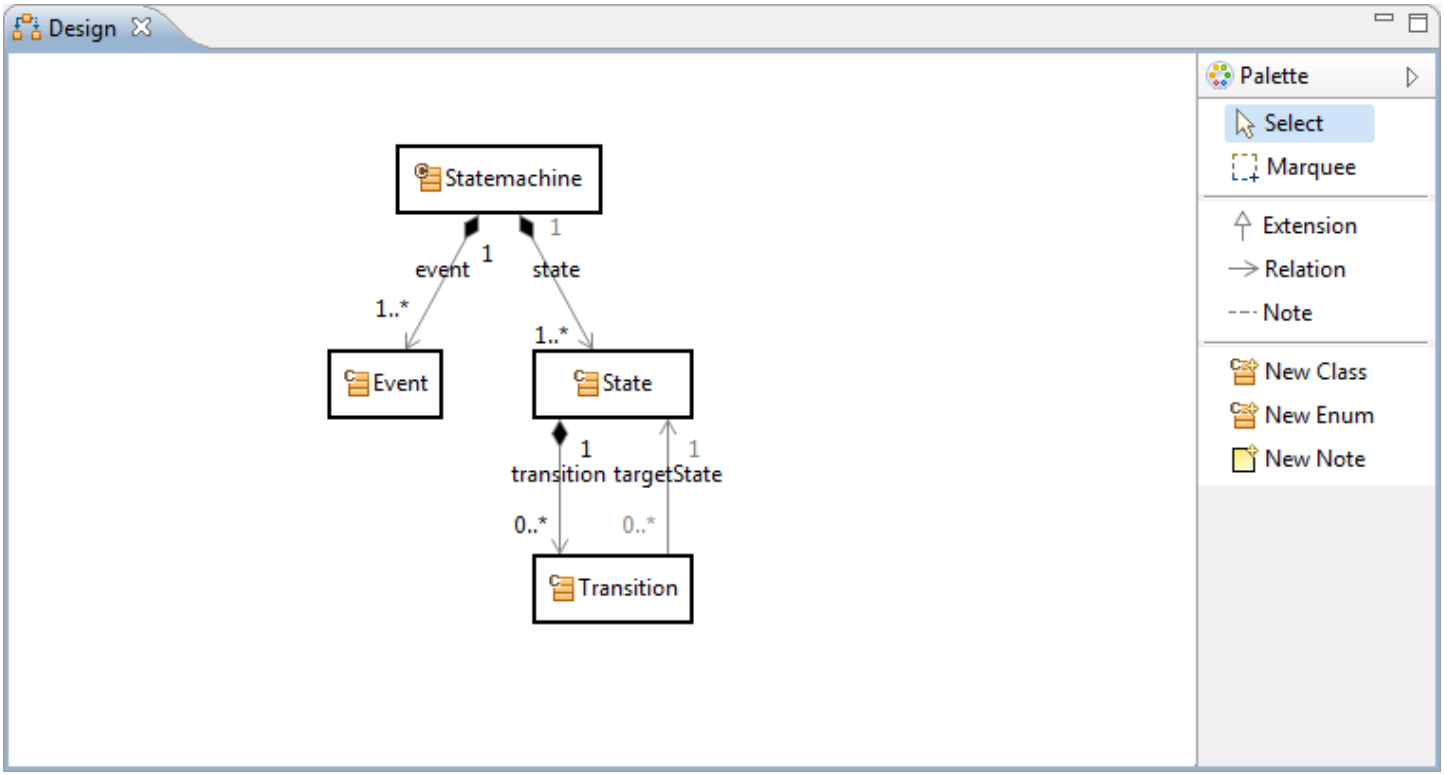


Part II: Create a State Machine

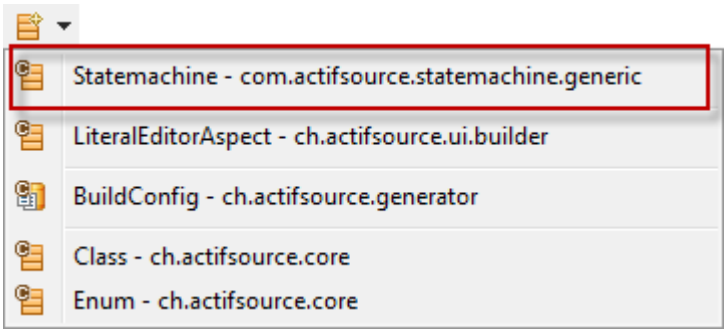
- ⓘ Create a simple state machine
- ⓘ Instantiate the state machine and see its deficits

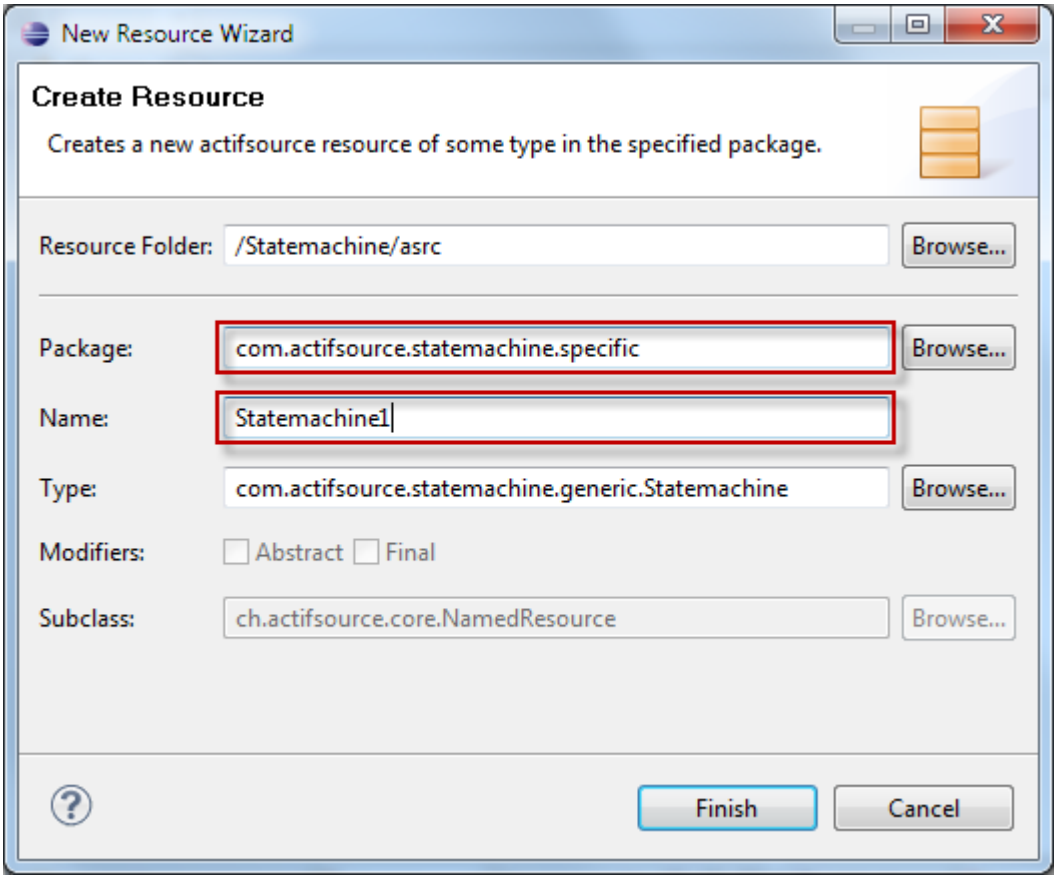


- Create a **Generic Domain Model** named *Design* in the **Package** *generic* using the **DiagramEditor**
- The Design shall contain the following **Domain Classes**
  - Statemachine, Event, State, Transition

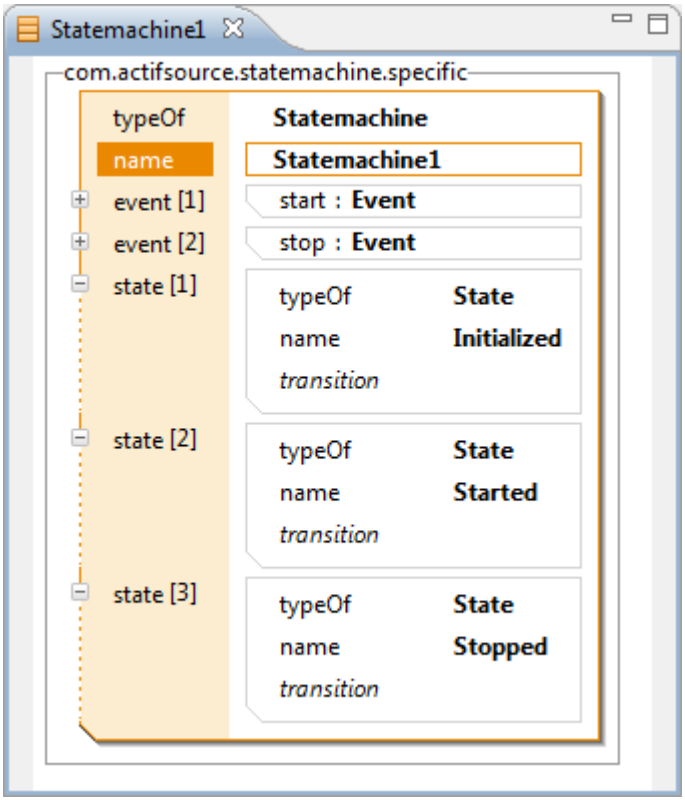


- Insert a **OwnRelation** between
  - Statemachine and Event
  - Statemachine and State
- Insert a **DecoratingRelation** between
  - State and Transition
- Insert a **UseRelation** between
  - Transition and State
- Adjust the **Cardinalities** as shown above
- ⓘ Warning: The layout for the realtions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





