Thesis Project: Initium Remote Job Submission Screensaver

Presented by
Francisco Castellanos
fsophisco@yahoo.com

A Thesis submitted to the Graduate Faculty of Fairfield University in partial fulfillment of the requirements for the degree of a Master of Science in the Electrical and Computer Engineering program

Advisor: Professor Douglas A. Lyon, Ph.D.

Electrical and Computer Engineering Department
Fairfield University,
Fairfield CT 06430
September 2006

TABLE OF CONTENTS

ABSTRACT

This thesis presents the Initium Remote Job Submission (IRJS), a screensaver system that enables volunteer-heterogeneous grid computing. The IRJS system takes advantage of computers during a period of user-computer quiescence. The IRJS middleware deploys jobs to non-geographically co-located clusters with decentralized look-up severs [32]. Computers are typically used between 40 and 60 hours out of a 168-hour week. This represents a 35% utilization. Screen-saver based *cycle scavenging* improves this number dramatically. Our prototype detects user-computer quiescence on a variety of platforms (Windows, Unix and Mac). We also provide a mechanism for releasing the computer back to the user quickly, when the user-computer quiescence period ends. Java enables the portability of most of our system, however, we have had to engage in native method development on multiple platforms in order to provide a framework. The overall goal of the IRJS system is to utilize computers executing the IRJS Screensaver, as volunteer resources for a grid-computing system.

1. PROBLEM STATEMENT

We are given heterogeneous, geographically-distributed compute servers, and seek to create a computational grid. Each Compute Server (CS) announces its availability to a Look Up Server (LUS), and provides benchmarks. The benchmark data describes computer resources, and it is used by the LUS to allocate tasks to the CS. The LUS holds a pool of previously partitioned jobs that need to be processed, and allocates resources. The CS computes jobs and transmits its results back to the LUS.

The goal is to make use of the CS during a period of user-computer quiescence. Additionally, we seek to minimize the intrusion into the desktop. A portable screensaver enables the heterogeneity of our grid computing framework. Thus a sub-problem includes the creation of a portable screensaver that can be downloaded, on demand. The screensaver invokes the CS and the CS announces itself to the LUS.

1.1. Approach

Our approach for the development of the screensaver addresses five aspects of resource management in the grid:

- User-computer quiescence detection, initiation, and termination,
- intrusion minimization,
- screensaver portability,
- screensaver deployment and
- screensaver integration with the IRJS (Initium Remote Job Submission) middleware.

User-computer quiescence-detection determines if the machine is in use, and this is a platform-specific activity [1]. Detection of quiescence enables the use of otherwise wasted CPU cycles. This event is used by the screensaver to invoke the CS in order to make itself known to the LUS, thus joining the grid.

The property of intrusion-minimization requires the restoration of computer resources. When the period of user-computer quiescence ceases, the screensaver should terminate any currently running compute jobs, releasing the computer back for general use. User-computer quiescence detection and intrusion minimization constitute a step toward utilizing otherwise idle compute resources.

For screensaver portability, we theorize that the creation of a Java-based screensaver, that is both cross-platform and easily installed, will help in the promotion of Java as a grid-based computing platform. We look to address screensaver solutions for the platforms of Windows, Linux/Unix and Macintosh.

Screensaver deployment should be an automatic process. Minimal user interaction is one of our goals. Due to differences on how screensavers are supported and setup in different platforms, custom solutions should be built without sacrificing the goal of minimal user interaction and portability.

Lastly, screensaver integration with the IRJS system intends to utilize the computer as a resource to the Grid system. The screensaver's role is mainly to serve as a launching and landing facility for the CS application. When the screensaver launches it should trigger the CS to initiate

and execute until the end of the user-computer quiescence. At that point the screensaver should terminate the CS as well as itself.

1.2. Motivation

Grid computation is a topic of current research that brings to the fore the issues of program partitioning, distributed computation, and supercomputing. It is of interest to a large number of researchers because it enables the use of resources that would otherwise remain idle. Further, it establishes virtual supercomputer facilities that are otherwise beyond the reach of most researchers.

We are motivated to study screensavers because they represent a minimally invasive technology for volunteering CPU services. Typically, computers are used between 40 and 60 hours out of a 168-hour week. This represents a 35% utilization. Screensaver based *cycle scavenging* improves this number dramatically.

We are motivated to address the sub-problem of devising a portable screensaver for volunteer computing, in part, because of the success of the SETI project. A major deficiency of SETI is that it is not portable and generally only available to accelerate a single application.

We are motivated to provide a Java-based environment in order to capitalize on Java's inherent heterogeneity. This makes a larger universe of grid-compute servers available without requiring changes to the computational program.

"Built on the Internet and the World Wide Web, the Grid is a new class of infrastructure. By providing scalable, secure, high-performance mechanisms for discovering and negotiating access to remote resources, the Grid promises to make it possible for scientific collaborations to share resources on an unprecedented scale, and for geographically distributed groups to work together in ways that were previously impossible." [13]

2. LITERATURE SURVEY

"Driven by the success of the SETI project and others like it, researchers have been working to exploit the vast pool of the computing resources connected to the Internet, but in a way that is secure, manageable, and extendable" [4].

JGRID is a Jini-based project that aims to create an infrastructure that relies on the features of JINI, such as, discovery, leasing, distributed events and transactions, security, etc…[6] JGRID is constructed by services that are joined together to compose a complex architecture. Each service is composed of a common set of modules and a specialized set of modules. The common set of modules provides services for discovery, lookup, registration, administration, etc… Compute services of JGRID bring remote Java Virtual Machines (JVM) into the Jini/GRID environment to execute JAVA objects in that JVM. Compute services support communication protocols and process management, such as, process scheduling and monitoring. Single and parallel tasks are sent to the remote compute services using Java RMI.

Our approach is similar to the JGRID compute service in that we seek to build a framework that takes care of the system environment locally in the CS. The implementation of the task, to be calculated by the CS, does not have to know about the details of the environment. Our approach is different from JGRID in that we seek to utilize Java Web Start as our tool for task-deployment, whereas JGRID relies on RMI and MPI mechanisms. In JGRID, it is unspecified

whether the compute service provides its services in a volunteer manner while the system is unutilized.

The Gridbus [15] project supports cluster and grid computing by developing open source middle-ware tools that aim to provide solutions to eScience and eBusiness applications. Gridbus technologies use a layered architecture. The core tools are located in a middle layer that interfaces between the service providers (bottom layer) and the applications that use the grid (top layer). Given a job to be executed, Gridbus utilizes a service broker that makes scheduling decisions based on the resources' characteristics such as availability, capability, and cost. Grid Market Directory (GMD) is queried to identify resources [15]. The GMD acts as a directory of services. GMD is developed over SOAP and XML. Gridbus middleware for service providers relies on the Alchemi and Globus frameworks. The Alchemi framework is built using the .NET technology and it aims to be used in the Windows platform. That is, the implementation of the service for resource providers is not cross-platform; therefore the same service must be developed for other platforms. This is a disadvantage that our approach resolves by creating a heterogeneous middleware and screensaver. Our approach seeks to utilize technologies based on web services, such as, Java Web Start, but at the same time avoids the utilization of XML due to the high cost of parsing.

The Globus [14] project provides a toolkit that defines the services necessary to construct a computational grid. These services include tools to manage resources' environment such as resource discovery, communication, and security. Globus uses an "hourglass" [14] -layer architecture. The neck are interfaces that are implemented by local services. High level services are defined by these interfaces. Resource management in Globus is provided by the Globus Resource Allocation Manager (GRAM). Globus may contain many GRAMs that are each operating in a set of local resources. Global resource management plans can be drawn with GRAM mechanisms. Resource Specification Language (RSL), is the medium for resource requests; constructed by definitions for global services. Globus differs from our approach in that it is a toolkit aimed to provide services necessary to build a custom computational grid. Globus is similar to our approach in that we seek to provide a general local management framework for the discovery and utilization of remote compute servers.

The GridLab [16] project provides a framework for users and developers that allow them to construct applications that can benefit from a grid system. The Grid Application Toolkit (GAT) offers an API that applications can use to utilize grid services. GridLab obtains generalization of grid services by following a layer of architecture. Both, the GAT and the GridLab services form the middle layer between the applications and Grid middleware. The goals of the GAT design are abstraction, flexibility and fail-safety [16]. The adapter pattern supports the abstraction of interfaces, and therefore is critical for the design of the GAT. Capability providers are interfaced to capabilities through adapters. Flexibility is achieved by allowing a dynamic interchange of capability providers at runtime; a registry is use for this purpose. Some degree of fail-safety is achieved by allowing other suitable adapters to be called in case of failure. An application uses the GAT-API to call the GAT. The GAT queries the registry for adapters. The adapter interacts with capability providers to complete the request. The GridLab project is similar to the Globus project in that both aim to provide services and policies that form a framework for the development of grid aware applications.

The MDS-2 [17] implementation for grid information services, part of the Globus toolkit, has the goal of providing mechanisms that allow the discovery, depiction, and monitoring of services. The implementation aims to overcome the challenges of these services due to volume,

variety, dynamics, and location of entities. Grid services differ in terms of the resources, demands, and information usage. The MDS-2 implementation is based on the idea that these services resemble in behavior: "one or more consumers (users or programs) wish to obtain information from one or more producers" [17]. The architecture is composed of two elements: a set of information providers, and higher-level services. The first element offers entities information obtained locally or through entity networking. Higher-level services manage that information, such as directory services. There are two protocols used to interact between these two elements, a registration protocol and inquiry protocol. They are used for the identification of entities and to query information about the entities.

The problems of first discovery and then organizing resources that are needed to meet the application requirements are not simple. Condor [18], uses matchmaking algorithms to support resource selection with the use of *ClassAd* language. Through this language users describe requirements, and resource owners describe resources. For example, users can specify resource properties, like total memory and bandwidth. The matchmaking algorithms allow one-to-many matching of requests to resources. The resource selector locates, evaluates, and returns the appropriate set of resources. The architecture for the Resource Selector Service (RSS) [18] entails three parts. The Resource Monitor is responsible for gathering resource data from a grid information system. The Set Matcher makes decisions on mapping resources to request. The Mapper is responsible for the arrangements of resources and workload allocation. The Mapper is provided by the user, and it is loaded at runtime. The RSS results describe the most appropriate resources for the request along with the mapping scheme, and are provided in XML format. Our approach, in the IRJS system, focuses on the mechanisms that a resource uses to announce its existence and to provide benchmark data about itself to the grid. From this viewpoint, our approach can be a complementary element to the resource selection service.

Deployment is a sub-problem of grid computing. Java Web Start [2] offers a deployment facility using the HTTP protocol. When a JWS application is executed, Java Web Start verifies that the latest version of the applications is executed. This is the same route used in a web application, the pages that you view are guaranteed to be the latest version posted. If a new version is found, Java Web Start downloads the files, installs them, and launches the application.

3. THE INITIUM RJS SCREENSAVER FOR WINDOWS

We are interested in screen-saver technologies, in Java, in order to facilitate a minimally invasive computing service able to make use of otherwise unused computational resources. There is little written on the subject of screen-saver based grid computing, in Java.

Screen-saver based grid computing systems are not new [20] but their use for Java computing is. Also, Java-based screensavers have, in the past, been restricted to MS Windows and Xwindows (UNIX)-based systems. The idea for using screensavers to accelerate Java grid computing has been mentioned in literature, but implementations have not been forthcoming [25].

We theorize that the creation of a Java-based screensaver that is both cross-platform and automatically installed will help in the promotion of Java as a grid-based computing platform. This section shows how to create a screensaver using an existing framework called *SaverBeans*. The *SaverBeans* development kit is an open-source, freely-available framework consisting of both C/C++ code and Java code.

3.1.A Java-Based Screensaver

The *SaverBeans Screensaver SDK* project, under the *Java.net* group, provides a set of native subroutines that invokes Java methods in the screensaver. The *SaverBeans* SDK has its roots in the JDIC project (**JD**esktop **I**ntegration **C**omponents). The **JDIC** project aims to make Java™ technology-based applications ("Java applications") first-class citizens of current desktop platforms without sacrificing platform independence. Its mission is to enable seamless desktop/Java integration [22]. The kit is available from [23] as an open-source distribution.

3.2. Building the Saverbeans SDK

Once the development kit has been downloaded, create a copy of the *saverbeans_startup* directory and rename it to *saverbeans_1*. Figure 3.2-1 shows the contents of the *SaverBeans* startup directory.

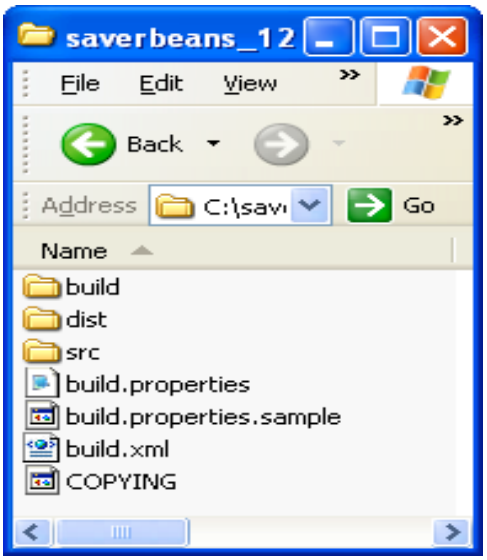

Figure 3.2-1. The Contents of the *SaverBeans* Startup Directory

The *build directory* contains the libraries and platform specific files needed in the building process. The *src directory* contains the documentation, packages, and Java code used by the screensaver.

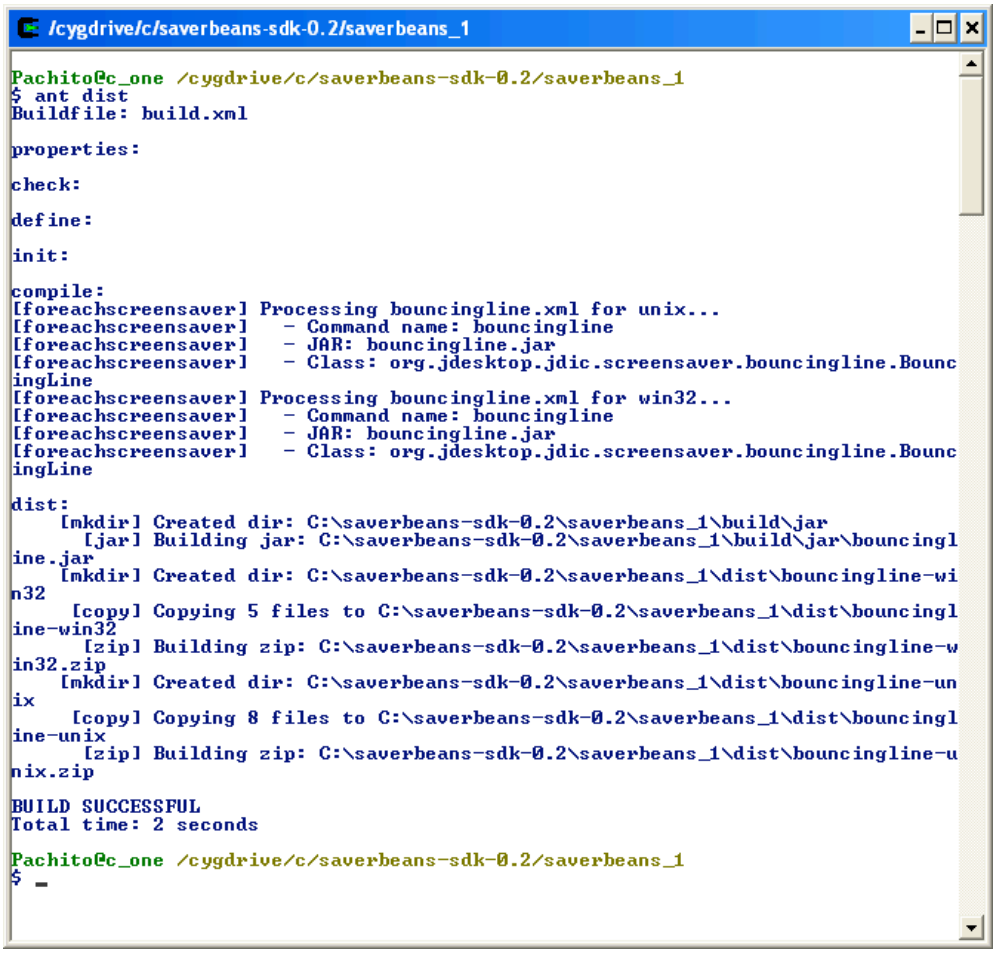

Figure 3.2-2. The *Dist* Directory is created during the *Ant* build.

