Launching Swing Applications from JBoss 3.0.0

Dimitri PISSARENKO
dimitri.pissarenko@gmx.net

August 14, 2002

1 Introduction

This paper presents an example of how an existing Java Swing application can be launched from a JBoss 3.0.0 server by means of an MBean.

2 The application

In this paper, the simple open-source graphical editor *GraphEd* (available at `http://jgraph.sourceforge.net/downloads.html` as of August 14, 2002) is used for demonstrating launching of Swing apps from JBoss.
The application source code consists of a single file `Editor.java`, the remaining components of the application are located in a jar file `jgraph.jar`.

3 Code Structure of the service

The code of the service is partially taken from [Turner(2002)]. The class diagram of the service is shown in figure 1. The MBean implementation class is called `GraphEdLauncher`. It has only two methods, `startService` and `stopService`, which start and stop the Swing application respectively (full code of the service is located in the appendix):

```
1  ...
2  public class GraphEdLauncher extends ServiceMBeanSupport implements
3    GraphEdLauncherMBean {
4     ...
5     protected void stopService() {
6         if (this.started) {
7             this.editor.stopService();
8             started = false;
9         }
10
11    }
12
13    protected void startService() {
14        if (!this.started) {
15            try {
16                this.editor = new LaunchableGraphEditor();
```

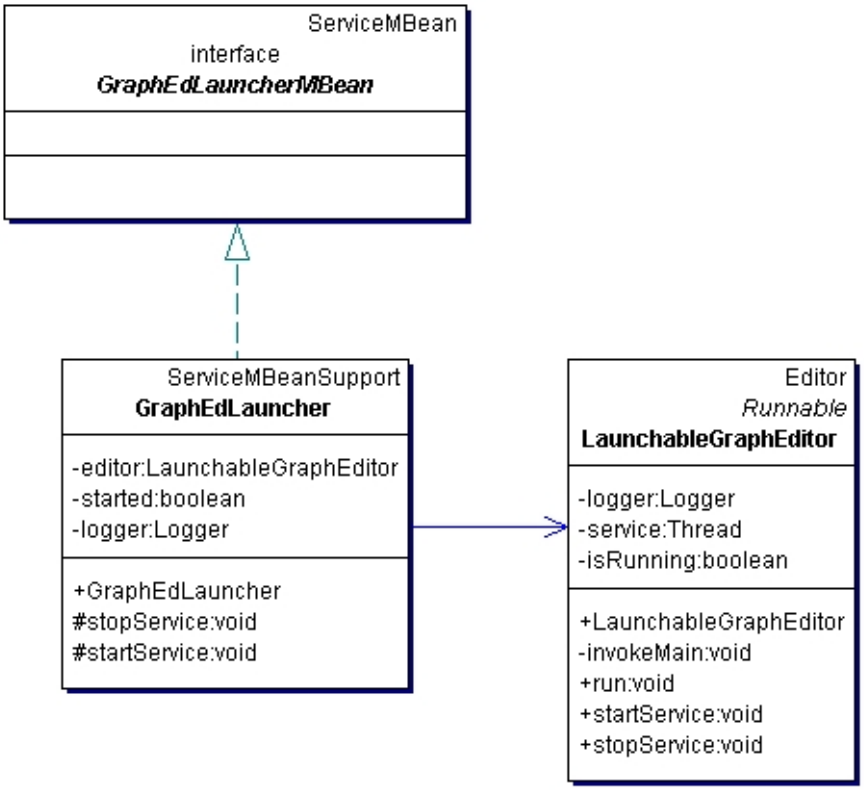

Figure 1: Class diagram of the service

```
17                this.editor.startService();
18            }
19            catch (Exception exception) {
20                logger.error("An error occured in startService", exception);
21            }
22            this.started = true;
23        }
24    }
25    ...
26 }
```