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

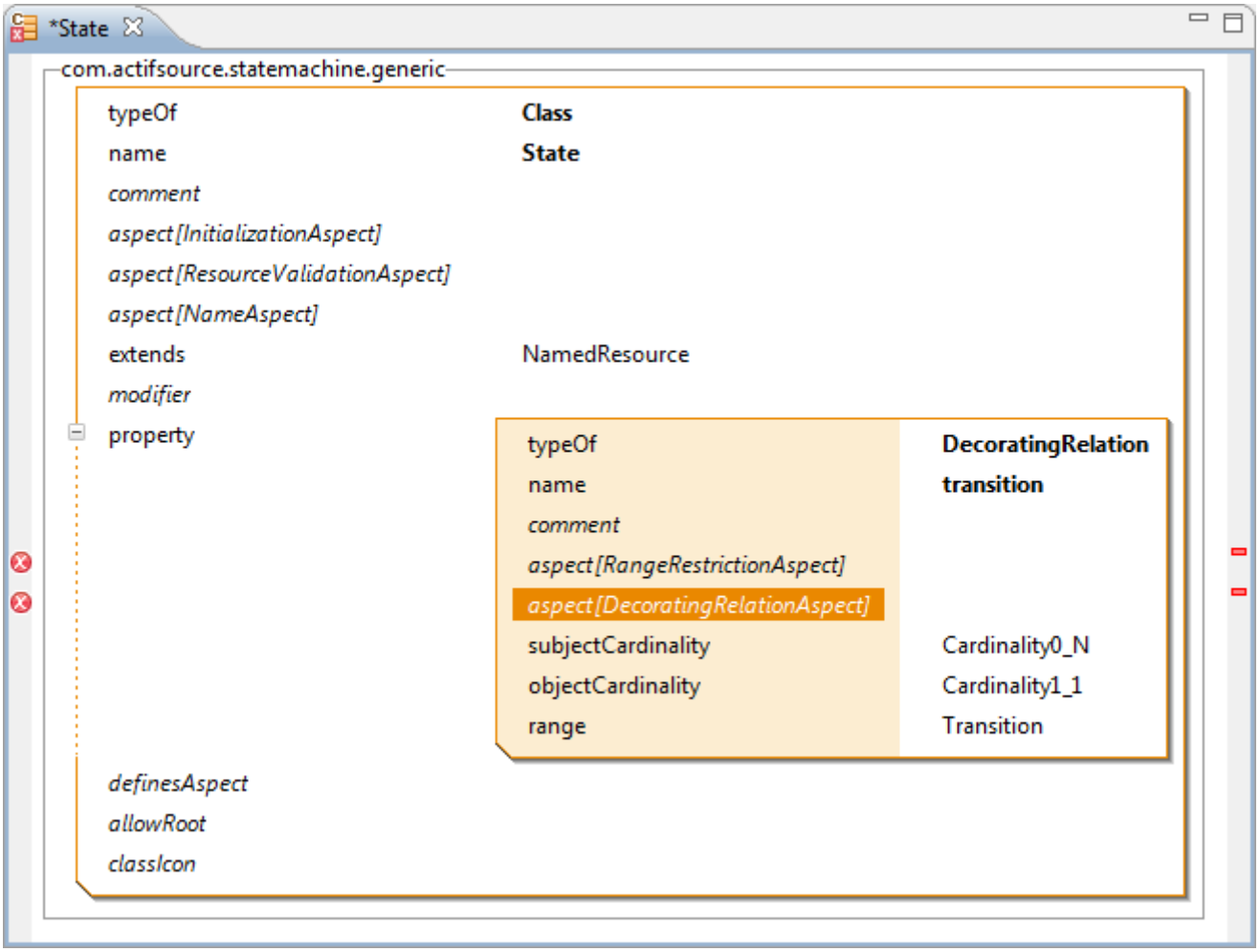


- Add the Events start and stop
- Add the States Initialized, Started and Stopped as shown above
- ⓘ At this stage, the generic model doesn't prevent you from mixing instances from different Statemachines when creating specific States, Events, and Transitions

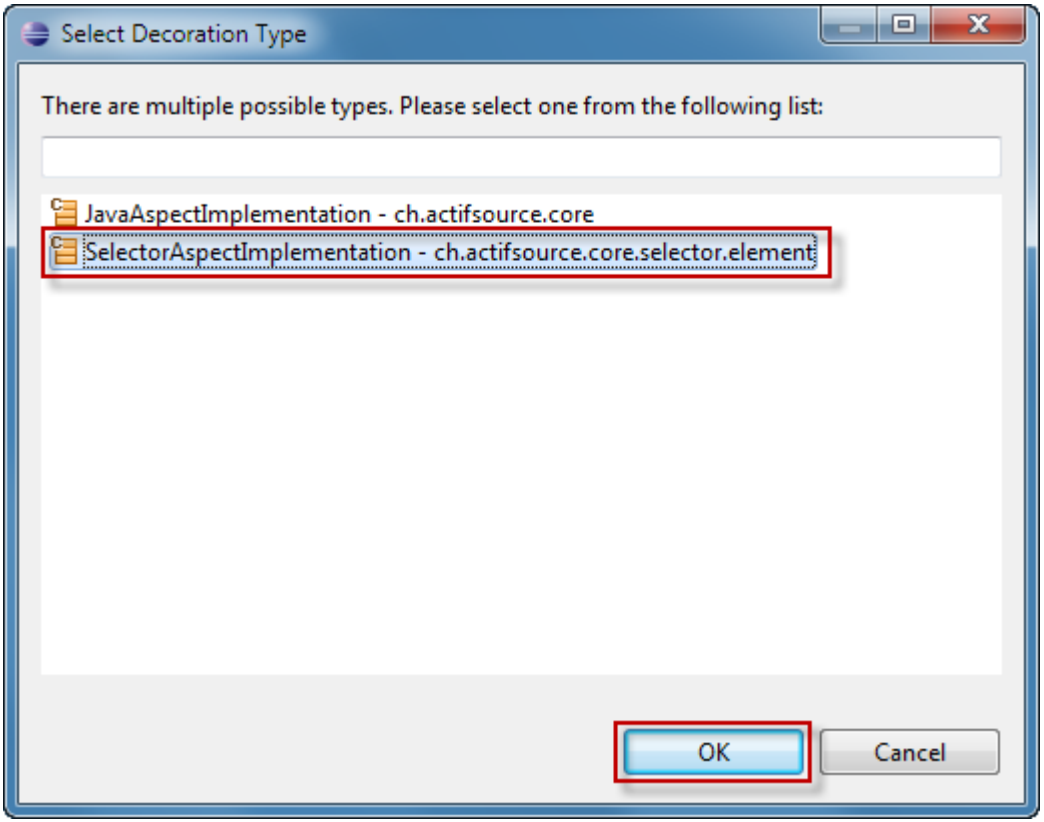
Part III: Decorating Relation Aspect

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

Add a Decorating Relation Aspect 12



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

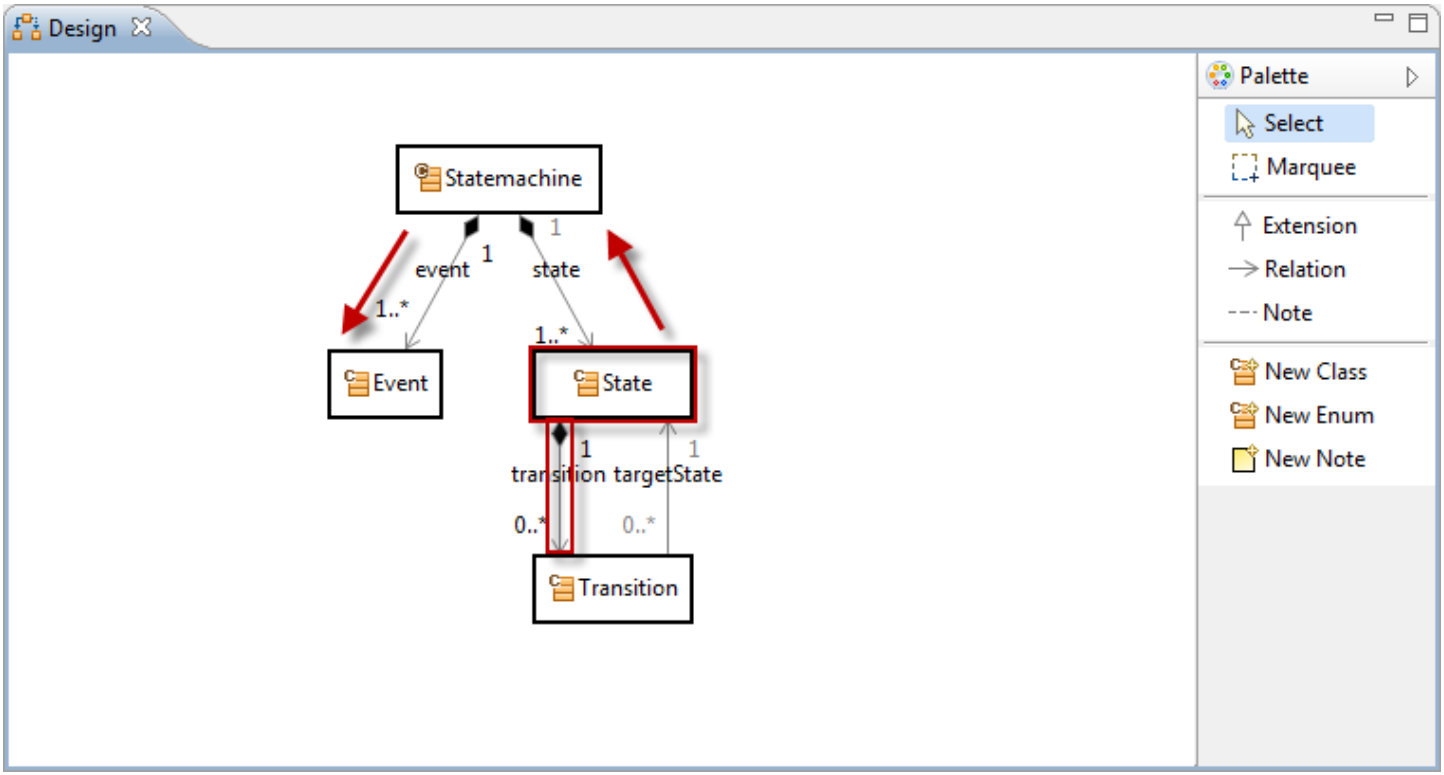


ⓘ Note that you can choose between a *JavaAspectImplementation* and a *SelectorAspectImplementation*
- Selecting the *JavaAspectImplementation* allows you to write Java Code for complex operations
- Selecting the *SelectorAspectImplementation* allows you to use the easy Selector syntax

