Guided tour

− recompile (using compile.seTf 9ders or compile.batTf 9ders)

− observe thatTstub classes areTf generated and recompiled

3.4 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 (ie the others alternate eating and thinking while one never eats!)

3.5 start the IC2D application

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

ProActive 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

The client − server example

This example implements a very simple client−server application. A client object display a `String` got 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 `Main`.

[redacted]

The Hello class

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

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

Here is a possible implementation for the `Hello` class:

Hello.java

```
/** This class represents an object, with its functionnal properties and
* behaviors.

*/
public class Hello {
    private String name;

    // ProActive requires a constructor with no parameter
    // in the class of the active object. That implies the presence
    // of a constructor with no parameter in the mother class

    /** constructor with no parameter, required by the ProActive model
    */
    public Hello() {
    }

    /** constructor
    * @param name the name of the agent
    *
    */
    public Hello(String name) {
        this.name = name;
    }

    /** a functionnal behavior of the object
    * @return what the agent has to say
```

ProActive PDC will provide an exception handler mechanism in order to process these exceptions in a separate place than the functional code. As class `String` is `final`, there cannot be any asynchronism here since the object returned from the call cannot be replaced by a future object (this restriction on `final` classes is imposed by ProActive's implementation).

Printing out the message

As already stated, the only modification brought to the code by ProActive PDC is located at the place where active objects are created. All the rest of the code remains the same, which fosters software reuse.

Hello World within the same VM

In order to run both the client and server in the same VM, the client creates an active object in the same VM if it doesn't find the server's URL. The code snippet which instantiates the Server in the same VM is the following:

```
if (args.length == 0) {
  // There is no url to the server, so create an active server within this VM

 myServer = (Hello)org.objectweb.proactive.ProActive.newActive(
                                                Hello.class.getName(),
                                                 new Object[]{"local"});
}
```

To launch the Client and the Server, just type:

```
> java org.objectweb.proactive.examples.hello.Client &
```

Hello World from another VM on the same host

Starting the server

Just start the `main` method in the `Hello` class.

```
> java org.objectweb.proactive.examples.hello.Hello &
```

Launching the client

```
> java org.objectweb.proactive.examples.hello.HelloClient //localhost/Hello
```

Hello World from abroad: another VM on a different host

Starting the server

Log on to the server's host, and launch the `Hello` class.

```
remoteHost> java org.objectweb.proactive.examples.hello.Hello &
```

Launching the client

Log on to the client Host, and launch the client

```
clientHost> java org.objectweb.proactive.examples.hello.HelloClient //remoteHost/Hello
```

Initialization of the activity

Active objects, as indicates their name, have an activity of their own (an internal thread).
It is possible to add pre and post processing to this activity, just by implementing the interfaces InitActive and EndActive, that define the methods initActivity and endActivity.

The following example will help you to understand how and when you can initialize and clean the activity.

When instanciated, the activityeir an object is automatically started, but it will first do what is written in the

```
        }

        /**
         * Constructor for InitializedHello.
         * @param name
         */
        public InitializedHello(String name) {
                super(name);
        }
        /**
         * @see org.objectweb.proactive.InitActive#initActivity(Body)
         * This is the place where to make initialization before the object
         * starts its activity
         */
        public void initActivity(Body body) {
                System.out.println("I am about to start my activity");
        }

        /**
         * @see org.objectweb.proactive.EndActive#endActivity(Body)
         * This is the place where to clean up or terminate things after the
         * object has finished its activity
         */
        public void endActivity(Body body) {
                System.out.println("I hasetished icemy activity");
        }


        /**
         * this method will end the activity of the active object
         */
        public void terminate() {
                // 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();
        }


}
```

Execution

Execution is similar to the previous example.
In order to see the end of the activity, you will of course need to add a call to hello.terminate() !

