CHAPTER 18

Example: Flashing Clocks

One of the early promises of Jini was that it would find its way into all sorts of devices that could advertise their presence. However, Jini does not run on the really small JVMs such as the KVM. But if it could, how would it be used?

Most people have a number of electronic clocks in their homes: alarm clocks, a clock on the oven, another clock on the microwave, and so on. When the electricity resumes after a power failure, all of these clocks start flashing, and you have to go to each one and reset it manually. Wouldn't it be nice if you had to reset only one of these clocks (or if it got a value from a time server somewhere) and all the others reset themselves from it?

In this chapter, we'll look at this "flashing clocks" problem from a Jini viewpoint, to see what a Jini solution might look like. This example uses `JoinManager` and `ServiceDiscoveryManager` to advertise and discover services.

Note On my site (`http://jan.netcomp.monash.edu.au/internetdevices/upnp/upnp-more-programming.html`) is an alternative solution using UPnP, a middleware system that is gaining more ground in the area of small devices than Jini is, probably due to lighter resource requirements and an active coordinating body.

Timer

Each clock is available as a service called a `Timer`. A timer has methods to get and set the time, and in addition it knows if it has a valid time or if it has an invalid time (and so should be shown flashing). A timer can have its time set; when it does, it becomes valid and the display stops flashing.

The interface for a timer is as follows:

```
/**
 * Timer service
 */
package clock.service;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Date;
public interface Timer extends Remote {
    public void setTime(Date t) throws RemoteException;
```

```
    public Date getTime() throws RemoteException;
    public boolean isValidTime() throws RemoteException;
}
```

TickerTimer

I'll give two implementations of this service. The first is given in this section and is the "dumb" one: when the timer starts, it guesses at a start time and enters an invalid state. It uses a separate thread (a "ticker") to keep increasing its time every second (approximately). When its time is set, it becomes valid, but it will probably drift from the correct time due to its use of sleep to keep changing the time.

The dumb ticker timer is as follows:

```
package clock.service;
import java.util.Date;
import java.rmi.RemoteException;
public class TickerTimer implements Timer {
    private Date time;
    private boolean isValid;
    private Ticker ticker;
    /**
     * Constructor with no starting time has
     * invalid timer and any time
     */
    public TickerTimer() {
        time = new Date(0);
        isValid = false;
        ticker = new Ticker(time);
        ticker.start();
    }
    public TickerTimer(Date t) {
        time = t;
        isValid = true;
        ticker = new Ticker(time);
        ticker.start();
    }

    public void setTime(Date t) {
        System.out.println("Setting time to " + t);
        time = t;
        isValid = true;
        if (ticker != null) {
            ticker.stopRunning();
        }
        ticker = new Ticker(time);
        ticker.start();
    }
```

```
    public Date getTime() {
        return ticker.getTime();
    }
    public boolean isValidTime() {
        if (isValid) {
            return true;
        } else {
            return false;
        }
    }
}
class Ticker extends Thread {
    private Date time;
    private boolean keepRunning = true;
    public Ticker(Date t) {
        time = t;
    }
    public Date getTime() {
        return time;
    }
    public void run() {
        while (keepRunning) {
            try {
                sleep(1000);
            } catch(InterruptedException e) {
            }
            time = new Date(time.getTime() + 1000);
        }
    }
    public void stopRunning() {
        keepRunning = false;
    }
}
```

ComputerTimer

This section describes the second implementation of a timer. This timer uses the computer's internal clock to always return the correct time on request. It is always valid.

```
package clock.service;
import java.util.Date;
import net.jini.core.event.*;
import java.util.Vector;
import java.rmi.RemoteException;
public class ComputerTimer implements Timer {
    public ComputerTimer() {
    }
    public void setTime(Date t) {
```

```
        // void
    }
    public Date getTime() {
        return new Date();
    }
    public boolean isValidTime() {
        return true;
    }
}
```

ClockFrame

To make the clocks more visual, we can put the timers into a Swing frame and watch them ticking away. The following code is based on that of Satoshi Konno for UPnP.