↳ Select *SelectorAspectImplementation*

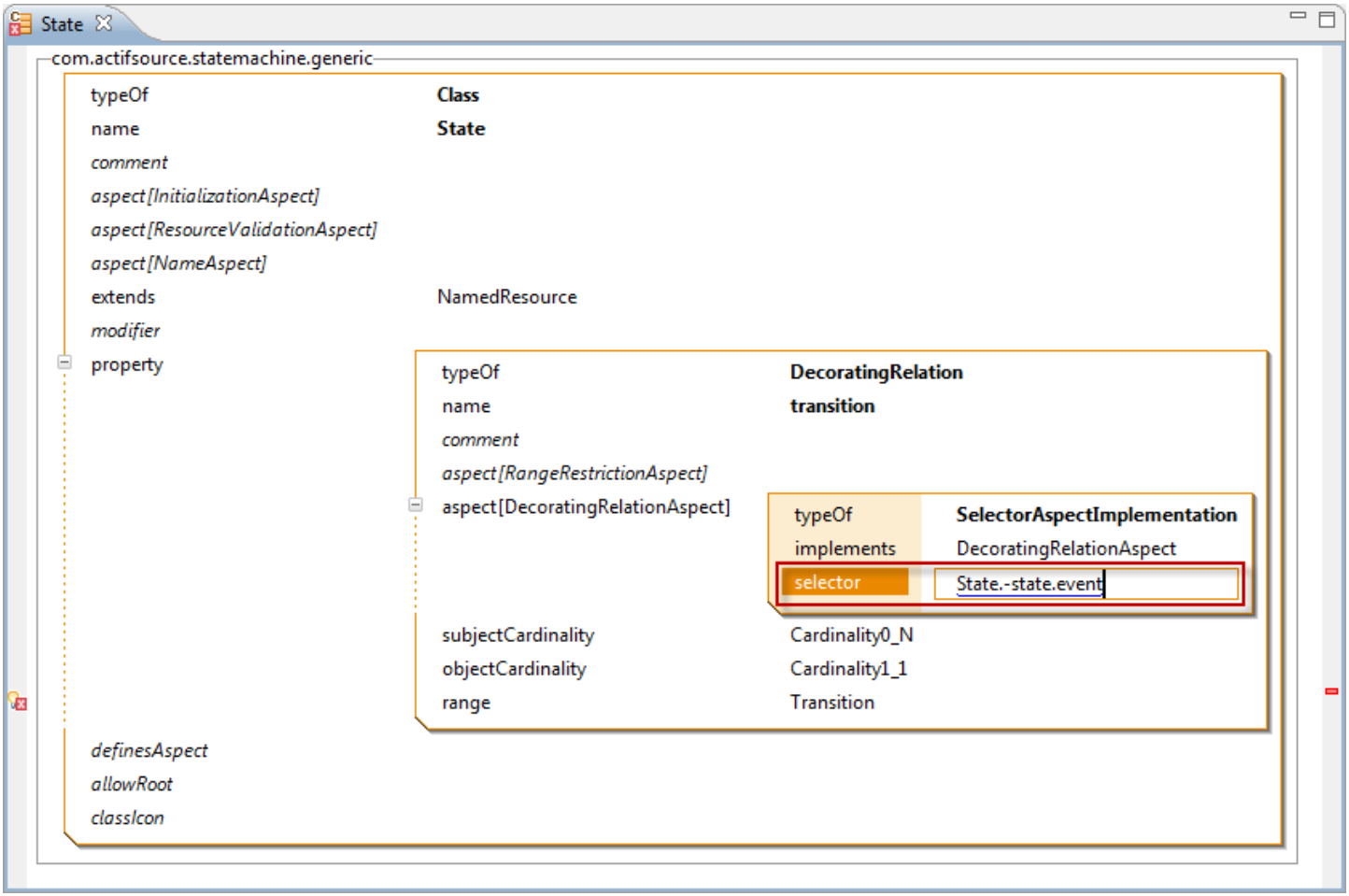
↳ Click *OK*

Add a Decorating Relation Aspect 14



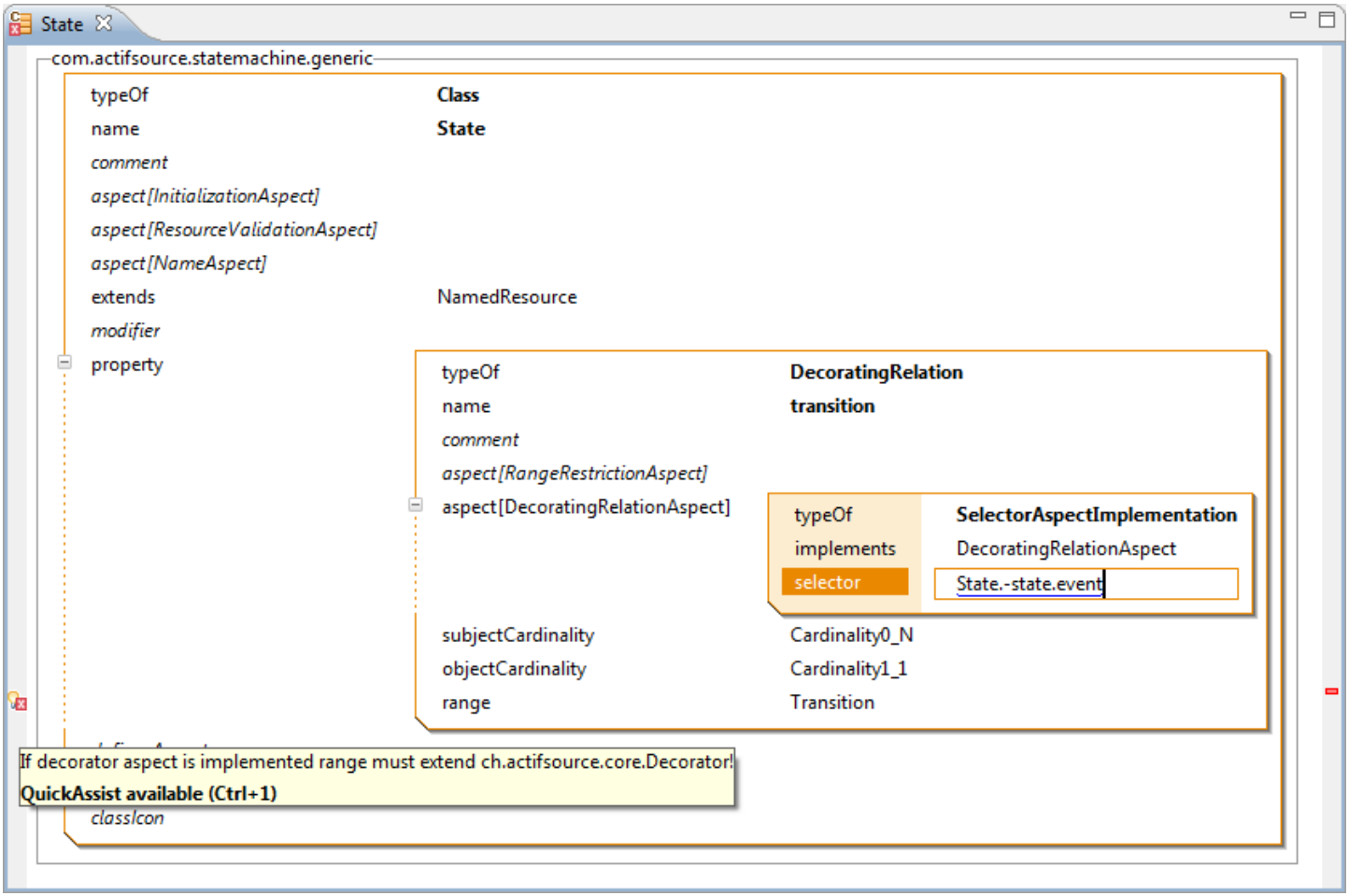
- Let's look at a possible Transition for every Event
- The **DecoratingRelation** transition is found in State
- We have to navigate from State to Event
  - Navigate backwards from State via state to Statemachine
  - Navigate forward from Statemachine via event to Event

