



# Motorola *WEAVR*

***UML 2.0 Weaver***  
*Add-In to Telelogic TAU G2*

## Programmer Tutorial

*by*

*Thomas Cottenier and Aswin van den Berg*

[thomas.cottenier@motorola.com](mailto:thomas.cottenier@motorola.com), [Aswin.vandenberg@motorola.com](mailto:Aswin.vandenberg@motorola.com)

(847) 538 37 39, (847) 538-2597

## 1. Introduction

**Note:** This tutorial assumes that the **WEAVR** AOM Add-In has been correctly installed. For installation instructions, consult the **WEAVR** Programmer's Guide.

This tutorial is based on a slightly modified version of the PingPong example from the Telelogic TAU G2 distribution. The different versions of PingPong are available in the **AOM\Tutorial\PingPong** folder. The Aspects used in the example are included in the **AOM\Tutorial\Aspects** folder.

This Section introduces the PingPong application.

### 1.1 The Ping Pong Application

The basic application is described as follows in Telelogic TAU G2 distribution:

*“The model used in the example contains a top level active class called 'Match', which in turn includes two active classes, 'Player1' and 'Player2'. 'Player1' and 'Player2' communicate with each other via the signals, 'Ping' and 'Pong', in an endless loop. No communication with the system environment takes place.”*

Figure 1 depicts the static structure of Ping Pong and Figure 2 presents the Architectural Diagram of the Match Active Class. The State Chart Diagrams of the State Machine Implementation of the Player1 and Player2 Active Classes are shown in Figure 3. The PingPong example shipped with the TAU G2 distribution has been slightly modified in order to highlight more of the features of the WEAVR AOM add-in. Player1 and Player2 instantiate an Object of the 'Hello' passive Class and call a 'hello' method on this Object. Also, the Signals that are exchanged between the two Active Classes have been modified to carry two Integer Parameters instead of one.

These modifications will allow us to illustrate particular features:

- Call Expression Action Joinpoints
- Start Transition Joinpoints
- Exposition of Action Joinpoint Arguments
- Control over the side-effects of Connectors

The next Session discusses how to run the model in the Model Verifier and shows some execution traces of the system.

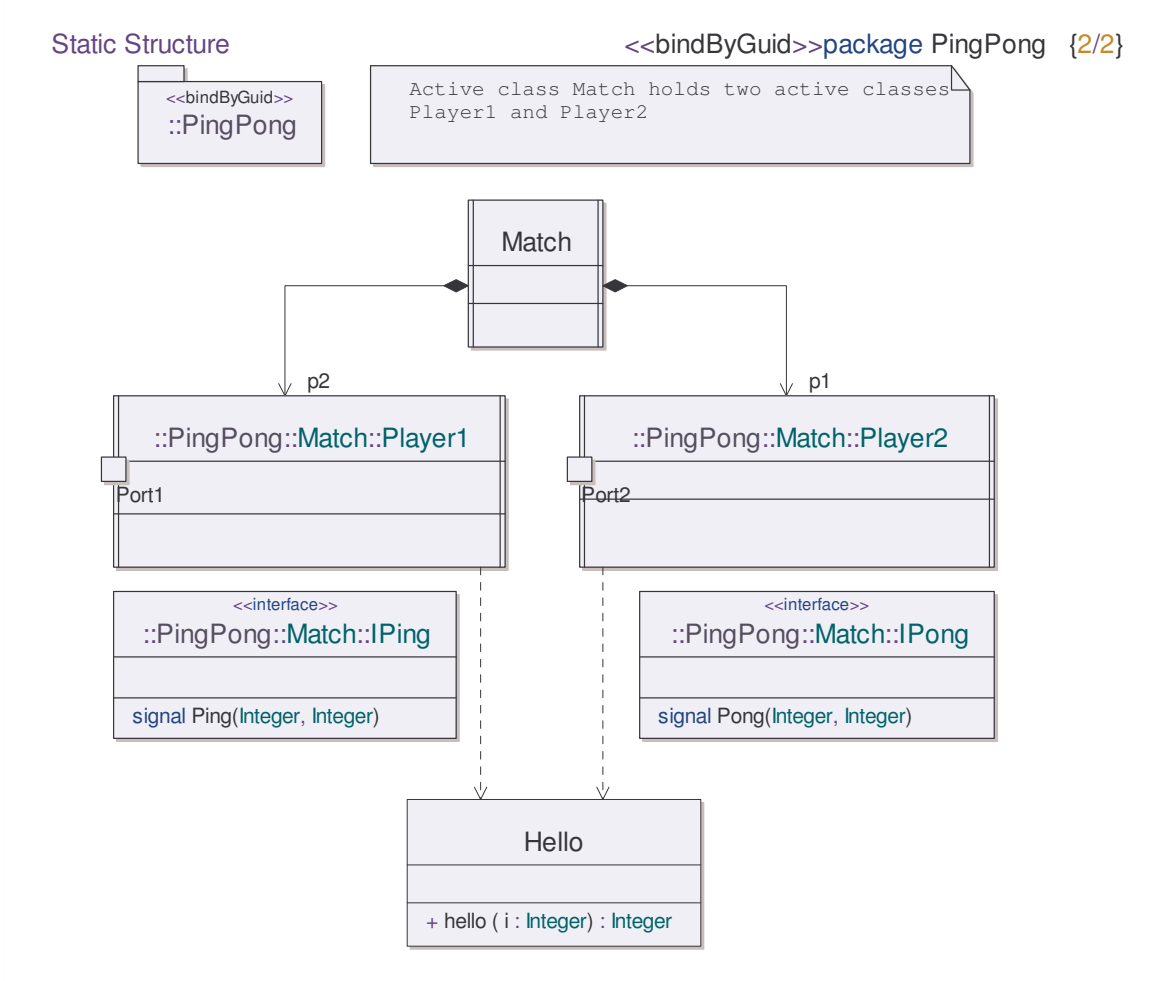


Figure 1. The static structure of PingPong

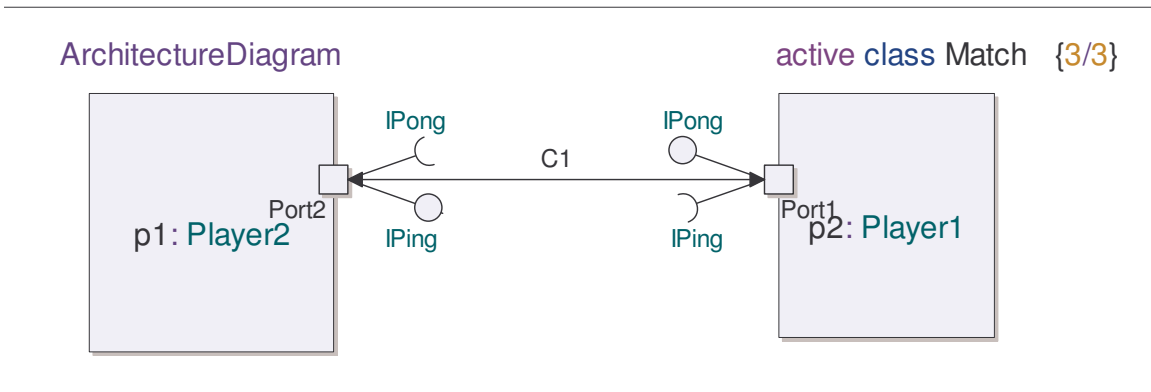
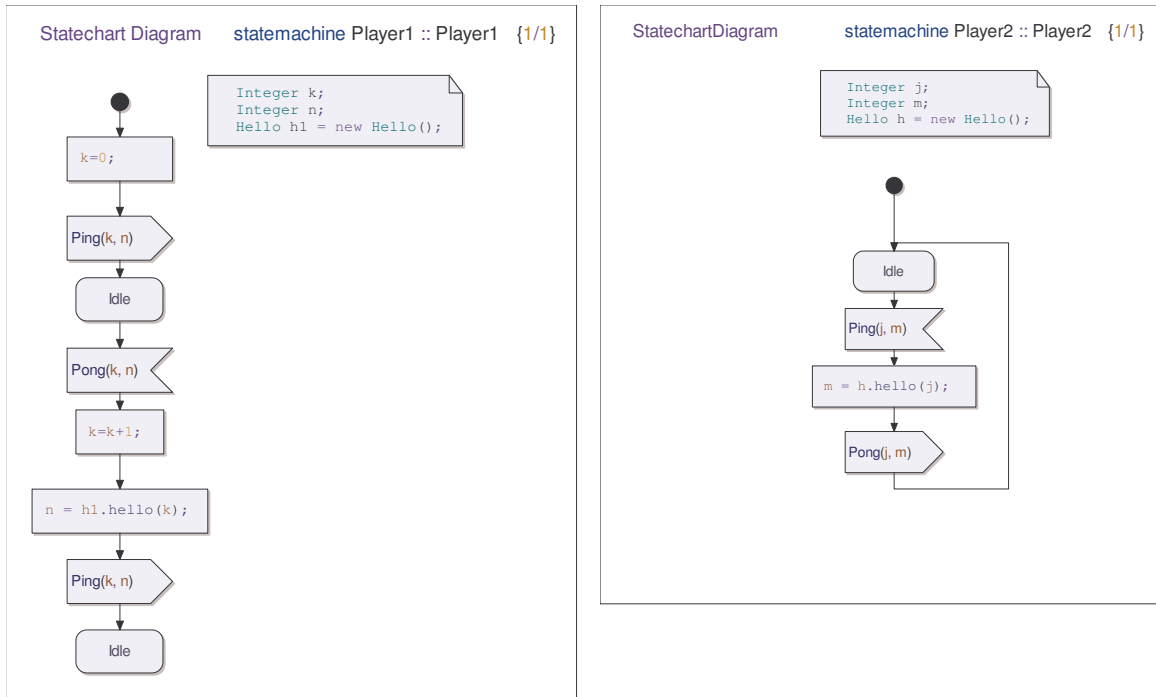
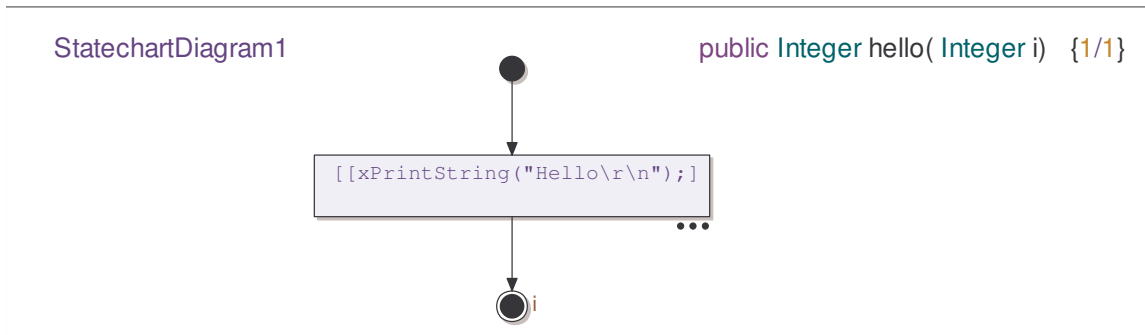


Figure 2. The architecture of PingPong



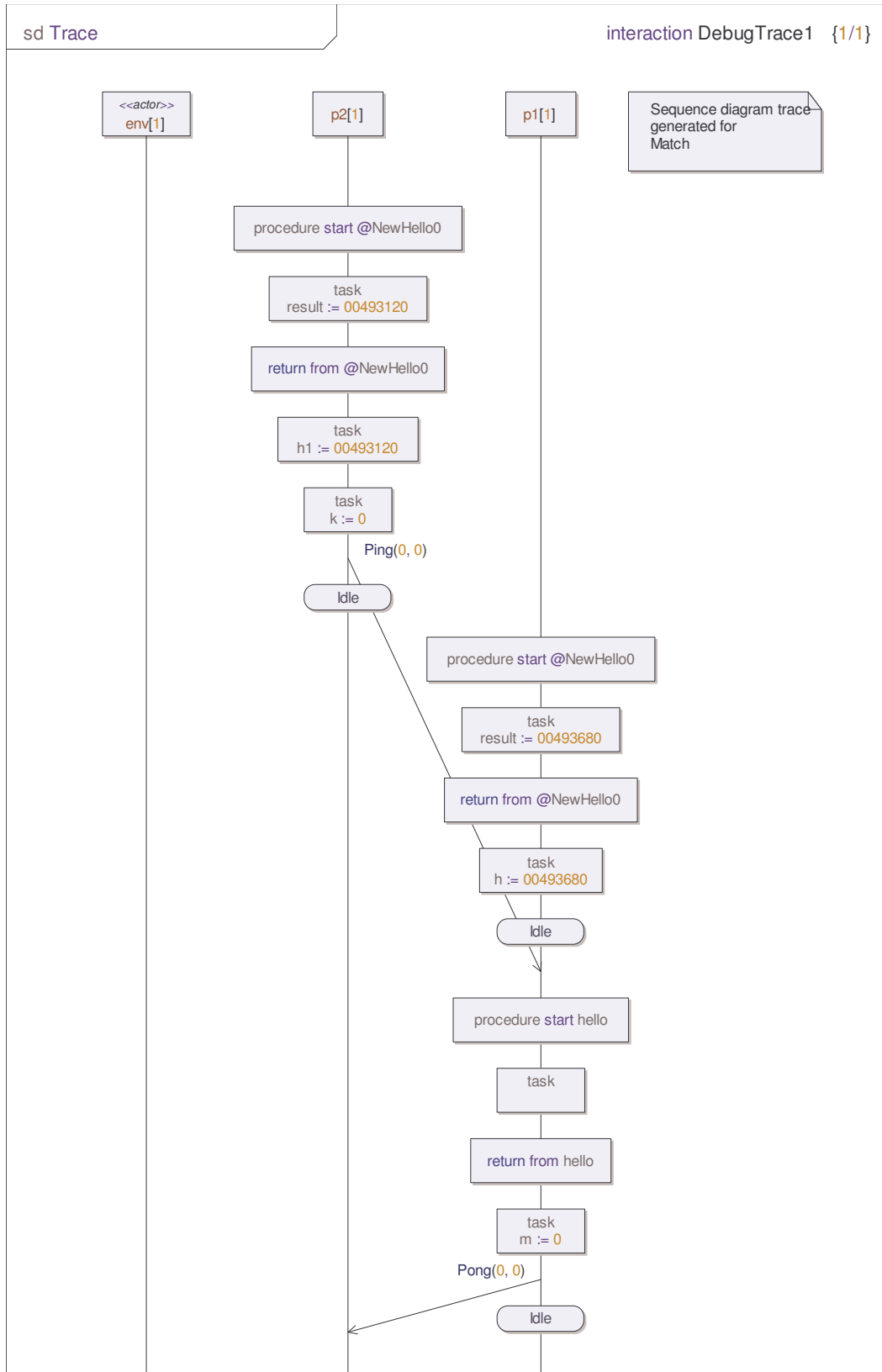
**Figure 3.** The State Machine Implementations of the Player1 and Player2 Active Classes



