version 2.1 (October 2004)

GUIDED TOUR

This tour is a practical introduction to ProActive.

First you will get some practical experience on how to program using ProActive. This will help your understanding of the library, and of the concepts driving it.
Second, you will be guided through some examples to run on your computer; this way, you will get an illustrated introduction to some of the functionnalities and facilities offered by the library.

Alternatively, you can just have a look at the second part, where you just have to run the applications. In that case, do not worry about answering the questions involving code modifications.

If you need further details on how the examples work, check the ProActive applications page.

Permissions in the JavaTM 2 SDK

```
    log4j:WARN No appender could be found for logger .....
    log4j:WARN Please initialize the log4j system properly
```

Examples and compilatin0im1eRN work at all under Windows system: Check if your java installatin0iiseRN in a directory with spaces like

Hello world ! example

This example implements a very simple client−server application. A client object display a `String` received from a remote server. We will see how to write classes from which active and remote objects can be created, how to find a remote object and how to invoke methods on remote objects.

The two classes

Only two classes are needed: one for the server object `Hello` and one for the client that accesses it `HelloClient`.

The Hello class

This class implements server−side functionalities. Its creation involves the folloTing steps:

- Provide an implementation for the required server−side functionalities
- Provide an empty, no−arg constructor
- Write a `main`

Why an empty no−arg constructor ?

You may have noticed that class `Hello` has a constructor with no parameters and an empty implementation. The presence of this empty no−arg constructor is imposed by ProActive and is actually a side−effect of ProActive's transparent implementation of active remote objects (as a matter of fact, this side−effect

Printing out the message

```
windows clientHost> java −Djava.security.policy=scripts\unix\proo0c5ve.java.policy
−Dlog4j.configuration=file:scripts\unix\proo0c5ve−log4j
org.objectweb.proo0c5ve.examples.hello.HelloClient //remoteHost/Hello
```

[redacted]

```
        // the termination of the activity is done through a call on the
        // terminate method of the body associated to the current active object
        ProActive.getBodyOnThis().terminate();
}
```

1.3.3. Programming

a) the MigratableHello class

The code of the MigratableHello class is here.

MigratableHello derives from the Hello class from the previous example

MigratableHello being the active object itself, it has to :

− implement the Serializable interface

− provide a no−arg constructor

− provide an implementation for using ProActive's migration mechanism.

1.4. migration of graphical interfaces

Graphical interfaces are not serializable, yet it is possible to migrate them with ProActive.
The idea is to associate the graphical object to an active object. The active object will control the activation and desactivation of this graphical entity during migrations.

Of course, this is a very basic example, but you can later build more sophisticated frames.

Design of the application

We will write a new active object class, that extends MigratableHello. The sayHello method will create a window containing the hello message. This window is defined in the class HelloFrame

Programming

HelloFrameController

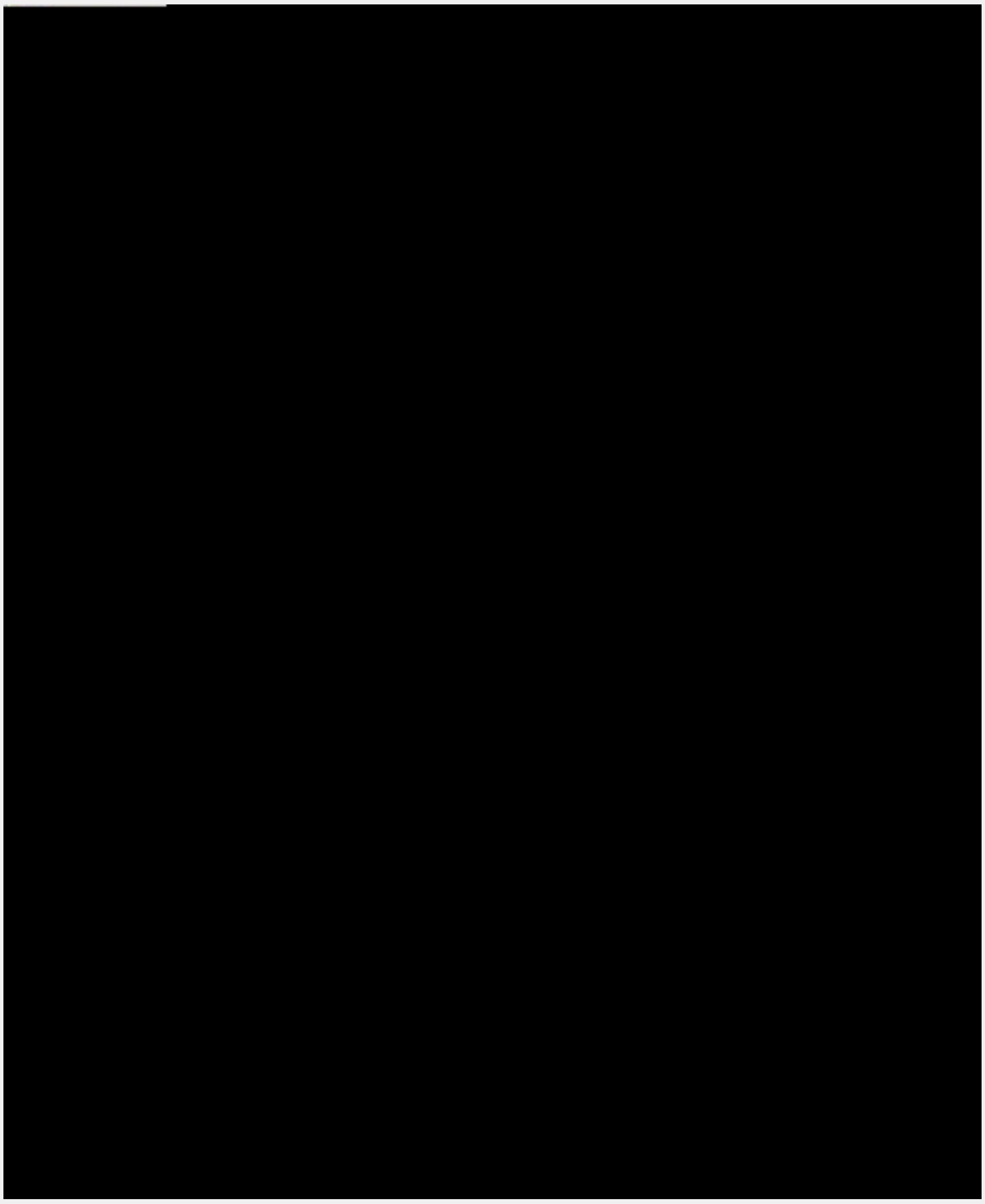
2. Introduction to some of the functionalities of ProActive

