Tutorial: How to use RSVP for Java sockets

by *George Koprinkov*

This tutorial explains how to program Java sockets that are RSVP enabled. The reader is supposed to have knowledge of: Java Socket programming, CORBA programming and a basic understanding of RSVP

1. Introduction

Part of the QUAM/J platform is the IIOP RSVP daemon (IIOP_RSVPD). This daemon offers a generic interface to create RSVP sessions. Currently, the daemon is implemented for Windows 2000 and Linux. To see how to configure a system with RSVP, please read [3]

In this tutorial we use a simple client - server socket application to show how socket communication can use RSVP. The simple application is built from a client part that sends 10 messages "Hello world!" to a server that prints these messages to the console window.

The client is called "Sender" and the server is called "Receiver".

This document contains three sections:

- The section 'WITHOUT RSVP' shows how to implement a Java Socket Receiver- Sender application in standard way.

- The section 'WITH RSVP' shows how to extend this standard application to make use of RSVP.

- The section 'SENDER-RECEIVER APPLICATON' shows how to use the IIOP_RSVPD

2. WITHOUT RSVP

The following program shows how the Sender is implemented with standard Java Socket programming:

```
import java.io.*;
import java.net.*;

public class Sender extends java.lang.Object
{
  private static java.net.ServerSocket socket_ = null;
  private static java.net.Socket ssocket_ = null;
```

```
 public Sender()
 {
 }

  public static void main(String args[])
  {
  try
  {
     ssocket_= new java.net.Socket(args[0], 2996);
  }
  catch (java.io.IOException ex)
  {
   ex.printStackTrace();
  }

  OutputStream os = null;

  try
  {
     os = ssocket_.getOutputStream();
  }
  catch (IOException ex)
  {
     ex.printStackTrace();
  }

for(int ii=1;i<11;i++)
{
  try
  {
    os.write("Hello, World!".getBytes());
  }
  catch (IOException ex)
  {
    ex.printStackTrace();
  }
  try
  {
    Thread.currentThread().sleep(100);
  }
  catch (InterruptedException ex)
  {
    ex.printStackTrace();
  }
}

try
{
  ssocket_.close();
}
catch (java.io.IOException ex)
{
  ex.printStackTrace();
}
56. }
57. }
```

12- 19	Open client Socket, connected to the Receiver. Where host name or IP address of the Receiver is command prompt parameter (args[0]), and the port is 2996.
20-28	Get the output stream trough which will be send data ("Hello World") to the Receiver.
29-47	Sending Loop.
31-38	Send one "Hello World" to the Receiver.
39-46	Wait for 100 milliseconds between two sends.
48-55	Close the socket.

The following program shows how the Receiver is implemented with standard Java Socket programming:

```
1. import java.io.*;
2. public class Receiver extends java.lang.Object
3. {


4. public Receiver()
5. {
6. }


7. public static void main(String args[])
8. {

 9. /**
10. *
11. Try to create new Server Socket.
12. *
13. */
14. try
15. {
16. socket_= new java.net.ServerSocket(2996, 50);
17. }
18. catch (java.io.IOException ex)
19. {
20. ex.printStackTrace();
21. }


22. java.net.Socket csocket = null;
23. java.io.InputStream is = null;

24. try
25. {
26. System.out.println();
27. System.out.println("Socket is Ready");
28. System.out.println();

29. /**
30. *
31. Server Socket goes to accept mode and wait for connection (We expect only one connection to be made).
32. Result is Client Socket through which all communication with the Sender are made.
33. *
34. */

35. csocket = socket_.accept();
36. is = csocket.getInputStream();
37. }
38. catch (java.io.IOException ex)
39. {
40. ex.printStackTrace();
41. }
42. int recv = 1;
43. while (recv > 0)
44. {
45. byte buffer[] = new byte[13];
46. recv = 0;
47. try{
48. recv = is.read(buffer);
49. }
50. catch (java.io.IOException ex)
51. {
52. ex.printStackTrace();
53. }
54. if (recv >0 ) System.out.println("Received message is : " + new String(buffer,0, recv));
55. }

56. try{
57. csocket.getInputStream().close();
58. csocket.close();
59. socket_.close();
60. }
61. catch (java.io.IOException ex)
62. {
63. ex.printStackTrace();
64. }
65. }
66. }
```

14-21	Open Server socket on port 2996 with poll of 50 connections. Receiver expects only one connection from the Sender.
35	Put the Server socket in accept state. This state is blocking and will return when Connection between the Receiver an the Sender is established

36	Get input stream trough which Receiver receives the data.
43-55	Receiving loop
45	Define buffer where to be stored data from the input stream
46	define recv variable needed to check if Receiver received something. Example Line 48 recv taking value of received bytes.
56 - 64	Close all opened sockets.

3. WITH RSVP

The first step to enable the use of RSVP is to made sure that the ORB's thread is started not from the main thread. The ORB is needed for communication between the IIOP_RSVPD and the UpcallListener.

