



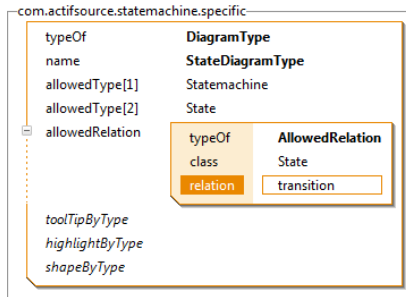


Tutorial

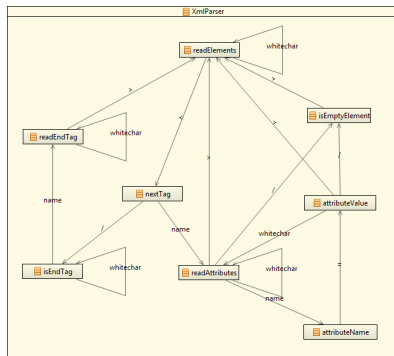
Domain Diagram Type

Tutorial	Actifsource Tutorial – Domain Diagram Type
Required Time	<ul style="list-style-type: none"> • 120 Minutes
Prerequisites	<ul style="list-style-type: none"> • Actifsource Tutorial – Installing Actifsource • Actifsource Tutorial – Simple Service • Actifsource Tutorial – Statemachine
Goal	<ul style="list-style-type: none"> • Create a Domain Diagram • Customize view using Diagram Type • Set highlighting paths • Set custom tooltips
Topics covered	<ul style="list-style-type: none"> • Create a Diagram Type • Use Domain Diagram • Set Relation mode • Adjust shapes and colors for DomainDiagram • Set highlighting path • Write ToolTipAspect using Low-Level API
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 • <code>Monospaced</code>: 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>

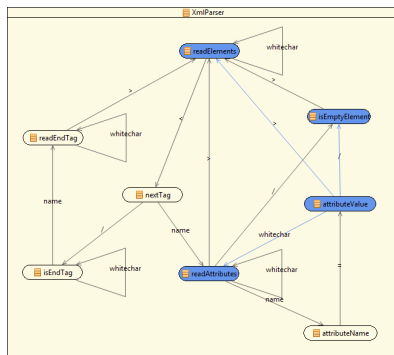
- Create a Diagram Type



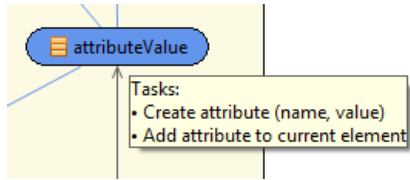
- Create a Domain Diagram



- Custom shapes and Interaction

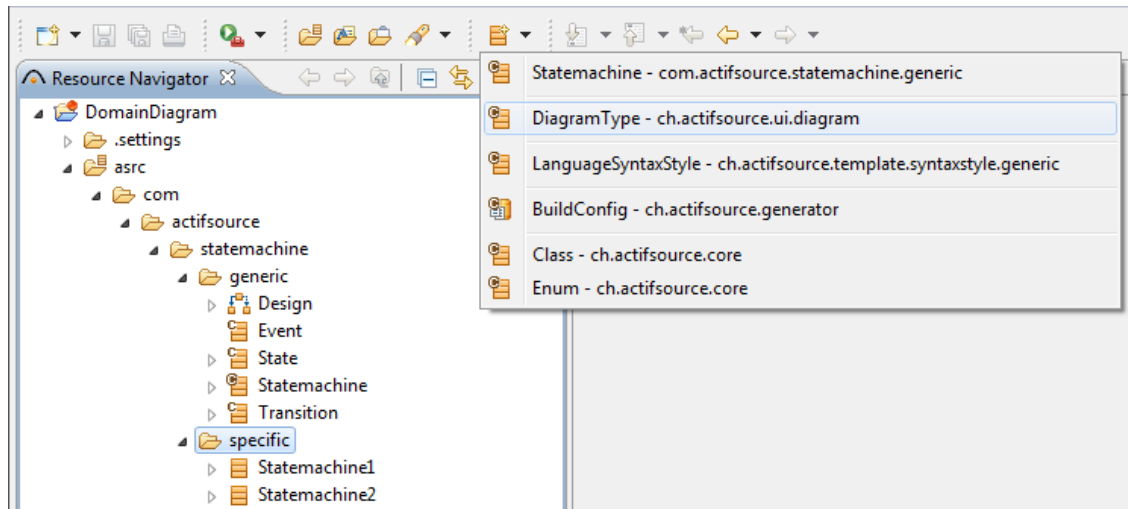


- Tooltips

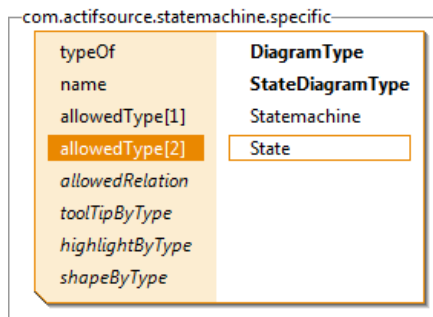


Create a Diagram Type

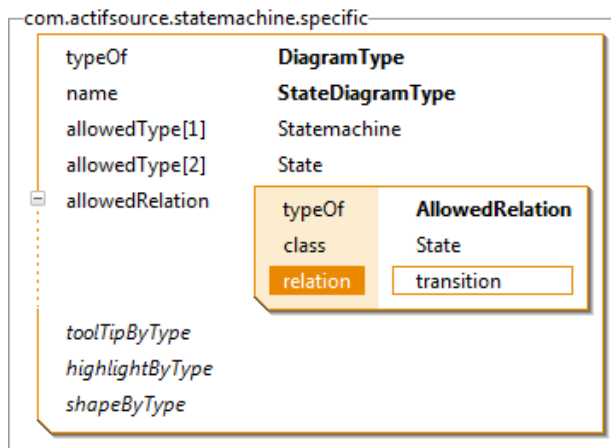
- ① For users of of the *Enterprise Edition* and the *CIP Edition* of *Actifsource*, there is a graphical editor for your domain model: the Domain Diagram editor.
- ① Domain Diagrams can be customized by Diagram Types.
- ① In this tutorial we will extend the Project *Statemachine* we have created in the *Actifsource Tutorial – State Machine*.
- ① Before we create a new Domain Diagram, let us specify, what to display:
- ↪ Create a new **Diagram Type** named StateDiagramType in the package *com.actifsource.statemachine.specific*.