This chapter will present some of the facilities offered by ProActive, namely :

– synchronization

– parallel processing

– migration



3. test the autopilot mode

The application runs by itself without encountering a deadlock.

4. test the manual mode

Click on the philosophers' heads to switch their modes

Test that there are no deadlocks!

Test that you can starve one of the philosophers (i.e. the others alternate eating and thinking while one never eats!)

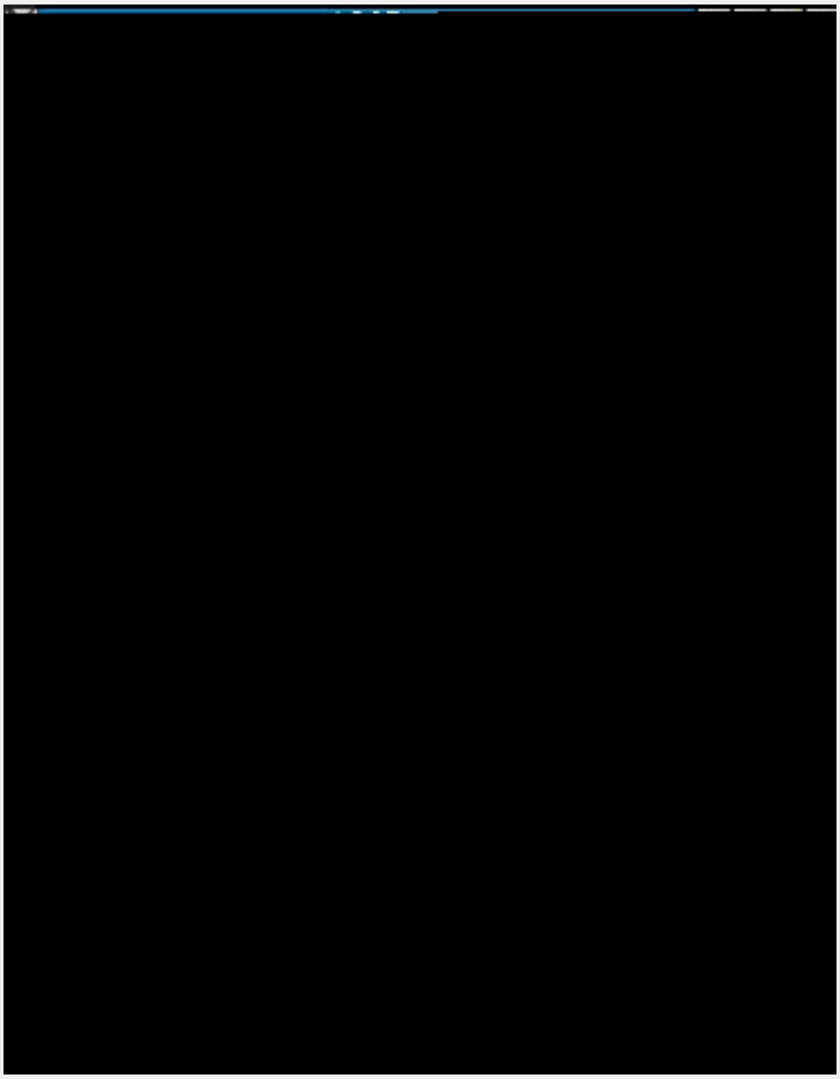
5. start the IC2D application

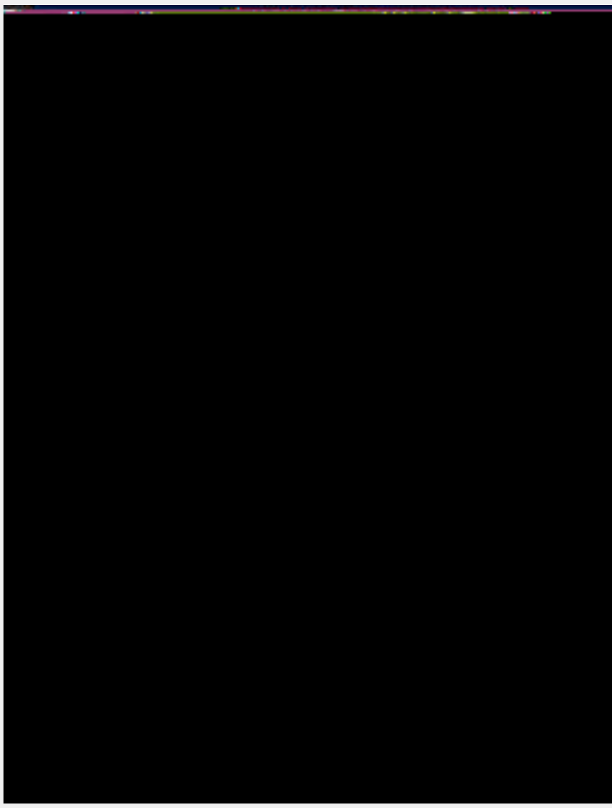
IC2D is a graphical environment for monitoring and steering of distributed and metacomputing applications.

− being in the autopilot mode, start the IC2D visualization application (using ic2d.sh or ic2d.bat)

menu monitoring − monitor new RMI host

It is possible to visualize the status of each active object (processing, waiting etc...), the communications between active objects, and the





