Tutorial

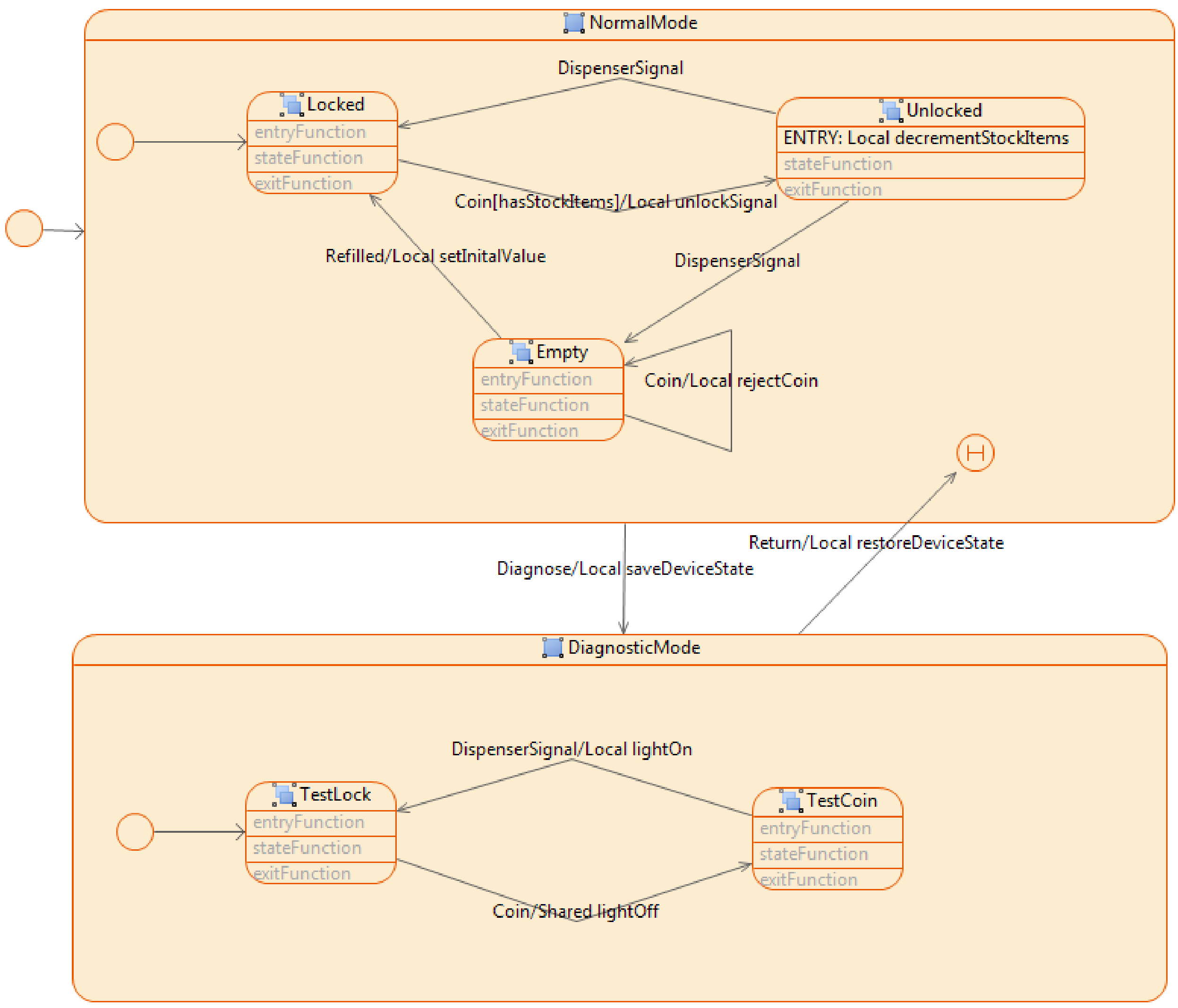
UML Statemachine

© 2010 actifsource GmbH, Switzerland - all rights reserved.

Tutorial	Actifsource Tutorial – State Machine
Required Time	• 70 Minutes
Prerequisites	• Actifsource Tutorial – Installing Actifsource • Actifsource Tutorial – Simple Service • Actifsource Tutorial – Domain Diagram Type • Actifsource Tutorial – Domain Diagram Type II
Goal	• Create UML State Machines using the built-in Actifsource solution
Topics covered	• Create UML State Machines with states, superstates, history states, (entry, exit, state) actions and transition guards
Notation	↳ To do ⓘ Information • **Bold**: Terms from actifsource or other technologies and tools • <u>**Bold underlined**</u>: actifsource Resources • <u>Underlined</u>: User Resources • <u>*UnderlinedItalics*</u>: 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 <u>www.actifsource.com</u>
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.
Compatibility	Created with actifsource Version 6.8.1

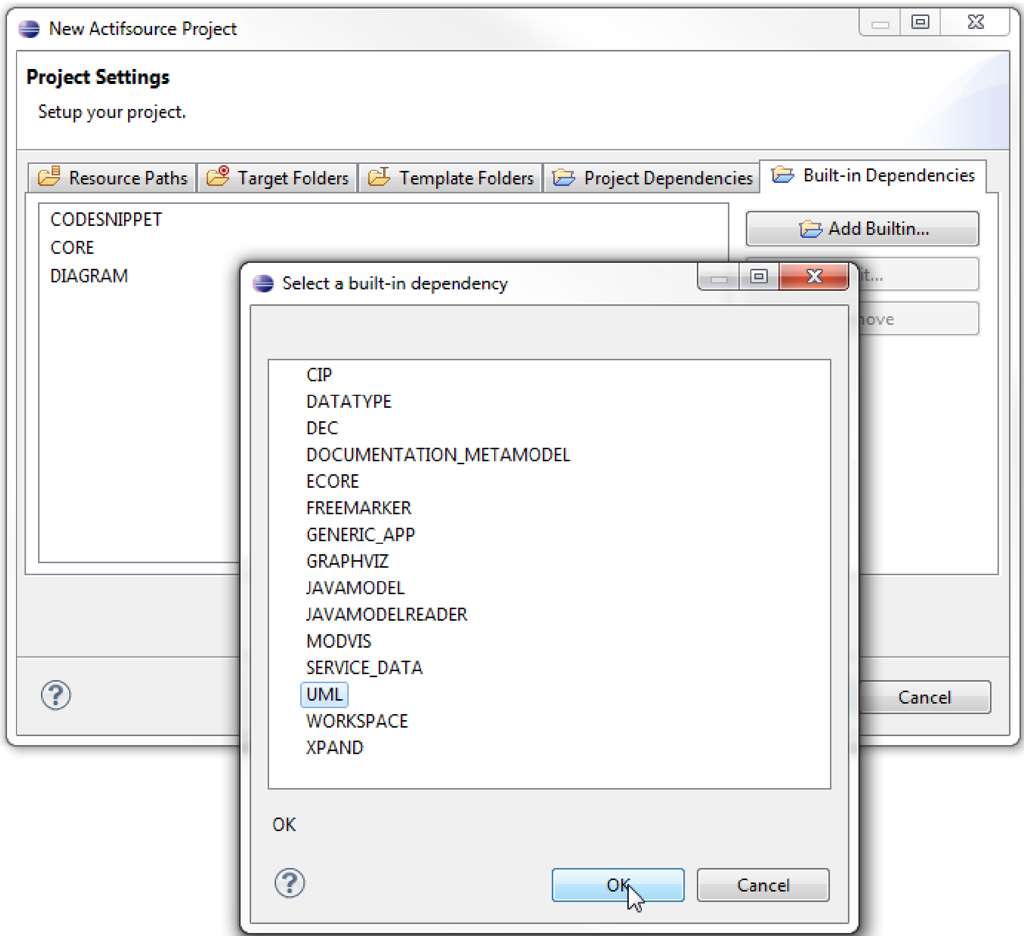
© 2010 actifsource GmbH, Switzerland - all rights reserved.

- Design an UML state machine for a simple coin machine process and create a state diagram that represents the UML state machine:



© 2010 actifsource GmbH, Switzerland - all rights reserved.

Part I: Preparation

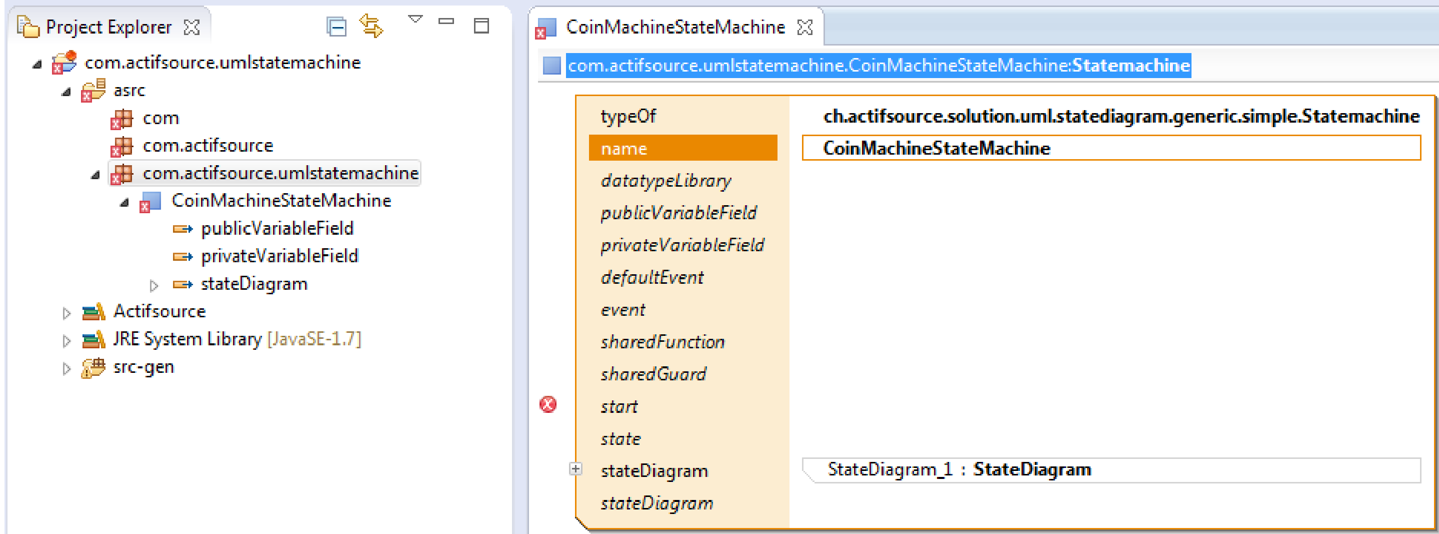


- Create a new Actifsource project with the name com.actifsource.umlstatemachine
- Change to the tab **Built-in Dependencies** in the **New Actifsource Project** dialog
- Add the built-in dependency **UML** (which makes all resource needed to build and represent UML state machines available in our new project).
- Close both dialogs by clicking OK and Finish

© 2010 actifsource GmbH, Switzerland - all rights reserved.