[redacted]

− provide a no−arg constructor

− provide an implementation fncetsing ProActive's migration mechanism.

```
                                System.ni.4println("problem while pausing :
                                                                         + 
                                ie.printStackTrace();
                        }
                }
                // possibly terminate the activity of the active object ...
                migratable_hello.terminate();
   } else {
        System.ni.4println("creation of the active object failed");
   }
}
```

4. Execution

− start several nodes using the `startnode` script.

[redacted]

```
public static HelloFrameController createHelloFrameController(String name) {
        try {
                // creates (and initialize) the active object
                HelloFrameController obj =
                        (HelloFrameController) ProActive.newActive(
                                                        HelloFrameController.class.getN
                                                        new Object[] { name });
                return obj;
        } catch (ActiveObjectCreationException aoce) {
                System.out.println("creation of the active object failed");
                aoce.printStackTrace();
                return null;
        } catch (NodeException ne) {
                System.out.println("creation of default node failed");
                ne.printStackTrace();
                return null;
        }
}
```

```
        /** 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.WindowEvent evt) {
                                exitForm(evt);
                        }
                });

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

                pack();
        }

        /** Kill the frame */
        private void exitForm(java.awt.event.WindowEvent 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);
        }
}
```

Execution

- Similarly to the simple migration example, you will start remote nodes and specify a migration path.
- don't forget to set the security permissions

you have 2ts68f 7.−29.2 selook d(the layrscreensrame)Tj mac0 −e·

[redacted]

public void writerPolicy(org.objectweb.proactive.Service service)

Look at the inner class `MyRequestFilterm`

ProActive creates a new node and instanciates the active objects of the application : DinnerLayout, Table, and Philosopher

4. test the manual mode

Click on the philosophers' heads to switch their modes

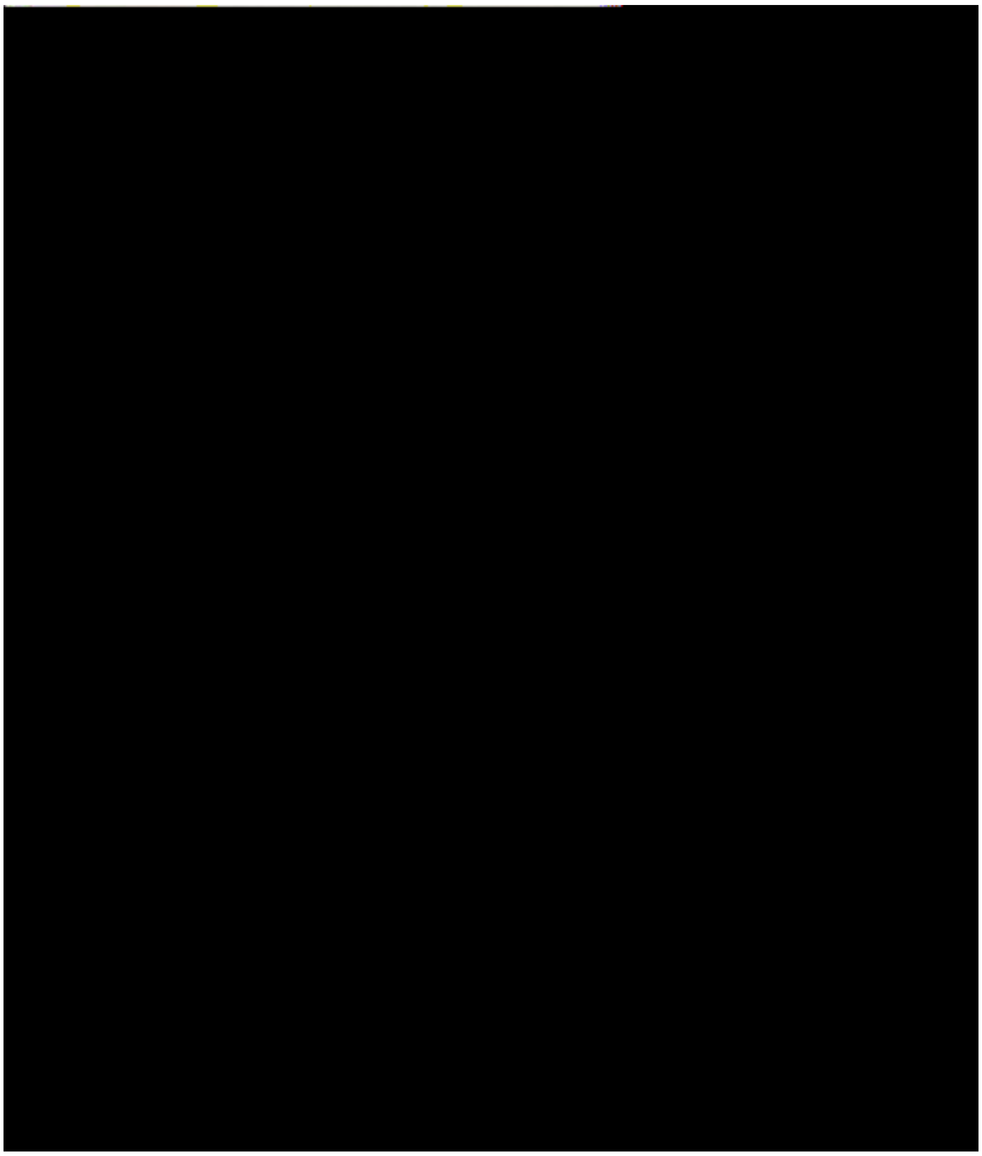
Test that there are no deadlocks!

