

ProActive *Parallel Suite*



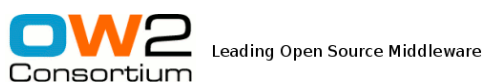
An Open Source Solution for Enterprise Grids & Clouds

ProActive Programming

Get Started

Version 2014-02-17

ActiveEon Company, in collaboration with INRIA



ProActive Programming v2014-02-17 Documentation

Legal Notice

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation; version 3 of the License.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

If needed, contact us to obtain a release under GPL Version 2 or 3, or a different license than the GPL.

Contact: proactive@ow2.org or contact@activeeon.com

Copyright 1997-2012 INRIA/University of Nice-Sophia Antipolis/ActiveEon.

Mailing List

proactive@ow2.org

Mailing List Archive

<http://www.objectweb.org/wws/arc/proactive>

Bug-Tracking System

<http://bugs.activeeon.com/browse/PROACTIVE>

Contributors and Contact Information

Team Leader

Denis Caromel
INRIA 2004, Route des Lucioles, BP 93
06902 Sophia Antipolis Cedex
France
phone: +33 492 387 631
fax: +33 492 387 971
e-mail: Denis.Caromel@inria.fr

Contributors from OASIS Team

Brian Amedro
Francoise Baude
Francesco Bongiovanni
Florin-Alexandru Bratu
Viet Dung Doan
Yu Feng
Imen Filali
Fabrice Fontenoy
Ludovic Henrio
Fabrice Huet
Elaine Isnard
Vasile Jureschi
Muhammad Khan
Virginie Legrand Contes
Eric Madelaine
Elton Mathias
Paul Naoumenko
Laurent Pellegrino
Guilherme Peretti-Pezzi
Franca Perrina
Marcela Rivera
Christian Ruz
Bastien Sauvan
Oleg Smirnov
Marc Valdener
Fabien Viale

Contributors from ActiveEon Company

Vladimir Bodnartchouk
Arnaud Contes
Cédric Dalmasso
Christian Delbé
Arnaud Gastinel
Jean-Michel Guillaume
Olivier Helin
Clément Mathieu
Maxime Menant
Emil Salageanu
Jean-Luc Scheefer
Mathieu Schnoor

Former Important Contributors

Laurent Baduel (Group Communications)
Vincent Cave (Legacy Wrapping)
Alexandre di Costanzo (P2P, B&B)
Abhijeet Gaikwad (Option Pricing)
Mario Leyton (Skeleton)
Matthieu Morel (Initial Component Work)
Romain Quilici
Germain Sigety (Scheduling)
Julien Vayssiere (MOP, Active Objects)

Table of Contents

List of figures	v
List of tables	vi
Preface	vii

Part I. Introduction

Chapter 1. ProActive Overview	2
1.1. ProActive Programming Frameworks	2
1.1.1. Introduction	2
1.1.2. Active Objects Model	3
1.1.3. Implementation Language	4
1.1.4. Communication through typed asynchronous messages	4
1.1.5. Programming With Components: GCM	5
1.1.6. Library features	5
Chapter 2. ProActive Installation	7
2.1. ProActive Installation Overview	7
2.1.1. Testing ProActive With Examples	7
2.1.2. Developing With ProActive	7
2.2. ProActive Detailed Installation Steps	7
2.2.1. Download And Expand The ProActive Archive	7
2.2.2. Set the CLASSPATH for using ProActive to write applications	8
2.2.3. Create a proactive.java.policy File To Set Permissions	9
2.2.4. A simple proactive-log4j file	10
2.2.5. Run A Few Examples For Testing	12
Chapter 3. ProActive Troubleshooting	13
3.1. Common problems	13
3.2. Enabling the loggers	14
3.3. Domain name resolution problems	14
3.4. RMI Tunneling	14

Part II. Guided Tour and Tutorial

Chapter 4. Introduction	17
4.1. Overview	17
4.2. Installation and setup	17
4.3. ProActive and IDEs (Eclipse, ...)	17
Chapter 5. ProActive Example Applications	21
5.1. C3D: A distributed 3D renderer	21
5.1.1. How to use C3D	22
5.2. Readers/Writers Application	25