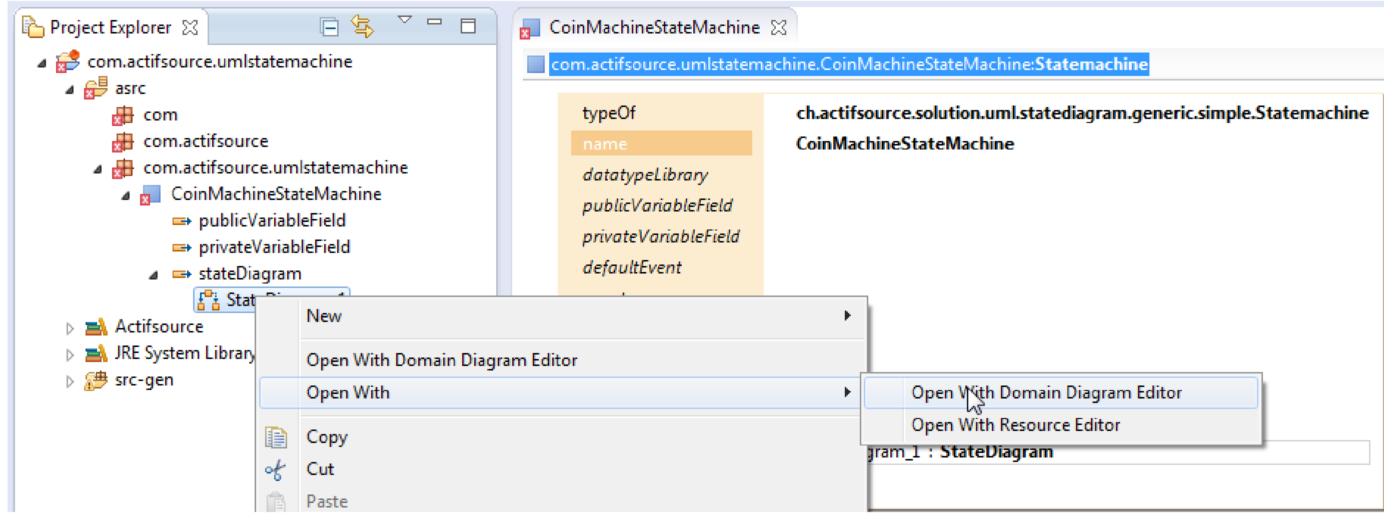
Part II:

Design an UML State Machine



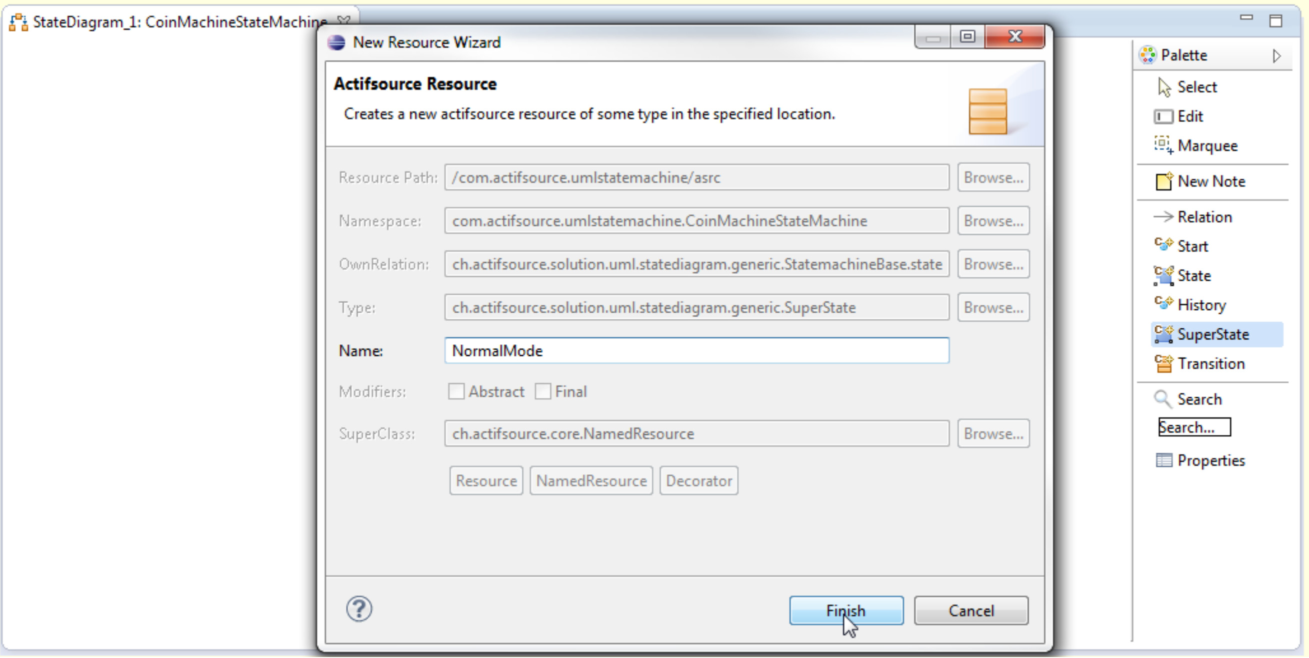
- Create a new resource of type ch.actifsource.solution.uml.statediagram.generic.simple.Statemachine
- Enter CoinMachineStateMachine as the name of the new resource

© 2010 actifsource GmbH, Switzerland - all rights reserved.



↳ Open the state diagram of CoinMachineStateMachine, StateDiagram_1, in the **Domain Diagram Editor**

© 2010 actifsource GmbH, Switzerland - all rights reserved.



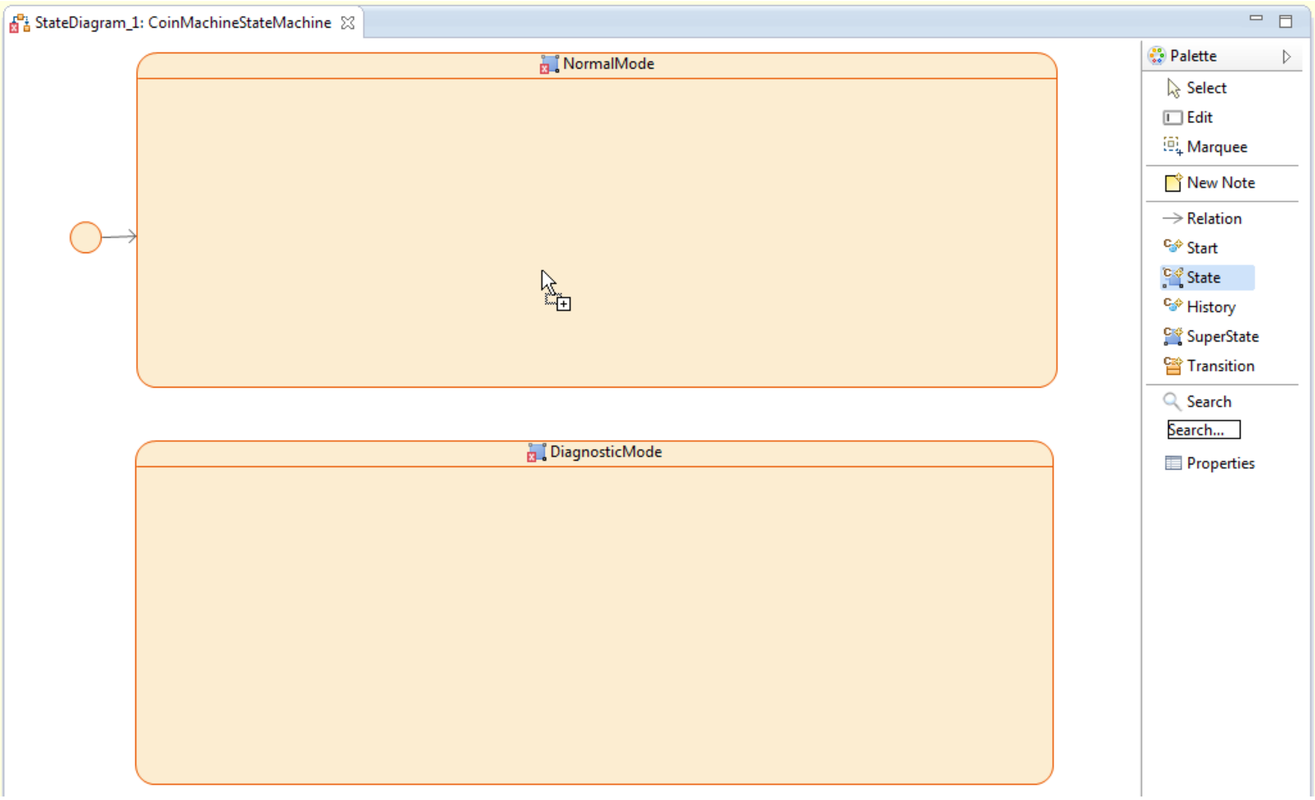
First, we create two superstates, called NormalMode and DiagnosticMode:

- Select SuperState from the Palette and left-click in the **Diagram Editor** to create a superstate
- Enter `NormalMode` as the name of the new SuperState in the **New Resource Wizard**
- In the same way, create a SuperState called DiagnosticMode
- Select Start from the palette and right-click in the **Diagram Editor** to create a start state (i.e., the default or initial state of the state machine)
- In the Select mode, you can now re-size and re-position the states as usual in the **Diagram Editor**

© 2010 actifsource GmbH, Switzerland - all rights reserved.

