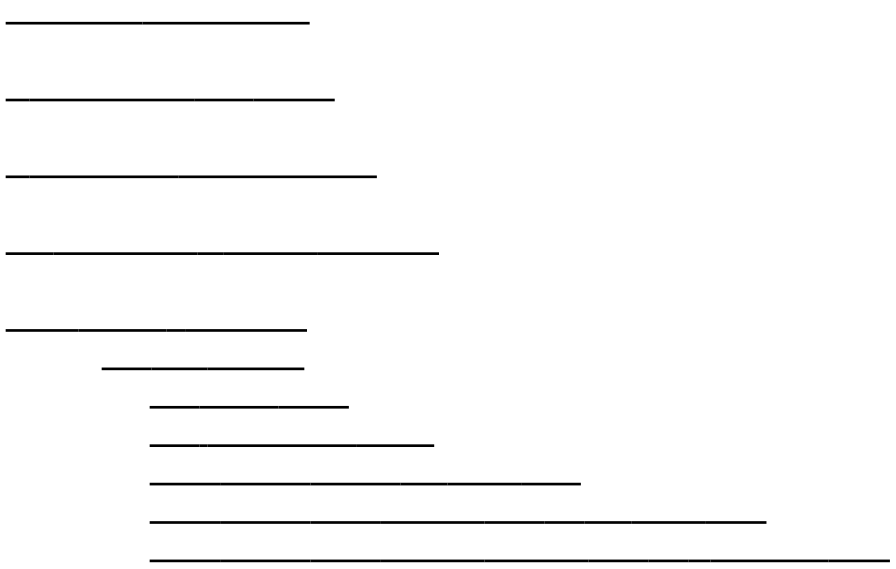


Table of Contents

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 understanding of the library, and of the c 2mpts.