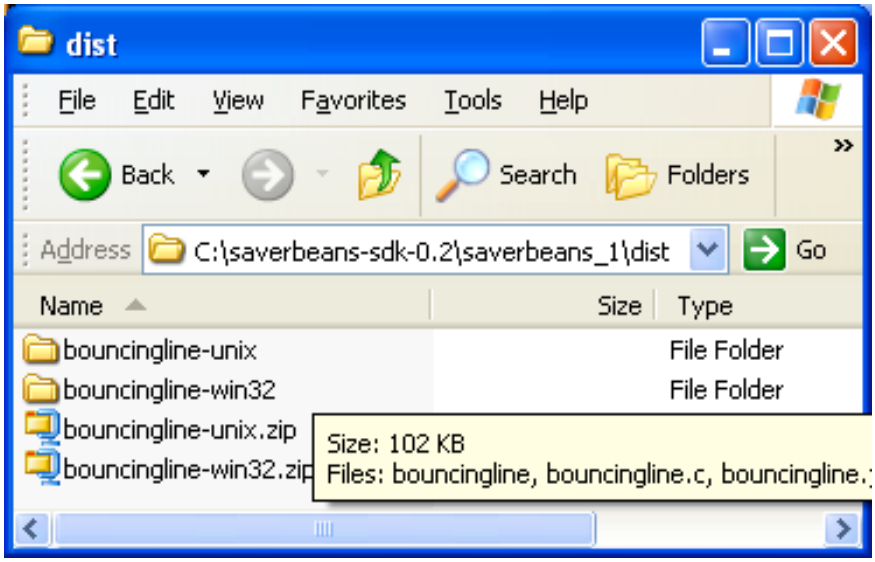

Figure 3.2-3. The Contents of the *SaverBeans* Dist Directory

The *dist directory* is created automatically during the compilation and construction process, as shown in Figure 3.2-2. Platform specific files of the screensaver are placed in this directory during the construction process.

Copy the *building.properties.sample* file to a new file called *build.properties*. This file contains the SDK home property, and this must be set correctly. For example:

```
sdk.home=C:\\j2sdk1.4.2_04
```

The *build.xml* file contains the *ant* build code. In order to perform a correct ant build, you must set the *saverbeans.path* in the *build.xml* file. To enable ant compilation, use:

```
saverbeans.path=C:/saverbeans-sdk-0.2
```

3.3. Compiling, Debugging and Deploying

Under windows, we installed the Cygwin system [21]. Using the command prompt, change directory to *saverbeans_1*. Type *ant clean* in order to remove anything left over from the last build. Type *ant debug* in order to compile and run the project. A frame will open, displaying the demo screensaver (a bouncing line).

In order to create a distribution, type *ant dist.* This step creates the *dist* directory containing windows specific files, ready for installation.

Install the screensaver by changing to the *bouncingline-win32* folder. This directory contains three files of interest: *bouncingline.jar, bouncingline.src, SaverBeans-api.jar*. Copy these files into the Windows system directory. The exact location is a function of the windows version:

For Windows XP, the location is: windows/system32
For Windows 98, the location is: windows/system
For Windows NT, the location is: winnt/system

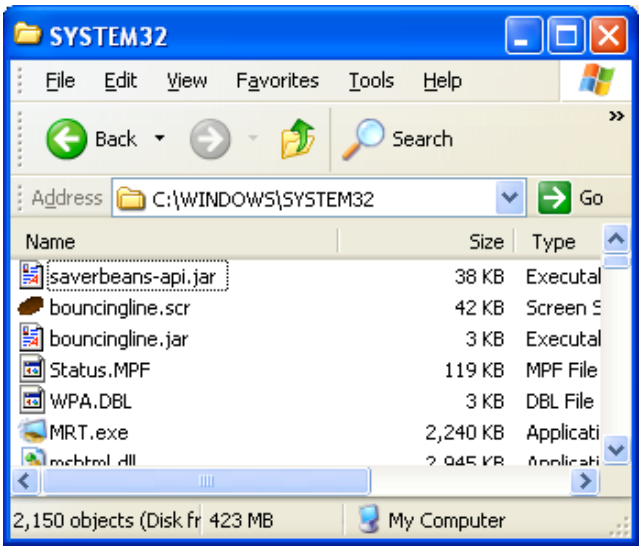

Figure 3.3-1. A Sample Windows XP Deployment

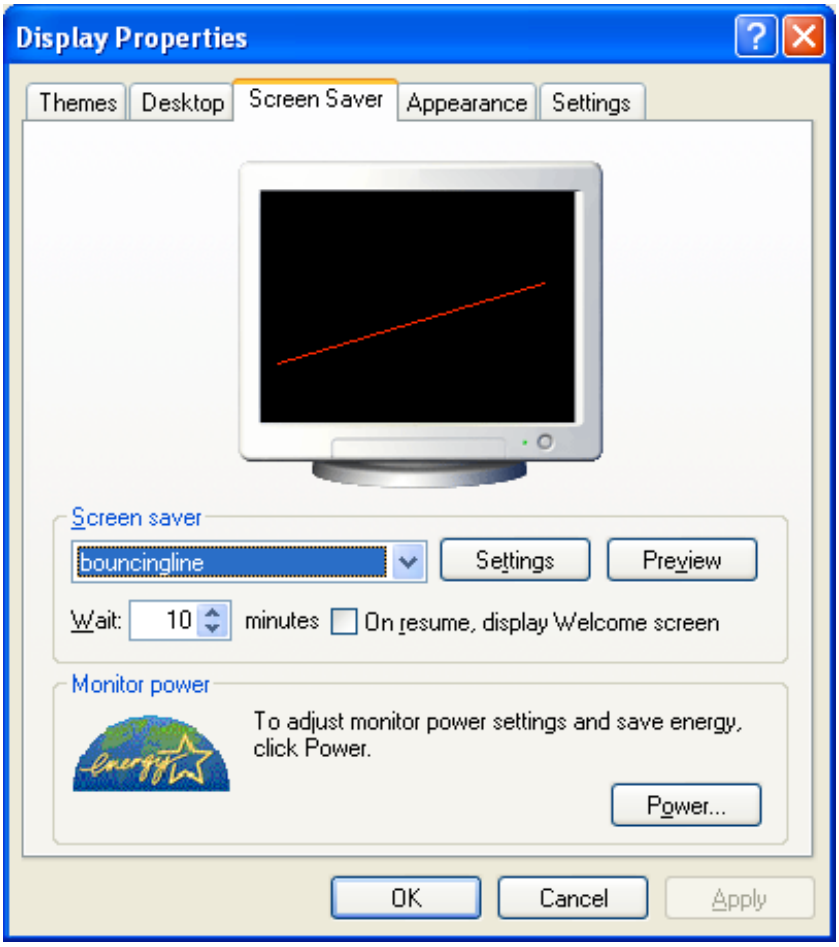

Figure 3.3-2. Setting the Screensaver

The last step is to set the screensaver to the *bouncingline* demo. Open the control panel and select the *bouncingline* screensaver in the screensaver tab. We set the time for "Wait" and apply the changes. The *bouncingline* screensaver is triggered automatically after the entered time has passed (given an idle machine, as shown in Figure 3.3-2.
The installation of a screensaver, in windows, requires that the user have write permission to the windows system directories. Typically, this is a non-issue, for a single users' machine. However, in an industrial setting, this can be a showstopper.

3.4. Implementation Details

3.4.1. Screensaver Class

As part of the startup package, the code of the *bouncingline* screensaver is included. The code is found in the *bouncingLine* class located in *src* directory. The complete path is *<startup project location>\src\java\org\jdesktop\jdic\screensaver\bouncingline*.

```
package org.jdesktop.jdic.screensaver.bouncingline;

        public class BouncingLine  extends SimpleScreensaver {
        public void init(){…}
```

```
        public void paint( Graphics g ) {…}
        public void destroy() {…}
        …
}
```

There are a few points to notice about this class. The *BouncingLine* extends *SimpleScreensaver*, an abstract class that is part of the *SaverBeans* API. Developers should extend either *SimpleScreensaver* or JOGLScreensaver (an OpenGL screensaver that typically makes use of 3D graphics).
SimpleScreensaver declares the abstract paint method. The frame is rendered by a regular callback to the *paint* method. In the *BouncingLine* class, the *paint* method erases the previous painted line and draws the new line. For example:

```
public void paint( Graphics g ) {
        Component c = getContext().getComponent();
        int width = c.getWidth();
        int height = c.getHeight();

        // Erase old line:
        g.setColor( c.getBackground() );
        g.drawLine( p1.x, p1.y, p2.x, p2.y );

        // Move points and bounce off walls:
        bounce( p1, dir1, width, height );
        bounce( p2, dir2, width, height );

        // Draw new line:
        g.setColor( lineColor );
        g.drawLine( p1.x, p1.y, p2.x, p2.y );
    }
```

The *SimpleScreensaver* class extends the abstract *ScreensaverBase* class [24]. *SimpleScreensaver* implements the *renderFrame* method, which is used as a call-back method from the *SaverBeans* framework.

Two other callback methods include *init and destroy*. These are called when the screensaver starts and stops. The *init* method is called once, and only once, after the screensaver starts. It will not be called, for example, if screen resolutions change. Our implementation of the *init* method follows:

```
public void init(){
        ScreensaverSettings settings = getContext().getSettings();
        Component c = getContext().getComponent();
        int width = c.getWidth();
        int height = c.getHeight();
        randomizePoint( p1, width, height );
        randomizePoint( p2, width, height );
        dir1 = new Point( randomVector(), randomVector() );
        dir2 = new Point( randomVector(), randomVector() );
        String colorOption = settings.getProperty( "color" );
```

The second useful method defined in *ScreensaverBase* is the *destroy* method, which we will not need now.

3.4.2. Screensaver Settings

The *SaverBeans* framework provides an XML file that is used to store screensaver properties. In the case of the *bouncingLine* screensaver, the file contains the following XML, located in *src/bouncingline.xml*:

```
<screensaver name="bouncingline" _label="Bouncing Line">
  <command arg="-root"/>
  <command arg="-jar bouncingline.jar"/>
  <command arg="-class
       org.jdesktop.jdic.screensaver.bouncingline.BouncingLine"/>

  <file id="jdkhome" _label="Java Home (blank=auto)" arg="-jdkhome %"
       />

  <select id="color">
    <option id="blue" _label="Blue Line" /> <!-- default -->
    <option id="green" _label="Green Line" arg-set="-color #00ff00" />
    <option id="red" _label="Red Line" arg-set="-color #ff0000" />
  </select>

  <_description> …..</_description>
</screensaver>
```

While the use of an XML file to establish these properties seem cumbersome, it is required because of the framework. These XML files do not need to be altered, once they are established for a screensaver with a stable class name.

4. THE INITIUM RJS SCREENSAVER FOR UNIX

This section describes the application of our technology to the UNIX platform. Our previous section covered the Windows platform, and our next section will cover the Macintosh platform. Our goal is to make use of these screensavers in the *Initium RJS system*, our grid-computing framework. While we acknowledge that the form of the screensaver installation, as presented in this section and last section, is tedious and error-prone, we will describe the automatic installation of a compute-serving screensaver in a later section of this thesis. It is our hope that our system will help in the promotion of Java as a grid-based computing platform.

This section shows how to create a screensaver using an existing framework called *SaverBeans*. The *SaverBeans* development kit is an open-source, freely-available framework consisting of both C/C++ and Java code. The kit is available for both the Windows and Linux systems. However, it is not available for the Macintosh. The alternative to creating a Macintosh-based screensaver is to run X-windows under the Macintosh (an atypical use of the Macintosh).

This section introduces the *SaverBeans SDK* for UNIX with an *XWindows* GUI. The idea of using screensavers for Java-based grid computing is not new [25]. However, the work was not continued and present implementations do not make use of screensavers for grid computing [26].

4.1. Building the Saverbeans SDK

In order to install the screensaver in a UNIX/*XWindows* workstation, you must have the standard *xscreensaver* installed. While most workstation installations come with the screensaver

already installed, most installations do not include the source code that is needed for the native method builds. The *xscreensaver* installation includes a daemon that detects quiescence [27].

To determine the version of the *xscreensaver* type:

```
xscreensaver &
xscreensaver-command -version
```

The computer responds with:

XScreenSaver 4.21

The '&' in the first line place the *xscreensaver* in the background.

The method for installation of the *xscreensaver* may vary from platform to platform. Download http://www.jwz.org/xscreensaver/xscreensaver-4.23.tar.gz and uncompress and untar the *xscreensaver* file and cd into that directory, then run configure and make. Type:

```
tar -vxzf xscreensaver-4.23.tar.gz
cd xscreensaver-4.23/
./configure
make
make install
```

Even if the *xscreensaver* binaries are already installed, you may still need the source code to compile the *SaverBeans* SDK. This has been bundled in a special distribution available at http://www.docjava.com. As an alternative, you can download the *SaverBeans* SDK from https://jdic.dev.java.net/files/documents/880/12349/saverbeans-sdk-0.2-beta.zip and unzip. Type:

```
cd saverbeans-sdk-0.2-beta
unzip saverbeans-startup.zip
cp build.properties.sample build.properties
```

Alter the *SaverBeans* path in the *build.properties* directory to reflect the installation location of the SDK. For example:

```
saverbeans.path=/opt/saverbeans
becomes:
saverbeans.path= saverbeans.path=/home/lyon/current/ssbeta/saverbeans-
        sdk-0.2-beta/
```

Make sure that the Java virtual machine is in the *PATH* and that the *JAVA_HOME* is set correctly:

```
show.docjava.com{lyon}113: which java
/usr/java/jdk1.5.0_04/bin/java
show.docjava.com{lyon}114: echo $JAVA_HOME
/usr/java/jdk1.5.0_04/
```

Finally build the project using:

ant dist

This will generate a directory called *dist*. The *dist* directory contains the distributable files. Edit the *Makefile* in the *dist/bouncingline-unix* directory altering *jdkhome* and *xscreensaverhome* to valid directories. For example:

```
jdkhome=/usr/java/jdk1.5.0_04
xscreensaverhome=/home/lyon/current/ssbeta/saverbeans-sdk-0.2-
        beta/xscreensaver-4.23
```

To compile the native methods for the current platform, type:

```
Make
```

Notice the files generated: *bouncingline-bin* and *bouncingline.o*.
The following section describes how to deploy the screensaver to an *XWindows* system under UNIX.