↳ Select Relation from the Palette and insert a relation from the start state symbol to the superstate NormalMode

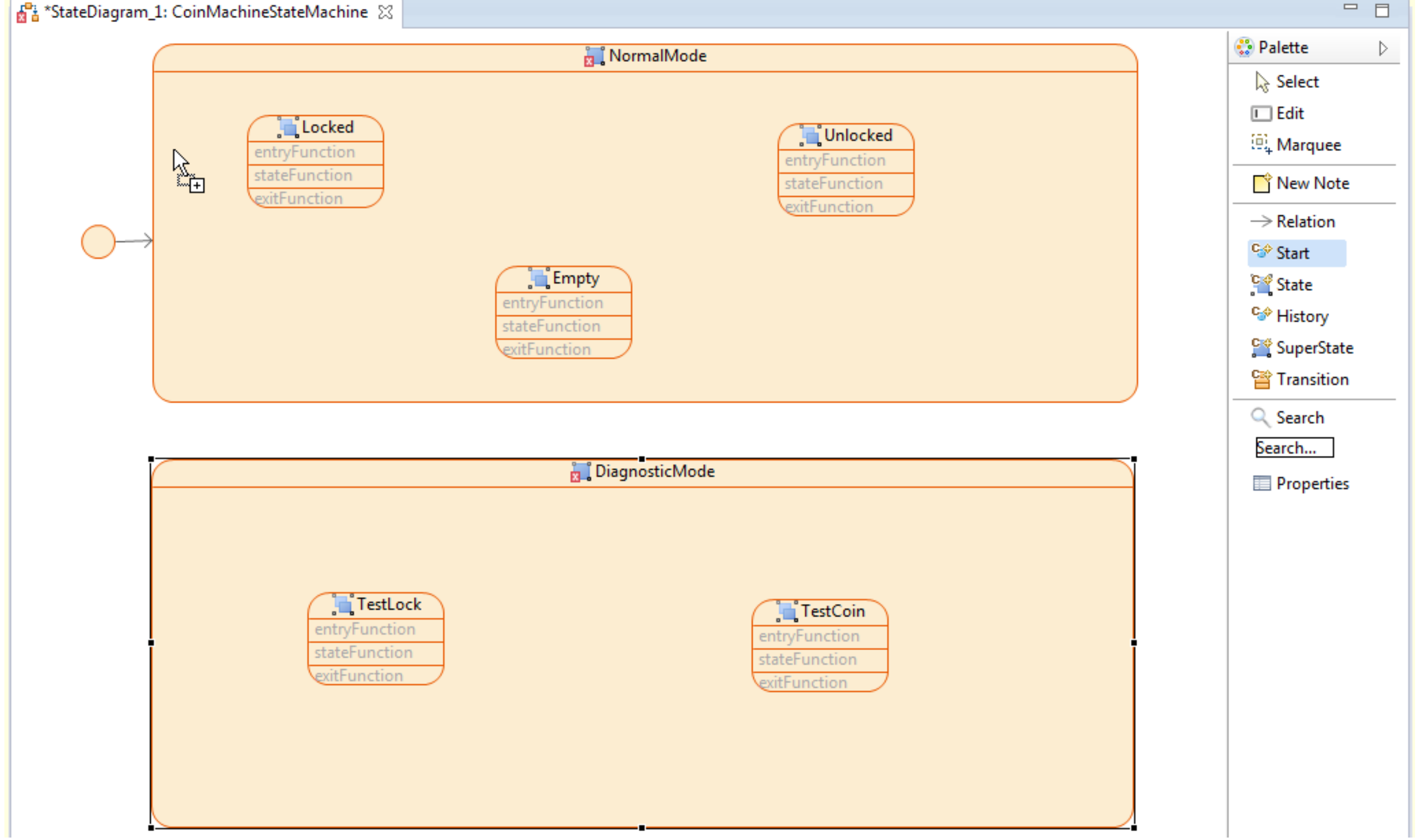
© 2010 actifsource GmbH, Switzerland - all rights reserved.



Now, we create nested states to define the behavior in the NormalMode:

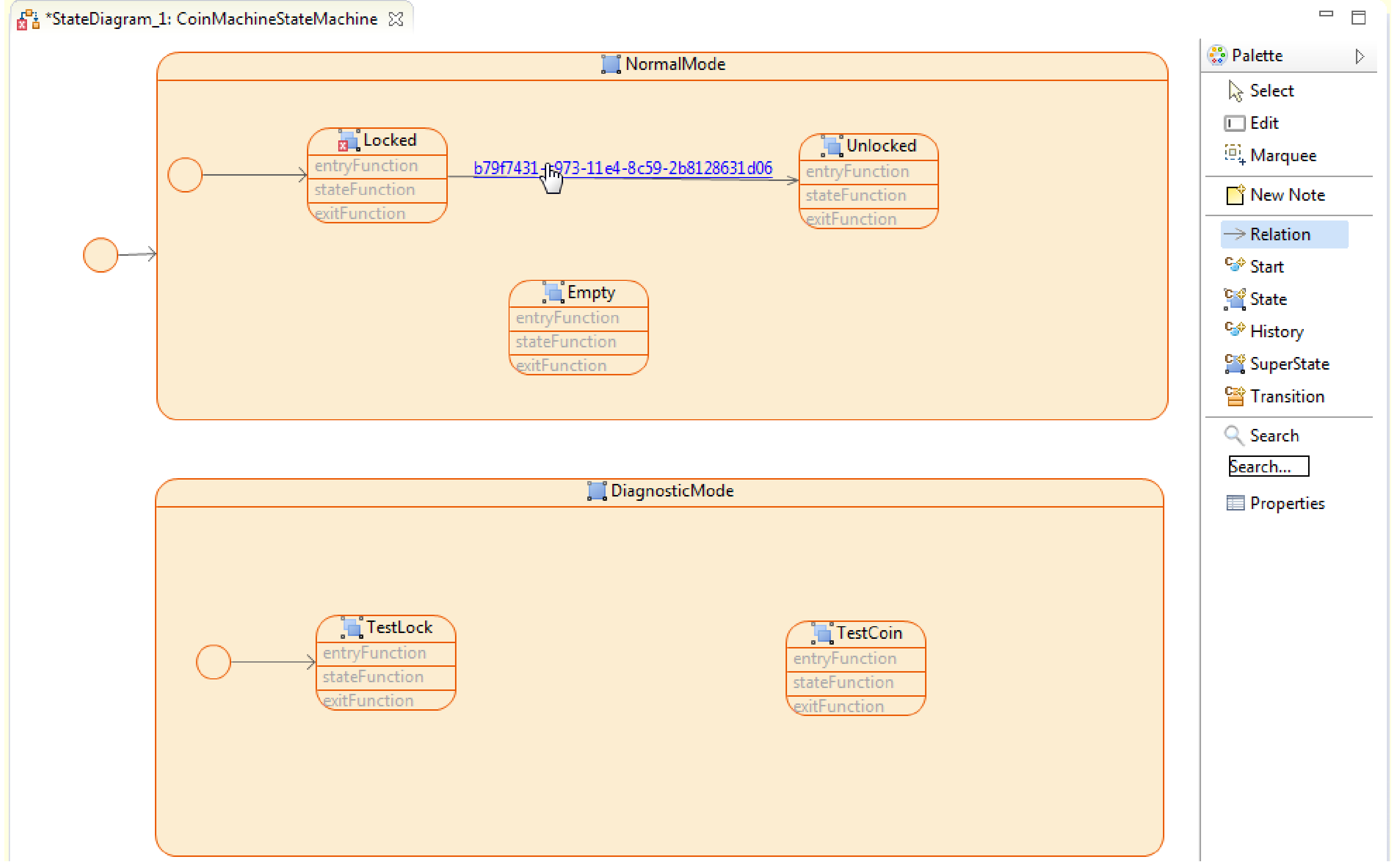
- Select State from the Palette and left-click in the lower section of the NormalMode state to create a nested state in the superstate NormalMode
- Enter the name `Locked` as name of the State in the **New Resource Wizard**
- In the same way create the two nested states Locked and Empty in the NormalMode and the two nested states TestLock and TestCoin in the superstate DiagnosticMode

© 2010 actifsource GmbH, Switzerland - all rights reserved.



- Select Start from the Palette and left-click in the lower section of the NormalMode state (see above) to create a default or start state
- As before, select Relation from the Palette and create a relation from the start state to the state Locked
- In the same way, create a start state in the DiagnosticMode and create a relation from this start state to the state TestLock

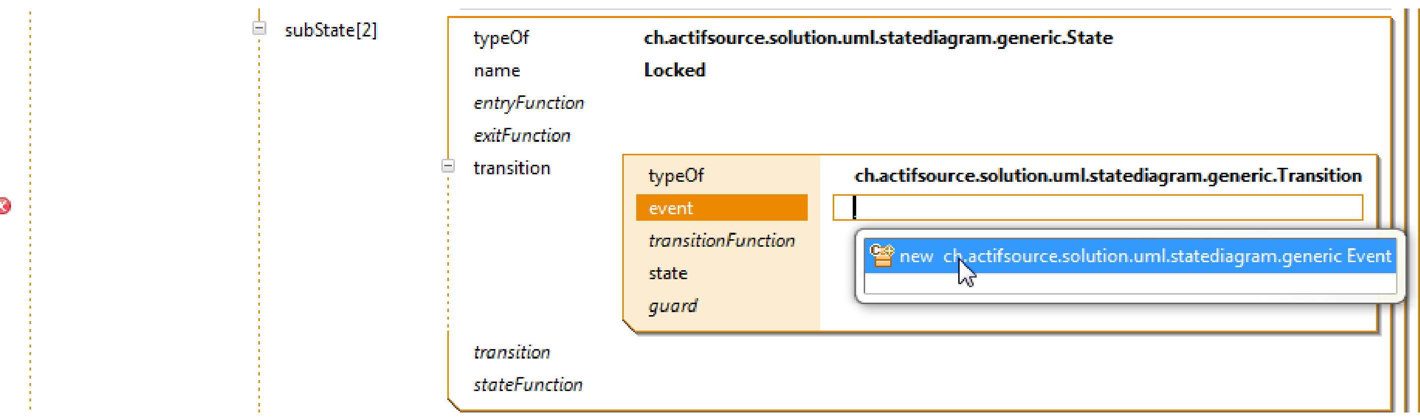
© 2010 actifsource GmbH, Switzerland - all rights reserved.