- ① Using a **DiagramType** allows us to restrict the set of Resources and Relations displayed in a Domain Diagram.
 - If **allowedType** is set to some types, the set of **Classes** and other Resources being displayed is restricted to the given types. If **allowedType** is not set or there is no **DiagramType** referenced from the **Domain Diagram**, then there is no such restriction.
 - If **allowedRelation** contains any **Relations**, only the given **Relations** are displayed in the **Domain Diagram**. If **allowedRelation** is not set, then no restriction applies.
- ↪ For our diagram, set **allowedType** to **Statemachine** and **State**.
- ① Otherwise also the **Events** would appear in the diagram.



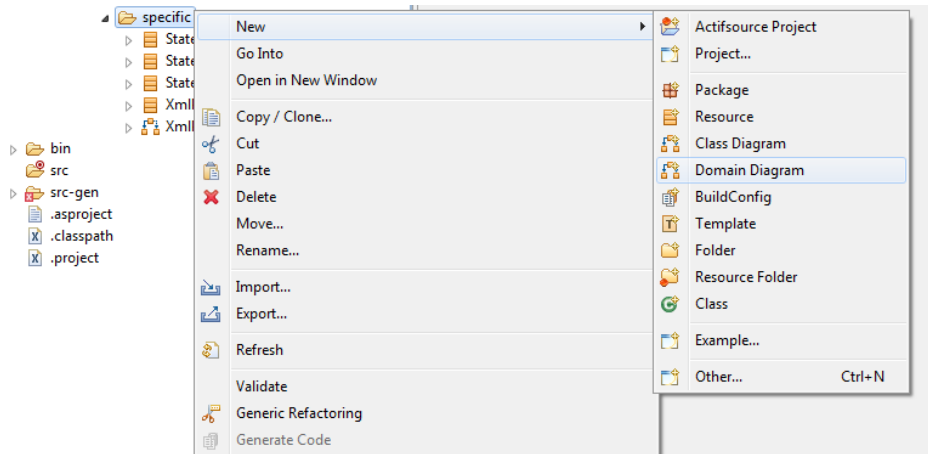
- ↪ Set **allowedRelation** to **State.transition**.



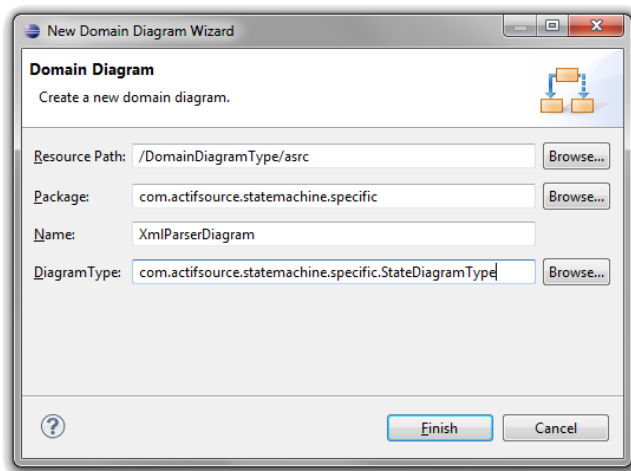
Create a Domain Diagram

Let us now create the diagram itself.

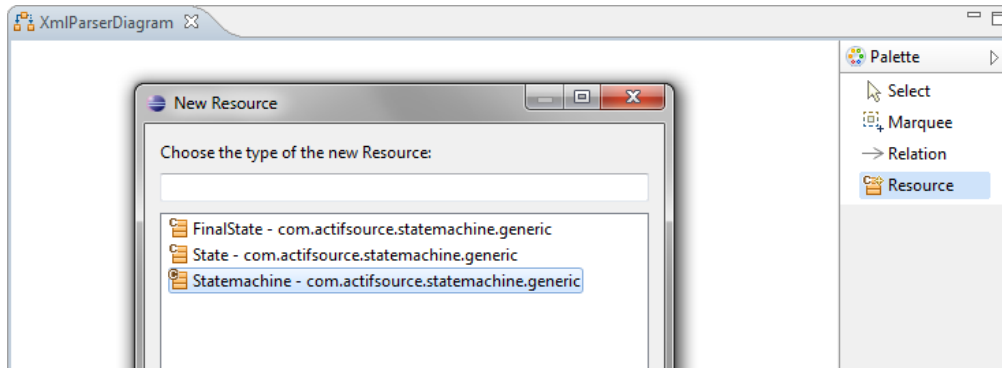
↪ Create a Domain Diagram



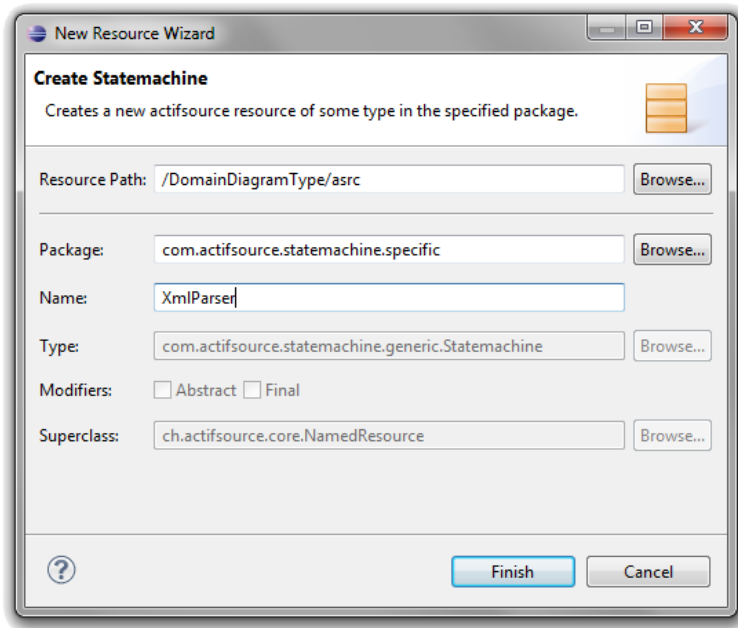
↪ Choose StateDiagramType as *Diagram Type*.



