The News Channel example using Jeremie

April 12, 2002

This document describes step by step how the News channel application has been developped with the Jeremie personality on top of Jonathan.

1 Introduction

The News channel example shows how to use the `EventChannel`[1] provided by Jonathan to build a one-to-many communication channel between a producer of news messages and an arbitrary number of news consumers. The example scenario proceeds as follows:

- a producer of news (`NewsSource`[2]) is started, it creates an event channel and registers it to Jeremie registry with a given symbolic name;
- new consumers (`NewsConsumer`[3]) can then be started with the name of the event channel to subscribe to.

`NewsTicker`[4] is the type of the event channel used in this example.

```
25
26 import java.rmi.*;
```

`NewsTicker` defines the interface through which news are delivered As any interface which can be invoked remotely via Jeremie, this interface extends `java.rmi.Remote`. Moreover, to be used as the type of an `EventChannel`, this interface must contains operations with no return value (i.e., void).

```
33 public interface NewsTicker extends java.rmi.Remote {
```

This operation sends several news as a array of strings

```
36    void latestNews(String msgs[]) throws RemoteException;
37 }
```

[1] see the package `org.objectweb.jeremie.libs.contexts.echannel`
[2] located in `examples/jeremie/newsChannel/source`
[3] located in `examples/jeremie/newsChannel/consumer`
[4] located in `examples/jeremie/newsChannel/source`

2 On the producer side

`NewsSource.java` contains the code for the source:

```
25
26 import org.objectweb.jonathan.apis.kernel.*;
27 import java.rmi.*;
28 import org.objectweb.jeremie.libs.services.registry.Naming;
29 import org.objectweb.jeremie.libs.binding.echannel.*;
32 import java.util.Random;
```

`NewsSource` is the class used to implement a source of news messages which will be delivered on an `EventChannel` to an arbitrary (and unknown) number of interested news consumers.

```
37 public class NewsSource {
```

In this simple example, the actual news messages are picked at random from the following list:

```
41    static String news[]=
42       { "Bill Gates  about Jonathan: \"a serious competitor\"",
43         "\"Jonathan is Y2K-compliant\" says Jonathan Inc. CEO B.A.M. Dumant",
44         "Jonathan Inc. up 10 points on NASDAQ",
45         "PGA Championship begins with rain",
46         "Pippen asks Rockets for a trade to Lakers",
47         "Shooting suspect returned to L.A. to face charges",
48         "Microsoft takes wraps off Embedded NT ",
49         "Customer satisfaction survey puts Dell on top ",
50         "NASA investigates space station sickness ",
51         "Sun to take wraps off Java-appliances chip",
52         "Sony developing plastic hard  drive disks",
53         "China to allow U.S. Air Force plane to land in Hong Kong"
54  };
```

A NewsSource is provided at creation time, the interface of the `EventChannel` on which it will produce news messages.

```
58  NewsTicker ticker;
59
60  public NewsSource(NewsTicker ticker) {
61  this.ticker=ticker;
62  }
```

This operation is used to start the periodic sending of random news on the event channel

```
66  public void produce() throws RemoteException {
67
68  String msgs[]=new String[1];
69  int newsCounter=0;
70  Random random=new Random(System.currentTimeMillis());
71
72  while(true) {
73  int index=random.nextInt();
74  if(index<0) index=-index;
75  msgs[0]=newsCounter++ + "\t" + news[index%news.length];
```

This operation is invoked on the `EventChannel` and will trigger the invocation of the method for each consumer (if any) which has registered itself to the channel.

```
 80  ticker.latestNews(msgs);
 81  try {
 82  Thread.currentThread().sleep(1500);
 83  } catch(Exception ex) {}
 84  }
 85  }
```

A `NewsSource` expects as argument upon initialization:

- the IP multicast address and port number to be used by the event channel;
- the name of the `EventChannel` which will be used to register it to a Name server;
- the name of the host on which a Name Server will be contacted;

```
 96  public static void main(String args[]) {
 97  if(args.length!=4) {
 98  System.err.println("Usage: NewsSource <address> <port> <channel name> <registry hos
 99  System.exit(1);
100  }
101
102  String address="";
103  int port=0;;
104  address=args[0];
105  try {
106  port=Integer.parseInt(args[1]);
107  } catch(NumberFormatException ex) {
108  System.err.println("Wrong port");
109  System.exit(1);
110  }
111
112  EventChannel channel;
113  try {
```

We access an `EventChannelFactory` and properly initialize the kernel by using this function:

```
116  EventChannelFactory channelFactory =
117             EventChannelFactoryFactory.newEventChannelFactory(NewsSource.class);
```

We request the creation of a new event channel. We need to provide the type (i.e., class name) of the class which will be used locally to instantiate stubs.

```
122          channel=channelFactory.newEventChannel(address,port,"NewsChannel");
```

We attempt to register the event channel with the name server whose host name has been provided.

```
126          Naming.rebind("//"+args[3]+"/"+args[2], channel);
127          System.out.println("Ready...");
```

We retrieve a consumer proxy object from the event channel.

```
130          NewsTicker ticker=(NewsTicker)channel.getConsumerProxy();
```

This local consumer object is supplied to the `NewsSource` object.

```
134  NewsSource producer=new NewsSource(ticker);
```

We start the emission of news messages

```
137  producer.produce();
138  } catch(Exception e ) {
139  System.err.println(e);
140  e.printStackTrace();
141  System.exit(1);
142  }
143  }
144 }
```

The `NewsChannel`[5] class is used during the creation of an `EventChannel` to specify the type of stubs which will used to marshal outgoing requests The name of this class needs to be specified to the Jeremie stub compiler to generate the corresponding stub class. It is worth noting that, contrary to client-server one-to-one bindings, the stub class used by a producer (`NewsChannel` in this case) bears no relationship with the stub class used by consumer objects.

```
33 public abstract class NewsChannel implements NewsTicker {}
```

3 On the consumer side

`NewsConsumer.java` contains the code for consumers:

```
25
26 import java.rmi.*;
27 import org.objectweb.jeremie.libs.services.registry.Naming;
28 import org.objectweb.jeremie.libs.binding.echannel.*;
```

`NewsConsumer` is the class representing a consumer of news. it implements the `NewsTicker` interface.

```
32 public class NewsConsumer implements NewsTicker {
```

On receipt of incoming event, the `NewsConsumer` just prints them on the screen.

```
36  public void latestNews(String msgs[])   {
37  for(int i=0;i<msgs.length;i++)
38  System.out.println(msgs[i]);
39  }
```

a NewsConsumer expects two arguments: (i) the name of the news channel it should subscribe to and (ii) the name of the registry host to contact.

[5] located in `examples/jeremie/newsChannel/source`

```
43  static public void main(String args[]) {
44  if(args.length!=2) {
45  System.err.println("Usage: NewsConsumer <existing channel name> <registry host>");
46  System.exit(1);
47  }
48
49  String channelName=args[0];
50
51  try {
```

This step is necessary to allow the downloading of Java classes during deserialization

```
54          System.setSecurityManager(new RMISecurityManager());
```

We look up the event channel name in the name server.

```
57          EventChannel channel=
58             (EventChannel) Naming.lookup("//" + args[1] + "/" + channelName);
```

The channel might not be available, i.e., no news source created and registered it to the name server.

```
61          if(channel==null) {
62             System.err.println("Channel "+ channelName + " not found");
63             System.exit(1);
64          }
```

If the required channel is there, we create a new consumer object and register it as a consumer to the event channel. If this step is successful, news should be delivered to the create a `NewsConsumer` object.

```
69          channel.addConsumer(new NewsConsumer());
70  } catch(Exception e) {
71  e.printStackTrace();
72  }
73  }
74 }
```

4 To compile and run your source and consumer

- In the `source` directory:
  - `make` compiles the code;
  - `make jrmiregistry` starts the name server;
  - `make source` starts the news channel source;
- In the `consumer` directory:
  - `make` compiles the code;
  - `make consumer` starts a consumer (you may start several consumers).