Next, we define the state transitions and events:

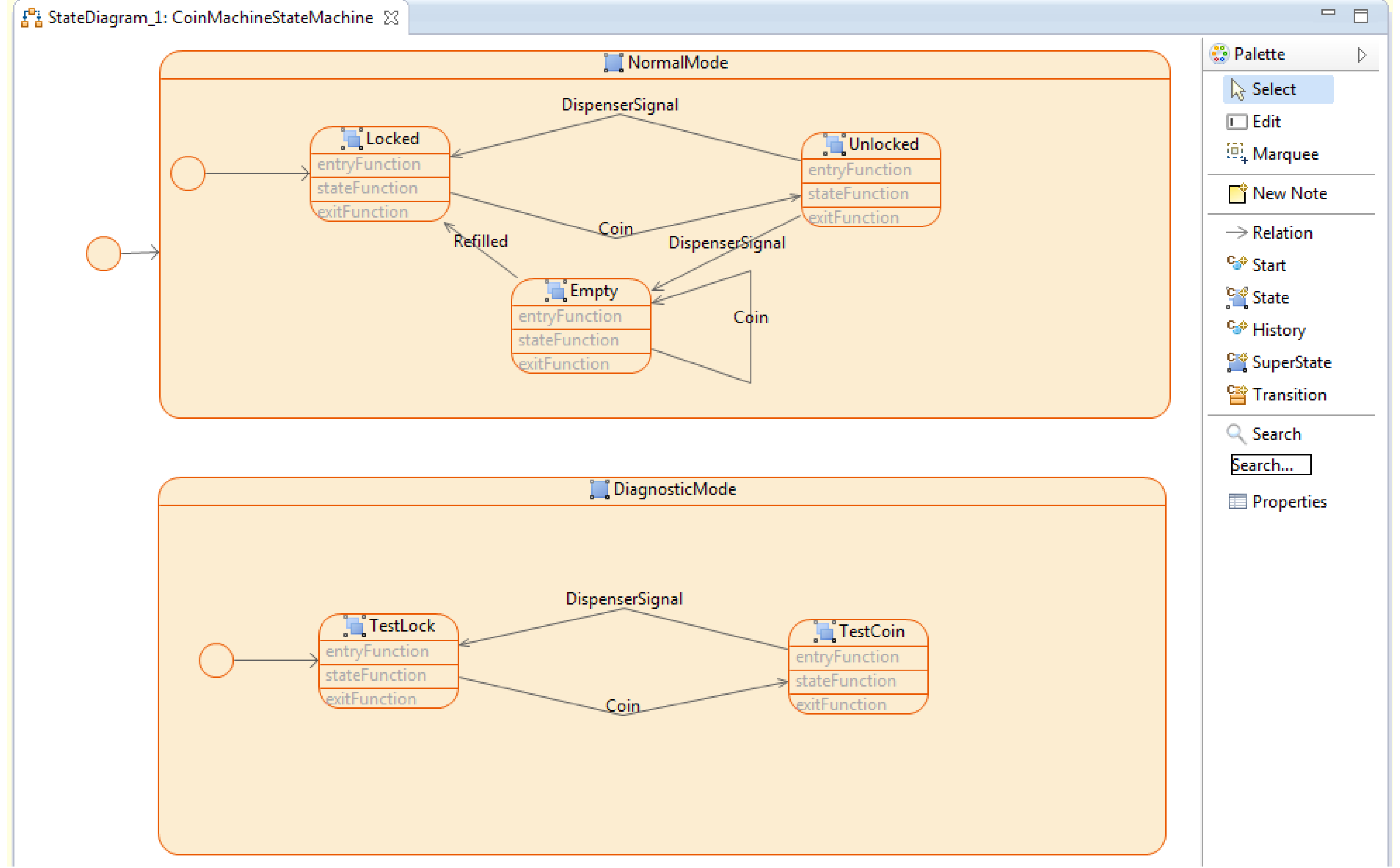
- Select Relation from the Palette and create a relation from the state Locked to the state Unlocked
- With Control+Click on the GUID of the newly created relation, you can now open the transition in the **Resource Editor**

© 2010 actifsource GmbH, Switzerland - all rights reserved.



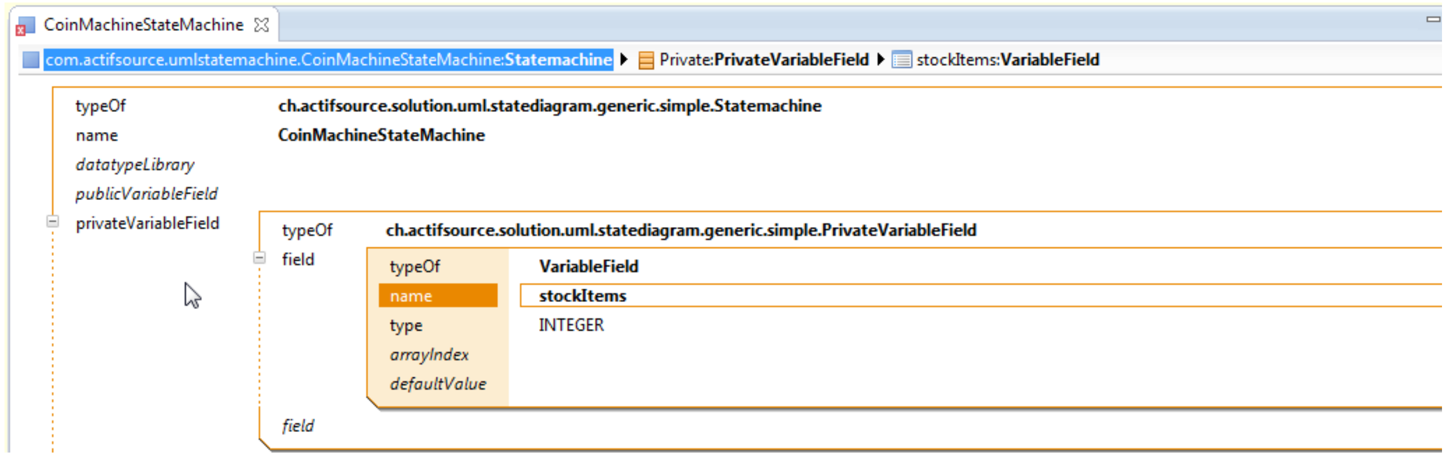
- In the **Resource Editor**, create a new Event for the transition from state Locked to Unlocked
- Give the name `Coin` to the newly created event

© 2010 actifsource GmbH, Switzerland - all rights reserved.



- Switch to the open StateDiagram_1 in the **Diagram Editor** and check that the transition from Locked to Unlocked is now labeled 'Coin'
- In the same way, create the following transition and events: Unlocked-(DispenserSignal)->Locked, Empty-(Refilled)->Locked, Empty-(Coin)->Empty, TestCoin-(DispenserSignal)->TestLock, TestLock-(Coin)->TestCoin

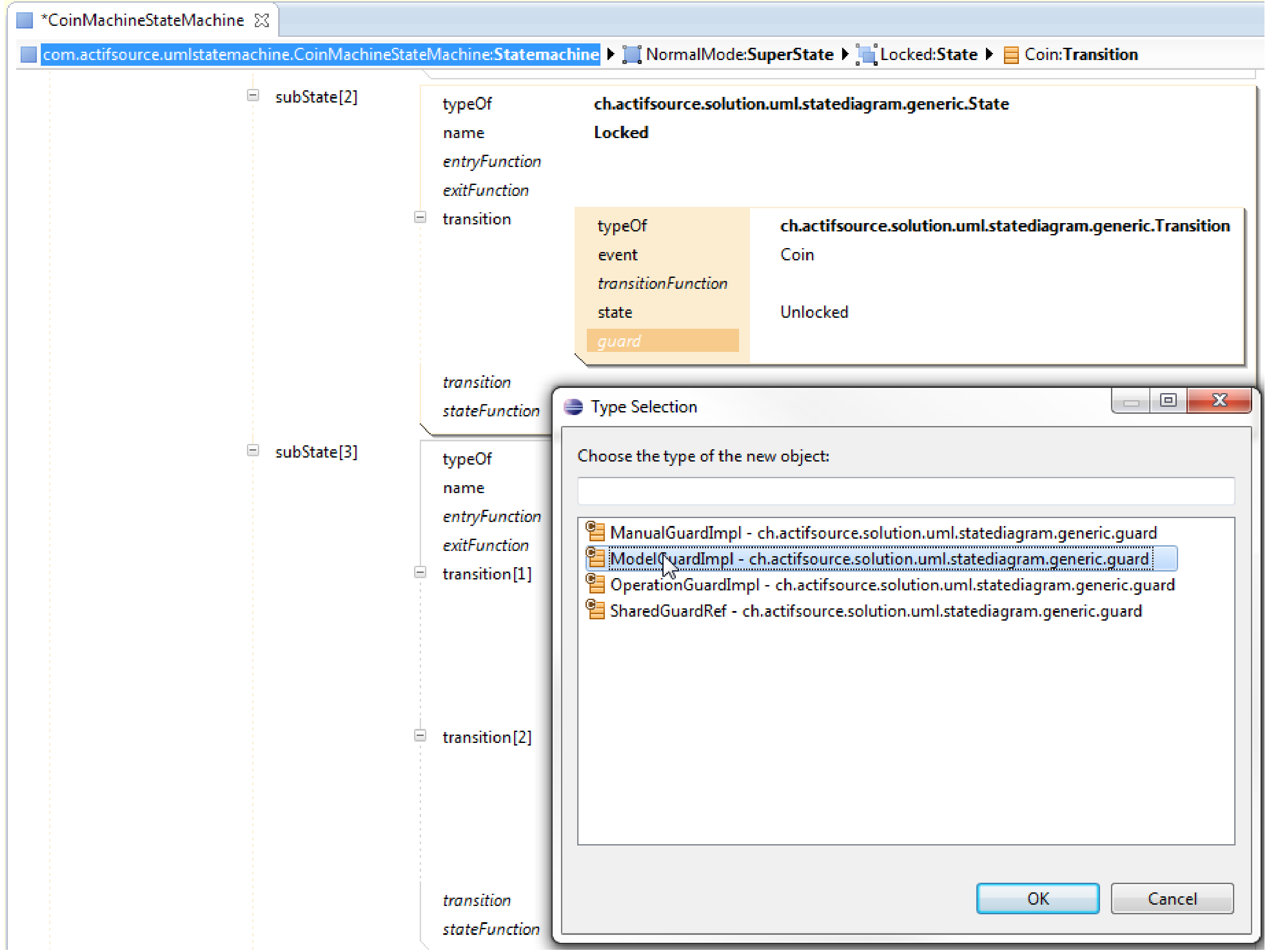
© 2010 actifsource GmbH, Switzerland - all rights reserved.