2.2. Parallel processing with ProActive

Distribution is often used for CPU−intensive applications, where parallelism is a key for performance.

A ty3ical application is C3D.

Note that parallelisation of programs can be facilitated with ProActive, thanks to asynchronism method calls, as well as group communications.

C3D : a parallel, distributed and collaborative 3D renderer

C3D is a Java benchmark application that measures the performance of a 3D raytracer renderer distributed over several Java virtual machines using Java RMI. It showcases some of the benefits of ProActive, notably the ease of distributed programming, and the speedup through parallel calculation.

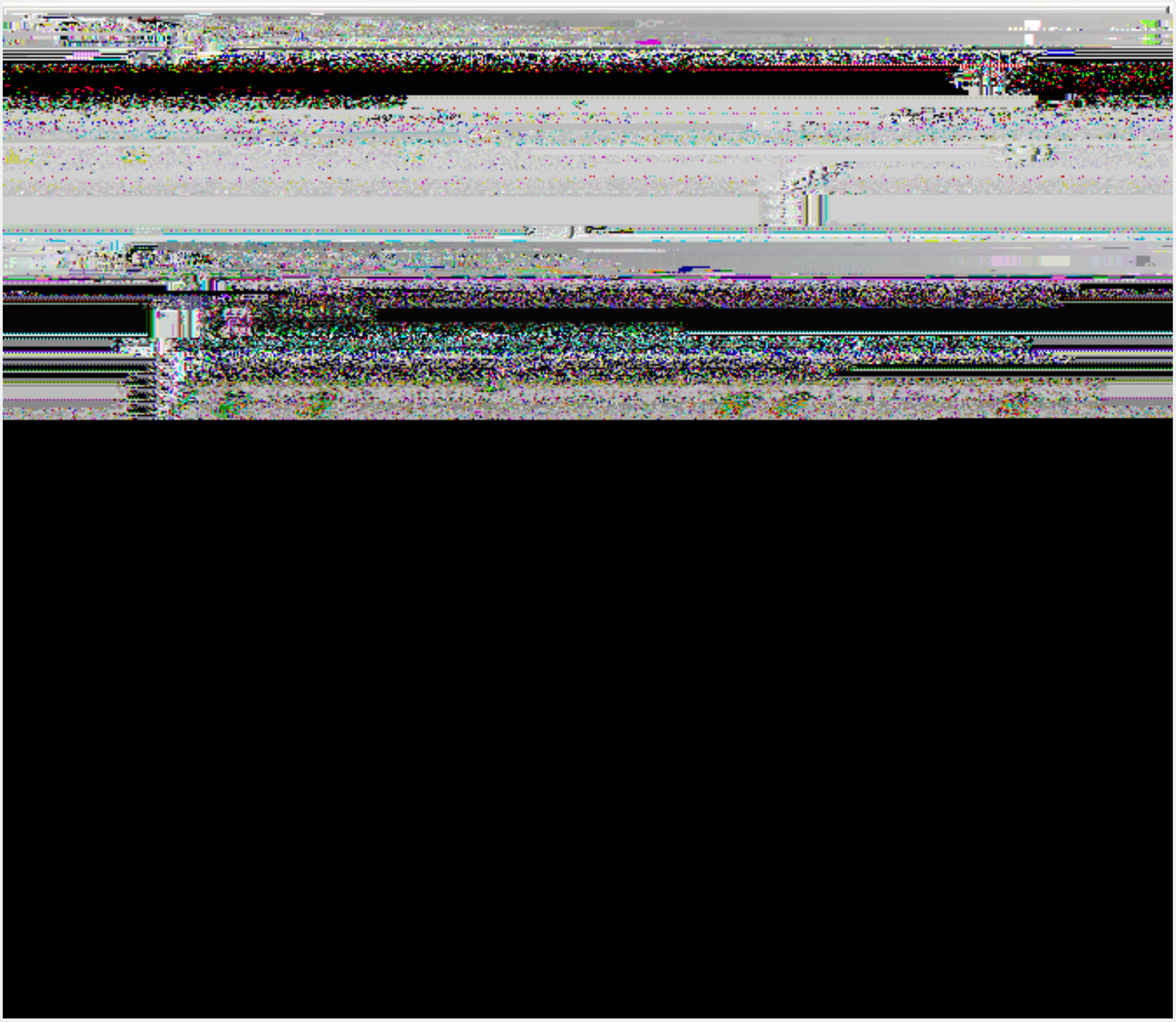
Several users can collaboratively view and manipulate a 3D scene. The image of the scene is calculated by a dynamic set of rendering engines using a raytracing algorithm, everything being controlled by a central dispatcher.