5.2.1. How to use the Readers/Writers	26
5.3. The dining philosophers	27
5.3.1. How to use the philosophers application	28
5.4. The migratory penguins	30
5.4.1. How to use the penguin application	30
5.4.2. How to use the Penguin Controller	32
5.5. Chat example	32
5.5.1. How to run the Chat application	32
5.5.2. Chat migration	34
5.6. Integral Pi	34
5.6.1. Introduction	34
5.6.2. Initialization	35
5.6.3. Communication primitives	36
5.6.4. Running the PI example	37
5.7. The nbody example	38
5.7.1. How to run the n-body example	39
5.7.2. Barnes-Hut	40
5.8. Conclusion	42
Chapter 6. Active Object Tutorial	43
6.1. Simple Computation And Monitoring Agent	43
6.1.1. Classes Used	44
6.1.2. CMA Architecture and Skeleton Code	44
6.1.3. Proposed Work	46
6.1.4. Solutions and Full Code	46
6.2. Active Objects Lifecycle: Using InitActive, RunActive and EndActive	47
6.2.1. Classes Used	48
6.2.2. Initialized CMA Architecture and Skeleton Code	48
6.2.3. Proposed Work	49
6.2.4. Solution and Full Code	50
6.3. Remote Monitoring Agent	51
6.3.1. Classes Used	51
6.3.2. Deployed CMA Architecture and Skeleton Code	52
6.3.3. Proposed Work	53
6.3.4. Solution and Full Code	53
6.4. Agent synchronization	56
6.4.1. Classes Used	56
6.4.2. Architecture and Skeleton Code	57
6.4.3. Proposed Work	59
6.4.4. Solution And Full Code	59
6.5. Monitoring Several Computers Using Migration	61
6.5.1. Classes Used	61
6.5.2. Design of Migratable Monitoring Agent and Skeleton Code	62
6.5.3. Proposed Work	64
6.5.4. Solution and Full Code	65
6.6. Groups of Monitoring Agents	67
6.6.1. Classes Used	67
6.6.2. Architecture and Skeleton Code	68
6.6.3. Proposed Work	70
6.6.4. Solution And Full Code	71
6.7. Monitoring Agent As A Web Service	73
6.7.1. Classes Used	73
6.7.2. Architecture and Skeleton Code	73

6.7.3. Proposed Work	76
6.7.4. Solution And Code	76
6.8. Primality test tutorial	76
6.8.1. A Sequential Version of the Primality Test	77
6.8.2. A Distributed Version of the Primality Test	78
6.8.3. A Distributed Version of the Primality Test Using the Master-Worker API	89
Bibliography	96
Index	98

List of Figures

1.1. ProActive Features	2
1.2. MOP architecture	5
5.1. Active objects in the c3d application	22
5.2. Started GUI illustrating the activities of Reader and Writer objects.	27
5.3. The Dining Philosophers Example	29
6.1. Distribution of the Test Range (Example with 2 Workers)	79
6.2. Killing a JVM from IC2D	84

List of Tables

2.1. ProActive archive contents	8
---------------------------------------	---

Preface

In order to make the ProActive Programming documentation easier to read, it has been split into four manuals:

- **ProActive Get Started Manual** - This manual contains an overview of ProActive Programming showing different examples and explaining how to install the middleware. It also includes a tutorial teaching how to use it. This manual should be the first one to be read for beginners.
- **ProActive Reference Manual** - This manual is the main manual where the concepts of ProActive are described. Information on configuration and deployment are described in that manual. It also includes guides for high-level APIs usage such as Master-Worker, SMPD APIs.
- **ProActive Advanced Features Manual** - This manual describes some advanced features like Fault-Tolerance, ProActive Compile-Time Annotations or Web Services Exportation. In Addition, it gives information on some other high-level APIs such as Monte-Carlo or Branch and Bound APIs. Finally, it helps advanced user to extend ProActive.
- **ProActive Components Manual** - ProActive defines a component model called ProActive/GCM suitable to support the development of efficient grid applications. This manual therefore contains all necessary information to be able to understand and use this component model. This model is really linked to ProActive so a ProActive/GCM user may have to refer to the ProActive manuals form time to time.

