Code explanation

The code required in order to submit communication events to the monitor consists in three main parts:

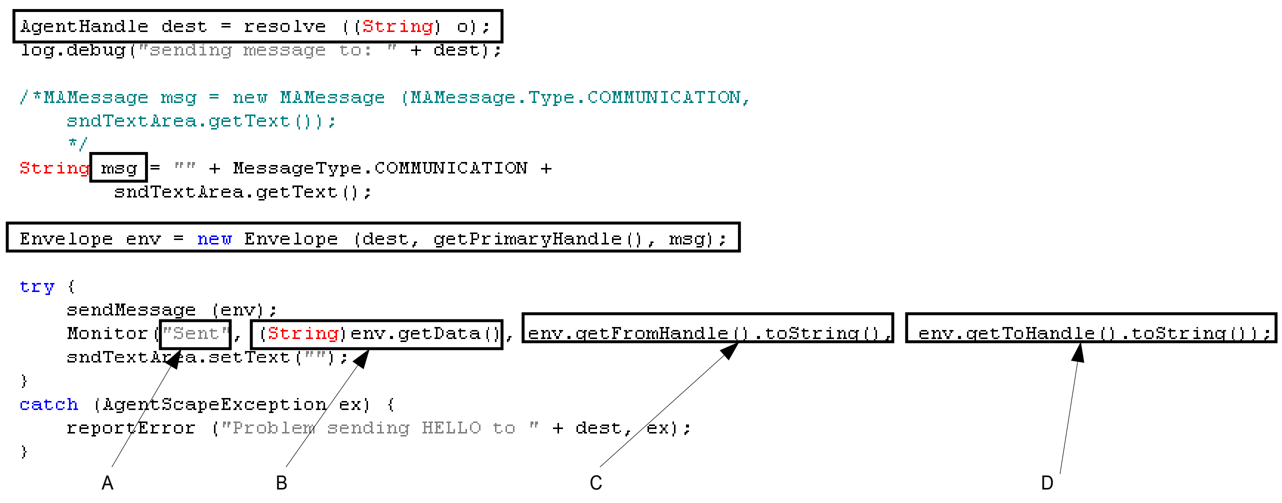
1. Importing required libraries: This is the set of libraries to be imported. The set can be copied from the file *'MonitorTest\monitor\MonitorizedChatter.java'* that is delivered along with this document.

```
import net.sf.ictalive.eventbus.EventBus;
import alive.EventModel.Fact.SendAct;
import alive.EventModel.Event.Actor;
import alive.EventModel.Event.EventFactory;
import alive.EventModel.Event.Event;
import alive.EventModel.Fact.FactFactory;
import org.eclipse.emf.ecore.EObject;
import alive.EventModel.Fact.Content;
import opera.OM.Atom;
import opera.OM.OMFactory;
import alive.EventModel.Fact.Message;
import alive.EventModel.Fact.Fact;
```

2. Wrapping event submission: A function that, given a sender, a receiver and a message performs a submission to the monitor has been coded. This function can be implemented in the agents that need to perform submissions to the monitor.

```
private void Monitor(String cause, String concept, String from, String to)
{
            //EventBus instance
            EventBus eb = new EventBus();
            Event dummyEvent;
            SendAct dummyFact;
            Actor senderAgent;
            Actor reciverAgent;
            Content dummyContent;
            Message dummyMess;
            Atom dummyAtom;

            //Start of the Russian Dolls game
            //Event to be sent
            dummyEvent = EventFactory.eINSTANCE.createEvent();
            //Fact inside the Content
            dummyFact = FactFactory.eINSTANCE.createSendAct();
            //Content inside the Event
            dummyContent= FactFactory.eINSTANCE.createContent();
            //Message inside the Fact
            dummyMess = FactFactory.eINSTANCE.createMessage();
            //Atom inside the message
            dummyAtom = OMFactory.eINSTANCE.createAtom();

            //Set cause and concept (typically 'Sent' and the message sent) into the atom
            dummyAtom.setProposition(cause + " : " + concept);
            dummyMess.getObject().add(dummyAtom);
            //Put the message inside the fact
            dummyFact.setSendMessage(dummyMess);
            //Initialize sender and receiver using Agent's handlers
            senderAgent = EventFactory.eINSTANCE.createActor();
            senderAgent.setName(from);
            senderAgent.setUrl("localhost");
            reciverAgent = EventFactory.eINSTANCE.createActor();
            reciverAgent.setName(to);
            reciverAgent.setUrl("localhost");
             //Set sender and receiver in the fact
             dummyFact.setSender(senderAgent);
             dummyFact.setReceiver(reciverAgent);
             //Put the fact into the content
             dummyContent.setFact(dummyFact);
             //Put the content into the event
             dummyEvent.setContent(dummyContent);

             //Set aserter of the event
             dummyEvent.setAsserter(senderAgent);

             //Publish the event
             eb.publish(dummyEvent);

 }

}
```