A clock pane is as follows:

```
/**
 * Copyright (C) Satoshi Konno 2002
 * Minor changes Jan Newmarch
 */
package clock.clock;
import clock.device.*;
import java.io.*;
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;
import javax.imageio.ImageIO;
import java.rmi.RemoteException;
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
public class ClockPane extends JPanel
{
    private ClockDevice clockDev;
    private Color lastBlink = Color.BLACK;
    private DateFormat dateFormat = new SimpleDateFormat("kk:mm:ss");
    public ClockPane(ClockDevice clockDev)
    {
        this.clockDev = clockDev;
        loadImage();
        initPanel();
    }

    ////////////////////////////////////////////////
    //        Background
    ////////////////////////////////////////////////
```

```
    private final static int DEFAULT_WIDTH = 200;
    private final static int DEFAULT_HEIGHT = 60;
    private final static String CLOCK_PANEL_IMAGE = "images/clock.jpg";
    private final static String CLOCK_PANEL_IMAGE_FILE = "resources/" +
CLOCK_PANEL_IMAGE;

    private Image panelmage;
    private int imageWidth = DEFAULT_WIDTH;
    private int imageHeight = DEFAULT_HEIGHT;

    private void loadImage()
    {
        // Try to get the image form the local file system
        File f = new File(CLOCK_PANEL_IMAGE_FILE);
        try {
            panelmage = ImageIO.read(f);
            imageWidth = ((BufferedImage) panelmage).getWidth();
            imageHeight = ((BufferedImage) panelmage).getHeight();
            return;
        }
        catch (Exception e) {
            // Not in local file system
        }
        // Try to get the image from classpath jar files
        java.net.URL url = getClass().getClassLoader().getRe-
source(CLOCK_PANEL_IMAGE);
        if (url != null) {
            ImageIcon icon = new ImageIcon(url);
            panelmage = icon.getImage();
            imageWidth = icon.getIconWidth();
            imageHeight = icon.getIconHeight();
            return;
        }

        // couldn't find an image, leave panelmage as null
    }

    private Image getPaneImage()
    {
        return panelmage;
    }

    ////////////////////////////////////////////
    //        Background
    ////////////////////////////////////////////

    private void initPanel()
    {
```

```
        Image panelmage = getPaneImage();
        setPreferredSize(new Dimension(imageWidth, imageHeight));
    }

    ////////////////////////////////////////////
    //        Font
    ////////////////////////////////////////////

    private final static String DEFAULT_FONT_NAME = "Lucida Console";
    private final static int DEFAULT_TIME_FONT_SIZE = 48;
    private final static int DEFAULT_DATE_FONT_SIZE = 18;
    private final static int DEFAULT_SECOND_BLOCK_HEIGHT = 8;
    private final static int DEFAULT_SECOND_BLOCK_FONT_SIZE = 10;

    private Font timeFont = null;
    private Font dateFont = null;
    private Font secondFont = null;

    private Font getFont(Graphics g, int size)
    {
        Font font = new Font(DEFAULT_FONT_NAME, Font.PLAIN, size);
        if (font != null)
            return font;
        return g.getFont();
    }

    private Font getTimeFont(Graphics g)
    {
        if (timeFont == null)
            timeFont = getFont(g, DEFAULT_TIME_FONT_SIZE);
        return timeFont;
    }

    private Font getDateFont(Graphics g)
    {
        if (dateFont == null)
            dateFont = getFont(g, DEFAULT_DATE_FONT_SIZE);
        return dateFont;
    }

    private Font getSecondFont(Graphics g)
    {
        if (secondFont == null)
            secondFont = getFont(g, DEFAULT_SECOND_BLOCK_FONT_SIZE);
        return secondFont;
    }
```

```
////////////////////////////////////////////
//        paint
////////////////////////////////////////////

private void drawClockInfo(Graphics g)
{
    int winWidth = getWidth();
    int winHeight = getHeight();

    boolean valid = false;
    try {
        valid = clockDev.isValidTime();
    } catch(RemoteException e) {
        // valid is already false
    }
    if (valid) {
        g.setColor(Color.BLACK);
    } else {
        if (lastBlink == Color.WHITE) {
            g.setColor(Color.BLACK);
            lastBlink = Color.BLACK;
        } else {
            g.setColor(Color.WHITE);
            lastBlink = Color.WHITE;
        }
    }

    //// Time String ////
    Date now = null;
    try {
        now = clockDev.getTime();
    } catch(RemoteException e) {
        now = new Date(0);
    }
    String timeStr = dateFormat.format(now);

    Font timeFont = getTimeFont(g);
    g.setFont(timeFont);

    FontMetrics timeFontMetric = g.getFontMetrics();
    Rectangle2D timeStrBounds = timeFontMetric.getStringBounds(timeStr, g);

    int timeStrWidth = (int)timeStrBounds.getWidth();
    int timeStrHeight = (int)timeStrBounds.getHeight();
    int timeStrX = (winWidth-timeStrWidth)/2;
    int timeStrY = (winHeight+timeStrHeight)/2;
    int timeStrOffset = timeStrHeight/8/2;
```

```
            g.drawString(
                        timeStr,
                        timeStrX,
                        timeStrY);

            //// Date String ////

            String dateStr = "Time";

            Font dateFont = getDateFont(g);
            g.setFont(dateFont);

            FontMetrics dateFontMetric = g.getFontMetrics();
            Rectangle2D dateStrBounds = dateFontMetric.getStringBounds(dateStr, g);

            g.drawString(
                        dateStr,
                        (winWidth-(int)dateStrBounds.getWidth())/2,
                        timeStrY-timeStrHeight-timeStrOffset);

        }

        private void clear(Graphics g)
        {
            g.setColor(Color.GRAY);
            g.clearRect(0, 0, getWidth(), getHeight());
        }


        private void drawPanelImage(Graphics g)
        {
            if (getPaneImage() == null) {
                return;
            }
            g.drawImage(getPaneImage(), 0, 0, null);
        }

        public void paint(Graphics g)
        {
            clear(g);
            drawPanelImage(g);
            drawClockInfo(g);
        }
    }
```