We want to create a condition that is only true if the machine is non-empty. Thereto, we introduce a variable stockItems that keeps track of the number of items left in the machine:

- Open CoinMachineStateMachine the **Resource Editor**
- Add a **PrivateVariableField** to the CoinMachineStateMachine
- Create a **VariableField** with name `stockItems` and with type INTEGER as **field**

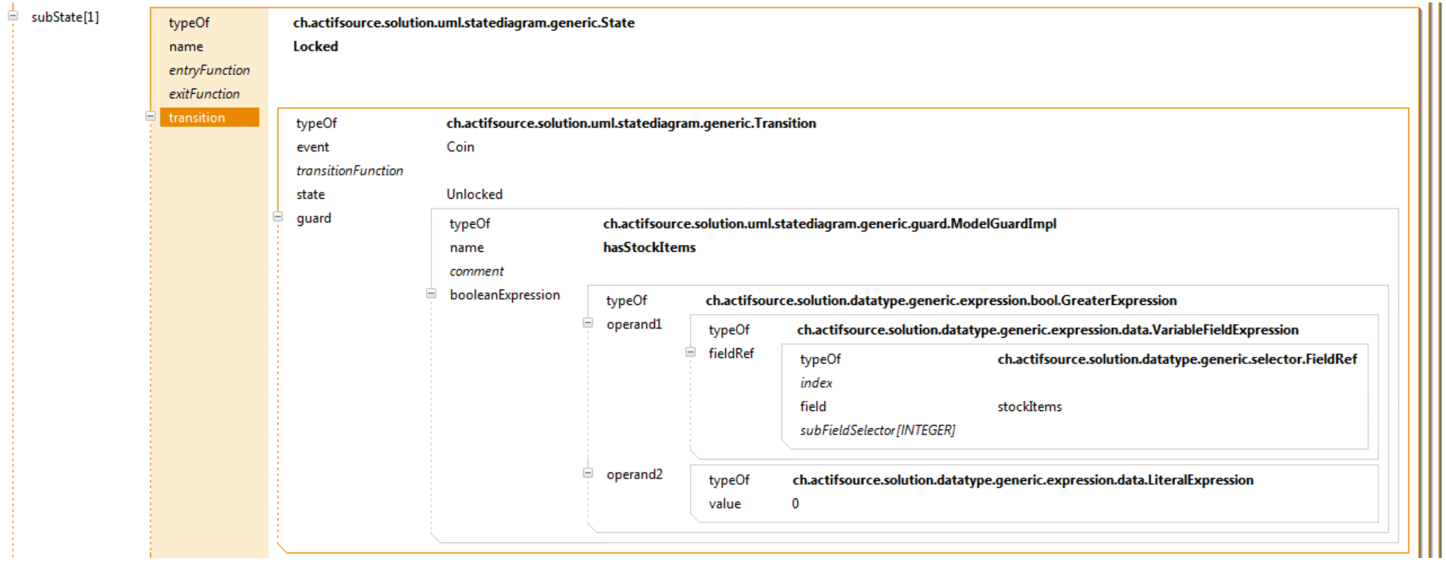
© 2010 actifsource GmbH, Switzerland - all rights reserved.



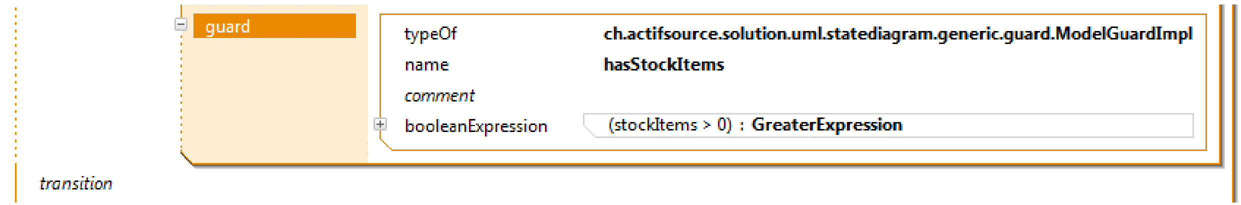
Next, we add guards to transitions such that the corresponding transitions only "fire" if the guard evaluates to TRUE:

- By Control+Click on the transition Unlocked-(Coin)->Locked, open the transition in the resource editor
- Use the Content Assist to create a new **ModelGuardImpl** named hasStockItems as guard of the transition (see above)

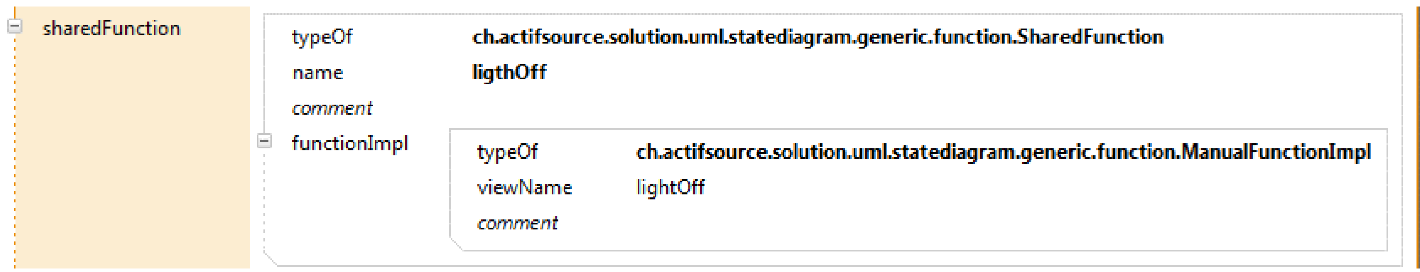
© 2010 actifsource GmbH, Switzerland - all rights reserved.



- Add a **GreaterExpression** as booleanExpression to the **ModelGuardImpl**
- Create an **operand1** of type **VariableFieldExpression** with a fieldRef with the **VariableField** stockItems as field
- Create an **operand2** of type **LiteralExpression** with value 0
- Close the **booleanExpression**: Note that the Boolean expression stockItems > 0 is now displayed to represent the condition of the guard



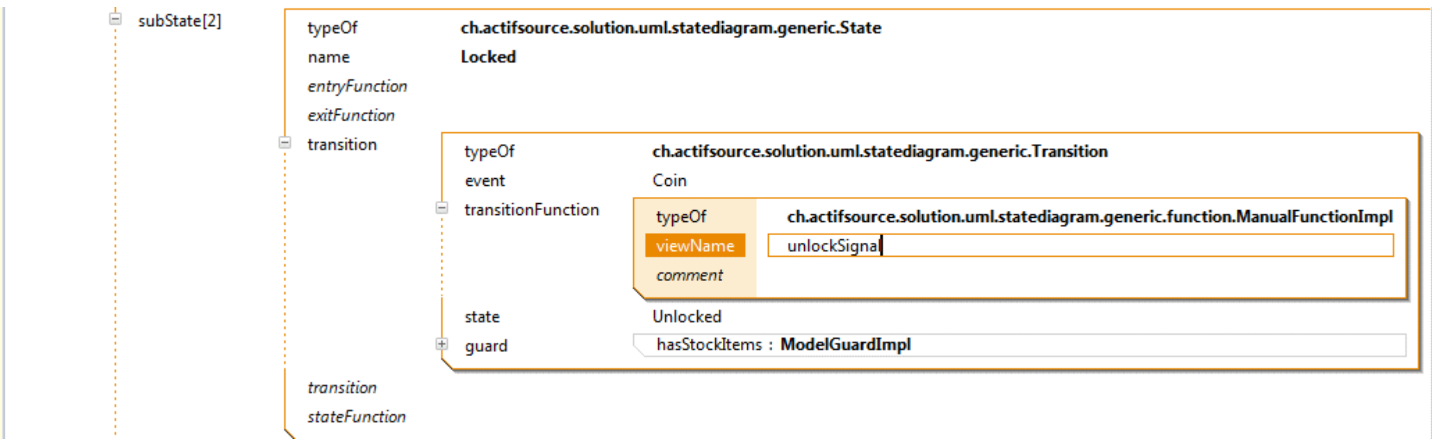
© 2010 actifsource GmbH, Switzerland - all rights reserved.