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)



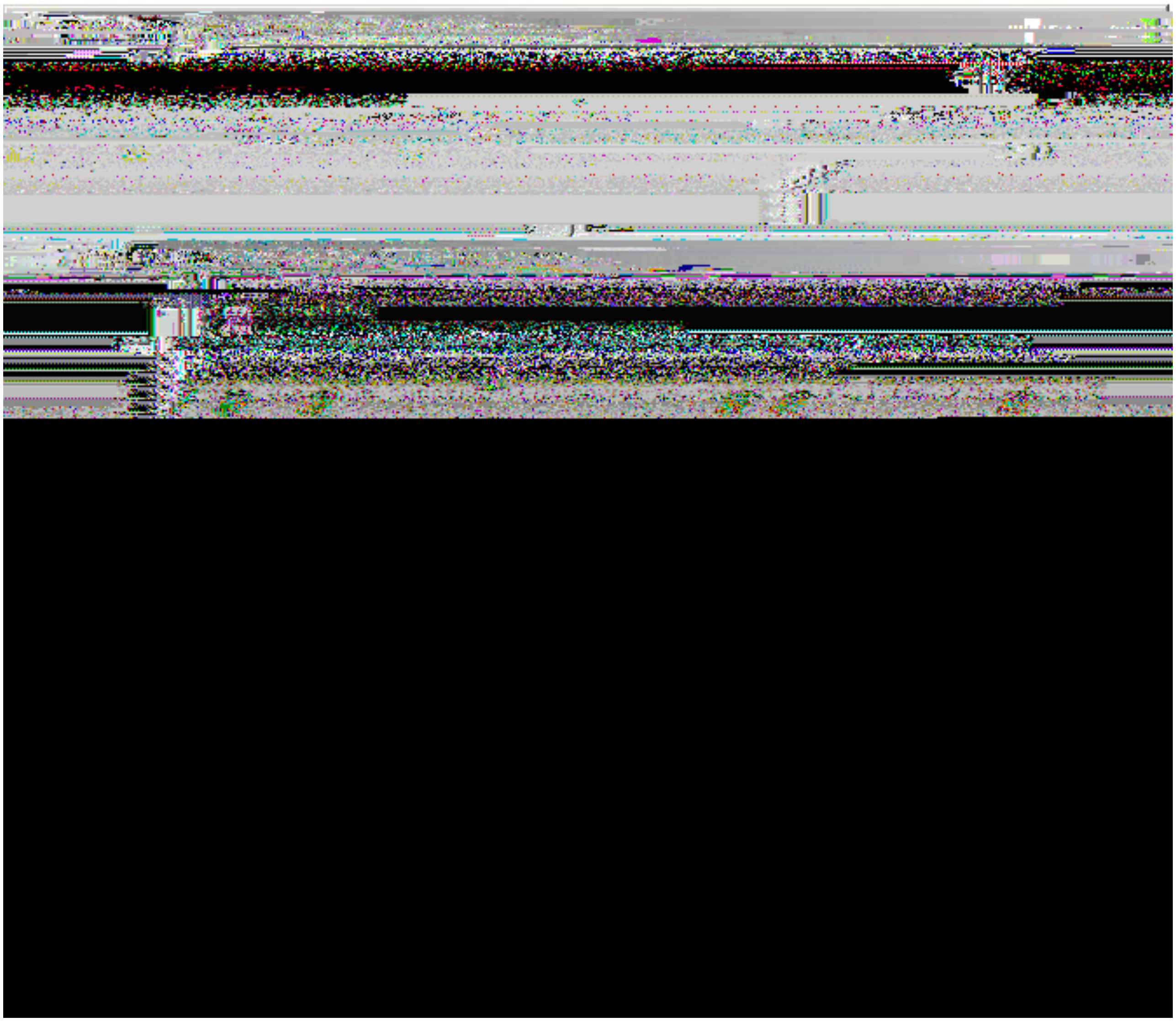
the ic2d GUI is started. It is composed of 2 panels : the main panel and the events list panel