These manuals should be read in the order defined above. However, this is not essential as these documentations are linked together. Besides, if some links seems to be dead in one of these manuals, make sure that all of them had been built previously (multiple html version). So as to build all the manuals at once, go to your ProActive `compile/` directory and type `build[.bat] doc.ProActive.manualHtml`. This builds all manuals in all formats. You may also need the javadoc documentation since these manuals sometime refer to it. To build all the javadoc documentations, type `build[.bat] doc.ProActive.javadoc.complete doc.ProActive.javadoc.published` inside the `compile/` directory. If you just want to build one of these manuals in a specific format, type `build[.bat]` to see all the possible targets and chose the one you are interested in.

Part I. Introduction

Table of Contents

Chapter 1. ProActive Overview	2
1.1. ProActive Programming Frameworks	2
1.1.1. Introduction	2
1.1.2. Active Objects Model	3
1.1.3. Implementation Language	4
1.1.4. Communication through typed asynchronous messages	4
1.1.5. Programming With Components: GCM	5
1.1.6. Library features	5
Chapter 2. ProActive Installation	7
2.1. ProActive Installation Overview	7
2.1.1. Testing ProActive With Examples	7
2.1.2. Developing With ProActive	7
2.2. ProActive Detailed Installation Steps	7
2.2.1. Download And Expand The ProActive Archive	7
2.2.2. Set the CLASSPATH for using ProActive to write applications	8
2.2.3. Create a proactive.java.policy File To Set Permissions	9
2.2.4. A simple proactive-log4j file	10
2.2.5. Run A Few Examples For Testing	12
Chapter 3. ProActive Troubleshooting	13
3.1. Common problems	13
3.2. Enabling the loggers	14
3.3. Domain name resolution problems	14
3.4. RMI Tunneling	14

Chapter 1. ProActive Overview

Grid computing is now a key aspect, from scientific to business applications, from large scale simulations to everyday life enterprise IT. After the old days of mainframes and servers with hundreds of persons sharing the same machines or the quite current days of the PCs with one person for one computer, we are just entering the era of Ubiquitous Computing with many computers at hand for every single individual.

Grids gather large amounts of heterogeneous resources across geographically distributed sites so as to be used by virtual organizations. Resources are usually organized in groups of desktop machines and/or clusters, which are managed by different administrative domains (labs, universities, companies, etc.). The large amounts of heterogeneous resources complicate the deployment task in terms of configuration and connection to remote resources. Targeted deployment sites may be specified beforehand, or automatically discovered at runtime. Each domain may have its own management and security policies. Grid applications aim at using a large number of resources, which are geographically distributed. Hence, Grid frameworks have to help applications with scalability issues, such as providing parallelism capabilities for a large number of resources.

The large number of resources distributed on different domains implies a high probability of faults, such as hardware failures, network downtime, or maintenance. Since a grid gathers geographically dispersed and administratively independent computational sites into a large federated computing system with common interfaces, new programming models have to be defined to abstract away these complexities from programmers.

1.1. ProActive Programming Frameworks

1.1.1. Introduction

ProActive is an open source Java library aiming to simplify the programming of multithreaded, parallel, and distributed applications for Grids, multi-cores, clusters, and data-centers. It allows concurrent and parallel programming and offers distributed and asynchronous communications, mobility, and a deployment framework. With a small set of primitives, ProActive provides an API allowing the development of parallel applications, which may be deployed on distributed systems and on Grids. ProActive does not require any modifications to Java or to the Java Virtual Machine (JVM), therefore allowing the deployment of applications using the ProActive API on any operating system that provides a compatible JVM.