3. Integrating event submission: Just call the function when sending messages to another agent. Typically, this can be done after '*sendMessage*' functions. The envelope used to send the message contains all the fields of interest for the monitor. The fields sent are:
   a) Type of event to be summited: in this case a message sent from one agent to another
   b) Data: Can be retrieved from the envelope. Notice in this example it is of type String. When sending other types of data, function presented in point 2 might need to be adapted.
   c) Sender: Can be easily retrieved using the '*getPrimaryHandle()*' function.
   d) Receiver: Typically (as seen in this example) retrieved by performing a look-up using an agent's name.



```
AgentHandle dest = resolve ((String) o);
log.debug("sending message to: " + dest);

/*MAMessage msg = new MAMessage (MAMessage.Type.COMMUNICATION,
    sndTextArea.getText());
    */
String msg = "" + MessageType.COMMUNICATION +
        sndTextArea.getText();

Envelope env = new Envelope (dest, getPrimaryHandle(), msg);

try {
    sendMessage (env);
    Monitor ("Sent", (String)env.getData(), env.getFromHandle().toString(), env.getToHandle().toString());
    sndTextArea.setText("");
}
catch (AgentScapeException ex) {
    reportError ("Problem sending HELLO to " + dest, ex);
}
```

Test set-up

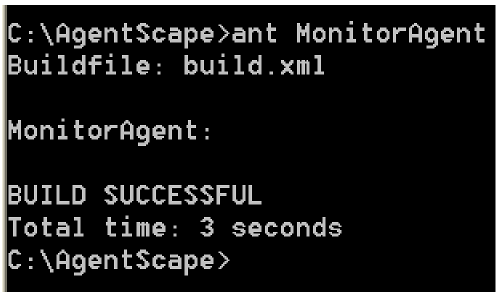
This tests have been updated so they are submitting the observations to a local eventbus. The event-bus server can be found at *'EventBus\contrib'* on the ALIVE SVN system.

1. Monitorized Agents
   1. Copy manifest file '*MonitorizedChatter.mf*' to $AGENTSCAPE/src/etc/manifests/examples
   2. Copy directory *monitor* $AGENTSCAPE\src\java\org\iids\aos\agents
   3. Add the following lines to build.xml file on $AGENTSCAPE. The lines can be found at *'minibuild.xml'* file.

```
<!-- MonitorAgent -->
 <target name="MonitorAgent">

                    <javac srcdir="build/java:src/java" destdir="build/classes" debug="on"
         includeAntRuntime="yes">
      <classpath>
       <fileset dir="lib/custom" includes="eventbus.jar"/>
       <fileset dir="lib/custom/eventbus_lib" includes="*.jar"/>
                    <fileset dir="lib/custom/event" includes="*.jar"/>
       <fileset dir="lib" includes="*.jar"/>
     </classpath>
     </javac>
   <!-- sender.jar -->
   <jar jarfile="lib/agents/MonitorizedChatter.jar"
        manifest="src/etc/manifests/agents/MonitorizedChatter.mf">
     <fileset dir="build/classes"
             includes="org/iids/aos/agents/**/*.class"/>
     <fileset dir="src/etc"
                          includes="org/iids/aos/agents/**/*.gif"/>

   </jar>
 </target>
```

