version 2.0 (April 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 4unctionnalities 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.

Table of Contents

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 fro8 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
```

1.2. Initialization of the activity

Active objects, as indicates their name, have an activity of their own (an internal thread).

Execution

Execution is similar to the previous example; just use the InitializedHelloClient client class.

```
/** 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 {
```

```
        System.out.println("frame is killed");
}
```

HelloFrame

This is an example of a graphical class that could be associated with the active object. Here is the code.

Execution

- Similarly to the simple migration example (use the HelloFrameController class), you will start remote l 6856 i specify a migration path.
- don't forget to set the security permissions
- you have 2 ways for handling the display of the graphical objects :
    - ♦ look on the display screens of the machines

2.1. Synchronization with ProActive

ProActive provides an advanced synchronization mechanism that allows an easy and safe implementation of potentially complex synchronization policies.

This is illustrated by two examples :

- The readers and the writers
- cf 0dining philosophers

The readers−writers

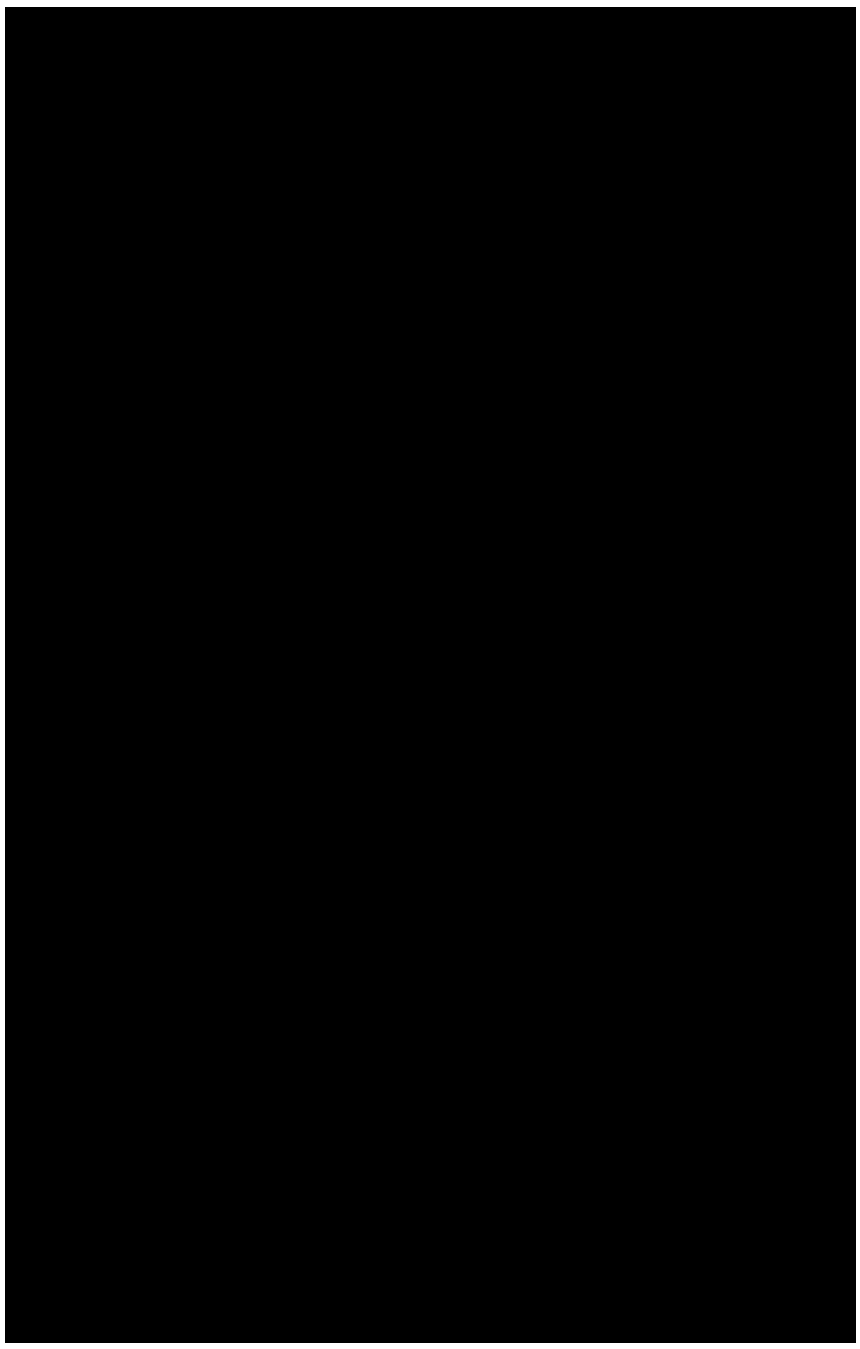
The readers and the writers want to access the sam 0data. In order to allow concurrency while ensuring the consistency of the readings, accesses to the data have to be synchronized upon a specified policy. Thanks to ProActive, the accesses are guaranteed to be allowed sequentially.

The implementation with ProActive uses 3 active objects : Reader, Writer, and the controller class (ReaderWriter).

1. start the application

using the readers script

ProActive starts a node (i.e. a JVM) on the current machine, and creates 3 Writer, 3 Reader, a ReaderWriter (the controller of the application) and a ReaderDisplay, that are active objects.