4.2. Deploying

Now that the binaries have been generated, these files can be used to deploy the screensaver into an *XWindow* platform. With the native method framework in place, a wide variety of different screensavers can be authored, in Java, without having to rebuild the native methods.

The *.xscreensaver* file needs to be modified to include the *bouncingline* screensaver. The *xscreensaver-demo* program generates this file and places it into the users' home. Type:

```
xscreensaver-demo
```

Look into the users' home directory to verify the existence of the *.xscreensaver* file. To inform the *xscreensaver* program that you have a new screensaver, you should edit the *.xscreensaver* file in your home directory. To add the *bouncingline* screensaver to the *.xscreensaver* file, use:

```
programs:
        "Bouncingline (java)"/home/lyon/ss/bouncingline -root -jdkhome
        /usr/java/jdk1.5.0_04
```

Now execute the screensaver by typing:

```
xscreensaver-demo
```

Figure 4.2-1 shows the screensaver dialog box, with the *bouncingline* screensaver selected.

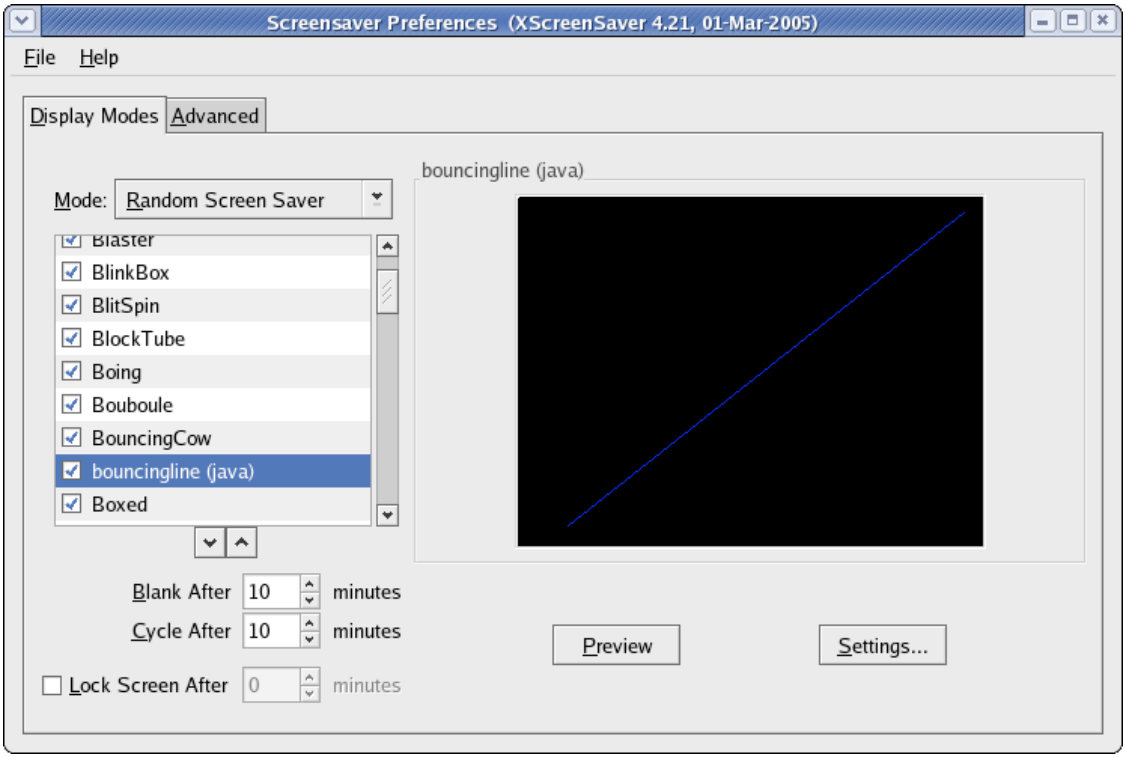

Figure 4.2-1. The Screensaver Dialog

For ease of use, the files required to run the screensaver are placed in a single directory called *ss*. No root permissions are required to install custom screensavers that reside in the home directory.

4.3.Makefile Synthesis

Ant is a multi-platform, java-based, make-like utility. The *ScreenSaver* SDK uses *ant* as well as custom *ant* tasks, stored in a jar file called *saverbeans-ant.jar*. In the course of running the *ant dist* a make file is synthesized. This is created for UNIX and Windows. The contents of the *dist* directory include:

```
   bouncingline-unix/     bouncingline-win32/
   bouncingline-unix.zip  bouncingline-win32.zip
```

The *bouncingline-unix* directory contains the files:

```
   bouncingline       bouncingline.jar  COPYING     saverbeans-api.jar
   bouncingline-bin*  bouncingline.o    Makefile
   bouncingline.c     bouncingline.xml  README.txt
```

The *Makefile* is generated from a template. We have altered this template in order to generate a file that is somewhat more automatic in its installation. For example:

```
   # Set this to your Java home directory
   jdkhome=${JAVA_HOME}

   # Set this to where the xscreensaver source bundle is installed
   xscreensaverhome=../../xscreensaver-4.23
```

Further, we have installed a version of the *xscreensaver* that makes compilation somewhat more automatic. In order to accomplish these changes, we altered the *Makefile.template* file in:
`saverbeans-ant/org/jdesktop/jdic/screensaver/autogen/resources/unix`

The template contains a file with variables that to help drive the *Makefile* synthesizer. The creation of *Makefiles* in this way is unique, as far as we know.

4.4.Writing a Screensaver

The following code shows how to write a screensaver by subclassing the *SimpleScreenSaver* class:

```
public class Test1
        extends SimpleScreensaver {

    public void paint(Graphics g) {
        Component c = getContext().getComponent();
        int x = 0;
        int y = c.getHeight() / 2;
        g.setColor(Color.WHITE);
        g.setFont(new Font("Dialog",Font.BOLD,30));
        g.drawString("Initium RJS see: http://www.docjava.com", x, y);
    }


    public static void main(String[] args) {
        new ScreensaverFrame(new Test1()).setVisible(true);
    }
}
```

The *main* method makes an instance of a *ScreensaverFrame*, used for testing. The *ScreensaverFrame* is a subclass of the *JFrame* and sets the *context* of the *SimpleScreensaver*. This context is an instance of a *Component* class. In the case of a *JFrame* it is also an instance of a *Container*. Knowing this, we are at liberty to establish a layout with standard swing components as a part of our screensaver. We need to obtain the *Container* of our *Component* via focus traversal in the *init* method. For example:

```
public class Test2
        extends SimpleScreensaver {
    JPanel buttonControlPanel = getButtonControlPanel();

    private JPanel getButtonControlPanel() {
        JPanel jp = new JPanel();
        jp.setLayout(new FlowLayout());
        jp.add(new RunButton("ok") {
            public void run() {
                System.out.println(getText());
            }
        });
        jp.add(new RunButton("cancel") {
            public void run() {
                System.out.println(getText());
            }
```

```
            });
            return jp;
        }

        public void init() {
            Container c =
           super.getContext().getComponent().getFocusCycleRootAncesto
r();
            c.add(buttonControlPanel);
        }

        public void paint(Graphics g) {
            Component c = getContext().getComponent();
            c.paint(g);
        }


        public static void main(String[] args) {
            new ScreensaverFrame(new Test2()).setVisible(true);
        }
}
```

The *getFocusCycleRootAncestor* enables the addition of an arbitrary swing panel to the display. This is useful for creating GUIs needed for controlling the compute server.

4.5.Summary

This section addressed the issue of implementing a Java-based screensaver under the X-Window system, as well as providing a solution to the automation of installation and deployment for these systems. Focus traversal techniques helped with the swing programming. We find that focus traversal works for screensavers written for either MS Windows or XWindows. Unlike the MS Windows screen saver, no system administration privileges were required for installation. Further, unlike MS Windows, Linux does not need to run a windows server and therefore may not have a screen saver. In such a case, no screen-saver based system can take advantage of the machine.

The following section will describe how to implement a Java-based screen saver on the Macintosh operating system. Screensaver integration with the grid-based middleware and automatic screen saver deployment are the topics of further sections in this thesis.

5. THE INITIUM RJS SCREENSAVER FOR MACINTOSH

This section describes how to create a Java-based screensaver for a Macintosh. Our previous sections covered the Windows and Unix platforms. Our goal is to make use of these screensavers in the *Initium RJS system*, our grid-computing framework.

Our past work described an existing framework, called the *SaverBeans* development kit, an open-source, freely-available framework consisting of both C and Java code. The kit is available for both the MS Windows and Linux systems. However, it is not available for the Macintosh. The alternative to creating a Macintosh-based screensaver is to run X-windows under the Macintosh. Our impression is that this is an idiosyncratic use of the Macintosh, and users prefer a solution that makes use of the native window manager (quartz) of the Macintosh.

5.1.A Java Screensaver Framework for Macintosh

The Macintosh has a development IDE available for simultaneous creation of both Objective C and Java programs. The IDE is freely available, as a part of the *XCode* distribution and is called *ProjectBuilder*.

The basic idea behind our screensaver is that it will run a Java Web Start Application. This enables deployment of updates to our screensaver without having to reinstall it.

We start with an Objective C program (a file with a *.m* suffix), inspired by [28]:

```
//  ScreenView.m
//  ScreenSaver
//
#import "ScreenView.h"
@implementation ScreenView

int i = 0;
- (void)animateOneFrame {
        //- (void)startAnimation{
        NSBezierPath *path;
        NSRect rect;
        NSSize size;
        NSColor *color;size = [self bounds].size;

        if(i==0){
          NSLog(@" First time   %d      SS start now", i);
          //Call to java class to start dhry.main.app
          [NSClassFromString(@"RunCS")
          newWithSignature:@"(Ljava/lang/String;)",@"start"];
}

rect.size=NSMakeSize(SSRandomFloatBetween(size.width/100,size.width/10),
SSRandomFloatBetween(size.height/100,size.height/10));
rect.origin = SSRandomPointForSizeWithinRect(rect.size,[self bounds]);
if (SSRandomIntBetween( 0, 1 ) == 0) {
    path = [NSBezierPath bezierPathWithRect:rect];
} else {
    path = [NSBezierPath bezierPathWithOvalInRect:rect];
}
color = [NSColor colorWithCalibratedRed:(SSRandomFloatBetween( 0.0,
        255.0)/ 255.0)
             green:(SSRandomFloatBetween( 0.0, 255.0 ) / 255.0)
             blue:(SSRandomFloatBetween( 0.0, 255.0 ) / 255.0)
             alpha:(SSRandomFloatBetween( 0.0, 255.0 ) /255.0)];
[color set];
i++;
[path fill];
}

- (void)stopAnimation{
        //Call to java class to stop dhry.main.app
    [NSClassFromString(@"RunCS")
        newWithSignature:@"(Ljava/lang/String;)",@"stop"];
        NSLog(@"SS  stop now %d  ", i);
}
```

```
@end
```

If the screensaver is started for the first time, the *counter* (i=0) is zero. Events are logged to the console using *NSLog*. The *stopAnimation* method is invoked when the screensaver terminates. The *RunCS* code follows:

```
//  RunCS.java
//  ScreenSaver
//
import com.apple.cocoa.foundation.*;
import com.apple.cocoa.application.*;
import java.io.IOException;
import java.util.Properties;
import java.io.File;
import java.io.FileOutputStream;

public class RunCS {
    private static String tmpDir = System.getProperty("java.io.tmpdir");
    private static String fileSep = System.getProperty("file.separator");
    public final static File killFile = new File(tmpDir +
                                fileSep +
                                "killcs");

    private static void startCs()  {
        if (killFile.exists())
            killFile.delete();

        final  String wsMacLocation = fileSep +
            "Applications" +
            fileSep +
            "Utilities" +
            fileSep +
            "Java" +
            fileSep +
            "Java Web Start.app" +
            fileSep +
            "Contents" +
            fileSep +
            "MacOS" +
            fileSep +
            "Java Web Start";
        String url = "http://show.docjava.com:8086/" +
            "book/cgij/code/jnlp/net.rmi.rjs.pk.main.CsMain.jnlp";

        System.out.println("webstart is here:"+ wsMacLocation);
        String args[] = {
            wsMacLocation,
            url
        };
        Runtime rt = Runtime.getRuntime();
        try {
            rt.exec(args, null, null);
        } catch (IOException e) {

        }
        System.out.println("finished!");
```

```
    }
    public void stopCs(){
        killFile.mkdir();
        System.out.println("Stoping the CS");
    }

    public RunCS (String cmd) {
        if (cmd.equals("start")){
            System.out.println("start");
            startCs();
        } else stopCs();
    }
}
```

The screensaver runs a computation server using the *RunCS* class. The*RunCS* constructor is created with a *String* argument. If the *String* argument is equal to "start", then the *startCs* method is invoked. The *startCs* method checks to see if the semaphore file, *killcs* exists, and deletes it if it does. A thread checks the file, and if it exists, the computation server is terminated. The semaphore file is stored in a temporary directory. If the argument to the constructor is "stop" then the *stopCs* method is invoked. The *stopCs* method creates the *killcs* file to trigger termination. The screensaver (written in Objective C) invokes the Java program using an objective C to Java bridge [29] [31].