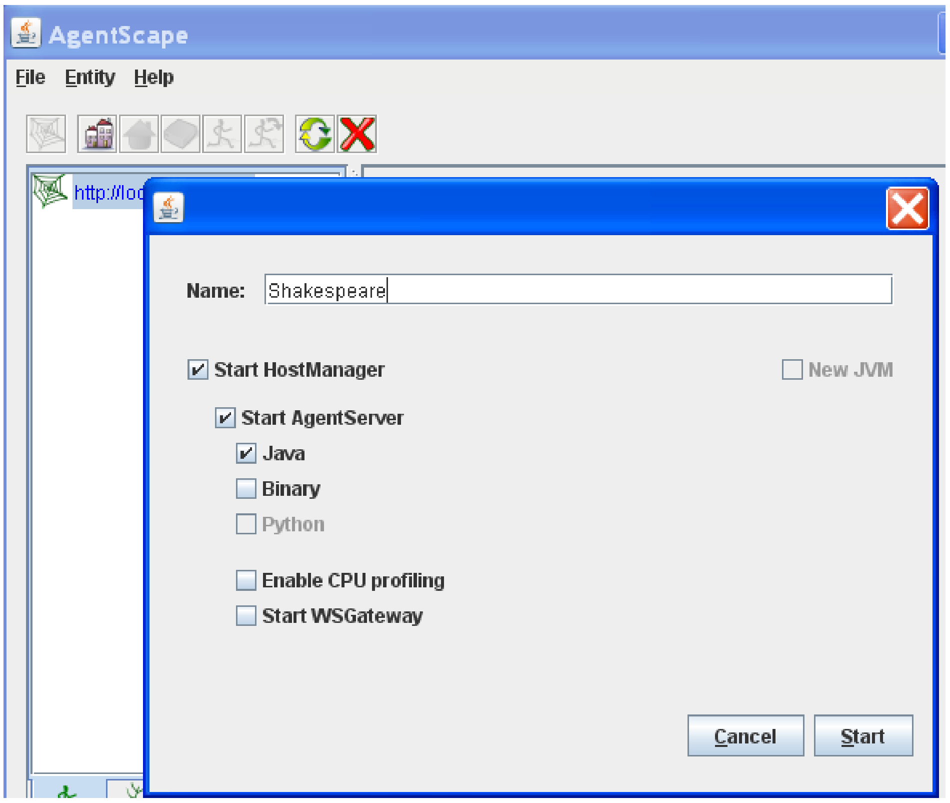
```
        <!-- MonitorAgent -->
 <target name="MonitorAgent">

                    <javac srcdir="build/java:src/java" destdir="build/classes" debug="on"
         includeAntRuntime="yes">
      <classpath>
       <fileset dir="lib/custom" includes="eventbus.jar"/>
       <fileset dir="lib/custom/eventbus_lib" includes="*.jar"/>
                    <fileset dir="lib/custom/event" includes="*.jar"/>
       <fileset dir="lib" includes="*.jar"/>
     </classpath>
     </javac>
   <!-- sender.jar -->
   <jar jarfile="lib/agents/MonitorizedChatter.jar"
        manifest="src/etc/manifests/agents/MonitorizedChatter.mf">
     <fileset dir="build/classes"
             includes="org/iids/aos/agents/**/*.class"/>
     <fileset dir="src/etc"
                          includes="org/iids/aos/agents/**/*.gif"/>

   </jar>
 </target>
```

4. Build the agent by using '*ant MonitorAgent*' command on $AGENTSCAPE directory.



```
C:\AgentScape>ant MonitorAgent
Buildfile: build.xml

MonitorAgent:

BUILD SUCCESSFUL
Total time: 3 seconds
C:\AgentScape>
```