Here's a clock frame:

```
/**
 * Copyright (C) Satoshi Konno 2002-2003
 * Minor changes Jan Newmarch
 */
package clock.clock;
import clock.device.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ClockFrame extends JFrame implements Runnable, WindowListener
{
    private final static String DEFAULT_TITLE = "Sample Clock";
    private ClockDevice clockDev;
    private ClockPane clockPane;
    public ClockFrame(ClockDevice clockDev) {
        this(clockDev, DEFAULT_TITLE);
    }
    public ClockFrame(ClockDevice clockDev, String title)
    {
        super(title);
        this.clockDev = clockDev;

        getContentPane().setLayout(new BorderLayout());

        clockPane = new ClockPane(clockDev);
        getContentPane().add(clockPane, BorderLayout.CENTER);

        addWindowListener(this);

        pack();
        setVisible(true);
    }

    public ClockPane getClockPanel()
    {
        return clockPane;
    }

    public ClockDevice getClockDevice()
    {
        return clockDev;
    }

    ////////////////////////////////////////////////
    //        run
    ////////////////////////////////////////////////
```

```
    private Thread timerThread = null;

    public void run()
    {
        Thread thisThread = Thread.currentThread();

        while (timerThread == thisThread) {
            // getClockDevice().update();
            getClockPanel().repaint();
            try {
                Thread.sleep(1000);
            }
            catch(InterruptedException e) {}
        }
    }

    public void start()
    {
        // clockDev.start();

        timerThread = new Thread(this);
        timerThread.start();
    }

    public void stop()
    {
        // clockDev.stop();
        timerThread = null;
    }

    ////////////////////////////////////////////////
    //        main
    ////////////////////////////////////////////////

    public void windowActivated(WindowEvent e)
    {
    }

    public void windowClosed(WindowEvent e)
    {
    }

    public void windowClosing(WindowEvent e)
    {
        stop();
        System.exit(0);
    }
```

```
    public void windowDeactivated(WindowEvent e)
    {
    }

    public void windowDeiconified(WindowEvent e)
    {
    }

    public void windowIconified(WindowEvent e)
    {
    }

    public void windowOpened(WindowEvent e)
    {
    }
}
```