Overall, ProActive promotes a few basic and simple principles:

- Activities are distributed, remotely accessible objects
- Interactions are done through asynchronous method calls
- Results of interactions are called futures and are first class entities.
- Callers can wait for results using a mechanism called wait-by-necessity

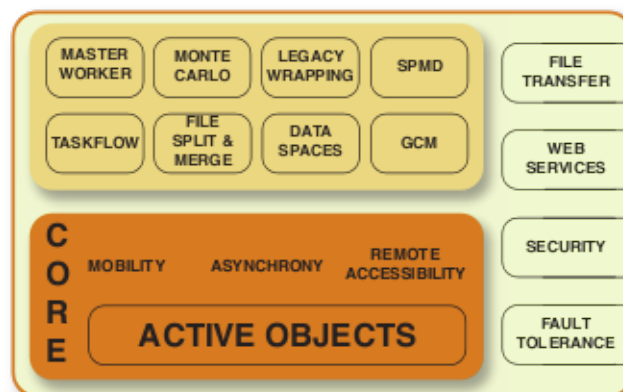


Figure 1.1. ProActive Features

Figure 1.1, “ProActive Features” exposes the main ProActive features:

- **Master-Worker** - The main goal of the ProActive Master-Worker API is to provide an easy to use framework for parallelizing embarrassingly parallel applications. You can find its documentation in "[Chapter 6. Master-Worker API](#)"¹ of the Reference Manual.
- **Monte Carlo** - The goal of the ProActive Monte-Carlo API is to provide an easy to use API for running Monte-Carlo simulations in a distributed environment. You can find its documentation in "[Chapter 14. Monte-Carlo API](#)"² of the Advanced Features manual.
- **Legacy Wrapping** - The Message Passing Interface (MPI) is a widely adopted communication library for parallel and distributed computing. This API helps you to deploy, wrap and couple MPI applications for distributed environments. You can find its documentation in "[Chapter 9. Wrapping Native MPI Application](#)"³ of the Reference Manual.
- **SPMD** - SPMD stands for Single Program Multiple Data, which is a technique used to parallelize applications by separating tasks and running them simultaneously on different machines or processors. ProActive allows the use of object oriented programming combined with the SPMD techniques. You can find its documentation in "[Chapter 8. OOSPM](#)"⁴ of the Reference Manual.
- **Taskflow** - ProActive allows to programmatically create a taskflow using the TaskFlow API for submitted it to the ProActive Scheduler. You can find some information on this API in "Chapter 2. User guide" of the Scheduler Manual.
- **File Split & Merge** - The "Files Split Merge" generic framework provides an easy way to develop a distribution layer for a native application in order to run it on a ProActive managed infrastructure through the ProActive Scheduler and Resource Manager. You can find its documentation in "Chapter 9. ProActive Scheduler's Files Split-Merge Extension" of the Scheduler Manual.
- **Data Spaces** - In the ProActive library, data can be accessed through Data Spaces API. You can find the Data Spaces documentation in "[Chapter 10. Accessing data with Data Spaces API](#)"⁵ of the Reference Manual.
- **GCM** - ProActive provides an extensible, dynamical and hierarchical component model, Grid Component Model (GCM) based on [Fractal](#)⁶. The GCM was defined by the [CoreGRID NoE project](#)⁷ and is available [here](#)⁸. You can find the GCM Documentation in the [Components Manual](#)⁹.
- **Web Services** - ProActive allows to expose an active object as a web service. You can find the documentation of this features in "[Chapter 4. Exporting active objects as Web Services](#)"¹⁰ of the Advanced Features manual.
- **Security** - ProActive security mechanism provides a set of security features from basic ones like communications authentication, integrity, and confidentiality to more high-level features including secure object migration, hierarchical security policies, and dynamically negotiated policies. You can find Security documentation in "[Chapter 2. Security Framework](#)"¹¹ of the Advanced Features manual.
- **Fault Tolerance** - ProActive can provide fault-tolerance capabilities through two different protocols: a Communication-Induced Checkpointing protocol (CIC) or a pessimistic message logging protocol (PML). You can find information on ProActive Fault Tolerance in "[Chapter 1. Fault-Tolerance](#)"¹² of the Advanced Features manual.