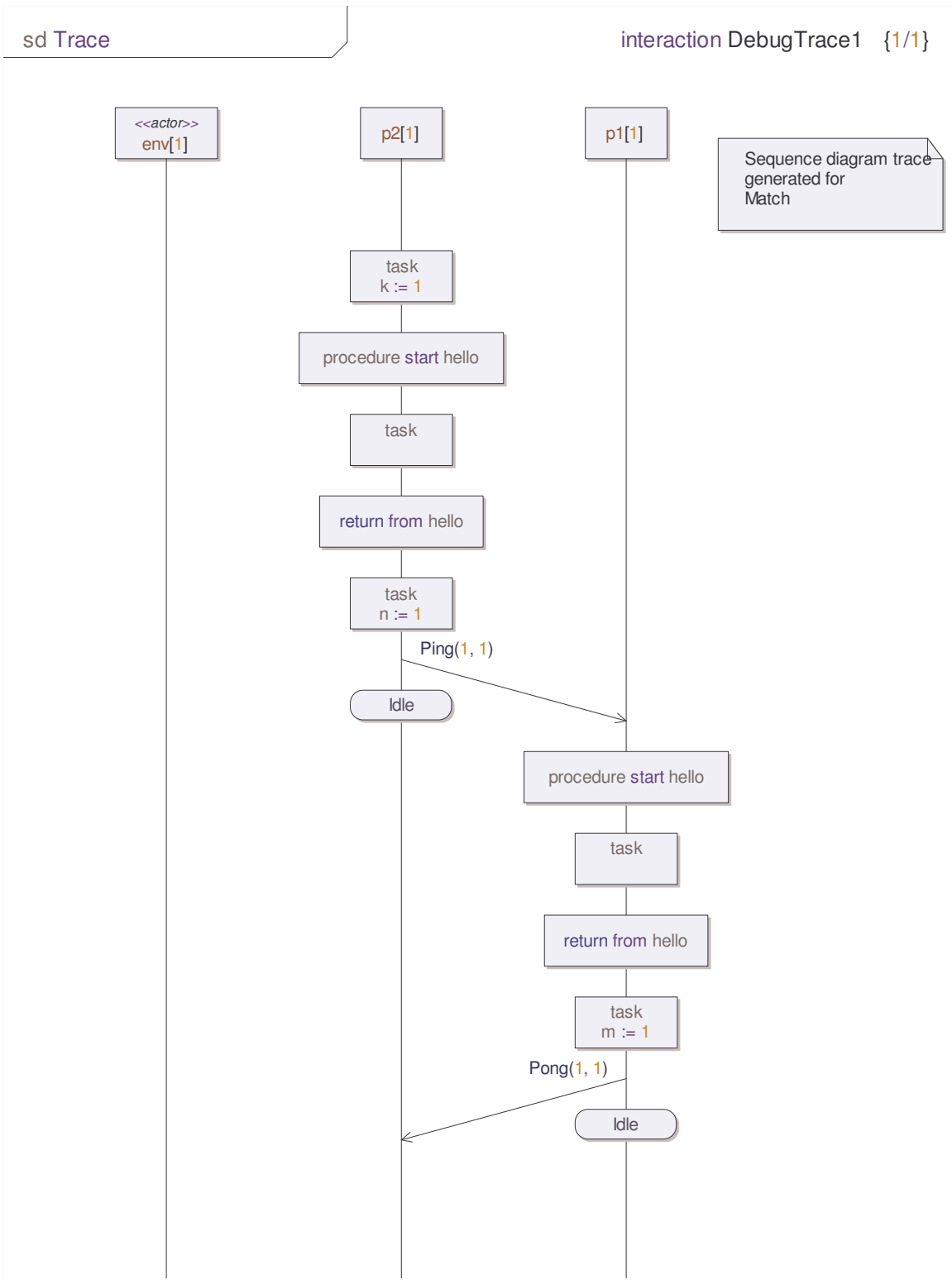
**Figure 4.** The State Machine Implementation of the hello Operation in the Hello Class

## 1.2 Running Ping Pong in the Model Verifier

1. Make a copy of the addins\AOM\Tutorial directory
2. Open the the PingPong.ttw workspace from the **Tutorial\PingPong\SimplePingPong** directory
3. In the Model View, browse to the 'Match' Active Class



**Figure 5.** Initialization of the Player1 and Player2 Active Classes in the Sequence Diagram generated by the Model Verifier



**Figure 6.** Signal exchange between the Player1 and Player2 Active Classes in the Sequence Diagram generated by the Model Verifier

4. Right-click on the Match Active Class and select 'New Artifact' for the Model Verifier
5. Right-click on the new Artifact and select Build, then select Launch
6. Active detailed Sequence Diagram tracing by entering `!U2::Debug start_msc 2` in the Model Verifier console
7. Run the Simulation, by pressing F5
8. The generated Sequence Diagram should look as displayed in Figures 5 and 6
9. Note that the output in the Model Verifier tab should display the 'Hello\r\n' output of the 'hello' operation, as shown in Listing 1.

```
*** TRANSITION START
*   Inst      : p1:1
*   State     : Idle
*   Trigger   : Ping
*   Sender    : p2:1
*   Now       : 0.0000
* OPERATION START hello
* TASK
Hello
* OPERATION RETURN hello
* ASSIGN m :=
* OUTPUT of Pong to p2:1
*** NEXTSTATE Idle
```

**Listing 1.** Model Verifier tab output generated by the Model Verifier

## 2 Getting Started: A Simple Tracing Aspect

This Section introduces the WEAVR Aspect-Oriented Modeling Add-In. By the end of this Section, you will know how to:

- Active the Add-In
- Define a simple Aspect
- Define a Pointcut Diagram
- Define a Connector Diagram
- Draw an Aspect Binding Diagram
- Deploy an Aspect
- Visualize the effects of an Aspect
- Weave an Aspect
- Run a woven model in the Model Verifier

## 2.1 Activating the Add-in

Activating the AOM add-in:

1. Make a copy of the addins\AOM\Tutorial directory if you haven't done that already.
2. Open the PingPong.ttw workspace from the **Tutorial/PingPong/SimplePingPong** directory.
3. Activate the **WEAVR** AOM add-in
4. Save the workspace
5. Close the project
6. Re-open the PingPong project

The Message Tab should display messages similar to Listing 2.

```
WEAVR>> Getting Model Access...
Information: Session in ::[LEEEEdTEEEwBn1EL0v7AIV]: TMI0759: Loading file
C:\Program Files\Telelogic\TAU_2.5\addins\AOM\Tutorial\PingPong\SimplePingPong\
PingPong.u2
WEAVR>> Loading resource
WEAVR>> Initialize WEAVR Resource
Add-in module AOM activated.
Add-in module CApplication activated.
Add-in module CppTypes activated.
Add-in module ModelVerifier activated.
Add-in module RTUtilities activated.
```

**Listing 2.** Message Tab output after loading the project.

This section shows how to define a simple Aspect, a simple Output Action Pointcut and how to bind a Connector to a Pointcut.

## 2.2 Defining an Aspect

We will define a simple tracing Aspect and apply it to the PingPong model.

1. Create a new Package and name it 'MyAspects'
2. Save the new Package in a new file by right-clicking on the Package and selecting 'Save in New File'. Save the file as MyAspects.u2 in the PingPong directory.
3. In the MyAspects Package, create a new Class and name it 'MyTracingAspect'

4. Apply the Aspect stereotype to the 'MyTracingAspect' class, by right-clicking on the class and selecting 'Stereotypes'. Check the 'AOMProfile::Aspect' checkbox

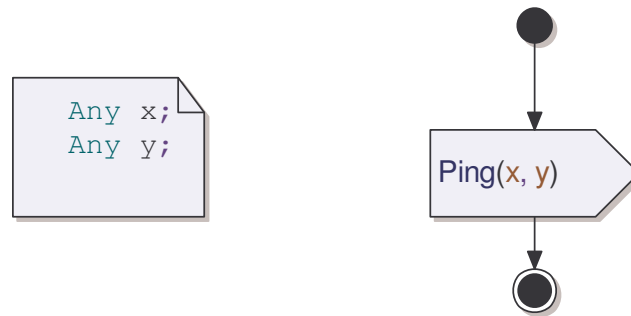
### 2.3 Defining a Pointcut

Our first Pointcuts (Output Action Pointcuts) will match the sending of the 'Ping' signal actions (Output Actions) in the 'Player1' Active Class and the 'Pong' Output Action in the 'Player2' Active Class .

1. Create a new Operation within MyTracingAspect and name it 'SendPing'
2. Apply the 'AOMProfile::Pointcut' stereotype to the new operation
3. Create a new State Machine Implementation for the Pointcut
4. In order to capture the sending of 'Ping', we need to create an Expression that matches the signature of that signal. Within the context of the State Machine Implementation, create a new Signal and name it 'Ping'
5. Apply the 'AOMProfile::Expression' stereotype to the new Signal
6. Add two Parameters to the new 'Ping' signal, and give them the type 'Any'
7. In order to capture the Output Action, we need to specify what kind of event is matched by the pointcut. Add a new State Chart Diagram to the State Machine Implementation
8. Within the State Machine Implementation, create two new Attributes. Name them 'x' and 'y' and give them the type Any.
9. Create a Start Transition which outputs the 'Ping' Expression, with 'x' and 'y' as Arguments and add a return symbol.
10. The State Chart Diagram should look like displayed in Figure 7.
11. In the Model View, copy the SendPing Pointcut, and paste it in the context of the MyTracingAspect Aspect
12. Rename the copied Pointcut as 'SendPong'
13. Browse to the 'Ping' Expression in the context of 'SendPong'
14. Rename the 'Ping' Expression as 'Pong'
15. The TracingAspect should now contain two Output Action Pointcuts: SendPing and SendPong

StatechartDiagram1

<<Pointcut>> void SendPing() {1/1}



**Figure 7.** The PingSignal Pointcut in the MyTracingAspect Aspect

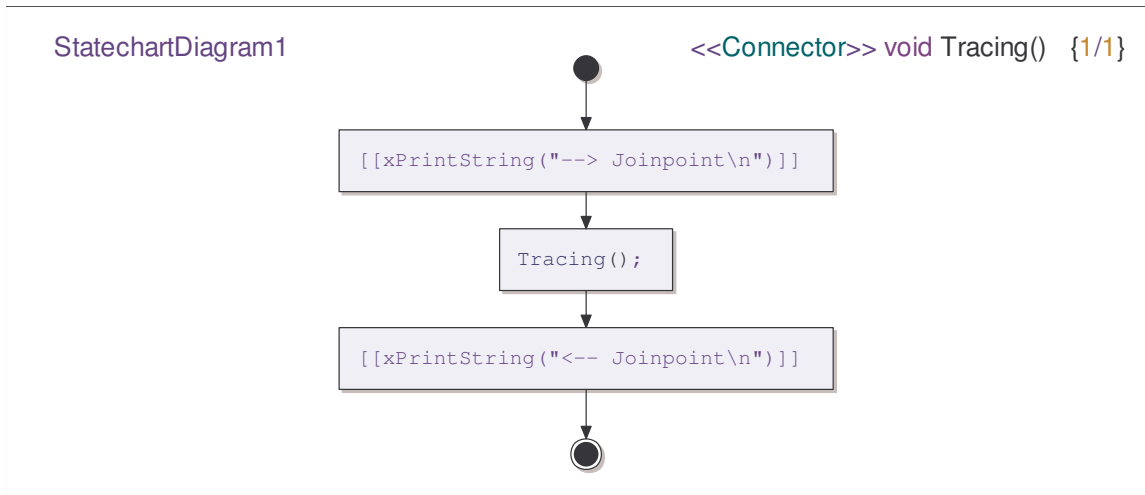
## 2.4 Defining a Connector

Our first Connector will trace the events matched by the SendPing and SendPong Pointcuts.

1. Create a new Operation within MyTracingAspect and name it 'Tracing'
2. Apply the 'AOMProfile::Connector' stereotype to the new operation
3. Create a new State Machine Implementation for the Pointcut
4. Create a new State Chart Diagram for the State Machine Implementation
5. Create a Start Transition, and add three Task Symbols, and a return symbol
6. Add a call to the Connector in the second Task Symbol
7. Apply the 'AOMProfile::Joinpoint' stereotype to the second Task Symbol. This task symbol represents the event matched by the Pointcut (a Joinpoint)

**Note:** Step 7 is very important. Without this stereotype, the connector will not invoke the original behavior matched by the pointcut (In this case, the Output Actions will not execute).

8. Add tracing statements to the first and the third task symbols. Use informal fragments to print out a string in the Model Verifier console. (We will see how to use a custom tracing method in Section X)
9. The Connector Diagram should look as illustrated in Figure 8

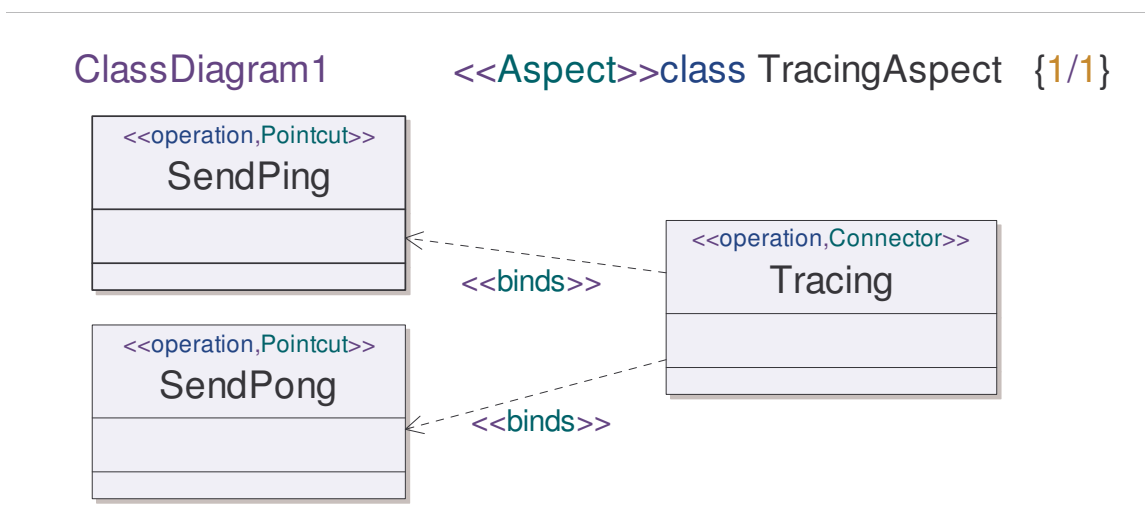


**Figure 8.** The Tracing Connector in the MyTracingAspect Aspect

## 2.5 Binding the Connector to the Pointcut

This step will bind the Tracing behavior to the Ping Output Action.

1. Create a new Class Diagram in the context of the Aspect.
2. Drag and Drop the SendPing and SendPong Pointcuts and the Connector from the Model View to the new Class Diagram
3. Create Dependencies from the Connector to the two Pointcuts
4. Apply the 'AOMProfile::binds' stereotype to the new Dependencies
5. The Binding Diagram should look like displayed in Figure 9



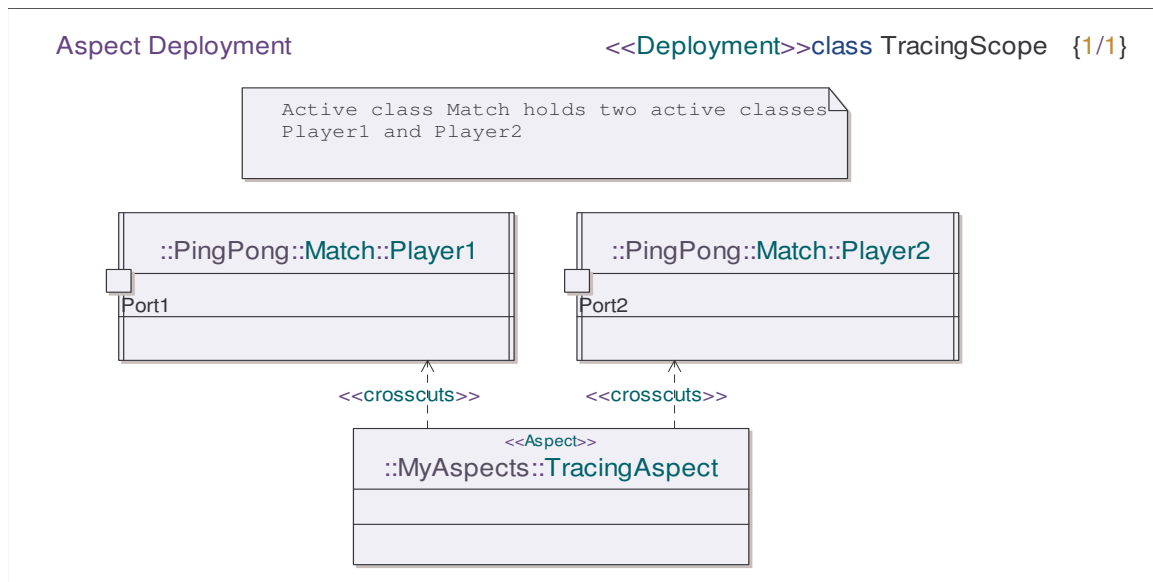
**Figure 9.** The Binding Diagram of the Tracing Aspect

## 2.6 Deploying the Aspect

This step specifies the scope of the Aspect.