TickerTimer Driver

A driver for the ticker timer in the preceding frame is as follows:

```
package clock.clock;
import clock.device.*;
import clock.service.*;
public class TickerClock {

    public static void main(String args[])
    {
        ClockDevice clockDev = new ClockDevice();
        clockDev.setTimer(new TickerTimer());
        ClockFrame clock;
        if (args.length > 0) {
            clock= new ClockFrame(clockDev, args[0]);
        } else {
            clock = new ClockFrame(clockDev);
        }
        clock.start();
    }
}
```

This driver can be run with the following:

```
java clock.clock.TickerClock "Ticking Clock"
```

ComputerTimer Driver

A driver for the computer timer in the preceding frame is as follows:

```
package clock.clock;
import clock.device.*;
import clock.service.*;
public class ComputerClock {

    public static void main(String args[])
    {
        ClockDevice clockDev = new ClockDevice();

        clockDev.setTimer(new ComputerTimer());
        ClockFrame clock;
        if (args.length > 0) {
            clock= new ClockFrame(clockDev, args[0]);
        } else {
            clock = new ClockFrame(clockDev);
        }
        clock.start();    }
}
```

This driver can be run with the following:

```
java clock.clock.ComputerClock "Computer Clock"
```

Multiple Clocks

Two (or more) clocks can be started. If ticking clocks are started, then they will all be flashing. Once a computer clock is started, though, the clocks will discover each other. Either the computer clock will discover ticking clocks and reset them, or the ticking clocks will discover the computer clock and reset themselves. I don't know which scenario occurs, and it doesn't matter.

When running, the two clocks look like Figure 18-1.