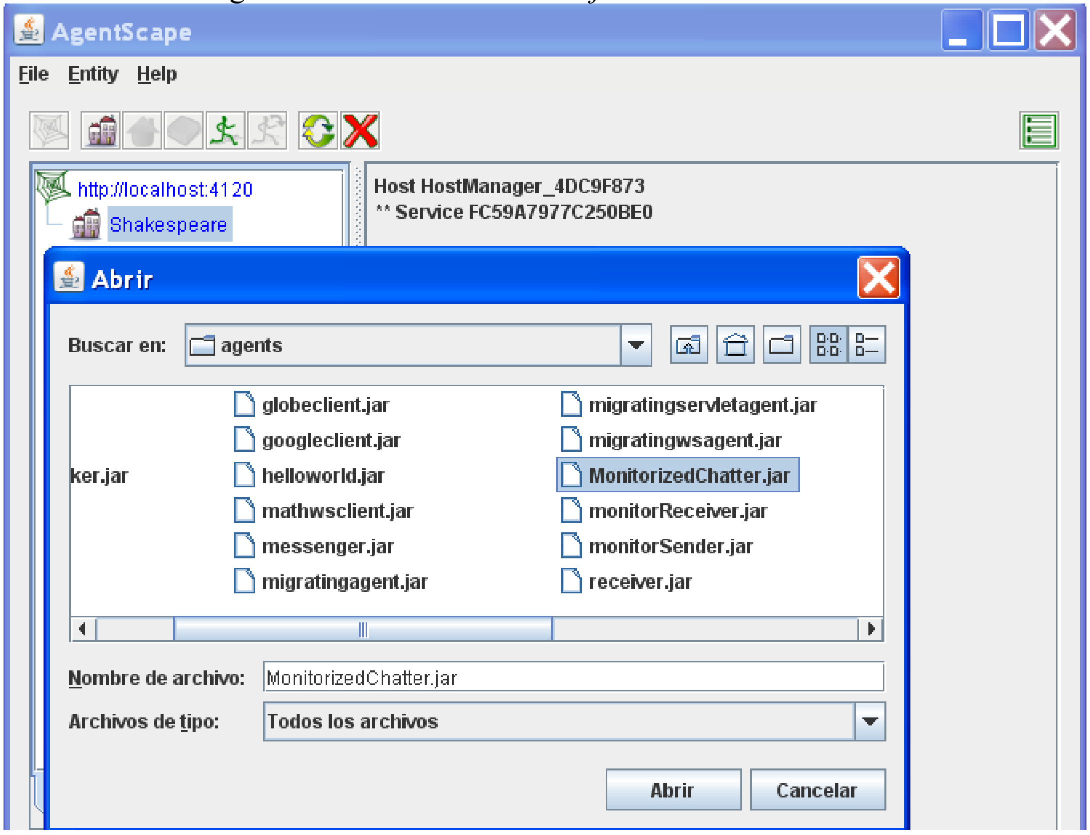
5. Start the platform (use '*java -Xmx1024m -jar lib/console.jar 1>run.log*')
6. Start a new container



Name: Shakespeare

[✓] Start HostManager
[ ] New JVM
[✓] Start AgentServer
[✓] Java
[ ] Binary
[ ] Python
[ ] Enable CPU profiling
[ ] Start WSGateway

7. Load the agent in '*MonitorizedChatter.jar*'. Load two different instances of the agent



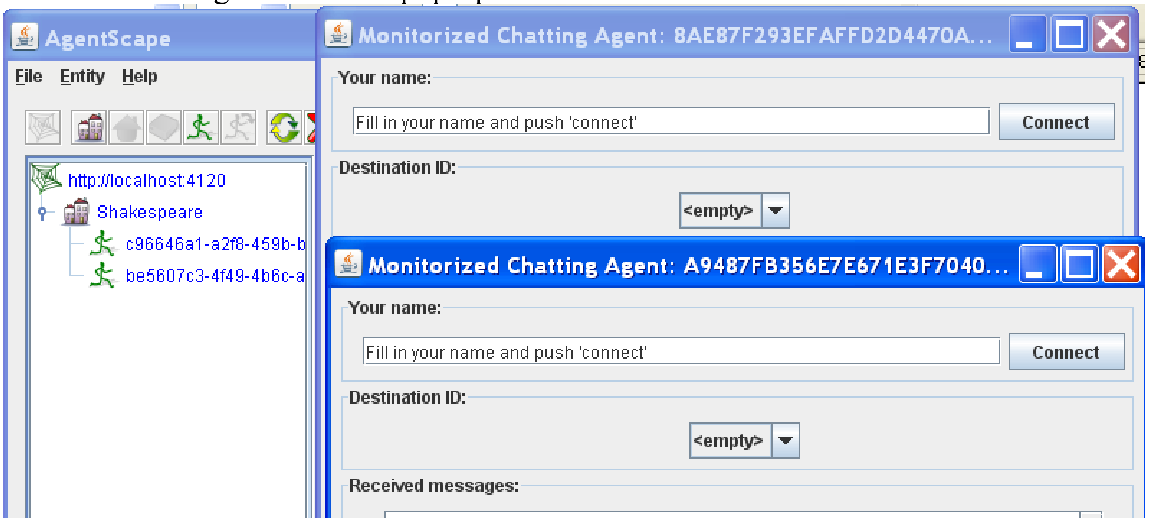
Host HostManager_4DC9F873
** Service FC59A7977C250BE0