1. Create a new Class in the context of the 'MyAspects' Package and name it 'TracingScope'
2. Apply the 'AOMProfile::Deployment' stereotype to the TracingScope Class
3. In the Model View, copy the 'Static Structure' Class Diagram from the 'Match' Active Class and paste it in the TracingScope Class. Rename the Class Diagram 'Aspect Deployment'
4. Remove the two interface boxes from the diagram.
5. Drag and drop the TracingAspect from the Model View in the new Class Diagram
6. Create Dependencies from the Aspect to the two Active Classes
7. Apply the 'AOMProfile::crosscuts' stereotype to the two Dependencies
8. The TracingScope Deployment Diagram for the TracingAspect should look as displayed in Figure 10.

**Note:** A deployment diagram that includes the Aspect needs to be defined for the Aspect to be taken into account. Yet, a specific scope does not need to be defined. If no crosscut dependencies are specified, the Aspects represented in the Deployment Diagram are applied to the complete system.



**Figure 10.** The TracingScope Deployment Diagram for the TracingAspect Aspect

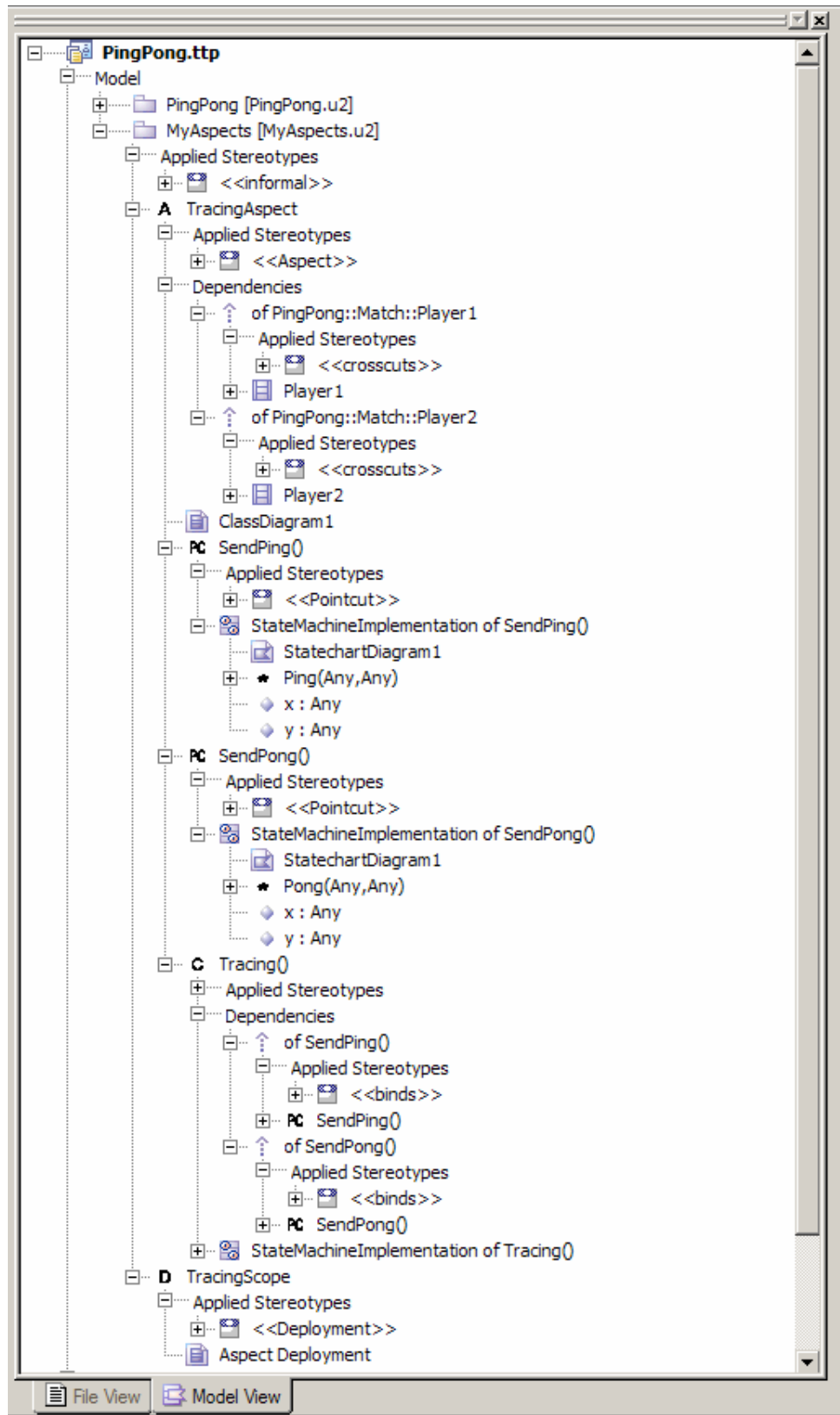


Figure 11. TracingAspect Aspect in the Model View


In the Model View, the complete structure of the Aspect and its deployment specification should look like shown in Figure 11.

## 2.7 Visualizing the effects of the Aspect

This step shows how to apply the aspect to the PingPong model and how to visualize its effects on the model.


1. Save the workspace

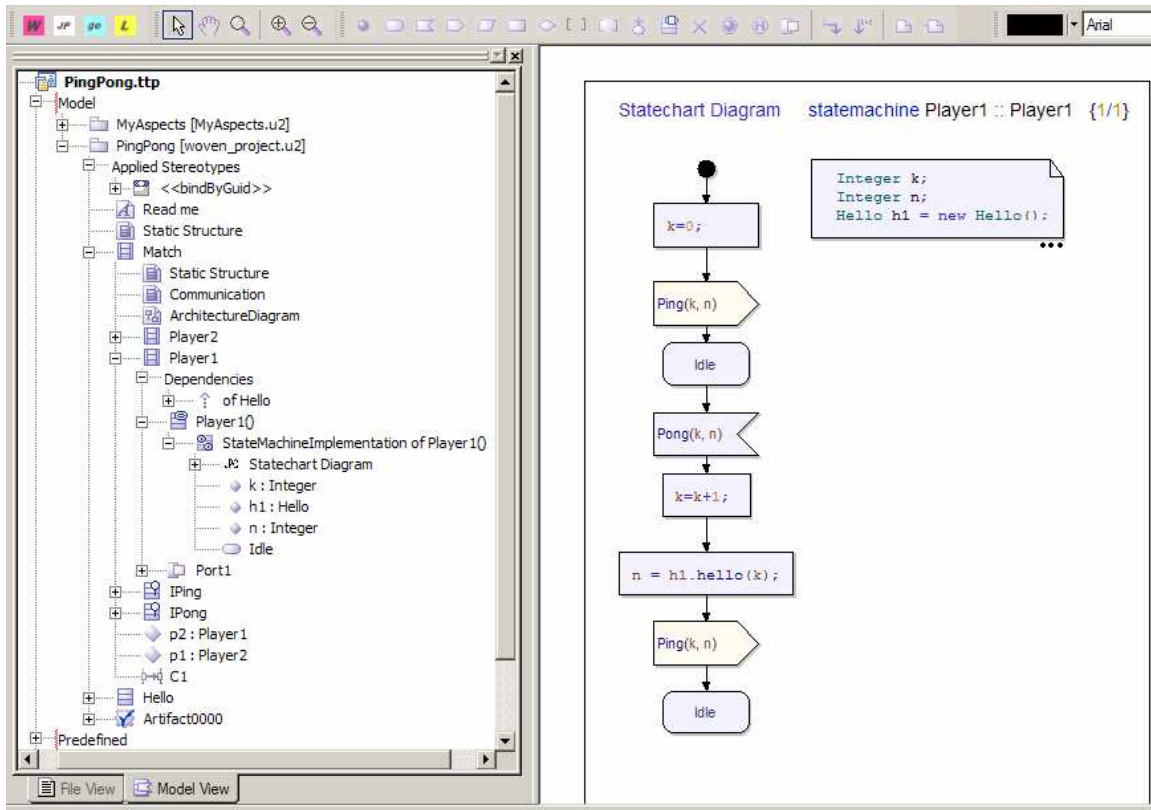
**Note:** This step is very important. If you do not save your workspace, all changes to the Model and to the Aspects since they were last saved will be lost. The add-in creates a copy of the model in its current state and loads it in the woven\_project.ttw workspace. The current workspace is thrown away.

2. Enter the weaving mode by clicking the ‘Enter WEAVR’ mode button (). The add-in should output the messages listed in Listing 3 in the Message Tab

```
WEAVR>> Entering WEAVR mode...
WEAVR>> loading PingPong...
WEAVR>> loading MyAspects...
WEAVR>> Remove original entity PingPong
WEAVR>> Entered WEAVR mode...
```

**Listing 3.** Message Tab output after entering the WEAVR mode.

3. Click the ‘Show Joinpoint’ button (). The add-in should output the messages listed in Listing 3 in the Message Tab
4. Browse to the Player1 State Machine Implementation. The State Chart Diagram and the model view should look as displayed in Figure 12. Note that the PingPong project is now contained in the woven\_project.u2 resource. Also, note that the State Chart Diagram is annotated with the “jpContext’ stereotype. Also, the presentation elements of the points in the model matching the Pointcut have been highlighted in a different color.



**Figure 12.** Model View after entering the Joinpoint visualization mode

5. Double Click on one of the highlighted Signal Output symbol. A new diagram appears in the Diagram View, as shown in Figure 13. This diagram represents an instantiation of the Connector Diagram in the context of the Output Action matched by the SendSignal Pointcut. The 'x' and 'y' Attributes have been replaced by context-specific bindings to the Parameters passed to the signal, 'binding\_a' and 'binding\_b'.

**Note:** If the Connector Instance Diagram does not appears, right-click on the symbol and follow the Outgoing Link hyperlink.

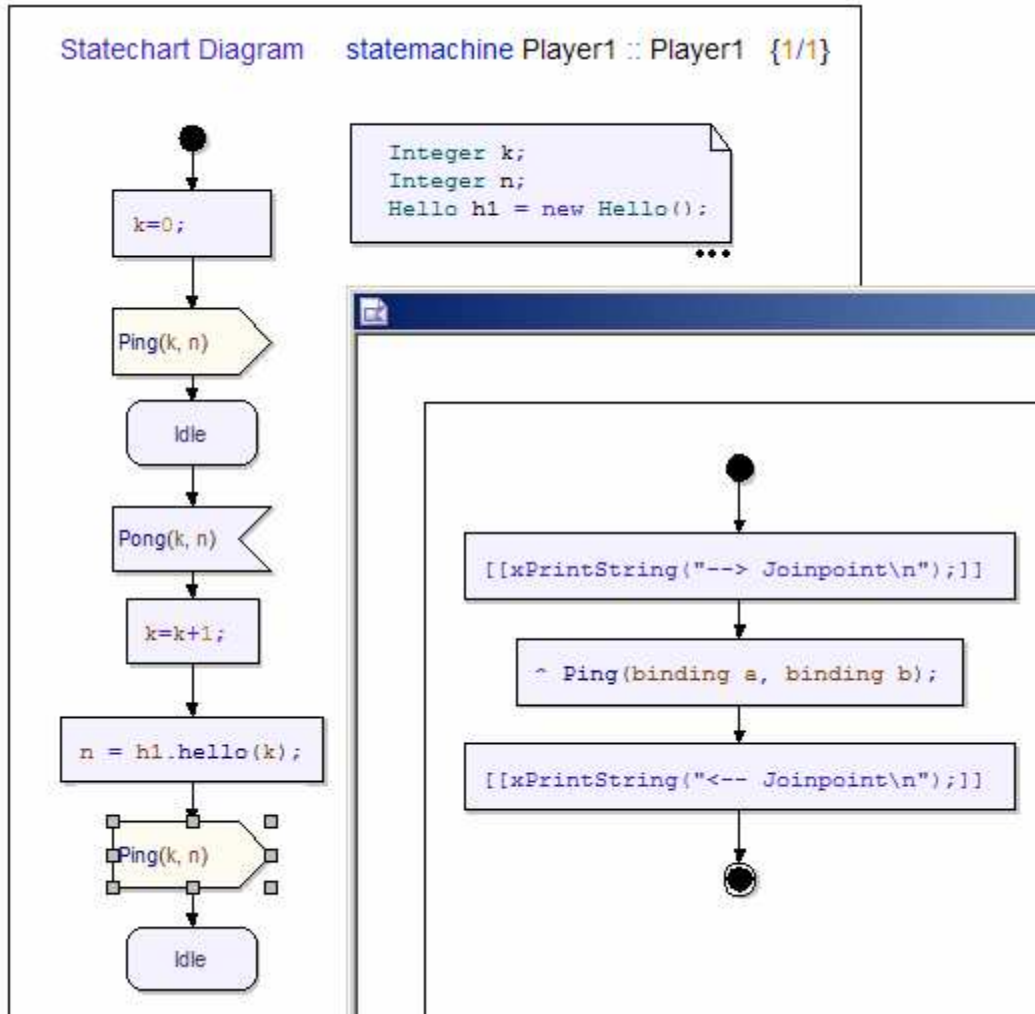
The Connector instance represents a wrapper around the Joinpoint it is bound to. It displays the actual effects that the Aspect has on the Joinpoint. Also note that neither the Model View, nor the State Chart Diagram shown in Figure 6 have been modified, except that stereotypes have been added to the model elements captured by the Aspect Pointcut.

```

WEAVR>> Found Deployment Diagram TracingScope
WEAVR>> Parsing Deployment....
WEAVR>> TracingAspect applied to Player1 Player2
WEAVR>> =====
WEAVR>> Aspect TracingAspect
WEAVR>> =====
WEAVR>> POINTCUT :
WEAVR>> Name : SendPong
WEAVR>> Type : OutputAction
WEAVR>> Pointcut Types : OutputAction
WEAVR>> Expression Name : Pong
WEAVR>> Expression Scope :
WEAVR>> Expression param : Any : Parameter1
WEAVR>> Expression param : Any : Parameter2
WEAVR>> =====
WEAVR>> POINTCUT :
WEAVR>> Name : SendPing
WEAVR>> Type : OutputAction
WEAVR>> Pointcut Types : OutputAction
WEAVR>> Expression Name : Ping
WEAVR>> Expression Scope :
WEAVR>> Expression param : Any : Parameter1
WEAVR>> Expression param : Any : Parameter2
WEAVR>> =====
WEAVR>> CONNECTOR :
WEAVR>> Name : Tracing
WEAVR>> =====
WEAVR>> Grep Joinpoints
WEAVR>> Output Action Joinpoint Ping matches pointcuts: SendPing
WEAVR>> Output Action Joinpoint Ping matches pointcuts: SendPing
WEAVR>> Output Action Joinpoint Pong matches pointcuts: SendPong
WEAVR>> ...
WEAVR>> Color Joinpoints
WEAVR>> ...
WEAVR>> Annotate JpContext
WEAVR>> ...
WEAVR>> Build Connectors
WEAVR>> ...
WEAVR>> Processed aspect TracingAspect
WEAVR>> Total Joinpoints: 3

```

**Listing 4.** Message Tab output after clicking the ‘Show Joinpoints’ button



**Figure 13.** Player1 State Machine Implementation and Connector Diagram Instance generated for the second Ping Output Action

## 2.8 Weaving the Aspect

1. Perform Model Weaving by clicking the 'Weave Model' button (**go**)
2. The Message Tab should display the messages listed in Listing 5.

```

WEAVR>> Bind Connectors
WEAVR>> ...
WEAVR>> Total Joinspace: 3

```

**Listing 5.** Message Tab output after clicking the 'Weave Model' button

At this point, the model has been woven. Note that all the presentation elements of the model affected by Model Weaving have been deleted.