Add a Decorating Relation Aspect



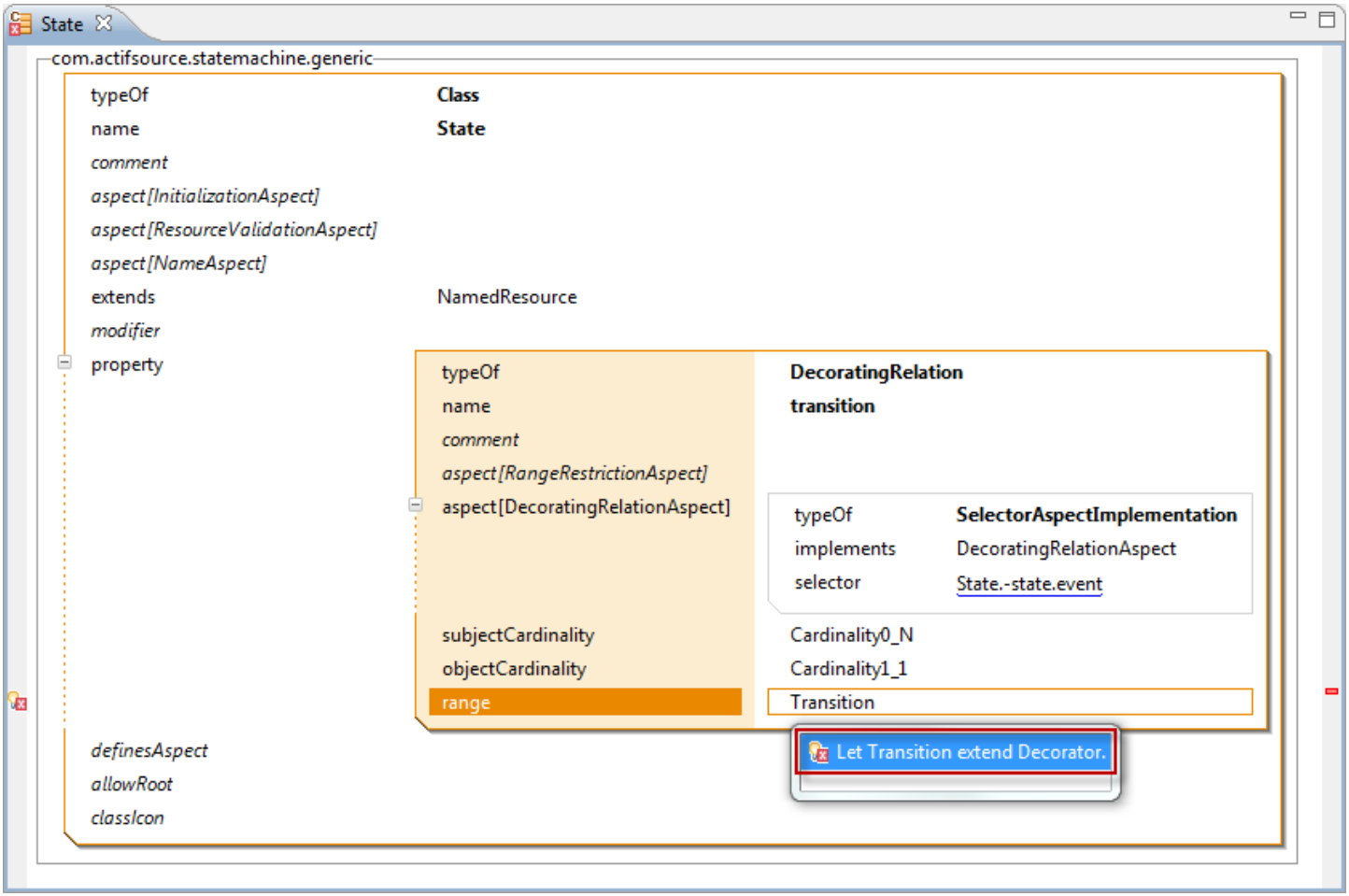
- Enter the Selector `State.-state.event` using Content Assist (Ctrl+Space)
- Note that State.–state navigates backwards from State to Statemachine

Add a Decorating Relation Aspect 16

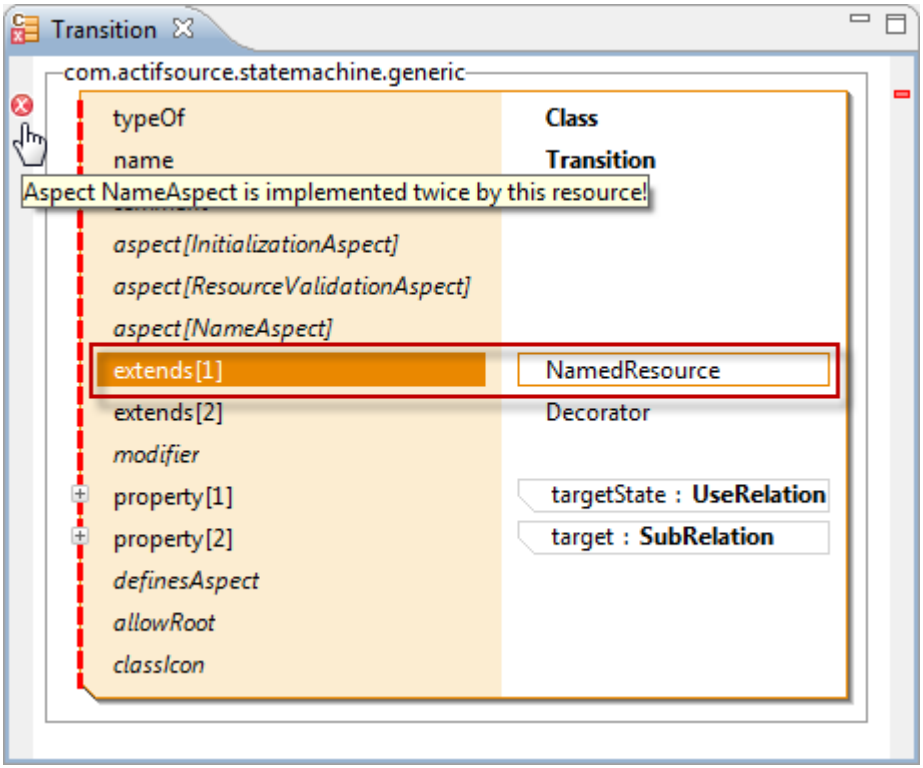


- ⓘ Implementing a *DecoratingRelationAspect* asks for a subclass of Decorator
- ⓘ Decorator has a **useRelation target** which is used to store the specific decorating Resource
  - Shown as: decoratingRelation[target]
- ↳ Open **Quick Assist** by clicking the light bulb or press Ctrl+1