5.2.Installing the Screensaver

We have created a web start method for automatically deploying and installing the screensaver to a Mac. The URL is available at http://show.docjava.com:8086/book/cgij/code/jnlp/net.rmi.rjs.MacScreenSaverUtils.jnlp and provides for an installation using a technique we call *beaming over* the files. The basic idea is that the screensaver files are transferred from the web server to the local disk. There they are uncompressed and placed into the proper location for user screensavers (~/Library/Screensavers/). The code for this type of beam over operation follows:

```
public class MacScreenSaverUtils {

    private static String screenSaverDirectoryName =
            SystemUtils.getUserHome() +
            "/Library/Screensavers/" ;
    private static File outputJarFile = new File(
            screenSaverDirectoryName+
            "screenSaver.jar");
    //download the screensaver in:
    //
        http://show.docjava.com:8086/book/cgij/code/jnlp/libs/mac/screenS
        aver.jar

    public static void downloadScreenSaverJar()
            throws IOException {

         URL screenSaverUrl = getResourceUrl();
         UrlUtils.getUrl(screenSaverUrl,outputJarFile);
```

```
}
public static void testStartRunCheckThread(){
    new CheckForDeathJobProperties();
    guiKillCS();
}

private static void guiKillCS() {
    while(In.getBoolean(("keep cs running?"))){
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            In.message(e);

        }
    }
    putPrefPropToDie();
}

private static void putPrefPropToDie() {
    Preferences p = Preferences.systemRoot();
    p.put(csKillKey, "true");
}

private static final String csKillKey = "timeToKillCS";
public static void startRunCheckThread(){
    final Preferences p = Preferences.systemRoot();
    p.put(csKillKey, "false");
    new RunJob(1){
        public void run(){
            final Preferences p = Preferences.systemRoot();
            String value = p.get(csKillKey, "false");
            if (value == null) return;
            if (value.equals("false")) return;
            killCS();//kill the CS
        }
    };
}
private static class RunCheckThread extends Thread {

    public void run() {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
        }
        final Preferences p = Preferences.systemRoot();
        String value = p.get(csKillKey, "false");
        if (value == null) return;
        if (value.equals("false")) return;
        killCS();//kill the CS
    }
}
private static void killCS() {
    System.out.println("cs is dead");
    System.exit(0);
}

private static URL getResourceUrl() throws MalformedURLException {
```

```
        URL screenSaverUrl= new URL("http://show.docjava.com:8086" +
                "/book/cgij/code/jnlp/libs/mac/screenSaver.jar");
        return screenSaverUrl;
    }

    public static void uncompressScreenSaverJar(){
        Unzipper.uncompressJarFile(outputJarFile);
        outputJarFile.deleteOnExit();
    }

    public static void main(String[] args) {
          //SystemUtils.printProps();
        installScreenSaver();
        In.message("The Computation Screensaver Now Exits...");
        System.exit(0);
        //testStartRunCheckThread();
    }
    private static boolean dateIsGood() {
        try {
            File dataDir = new File(screenSaverDirectoryName);
            long dataDirTime = dataDir.getCanonicalFile().lastModified();
            URL resourceUrl = getResourceUrl();
            final URLConnection urlConnection =
           resourceUrl.openConnection();
            long resourceUrlTime = urlConnection.getLastModified();
            return dataDirTime > resourceUrlTime;
        } catch (IOException e) {
            In.message(e);
        }
        return false;
    }

    public static void installScreenSaver() {
        if (dateIsGood()) return;
        if (!OsUtils.isMacOs()){
            In.message("This only works on macos! Program exits!");
            return;
        }
        if (!In.getBoolean("install screensaver?")) return;
        System.out.println("check for output in:"+outputJarFile);
        try {
            downloadScreenSaverJar();
            uncompressScreenSaverJar();
        } catch (IOException e) {
            In.message(e);

        }
        System.out.println("finished!");
        In.message("set screensaver to ScreenSaver and check hot
       corners!");
    }
}
```

The *dateIsGood* method checks the local screensaver installation to see if there were any updates. If the old screensaver is newer than the screensaver on the web server, no download

occurs. Once the user sets up the screensaver, it can be tested with a preview command. During preview, the new screensaver starts the *Initium RJS Compute Server*.

5.3.Deploying

The *screensaver.jar* mentioned in Section 2.2 has the following files in it:

```
./ScreenSaver.saver
./ScreenSaver.saver/Contents
./ScreenSaver.saver/Contents/MacOS
./ScreenSaver.saver/Contents/MacOS/ScreenSaver
./ScreenSaver.saver/Contents/pbdevelopment.plist
./ScreenSaver.saver/Contents/Info.plist
./ScreenSaver.saver/Contents/Resources
./ScreenSaver.saver/Contents/Resources/Java
./ScreenSaver.saver/Contents/Resources/Java/ScreenSaver.jar
./ScreenSaver.saver/Contents/Resources/RunSystemCmd.java
./ScreenSaver.saver/Contents/Resources/English.lproj
./ScreenSaver.saver/Contents/Resources/English.lproj/InfoPlist.strings
```

The file is uncompressed and moved into the users' screensaver folder automatically. The key to this effort is the ability to beam over the resources from a given URL and uncompress them. Beaming a resource from a web server into a local file is a service performed by a helper method in the *UrlUtils* class:

```
/**
     * Read a url and put it into a file. This is very good when dealing
     * with large files.
     *
     * @param url input file (like data.jar)
     * @param f   locally created output file.
     */
    public static void getUrl(URL url, File f)
            throws IOException {

        FileOutputStream fos = new FileOutputStream(f);
        BufferedInputStream bis = new
                BufferedInputStream(url.openStream());
        int numberOfBytesRead = 0;
        int buffSize = 65536;
       byte b[] = new byte[buffSize];
         pd.setVisible(true);
       while ((numberOfBytesRead = bis.read(b)) != -1) {
            fos.write(b, 0, numberOfBytesRead);

       }
       bis.close();
       fos.close();
       pd.setVisible(false);
    }
```

To unpack the Jar file, we have a class called the *Unzipper*:

```
public static void uncompressJarFile(File inputJarFile){
         Unzipper uz = new Unzipper(inputJarFile);
```

```
          String s[] = uz.getNames();
          File dir = inputJarFile.getParentFile();
        for (int i=0; i < s.length;i++){
          File outputFile = new File(dir,s[i]);
          byte b[] = uz.getBlob(s[i]);
          File parentFile = outputFile.getParentFile();
          if (parentFile != null && ! parentFile.exists())
                 parentFile.mkdirs();
          Futil.writeBytes(outputFile,b);
        }
}
```

5.4.Summary

This section illustrates the details of creating a Java-based screensaver for the Macintosh. The screensaver launches a Java Web Start application upon detection of a quiescent period. Web start applications upload to a web server asynchronously with respect to the screensaver. New web start applications will be automatically downloaded, and verified, by the web start launching framework.

The web start application launched by the screensaver framework is a CS. The CS volunteers the spare CPU cycles of the host to the grid.

We have also disclosed a beam-over technique that enables the transfer of a screensaver resource from a compressed file stored on the web server. The beam over includes an uncompressed phase, as well as, an installation phase that places the files into the users screensaver library. Activating the grid screensaver as the default requires manual user intervention. The question of how to automate this process remains open. Jar verification should help thwart man-in-the-middle attacks on the Jar file during transfer. The question of how to do the verification against a trusted certificate remains open.

The question of how to make a screensaver more like the SaverBeans SDK has also been left for future work.

6. THE INTIUM RJS SCREEENSAVER: AUTOMATIC DEPLOYMENT

The Initium RJS System makes use of screensavers to perform CPU scavenging for grid computing. This section shows how to automate the installation of the IRJS screensaver. The manual installation of the IRJS screensaver presented in sections 3 and 4 has been automated to ease installation. A Java Web Start application has been created to complete the installation of the screensaver for Windows and Unix/Linux platform. This application performs an operating system identification and proceeds to download, install, and configure the screensaver files. The installation of the screensaver is a function of the OS, due to differences in screensavers support.

6.1. Architecture

We use Java Web Start technology to install the screensaver. A screensaver volunteers the computer into the grid by running a web start application. The installation application beams the resource files into the computer from a web server, updating them, automatically, when needed. The files are decompressed and verified before they are installed. Lastly, the application configures the screensaver according to OS requirements.

Figure 6.1-1 shows an overview of the events and parties involved in first deploying and installing the screensaver and second in volunteering the user computer to a grid. This section addresses events 1 and 2 in Figure 6.1-1.

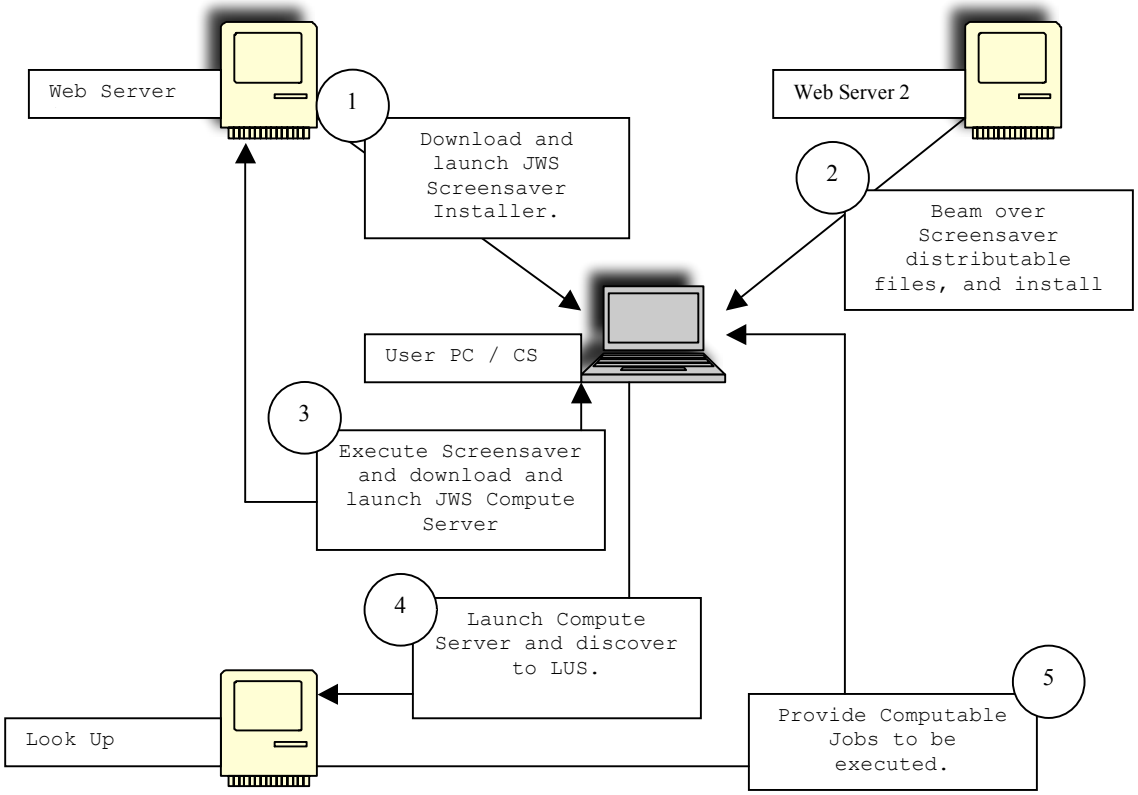

Figure 6.1-1 Partial diagram of the IRJS System Architecture

6.2. Operating System Identification

Identification of the user's operating system is an important step in the installation process, due to the differences between the operating systems in supporting screensavers. Different flavors of the Windows OS have different ways of supporting screensavers. In particular, the screensaver files belong to different directories depending on the version of Windows. The web start application uses some utility methods, as shown in Example 6.2-1, to help in the task of identifying the OS. These methods utilize the *System.getProperties()* method from the Java API which determines the system properties and returns a *Properties* object.

Example 6.2-1

```
public static String getOsName() {
           Properties prop = System.getProperties();
           return prop.getProperty("os.name");
}
public static boolean isWindows(String str) {
        if (isWindows()) {
            String os = getOsName().toLowerCase();
            if (os.indexOf(str) > -1) return true;
```

```
        }
        return false;
}
public static boolean isOsPrefix(final String prefix) {
        String os = getOsName();
        return os != null && os.toLowerCase().startsWith(prefix) ?
                true :
                false;
 }
public static boolean isWindowsXp() {
        return isWindows("xp");
 }
…
public static boolean isLinux() {
        return isOsPrefix("linux");
 }
```

6.3. Beamover and Decompression

A file-transfer process called beam over enables the transfer of native resources into critical file areas. The web start application beams over the IRJS screensaver files, based on the OS identified. Beam over and decompression take cares of the job of downloading needed resources, and placing them where appropriate. Resources needed, and their location, are a function of the OS, as shown in Example 6.3-1.

Example 6.3-1

```
   properties.put(SS_WIN_URL,
"http://www.myjavaserver.com/~fsophisco/thesis/libs/screensaver/win/rjssaverj
ar.jar");

   properties.put(SS_UNIX_URL,
"http://www.myjavaserver.com/~fsophisco/thesis/libs/screensaver/unix/rjssaver
jar.jar");
```

Example 6.3-2 shows the location where files will be installed.

Example 6.3-2

```
   windowsSystemDir = "";
   if (OsUtils.isWindows98()){
      windowsSystemDir = "C:"      + File.separator +
                     "windows" + File.separator +
                     "system"  +  File.separator;
      }
   else if (OsUtils.isWindowsNt()){
      windowsSystemDir = "C:"      + File.separator +
                   "winnt"         + File.separator +
                  "system"       + File.separator;    }
```

File beam over enables downloads on demand. This simplifies resource bundling and lowers average download time (since only the resources that are needed are downloaded). This is shown in Example 6.3-3.

Example 6.3-3

```
   public static void downloadScreenSaverJar(File outputJarFile, String
urlStr) throws IOException {
      URL screenSaverUrl = getResourceUrl(urlStr);
      UrlUtils.getUrl(screenSaverUrl, outputJarFile);
   }
   …
   //Beam over jar file for Windows
   outputJarFile = new File (windowsSystemDir + File.separator +
SSInstallerUtil.getJarName() + ".jar");
   String urlString = SSInstallerUtil.getSSWinUrl();
   try{
      SSInstallerUtil.downloadScreenSaverJar(outputJarFile, urlString);
   }catch(Exception e){
      return "Error Downloading Jar File: " + e.toString();
   }
   return "";
   …
   //Beam over jar file for Unix
   outputJarFile = new File (ssHome + File.separator +
SSInstallerUtil.getJarName() + ".jar");
   String urlString = SSInstallerUtil.getSSUnixUrl();
   try{
      SSInstallerUtil.downloadScreenSaverJar(outputJarFile, urlString);
   }catch(Exception e){
      return "Error Downloading Jar File: " + e.toString();
   }
   return "";
```

After downloading, the web start application decompresses the jar files, as shown in example 6.3-4.

Example 6.3-4

```
   public static void uncompressScreenSaverJar(File jarFile){
       Unzipper.uncompressJarFile(jarFile);
       jarFile.deleteOnExit();
   }
   …
   public String uncompressFiles(){
   try{
      SSInstallerUtil.uncompressScreenSaverJar(outputJarFile);
   }catch(Exception e){
            return "Error uncompressing Jar File: " + e.toString();
   } return "";

   }
```

6.4. Configuration

Up until this point deployment has been platform independent. However, differences in how screensavers are supported reducing the portability of the configuration task. For example, the screensaver files for the Windows OS must be placed in the directory *system* or *system32* under the *Windows* directory. These directories are part of the OS itself; therefore administrators may choose to protect them from being modified. Due to this type of constraint, we require that the installer have privileges to write to these directories.

For Windows screensaver configuration, the web start application places the screensaver files in version sensitive directories. For example:

For Windows 98: C:/windows/system/.
For Windows NT: C:/winnt/system/.
For other version: C:/windows/sytem32/.

The screensaver file-package contains a file with extension .scr E.g.<scrensaverName>.SCR. This file allows the screensaver to be visible on the *Display Properties* screen next time it is opened as shown in Figure 6.4-1 After the screensaver installation is complete there is a manual step that the user must accomplish. He/She must select the *rjsssaver* from the screensaver list, as shown in Figure 6.4-1, and set the time to wait before launching. This step completes the IRJS screensaver configuration for the Windows platform.

