Exercise 4
Java Message Service

1 Goal

The goal of this exercise is to teach you how to program a very simple chat application using the Java Message Service (JMS) APIs. As usual, you will be able to reuse your existing implementation in order to fulfill the goal of the exercise. The chat application you have to implement must be able to: (1) connect/disconnect to a JMS server, (2) create a chat room and (3) send messages to this chat room. The chat application does not need to show the list of the connected clients.

2 OpenJMS

If we want to launch a chat application that uses a JMS provider, we need this provider. There are different JMS providers available on the market (*OpenJMS*, *Sun Application Server*, *Joram*, ...). We will focus here on *OpenJMS* which is fairly simple to use.

To use *OpenJMS* first get it from *http://openjms.sourceforge.net/*. Once you have installed it, you can start/stop *OpenJMS* with the help of the *startup* and *shutdown* scripts, via a terminal.

Once the *OpenJMS* server is started, you can run your chat application by using the predefined *JmsTopicConnectionFactory*.

3 Guidelines

The application will contain one main class: (1) a chat client (`ChatClient`). The server will be the running instance of the JMS provider. As usual, different instances of a `ChatClient` can be launched on different hosts and will send messages to the JMS provider on a topic which represents the chat room. The different clients are able to connect/disconnect to the chat room and to see the different messages sent to this chat room. However, they do not need to list the clients that are currently connected to the chat room (this limitation is imposed by the specification of JMS)[1]. Please note that the clients must receive the messages in an asynchronous manner (i.e., via a `javax.jms.MessageListener`). We will focus on the publish/subscribe scheme in which a message is sent to all the clients by the JMS provider.

The implementation of our application can be done either using the different `javax.jms.Topic*` APIs (e.g., `javax.jms.TopicSession`, `javax.jms.TopicConnection`, `javax.jms.TopicConnectionFactory`, `javax.jms.TopicPublisher` and `javax.jms.TopicSubscriber`), as presented during the course or using the new features of *Domain Unification* [4]. We explain in the following how to use the new features of JMS 1.1.

3.1 Applications Overview

3.1.1 Chat Client

The user should be able to launch the client application via the following command:

```
java ch.epfl.lpd.ids.client.ChatClient <contextFactoryClassName> <providerURL>
                          <connectionFactoryName>
```

The following window should appear (Figure 1).

[1]We could overcome it, but only if we use the native APIs of the specific JMS provider.

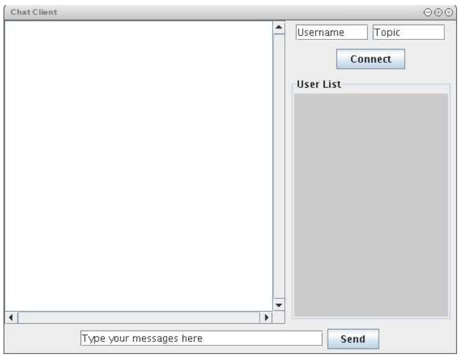

Figure 1: A Chat Client GUI

The user can connect/disconnect (via the `Connect` button) to a chat room (whose name is represented by the second textfield), send messages (via the `Send` button) and receive messages. Please note that a client **does not have** to list the other clients to the JMS provider.

The *contextFactoryClassName*, *providerURL* and *connectionFactoryName* will be used by the application to get a reference to the JMS provider. The first two parameters represent, in a way, the yellow pages needed to get the JMS provider reference. The third parameter represents the name of the factory that will be used to create a connection for sending messages.

3.2 Design Overview

The architecture of the applications is fairly simple (see Figure 2).

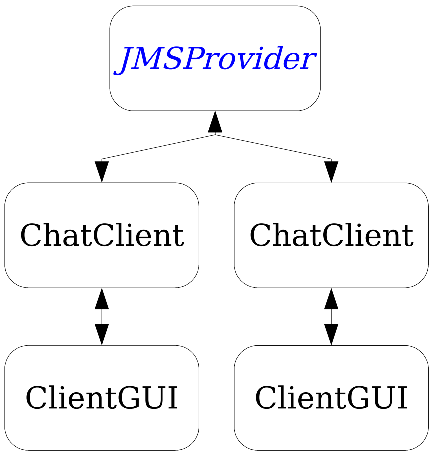

Figure 2: JMS Chat Architecture

The different instances of the `ChatClient` will communicate through the JMS provider by sending `Message`.