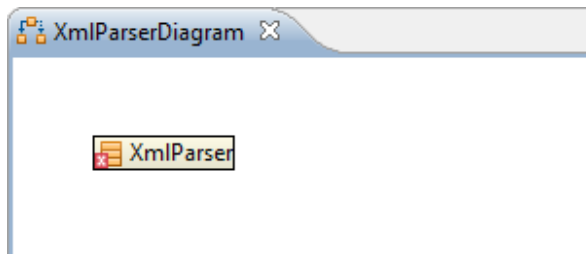
- ① As an example, let us now create a new Statemachine named XmlParser.
- ↶ Select the *Resource* tool in the palette.
- ↶ Click into the diagram area, where you want to position the state machine object.
- ↶ Select Statemachine.



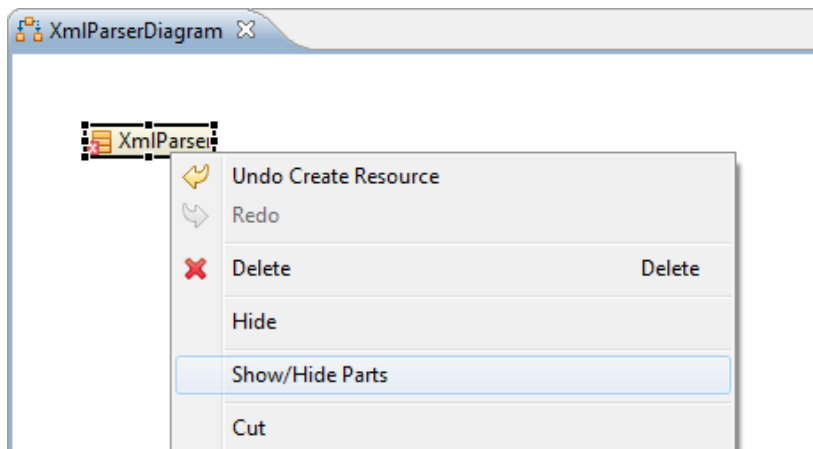
- ↶ In the wizard, enter `XmlParser` as *Name*.



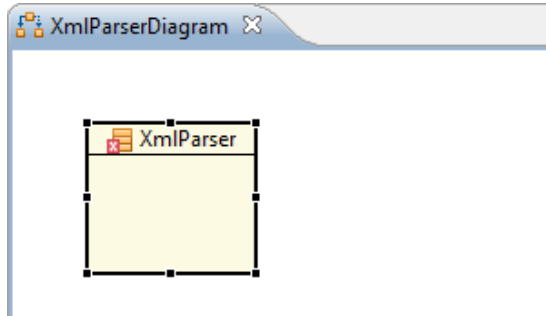
① The created Statemachine initially looks like this:



↶ Unhide the contained **Resources**.

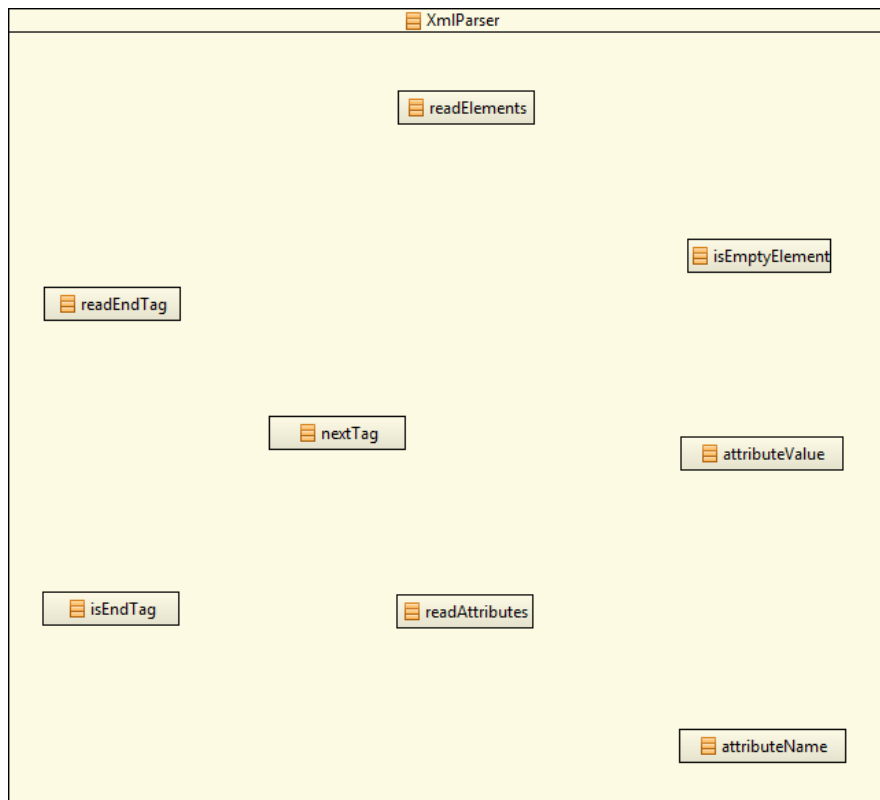


- ↩️ Resize the shape so you have enough space to create new States in the XmlParser. The name is displayed in a title bar.