The interface `GraphEdLauncherMBean` does not have any method declarations, since the service does not have any parameters.
The class `LaunchableGraphEditor` is a sub-class of `Editor`, the main class of the application to launch. Subclassing was chosen instead of directly changing the `Editor` class for the reason of clarity; in this way it is more obvious what changes must be made to an existing Swing application so that it is JBoss-launchable.
We cannot use the `Editor` class directly for the following reason: Swing appli-

cations normally terminate the JVM at the end of the application. This is not desirable in the context of JBoss. If the Swing app terminates the JVM, the JBoss server is forced to shutdown. The normal behaviour one would expect from a JBoss service is that if it stops, other services are unaffected by this. Therefore, we must implement the Swing application as a thread. Further, we must provide `startService` and `stopService` methods, so that our MBean can start and stop the Swing application.
Let's have a look at the methods for starting the Swing app:

```
1  public class LaunchableGraphEditor extends Editor implements Runnable {
2  ...
3      public void startService() {
4          try {
5              this.service = new Thread(this);
6              this.service.start();
7          }
8          catch (Exception exception) {
9              this.logger.error("An error occured in startService", exception);
10         }
11     }
12
13     public void run() {
14         this.isRunning = true;
15         invokeMain();
16         while (this.isRunning);
17     }
18 ...
19 }
```

`startService` method creates a thread and starts it. After the thread is started, its `run` method is called. The only non-trivial part of this method is the `invokeMain` call. It is called so, because it contains the code, which normally would be in the `main` method of a Java app. Why not use the `main` method of the super-class of our Swing app? Have a look at it:

```
1  public class Editor extends JPanel implements GraphSelectionListener,
2                                                 KeyListener
3  {
4  ...
5    public static void main(String[] args) {
6      JFrame frame = new JFrame("GraphEd");
7
8    /* We cannot use JFrame.EXIT_ON_CLOSE with JBoss!!! */
9    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10
11
12     frame.getContentPane().add(new Editor());
13     URL jgraphUrl = Editor.class.getClassLoader().getResource("jgraph.gif");
14     if (jgraphUrl != null) {
15       ImageIcon jgraphIcon = new ImageIcon(jgraphUrl);
16       frame.setIconImage(jgraphIcon.getImage());
```

```
17     }
18     frame.setSize(520, 390);
19     frame.show();
20   }
21 ...
22 }
```

On line 9 the close operation of the editor frame is set to `EXIT_ON_CLOSE`. That would terminate the JVM and shutdown the entire server.
So we can not transfer this behaviour to our MBean-driven Swing app. We simply take the code of the `main` method, and copy it into the `invokeMain` method of the `LaunchableGraphEditor` class. We replace `EXIT_ON_CLOSE` by `DISPOSE_ON_CLOSE` and leave everything else unchanged. The `invokeMain` method then has the following code:

```
1  private void invokeMain() {
2     JFrame frame = new JFrame("GraphEd");
3     frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
4
5     frame.getContentPane().add(new Editor());
6
7     URL jgraphUrl = Editor.class.getClassLoader().getResource("jgraph.gif");
8
9     if (jgraphUrl != null) {
10       ImageIcon jgraphIcon = new ImageIcon(jgraphUrl);
11       frame.setIconImage(jgraphIcon.getImage());
12    }
13    frame.setSize(520, 390);
14    frame.show();
15 }
```

The last method we must implement, is the `stopService` method. It simply sets the `isRunning` attribute to false, which terminates the execution of the `run` method (see above):

```
1 public void stopService() {
2    this.isRunning = false;
3 }
```

4 SAR file

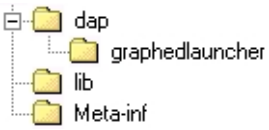

Figure 2: Directory structure of the SAR file

Having coded the MBean and the adapted Swing app, we must now prepare the SAR file. Its directory structure is shown in figure 2. The root directory

contains the `Editor` class file as well as the class files of all the classes declared in `Editor.java` (such as listeners). It is also important to put all the GIF image files used by the Swing App into the root directory. `dap/graphedlauncher` contains the class files `GraphEdLauncher.class`, `GraphEdLauncherMBean.class` and `LaunchableGraphEditor.class`. The directory is called `dap/graphedlauncher` after the package name `dap.graphedlauncher`.
In the `lib` directory, the `jgraph.jar` file of the original Swing application should be placed.