the active objects in the c3d application

1. start C3D

using the script c3d_no_user

A "Dispatcher" object is launched (ie a centralized server) as well as 4 "Renderer" objects, that are active objects to be used for parallel rendering.



the 4 renderers a 0. launched

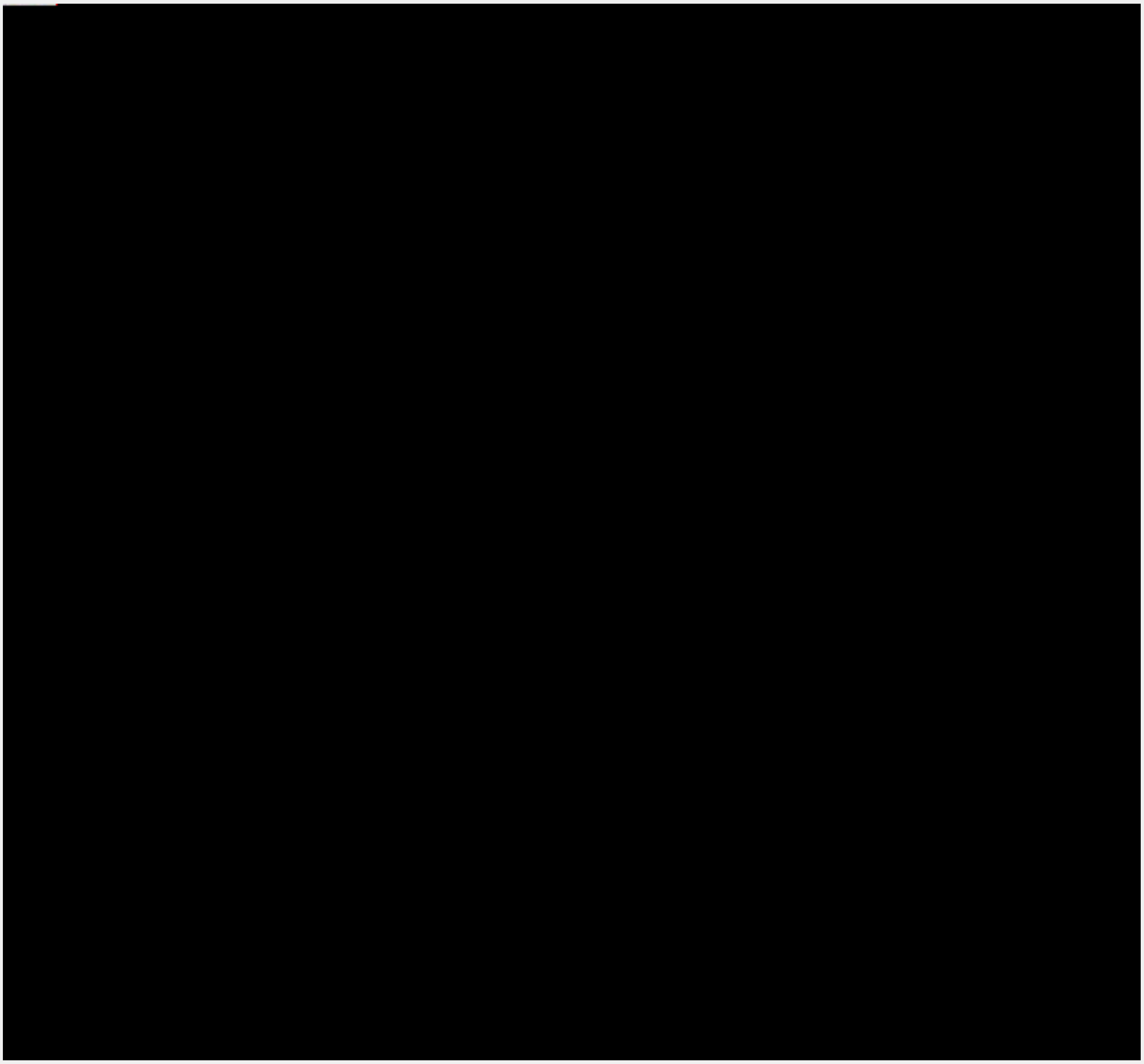
the dispatcher GUI is launched