Hello world ! example

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

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
```

1.2. Initialization of the activity

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

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

Hellof 28}

use the (Hellof 28Cor

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
- The dining philosophers

The readers−writers

The readers and the writers want to access the same data. 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 dining philosophers

The "dining philosophers" problem is a classical exercise in the teaching of concurrent programming. The goal is to avoid deadlocks.

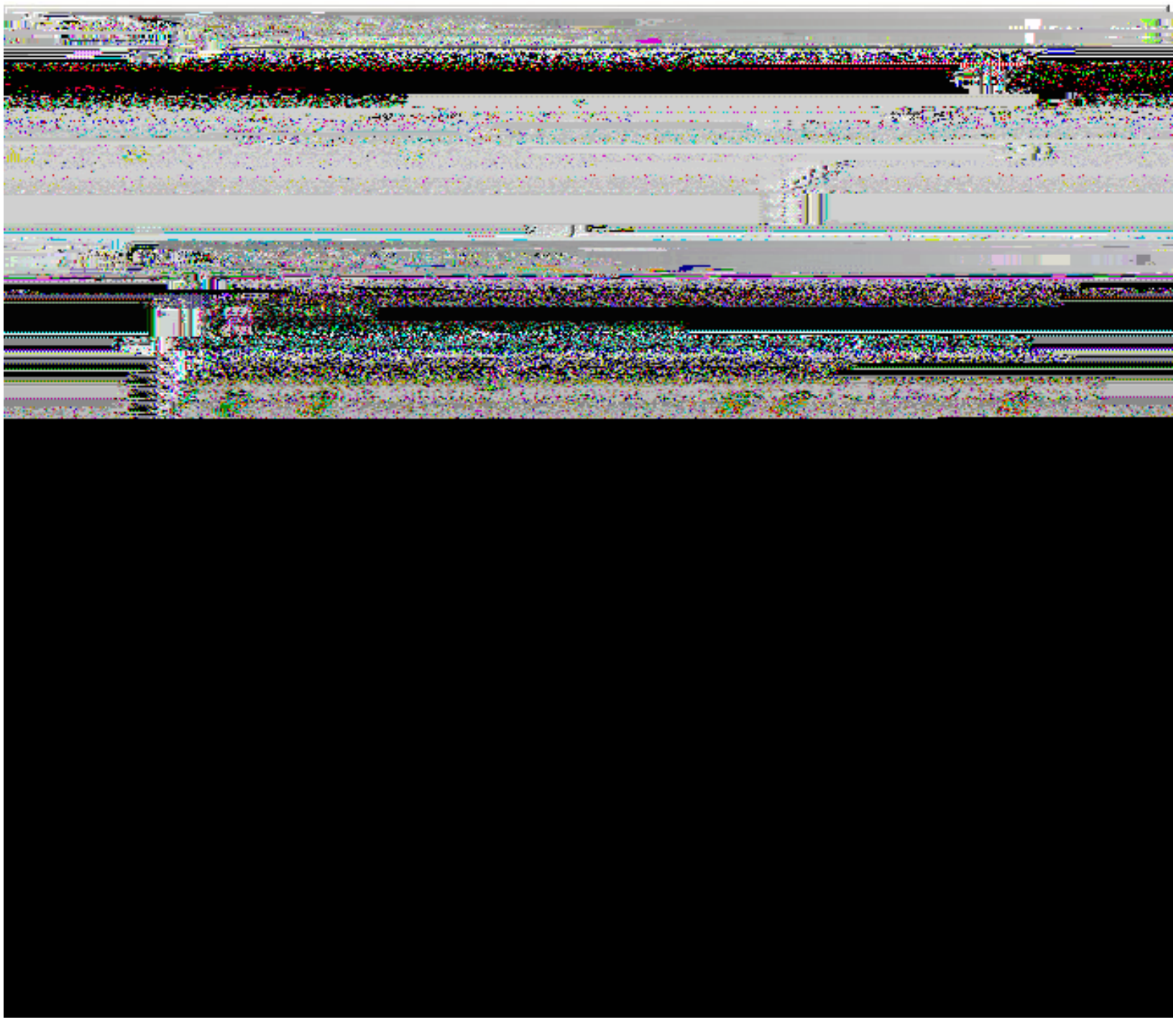
We have provided an illustration of the solution using ProActive, where all the philosophers are active objects, as well as the table (controller) and the dinner frame (user interface).

1. start the philosophers application

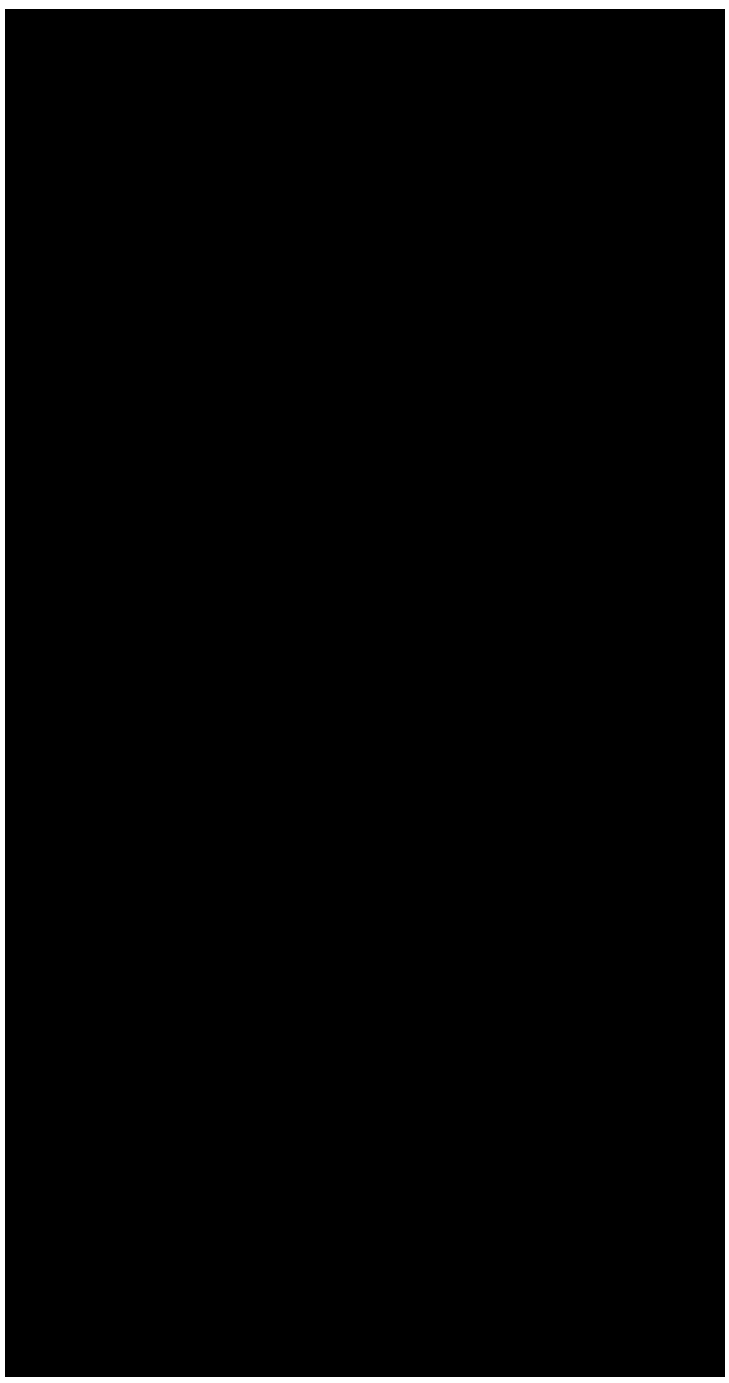
with philosophers.sh or philosophers.bat

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

[redacted]



the 4 renderers are launched

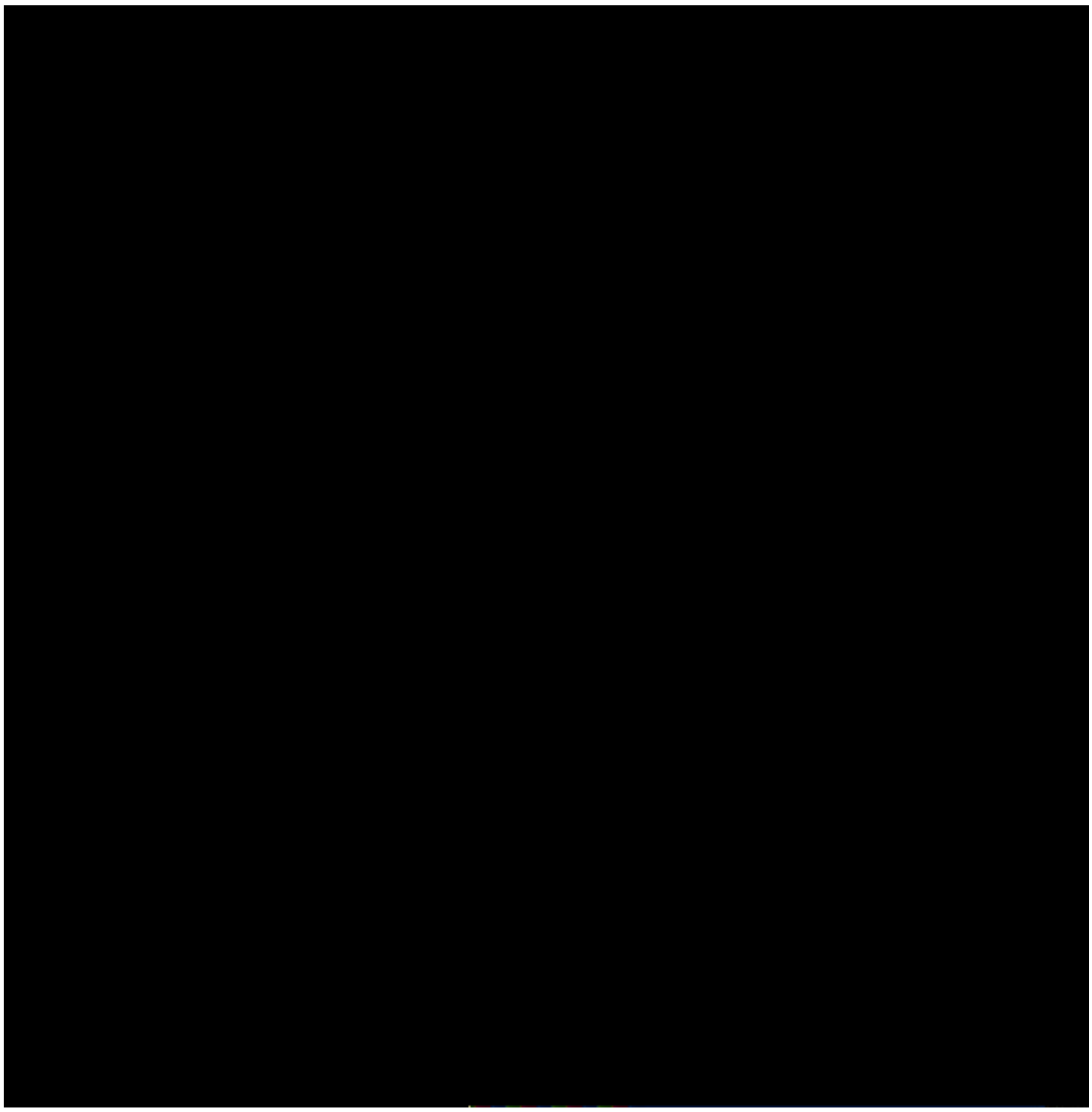


the dispatcher GUI is launched