- ↩️ Ctrl + Click onto XmlParser to open it in the Resource Editor.
- ↩️ Enter the following Events (using Ctrl + Enter to insert after selection):
 - <
 - *name*
 - =
 - /
 - >
 - *whitechar*

- ↵ Switch to the Domain Diagram Editor.
- ↵ Enter the following States by clicking into the area of XmlParser.
 - *readElements*
 - *nextTag*
 - *isEndTag*
 - *readEndTag*
 - *readAttributes*
 - *attributeName*
 - *attributeValue*
 - *isEmptyElement*
- ① The resulting Statemachine diagram will look like this:



- ① As we have seen, owned Resources can simply be placed into its owner.
- ① The mode, how **OwnRelation** are interpreted in the Domain Diagram can be changed, as well as for **UseRelations** by setting the relation's **relationMode** property.
 - *functional*: The Relation is not displayed at all.
 - *structuralContainment*: The second **Resource** is nested into the first **Resource**. Default for **OwnRelation**.
 - *structuralDirect*: An arrow from one to the other **Resource** is drawn. Default for **UseRelation**.
 - *structuralIndirect*: The given Relation is interpreted as range of a UseRelation. So it is not displayed itself, but the owner of the Relation is displayed as arrow instead.
- ① Note that the **Relation**'s **name** is only displayed, when the **relationMode** is *structuralDirect*.
- ① We would like to display the Statemachine's Transitions as arrows.
- ↪ So let's set the **relationMode** of `Transition.targetState` to *structuralIndirect*.

The screenshot displays the configuration for a **UseRelation** in a software development tool. The left pane shows the properties of the relation, with **relationMode** highlighted. The right pane shows the configuration for the **UseRelation**, including the **targetState** and the **RangeRestrictionAspect** (set to **SelectorAspectImplementation**). The **relationMode** dropdown menu is open, showing the following options:

functional	ch.actifsource.core.RelationMode	RelationMode
structuralContainment	ch.actifsource.core.RelationMode	RelationMode
structuralDirect	ch.actifsource.core.RelationMode	RelationMode
structuralIndirect	ch.actifsource.core.RelationMode	RelationMode

The **structuralIndirect** option is selected. The **target** is set to **SubRelation**.

- ① For our example, we would restrict our state machine metamodel to cover only deterministic state machines.
 - ① We can express this by setting the **subjectCardinality** of the **State.transition**.
 - ① In **DecoratingRelations** the **subjectCardinality** gives the number of **Decorator** objects per decorated object: The cardinalities are the same as those of **Decorator.target**.
- ↪ Set the subjectCardinality of **State.transition** to 0..1, so that there is for a given **State** at most 1 **Transition** per **Event**.

typeOf name comment aspect[RangeRestrictionAspect] aspect[DecoratingRelationAspect]	DecoratingRelation transition						
subjectCardinality objectCardinality relationMode style defaultValue range	<table border="1"> <tr> <td>typeOf</td> <td>SelectorAspectImplementation</td> </tr> <tr> <td>implements</td> <td>ch.actifsource.core.DecoratingRelation.DecoratingRelationAspect</td> </tr> <tr> <td>selector</td> <td><u>State.-state.event</u></td> </tr> </table> <input type="text" value="Cardinality0_1"/> Cardinality1_1 com.actifsource.statemachine.generic.Transition	typeOf	SelectorAspectImplementation	implements	ch.actifsource.core.DecoratingRelation.DecoratingRelationAspect	selector	<u>State.-state.event</u>
typeOf	SelectorAspectImplementation						
implements	ch.actifsource.core.DecoratingRelation.DecoratingRelationAspect						
selector	<u>State.-state.event</u>						

- ① Now let us enter the transitions:
- ↵ Select the *Relation* tool.
- ↵ Click on *readElements* and click on *nextTag*.
- ↵ The Resource Editor will open and show the created Transition.
- ↵ Select target and activate the ContentAssist by pressing Ctrl + Space.
- ↵ As target Event, select <.

The screenshot displays the Resource Editor interface. On the left, a vertical list of states is shown, with state[1] expanded. The expanded state[1] shows a table with columns 'typeOf' and 'State', and a row for 'readElements'. Below this, a 'transition' section is visible, containing a table with columns 'typeOf' and 'Transition', and a row for 'readEndTag'. A content assist menu is open over the 'target' field, listing several XML tags with their corresponding event classes. The '<' tag is selected.