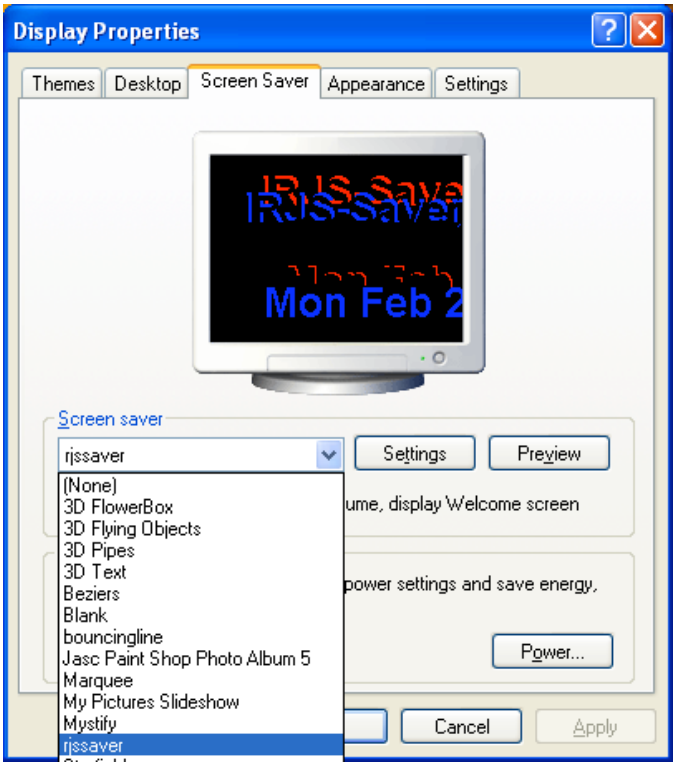

Figure 6.4-1 Display properties-user interface for the Windows platform.

In Linux/Unix OS, the IRJS screensaver configuration is a little more challenging. The Saverbeans SDK relies on a software called Xscreensaver; which is often part of the OS installation package. The **Xscreensaver** program waits until the keyboard and mouse have been idle, and then runs a graphics demo. It turns off as soon as there is any mouse or keyboard activity [27]. **Xscreensaver** consists of two parts: a driver or daemon that detects idleness and does locking; and the many graphics demos that are launched by Xscreensaver [27]. To learn more about this package visit http://www.jwz.org/xscreensaver/.

The first part of the configuration happens in the beam over and decompression task. There are no strict requirements that dictate the location of screensavers in the file system. Therefore, for convenience and to prevent any access issues the web start application installs the screensaver files in the user home in the directory <userHome>/ss/<screensaverName> as shown in Example 6.4-1.

Example 6.4-1

```
public String createSSHome(){
      String ssloc =  SSInstallerUtil.getUserHome() + File.separator
                              + SSInstallerUtil.getSSHome();
      ssHome = SSInstallerUtil.getUserHome() + File.separator
                  + SSInstallerUtil.getSSHome() + File.separator
                  + SSInstallerUtil.getSSName();
      try{
            SSInstallerUtil.createDir(ssloc);
            SSInstallerUtil.createDir(ssHome);
      }catch (Exception e){
            return "Error creating ss home: " + e.toString();
      }
      return "";
}
```

In Unix/Linux the Xscreensaver application uses a file named *.xscreensaver*, located at the user home. This file lists user properties, and active screensavers and their location. The *.xscreensaver* file is created automatically the first time that the client application (xscreensaver-demo) of the Xscreensaver is executed.

The web start application modifies the *.xscreensaver* file to make the IRJS screensaver available. In particular the IRJS screensaver must be included and must be selected as the only active screensaver. The web start application reads the contents of the *.xscreensaver* file, makes modifications where necessary, and rewrites the file, as shown in Example 6.4-2.

Example 6.4-2

```
//mode: one -> one screensaver working
sloc = sline.indexOf("mode:");
if (sloc > -1){
      lines.add("mode:\tone");
      sline = raf.readLine();
      continue;
}

//Once it finds the string "programs:" then insert the line with the SS info
into the List.
if (sloc > -1){
```

```
      lines.add("\t\" " + SSInstallerUtil.getSSName() + " (java)\"" + ssHome
       + File.separator + SSInstallerUtil.getSSName() + " -root -jdkhome " +
jhome +" \\n\\");
}
```

To complete the screensaver configuration in Unix/Linux platform, the web start application grants executable privileges to the screensaver executables. This task is necessary because when files are archived into a jar file and at later time extracted, they do not retain file permissions. To overcome this issue, the web start application executes a small shell script using the Runtime Java API, as shown in Example 6.4-3.

Example 6.4-3

```
Script:
LOC=$HOME/ss/rjssaver
chmod +x $LOC/rjssaver $LOC/rjssaver-bin

public String changeFilePrivs(){
      String[] pc = new String[2];
      int exitValue = 0;
      //Grant execute permissions
      try{  pc[0]= "sh";
            pc[1]= ssHome + File.separator + scriptName;
            p = Runtime.getRuntime().exec(pc);
            exitValue = p.waitFor();
      }catch(Exception e){
            return "Error granting privs Exit Value: " + exitValue +
e.toString();
      }
      if (exitValue != 0)
            return "Error granting privs Exit Value: " + exitValue;
      return "";
}
```

Lastly, after the screensaver installation is completed there is one manual step remaining. The user must execute the program xscreensaver-demo, so the *.xscreensaver* file is read and the IRJS screensaver is recognized as one of the screensavers. This step completes the IRJS screensaver configuration for the Unix/Linux platform.

6.5. Interface

The web start application uses a small interface to indicate the steps of the installation. Each step will be displayed in the interface at completion, as shown Figure 6.5-1 and Figure 6.5-2. In case of a failure the interface describes the failed step and the exception caught.

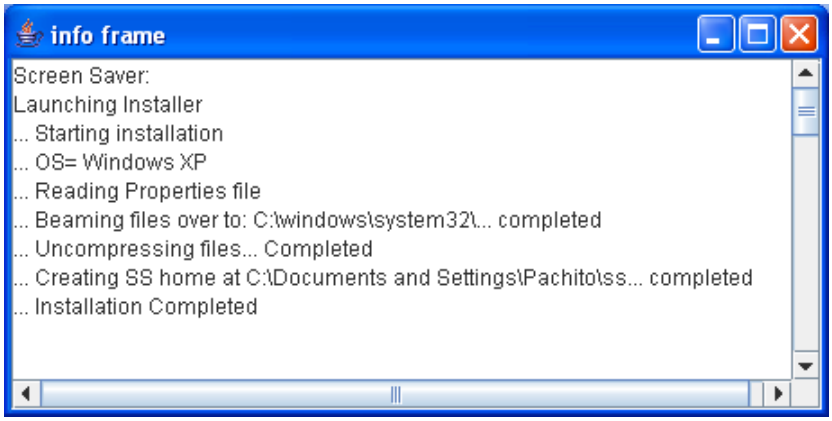

Figure 6.5-1 Installer interface indicating the installation steps for the Windows platform.

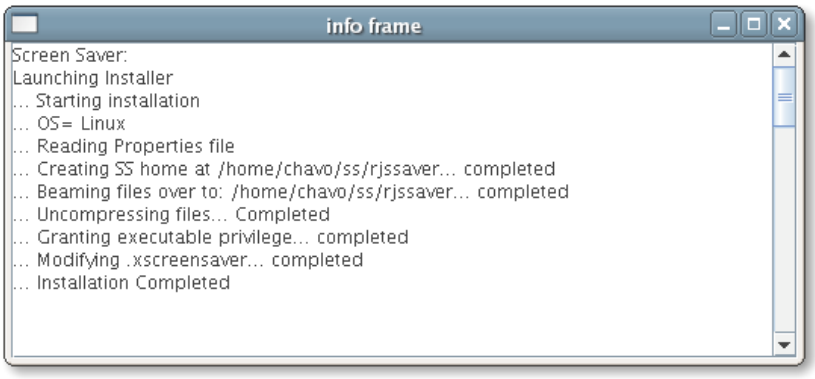

Figure 6.5-2 Installer interface indicating the installation steps for the Linux platform.

6.6. Summary

In this section describes the automation of the installation of the IRJS screensaver for the Unix/Linux and Windows platforms using a three step process. The first step, the system identifies the operating system of the user. Secondly, a file-transfer process called beam over is used to download the appropriate IRJS screensaver files, according to the OS identified. Lastly, the files are installed and configured based on the requirements of each OS. We use Java Web Start technology to execute the installation and deploy the IRJS screensaver to the user's computer.

7. THE SAVERBEANS SCREENSAVER AND IRJS SYSTEM INTEGRATION

This section describes the integration of our Java-based screensaver framework with our Initium Remote Job Submission (IRJS) grid computing middleware. Initium RJS is a Java Web Start (JAWS) based grid-computing technology. We provide an example of the transformation of a von Neumann style program into a concurrent program that makes use of our grid framework. We shall confine ourselves to a simple first example that makes use of a well-known fractal

computation called the Mandelbrot set. We also summarize the basis theory of operation of our grid framework.

7.1. Integration of a Screensaver and Grid System

We use a Compute Server (CS) that uses multicasting to discover a Lookup Server (LUS) over a local network. The CS announces that it is available, and provides benchmark data to the LUS. The benchmark data describes computer resources, and it is used by the LUS to allocate tasks to the CS. The LUS holds a pool of previously partitioned jobs that need to be processed, and it allocates these jobs to available resources. The task of the CS is to process or compute the job and to transmit the results back to the LUS [32]. The role of the screensaver is to detect user-computer quiescence and use this interval to volunteer CPU cycles to the LUS.

The *SaverBeans* [24] framework provides two methods that can be shadowed to alter the behavior of screensaver initiation and termination. These two methods are *init()* and *destroy()*, and they are defined in the abstract class *ScreensaverBase*. We subclass the *SimpleScreensaver* in order to create our own screensaver. The *init()* method is called during the screensaver startup. In our Java class IRJS Saver we have implemented the *init()* method to not only initiate the state of our screensaver, but also to invoke the CS, as shown in Example 7.1-1.

Example 7.1-1

```
public void init() {
    ScreensaverSettings settings = getContext().getSettings();
    Component c = getContext().getComponent();
    int width = c.getWidth();
    int height = c.getHeight();
    randomizePoint( p1, width, height );
….

 /*Initiate Compute Server and Monitor once. Init method is invoked more than
once during the execution the screensaver.
 */the variable iCount is static.

   iCount = iCount + 1;
        if (iCount < 2){
            startComputeServer();
            launchLogMonitor();
        }
 }
```

The *startComputeServer()* method called in Example 7.1-1 invokes the CS application using Java Web Start as shown in Example 7.1-2**.** Notice the parameters passed to the ***exec*** method in the Runtime object ***rt.*** They are the application name "javaws" (Java Web Start), the command – ***Xnosplash*** to avoid any splash screens, and the location or URL of the CS.

Example 7.1-2

```
    private void startComputeServer(){
    Runtime rt;
   Process p;
```

```
     String[] params = {"javaws", "-Xnosplash",
"http://www.myjavaserver.com/~fsophisco/" +
                  "net.rmi.pawelGrid.LusCs.CsMain.jnlp"};

            rt = Runtime.getRuntime();
            try{
                  p = rt.exec(params);
                  p.waitFor();

            }catch(Exception e){
                  System.out.println("Error @ startComputeServer()");
                  e.printStackTrace();
            }
    }
```

Example 7.1-1 shows a call to the *launchLogMonitor()* method and this is listed in Example 7.1-3**.** It has the purpose of creating a thread that periodically monitors and reads from the CS log file to find the state of the CS. We use the state of the CS to help create a display in the screensaver frame, and it occurs in the overwritten *paint()* method, which is discussed later in this section.

Example 7.1-3

```
private void launchLogMonitor(){

      Thread t =  new Thread(new Runnable(){
                  public void run(){
                        try{
                              while (true){
                                    Thread.sleep(5000);
                                    readCSLog();
                              }
                        }catch(Exception e){
                              e.printStackTrace();
                        }
                  }
                  //Read CS status from Log. Set message to be displayed.
                  public synchronized void readCSLog(){
                        try{
                              fout = new RandomAccessFile(cslogFile, "rw");
                                    message = fout.readLine();
                                    if (ms_length != message.length())
                                    {
                                          ms_length = message.length();
                                          new_ms = true;
                                    }
                                    fout.close();
                        }catch(Exception e){
                                    e.printStackTrace();
                        }
                  }
      });
      t.start();
}
```

7.2. Detection of User Input and Termination of Compute Server

The CS has been built with the capability of monitoring and detecting termination messages from an external application. In the case of receiving a termination message, the CS proceeds to perform any clean up and communicates with the LUS, and finally terminates execution. This process restores the CS to its initial state when the user returns to use his/her computer.

Subclasses of SimpleScreensaver can optionally implement the *destroy()* method to perform any cleanup before the screensaver is destroyed. In our case we want to communicate with the CS to inform that user input has been detected and that it needs to terminate itself. We accomplish this by creating a directory in a common place that the CS monitors periodically, as shown in Example 7.2-1**.**

Example 7.2-1

```
//file separator
private static String fileSep = System.getProperty("file.separator");

//  java temp
private static String tmpDir = System.getProperty("java.io.tmpdir");

 //  kill file
public final static File killFile = new File(tmpDir +
                                    fileSep +
                                    "killcs");


….
protected void destroy(){

          cal= Calendar.getInstance();
          killFile.mkdir();
          System.out.println("CS Stoping at "+ cal.getTime().toString());

 }
```

7.3. Painting the Next Frame

Lastly we have shadowed the method *paint()* to enable a screen display. The method *LaunchLogMonitor()* shown in Example 7.1-3 starts a monitor to periodically obtain the status of the computer and to place it, in a string format, into an instance variable. For the purpose of observing the status of the CS in the screen, we have included the string message obtained from the monitor as shown in Example 7.3-1**.** The String simply bounces against the walls, and it is changed every time the CS status changes (Figure 7.3-1 & Figure 7.3-2).