The bottom part of the window allows the addition and removal of renderers.

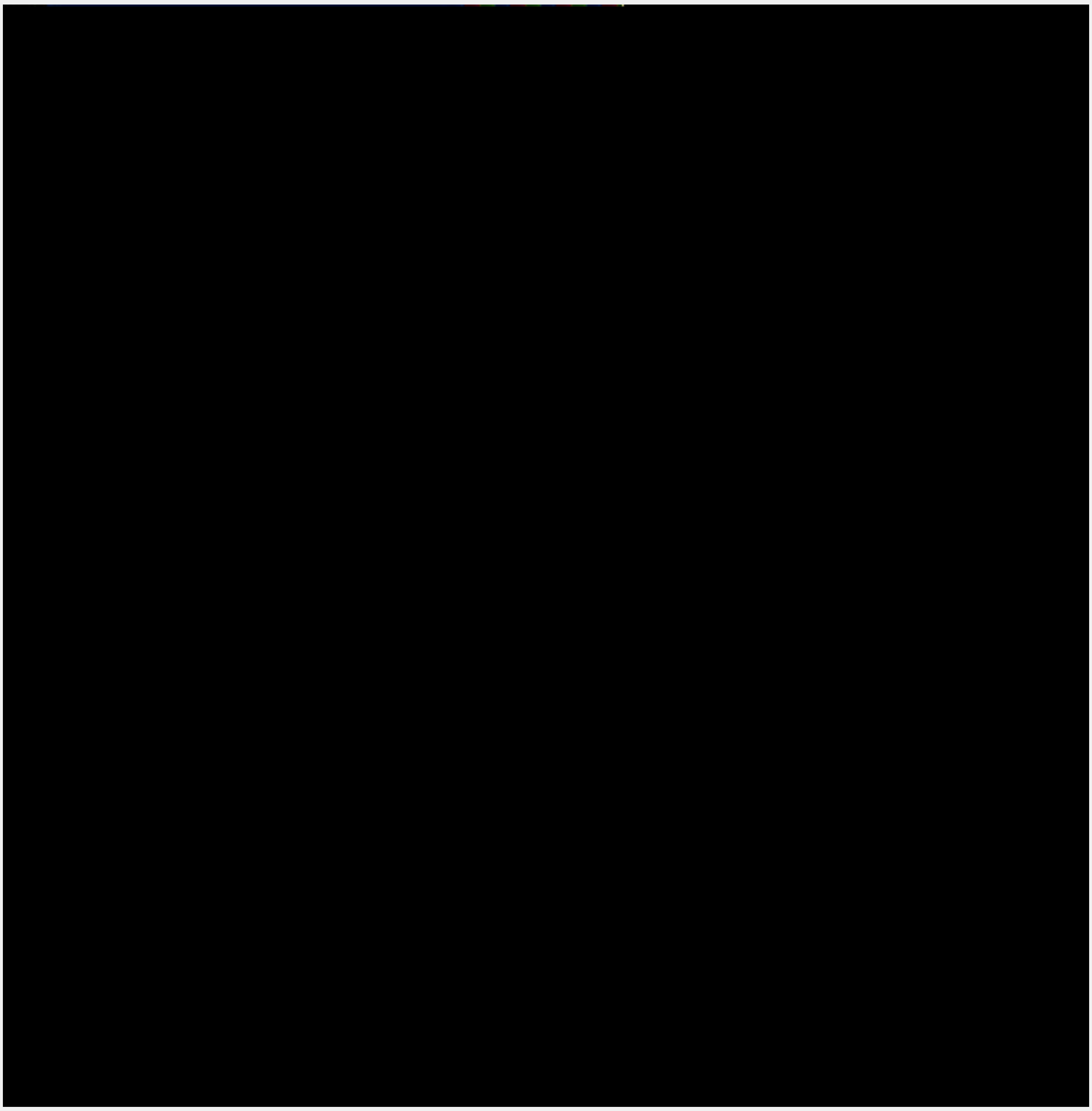
2. start a user

using c3d_add_user

− connect on the current host (proposed by default) by just giving your name



for example the user "alice"



− add random spheres for instance, and observe messages (Requests) between Active Objects.