− acquire you current machine

menu monitoring − monitor new RMI host

look at the active objects, and at the topology

objects to be used for parallel rendering.



the 4 renderers are launched

for example the user "alice"

− spin the scene, add a random sphere, and observe how the action takes place immediately

− add and remove renderers, and observe the effect on the "speed up" indication from the user window.

Which configuration is the fastest for the rendering?

Are you on a multi−processor machine?

** you might nit perceive the difference of the performance. The difference is better se 0epith more distributed nodes and objects (for example on a clusterepith 30+ renderers).*

3. start a user from another machine

using the c3d_add_user script, and specifying the host (NOT set by default)

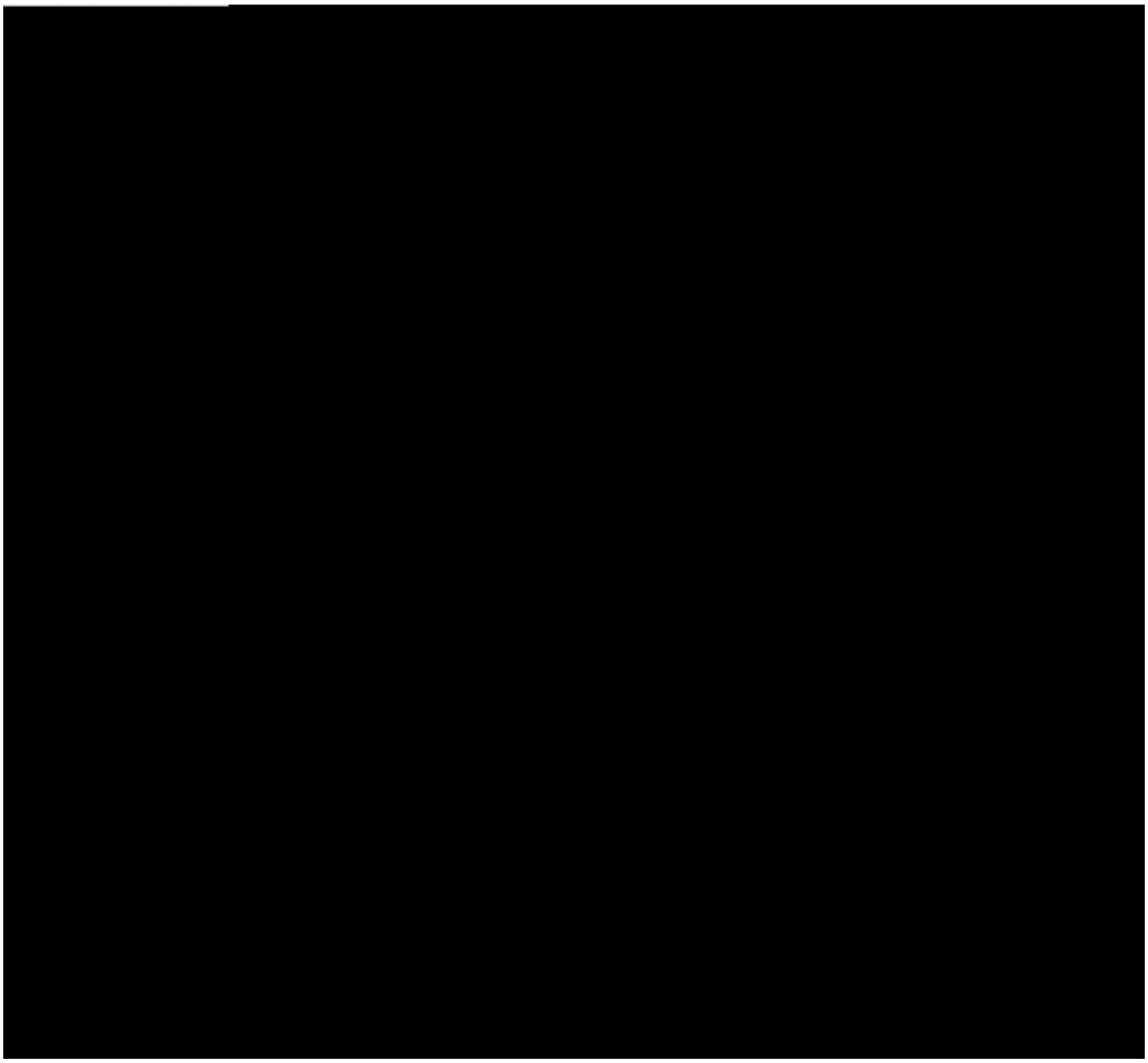
If you use rlogin, make sure the DISPLAY is properly set.

You must use the same version of ProActive on both machines!

Notice that a collaborative consensus must be reached before starting some actions (or that a timeout occured).

4. start IC2D to visualize the topology

− to visualize all Active objects, you need to acquire ("monitoring" memetnsnsnrhy −13.n4.9 cm6.4a appline



− 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.

− as migration and communications are implemented in a fully compatible manner, you can even migrate with

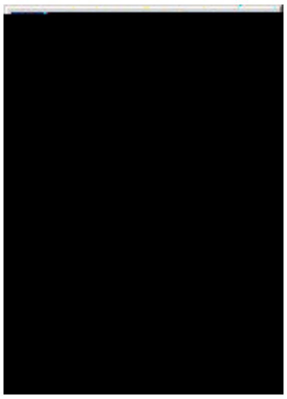
2.3. Migration of active objects

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 still communicating with thee60 rse and with each other. It also features the capability to move a swing window between s86 arof.dd8w8 ihile still communQrome p



4. add several agents

3. Conclusion

This tour was intented to guide you through an overview of ProActive.

You should now be able to start programming with ProActive, and you should also have an idea of the capabilities of the library.

We hope that you liked it and we thank you for you interest in ProActive.

e lib6a6a6ais ur sPbca6a.l abfou yobca6a6 websitean d yosuggt iPbcs ar6 welcom