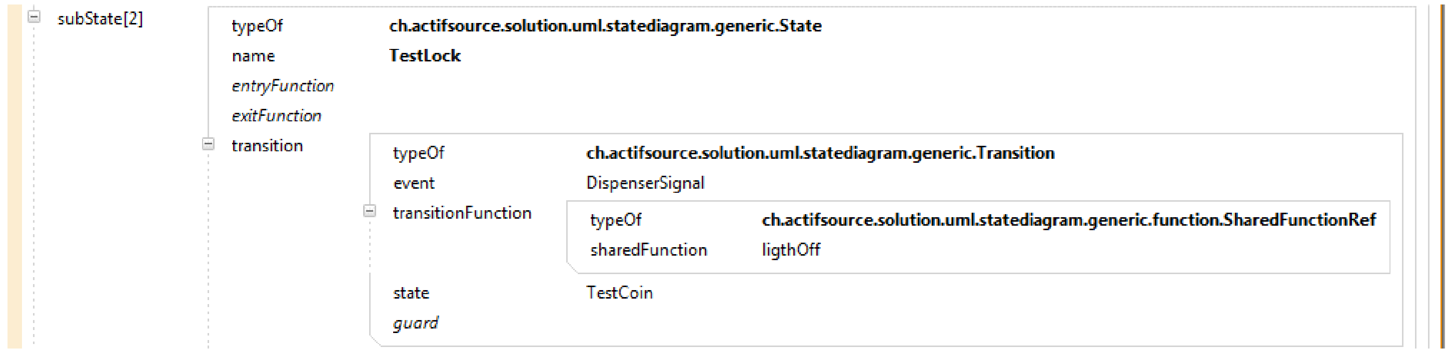
Next, we add actions that are executed together with transitions and introduce shared functions which can be used as actions by multiple transitions:

- Open CoinMachineStateMachine in the **Resource Editor**
- Create a new **SharedFunction** called lightOff as **sharedFunction** to CoinMachineStateMachine
- Create a new **ManualFunctionImpl** with **viewName** lightOff as **functionImpl** to the **SharedFunction**

© 2010 actifsource GmbH, Switzerland - all rights reserved.

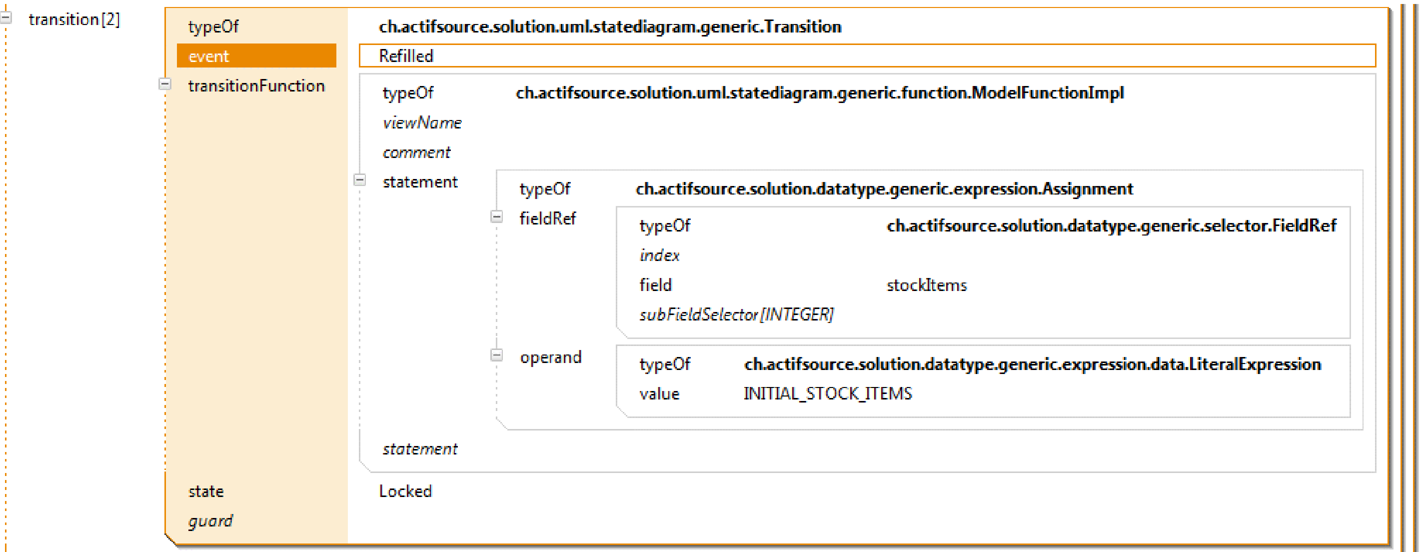


- Add a **transitionFunction** of type **ManualFunctionImpl** to the transition Locked->Unlocked (see above)
- In the same way create the following transitionFunctions of type **ManualFunctionImpl**:
  - rejectCoin (Empty->Empty)
  - lightOn (TestCoin->TestLock)



- Add a **transitionFunction** of type **SharedFunctionRef** to the transition TestLock->TestCoin which uses the **sharedFunction** lightOff

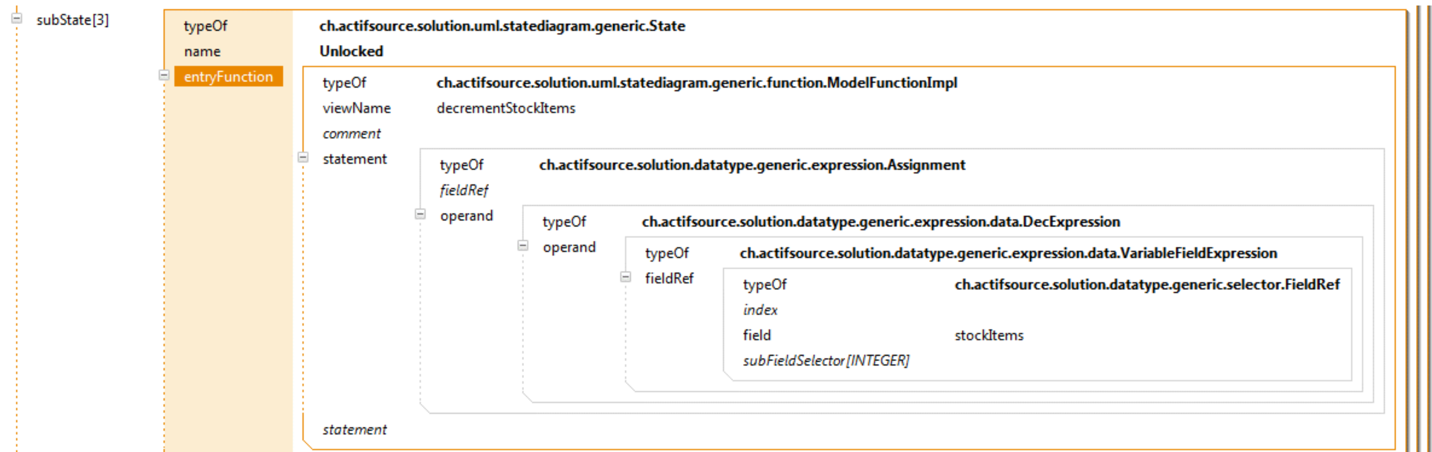
© 2010 actifsource GmbH, Switzerland - all rights reserved.



When the event Refilled occurs, the variable stockItems (i.e, the private variable that counts the number of stock items) should be set to the initial numbers of items:

- Add a transitionFunction of type **ModelFunctionImpl** to the transition Empty-(Refilled)->Locked
- Add a **statement** of type **Assignment** to the **ModelFunctionImpl**
- Insert a fieldRef with field stockItems
- Add an **operand** of **LiteralExpression** with **value** INITIAL_STOCK_ITEMS

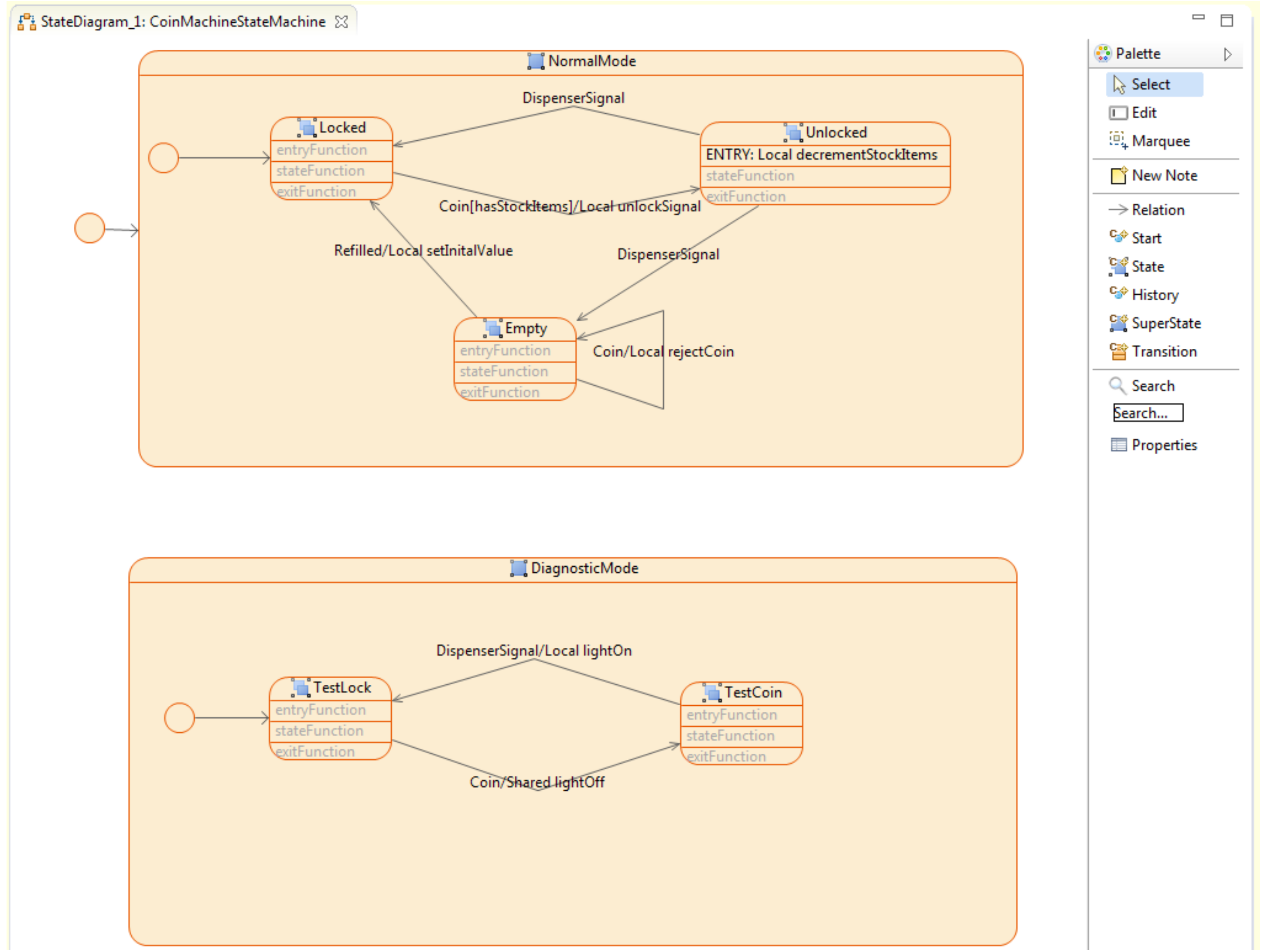
© 2010 actifsource GmbH, Switzerland - all rights reserved.