Buscar en: agents

ker.jar
globeclient.jar
googleclient.jar
helloworld.jar
mathwsclient.jar
messenger.jar
migratingagent.jar
migratingservletagent.jar
migratingwsagent.jar
MonitorizedChatter.jar
monitorReceiver.jar
monitorSender.jar
receiver.jar

Nombre de archivo: MonitorizedChatter.jar
Archivos de tipo: Todos los archivos

8. Two Agent windows pop-up



AgentScape

File Entity Help

http://localhost:4120
Shakespeare
c96646a1-a2f8-459b-b
be5607c3-4f49-4b6c-a

Monitorized Chatting Agent: 8AE87F293EFAFFD2D4470A...

Your name:

Fill in your name and push 'connect' Connect

Destination ID:

<empty>

Monitorized Chatting Agent: A9487FB356E7E671E3F7040...

Your name:

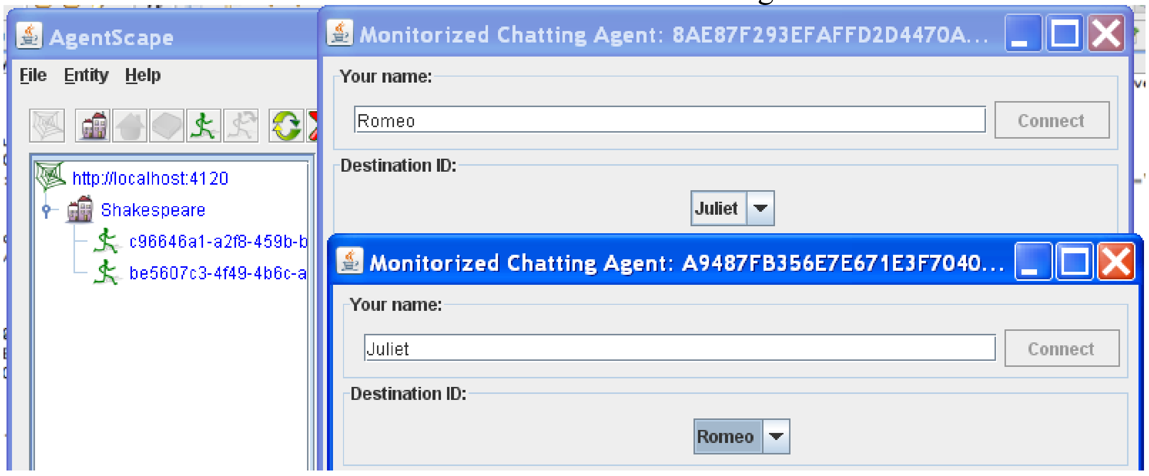
Fill in your name and push 'connect' Connect

Destination ID:

<empty>

Received messages:

9. Provide a different name to each instance of the agents and connect them.



AgentScape

File Entity Help

http://localhost:4120
Shakespeare
c96646a1-a2f8-459b-b
be5607c3-4f49-4b6c-a

Monitorized Chatting Agent: 8AE87F293EFAFFD2D4470A...

Your name:

Romeo Connect

Destination ID:

Juliet

Monitorized Chatting Agent: A9487FB356E7E671E3F7040...