3.3 Enhancing your Previous Application

3.3.1 Contacting the Yellow Pages

When starting the `ChatClient`, the first thing to do is to contact the yellow pages in order to retrieve a `javax.jms.ConnectionFactory` instance.

In the course example, this factory is created manually using directly the native implementation of the JMS provider. We want our chat application to be independent from the JMS provider and to call methods only of the JMS APIs. This is why we have to go through the Java Naming Directory Interface (JNDI) which represents the yellow pages.

To do so, you have to create an `javax.naming.InitialContext`. The newly created instance of this `Context` should take as parameters the *contextFactoryClassName* and the *providerURL*. This can easily be done by passing as a parameter in the `javax.naming.InitialContext` constructor, a correctly initialized `java.util.Hashtable`:

Hashtable	
key	value
Context.INITIAL_CONTEXT_FACTORY	*contextFactoryClassName*
Context.PROVIDER_URL	*providerURL*

For instance, the *contextFactoryClassName* for OpenJMS is:
org.exolab.jms.jndi.InitialContextFactory and its provider URL is: *rmi://localhost:1099*.

Once you have an `javax.naming.InitialContext`, you can now lookup (via the *lookup()* method) the *connectionFactoryName* (i.e., `JmsTopicConnectionFactory`). The `javax.jms.ConnectionFactory` will be used, as seen during the course, to create a `javax.jms.Connection` representing a connection between the client and the JMS provider.

3.3.2 Sending/Receiving Messages

Now that you have a `javax.jms.ConnectionFactory`, you can follow the guidelines of [4] in order to create a `javax.jms.Connection`, a `javax.jms.Session`, a `javax.jms.MessageProducer` and a `javax.jms.MessageConsumer`. Do not forget to set a `javax.jms.MessageListener` for the `javax.jms.MessageConsumer`.

For sending messages, you will use `javax.jms.ObjectMessage` and put, inside them, your own `IMessage`.

3.4 Compiling your Applications

To be able to compile your application, you will need to incorporate, in your classpath, the JMS APIs. Those APIs are stored in a jar file: *openjms-0.7.7-alpha-3/lib/jms-1.1.jar*.

3.5 Running your Applications

First start the *OpenJMS* provider as presented in [3], then run your application according to Section 3.1. You have to be sure that the correct jar files of the implementation of the JMS provider you use are in your classpath.

Namely, you need the following jar files:

- openjms-0.7.7-alpha-3.jar
- jms-1.1.jar

4 Due

You have to give, for May 30th, the complete source code of your application. You can send your code, **without** the compiled classes, in a **.zip** file archive (or **.tgz**) to the following email address: *Sebastien.Baehni@epfl.ch*. The archive **has** to contain a script file used for compiling your code (either on Windows or on Linux). Your email should arrive before 1pm and its subject has to be: *IDS EX4*. Also, please put in the content of your mail the first name and name of your partners.

The application has to run correctly (i.e, without bugs) for you to receive 0.33 bonus points. If the code is well designed, well written and generic, you will receive additionally 0.33 points. Finally, if the code is well documented (i.e., javadoc preferably) you will get the final 0.33 points of the bonus.

References

[1] Java API Documentation. *http://java.sun.com/j2se/1.5.0/docs/api/index.html.*

[2] Java Message Service (JMS) API Documentation. *http://java.sun.com/javaee/5/docs/api/.*

[3] Open JMS. *http://openjms.sourceforge.net/.*

[4] JMS Version 1.1, change log. *http://java.sun.com/products/jms/changelog.html.*

[5] Java Message Service (JMS) Tutorial. *http://java.sun.com/products/jms/tutorial/1_3_1-fcs/doc/copyright.html.*