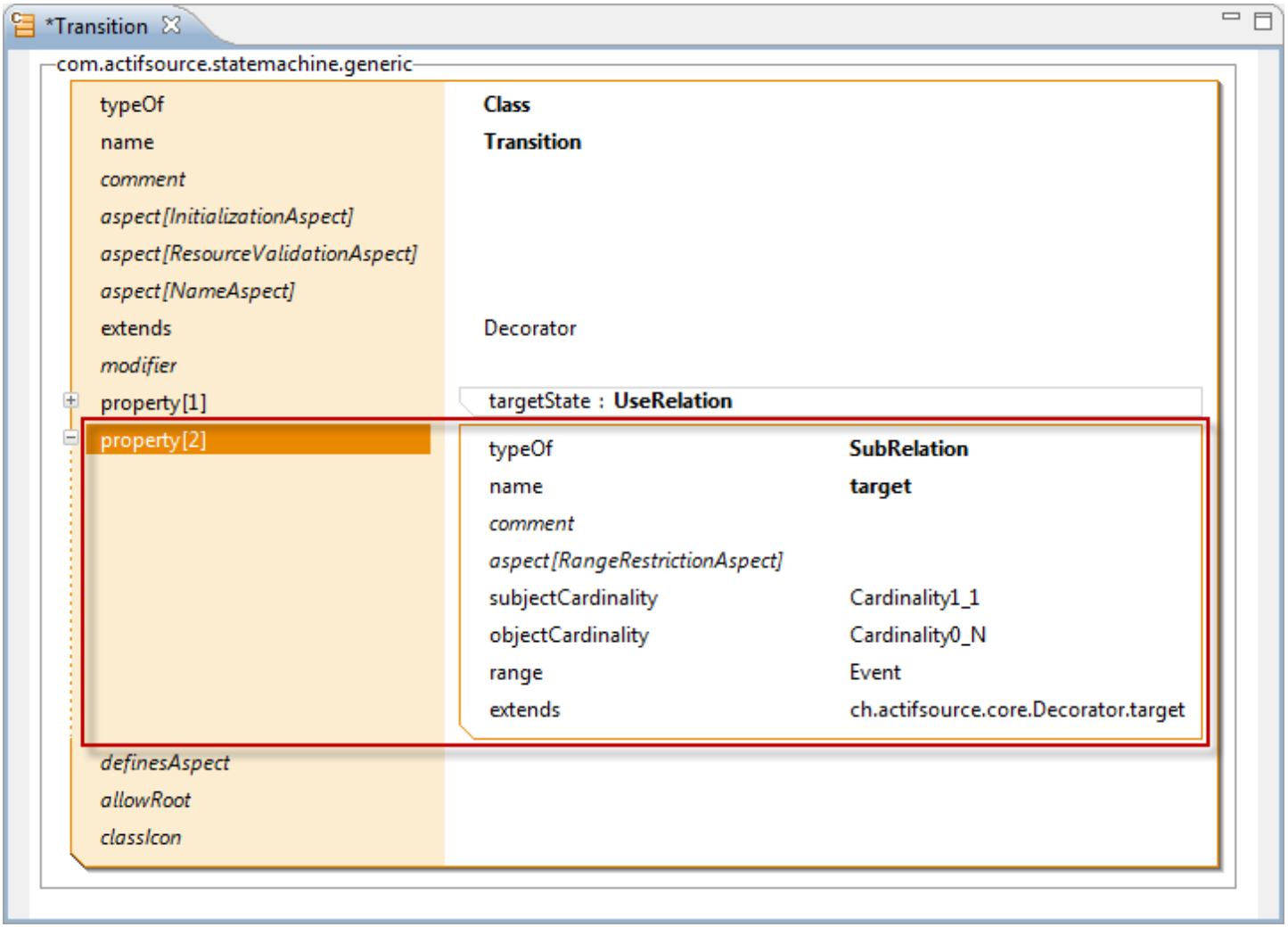
Add a Decorating Relation Aspect 17



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

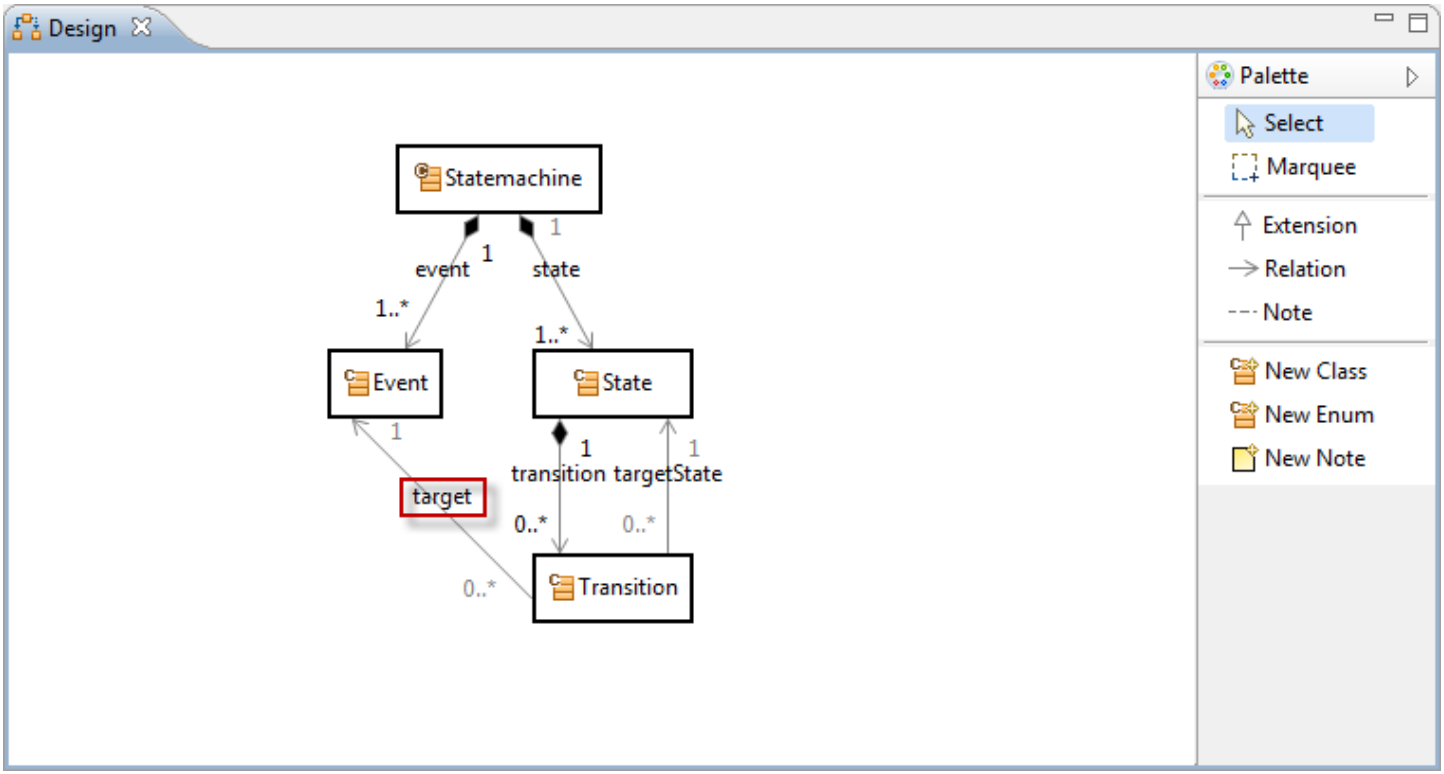


- Open Transition
- ⓘ By default a **Class** extends **NamedResource**
- Remove **extends NamedResource** as **Decorator** already implements a name aspect



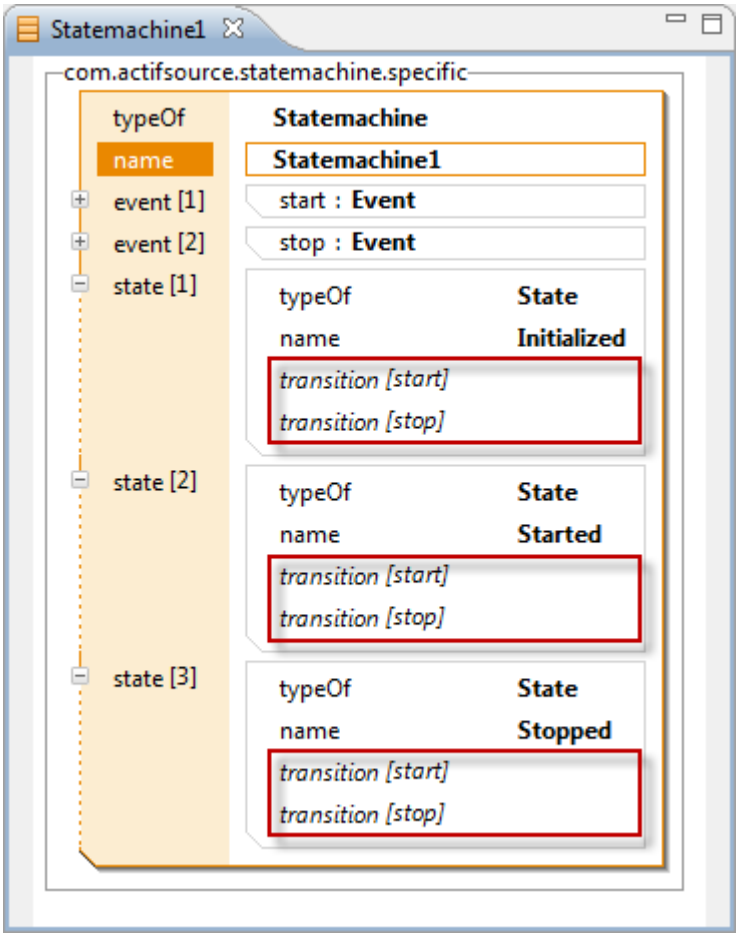
- ⓘ Quick Assist has done the following
  - Added **extend Decorator**
  - Added **SubRelation** target
- ⓘ The **range** of Decorator.target is **Resource** and therefore untyped in the context of your domain
- ⓘ The new **SubRelation** target extends **Decorator.target** but with Event as its **range**
- ⓘ When writing template code, you are able to access Transition.target typed as Event

Add a Decorating Relation Aspect 20

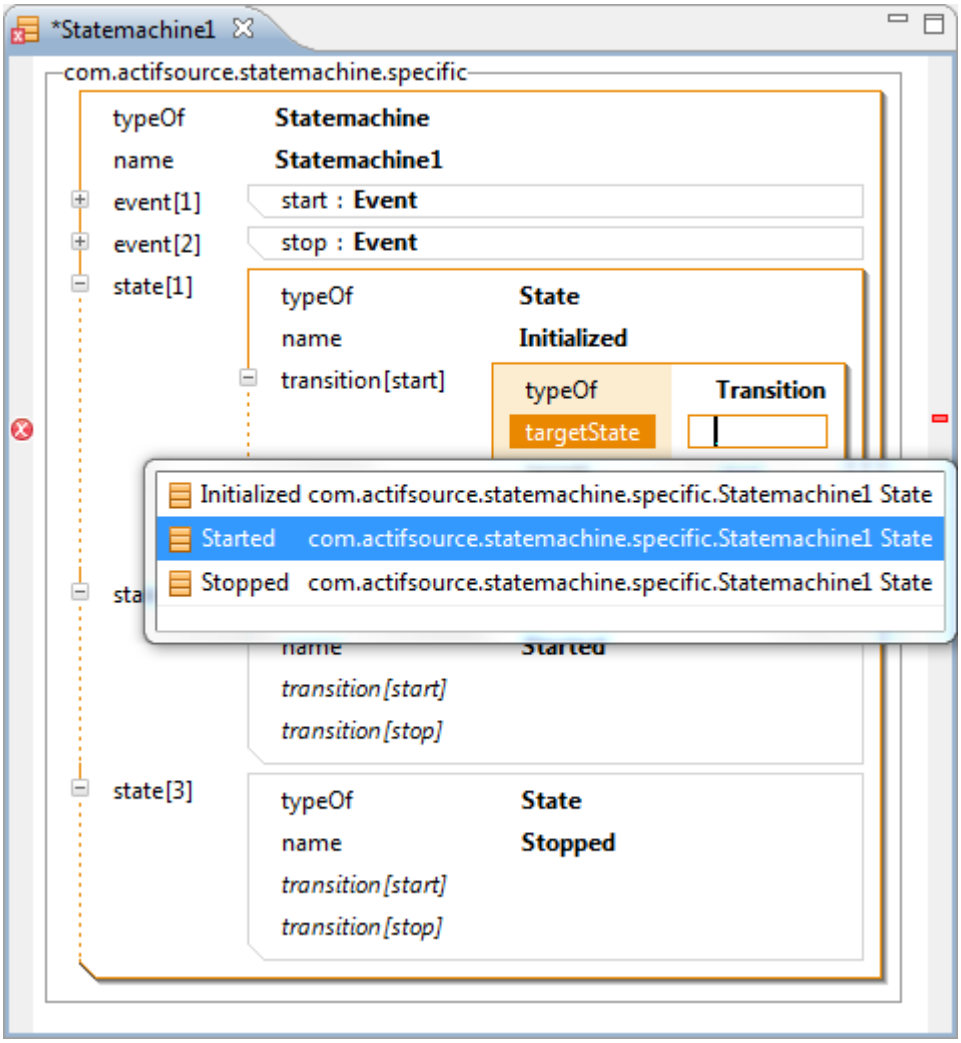


ⓘ Note that the **SubRelation** target has been added in the *Design* **Diagram** automatically

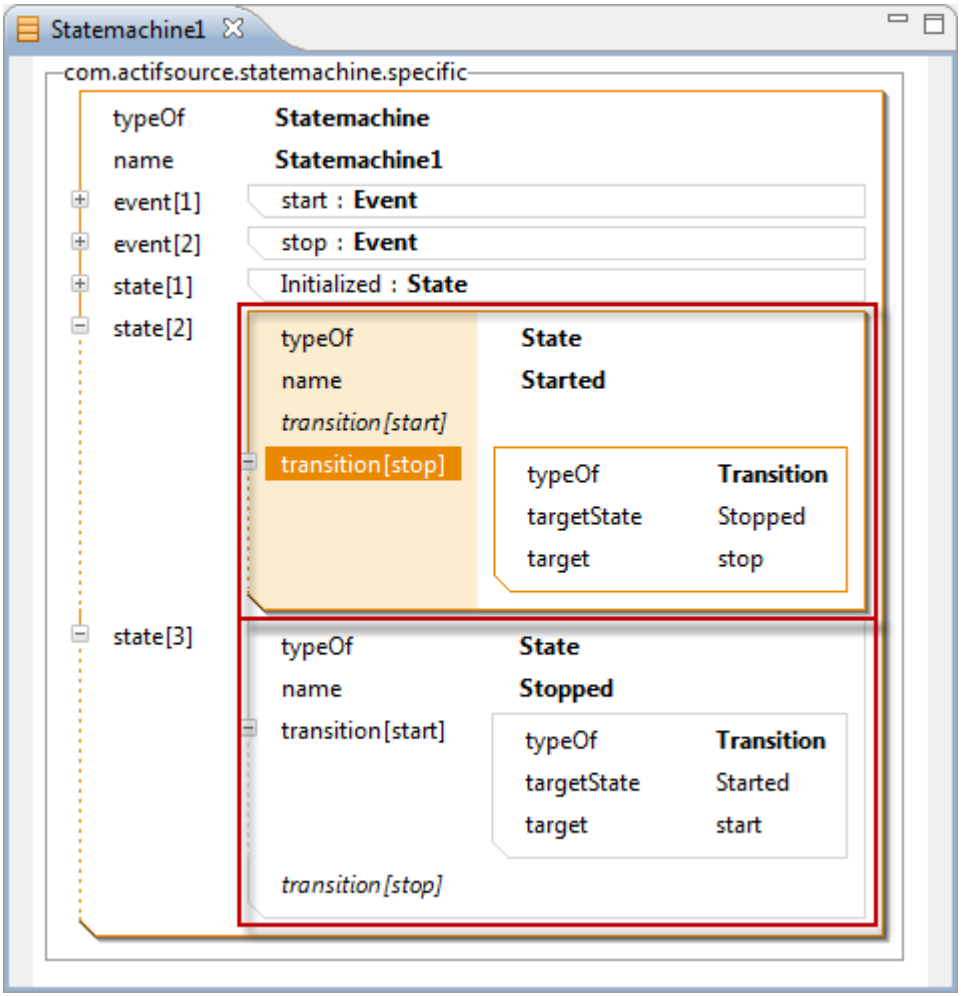
ⓘ Warning: The layout for the realtions transition and targetState might differ in your editor