Each time the machine is unlocked, the number of stock items should be decremented by one:

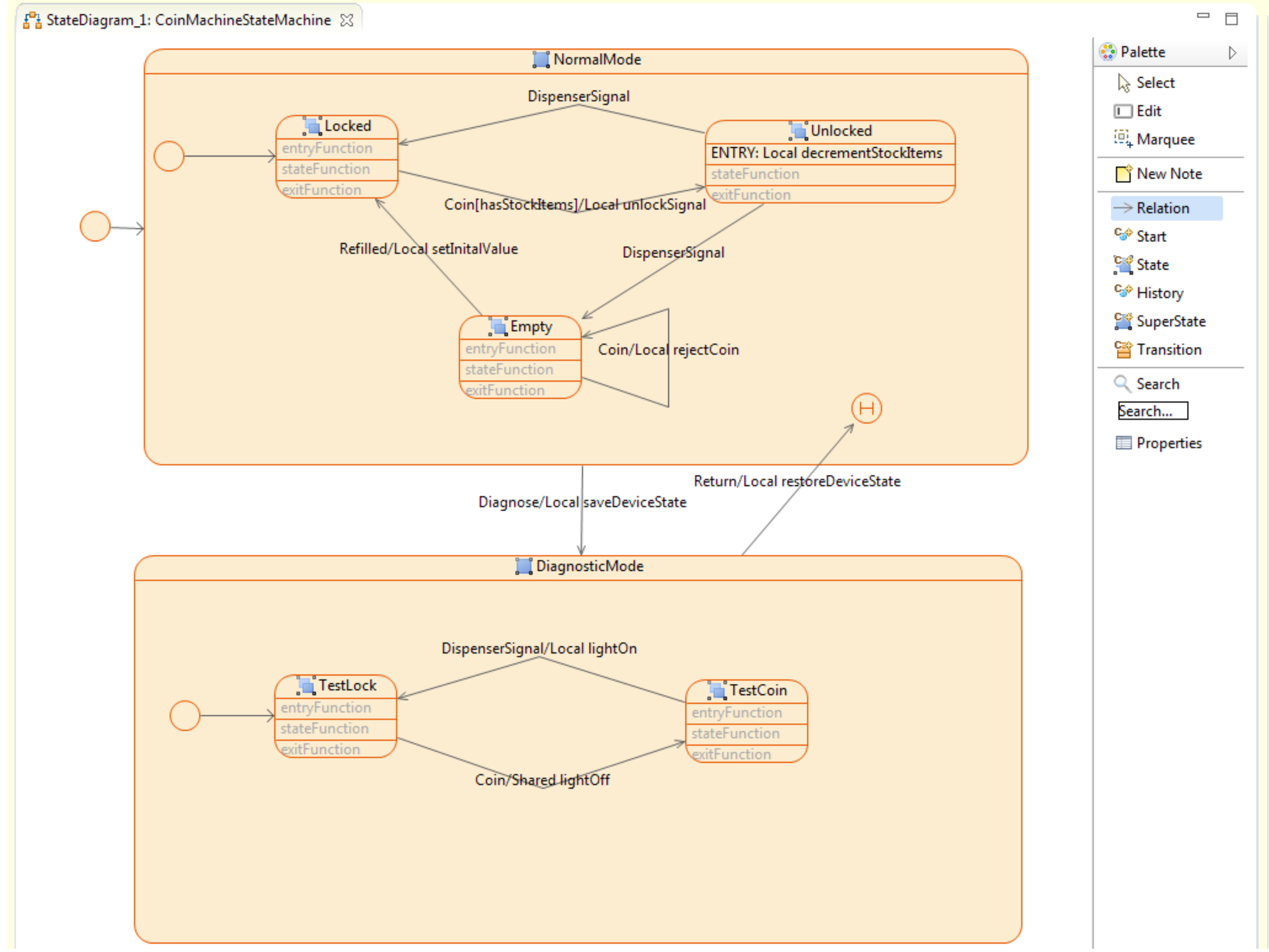
- Create a new **ModelFunctionImpl** named decrementStockItems as entryFunction of the state Unlocked
- Add an **Assignment** as **statement** with an **operand** of type **DecExpression**
- Add an **operand** of type **VariableFieldExpression** to the **DecExpression**
- Add a **fieldRef** of type **FieldRef** to the **VariableFieldExpression** and use stockItems as field

© 2010 actifsource GmbH, Switzerland - all rights reserved.



↳ Open the StateDiagram_1 in the **Diagram Editor** and check that all the actions are displayed correctly as shown above

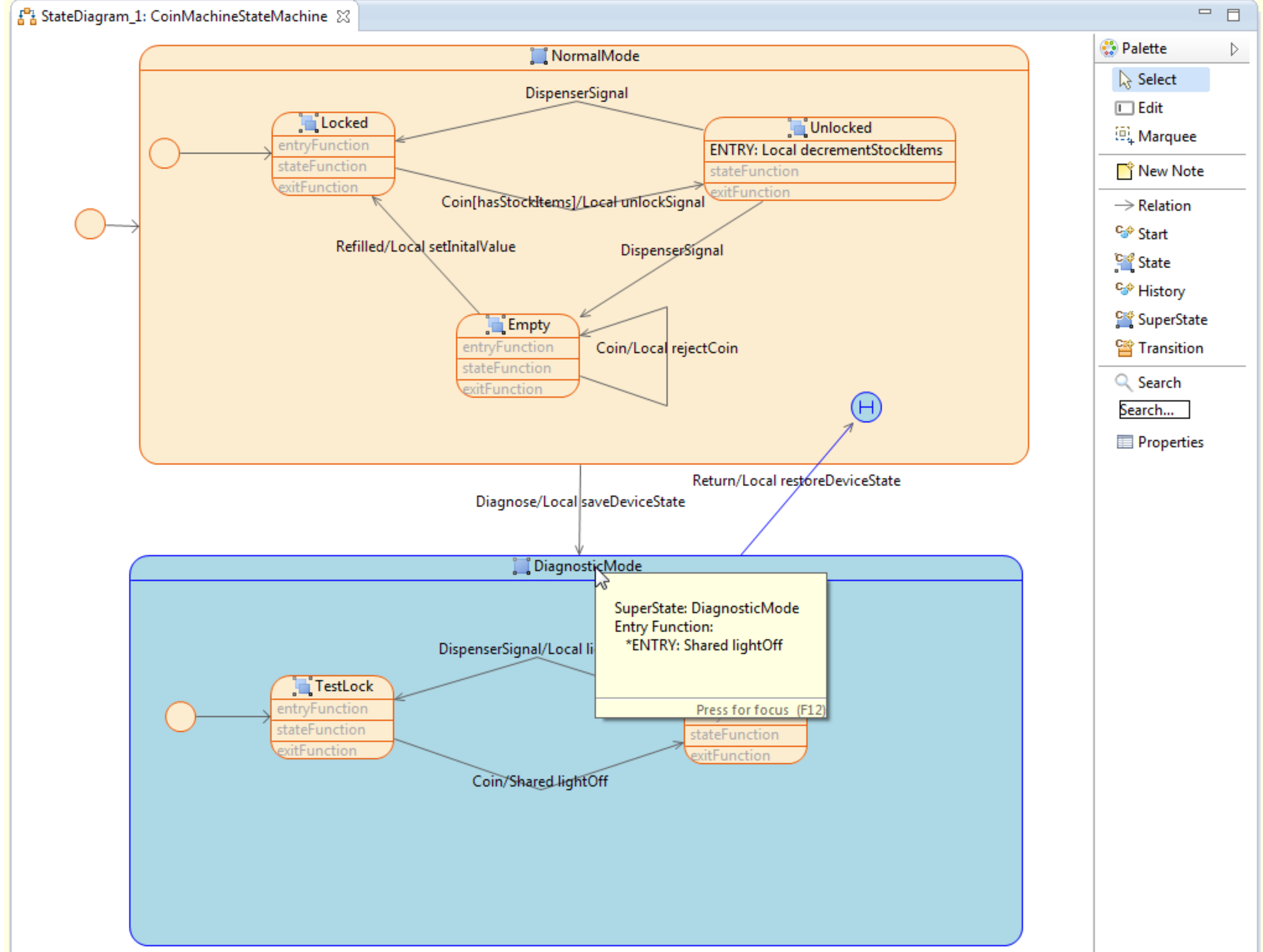
© 2010 actifsource GmbH, Switzerland - all rights reserved.