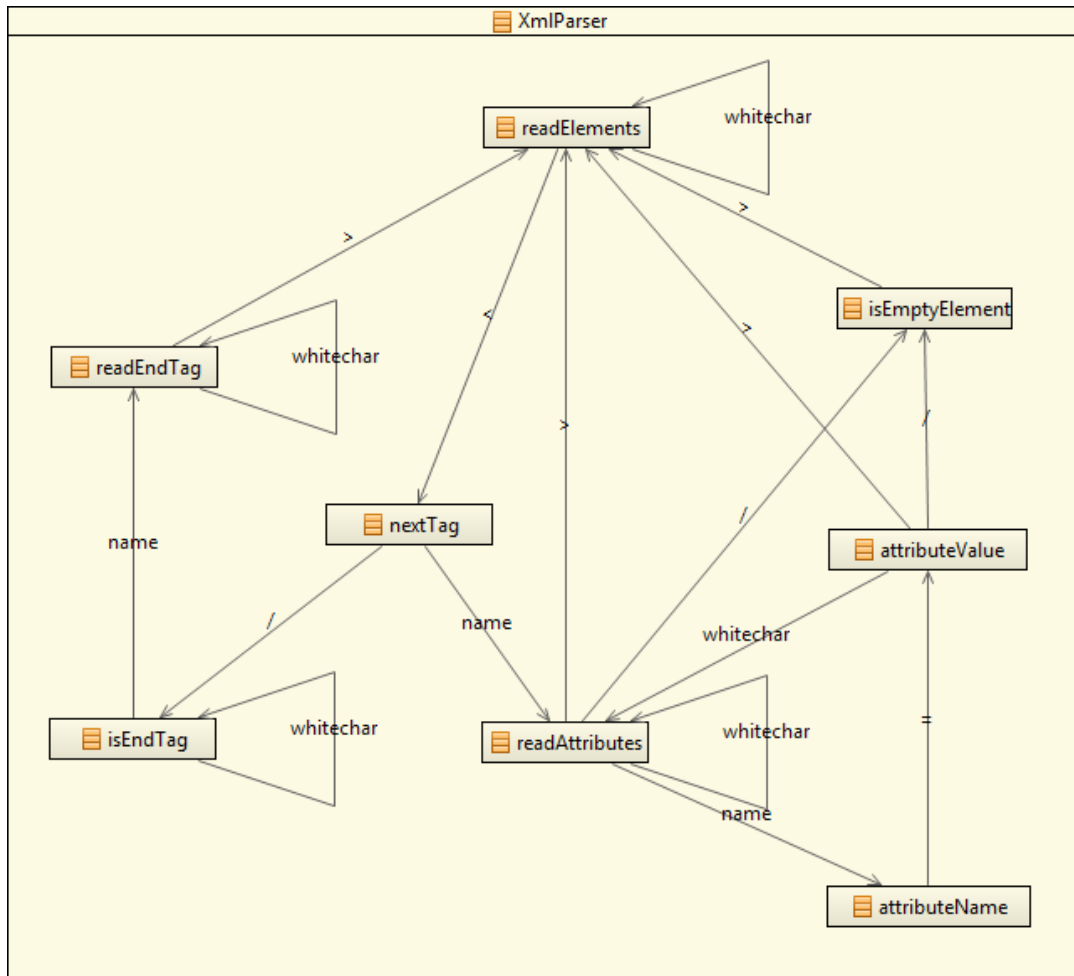
typeOf	State
name	readElements
transition [<]	
transition [name]	
transition [=]	
transition [/]	
transition [>]	
transition [whitechar]	

typeOf	Transition
targetState	readEndTag
target	

nextTag : State
state[2]
state[3]
state[4]
state[5]
state[6]
state[7]
state[8]

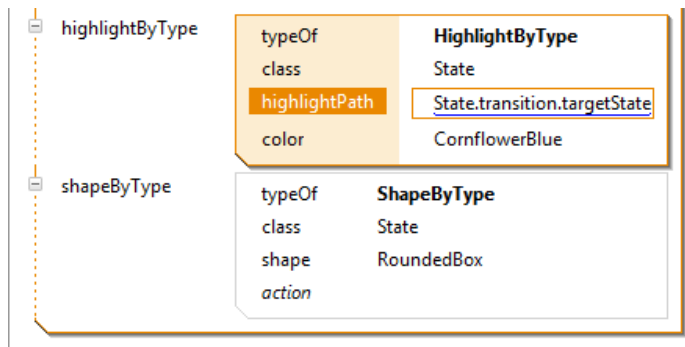
target	Event
/	com.actifsource.statemachine.specific.XmlParser Event
<	com.actifsource.statemachine.specific.XmlParser Event
=	com.actifsource.statemachine.specific.XmlParser Event
>	com.actifsource.statemachine.specific.XmlParser Event
name	com.actifsource.statemachine.specific.XmlParser Event
whitechar	com.actifsource.statemachine.specific.XmlParser Event

↪ Enter all of the following Transitions:

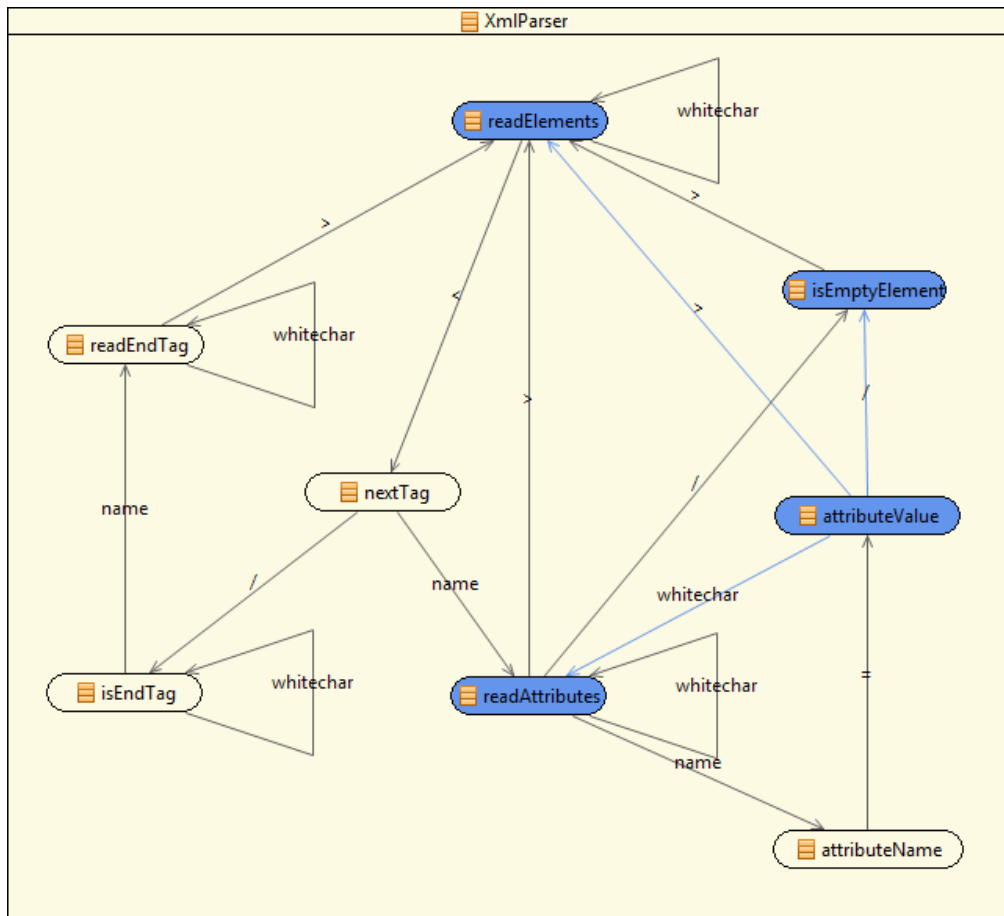


Custom shapes & Interaction

- ① The shape a Resource is displayed in the Domain Diagram can be changed by setting **shapeByType** in the **DiagramType**.
- ↪ For our state machine example, add a **ShapeByType** object for class **State** and choose **RoundedBox** as shape.
- ① It is now possible to highlight a path and its nodes, when the mouse hovers over a node:
- ↪ Create a new **HighlightByType** setting for **State** which emphasizes the path *State.transition.targetState*.

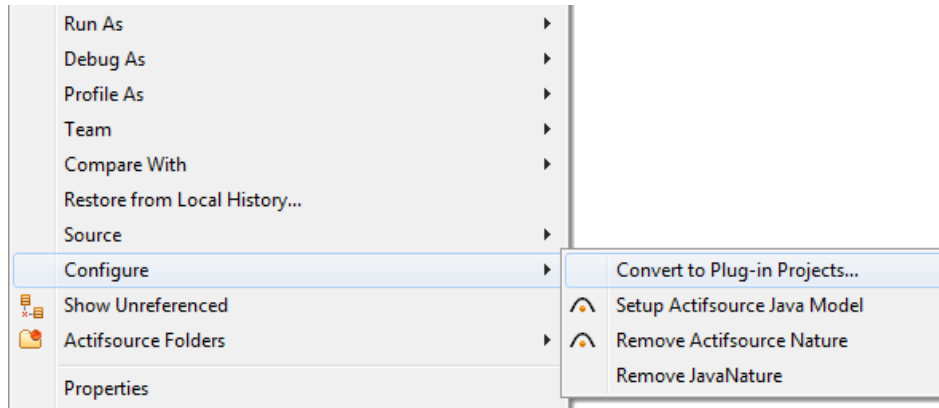


① If we hover over the State *attributeValue* the Statemachine will be displayed as follows:

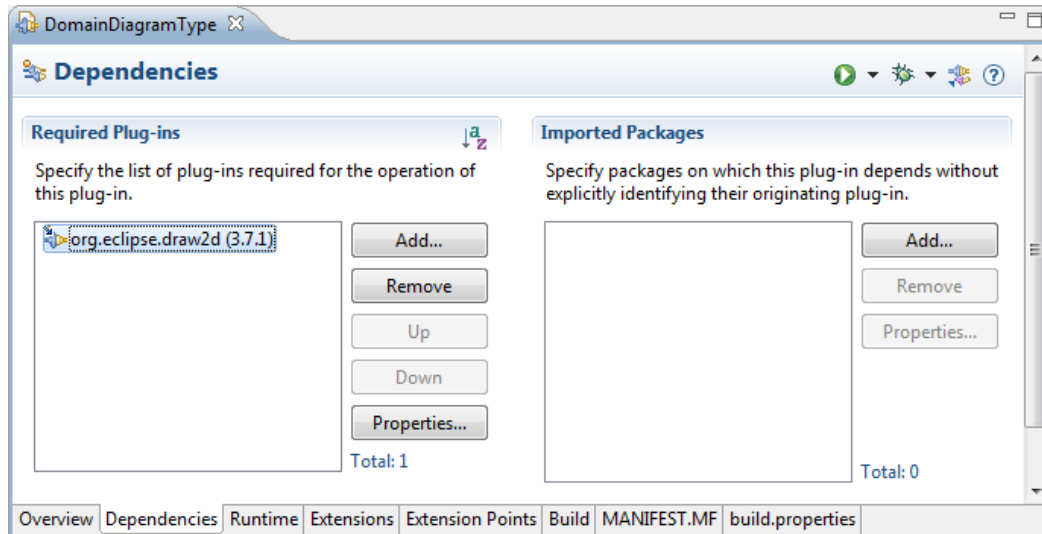


- ① It is possible to set a tooltip, which is shown when the mouse hovers over a shape.
- ① For this, we have to create a *ToolTipAspect* Java class which returns the tooltip figure.
- ① The necessary data is read from the model through the Low-Level API.
- ① We will show as a simple example, how there can task description be displayed for the states.

- ① We need to add a dependency to Eclipse Draw2d.
- ↩ Convert the Project to a Plug-in Project.



- ↪ In the *META-INF* folder, open the *MANIFEST.MF* file
- ↪ Select the *Dependencies* tab.



- ↪ Add a dependency to *org.eclipse.draw2d*.

↪ Add a string attribute taskDescription to the **Class State**.

property[1]	typeOf	ch.actifsource.core.Attribute
	name	taskDescription
	comment	
	subjectCardinality	Cardinality0_N
	range	ch.actifsource.core.StringLiteral
	defaultValue	