- Open the specific Statemachine Statemachine1
- Note there is a **decoratingRelation** transition for every Event
- Add new Events and observe the **decoratingRelation** transition



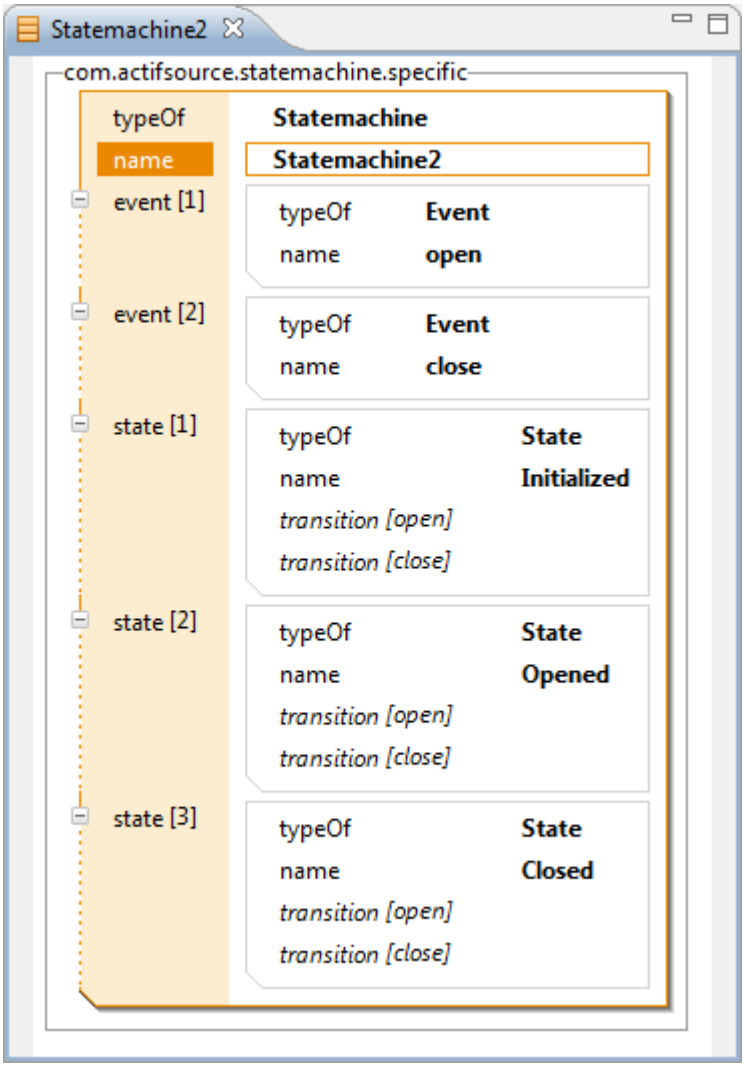
- In the State Initialized create a new Transition for transition[start]
- Select Started as targetState
- ⓘ Note that the relation target has been completed automatically with the specific decorating Event start



↳ Configure the State instances Started and Stopped as shown above

Part IV: 24
Range Restriction Aspect

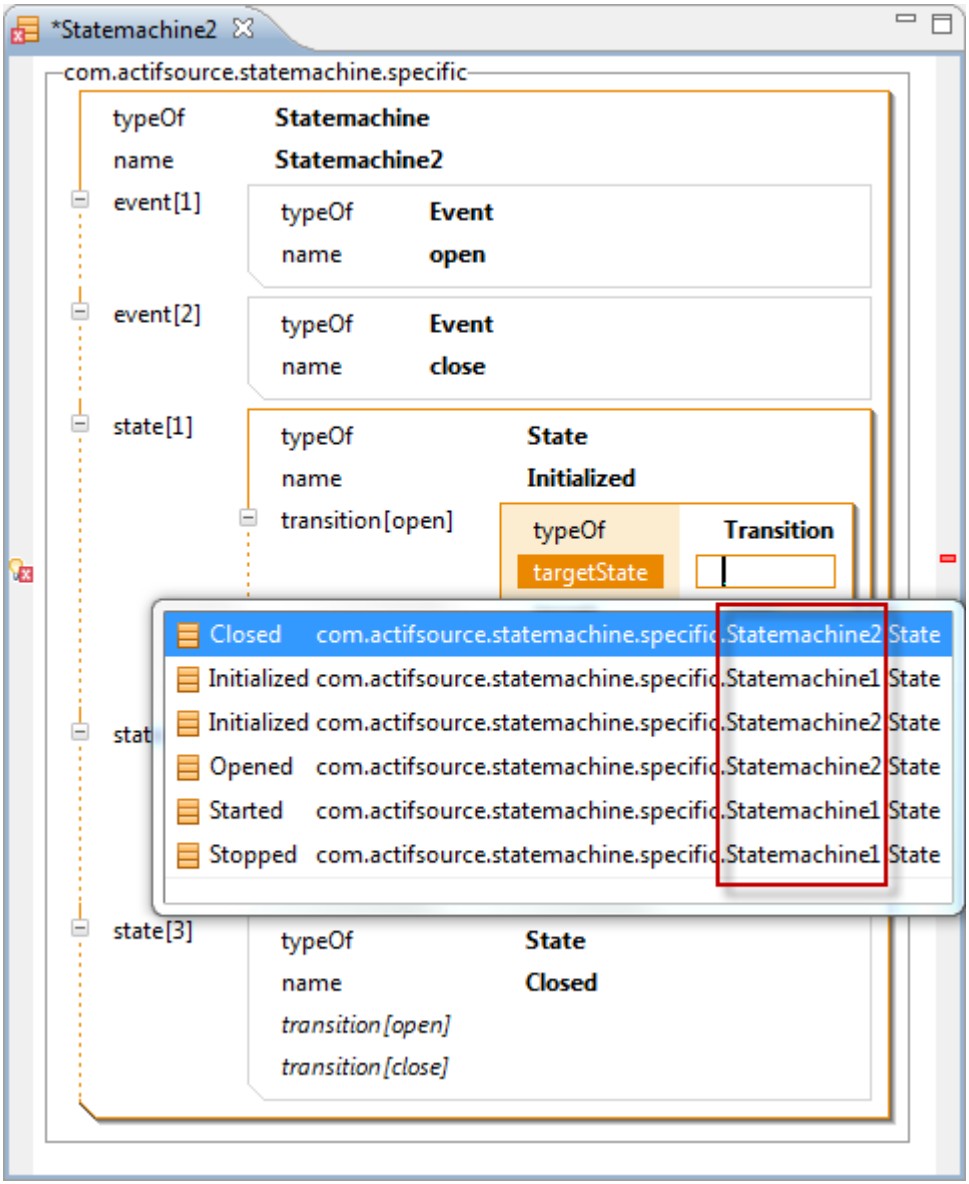
- ⓘ Content Assist (Ctrl+Sapce) in actifsource shows all instances of a desired type; It is often useful to restrict this selection
- ⓘ Learn how to apply range restrictions to filter instances for a given type



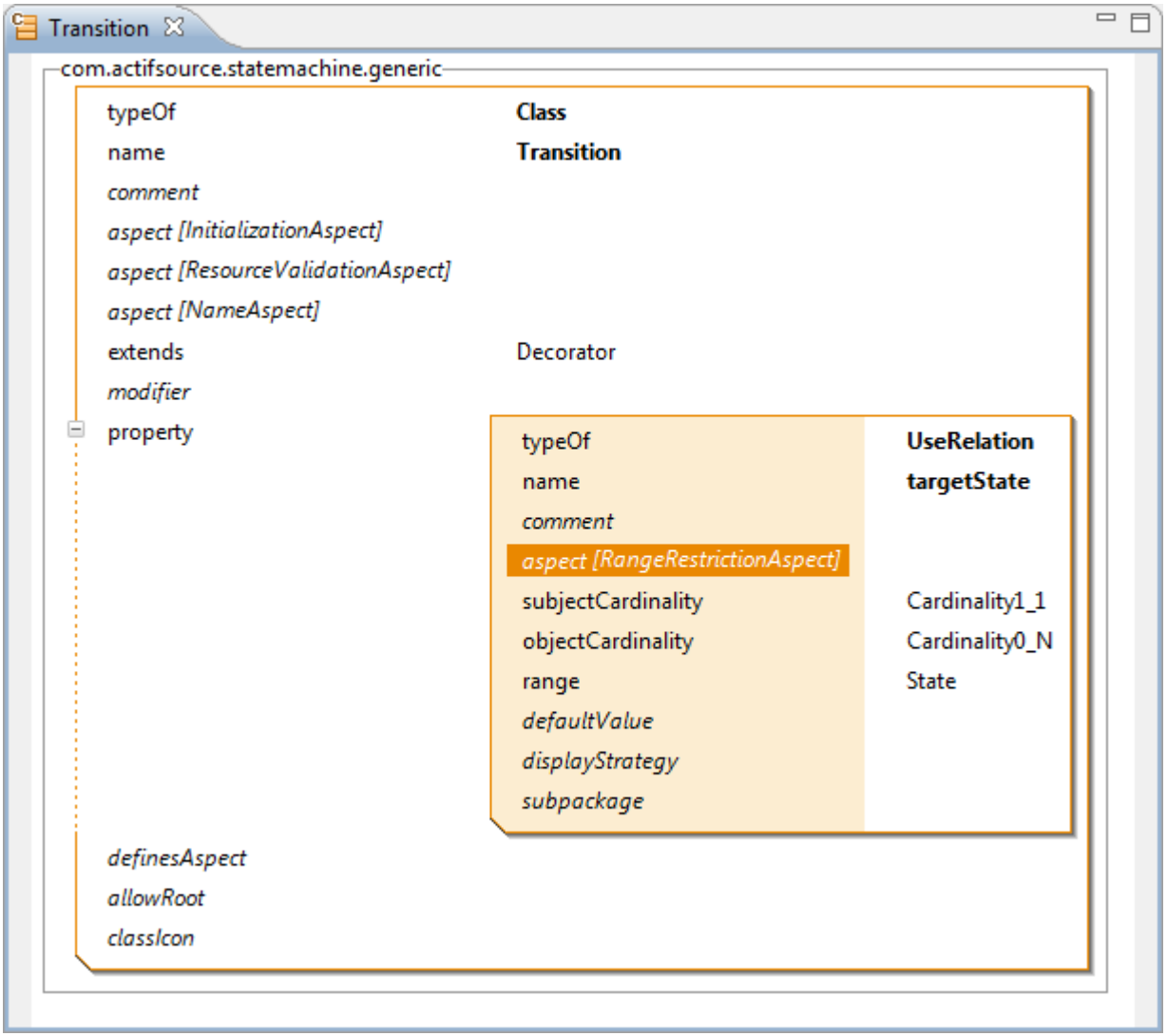
ⓘ Let's discover the needs for a range restriction aspect