Example 7.3-1

```
public void paint( Graphics g ) {

            Component c = getContext().getComponent();
           int width = c.getWidth();
           int height = c.getHeight();
           Point c_point = new Point();
```

```
            //Write SS Name
            g.setFont(new Font("Arial", Font.BOLD , 25));
            g.setColor(Color.red);
            g.drawString(ssName, 30, 30);

        ….

            g.setFont(new Font("Arial", Font.BOLD , 25));
            g.drawString(message, p1.x, p1.y);

            // Erase old drawings:
            Point pe = points[indx_b];
            if (pe != null)
            {
               g.setColor( c.getBackground() );
               g.setFont(new Font("Arial", Font.BOLD , 25));
               g.drawString(message, pe.x, pe.y);
            }

            // Move points and bounce off walls:
            bounce( p1, dir1, width, height );

            //Draw String
            g.setColor(Color.blue);
            g.setFont(new Font("Arial", Font.BOLD , 25));
            g.drawString(message, p1.x, p1.y);
    ….
}
```

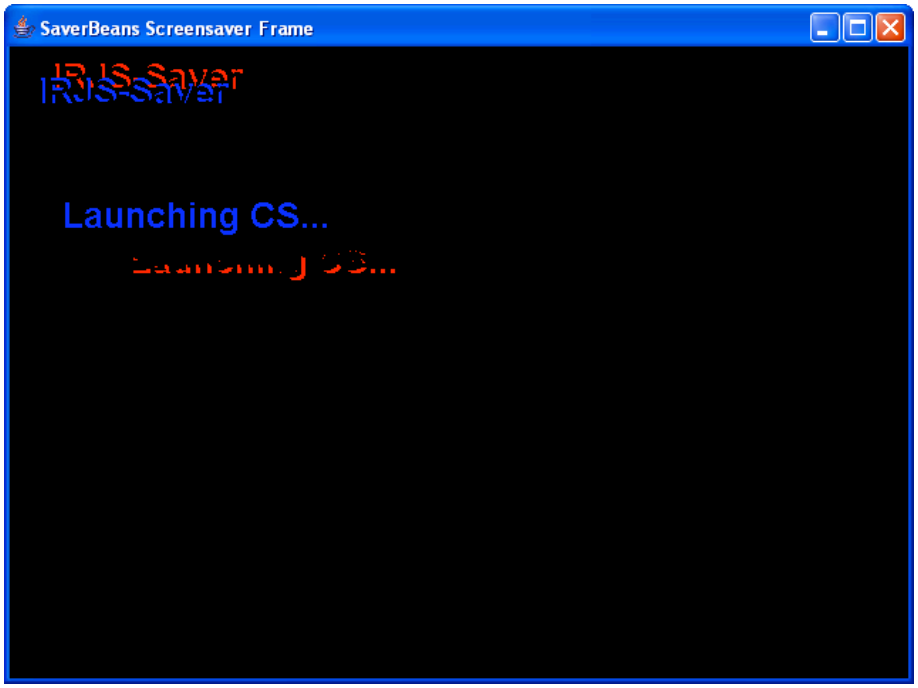

Figure 7.3-1 IRJS Screensaver: Compute Server is launching

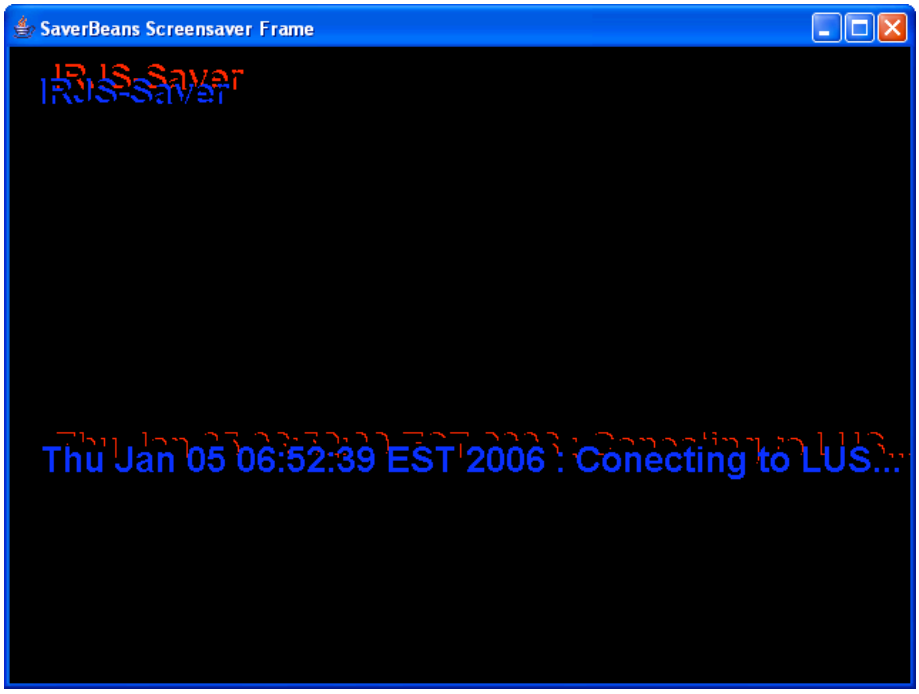

Figure 7.3-2 IRJS Screensaver: Compute Server is connecting to LUS

7.4. Partitioning a Von Neumann-Style Sample Program

To demonstrate the use of the IRJS System, computable jobs must be submitted to it. In this section, our goal is to take a von Neumann style program and to partition it into executable pieces. The initial sample program produces an image of the Mandelbrot set. The main goal is to process an image using the Mandelbrot set algorithms, and to display the output image into a frame as shown in Example 7.4-1.

Example 7.4-1

```
public void testMandelbrot() {
        final Dimension dim = new Dimension(400, 400);
        ClosableJFrame cjf = new ClosableJFrame("mandlebrot") {
            public void paint(Graphics g) {
                Dimension d = getSize();
                Image i = getMandelbrot(d.width, d.height);
                g.drawImage(i, 0, 0, null);
            }
        };
        cjf.setSize(dim.width, dim.height);
        cjf.setVisible(true);
}
```

The *getMandelbrot()* method, called in Example 7.4-1, takes two parameters, the width and height of the image to be processed. It uses these parameters to create an image bean, where all

the data is stored. It continues to call the *Mandelbrot()* method and passes the RGB arrays from the image bean. Example 7.4-2 shows *getMandelbrot()* method.

Example 7.4-2

```
public Image getMandelbrot(int w, int h)
{
      ShortImageBean sib = new ShortImageBean(w, h);
      mandelbrot(sib.getR(), sib.getG(), sib.getB());
      return sib.getImage();
}
```

The *Mandelbrot()* method applies the Mandelbrot set algorithms and generates data into the RBG bean arrays as shown in Example 7.4-3.

Example 7.4-3

```
public void mandelbrot(short[][] r, short[][] g, short[][] b) {
        int height = r[0].length;
        int width = r.length;
        int Clr;
        float pixelr, pixeli;
        for (int y = 0; y < height; y++)
            for (int x = 0; x < width; x++) {
                pixelr
                        = mandleBrotDimensions.getxMin() +
                        (float) x / width *
                        (
                        mandleBrotDimensions.getxMax() -
                        mandleBrotDimensions.getxMin());
                pixeli
                        = mandleBrotDimensions.getyMin() +
                        (float) y / height *
                        (
                        mandleBrotDimensions.getyMax() -
                        mandleBrotDimensions.getyMin());
                Clr = getColor(pixelr, pixeli);
                if (Clr == -1) {
                    r[x][y] = 255;
                    g[x][y] = 128;
                    b[x][y] = 0;
                } else {
                    r[x][y] = mandelTables.colorR[Clr %
mandelTables.maxIter];
                    g[x][y] = mandelTables.colorG[Clr %
mandelTables.maxIter];
                    b[x][y] = mandelTables.colorB[Clr %
mandelTables.maxIter];
                }
            }
    }
```

The output image is shown in Figure 7.4-1.

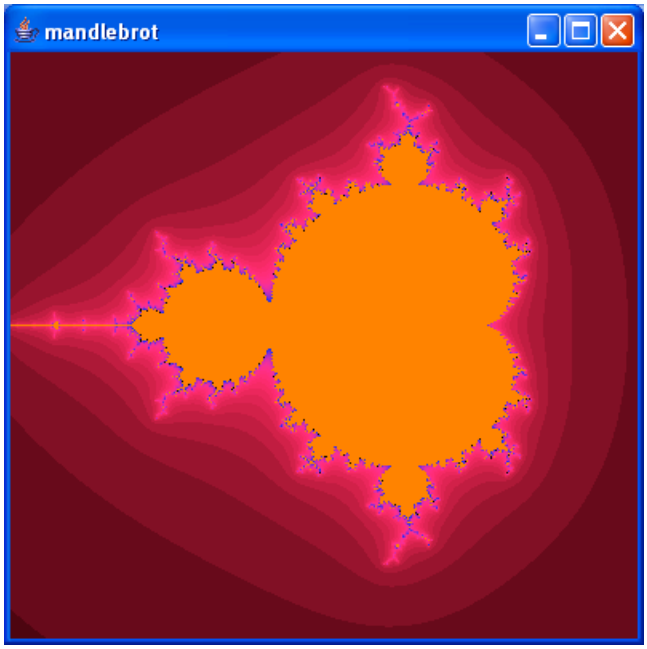

Figure 7.4-1. Shows the output image from the Von Neumann style program

The next task is to modify the sample program, so the task at hand can be divided into smaller logical parts. The *Mandelbrot* method was overloaded to apply the Mandelbrot set algorithm to a single point as shown in Example 7.4-3.

Example 7.4-3

```
public void mandelbrot(int x, int y, short[][] r, short[][] g, short[][] b) {
        int height = r[0].length;
        int width = r.length;
        int Clr;
        float pixelr, pixeli;

        pixelr
                = mandleBrotDimensions.getxMin() +
                (float) x / width *
                (
                mandleBrotDimensions.getxMax() -
                mandleBrotDimensions.getxMin());
        pixeli
                = mandleBrotDimensions.getyMin() +
                (float) y / height *
                (
                mandleBrotDimensions.getyMax() -
                mandleBrotDimensions.getyMin());
        Clr = getColor(pixelr, pixeli);
        if (Clr == -1) {
            r[x][y] = 255;
            g[x][y] = 128;
            b[x][y] = 0;
        } else {
            r[x][y] = mandelTables.□olor[Clr % mandelTables.maxIter];
```

```
            g[x][y] = mandelTables.colorG[Clr % mandelTables.maxIter];
            b[x][y] = mandelTables.colorB[Clr % mandelTables.maxIter];
        }
    }
```

Using this method we synthesize selected areas of the image as show in Example 7.4-4 and Figure 7.4-2.

Example 7.4-4

```
short[][] r = sib.getR();
short[][] g = sib.getG();
short[][] b = sib.getB();

int height = 150;
int width = r.length;

for (int y = 0; y < height; y++)
     for (int x = 0; x < width; x++) {

          mandelbrot(x, y, r, g, b);
 }
```

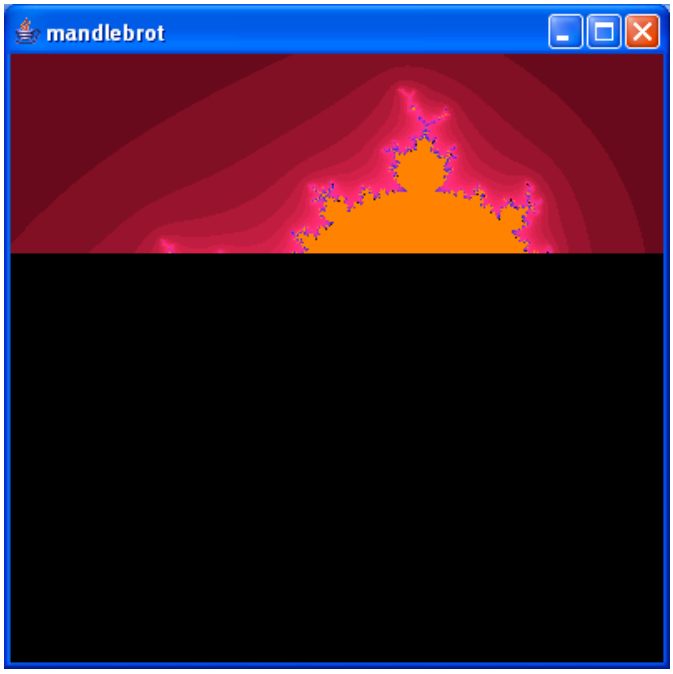

Figure 7.4-2. Partial processed area of an image

Part of the requirements of the IRJS System is that each single job submitted must generate a jar file with the answer of the job as contents. In this example, it is a partial image. Example 7.4-5 shows the method used by independent jobs that will work with different sections of the image. Part of the parameters passed to this method is the region of the image, and the jar file name for the answer. The result of this method is a jar file containing a section of the image.

Example 7.4-5

```
public static void computeStrip(Dimension d, Point position, Point from,
Point to, String outputFileName) {

      ShortImageBean sib = new ShortImageBean(d.width, d.height);
      FractalShortImageBean s = new FractalShortImageBean(position, from,
to);
      FractalLogic fl = new FractalLogic();

           s.setR(sib.getR());
           s.setG(sib.getG());
           s.setB(sib.getB());
           System.out.println("Processing piece Image fp=" + from.x + " tp="
+ to.y);
           for (int y = from.y; y < to.y; y++) {
               for (int x = from.x; x < to.x; x++) {
                   fl.mandelbrot(x, y, s.getR(), s.getG(), s.getB());
               }
           }

   String ds = SystemUtils.getUserHome() +
SystemUtils.getDirectorySeparator() +
                   "rjs" + SystemUtils.getDirectorySeparator();

           File f = new File(ds);
           if (!f.exists()) f.mkdir();
           String fn = ds + outputFileName;
           f = new File(fn);

           Image i = s.getProcessedImage();
           ImageUtils.saveAsPPMJar(i, f);   // saving the image as jar
           System.exit(0);
    }
```

The last step is to create the jobs. For this example we have created eight jobs. Example 7.4-6 shows the code for one of them. The original program was also modified to read the images from jar files and to build them together to form the original output image.

Example 7.4-6

```
public class FractalsJob_1 {
    public static void main(String[] args) {
        Point from = new Point(0, 0);
        Point to = new Point(400, from.y + 100);
        Utils.computeStrip(from, to, "Fractals_out1.ppm.jar")}}
```

7.5. Summary

A screensaver is capable of detecting user input. We have used this feature to serve as a launching facility for the CS. Through this process the resources of a computer maybe volunteered and utilized close to 100%. The IRJS Screensavers is minimally intrusive, detects user input, and terminates any additional execution of the CS. A von Neumann style program was partitioned it into executable pieces to demonstrate the use of the IRJS System. In addition, we have put to use technologies that promise cross platform solutions such as Java, Java Web Start and Saverbeans.

8. SUBMITTING JOBS TO THE IRJS SYSTEM

In this section we describe an example on how to write jobs that can be submitted to the IRJS System. We use the Mandelbrot Set [38] to create jobs that are suitable for submitting to the IRJS system. The IRJS system accepts tasks or jobs that have the following characteristics:

- They are written in the Java language.
- The class to execute has a *main(...)* method.
- They are independent tasks that do not require any user input during their execution.
- They are known to be large CPU-time consumers.
- They do not require any GUI.
- They are deployed using Java Web Start.
- The outputs of the jobs are written into a jar file.

We have created eight different jobs that use the Mandelbrot algorithms to process a large image. Each job processes a section of the image which horizontal and vertical locations are specified as parameters of the job as shown in Example 8-1.

Example 8-1

```
public class FractalsJob_1 {

    public static void main(String[] args) {
        Point from = new Point(0, 0);
        Point to = new Point(400, from.y + 100);
        Utils.computeStrip(from, to, "Fractals_out1.ppm.jar");
    }
}
```

The jobs are packaged and deployed as Java Web Start applications into a web server using the Initium[5] utility. Initium performs a dependency study of the class that will be deployed, in our case class *FractalsJob_1*. It continues to package only the dependencies to this class and generates a jar file. The jar file is signed with the credentials of the IRJS system, and a JNLP file is generated. These two outputs, the signed jar file and the JNLP file, are deployed to the web server where they will be available to the IRJS system. Initium provides an interface that is used to enter all the parameters necessary to package and deploy the jobs as shown in Figure 8-1.

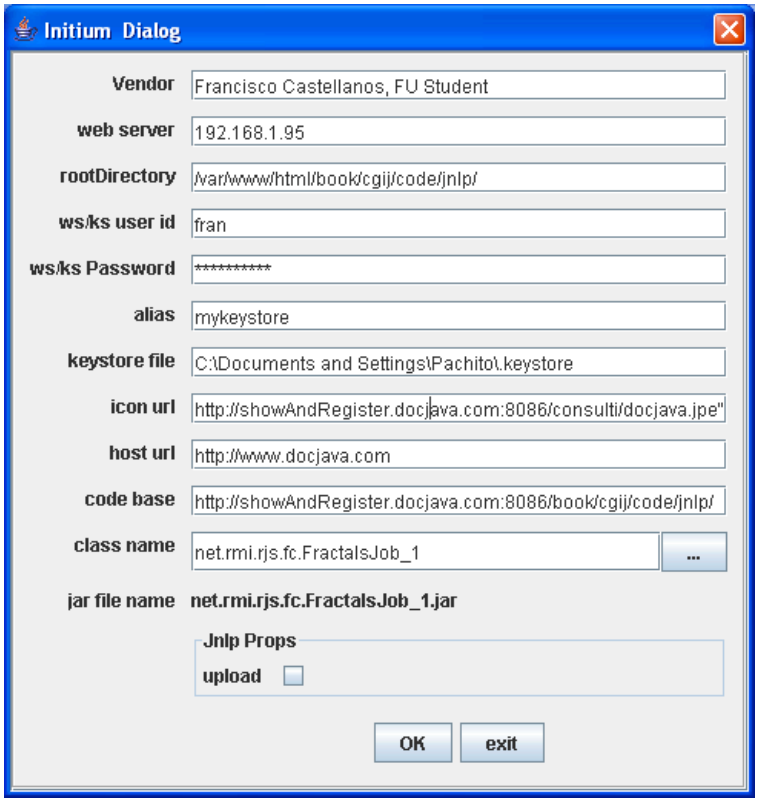

Figure 8-1 Initium Graphical User Interface

The results from the Initium utility are eight Java Web Start applications as shown in figure 8-2. Each JWS application is ready to process a section of an image applying the Mandelbrot algorithms.

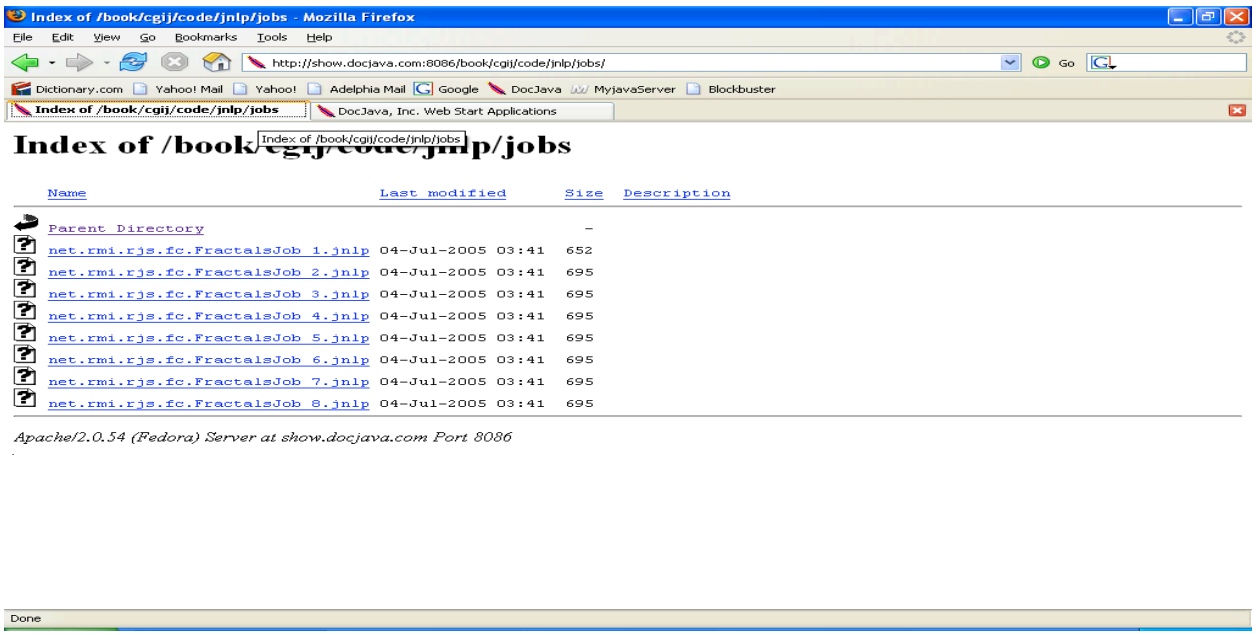

Figure 8-2 Results of Initium: List of eight Java Web Start applications

9. IRJS SYSTEM PROCESSING JOBS

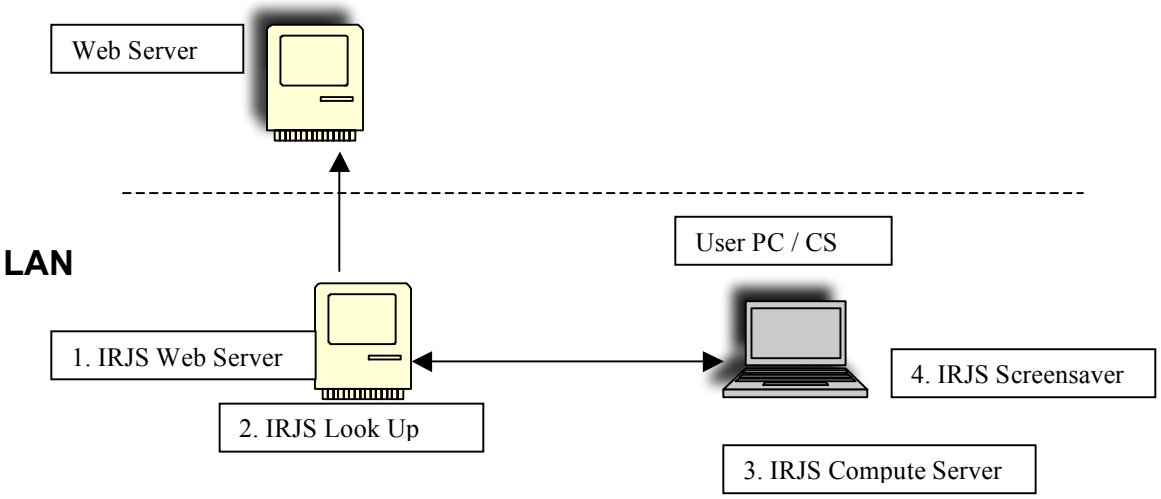

Figure 9-1 IRJS components

The IRJS System is composed of four main modules as depicted in figure 9-1. These four components are stand-alone JWS applications. The first module, the IRJS Web Server located at [37], has the responsibility of pulling jobs on demand from an external location (for our example, it is the location where the eight jobs are placed). The second module, the Lookup serve (LUS) located at [35], is responsible of managing several activities including job-resource matching, available resources, leasing, and feeding answers back to the web server module. The third component is the Compute Server (CS), located at [34]. The CS is triggered by the fourth component, the IRJS screensaver, located at [36]. The CS announces its availability and benchmark data to the LUS and waits for jobs to be submitted. Figure 9-2 shows the events mentioned between the IRJS system components.

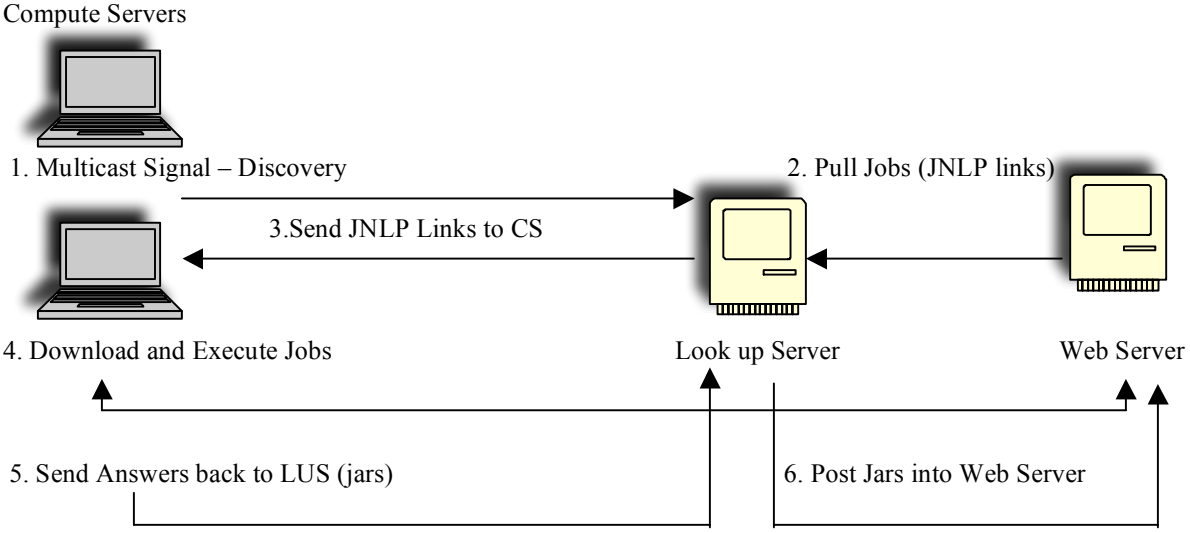

Figure 9-2. IRJS System Events

For the example at hand, all the components must be launched. We use a separate server to launch the IRJS Web server and the LUS. On separate computers we install the IRJS screensaver. The screensaver display status messages as shown in figures 7.3-1 and 7.3-2.

The interface of the LUS is used to start the execution of the system when all components are running. Figure 9-3 shows the LUS interface depicting the available compute servers and their characteristics.

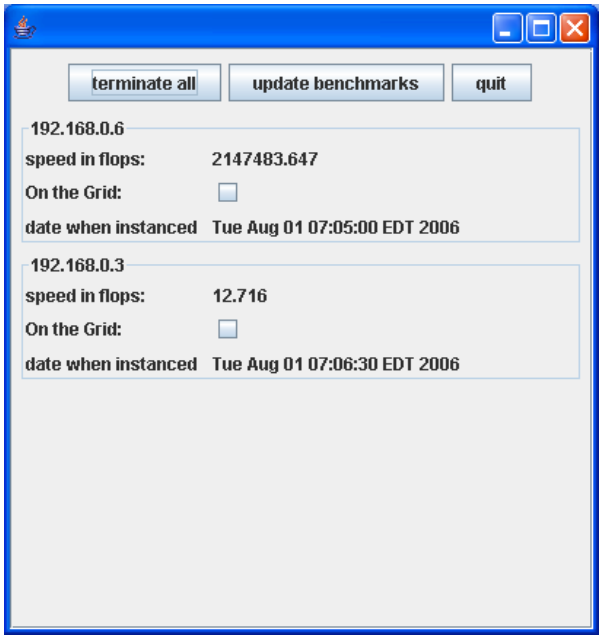

Figure 9-3 LUS interface shows available Compute Servers.

We use the LUS interface to pull available jobs by clicking the Get Jobs button (Figure 9-4). Once the jobs are downloaded they are matched to available CS(s) and deployed. The CS(s) execute the jobs and returns jar files as answers. When the LUS receive the answers, it logs them on the interface as shown in figure 9-4.

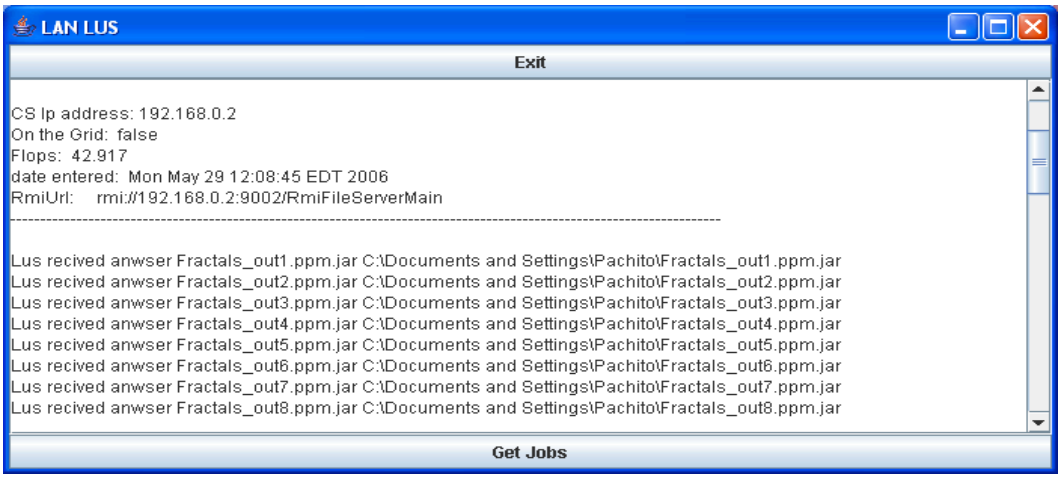

Figure 9-4. Once jobs are completed and answers received, they are logged into the LUS interface.

Lastly, we use the eight answers to build the whole image as shown in figures 9-5 and 9-6.

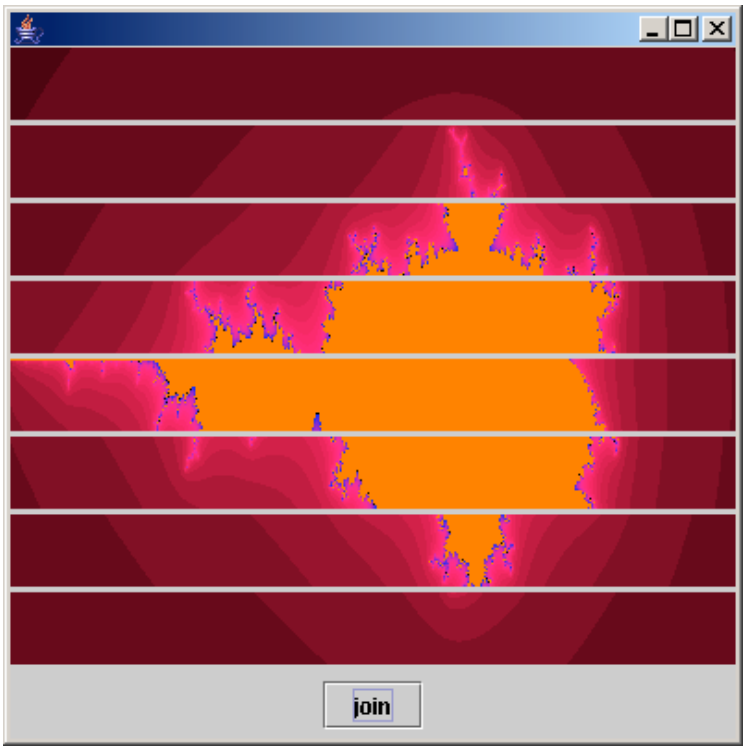

Figure 9-5. This is the set of eight different Images already processed (Mandelbrot Set) by the IRJS System.