We add a transition that is triggered by an event Diagnose from the NormalMode to the DiagnosticMode (i.e., a technician should be able to switch to this diagnose state from any state in the normal mode). To save the state of the machine before switching modes, we introduce a history state:

- Add a history state by selecting History from the Palette
- Insert a transition triggered by a new event Return from the DiagnosticMode to the history state
- Create a transition triggered by a new event Diagnose from the NormalMode to the DiagnosticMode

© 2010 actifsource GmbH, Switzerland - all rights reserved.

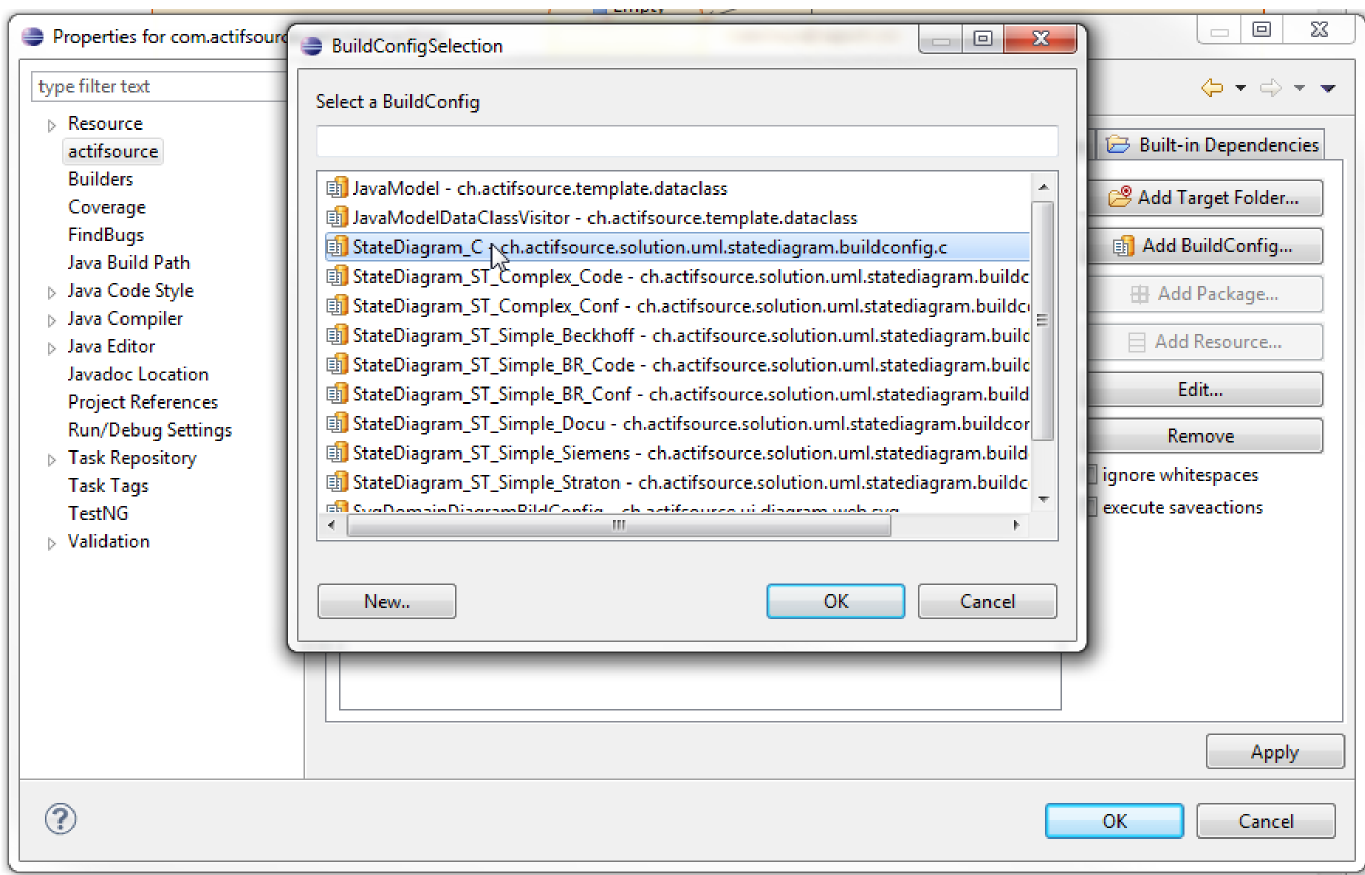


Since the state of the DiagnosticMode is not saved before returning to the NormalMode, the light should be switched off when entering the diagnostic mode:

- Open the CoinMachineStateMachine in the **Resource Editor**
- Add an **entryFunction** of type **SharedFunctionRef** to the DiagnosticMode and use lightOff as the **sharedFunction**
- The entry function is now displayed when selecting the DiagnosticMode in the **Diagram Editor** (see above)

© 2010 actifsource GmbH, Switzerland - all rights reserved.

Part III: Generate code from an UML State Machine



Finally, we want to generate code that implements the specified UML state machine:

- Right-click on the project com.actifsource.umlstatemachine in the **Project Explorer** and select Properties from the menu
- Select actifsource in the Properties dialog and go to the **Target Folder** tab
- Click on **Add Target Folder**, create a new folder called src, select this folder and click OK
- Select the folder src and click on **Add BuildConfig** and choose the build configuration **StateDiagram_C** from the dialog. Close both dialogs by clicking OK

© 2010 actifsource GmbH, Switzerland - all rights reserved.

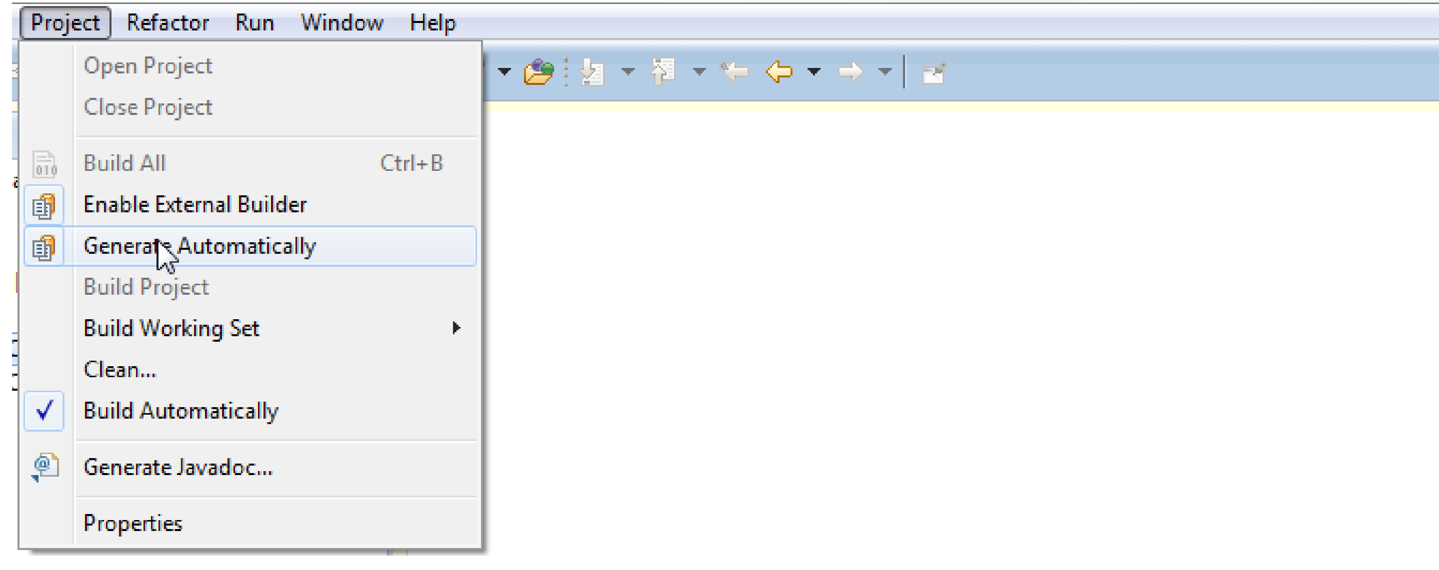
Project Explorer

- com.actifsource.umlstatemachine
  - asrc
  - Actifsource
  - JRE System Library [JavaSE-1.7]
  - src
    - Statemachine_CoinMachineStateMachine.c
    - Statemachine_CoinMachineStateMachine.h
  - src-gen

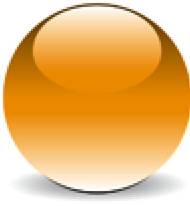
Statemachine_CoinMachineStateMachine.h | Statemachine_CoinMachineStateMachine.c

```
// (C) Copyright Actifsource GmbH  2012-2015

#ifndef Statemachine_CoinMachineStateMachine_H_
#define Statemachine_CoinMachineStateMachine_H_

// Header files from the environment

// Header files from external components

// Header files from own component

// Type and Forward declarations

/** Events for StatemachineBase CoinMachineStateMachine. */
typedef enum Event_CoinMachineStateMachine
{
  CoinMachineStateMachineEvent__Entry = -3,
  CoinMachineStateMachineEvent__State = -2,
  CoinMachineStateMachineEvent__Exit = -1,
  CoinMachineStateMachineEvent_Coin,
  CoinMachineStateMachineEvent_DispenserSignal,
  CoinMachineStateMachineEvent_Refilled,
  CoinMachineStateMachineEvent_Return,
```

- Open the folder src and make sure that the two files Statemachine_CoinMachineStateMachine.c and Statemachine_CoinMachineStateMachine.h have been generated and inspect the generated code
- ⓘ If the two files have not been generated, check that the option **Generate Automatically** is active. If not, the code can also be generated manually as follows: right-click on the project in the Project Explorer and choose **Generate Code** from the menu.



© 2010 actifsource GmbH, Switzerland - all rights reserved.



© 2010 actifsource GmbH, Switzerland - all rights reserved.