1.1.2. Active Objects Model

ProActive is based on the concept of active object, which is an entity with its own configurable activity. A distributed or concurrent application using ProActive is composed of a number of entities called active objects. Each active object has one distinguished element,

¹ file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/././ReferenceManual/multiple_html/MasterWorker.html
² file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/././AdvancedFeatures/multiple_html/MonteCarlo.html
³ file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/././ReferenceManual/multiple_html/WrappingMpiAndLegacyCode.html
⁴ file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/././ReferenceManual/multiple_html/OOSPM.html
⁵ file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/././ReferenceManual/multiple_html/DataSpaces.html
⁶ <http://fractal.objectweb.org>
⁷ <http://www.coregrid.net/>
⁸ <http://www.coregrid.net/mambo/images/stories/Deliverables/d.pm.04.pdf>
⁹ file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/././Components/multiple_html/index.html
¹⁰ file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/././AdvancedFeatures/multiple_html/WebServices.html
¹¹ file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/././AdvancedFeatures/multiple_html/Security.html
¹² file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/././AdvancedFeatures/multiple_html/FaultTolerance.html

the root, which is the only entry point to the active object. Each active object has its own thread of control and is granted the ability to decide in which order to serve the incoming method calls that are automatically stored in a queue of pending requests. Method calls sent to active objects are asynchronous with transparent future objects and synchronization is handled by a mechanism known as wait-by-necessity. There is a short rendezvous at the beginning of each asynchronous remote call, which blocks the caller until the call has reached the context of the callee. More detailed information on active objects will be presented in "ProActiveBasis".

The ProActive solution relies on asynchronous method calls mastering both complexity and efficiency. It also proposes advanced features such as groups, mobility, and components. In the framework of formal calculus, Asynchronous Sequential Processes (ASP), confluence, and determinism have been proven for this programming model: [CH05](#) and [CHS04](#).

1.1.3. Implementation Language

Grids gather large amounts of heterogeneous resources, different processor architectures and operating systems. In this context, using a language which relies on a virtual machine allows maximum portability. ProActive is developed in Java, a cross-platform language and the compiled application may run on any operating system providing a compatible virtual machine. Moreover, ProActive only relies on standard APIs and does not use any operating-system specific routines, other than to run daemons or to interact with legacy applications. There are no modifications to the JVM nor to the semantics of the Java language, and the bytecode of the application classes is never modified.

1.1.4. Communication through typed asynchronous messages

In ProActive, the distribution is transparent: invoking methods on remote objects does not require the developer to design remote objects with an explicit remote mechanism. Therefore, the developer can focus on the business logic since the distribution is automatically handled and transparent. Moreover, the ProActive library preserves polymorphism on remote objects (through the reference stub, which is a subclass of the remote root object).

Communication between active objects is realized through method invocations, which are reified and passed as messages. These messages are serializable Java objects which may be compared to TCP packets. Indeed, one part of the message contains routing information towards the different elements of the library, and the other part contains the data to be communicated to the called object. Although all communications proceed through method invocations, the communication semantics depends upon the signature of the method, and the resulting communication may not always be asynchronous. Three cases are possible: synchronous invocation, one-way asynchronous invocation, and asynchronous invocation with future result.

- Synchronous invocation:
 - The method returns a non reifiable object (primitive type or final class):

```
boolean foo()
```

- The method declares a thrown exception:

```
public void bar() throws AnException
```

In this case, the caller thread is blocked until the reified invocation is effectively processed and the eventual result (or exception) is returned. It is fundamental to keep this case in mind, because some APIs define methods which throw exceptions or return non-reifiable results.

- One-way asynchronous invocation:
 - The method does not throw any exception and does not return any result:

```
public void gee()
```

The invocation is asynchronous and the process flow of the caller continues once the reified invocation has been received by the active object (in other words, once the rendezvous is finished).

- Asynchronous invocation with future result:
 - the return type is a reifiable type, and the method does not throw any exception:

```
public MyReifiableType baz()
```

In this case, a future object is returned and the caller continues its execution flow. The active object will process the reified invocation according to its serving policy, and the future object will then be updated with the value of the result of the method execution.

If an invocation from an object A on an active object B triggers another invocation on another active object C, the future result received by A may be updated with another future object. In that case, when the result is available from C, the future of B is automatically updated, and the future object in A is also update with this result value, through a mechanism called automatic continuation.