a GUI is started that illustrates the activities of the Reader and Writer objects.

2. look and check the effect of different policies : even, writer priority, reader priority

What happens when priority is set to "reader priority" ?

3. look at the code for programming such policies

in `org.objectweb.proactive.examples.readers.ReaderWriter.java`

More specifically, look at the routines in :

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

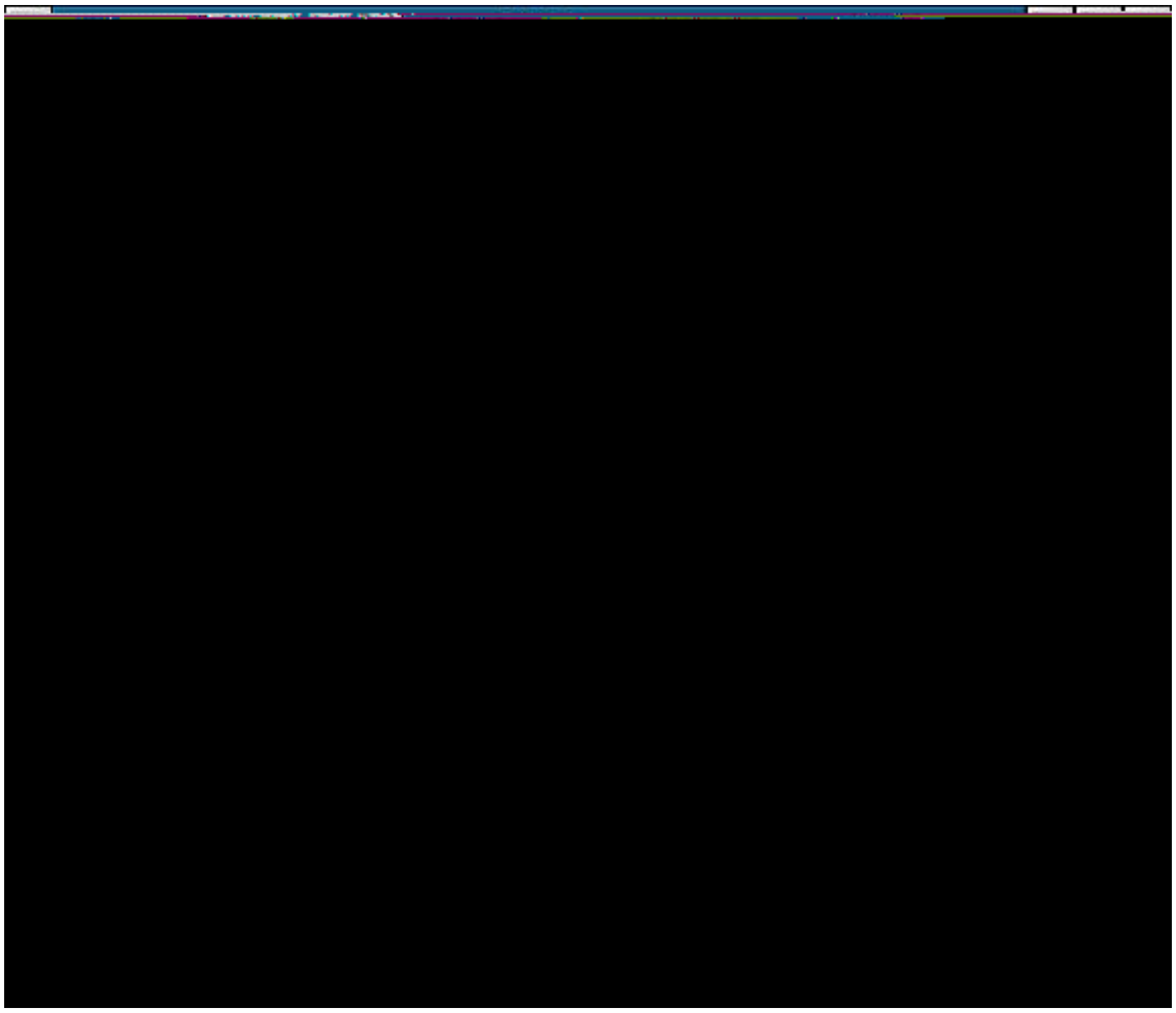
public void readerPolicy(org.objectweb.proactive.Service service)

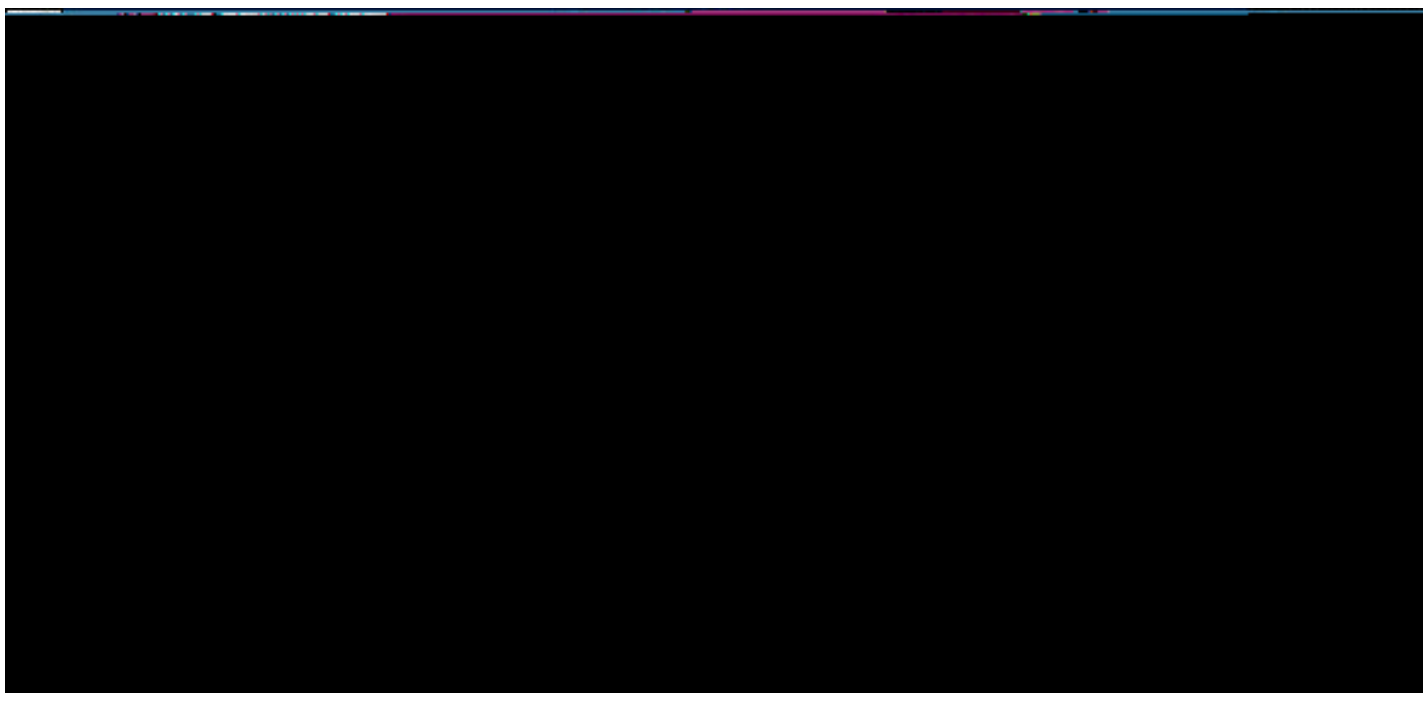
public void writerPolicy(org.objectweb.proactive.Service service)

Look at the class MyRequestFicor)Tj /F4 12 Tf (that implements)Tj /F0 12 Tf (org.objectweb.proactive.core.

The dining philosophers

The "4 .3 9 problem is a classical exercise in t Tdteach000 of concurrent programm000. 2 Tdgoal is to avoid deadlocks.g ph





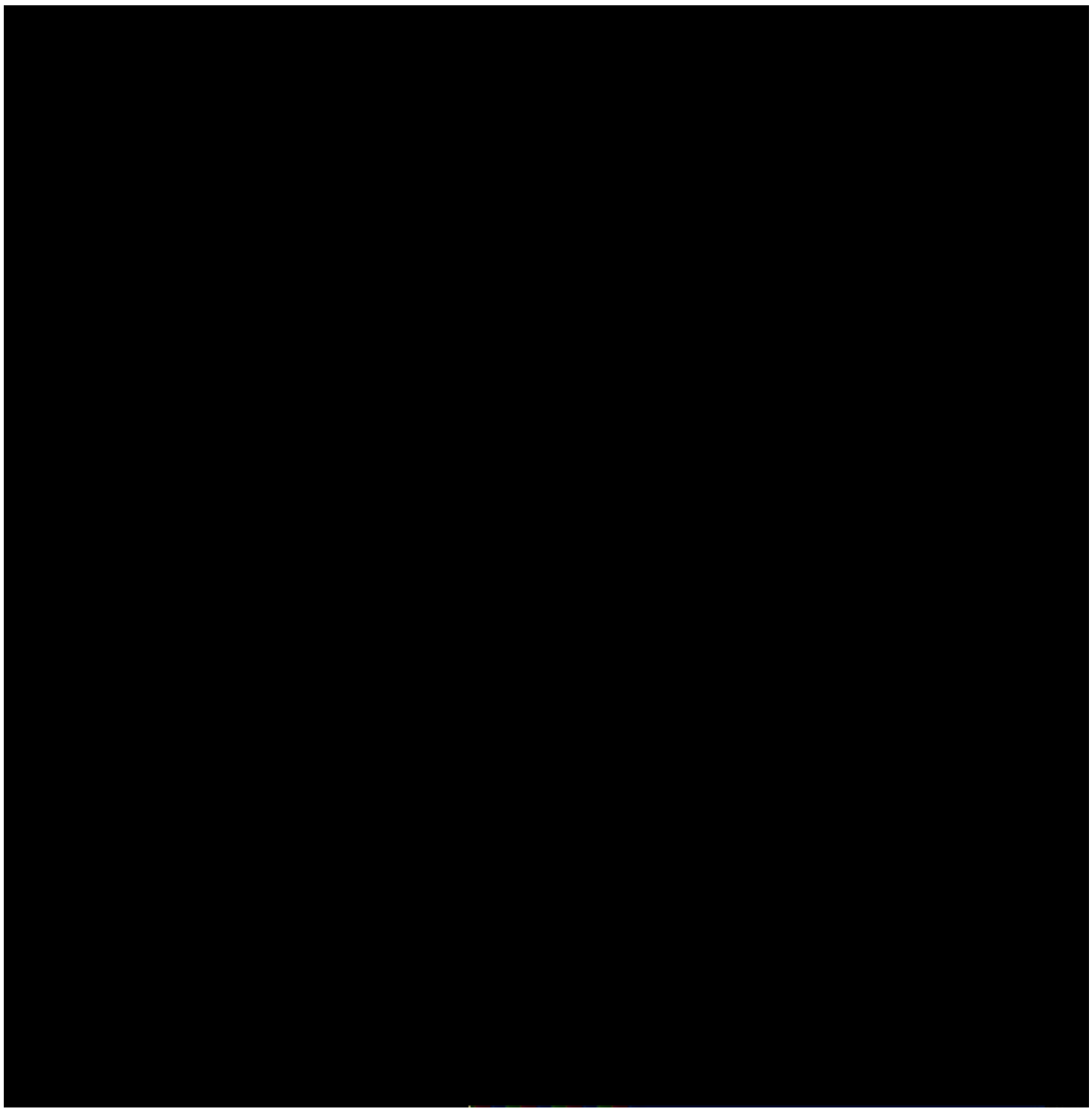
the ic2d GUI is started. It is composed of 2 panels : the main panel and the events list panel

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



```
        *
        */
public static HelloFrameController createHelloFrameController(String name) {
try {
// creates (and initialize) the active object
HelloFrameController obj =
(HelloFrameController) ProActive.newActive(HelloFrameController.class.getName(), new Object[] { name });
return obj;
} catch (ActiveObjectCreationException Ac /F3 10oudohnj /F3 10p /Ftl}
                                          }
```

```
    java.net.InetAddress localhost = java.net.InetAddress.getLocalHost();
    org.objectweb.proactive.ProActive.register(hello,
        "//" + localhost.getHostName() + "/Hello");
[ catch (Exception e) {
    System.err.println
```

```
/** method for migrating
* @param destination_node destination node    * @para/
public void moveTo(String destination_node) {    * System.out.println("\n----------------------+-destination_node);    */
```