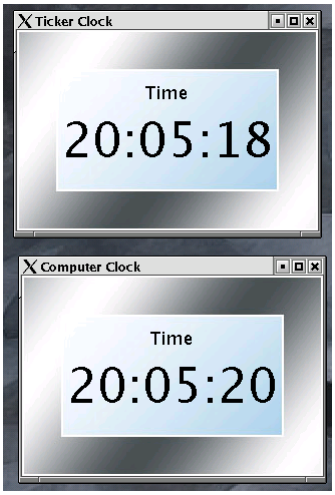

Figure 18-1. *A ticking clock and a computer clock*

ClockDevice

The final part of the code for each clock is to advertise each timer as a Jini service, to try to locate other timer services and to listen to events from each one. This is handled by the clock device (really, it is what we have been calling a Jini server; we have just adopted the UPnP). The device has a timer installed by `setTimer()`, and it advertises this using a `JoinManager`. In the meantime, it uses a `ServiceDiscoveryManager` to find other timers.

```
package clock.device;
import clock.service.*;
import java.io.*;
import java.util.Date;
import java.rmi.*;
import java.rmi.server.ExportException;
import net.jini.export.*;
import net.jini.jeri.BasicJeriExporter;
import net.jini.jeri.BasicILFactory;
import net.jini.jeri.tcp.TcpServerEndpoint;
import net.jini.lookup.JoinManager;
import net.jini.core.lookup.ServiceID;
import net.jini.discovery.LookupDiscovery;
import net.jini.core.lookup.ServiceRegistrar;
import java.rmi.RemoteException;
import net.jini.lookup.ServiceIDListener;
import net.jini.lease.LeaseRenewalManager;
import net.jini.discovery.LookupDiscoveryManager;
import net.jini.lookup.ServiceDiscoveryListener;
import net.jini.lookup.ServiceDiscoveryEvent;
import net.jini.core.lookup.ServiceTemplate;
```

```
import net.jini.core.lookup.ServiceItem;
import net.jini.lookup.ServiceDiscoveryManager;
import net.jini.discovery.LookupDiscoveryManager;
import net.jini.lease.LeaseRenewalManager;
import net.jini.lookup.LookupCache;
public class ClockDevice implements ServiceIDListener, ServiceDiscoveryListener {
    private Timer timer;
    public ClockDevice() {
        System.setSecurityManager(new RMISecurityManager());
        // Build a cache of all discovered clocks and monitor changes
        ServiceDiscoveryManager serviceMgr = null;
        LookupCache cache = null;
        Class [] classes = new Class[] {Timer.class};
        ServiceTemplate template = new ServiceTemplate(null, classes,
                                                       null);
        try {
            LookupDiscoveryManager mgr =
                new LookupDiscoveryManager(LookupDiscovery.ALL_GROUPS,
                                           null,  // unicast locators
                                           null); // DiscoveryListener
            serviceMgr = new ServiceDiscoveryManager(mgr,
                                                     new LeaseRenewalManager());
        } catch(Exception e) {
            e.printStackTrace();
            System.exit(1);
        }

        try {
            cache = serviceMgr.createLookupCache(template,
                                                 null,  // no filter
                                                 this); // listener
        } catch(Exception e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
    public void setTimer(Timer t) {
        timer = t;
        System.out.println("Our timer service is " + t);
        Exporter exporter = new BasicJeriExporter(TcpServerEndpoint.getInstance(0),
                                      new BasicILFactory());
        // export a Timer proxy
        Remote proxy = null;
        try {
            proxy = exporter.export(timer);
        } catch(ExportException e) {
            System.exit(1);
```

```
        }
        // Register with all lookup services as they are discovered
        JoinManager joinMgr = null;
        try {
            LookupDiscoveryManager mgr =
                new LookupDiscoveryManager(LookupDiscovery.ALL_GROUPS,
                                           null,  // unicast locators
                                           null); // DiscoveryListener
            joinMgr = new JoinManager(proxy, // service proxy
                                      null,  // attr sets
                                      this,  // ServiceIDListener
                                      mgr,   // DiscoveryManager
                                      new LeaseRenewalManager());
        } catch(Exception e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    public void serviceIDNotify(ServiceID serviceID) {
        // called as a ServiceIDListener
        // Should save the ID to permanent storage
        System.out.println("got service ID " + serviceID.toString());
    }
    public void serviceAdded(ServiceDiscoveryEvent evt) {
        // evt.getPreEventServiceItem() == null
        ServiceItem postItem = evt.getPostEventServiceItem();
        System.out.println("Service appeared: " +
                           postItem.service.getClass().toString());
        tryClockValidation((Timer) postItem.service);
    }
    public void serviceChanged(ServiceDiscoveryEvent evt) {
        ServiceItem preItem = evt.getPostEventServiceItem();
        ServiceItem postItem = evt.getPreEventServiceItem() ;
        System.out.println("Service changed: " +
                           postItem.service.getClass().toString());
    }
    public void serviceRemoved(ServiceDiscoveryEvent evt) {
        // evt.getPostEventServiceItem() == null
        ServiceItem preItem = evt.getPreEventServiceItem();
        System.out.println("Service disappeared: " +
                           preItem.service.getClass().toString());
    }
    private void tryClockValidation(Timer otherTimer) {
        try {
            if (timer.isValidTime() &&  ! otherTimer.isValidTime()) {
```

```
                // other clock needs to be set by us
                otherTimer.setTime(timer.getTime());
            } else if (! timer.isValidTime() && otherTimer.isValidTime()) {
                // we need to be set from the other clock
                timer.setTime(otherTimer.getTime());
            }
        } catch(RemoteException e) {
            // ignore other timer!
        }
    }
    public void setTime(Date t) throws RemoteException {
        timer.setTime(t);
    }
    public Date getTime()  throws RemoteException {
        return timer.getTime();
    }
    public boolean isValidTime()  throws RemoteException {
        return timer.isValidTime();
    }
}
```

Runtime Behavior

If several clocks are started, they will advertise themselves and also attempt to find other clocks. When one clock finds another, it tries to determine its state. If one is valid and the other is invalid, then either the valid one sets the time on the invalid one, or the invalid one gets the correct time from the valid one. Which procedure takes place depends on whether the valid one discovers the invalid one or vice versa; it doesn't matter, since the result is the same. Two valid clocks do nothing to each other, as do two invalid ones.

The Ant file `clock.clock.xml` runs a ticker clock, pauses 60 seconds, and then runs a computer clock. When one clock discovers the other, the ticker clock has its time reset.

Summary

This chapter looked at a common household problem to show how Jini could be used in such a situation. The solution uses many of the features of Jini that we have covered so far. When running, it gives a visual demonstration of Jini service discovery and invocation techniques.