Example:

```
class orb_run extends java.lang.Thread
{
public orb_run()
{
}
public void run()
{
orb_.run();
}
}
private static orb_run thread_ = null;

/**
*
* Creates new Client
*
*/
public Sender()
{
/**
*
* Creating instance of the thread
*
*/
thread_ = new orb_run();
}
.
.
..
public static void main(String args[])
{
Sender r = new Sender(); //This creates the thread in which the ORB runs.
.
.
```

```
thread.start(); //later after initialization of all parameters and objects the ORB is started
```

The second step is to initialize the use of a specific implementation of the Java Socket and RSVP activation strategy. For the Windows 2000 platform it is necessary to use an internal iiop_rsvpd daemon, for Linux platform this daemon runs separately, and its reference is found in "/tmp/RSVPSessionFactory.ior".

For Windows 2000 we need our own implementation of the Java Socket because the standard implementation of the JVM does not allow to use the SIO_REUSEADDRESS flag on TCP Sockets.

Additionally, MS Windows 2000 QoS API does not allow a program to set the QoS through the socket which is different from the socket that will send QoS enabled data. Consequently, a QoS-aware Java Socket implementation has been realized for Windows 2000 which behaves like a standard one, but has additional methods for RSVP operations.

The IIOP_RSVPD daemon provides an interface to create an RSVP session object. On Windows 2000 platform, this session object has to run in the same Java Virtual Machine as the IIOP_RSVPD daemon and our specific implementation of the Java Socket to access the added methods for RSVP operations. On Linux the IIOP_RSVPD runs as a separate process.

The IIOP_RSVPD daemon provides an interface to RSVP Session factory, through which is created later RSVP sessions. RSVP activation strategy is the way how IIOP_RSVPD will be activated - internal in the same Java Virtual Machine or external separate process.

The following code fragment looks for two arguments ("-QIOPSocketFactory", -"RSVPStrategy")from the command line. First argument ("-QIOPSocketFactory"), if exist has to point to a Specific Socket implementation that will be used, if it is missing then is used a standard one. Second argument ("-RSVPStrategy") is used to point to the IIOP_RSVPD daemon implementation that will be used. If it is missing is used external daemon, which IOR has to be located under "/tmp" directory.

```
/**
*
Check arguments.
We are expecting to have at least one argument which is the fully qualified hostname (IP)
of the machine on which is running the Receiver.
*
*/

if (args.length < 1)
{
```

```
System.out.println("Usage: client <receiver host> [-QIOPSocketFactory <SocketFactory>] [-RSVPStrategy <RSVp Activation Strategy>]          ");
System.out.println("Result should be the exchange of rsvp messages between the receiver and sender");
return;
}
/**
*
Try to find specific Socket factory if not found we will use standard          one
*
*/
i = 1;
while (i < args.length)
{
if (args[i].equals("-QIOPSocketFactory"))
{
try
{
java.lang.Class socketImplFactoryClass = java.lang.Class.forName(args[i+1]);
java.net.SocketImplFactory serverSocketFactory = (java.net.SocketImplFactory)          socketImplFactoryClass.newInstance();
java.net.ServerSocket.setSocketFactory(serverSocketFactory);
java.net.Socket.setSocketImplFactory(serverSocketFactory);
System.out.println("QIOP SOCKET FACTORY INIT OK");
}
catch (java.lang.ClassNotFoundException ex)
{
ex.printStackTrace();
}
catch (java.lang.IllegalAccessException ex)
{
ex.printStackTrace();
}
catch (java.lang.InstantiationException ex)
{
ex.printStackTrace();
}
catch (java.io.IOException ex)
{
ex.printStackTrace();
}
}
i++;
}

/**
*
Preparation for initializing of Activation Strategy (RSVP Daemon)
*
*/
ActivationStrategy activationStrategy = null;
quamj.iiop_rsvpd.SessionFactory sessionFactory = null;
boolean dfactory = false;;

/**
```

```
*
Set ORBacus as a standard Java ORB
*
*/
java.util.Properties props = java.lang.System.getProperties();
props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");
props.put("org.omg.CORBA.ORBSingletonClass",
"com.ooc.CORBA.ORBSingleton");
java.lang.System.setProperties(props);

orb_ = null;
try
{
/** init ORB */
orb_ = org.omg.CORBA.ORB.init(args, props);
System.out.println("Client ORB initialization is OK!");
System.out.println();
}
catch (Exception ex)
{
ex.printStackTrace();
}

/**
*
Try to find if we will have to use specific RSVP Strategy (RSVP deamon)
or to use standart one (External stand alone runing)
*
*/
i = 1;
while (i < args.length)
{
if (args[i].equals("-RSVPStrategy"))
{
try
{
java.lang.Class activationStrategyClass =
java.lang.Class.forName(args[i+1]);
activationStrategy = (ActivationStrategy)
activationStrategyClass.newInstance();
System.out.println("RSVP STRATEGY INIT OK!");
}
catch (java.lang.ClassNotFoundException clNotFoundedEx)
{
System.err.println("ClassNotFoundException: " + args[i+1]);
}
catch (java.lang.IllegalAccessException illAccError)
{
System.err.println("IllegalAccessException: " + args[i+1]);
}
catch (java.lang.InstantiationException instEx)
{
System.err.println("Unable to load the RSVP activation strategy:          "+
args[i+1]);
}
sessionFactory = activationStrategy.activate(orb_);
111. the_factory_ = sessionFactory;
112. dfactory = true;
113. }
114. i++;
115. }

116. if(!dfactory)
117. {
118. /**
119. *
120. We didn't find specific RSVP Strategy we will try to standard one,
121. which IOR has to be located in /tmp directory
122. *
123. */
124. try
125. {
126. BufferedReader inFile = new BufferedReader (new
FileReader("/tmp/RSVPSessionFactory.ior"));
127. String iorStr = inFile.readLine();
128. inFile.close();
129. org.omg.CORBA.Object factoryObj = orb_.string_to_object(iorStr);
130. the_factory_ = SessionFactoryHelper.narrow(factoryObj);
131. }
132. catch(IOException ex)
133. {
134. System.err.println("Can't read from `" + ex.getMessage() + "'");
135. }
136. }
```