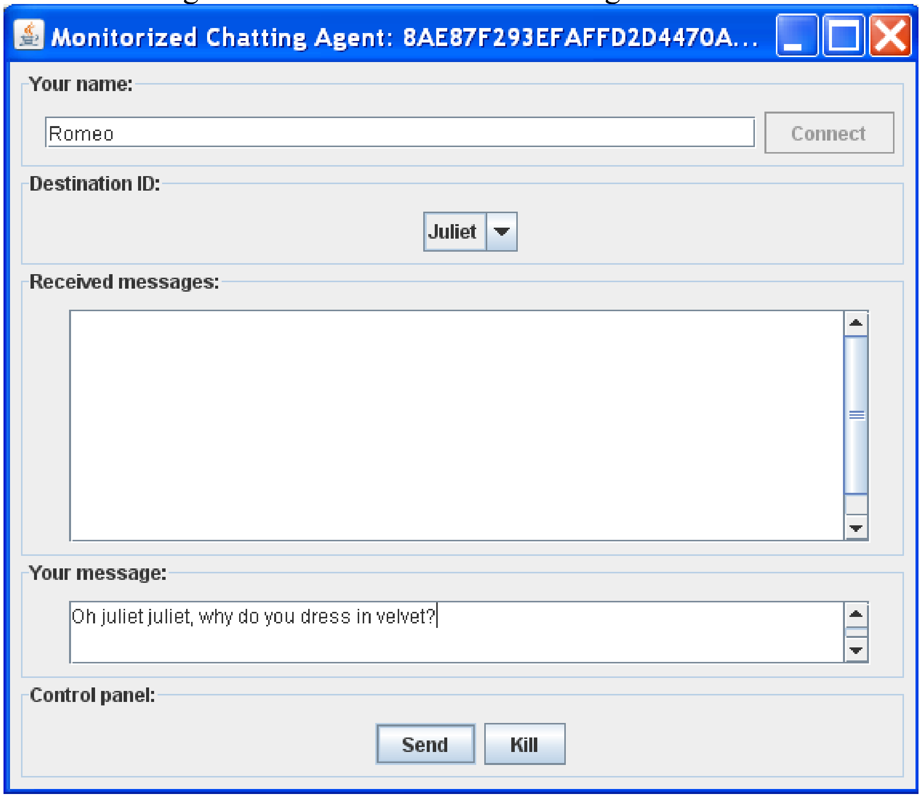
Your name:

Juliet Connect

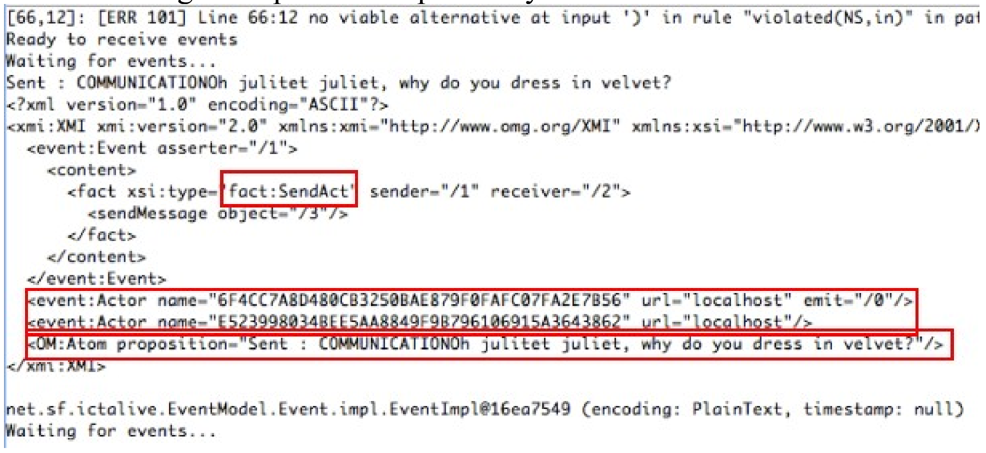
Destination ID:

Romeo

10. Send a message between the instances of the agents.


Monitorized Chatting Agent: 8AE87F293EFAFFD2D4470A...
Your name:
Romeo
Connect
Destination ID:
Juliet
Received messages:
Your message:
Oh juliet juliet, why do you dress in velvet?
Control panel:
Send
Kill

11. The message is captured and parsed by the local monitor



```
[66,12]: [ERR 101] Line 66:12 no viable alternative at input ')' in rule "violated(NS,in)" in pa
Ready to receive events
Waiting for events...
Sent : COMMUNICATIONOh julitet juliet, why do you dress in velvet?
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/
  <event:Event asserter="/1">
    <content>
      <fact xsi:type="fact:SendAct" sender="/1" receiver="/2">
        <sendMessage object="/3"/>
      </fact>
    </content>
  </event:Event>
  <event:Actor name="6F4CC7A8D480CB3250BAE879F0FAFC07FA2E7B56" url="localhost" emit="/0"/>
  <event:Actor name="E5239980348EE5AA8849F9B796106915A3643862" url="localhost"/>
  <OM:Atom proposition="Sent : COMMUNICATIONOh julitet juliet, why do you dress in velvet?"/>
</xmi:XMI>

net.sf.ictalive.EventModel.Event.impl.EventImpl@16ea7549 (encoding: PlainText, timestamp: null)
Waiting for events...
```