5 Deployment descriptor

The deployment descriptor `jboss-service.xml` looks like

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <service>
4   <mbean code="dap.graphedlauncher.GraphEdLauncher" name="Dap:service=graphEd">
5   <classpath codebase="lib" archives="jgraph.jar"/>.
6   </mbean>
7 </service>
```

It contains the `classpath` which contains a description, of where the class files of the Swing app should be searched for.

6 SAR deployment

The SAR file should be dropped into the `deploy` directory of the server. Shortly after that, the window of the Swing app should appear.

References

[Turner(2002)] G. Turner. Tiburon Enterprise Systems Newsletter - August 2002, 2002. URL `http://www.tiburon-e-systems.com/nl20020801/`. (URL accessed on August 14, 2002).

A Configuration information

This example was developed for *JDK 1.3* and *JBoss 3.0.0* under the *Windows 98* operating system. It may not function with other versions of the software.

B GraphEdLauncher.java

```
1 package dap.graphedlauncher;
2 import org.jboss.system.ServiceMBeanSupport;
3 import org.apache.log4j.Logger;
4
5 public class GraphEdLauncher extends ServiceMBeanSupport implements GraphEdLauncherMBean {
```

```
6     public GraphEdLauncher() {
7         this.editor = null;
8         this.started = false;
9         this.logger = Logger.getLogger(GraphEdLauncher.class);
10    }
11
12    protected void stopService() {
13        if (this.started) {
14            this.editor.stopService();
15            started = false;
16        }
17
18    }
19
20    protected void startService() {
21        if (!this.started) {
22            try {
23                this.editor = new LaunchableGraphEditor();
24                this.editor.startService();
25            }
26            catch (Exception exception) {
27                logger.error("An error occured in startService", exception);
28            }
29            this.started = true;
30        }
31    }
32
33    private LaunchableGraphEditor editor;
34    private boolean started;
35    private Logger logger;
36 }
```

C GraphEdLauncherMBean.java

```
1 package dap.graphedlauncher;
2 import org.jboss.system.ServiceMBean;
3
4 public interface GraphEdLauncherMBean extends ServiceMBean {
5 }
```

D LaunchableGraphEditor.java

```
1 package dap.graphedlauncher;
2
3 import Editor;
4 import org.apache.log4j.Logger;
5 import javax.swing.JFrame;
6 import java.net.URL;
7 import javax.swing.ImageIcon;
```

```
8
9  public class LaunchableGraphEditor extends Editor implements Runnable {
10     public LaunchableGraphEditor() {
11         this.logger = Logger.getLogger(LaunchableGraphEditor.class);
12     }
13
14     private void invokeMain() {
15         // Construct Frame
16         JFrame frame = new JFrame("GraphEd");
17         // Set Close Operation to Exit
18         //frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19         frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
20
21         // Add an Editor Panel
22         frame.getContentPane().add(new Editor());
23         // Fetch URL to Icon Resource
24         URL jgraphUrl = Editor.class.getClassLoader().getResource("jgraph.gif");
25         // If Valid URL
26         if (jgraphUrl != null) {
27             // Load Icon
28             ImageIcon jgraphIcon = new ImageIcon(jgraphUrl);
29             // Use in Window
30             frame.setIconImage(jgraphIcon.getImage());
31         }
32         // Set Default Size
33         frame.setSize(520, 390);
34         // Show Frame
35         frame.show();
36     }
37
38     public void run() {
39         this.isRunning = true;
40         invokeMain();
41         while (this.isRunning);
42     }
43
44     public void startService() {
45         try {
46             this.service = new Thread(this);
47             this.service.start();
48         }
49         catch (Exception exception) {
50             this.logger.error("An error occured in startService", exception);
51         }
52     }
53
54     public void stopService() {
55         this.isRunning = false;
56     }
57
```

```
58      private Logger logger;
59      private Thread service;
60      private boolean isRunning;
61  }
```