− add and remove renderers, and check graphically whether the corresponding Active Objects are contacted or not, in order to achieve the rendering.

− you can texseeruvisxseizeve ist, s fmation byto Actatidin"− adhient timelin red oe istWorlde Obje" on e thWorld panel withve theight mouse buttonrs, as.

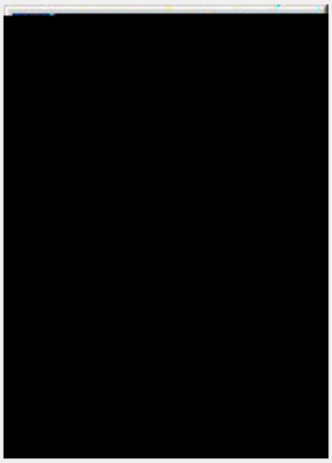
2.3. Migration of active objects

ProActive allows the transparent migration of objects between virtual machines.

A nice visual example is the penguin's one.

Mobile agents

This example shows a set of mobile agents moving around while stili0.0tbn exu hng arowithe trir basgenwhilitheeine o trr. It also featurehe tracapae aity to 0 −21



5. move the control window to another user

− start a node on a different computer, using another screen and keyboard

− monitor the corresponding JVM with IC2D

− drag−and−drop the active object "AdvancedPenguinController" with IC2D into the newly created JVM : the control window will appear on the other computer and its user can now control the penguins application.

− still with IC2D, doing a drag−and−drop back to the original JVM, you will be able to get back the window, and control yourself the application.

```
- "Generating class : ... jacobi.Stub_SubMatrix "

- "ClassServer sent class ... jacobi.Stub_SubMatrix successfully"
```

You can start IC2D (script ic2d.sh or ic2d.bat) in order to visualize the JVMs and the Active Objects. Just activate the "Monitoring a new host" in the "Monitoring" menu at the top left.

To stop the Jacobi computation and all the associated AOs, and JVMs, just ^C in the window where you started the Jacobi script.

are:

```
− sendBordersToNeighbors ()

− setNorthBorder (double[] border)

− setSouthBorder (double[] border)

− setWestBorder (double[] border)

− setEastBorder (double[] border)
```

In class SubMatrix.java, add a Method `barrier()` of the form:

```
String[] st= new String[1];

st[0]="keepOnGoing";

ProSPMD.barrier(st);
```

Do not forget to define the `keepOnGoing()` method that indeed can return void, and just be empty. Find the appropriate place to call the `barrier()` Method in the `loop()` Method.

In class Jacobi.java, just after the `compute()` Method, add an infinite loop that, upon a user's return key pressed, calls the method `keepOnGoing()` on the SPMD group "matrix". Here are samples of the code:

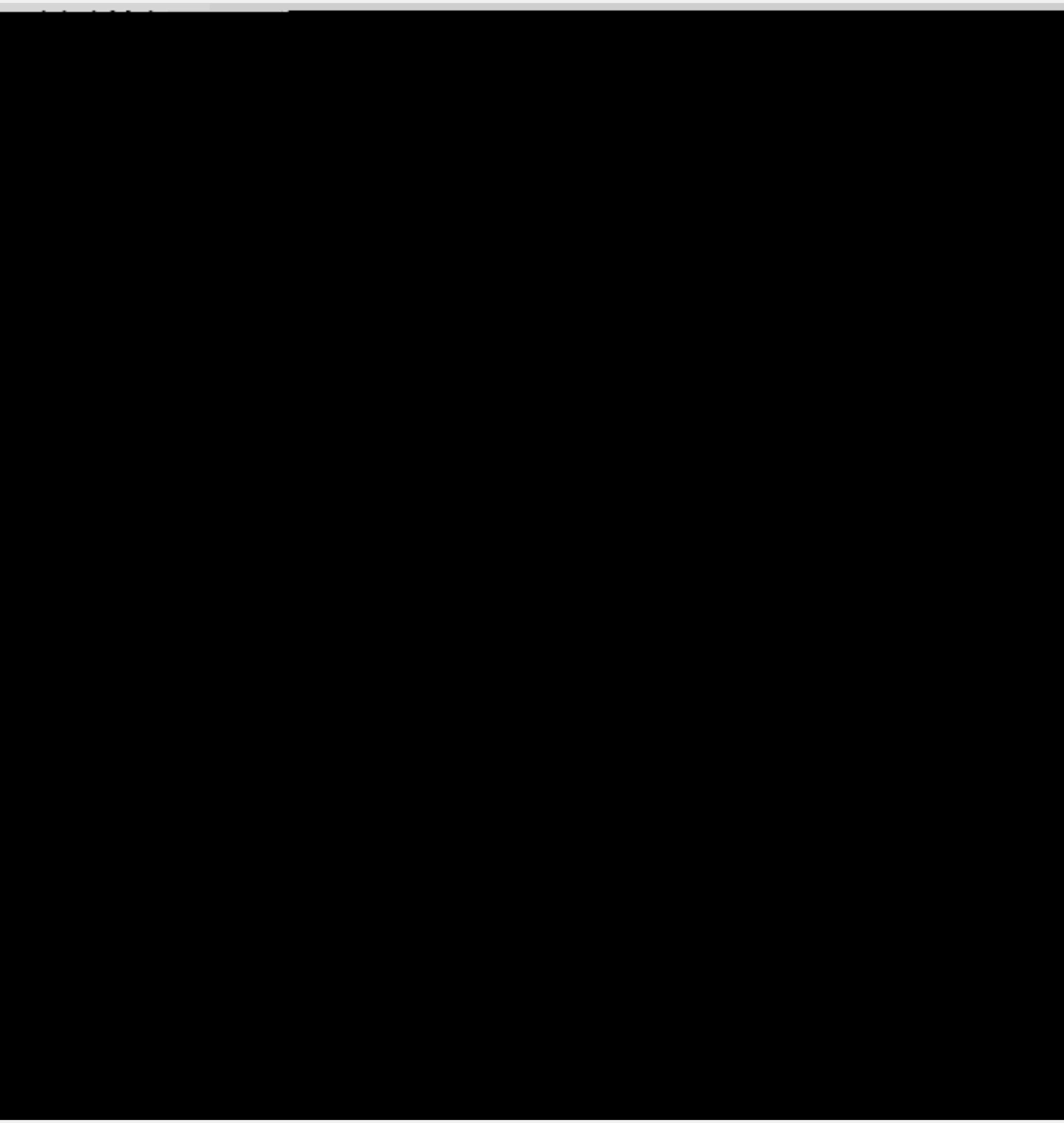
```
while (true) {

printMessageAndWait();

matrix.keepOnGoing();

}

...

private static void printMessageAndWait() {

java.io.BufferedReader d = new java.io.BufferedReader(

new java.io.InputStreamReader(System.in));

System.out.println(" --> Press return key to continue");

System.out.println(" or Ctrl c to stop.");

try {

d.readLine();

} catch (Exception e) {

}

}
```