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

2. start a user

using c3d_add_user



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 their base and with each other. It also features the capability to move a swing window between screens while moving an agent from one JVM to the other.

1. start the penguin application

using the `penguin` script.

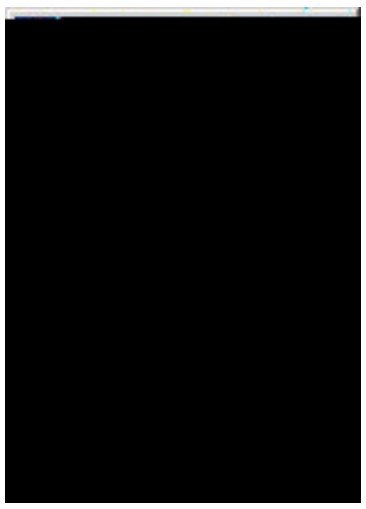
2. start IC2D to see what is going on

using the `ic2d` script

acquire the machines you have started nodes on

3. add an agent

− on the Advanced Penguin Controller window : button "add agent"



an agent is materialized by a picture in a java window.

− select it, and press button "start"

− observe that the active object is moving between the machines, and that the penguin window disappears and reappears on the screen associated with the new JVM.

4. add several agents

after selecting them, use the buttons to :

− communicate with them ("chained calls")

− start, stop, resume them

− trigger a communication between them ("call another agent")

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.

```
        *
        */
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
* @paraT 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
```