3. Close the workspace

## 2.9 Running the Woven Model in the Model Verifier

1. Open the woven workspace `woven_project.ttw` which has been created in the same directory as the PingPong workspace.
2. In the Model View, browse to the State Machine Implementation of Player1
3. Right-click on the State Machine Implementation and select New State Chart Diagram
4. Drag and drop the State Machine Implementation from the Model View to the new State Chart Diagram.
5. The generated State Chart Diagram should look like the diagram displayed in Figure 14
6. Browse to the 'Match' Active Class in the Model View and right-click on it. Create a new Artifact in the Model Verifier.
7. Build the created Artifact and launch the Model Verifier
8. Run the Simulation, by pressing F5
9. The output in the Model Verifier Tab should contain the extra tracing statement as shown in Listing 5

```
* ASSIGN k :=
* OPERATION START Ping_SendSignal__QrmN4Ia9Ab5LPDWZwLrKknXV#Tracing_0
* TASK
--> Joinpoint
* OUTPUT of Ping to p2:1
* TASK
<-- Joinpoint
* OPERATION RETURN
Ping_SendSignal__QrmN4Ia9Ab5LPDWZwLrKknXV#Tracing_0
*** NEXTSTATE Idle
```

**Listing 5.** Model Verifier Tab output when running the `woven_project` workspace

StatechartDiagram1

statemachine Player1 :: Player1 {1/1}

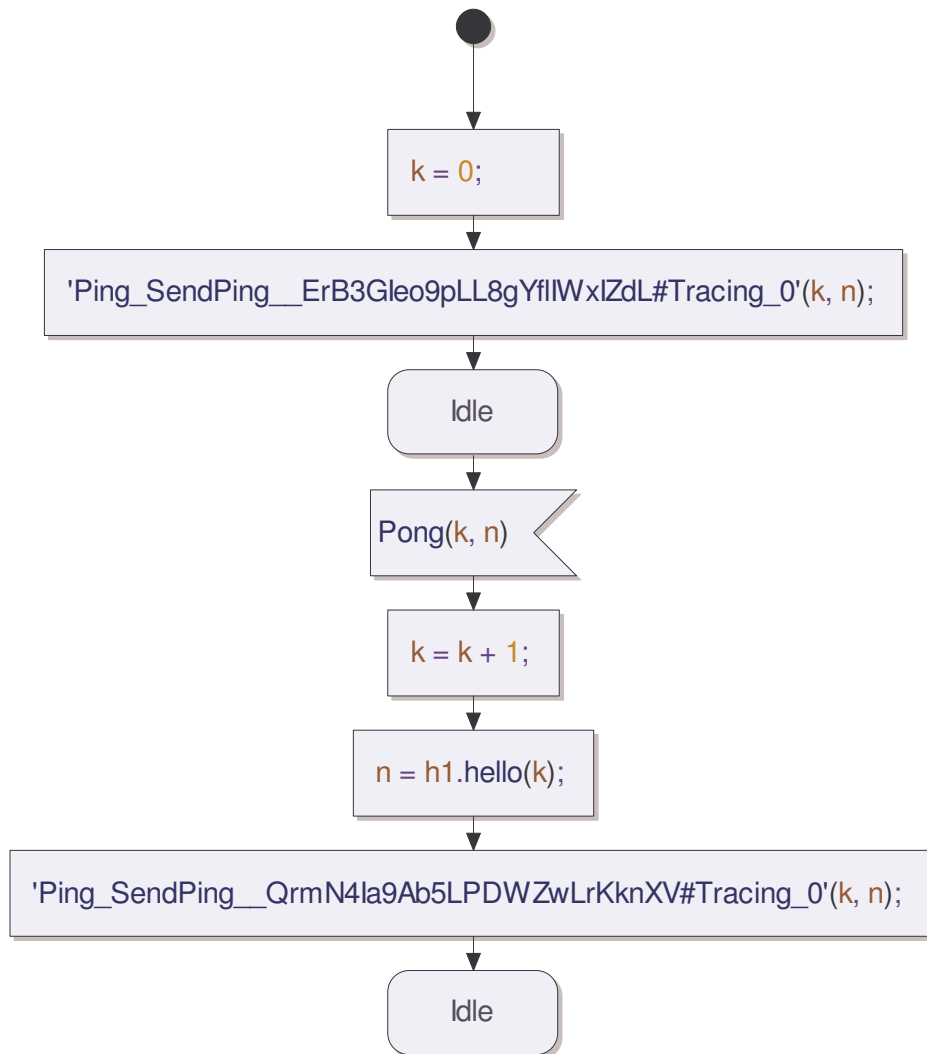


Figure 14. Generated State Chart Diagram in the woven\_project.ttw workspace

### 3 Types of Pointcuts and Pointcut Composition

In this section, we will extend the Tracing Aspect to trace different kind of events. The different types of Pointcuts are presented. We will also look into Pointcut composition operators.

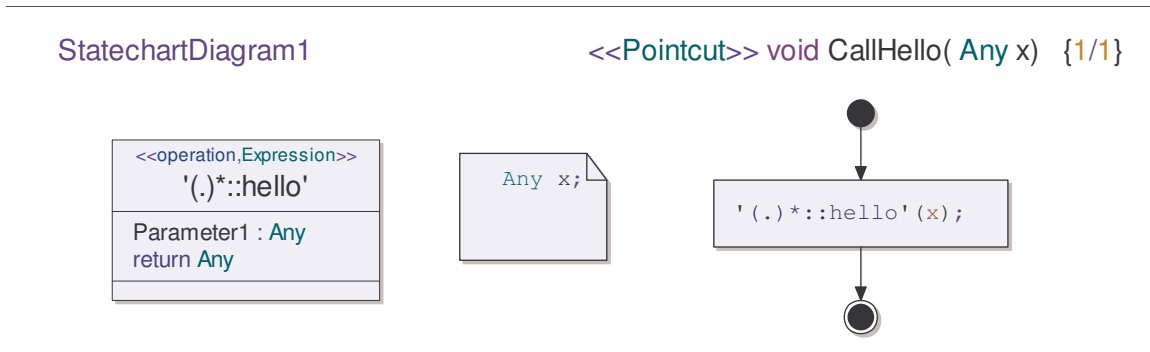
#### 3.1 Defining Action Pointcuts

Two types of Action Pointcuts are supported: Output Action Pointcuts and Call Expression Action Pointcuts.

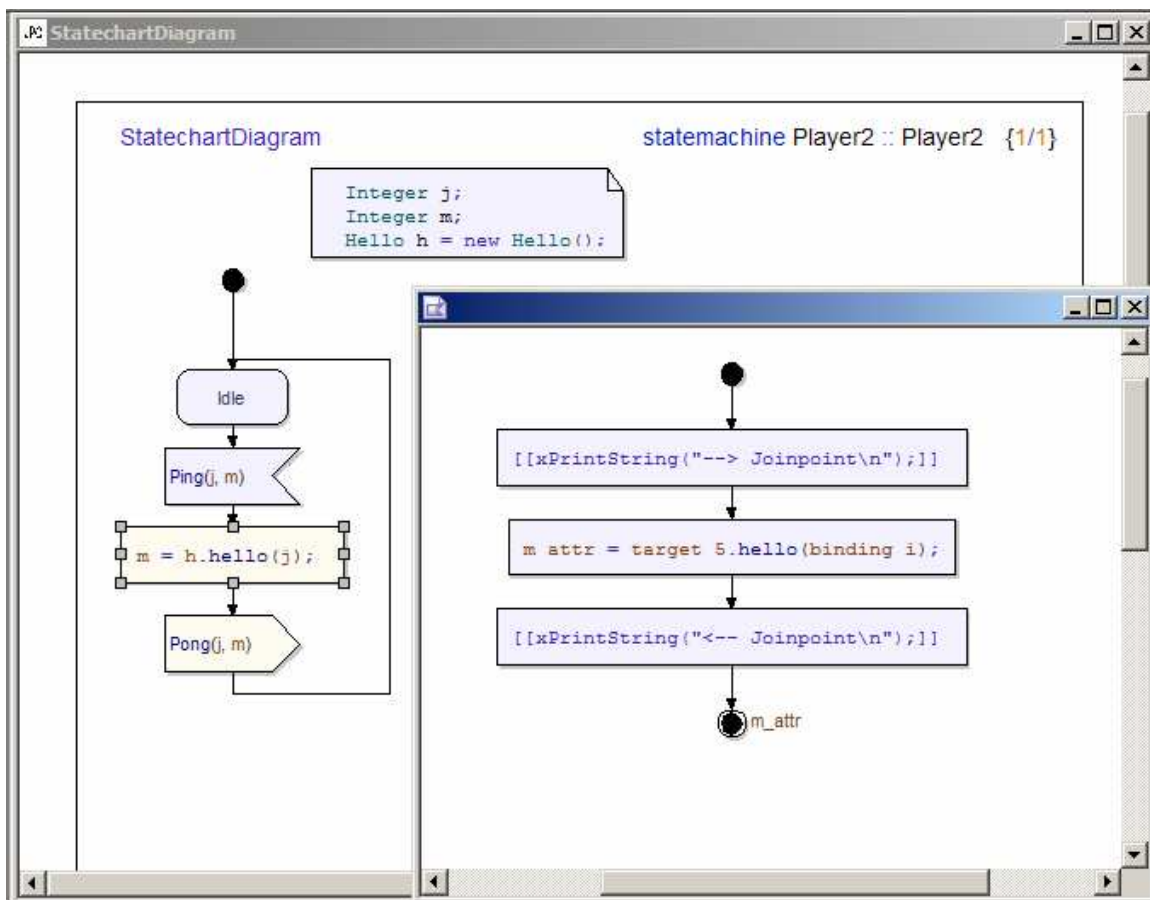
Section 2.2 showed how to define Output Action Pointcuts. We will now write a simple Call Expression Action Pointcut.

1. In the Model View, copy and paste the SendPing Pointcut in the context of the TracingAspect.
2. Rename the new Pointcut 'CallHello'
3. In the new Pointcut, delete the Ping Expression
4. In the context of the Pointcut State Machine Implementation, delete the 'y' Attribute
5. In the context of the Pointcut State Machine Implementation, create a new Operation, and name it '(.)\*::hello'. The wildcard indicates that we are capturing all the 'hello' methods defined in any class of the system
6. Apply the 'AOMProfile::Expression' stereotype to the new Operation
7. Add a Parameter to the new Expression, and give it the type 'Any'
8. Add a return Parameter to the new Expression, and give it the type 'Any'
9. In the State Chart Diagram of the CallHello Pointcut, delete the Ping Output Symbol
10. Create a new Task Symbol, and place it into the execution of the Start Transition
11. Create a Call Expression to '(.)\*::hello'(x) in the Task Symbol, and pass it the Attribute 'x' as an Argument

**Note:** The regular expression symbols such as '\*', ')' and '(' are a special symbol in TAU. In order to define wildcard Expressions, use quotes around the name of the Expressions, such as in '(.)\*::hello'(x)



**Figure 15.** The CallHello Pointcut in the MyTracingAspect Aspect



**Figure 16.** Player2 State Machine Implementation and Connector Diagram Instance generated for the 'hello' Call Expression Action Joinpoint.

12. The CallHello Pointcut State Chart Diagram should look as displayed in Figure 15
13. Bind the Tracing Connector to the new Pointcut in the Binding Diagram
14. Save the workspace
15. Enter the WEAVR mode and click the 'Show Joinpoints' button

16. The Message Tab should indicate that 5 Joinpoints have been found
17. The State Machine Diagram for Player2, and the Connector Instance for the 'hello' Call Expression Action Joinpoint should now look as displayed in Figure 16
18. Weave the model, close the workspace, open the woven\_project workspace and launch the simulation

### **3.2 Transition Pointcuts**

Two types of Transition Pointcuts are supported: Start Transition Pointcuts and Triggered Transition Pointcuts. Start Transition Pointcuts match the execution of a State Machine Implementation. Triggered Transition Pointcuts match Transitions that are triggered by the reception of a Signal or the expiration of a Timer.

#### **3.2.1 Start Transition Pointcuts**

We will define a Start Transition Pointcut that matches the execution of the Hello::hello() method.

1. In the Model View, copy and paste the CallHello Pointcut in the context of the TracingAspect
2. Rename the Pointcut as 'ExecuteHello'
3. Delete the State Machine Implementation of the new Pointcut
4. Create a new State Machine Implementation and a new State Chart Diagram for the ExecuteHello Pointcut
5. In the Model View, copy the Expression of the CallHello Pointcut and paste it in the context of the State Machine Implementation of the ExecuteHello Pointcut.
6. Rename the Parameter of the Pointcut Expression as 'x'
7. Drag and drop the Expression of the ExecuteHello Pointcut from the Model View into the new State Chart Diagram
8. Create a new State Chart Diagram in the context of the Expression of the ExecuteHello Pointcut
9. Create a new Start Transition in the State Chart Diagram
10. Add an Attribute 'a' of type Any to the Expression State Machine Implementation

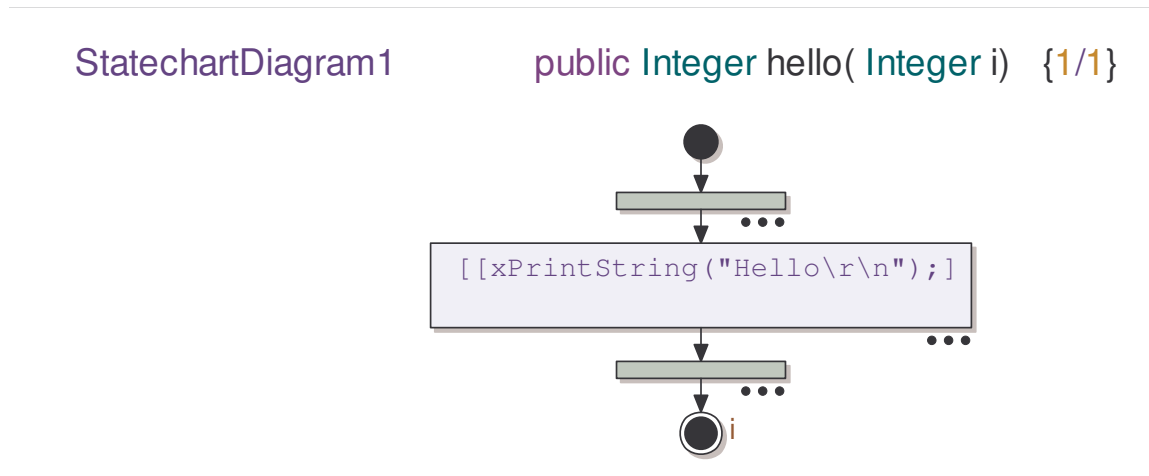


```
WEAVR>> Start Transition Joinpoint PingPong::Hello::hello matches pointcuts:  
ExecuteHello  
WEAVR>> Output Action Joinpoint Ping matches pointcuts: SendPing  
WEAVR>> CallExpr Action Joinpoint PingPong::Hello::hello matches pointcuts:  
CallHello  
WEAVR>> Output Action Joinpoint Ping matches pointcuts: SendPing  
WEAVR>> CallExpr Action Joinpoint PingPong::Hello::hello matches pointcuts:  
CallHello  
WEAVR>> Output Action Joinpoint Pong matches pointcuts: SendPong
```

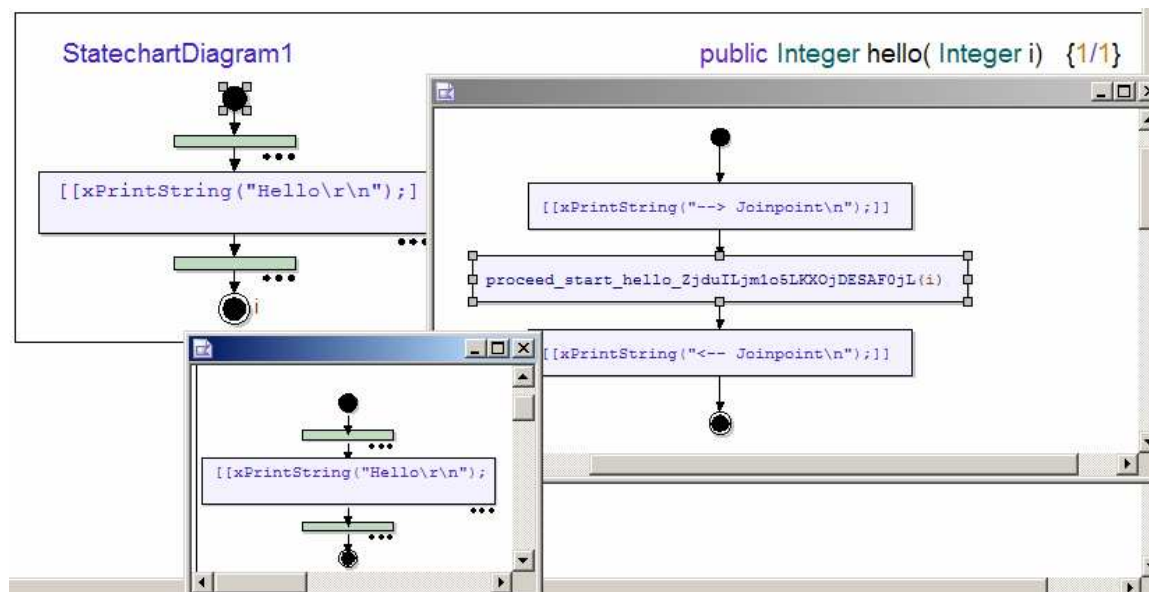
**Listing 6.** Message tab output for the SendPing, SendPong, CallHello and ExecuteHello Pointcuts

12. The Pointcut Diagrams for the Start Transition Pointcut should now look as displayed in Figure 17. In the Model View, the ExecuteHello Pointcut should look as displayed in Figure 18
13. Add the ExecuteHello Pointcut to the Binding Diagram and add a binding Dependency from the Connector to the new Pointcut
14. Save the workspace
15. Enter the WEAVR mode and click the ‘Show Joinpoints’ button
16. The Message Tab should indicate that 6 Joinpoints have been found. The new Joinpoint is identified at the first line of Listing 6
17. Browse to the State Machine implementation of the Hello::hello(Integer): Integer Operation. The State Chart Diagrams should look as displayed in Figure 19. The State Chart Diagram has been annotated with green marks that delimit the entry point and the exit points of the Start Transition (in this case, the complete Transition). Also, note that a new operation has been generated in the context of the State Machine Implementation.
18. Click on either the Start Symbol or one of the green marks. A Connector Instance for the Start Transition appears, as displayed in Figure 19
19. In the Connector Instance Diagram, press ctrl and click on the call to the generated operation. A diagram representing the Transition matched by the Triggered Transition Pointcut appears, as illustrated in Figure 20. This diagram represents the Joinpoint matched by the Start Transition Pointcut (in this case, the complete Start Transition)

**Note:** The generated Operation is only added to the model for visualization purposes. It will not appear in the woven model. It contains a State Chart Diagram that represents the Transition matched by the Transition Pointcut.



**Figure 19.** The State Chart Diagram for the hello State Machine Implementation after entering the WEAVR mode and clicking the 'Show Joinpoints' button



**Figure 20.** The State Chart Diagram for the hello State Machine Implementation, the Connector Instance for the Start Transition Joinpoint and the State Chart Diagram representing the matched Transition (in this case, the complete Start Transition)

**Note:** The layout of the State Chart Diagrams might have been slightly modified due to the introduction of the Transition delimitation marks. To clean up the State Chart presentation, right-click on the diagram and select 'Automatic layout'

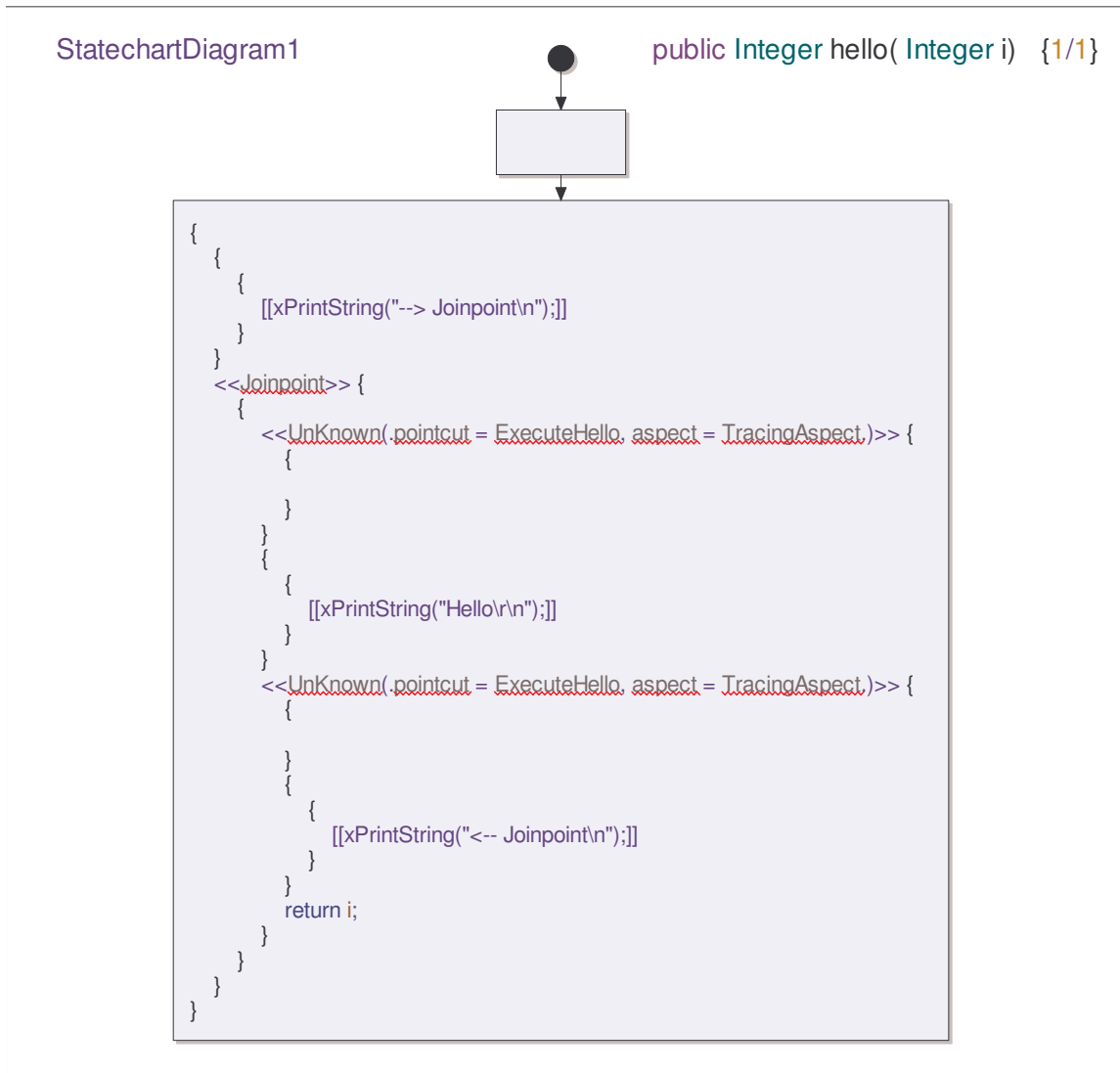
20. Weave the model by pressing the ‘Weave Model’ button
21. Open the woven\_project workspace
22. Re-generate to State Chart Diagram for the Hello::hello State Machine Implementation. They should look similar to the diagrams illustrated in Figure 21. Note that the generated Operation of Figure 20 does not appear in the woven Model. The Connector Instance applied to the Start Transition Joinpoint has been inlined in the model whereas the Connector Instances applied to the Action Joinpoints define wrappers around the Joinpoints.
23. Build the woven model, launch the Model Verifier and run the simulation. In the Model Verifier tab, note that there are two distinct tracing statements around the execution of the Hello::hello Operation: one for the call to hello and one for its execution.

```

* OPERATION START
hello_CallHello_ro_0ahFEIyCbBIL3fOfuLC58U3L##Tracing_0
* TASK
--> Joinpoint
* OPERATION START hello
* TASK
--> Joinpoint
* TASK
Hello
* TASK
<-- Joinpoint
* OPERATION RETURN hello
* ASSIGN n_attr :=
* TASK
<-- Joinpoint
* OPERATION RETURN
hello_CallHello_ro_0ahFEIyCbBIL3fOfuLC58U3L##Tracing_0
* ASSIGN n :=

```

**Listing 7.** Model Verifier tab output around the execution of the Hello::hello Operation, when both the CallHello and ExecuteHello Pointcuts are applied



**Figure 21.** The generated State Chart Diagram for the hello State Machine Implementation. The Connector Instance applied to the Start Transition has been inlined in the body of the Operation

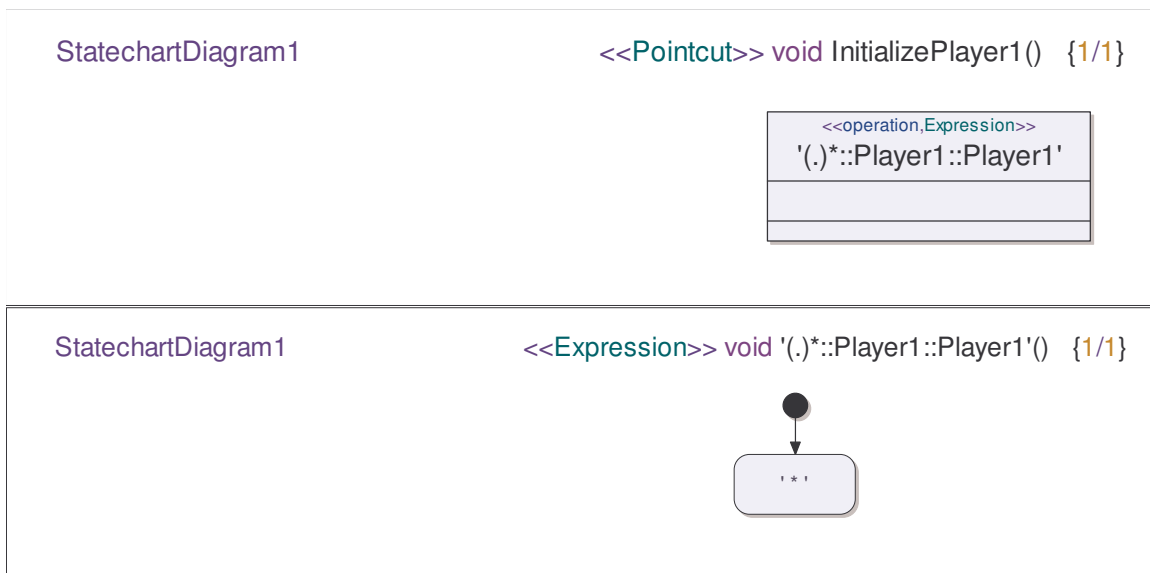
### 3.2.2 Initialization Transition Pointcuts

Initialization Transition Pointcuts are a particular form of Start Transition Pointcuts. They capture the initial Transition in a State Machine Implementation which executes before the first State is entered. We will write two Initialization Transition Pointcuts for the Player1 and Player2 State Machine Implementations.

1. In the Model View, copy the ExecuteHello Pointcut and paste it in the context of the TracingAspect
2. Rename the new Pointcut as 'InitializePlayer1'
3. Rename the Expression of the InitializePlayer1 as '(.)\*::Player1::Player1()'

4. Delete the Parameters of the Expression
5. Delete the Attribute 'a' in the State Machine Implementation of the Expression
6. In the State Chart Diagram of the Expression, delete the Return Symbol and replace it by a new State
7. Name the new State ' \* '
8. The Pointcut and Expression State Chart Diagrams should now look as displayed in Figure 22

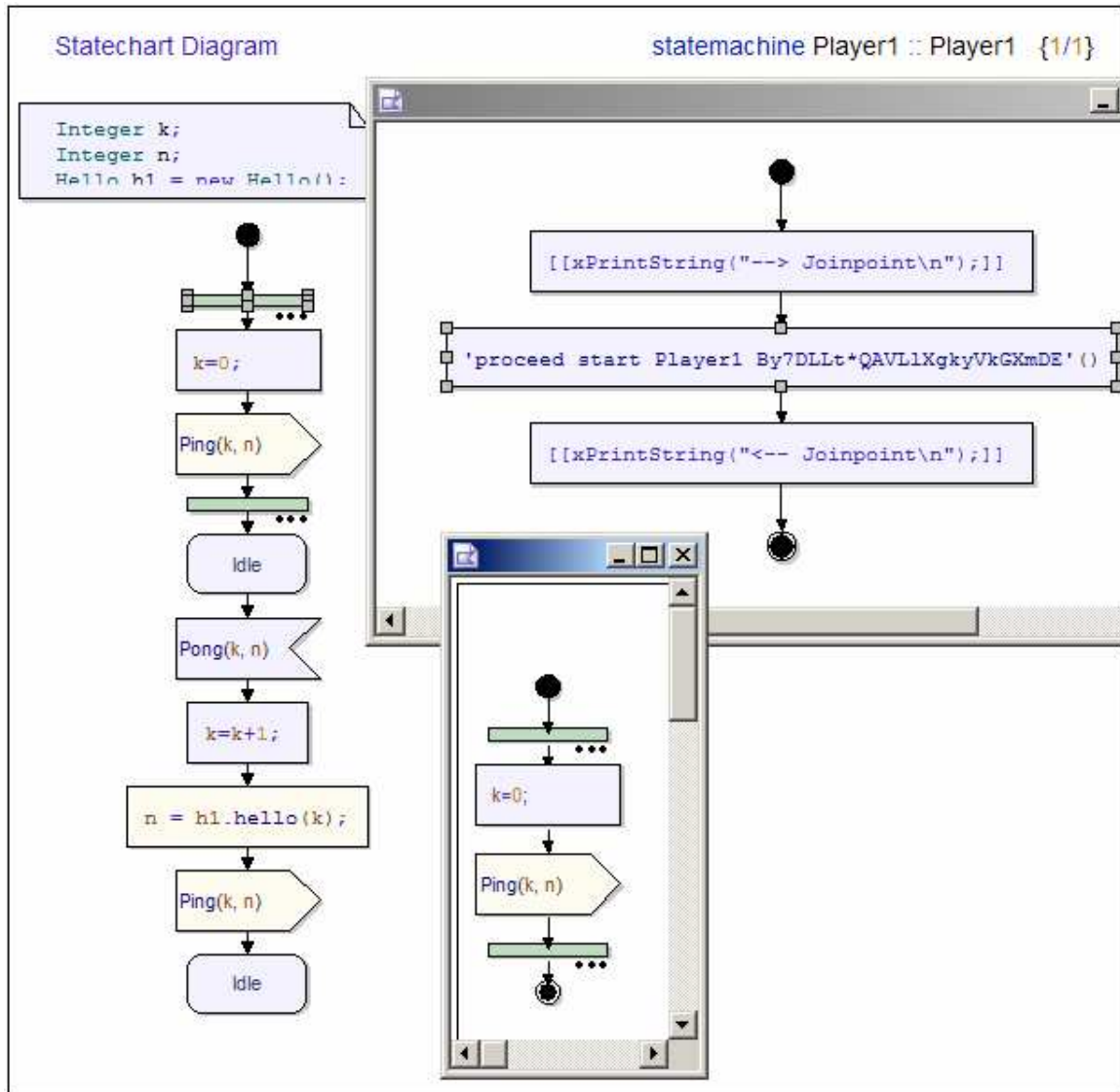
**Note:** You have to include spaces between the asterisk and the quotes.



**Figure 22.** Initialization Transition Pointcut Diagrams

9. In the Model View, copy and paste the InitializePlayer1 Pointcut in the context of the Tracing Aspect
10. Rename the new Pointcut 'InitializePlayer2'
11. Rename the Expression of the InitializePlayer2' as '<(.)\*::Player2::Player2()'
12. Bind the Connector to the new Pointcuts in the Binding Diagram of the Aspect
13. Save the workspace
14. Enter the WEAVR mode and click the 'Show Joinpoints' button
15. The Message Tab should indicate that 8 Joinpoints have been found.

16. Browse to the State Machine implementation of the Player1 Active Class. The State Chart Diagrams should look as displayed in Figure 23. The State Chart Diagram has been annotated with green marks that delimit the entry point and the exit points of the Initialization Transition. Also, note that a new operation has been generated in the context of the State Machine Implementation. It represents the Transition that is matched by the Pointcut
17. Click on either the Start Symbol or one of the green marks. A Connector Instance for the Initialization Transition appears, as displayed in Figure 23



**Figure 23.** The State Chart Diagram for the Player1 State Machine Implementation, the Connector Instance for the Initialization Transition Joinpoint and the State Chart Diagram representing the matched Transition

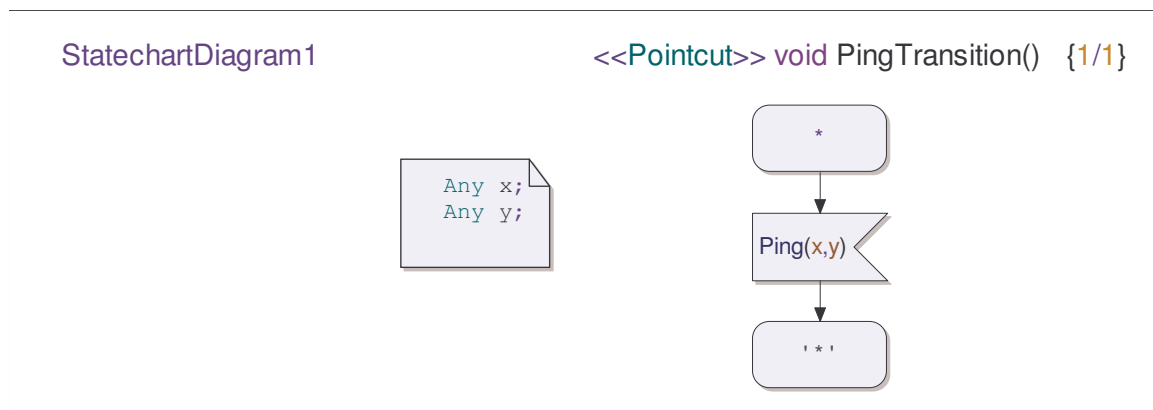
### 3.2.3 Triggered Transition Pointcuts

Triggered Transition Pointcuts are Pointcuts that match Transitions triggered by a Signal reception event or a Timer expiration event.

1. In the Model View, copy and paste the SendPing Pointcut in the context of the TracingAspect
2. Rename the new Pointcut 'PingTransition'
3. In the PingTransition Pointcut State Chart Diagram, delete the Start Transition and the Ping Output Action
4. Create an asterisk State Transition, triggered by the Ping Signal Expression, and taking attribute 'x' as a Left Value
5. Create a new State and call it ' \* '

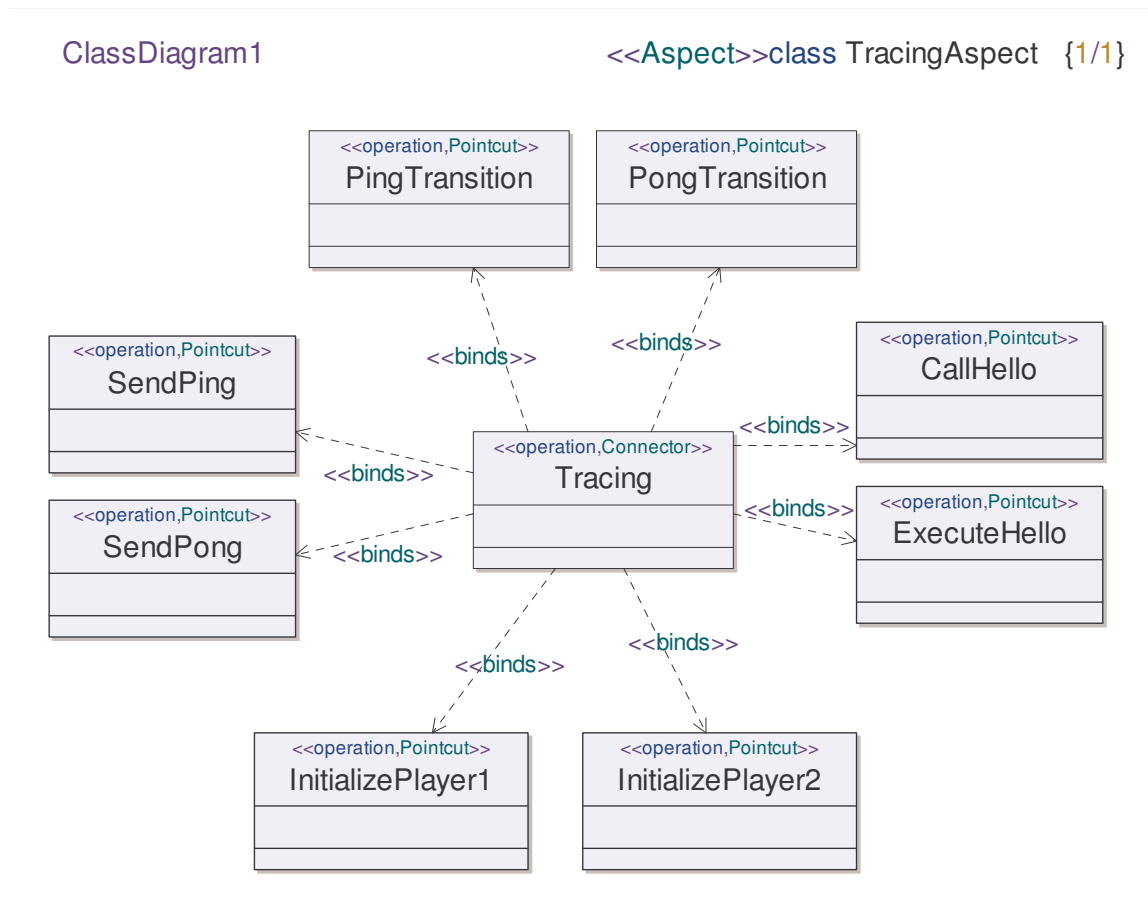
**Note:** You have to put a space between the asterisk and the quotes.

6. Create a Next State Action from the Transition Trigger to the new State
7. The PingTransition Pointcut should look as displayed in Figure 24



**Figure 24.** The PingTransition Pointcut in the MyTracingAspect Aspect

8. In the Model View, copy and paste the PingTransition Pointcut in the context of the TracingAspect. Rename the copy 'PongTransition' and rename its Expression 'Pong'
9. Bind the Tracing Connector to the PingTransition and PongTransition Pointcuts
10. The Binding Diagram of the TracingAspect Aspect should now look as displayed in Figure 25
11. Save the workspace
12. Enter the WEAVR mode and click the 'Show Joinpoints' button. The Message Tab should indicate that 10 Joinpoints have been matched, as indicated in Listing 8



**Figure 25.** The Binding Diagram of the Tracing Aspect

```

WEAVR>> Start Transition Joinpoint PingPong::Hello::hello matches pointcuts:
ExecuteHello
WEAVR>> Start Transition Joinpoint PingPong::Match::Player1::Player1 matches
pointcuts: InitializePlayer1
WEAVR>> Output Action Joinpoint Ping matches pointcuts: SendPing
WEAVR>> CallExpr Action Joinpoint PingPong::Hello::hello matches pointcuts:
CallHello
WEAVR>> Output Action Joinpoint Ping matches pointcuts: SendPing
WEAVR>> Start Transition Joinpoint PingPong::Match::Player2::Player2 matches
pointcuts: InitializePlayer2
WEAVR>> Triggered Transition Joinpoint Ping matches pointcuts: PingTransition
WEAVR>> Triggered Transition Joinpoint Pong matches pointcuts: PongTransition
WEAVR>> CallExpr Action Joinpoint PingPong::Hello::hello matches pointcuts:
CallHello
WEAVR>> Output Action Joinpoint Pong matches pointcuts: SendPong
WEAVR>> Total Joinpoints: 10

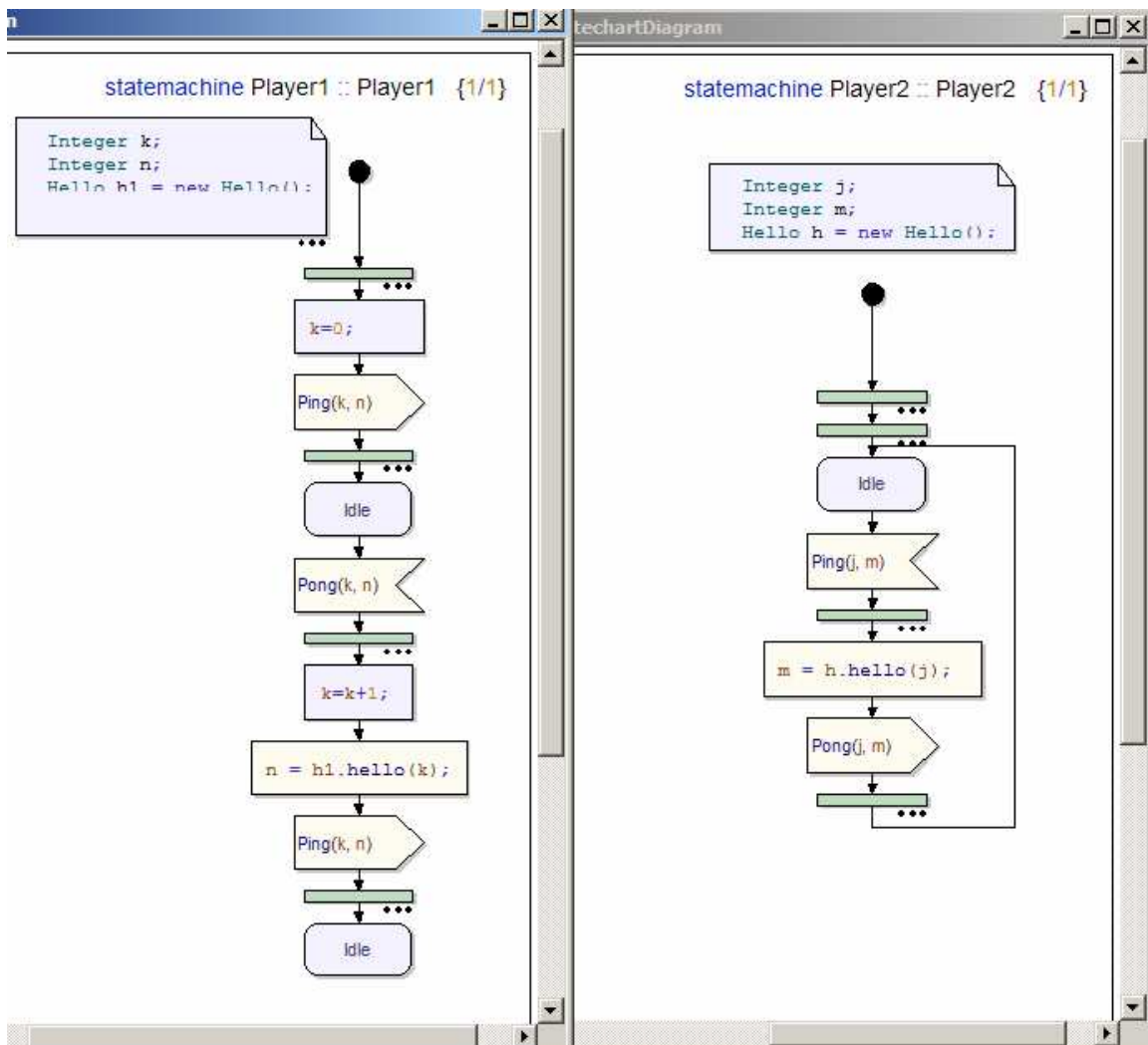
```

### Listing 8. Message Tab output for the TracingAspect Aspect

**Note:** The generated Operation is only added to the model for visualization purposes. It will not appear in the woven model. It contains a State Chart Diagram that represents the Transition matched by the Transition Pointcut.

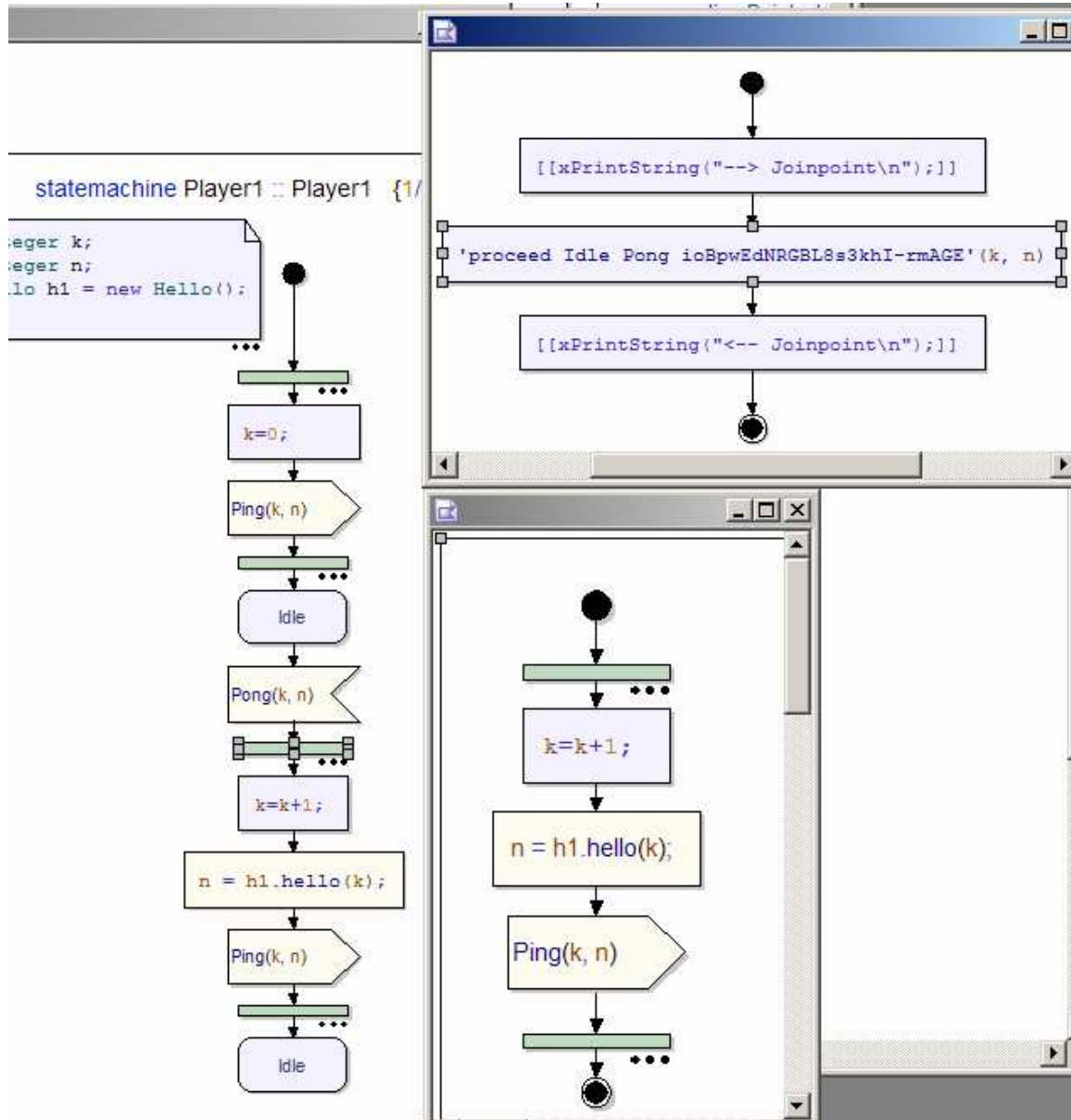
13. The State Chart Diagrams should look like displayed in Figure 26. In addition to the Action Joinpoint highlights, the diagrams have also been annotated with green marks that delimit the entry point and the exit points of the transitions that match the Ping and Pong Transition Pointcuts. The Input Symbol of the transition has also been highlighted. Also, note that a new operation has been generated in the context of the State Machine Implementation where a Transition matches the Transition Pointcut
14. Click on either the Input Symbol or one of the Marks. A Connector Instance for the Triggered Transition appears, as displayed in Figure 27
15. In the Connector Instance Diagram, press ctrl and click on the call to the generated operation. A diagram representing the Transition matched by the Triggered Transition Pointcut appears, as illustrated in Figure 27. This diagram represents the Joinpoint matched by the Transition Pointcut, represented as a Start Transition.
16. Weave the model by pressing the 'Weave Model' button
17. Open the woven\_project workspace

18. Re-generate to State Chart Diagram for the Player1 and Player2 Active Classes.  
They should look similar to the diagrams illustrated in Figure 28. Note that the generated Operation of Figure 27 does not appear in the woven Model. The Connector Instances applied to the Triggered Transition Joinpoints have been inlined in the model whereas the Connector Instances applied to the Action Joinpoints define wrappers around the Joinpoints.
19. Build the woven model, launch the Model Verifier and run the simulation. Note that tracing is applied to all the Action and Transition Joinpoints



**Figure 26.** The State Chart Diagrams of Player1 and Player2 after clicking 'Show Joinpoints'

**Note:** The layout of the State Chart Diagrams might have been slightly modified due to the introduction of the Transition delimitation marks. To clean up the State Chart presentation, right-click on the diagram and select 'Automatic layout'



**Figure 27.** The Player1 State Chart Diagram, the Connector Instance for the Triggered Transition Joinpoint, and a representation of the Triggered Transition Joinpoint as a Start Transition



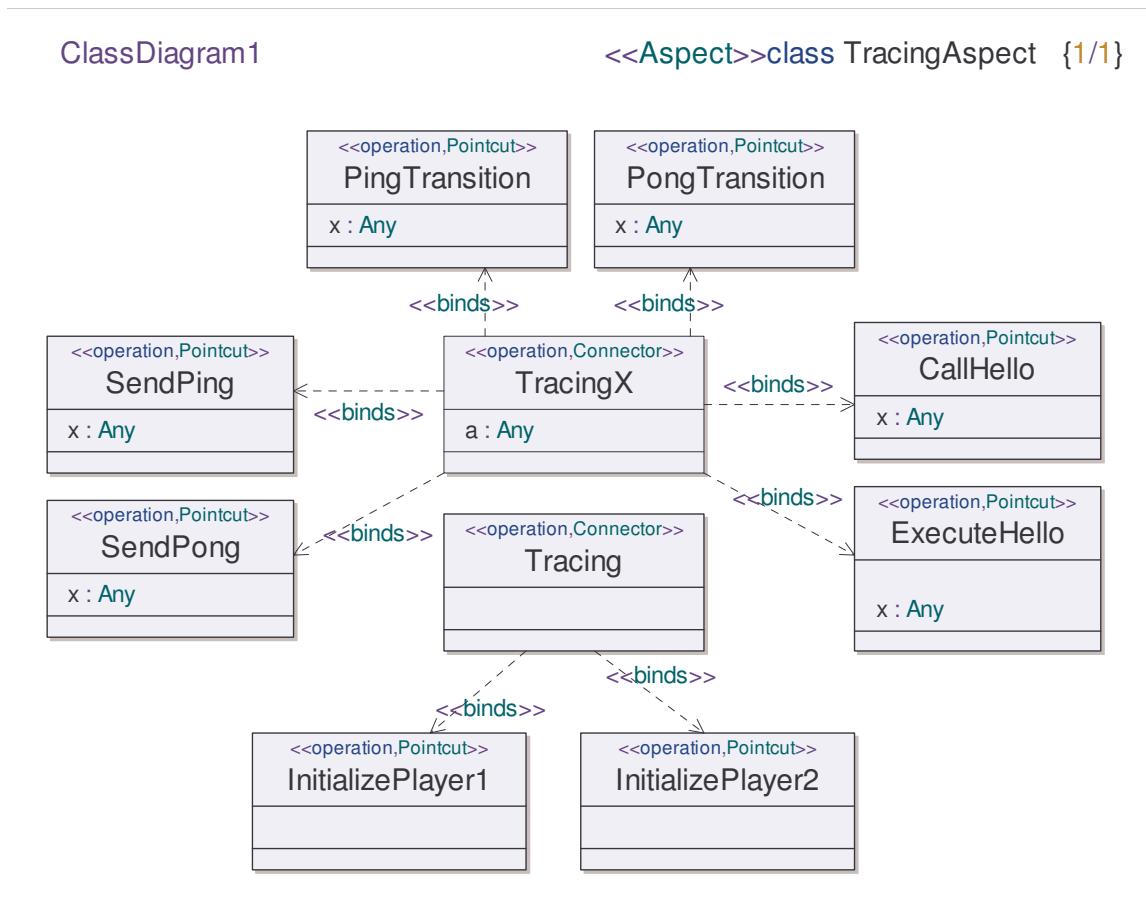
**Figure 28.** Player1 and Player2 generated diagrams for the woven model

### 3.3 Accessing Action Joinpoint Arguments and Transition Joinpoint Parameters

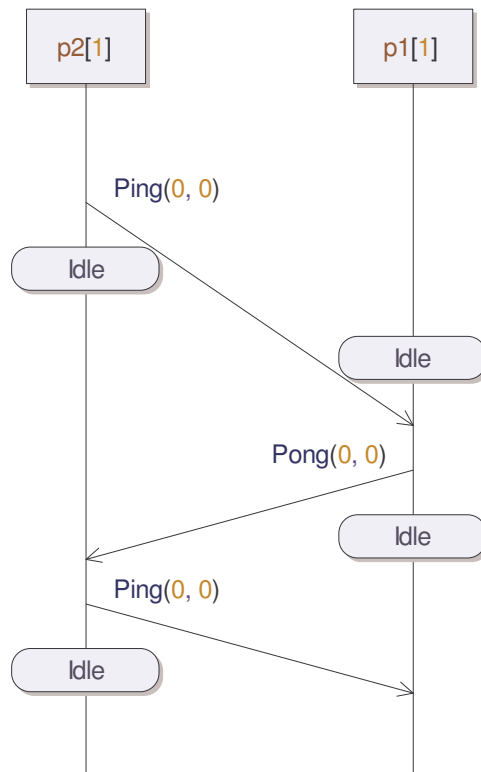
This Section illustrates how to pass Action Joinpoint Arguments or Transition Joinpoint Parameters to the Connector. Pointcuts can expose some or all of the Joinpoint Arguments/Parameters to Connectors. The Connector can then take specific actions with respect to their values or modify their value before or after the Joinpoint is executed.

1. Add a new Parameter to the SendPing Pointcut. Name it 'x' and give it a type Any
2. In the Model View, copy the Parameter and paste it in all the Action Pointcuts (SendPing, SendPong and CallHello), in the Start Transition Pointcut (ExecuteHello) and in the Triggered Transition Pointcuts (PingTransition and PongTransition). The Initialization Transition Pointcuts cannot expose any Parameter because the Player1() and Player2() State Machine Implementations do not have any Parameter
3. Delete the 'x' attributes in the State Machine Implementations of the Action Pointcuts (SendPing, SendPong and CallHello), the Start Transition Pointcut (ExecuteHello) and the Triggered Transition Pointcuts (PingTransition and PongTransition)
4. Make sure the ExecuteHello Pointcut uses 'x' as the name of the first Parameter of its Expression
5. In the Binding Diagram, delete all the binding Dependencies from the Tracing Connector to all Pointcuts. (right-click on the Dependencies, and select 'Delete Model' or delete the Dependencies directly in the Model View)
6. In the Model View, copy and paste the Tracing Connector
7. Rename the new Connector 'TracingX'
8. Paste the 'x' Parameter in the TracingX Connector and rename it as 'a'
9. Update the Joinpoint (the Call Expression Action to the Connector, which is annotated by the 'Joinpoint' stereotype) of the TracingX Connector State Chart Diagram so that it passes it a '0' Integer value. This will have the effect of reinitializing the Integer value of the Arguments passed to the Output Actions or reinitializing the LValue of the Signal Parameters in the case of Triggered Transitions to 0.
10. In the Binding Diagram, create new binding Dependencies from the TracingX Connector to all the Pointcuts that have a Parameter

11. In the Binding Diagram, create new binding Dependencies from the Tracing Connector to the Initialization Transition Pointcuts (InitializePlayer1 and InitializePlayer2). The Binding Diagram should now look as displayed in Figure 29
12. Weave the model and open the woven\_project workspace
13. When running the simulation, the first Parameter of the Signals is reinitialized to 0 at every Output Action and Triggered Transition, as shown in Figure 30



**Figure 29.** Binding Diagram for the Tracing Connector and the TracingX Connector, with Parameter passing



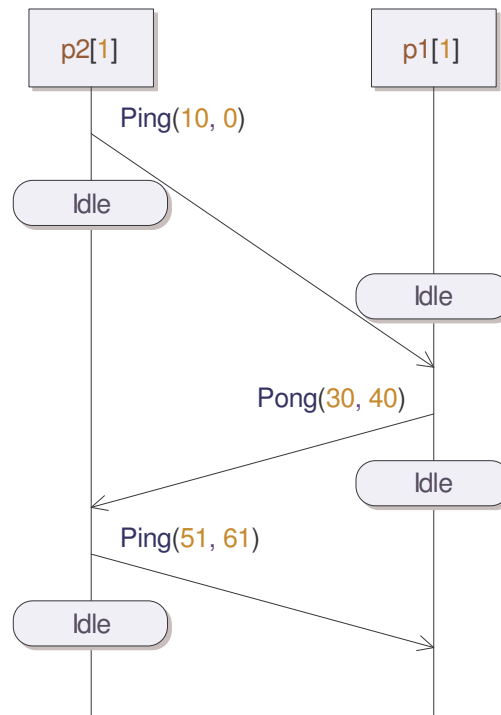
**Figure 30.** Generated Sequence Diagram when setting the Joinpoint Parameter to '0' in the Connector Diagram

14. In the Connector, add a 'a = a + 10;' statement before the Call to the Joinpoint, and set the Parameter of the Joinpoint (The call to TracingX, annotated with the Joinpoint stereotype) to 'a' in the Connector. The generated trace should now look like displayed in Figure 31.

The Parameters are incremented as follow:

- The first Parameter is incremented before the Signal is Send by p1[1]  
(SendPing Pointcut) : (k, n) = (0 + 10 = 10, 0)
- The first Parameter is incremented upon reception of a Signal by p2[1]  
(PingTransition Pointcut) : (j, m) = (10 + 10 = 20, 0)
- The Argument of the call to Hello::hello() is incremented in the wrapper around the hello method by p2[1]  
(CallHello Pointcut) : (binding\_i) = (20 + 10 = 30)
- The Parameter of the hello method is incremented by p2[1]  
(ExecuteHello Pointcut) : (i) = (30 + 10 = 40, 0)

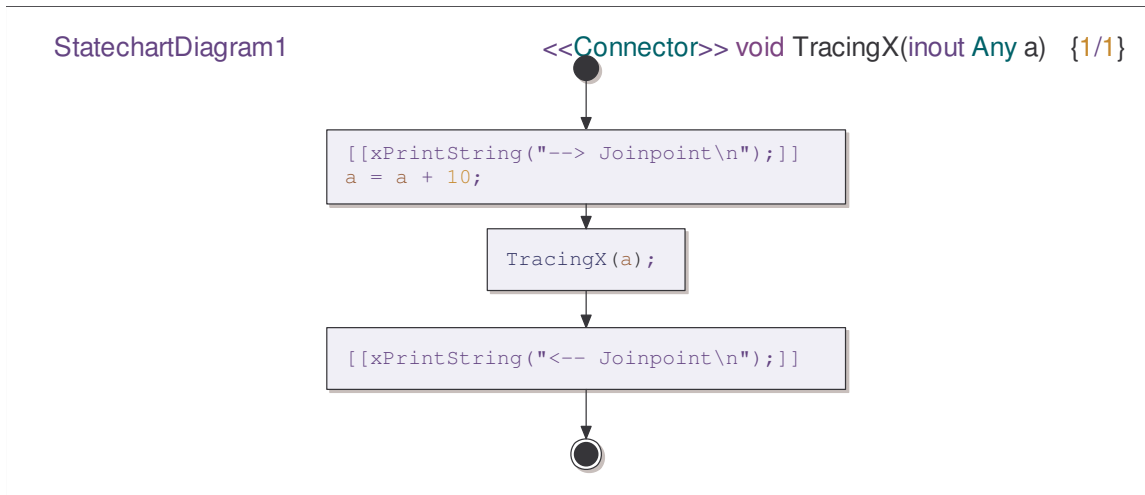
- The second parameter is assigned the return value of the Hello::hello() call  
(j, m) = (20, 40)
- The first Parameter is incremented before the Signal is Send by p2[1]  
(SendPong Pointcut) : (j, m) = (20 + 10 = 30, 40)



**Figure 31.** Generated Sequence Diagram when incrementing the Connector Parameter by 10 in the Connector State Machine Implementation

Note that the Connector Instances applied to the Hello::hello() method call and execution do not have side effects. When the method is called, the Connector Instance increments the generated Parameter binding\_i, while it does not have effects on the ‘j’ Attribute which is passed as an Argument to the method.

The effects of the Connector can be controlled by setting the Direction of its Parameters. By default, the Direction is set to ‘In’ and Connectors bound to Action Pointcuts do not have side effects on the Action Arguments (Spectative and Regulative Connectors). To allow Connectors to have side effect on Action Joinpoint Arguments (Invasive Connectors), set the Direction of the Connector Parameters to ‘In/Out’ as shown in Figure 32.

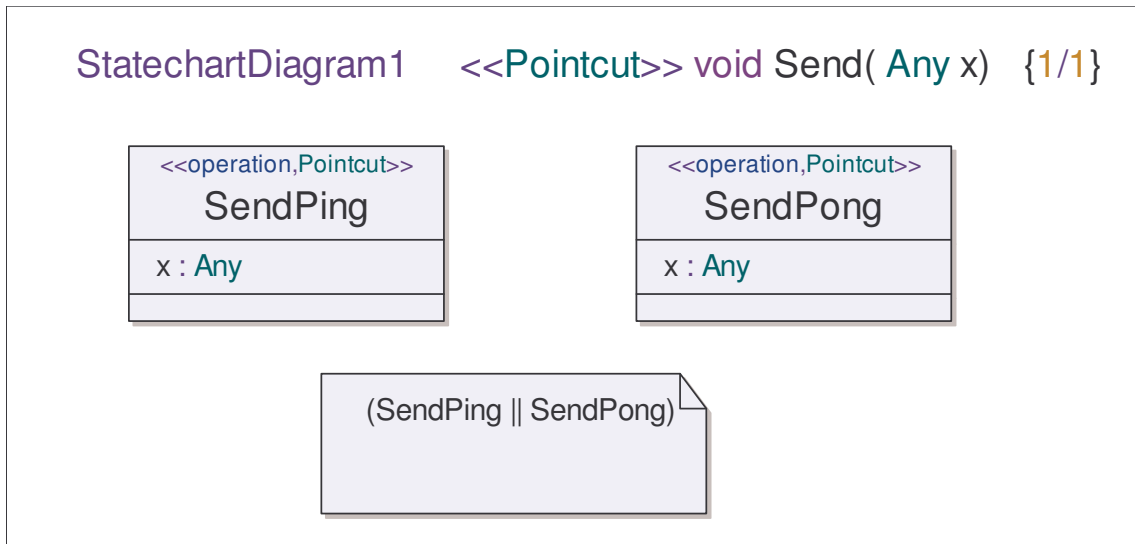


**Figure 32.** TracingX Connector Diagram with side effects on the Action Joinpoint Arguments exposed by the Pointcuts

### 3.4 Composing the Pointcuts

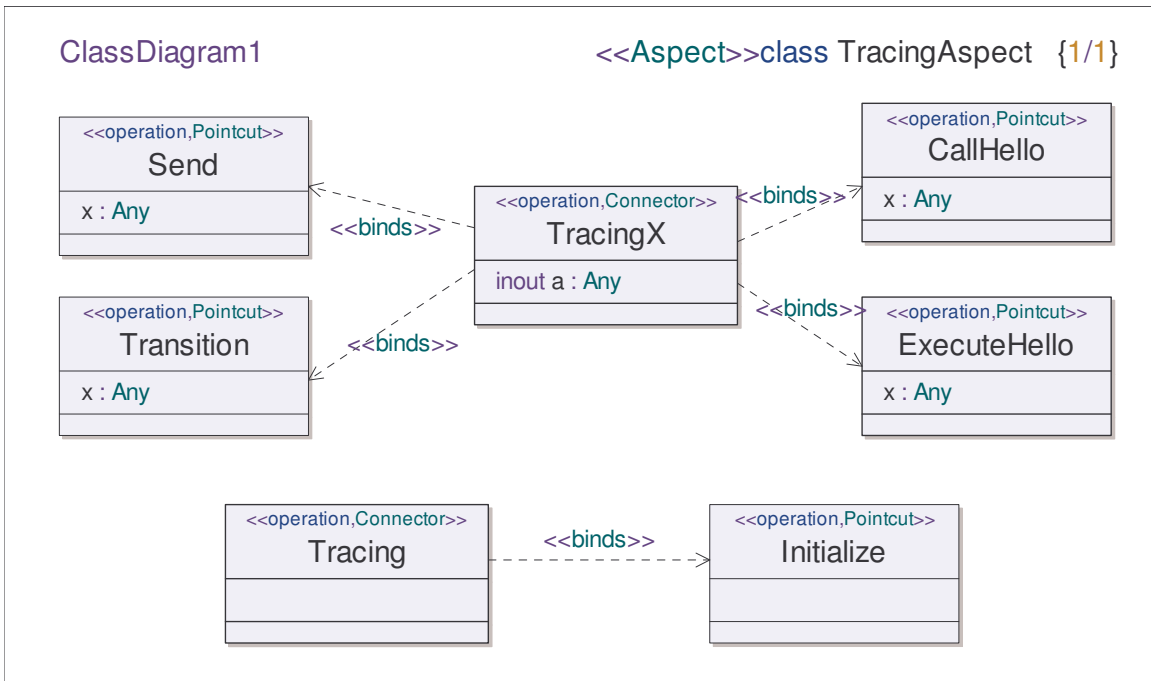
The Binding Diagram of Figure 29 is very explicit. Yet, having multiple binding dependencies can become hard to manage. When Aspects apply to particular Joinpoints in the base model it is often necessary to compose the Pointcuts into more complex expressions. This Section presents Pointcut Composition operators.