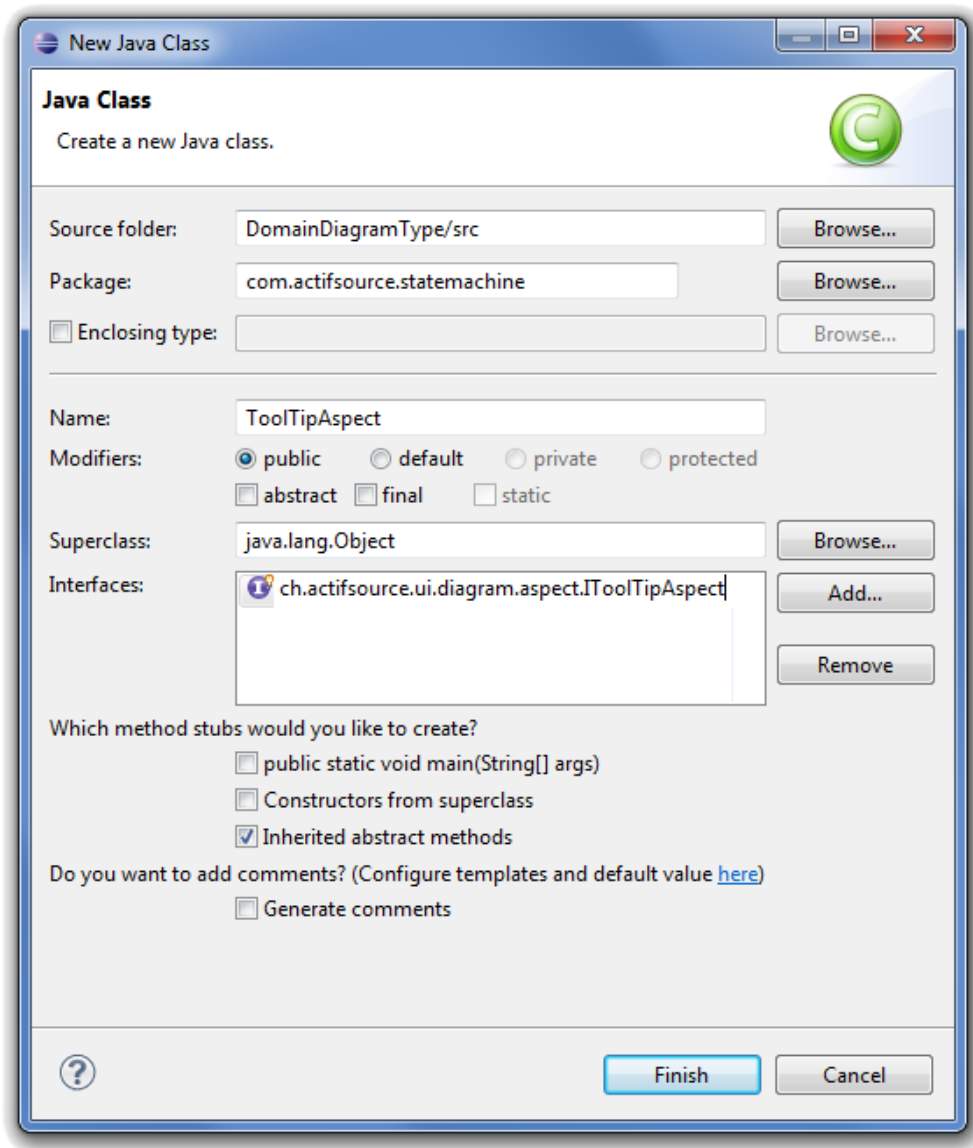
ⓘ We will display the taskDescription strings in a tooltip, which appears, when the mouse hovers over a state.

ⓘ You can add now the taskDescriptions to the model.

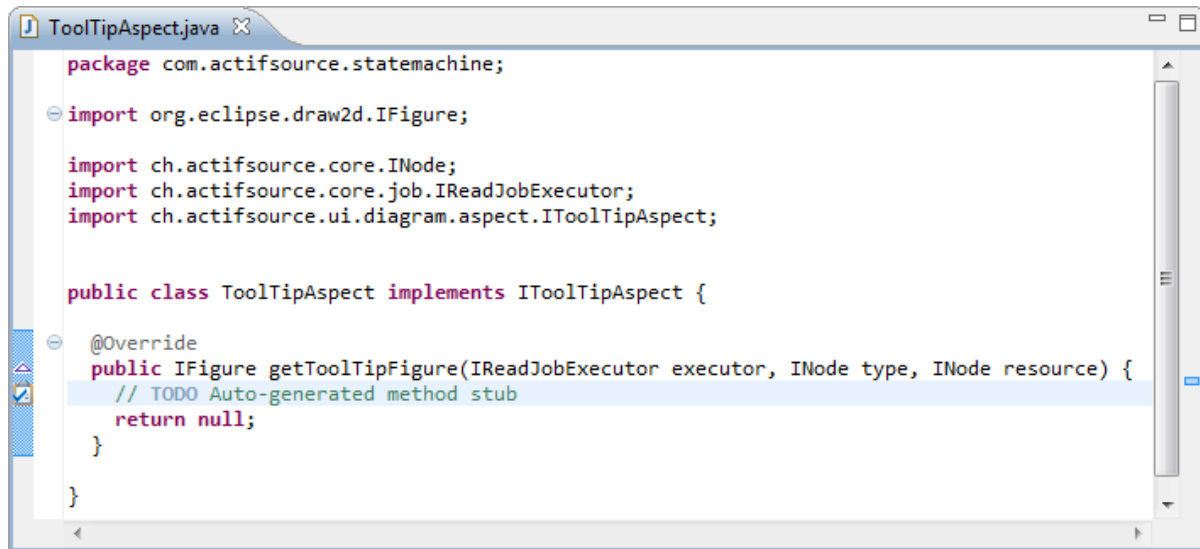
↪ For example, add some for the State attributeValue as shown below:

state[5]	typeOf	com.actifsource.statemachine.generic.State
	name	attributeValue
	taskDescription[1]	Create attribute (name, value)
	taskDescription[2]	Add attribute to current element
	transition[<]	
	transition[name]	
	transition[=]	
	transition[/]	/ : Transition
	transition[>]	> : Transition
	transition[whitechar]	whitechar : Transition

- ① Add a new Java named `ToolTipAspect` to the `src` folder which implements `ITooltipAspect`.