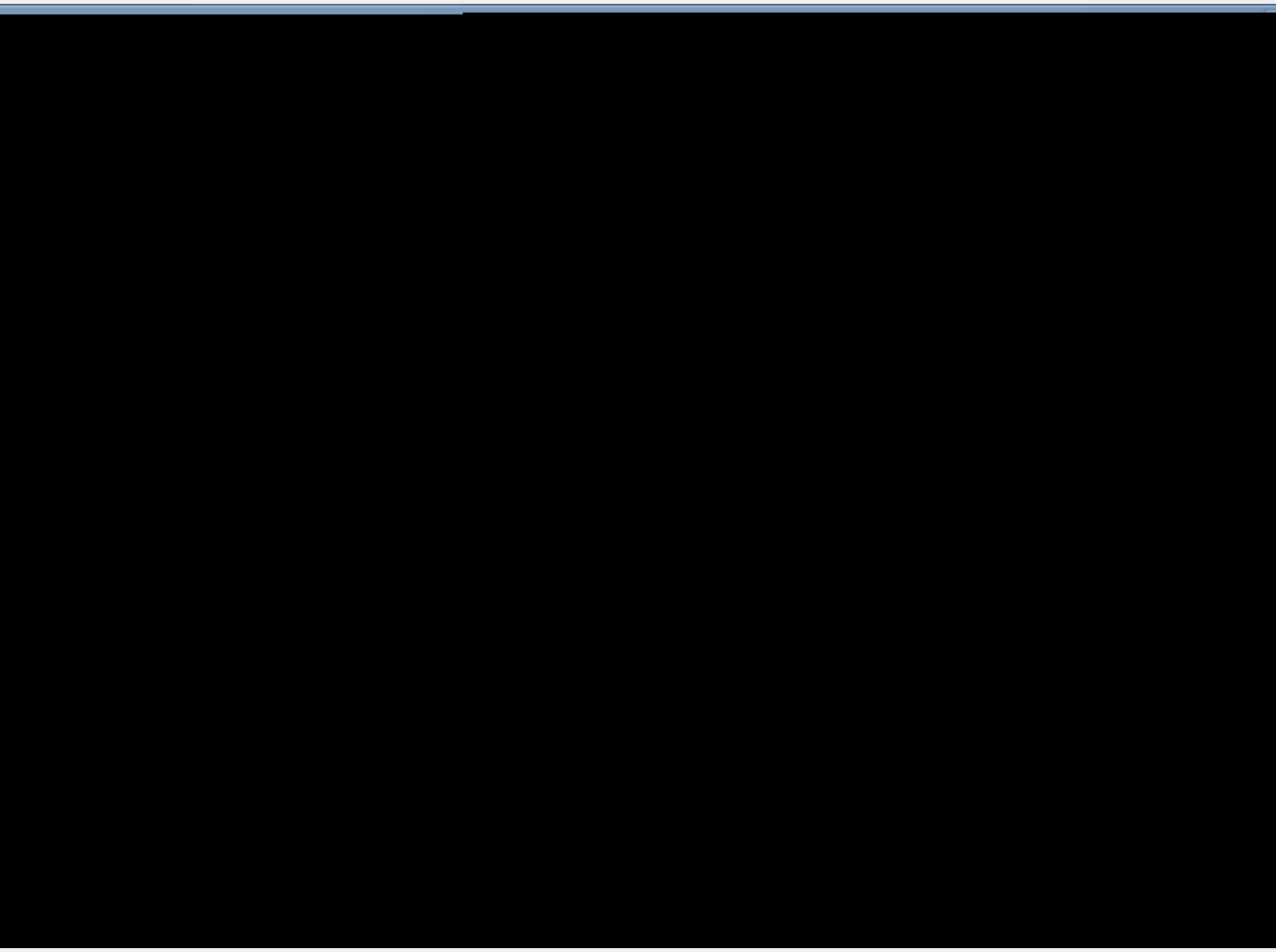
Recompile, and execute the code. Each iteration needs to be activated by hitting the return key in the shell window where Jacobi was launched. Start IC2D (./ic2d.sh or ic2d.bat), and visualize the communications as you control them. Use the "Reset Topology" button to clear communication arcs. The green and red dots indicate the pending requests.