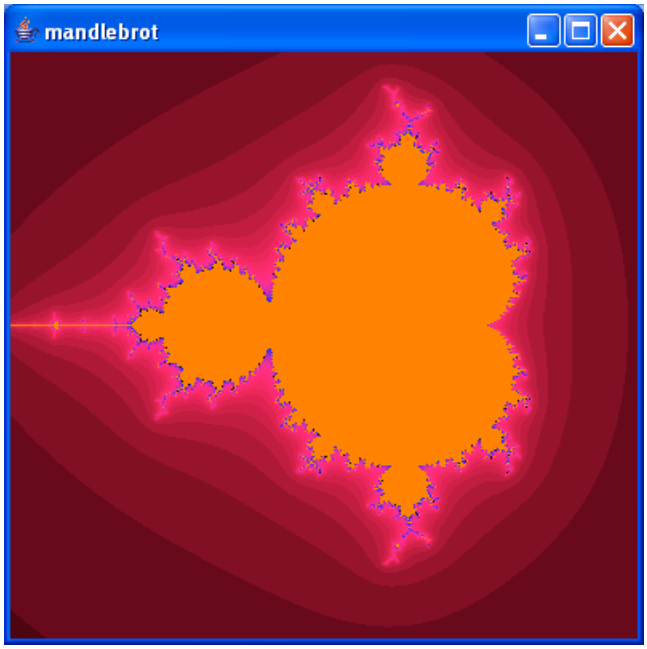

Figure 9-6. The complete processed image.

10. CONCLUSION

In this thesis we presented the IRJS screensaver that performs CPU scavenging for the use of grid computing. The IRJS screensaver integrates with the IRJS middleware to volunteer, otherwise wasted, CPU cycles to the grid. The IRJS screensaver makes use of computer resources during the period of user-computer quiescence. Typically, computers are used between 40 and 60 hours out of a 168-hour week. This represents a 35% utilization. We have provided solutions for discovery of user-computer quiescence, intrusion minimization, and portability. We used the Saverbeans Framework to develop a portable screensaver for the Unix and Windows platforms. We used a customized solution to develop a screensaver in the Macintosh platform. We made use of Java Web Start to deploy and install the IRJS screensaver into the computers to be volunteered. The IRJS screensaver is used as a launching facility to the CS. The CS uses multicasting to discover the LUS, it provides local benchmark data and volunteers its resources. The IRJS screensaver kills any execution of the CS at any detection of user input.

10.1. Experimental Results

We experimented the IRJS system to observe the benefits and disadvantages that the system offers. The eight jobs created for the Mandelbrot set example, mentioned in section 9, were used to benchmark the IRJS system. Four computers were used, and the IRJS screensaver was installed in each of them for the experiment. To notice the processing time gained, the amount of total processing time for the eight jobs was taken in different scenarios. We first used a single computer and then incremented the amount of computers by one for each experiment. The four computers have the following characteristics:

In order of addition:

A. Linux Fedora Core 4, PIII, 500 MHz, 512 m Ram.
B. Windows XP, Celeron M, 1.5 GHz, 512m Ram
C. Linux Fedora Core 2, Celeron, 500 MHz, 512 Ram
D. Windows XP, P4, 3.0GHz, 1G Ram

Experiment #	Experiment Desc.	Computers	Number of Jobs	Total Processing Time
1	One CS	A	8	2m 53 sec
2	Two CS(s)	A, B	8	1m 7 sec
3	Three CS(s)	A, B, C	8	1 m 2 sec
4	Four CS(s)	A, B, C, D	8	50 sec

Table 10.1-1. Experimental Results

As shown in Table 10.1-1, we observed that the amount of processing time decreases as the amount of computers/CS(s) increases. This is perhaps the greatest benefit of the computing grid. Between experiment 1 and 2, the jobs were processed in 1 minute and 46 seconds less, or in ~ 38% of the initial processing time. Between experiment 2 and 3, the jobs were processed in 5 seconds less, or 92% of the processing time in experiment 2. Between experiment 3 and 4 the jobs were processed in 12 seconds less, or ~ 80% of the processing time in experiment 3.

Our second observation is that the IRJS System is as strong as the weakest link. The smallest gain in time was observed after computer C was added to experiment 3. This computer was the slowest computer of the group. In experiment 4, the CS in computer C was the last to complete, 10 seconds later than the rest of the computers, that represents ~ 25% more time. The Graph in Figure 10.1-1 shows both of the observations mentioned.

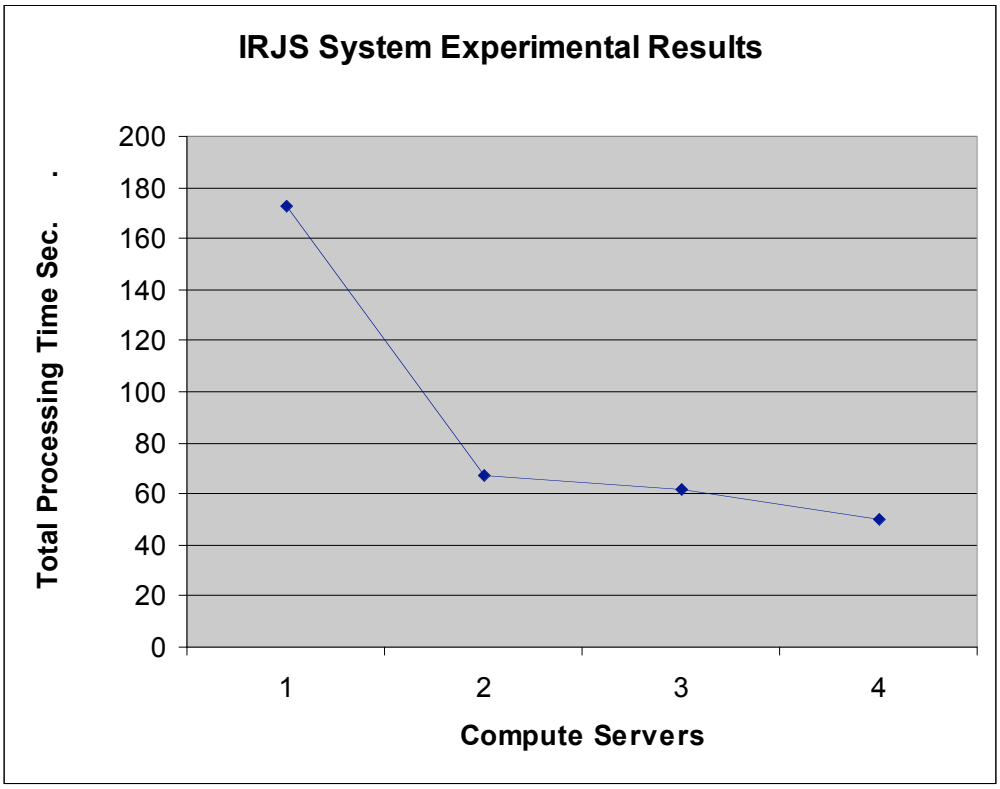

Figure 10.1-1 Graph of experimental results

10.2. Known Issues

Screensaver portability is challenging due to different standards used in each platform. For example, in Windows, screensavers are supported differently in each version of this OS. In particular, the locations where screensaver files reside changes, and in most cases these locations are part of the OS itself. Therefore, a network administrator may choose to protect these locations. In this case the automatic installation of the IRJS screensaver would fail. On the Unix platform the IRJS screensaver relies on an external software, Xscreensaver. This software is known to be part of the OS installation package for several brands of Unix/Linux such as Fedora and Red Hat. However, if it is not installed, the process to install it and make it run is not easy.

Currently, the binaries of the IRJS Screensaver for Unix/Linux have been compiled and generated using a Fedora Core 2 system. These binaries are known to work on Fedora Core 3 and Core 4. However, they have not been tested in other brands of Unix or Linux. Therefore the binaries might have to be generated for other flavors of Unix/Linux.

The Saverbeans framework requires JDK 1.5, although future releases promise to support earlier versions of JDK. The Saverbeans framework allows developers to build portable Java based screensavers for Unix and Windows, however, the Macintosh OS is not currently supported. Therefore we developed a customized solution for the Macintosh OS.

10.3. Future Work

The implementation of the Macintosh screensaver using the Saverbeans framework was left for future work since the Saverbeans frameworks is not currently supporting this OS. An improved strategy for tasks-resource matching was left for future work. Currently, the IRJS system uses the first come-first serve strategy to match resources to tasks. For this process to be improved, we would need to know more about the jobs submitted to the IRJS system, such as their type and characteristics. Therefore an improved interface to submit jobs to the grid is needed. The issue of job partitioning is left for future work. Currently partitioning is accomplished manually by the programmer. We look to provide a framework that would allow this process to be automatic.

11. REFERENCES

[1] Douglas Lyon: "Asynchronous RMI for CentiJ", in *Journal of Object Technology*, vol. 3, no. 3, March-April 2004, pp. 49-64. http://www.jot.fm/issues/issue_2004_03/column5

[2] Sun Microsystems: "Java Web start Technology", http://java.sun.com/j2se/1.4.2/docs/guide/jws/Readme.html

[3] Edward Harned: "A Java RMI server framework", October 2001, http://www-106.ibm.com/developerworks/java/library/j-rmiframe/

[4] Anthony Karre: "A do-it-yourself framework for Grid Computing", April 2003, http://www.javaworld.com/javaworld/jw-04-2003/jw-0425-grid_p.html

[5] Project Initium: Programmatic Deployment by Douglas A. Lyon, Journal of Object Technology, vol. 3, no. 8, September-October 2004, pp. 55-69 http://show.docjava.com:8086/pub/document/jot/web.pdf Last access August 5, 2006

[6] Zoltan Juhasz, Kristian Kuntner, Mark Magyarodi, Gabor Major, Szabolcs Pota: "JGrid Design Document", June 2003.

[7] James Linn and Douglas Lyon: "Java For Programmers: Chapter 43 RMI", January 2004.

[8] Steve Kim: "Java Web Start", September 2001, http://www-106.ibm.com/developerworks/java/library/j-webstart/

[9] Sun Microsystems: "Developer's Guide Java ™ Web Start Technology" http://java.sun.com/products/javawebstart/docs/developersguide.html

[10] Sun Microsystems: "Trial RMI", http://java.sun.com/docs/books/tutorial/rmi/

[11] Rajkumar Buyya: "Grid Computing Info Center: Frequently Asked Questions(FAQ)" , http://gridcomputing/gridfaq.html

[12] Mark Baker, Rajkumar Buyya and Domenico Laforenza: "Grids and Grid technologies for wide-area distributed computing", http://www.gridbus.org/papers/gridtech.pdf, SOFWARE-PRACTICE AND EXPERTICE, 2002, John Wiley and Sons,Ltd.

[13] Ian Foster: "The Grid: A new Infrastructure for 21st Century Science", American Institute of Physics, 2002, http://www.aip.org/pt/vol-55/iss-2/p242.html

[14] I. Foster, C. Kesselman: "The Globus Project: A Status Report" *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*, pp. 4-18, 1998, http://studies.ac.upc.edu/FIB/DSO/papers/globus-hcw98.pdf

[15] Rajkumar Buyya, Srikumar Venugopal: "The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report", Department of Computer Science and Software Engineering, University of Melbourne, Australia, http://www.gridbus.org/papers/gridbus2004.pdf

[16] Gabrielle Allen, Kelly Davis, Konstantinos N. Dolkas, Nikolaos D. Doulamis, Tom Goodale, Thilo Kielmann1, André Merzky, Jarek Nabrzyski, Juliusz Pukacki, Thomas Radke, Michael Russell, Ed Seidel, John Shalf and Ian Taylor: "Enabling Applications on the Grid: A GridLab Overview", 2003, http://www.gridlab.org/Resources/Papers/ijhpca_gridlab_2003.pdf

[17] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman: "Grid Information Services for Distributed Resource Sharing". Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001. http://www.globus.org/alliance/publications/papers/MDS-HPDC.pdf

[18] D. Angulo, I. Foster, C. Liu, and L. Yang: "Design and Evaluation of a Resource Selection Framework for Grid Applications". *Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC-11),* Edinburgh, Scotland, July 2002. http://people.cs.uchicago.edu/~dangulo/papers/hpdc-resource-selector.pdf

[19] A. Iamnitchi and I. Foster: "On Fully Decentralized Resource Discovery in Grid Environments". *International Workshop on Grid Computing, Denver, CO, November 2001*. http://www.globus.org/alliance/publications/papers/GC2001.pdf

[20] Boinc: http://boinc.berkeley.edu/ Last accessed March 14, 2005.

[21] Cygwin: **http://www.cygwin.com** Last accessed March 14, 2005.

[22] JDIC1: Java.net : "JDIC project home", https://jdic.dev.java.net/ Last accessed March 14, 2005.

[23] JDIC2: https://jdic.dev.java.net/documentation/incubator/screensaver/index.html Last accessed March 14, 2005.

[24] SaverBeans: https://jdic.dev.java.net/documentation/incubator/screensaver/index.html Last accessed March 14, 2005.

[25] William L. George and Jacob Scott, "Screen Saver Science: Realizing Distributed Parallel Computing with Jini and JavaSpaces" in *2002 Conference on Parallel Architectures and Compilation Techniques* (PACT2002), Charlottesville, VA, September 22-25, 2002.

[26] Private communications with William L. George, Ph.D., National Institute of Standards and Technology, 100 Bureau Dr. Stop 8911, Gaithersburg, MD 20899-8911, email: wgeorge@nist.gov, March 15, 2006.

[27] Jamie Zawinski: "A screen saver and locker for the X Window System" http://www.jwz.org/xscreensaver/

[28] Brian Christensen: "Writing a Screensaver Module", April 10, 2001, http://www.cocoadevcentral.com/articles/000011.php

[29] Douglas A. Lyon and Christopher L. Huntley: "There's More Than One Way to Build a Bridge", *Computer*, May, 2002, pp. 102-103.

[30] Douglas A. Lyon: "*Java for Programmers"*, Prentice Hall, Feb. 2004.

[31] Andy Monitzer: "The Java Bridge", March 17, 2002, http://www.cocoadevcentral.com/articles/000024.php

[32] Remote Job Submission Security by Pawel Krepstzul and Douglas A. Lyon, in Journal of Object Technology, vol. 5, no. 1, January-February 2006, pp. 13-29. http://show.docjava.com:8086/pub/document/jot/rjs.pdf Last access August 5, 2006

[33] Project Initium: Programmatic Deployment by Douglas A. Lyon, Journal of Object Technology, vol. 3, no. 8, September-October 2004, pp. 55-69 http://show.docjava.com:8086/pub/document/jot/web.pdf Last access August 5, 2006

[34] IRJS CS http://show.docjava.com:8086/book/cgij/code/jnlp/net.rmi.rjs.pk.main.CSMainForSS.jnl Last access September 16, 2006

[35] IRJS LUS http://show.docjava.com:8086/book/cgij/code/jnlp/net.rmi.rjs.pk.main.Cr320Lus.jnlp Last access September 16, 2006

[36] IRJS SS http://show.docjava.com:8086/book/cgij/code/jnlp/net.rmi.rjs.fc.ssinstaller.Main.jnlp Last access September 16, 2006

[37] IRJS Web Server http://show.docjava.com:8086/book/cgij/code/jnlp/net.rmi.rjs.pk.main.WebServerMain.jnlp Last access September 16, 2006

[38] Mandelbrot set. (2006, August 5). In *Wikipedia, The Free Encyclopedia*. Retrieved 18:44, August 5, 2006, from http://en.wikipedia.org/w/index.php?title=Mandelbrot_set&oldid=67754296.