- ① There is a single method to implement that returns an IFigure: `getTooltipFigure()`.
- ① The following parameters are given in the parameter list:
 - *executor*: The *IReadJobExecutor* which gives us access to the internal Actifsource database.
 - *type*: The type of the resource.
 - *resource*: The resource itself.



```
package com.actifsource.statemachine;

import org.eclipse.draw2d.IFigure;

import ch.actifsource.core.INode;
import ch.actifsource.core.job.IReadJobExecutor;
import ch.actifsource.ui.diagram.aspect.IToolTipAspect;

public class TooltipAspect implements ITooltipAspect {

    @Override
    public IFigure getTooltipFigure(IReadJobExecutor executor, INode type, INode resource) {
        // TODO Auto-generated method stub
        return null;
    }
}
```

- ① Note: We could use the same `TooltipAspect` class for different resource types by just checking the *type* before reading the data needed.

↪ Let us first select the State node's taskDescription nodes.

```
@Override
public IFigure getToolTipFigure(IReadJobExecutor readJobExecutor, INode type, INode resource) {
    Select.objectsFor
    // TODO Auto-generated method stub
    return null;
}
```

Ⓢ objectForAttribute(IReadJobExecutor executor, INode attribute, INode resource) : INode - Select
Ⓢ objectForAttributeOrNull(IReadJobExecutor executor, INode attribute, INode subject) : INode - Select
Ⓢ objectForAttributeOrNull(IReadJobExecutor executor, INode attribute, INodeSet resources) : INode - Select
Ⓢ objectForPropertyOrNull(IReadJobExecutor executor, INode property, INode resource) : INode - Select
Ⓢ objectForRelationOrNull(IReadJobExecutor executor, INode relation, INode subject) : INode - Select
Ⓢ objectsForAttribute(IReadJobExecutor executor, INode attribute, INode resource) : INodeList - Select
Ⓢ objectsForAttribute(IReadJobExecutor executor, INode attribute, INodeSet resources) : INodeList - Select
Ⓢ objectsForRelation(IReadJobExecutor executor, INode relation, INode subject) : INodeList - Select
Ⓢ objectsForRelation(IReadJobExecutor executor, INode relation, INodeSet subjects) : INodeList - Select
Ⓢ objectsForRelationOfType(IReadJobExecutor executor, INode relation, INode subject, INode type) : INodeList - Select

Press 'Ctrl+Space' to show Template Proposals

Returns the object nodes for an attribute.

Parameters:

executor the context for the execution
attribute resource typeOf [CorePackage.Attribute](#)
resource resource to query the attribute

Returns:

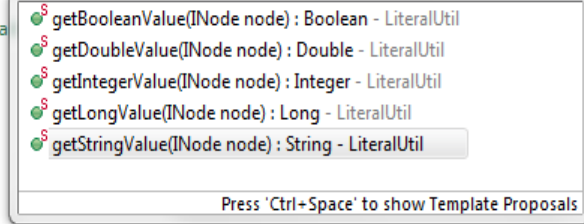
the requested attribute for resource

Press 'Tab' from proposal table or click for focus

- ↪ Since there can be any number of taskDescriptions, enter the call statement *Select.objectsForAttribute*.
- The *executor* will be the *executor* given in the parameter list.
 - The *attribute* is `GenericPackage.State_taskDescription` which is provided by the exported `JavaModel`.
 - The *resource* is the given *resource* (i.e. the State, since we will reference this *IToolTipAspect* from State).

↪ Iterate over the returned INodeList.

```
public IFigure getToolTipFigure(IReadJobExecutor readJobExecutor, INode type, INode resource) {  
    for (INode taskDescription : Select.objectsForAttribute(readJobExecutor, GenericPackage.State_taskDescription, resource)) {  
        LiteralUtil.get  
    }  
    // TODO Auto-generated method stub  
    return null;  
}
```



↪ Read the attribute's String value.

↳ Collect all in a StringBuffer:

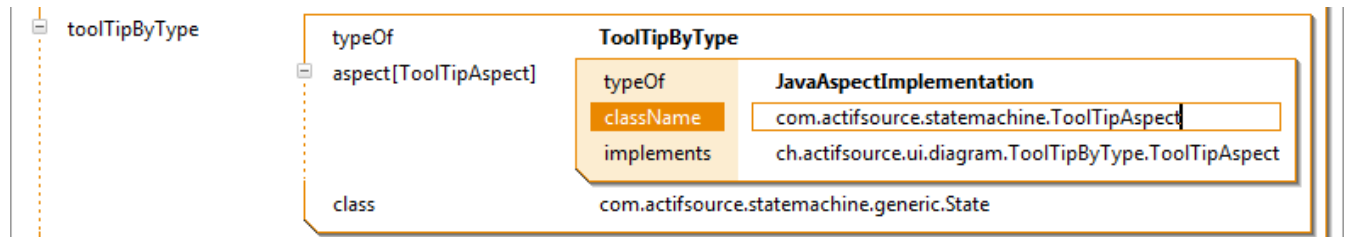
- Let the first line contain the string `Tasks :`
- Append for every `taskDescription` a line with a bullet (Unicode character `0x2022`) at the beginning.

```
StringBuilder sb = new StringBuilder("Tasks:");
for (INode taskDescription : Select.objectsForAttribute(readJobExecutor, GenericPackage.State_taskDescription, resource)) {
    sb.append("\n\u2022 ");
    sb.append(LiteralUtil.getStringValue(taskDescription));
}
```

↳ Return a `ToolTipLabel` containing the resulting string.

```
@Override
public IFigure getToolTipFigure(IReadJobExecutor readJobExecutor, INode type, INode resource) {
    StringBuilder sb = new StringBuilder("Tasks:");
    for (INode taskDescription : Select.objectsForAttribute(readJobExecutor, GenericPackage.State_taskDescription, resource)) {
        sb.append("\n\u2022 ");
        sb.append(LiteralUtil.getStringValue(taskDescription));
    }
    return new ToolTipLabel(sb.toString());
}
```

- ① Now we must reference the Java class from the **DiagramType**:
- ↪ In our StateDiagramType, add a TooltipByType.
- ↪ Set **class** to **State**.
- ↪ Choose *com.actifsource.statemachine.TooltipAspect* as **className**.



- ↪ Now the diagram shows the tasks of a State in a tooltip:

