Conversations

Example File (Builder project Advanced Modeling/PState):

	<your example path>\Advanced Modeling\PState\uml\pstateConversation.xml

Related Pages:

- Contents of the Transaction Log

In certain cases a communication from an object to a synchronous service is required.
Imagine a service using persistent state, that sends a **GiveCreditSignal** to a customer object and has to wait for approval or rejection before proceeding. An inelegant and very inefficient way to implement this is a loop that invokes **getStateInfo** or **getObjectCopy** until the approval went through. A much better solution would allow the service to wait for a signal sent back by the object.

This construct of a conversation is implemented by two elements: a signal and a conversationID. The waiting service waits for a specific signal sent by the conversation partner. This signal is identified by the **signal name** and the **conversationID**, that can be passed as a parameter to each signal action.

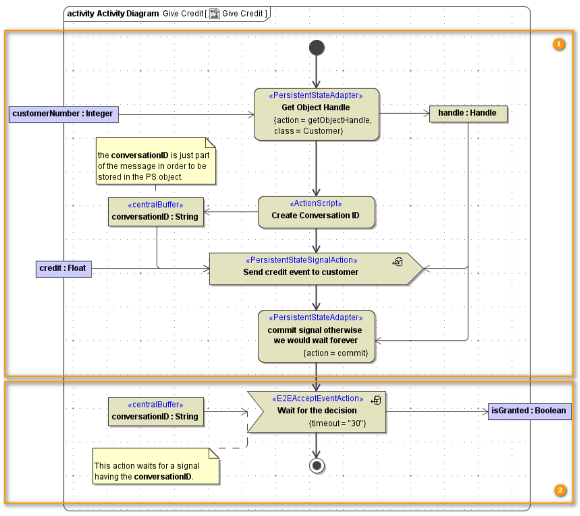
The conversation ID will be logged as correlation ID to the transaction log for **send signal actions** and **accept event actions** (see Contents of the Transaction Log).

The conversations are persisted in a persistent state database. If you stop a service or the service crashes between sending and accepting, the service will try to complete the conversation after service restart. Nevertheless, the accept event timeouts continue counting down and is considered after service restart.
If you use an external persistent state database, multiple services with same persistent state **owner** can send conversation signals between each other.

Initiating a Conversation and Waiting for a Conversation Response

Look at the activity diagram below. It is depicting the process of initiating a conversation and waiting for the response afterwards.

Figure: Sending Signal to an Object and Waiting for the Response


activity Activity Diagram Give Credit[Give Credit]
1
customerNumber : Integer
«PersistentStateAdapter»
Get Object Handle
{action = getObjectHandle, class = Customer}
handle : Handle
the conversationID is just part of the message in order to be stored in the PS object.
«centralBuffer»
conversationID : String
«ActionScript»
Create Conversation ID
credit : Float
«PersistentStateSignalAction»
Send credit event to customer
«PersistentStateAdapter»
commit signal otherwise we would wait forever
{action = commit}
«centralBuffer»
conversationID : String
«E2EAcceptEventAction»
Wait for the decision
{timeout = "30"}
isGranted : Boolean
This action waits for a signal having the conversationID.
2

1. Initiating the Conversation

In the example above, a request for credit is sent to a persistent state object **Customer**. The object is identified by its customer number, which is the primary key of the **Customer** class.

Then, a conversation id is created and a **GiveCredit** signal is sent to the customer object. The conversationID is passed to the Customer object, so it can be used later on to send the reply (see Sendin g Conversation Signals).

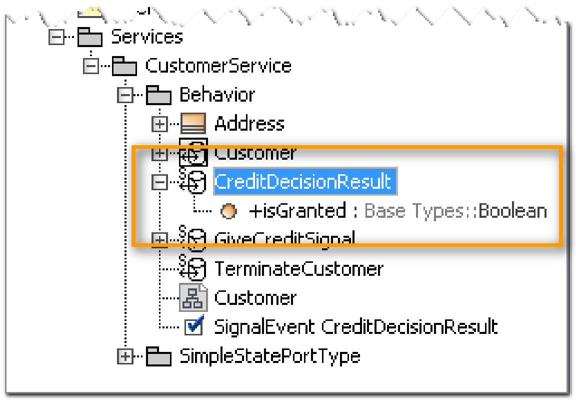
Persistent state signals are implicitly committed on session end only and signals will be transferred to the recipient **after commit** (also see xUML Runtime Transaction Concepts). So in this case, the persistent state adapter has to be manually committed, because the accept event action was added to the same activity diagram. Manually committing is done by the use of the action **commit**.

2. Waiting for the Conversation Response

In part two of the activity diagram, the service is waiting for the response to arrive. Response means a signal sent by a send signal action with a given conversationID. In this case, this is the credit decision signal (the very same signal that was sent by the send signal action described below in Sending Conversation Signals). The service will wait for the signal until the timeout given in the tagged value **time out** is reached.

The result of the credit check is transported in an attribute of the signal.

Figure: Attributes of the Returning Signal



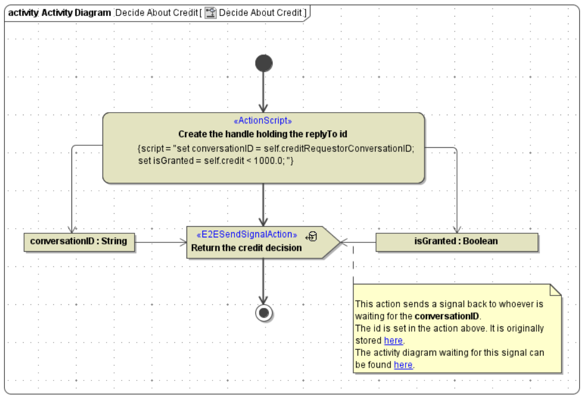
This is done using the <<E2EAcceptEventAction>> stereotype. When a signal arrives, its attributes are passed as output. If no signal arrives within the given timeout, an exception will be thrown. Default timeout is 30 seconds.

The conversation id passed to the accept event action must be exactly assigned that name: **convers ationID**. Otherwise, it will not be recognized as a conversation id.

Sending Conversation Signals

When a conversation partner is waiting for a response, you can send a signal to this conversation using the <<E2ESendSignalAction>>. The interface is very similar to sending **signals to objects**. However only the provided **conversationID** is relevant to identify the target conversation of the signal.

Figure: Constructing a Handle and Sending a Conversation Signal


activity Activity Diagram Decide About Credit[Decide About Credit]
«ActionScript»
Create the handle holding the replyTo id
{script = "set conversationID = self.creditRequestorConversationID;
set isGranted = self.credit < 1000.0; "}
conversationID : String
«E2ESendSignalAction»
Return the credit decision
isGranted : Boolean
This action sends a signal back to whoever is waiting for the conversationID.
The id is set in the action above. It is originally stored here.
The activity diagram waiting for this signal can be found here.

In the example above, the conversation ID received earlier has been stored in attribute **creditRequestor ConversationID** of persistent state class customer. Now, it is re-used when sending the signal the conversation partner is waiting for. Additionally a Boolean is passed as a signal parameter.

The conversation id passed to the send signal action must be exactly assigned that name: **conversationID**. Otherwise, it will not be recognized as a conversation id.