9-14	Check that at least two arguments are supplied. The first argument has to be the host name or IP address of the RSVP session, the second and third argument are specific Socket implementation and IIOP_RSVPD.
20-51	is looping trough the command parameters and try to find Specific Socket implementation which has to be supplied with "-QIOPSocketFactory" switch and next parameter then is the name of the Socket implementation. Most important to mention here is that Java allows ONLY one change of the standard Socket implementation and it has to be made before ANY socket is open.
27-31	Load specific Socket implementation and change standard implementation whit it (Line 29 - Change of Server Socket implementation and Line 30 - Change of Socket implementation)
65-80	Initialize and prepare ORB for running. This example uses ORBacus.
87-115	Loop for search through command line parameters for IIOP_RSVPD. This is given with "-RSVPStrategy" switch. If this switch is not found then dfactory flag is false and will be used external IIOP_RSVPD daemon.
92-113	When IIOP_RSVPD is found then it is activated.
116-136	Find and store reference to the external IIOP_RSVPD daemon. The reference IOR is red from "RSVPSessionFactory.ior" file located under /tmp directory (Lines 126-130).

The next Step is to initialize the UpcallListener Object which will receive information from the RSVP Session when RSVP messages arrive.

```
/**
 *
 * Create and start Upcall Listener.
 *
 */
listen_thread_ = new UpcallThread(orb_);
listen_thread_.start();

/**
 *
 * Obtain the callback object to which we will send RSVP messages.
 *
 */
UpcallListener callback = listen_thread_.get_listener();
```

The next Step is to create an RSVP Session which will be responsible for RSVP communication, arguments that are needed for creation are host name or IP address of the session, port of the session and protocol. The current implementation supports two protocols 6 (TCP) and 17 (UDP):

```
/**
 *
 * Ask session factory for a new Session
 * destAddress = first argument
 * destPort and protocolID - 6 for TCP *
 */
Session the_session = the_factory_.newSession(args[0], (short)2996, (short)6,
callback);
```

The next step is to ask the RSVP session to make a reservation after we connected both sides (Sender and Receiver), arguments that are needed for the Reservation - hostname or IP address of the session, port of the session, TSpec - traffic specification for the flow, RSpec - delay and latency specification, Time To Live (TTL), and AdSpec:

```
/**
 *
 * Ask Session for the connection and Parameters that we want for the
Reservation(TSpec, RSpec, TTL and AdSpec, ).
 *
 */
```

```
the_session.sender(csocket.getLocalAddress().getHostAddress(),(short)csocket.getLocalPort(),
tspec, rspec, (byte)12, adspec);
```

And finally we have to clean up after the sending (and receiving) has finished:

```
/**
 *
 * Shutdown the Callback Object (UpcallListener)
 *
 */
 listen_thread_.shutdown();

/**
 *
 * Shutdown the ORB
 *
 */
 orb_.destroy();
```

---

4. RSVP RECEIVER-SENDER APPLICATION

An example that shows how to combine Standard Java Socket programming and IIOP_RSVPD to use RSVP in Receiver can be found in iiop_rsvpd/tests/test1/src/quamj/iiop_rsvpd/test1/Receiver.java.

A program that shows how the SENDER is implemented with the use of IIOP_RSVPD can be found in iiop_rsvpd/tests/test1/src/quamj/iiop_rsvpd/test1/Receiver.java

5. REFERENCES:

[1] Elliote Rusty Harold,Java network programming , O'Reilly & Associates. ISBN 1-56592-870-9, 2nd Edition August 2000
[2] RFC 2205 (http://www.ietf.org/rfc/rfc2205.txt), RFC 2210 (http://www.ietf.org/rfc/rfc2210.txt), RFC 2212 (http://www.ietf.org/rfc/rfc2212.txt)
[3] George Koprinkov, Aart van Halteren, Qos Test Report, AMIDST/WP2/N024, December, 2001