- Create a Statemachine named Statemachine2 in the **Package** *specific*
- Add the Event instances open and close
- Add the States instances Initialize, Opened and Closed

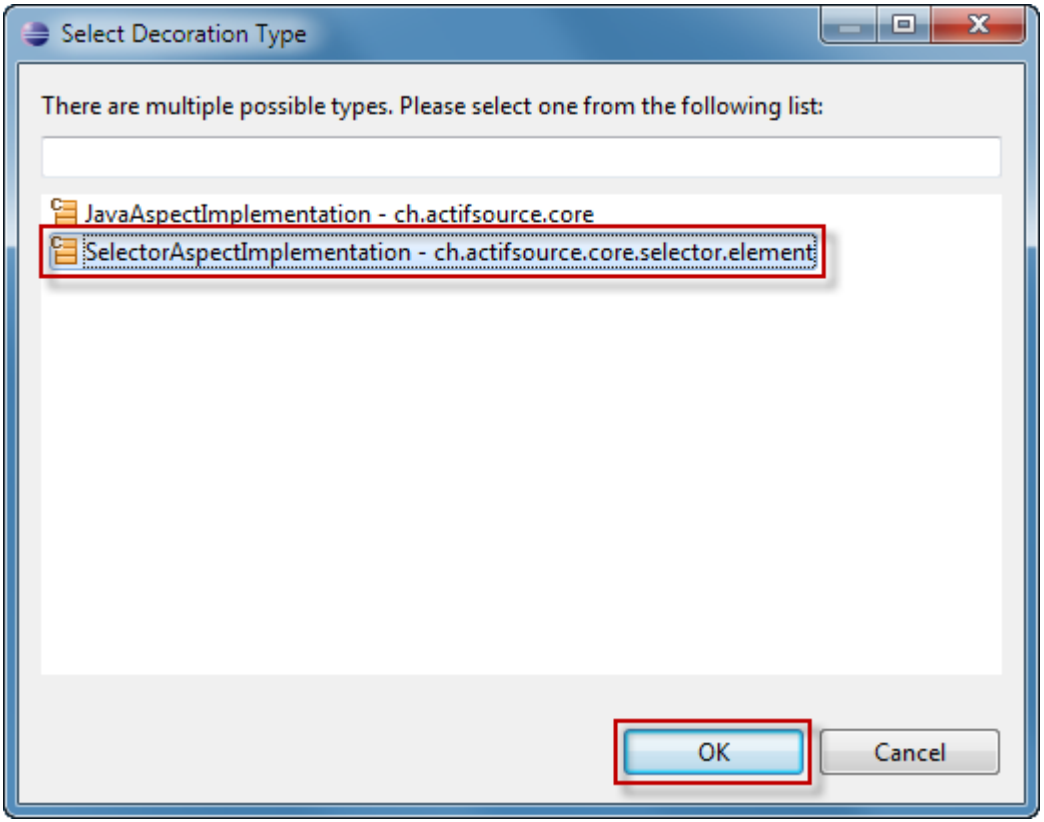
Without Range Restriction



- Create any new Transition
- Use **Content Assist** (Ctrl+Space) to add a targetState of type State
- Note that all instances of State are listened instead of just the ones from Statemachine2



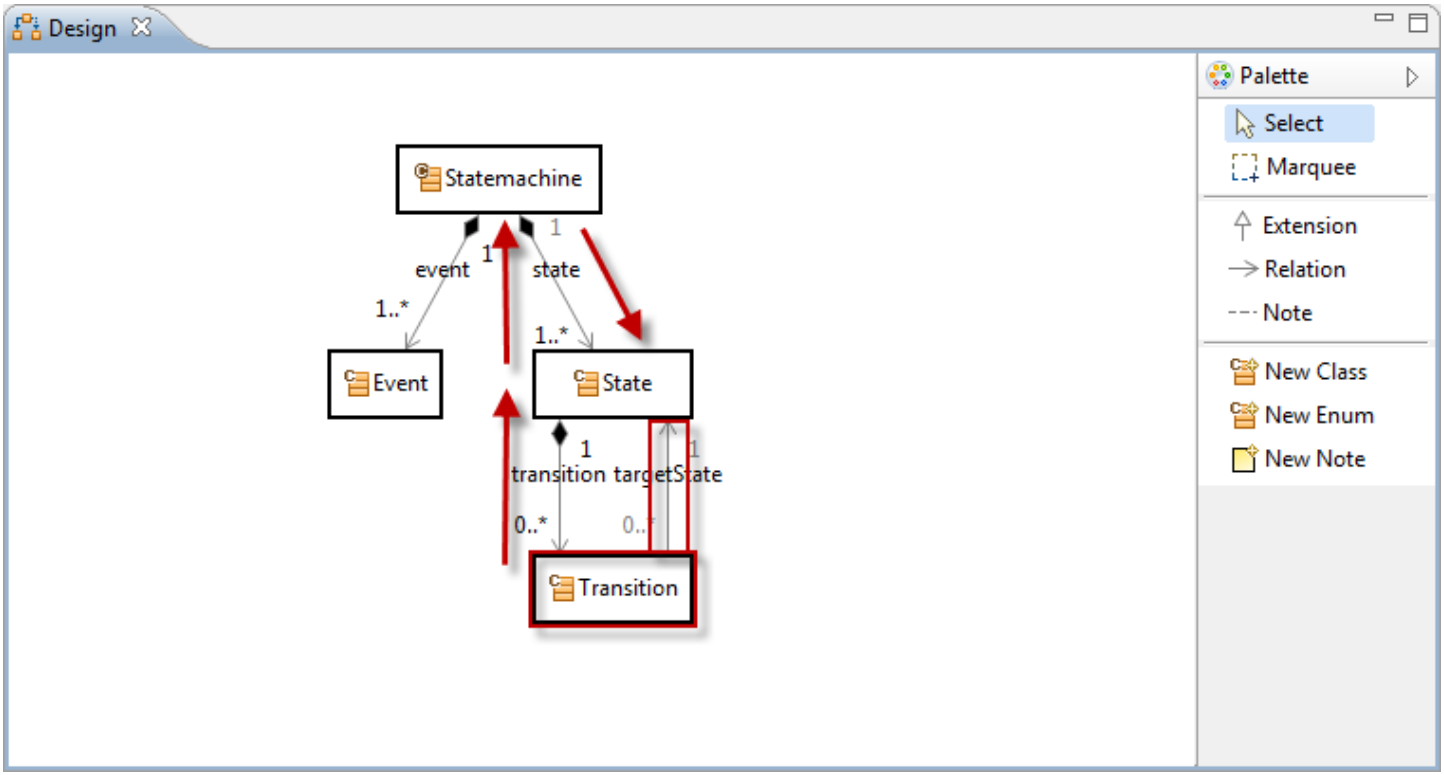
- In Transition open the **useRelation** targetState
- Press Enter on *aspect[RangeRestrictionAspect]*



ⓘ 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 *SelectorAspectImplementation* allows you to use the easy Selector syntax