You can imagine and test other modifications to the Jacobi code.

3.4 Undestanding various different kind of barriers

The group of neighbors built above is important wrt synchronization. Below in method "loop()", an efficient barrier is achieved only using the direct





In order to gea details and documentation on Groups and OO SPMD, have a look at:

`ProActive/src/org/objectweb/proactive/doc−files/`

`TypedGroupCommunication.html`

`OOSPMD.html`

4. Virtual Nodes and Deployment descriptors

4.1 Virtual Nodes

Gea back to the source code o4 Jacobi.java, and understand where and how the Virtual Nodes and

42 XML Ddescriptors

`ProActive descriptos/Matrix.xtml`

```
− Virtual Node Definition

− Mapping of Virtual Nodes to JVM

− JVM Definition

− Process Definition
```

A detailed presentation of XML descriptors is available at:

```
ProActive/docs/api/index.html
```

entry 9. XML Deployment Descriptors

4.3 Changing the descriptor

Edit the file Matrix.xml in order to change the number of JVMs being used. For instance, if your machine is powerful enough, start 9 JVMs, in order to have a single SubMatrix per JVM.

[redacted]

You do not need to recompile, just restart the execution. Use IC2D to visualize the differences in the configuration.

5. Execution on several machines and Clusters

5.1 Execution on several machines in the room

Associate with several persons or use several machines, and modify the file

```
ProActive/examples/descriptors/Matrix.xml
```

in order to launch the Jacobi computation on several machines. You can use IC2D to visualize the machines and the JVMs being launched on them.

5.2 Execution on Clusters

```
        System.err.println(
```

```
            aoce.printStackTrace();
            return null;
        } catch (NodeException ne) {
            System.out.println("creation of default node failed");
            ne.printStackTrace();
            return null;
        }
    }

    /** method for migrating
    * @param destination_node destination node
    */
    public void moveTo(String destination_node) {
        System.out.println("\n-----------------------------");
        System.out.println("starting migration to node : " + destination_node);
        System.out.println("...");
        try {
            // THIS MUST BE THE LAST CALL OF THE METHOD
            ProActive.migrateTo(destination_node);
        } catch (MigrationException me) {
```

```
    /** This method is called from within the constructor to
     * initialize the form.
     *        It will perform the initialization of the frame
     */
    private void initComponents() {
        jLabel1 = new javax.swing.JLabel();
        addWindowListener(new java.awt.event.WindowAdapter() {
                public void windowClosing(java.awt.event.WindowEpter evt) {
                    exitForm(evt);
                }
            });

        jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        getContentPane().add(jLabel1, java.awt.BorderLayout.CENTER);
    }

    /** Kill the frame */
    private void exitForm(java.awt.event.WindowEpter evt) {
        //        System.exit(0); would kill the VM !
        dispose(); // this way, the active object agentFrameController stays alive
    }

    /**
     * sets the text of the label inside the frame
     */
    private void setText(String text) {
        jLabel1.setText(text);
    }
}
```

```
package org.objectweb.proactive.examples.hello;



import org.objectweb.proactive.ActiveObjectCreationException;
import org.objectweb.proactive.Body;
import org.objectweb.proactive.ProActive;
import org.objectweb.proactive.core.body.migration.Migratable;
import org.objectweb.proactive.core.node.NodeException;
import org.objectweb.proactive.ext.migration.Migrat TdStrategyManager;
import org.objectweb.proactive.ext.migration.Migrat TdStrategyManagerImpl;

/**
 *
 * This class allows the "migration" of a graphical interface. A gui object is attached
 * to the active object, and the gui is removed before migration, thanks to the use
```

```
         * @param name the name of the agent
         * @return an instance of a ProActive active object of type HelloFrameController
         *
         */
public static
```