Solution 4

Java Message Service

1 Overview

We give here the complete solution for Exercise 4.

2 Getting, Compiling and Running the Application

2.1 Compilation

The complete source code is available for download at the exercise page of the course web site. The archive contains an ant (*http://ant.apache.org*) build file that can be used to compile the sources and generate the Java documentation. To do this, simply type in the *Ex4* directory[1]:

```
ant
```

2.2 Starting the JMS Provider

Please have a look at the statement of Exercise 4 in order to know how to start the *OpenJMS* provider.

2.3 Starting the Client

To start the client, do the following in a console in the *Ex4* directory (*Readme.txt* file):

java -classpath ./build/classes/:./openjms-0.7.7-alpha-3/lib/jms-1.1.jar:./openjms-0.7.7-alpha-3/lib/openjms-0.7.7-alpha-3.jar ch.epfl.lpd.ids.client.ChatClient org.exolab.jms.jndi.InitialContextFactory rmi://localhost:1099 JmsTopicConnectionFactory

If the JMS provider is correctly started, you can use the chat application.

3 Modifications

If you look at the different packages, you can see that the server package is not present anymore as well as the classes implementing a blocking queue.

3.1 IChatClient Interface

The only method that has changed in the `connect()` one. It accepts now the name of a chat room.

[1]Please use the latest version of ant.

3.2 ChatClient Class

Different methods have been changed in this class. First the `initialize()` method has been created in order to retrieve the `javax.naming.InitialContext` to lookup the `ConnectionFactory`.

The `connect()` method now create a `javax.jms.Connection`, a `javax.jms.Session`, which in turn creates a `javax.jms.Topic` (i.e., the chat room) and the `javax.jms.MessageProducer` (i.e., the publisher) and `javax.jms.MessageConsumer` (i.e., the subscriber) instances. We also set a `TopicListener` to receive the message asynchronously.

The creation of the chat room is done via the `createTopic()` method of the `javax.jms.Session`. The use of this method makes our code not portable. It is possible to have a code completely portable but this would imply using *physical topics* which are created administratively. Have a look in the *openjms.xml* configuration file to see how to create such topics and in the code to see how to retrieve the physical topics in the implementation.

When the client calls the `disconnect()` method, we close the `javax.jms.Connection` (this closes the `javax.jms.MessageConsumer`, the `javax.jms.MessageProducer` and the `javax.jms.Session`.

When the client wants to send a message, we create an `javax.jms.ObjectMessage`, put an instance or our own `Message` object in it and publish the message.

Finally, the new `main()` method creates a new instance of a `ChatClient`, initialize it and starts the GUI associated to it. We had a `ClientShutdownHook` in order to correctly disconnect the client if a *Ctrl-c* is typed in the shell.

3.3 ClientGUI Class

We have just put another textfield into this class in order to specify the name of the chat room we want to connect to. We also changed the call to the `connect()` method to give, as a parameter, the name of the chat room.

3.4 TopicListener Class

This class has been created in order to receive the JMS message asynchronously. It implements the `javax.jms.MessageListener` interface and implements the `onMessage()` method. This method is called when a new message is received.

When a new message is received, we extract the `IMessage` out of it and update the GUI accordingly.