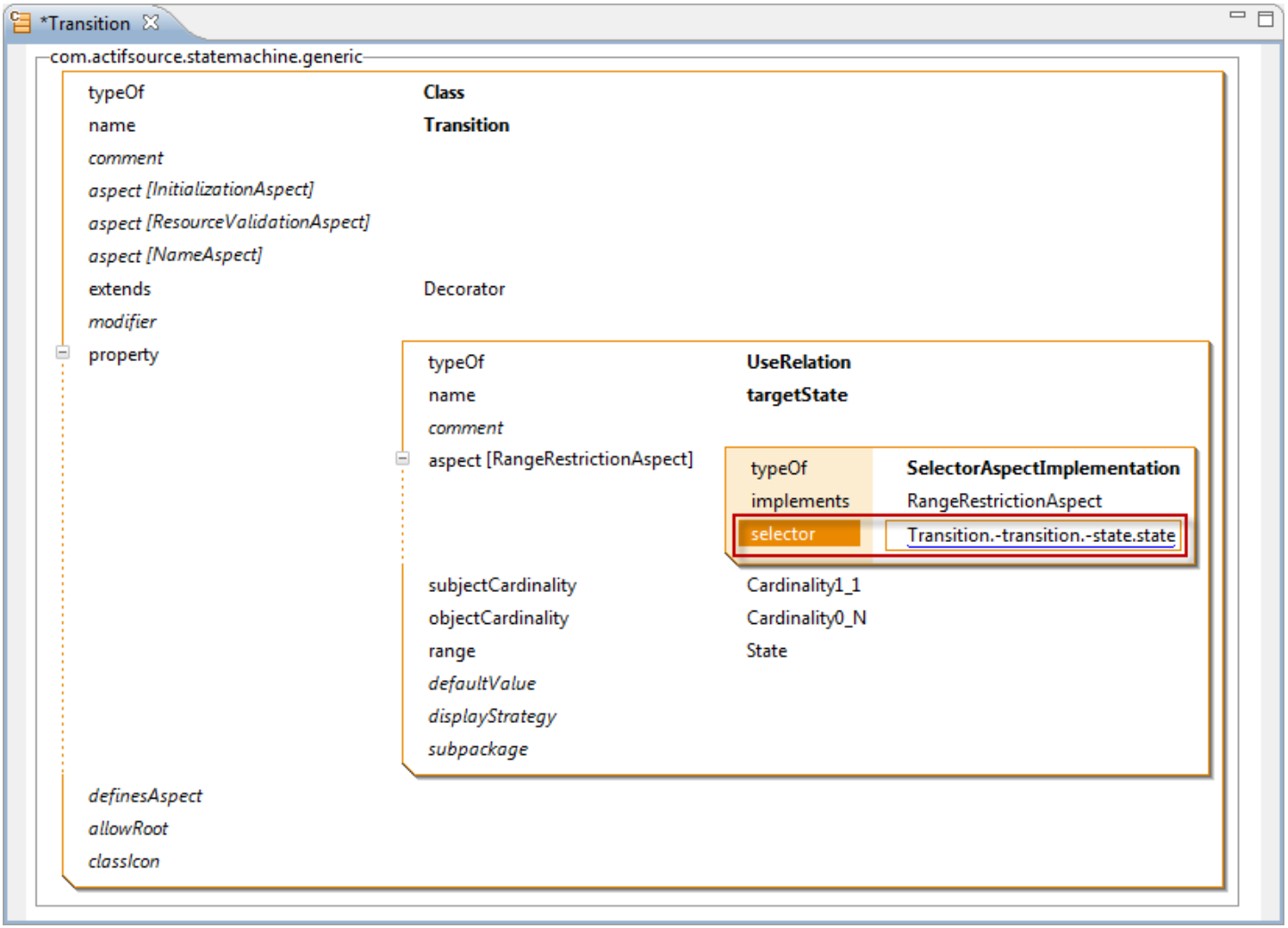
↳ Select *SelectorAspectImplementation*

↳ Click *OK*

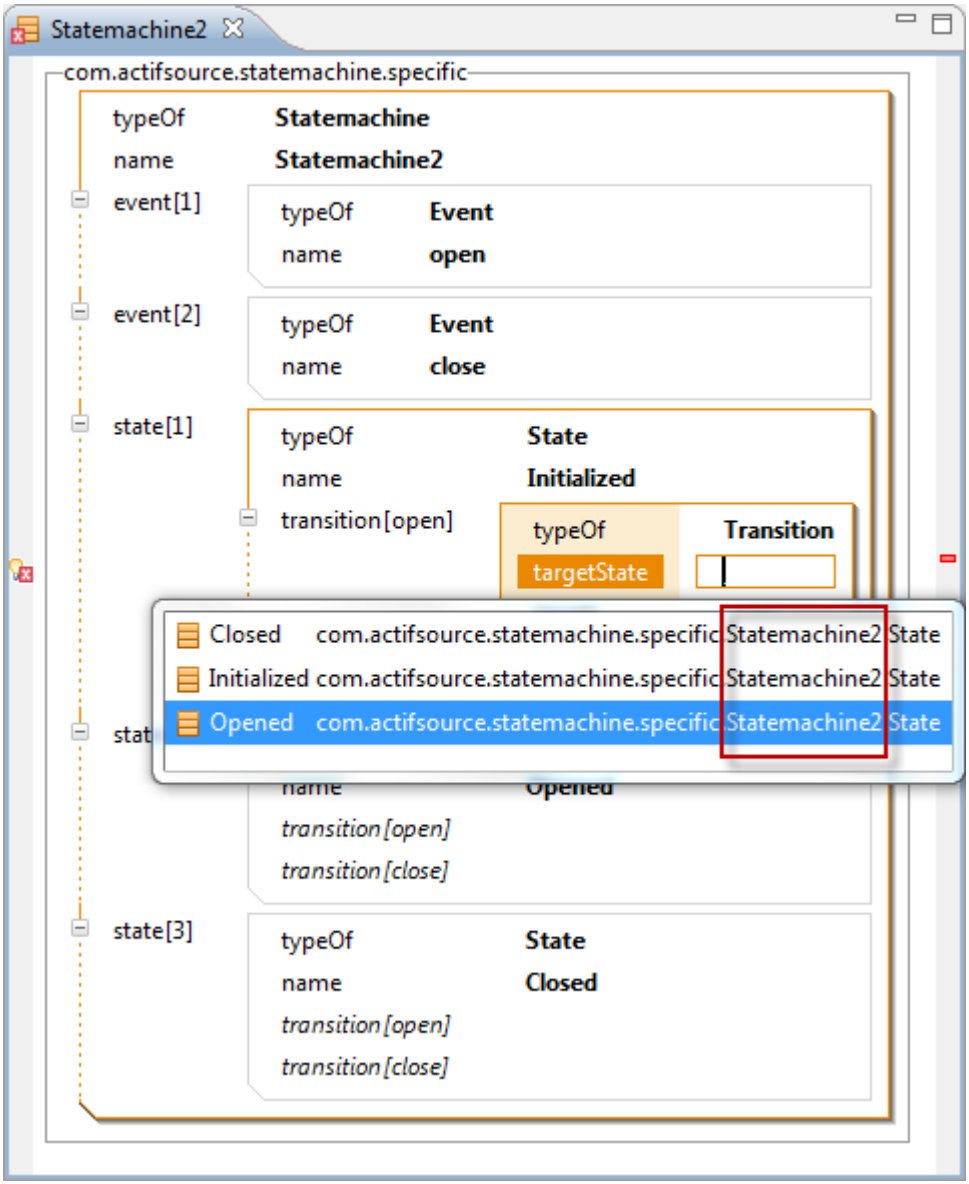


- Let's restrict the **range** of targetState to instances of States owned by the own Statemachine
- The **useRelation** targetState is found in Transition
- We have to navigate from Transition to all States of the Statemachine
  - Navigate backwards from Transition via transition to State
  - Navigate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

Add a Range Restriction Aspect

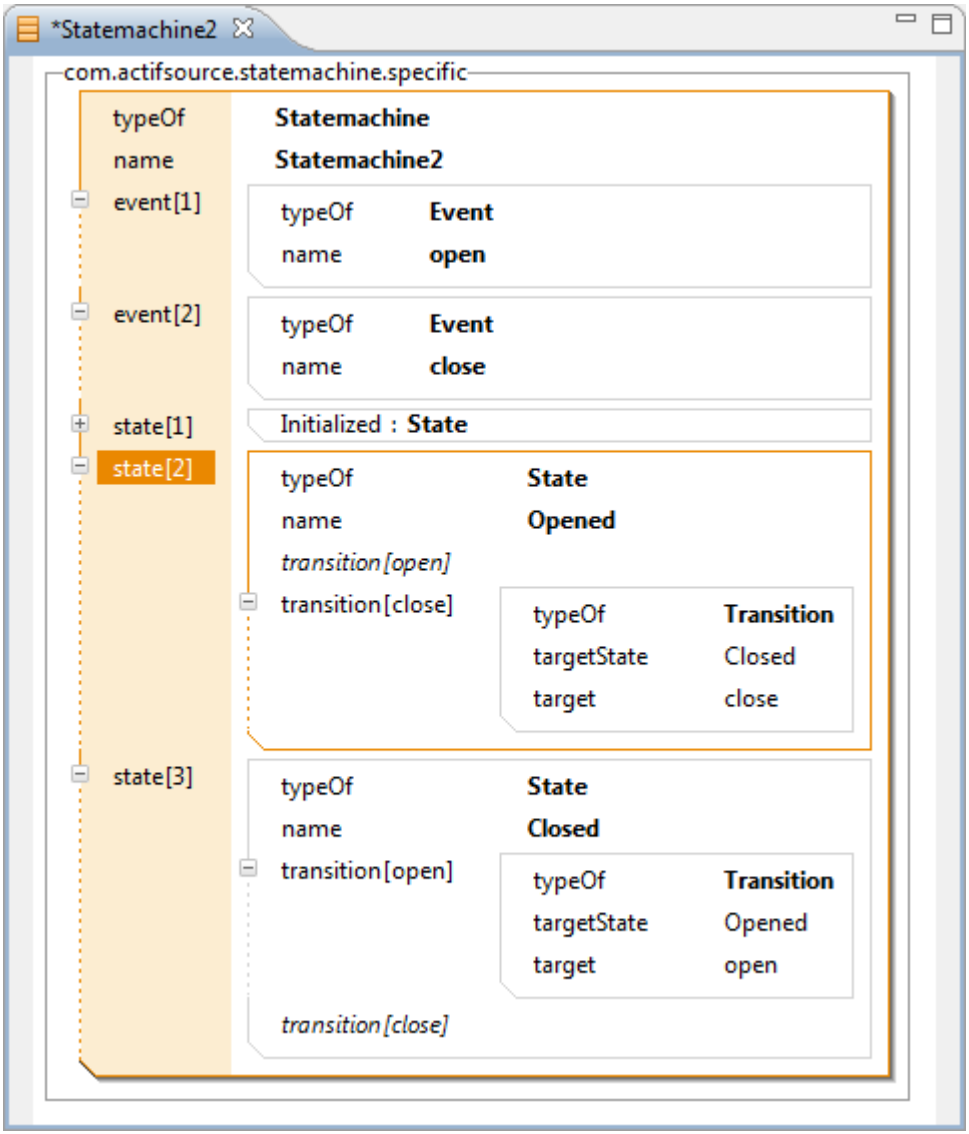


↳ Enter the Selector `Transition.-transition.-state.state` using Content Assist (Ctrl+Space)



↳ Use **Content Assist** (Ctrl+Space) again to add the targetState Opened of type State

ⓘ Note that only instances of State from Statemachine2 are listed



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