1. In the Model View, copy and paste the SendPing Pointcut in the context of the TracingAspect Aspect
2. Rename the new Pointcut Send
3. Delete its State Machine Implementation
4. Create a new State Chart Diagram for the Pointcut
5. In the Model View, drag and drop the SendPing and SendPong Pointcuts from the TracingAspect into the State Machine Implementation of the Send Pointcut
6. Drag and drop the SendPing and SendPong Pointcuts from the Model View into the new State Chart Diagram
7. Create an new Comment Box in the State Chart Diagram and type '(SendPing || SendPong)' as displayed in Figure 18

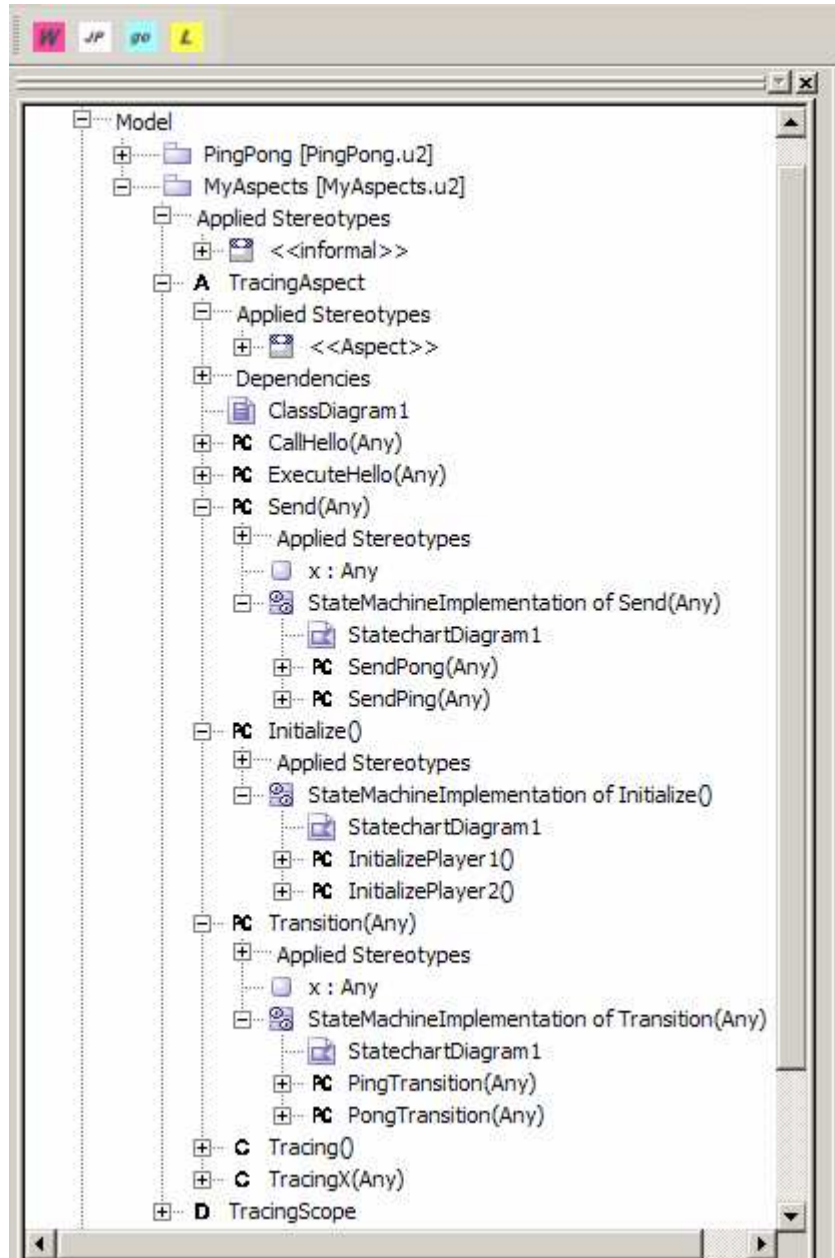


**Figure 33.** The Send Pointcut is an ‘OR’ composition of Pointcut SendPing and Pointcut SendPong

8. Repeat steps 1 through 7 for the PingTransition and PongTransition Pointcuts and call the new Pointcut ‘Transition’
9. Repeat steps 1 through 7 for the InitializePlayer1 and InitializePlayer2 Pointcuts and call the new Pointcut ‘Initialize’
10. In the Binding Diagram, delete the dependencies to the SendPing, SendPong, PingTransition, PongTransition, InitializePlayer1 and InitializePlayer2 Pointcuts. (right-click on the Dependencies, and select ‘Delete Model’ or delete the Dependencies directly in the Model View)
11. Create new bindings from the TracingX Connector to the Send and Transition Pointcuts. Create a new binding from the Tracing Connector to the Initialize Pointcut. The Binding Diagram should now look as displayed in Figure 34 and the corresponding Model View should look as shown in Figure 35
12. In the Model View, copy and paste the Send Pointcut from the TracingAspect Aspect into the State Machine Implementation of the Send Pointcut (its own State Machine Implementation). Rename the new Pointcut SendPingPong
13. Rename the SendPing and SendPong pointcuts contained in the SendPingPong Pointcut as SendPingPongPing and SendPingPongPong
14. Change the composition rule of the SendPingPong Pointcut to ‘SendPingPongPong || SendPingPongPing’



**Figure 34.** Binding Diagram with binding Dependencies to the composite Pointcuts Send, Transition and Initialize

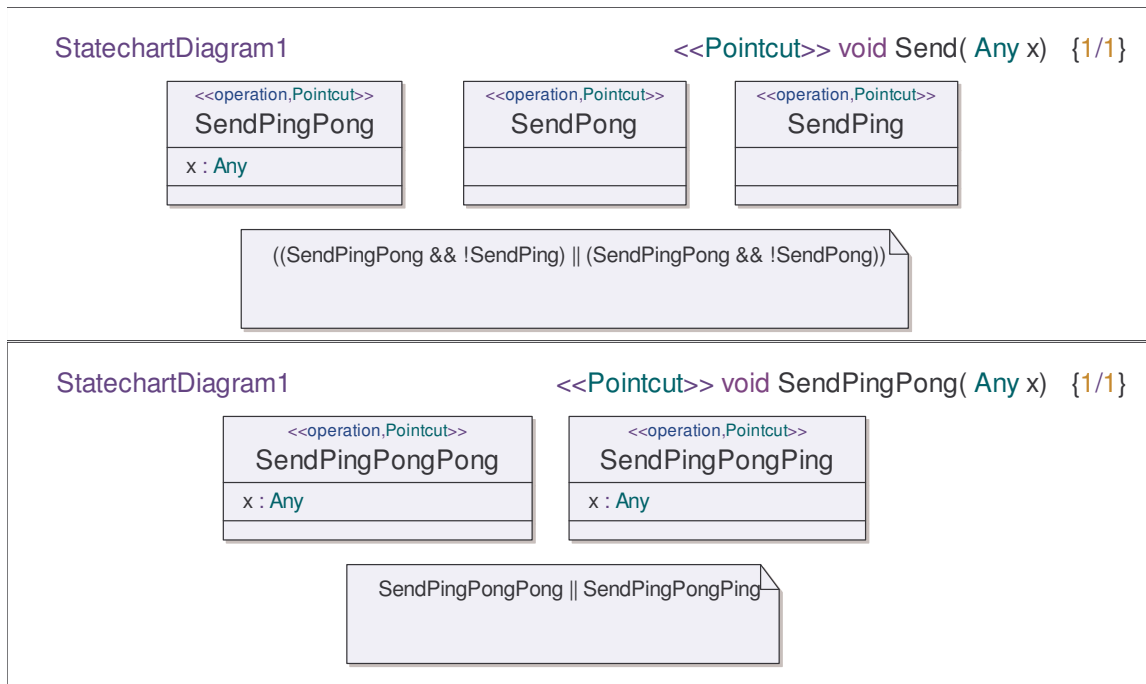


**Figure 35.** Model View of the Send, Transition and Initialize composite Pointcuts

15. Delete the Parameters of the SendPing and SendPong Pointcuts. Add a new 'x' Attribute of type Any in the State Machine implementation of those Pointcuts to represent the Output Action Arguments.
16. In the State Chart Diagram of the Send Pointcut, change the composition rule into '((SendPingPong && !SendPing) || (SendPingPong && !SendPong))'. The State Chart Diagram for the Send Pointcut and the SendPingPong Pointcut should now look

as displayed in Figure 36. The Model View of the composite Pointcuts is presented in Figure 37

17. Enter the WEAVR model, weave the model and launch the simulation. There should still be 10 Joinpoints in the model



**Figure 21.** The State Chart Diagrams for the Send and SendPingPong composite Pointcuts

**Note:** When using the 'NOT' logical operator, the Pointcut that is negated cannot expose and Parameters. Pointcuts that are negated do not translate into particular Joinpoints, and can therefore not expose any Joinpoint Argument/Parameter

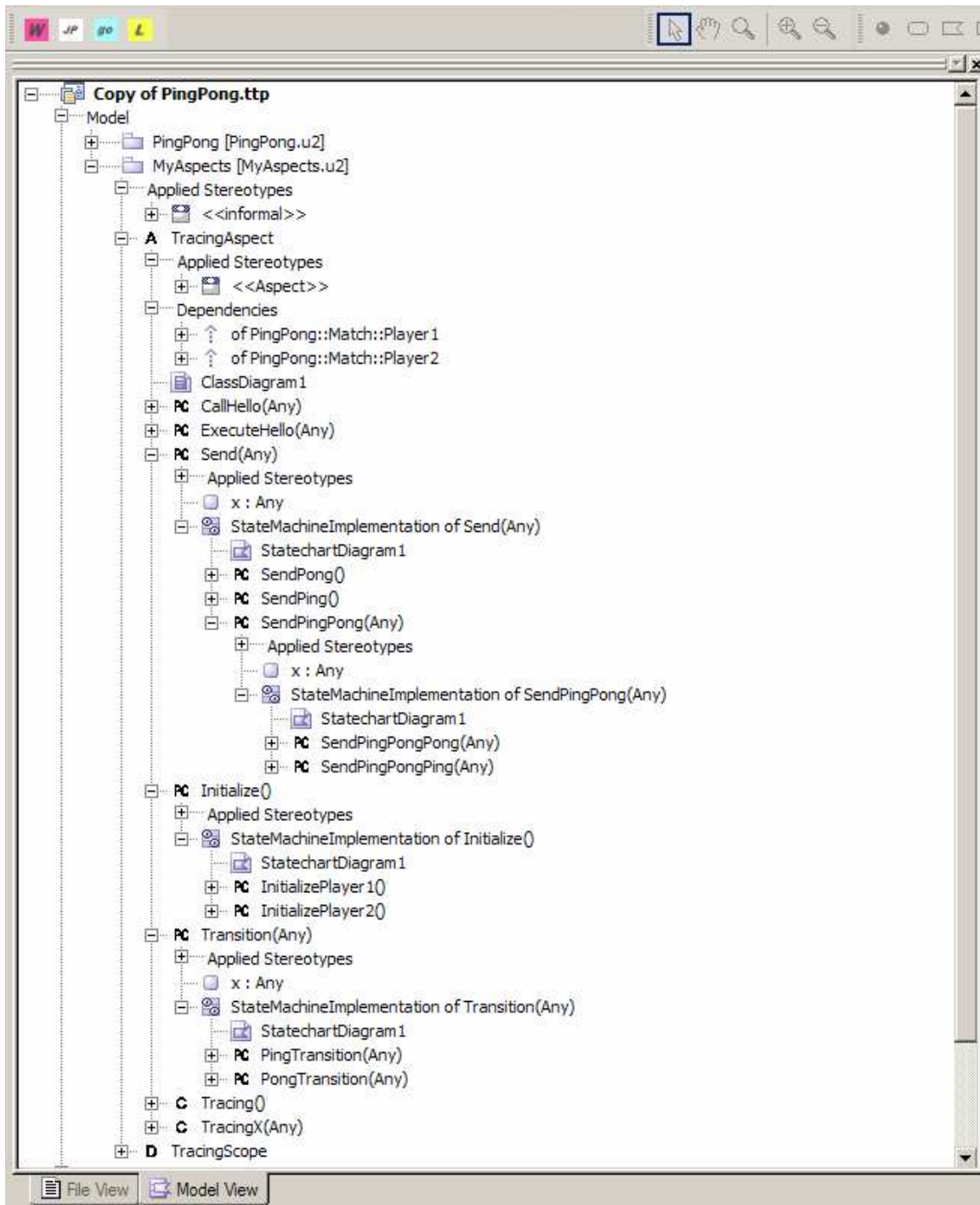
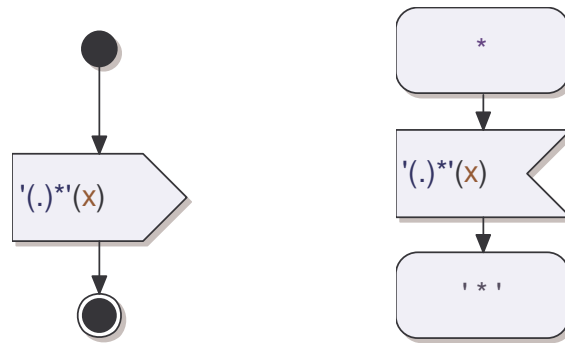


Figure 37. The Mode View for the Send and SendPingPong composite Pointcuts

### 3.5 Using Wildcards

Pointcuts can match a broad set of events by using wildcards in their Expression. The Signature of the Expression is interpreted as a regular expression.

For example, the Send Pointcut and the Transition Pointcut can be expressed as non-composite Pointcuts as illustrate in Figure 21



**Figure 21.** The Send and Transition Pointcut State Charts, when expressed as non-composite Pointcuts using wildcard Expressions

## Capturing the Joinpoint Context

### 18.Aspect Introductions

### 19.Modularizing Inter-Process Dependencies

### 20.Modularizing Inter-Process Protocols

### 21.