1.1.5. Programming With Components: GCM

ProActive programming also features a general purpose component framework allowing software architects to design their parallel applications using a hierarchical component framework: the Grid Component Model (GCM).

The GCM is a high level programming framework which relies on Active Object and therefore inherits usual ProActive properties like asynchronous call, automatic continuation, deployment interoperability...

The GCM extends the Fractal component model to cover distribution and parallel application's needs. Using the GCM, you can easily express at design time the dependencies between software blocks composing your application. In addition, GCM components provide non-functional features allowing developers to benefits extra capabilities such as monitoring, autonomic management...

The GCM provides software architects with a comprehensive framework to express at design time the parallelism and the distribution of an application. Thus, the architecture of the system itself captures the parallel/distributed aspects, acting as a powerful specification and documentation. Furthermore, developers do not need to spend extensive time to learn distribute programming or implement collective communication, but rather concentrate on the business code and leverage the GCM framework.

1.1.6. Library features

ProActive uses the "MOP" to provide flexibility and configurability: it allows the addition of meta-objects for managing new required features. Moreover, the library also proposes a deployment framework, which allows the deployment of active objects on various infrastructures. The MOP architecture is represented in the next figure.

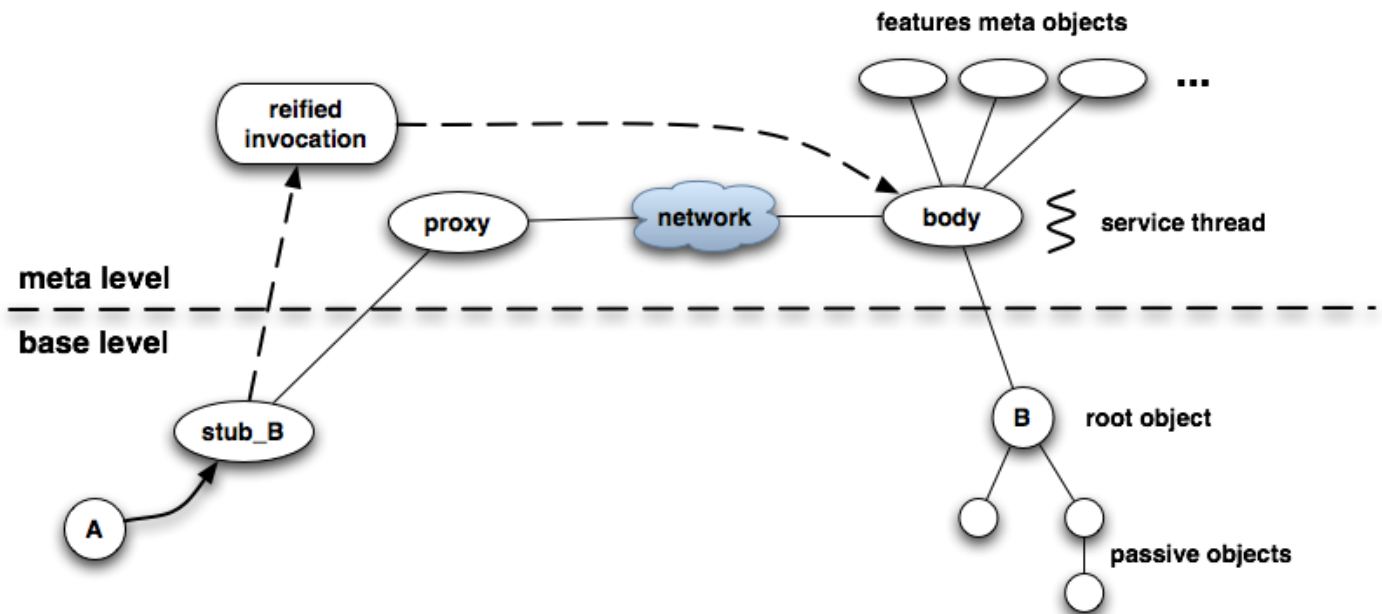


Figure 1.2. MOP architecture

The library may be represented in four layers: IDE, programming model, deployment facilities, and non-functional features.

ProActive offers IC2D, an interactive graphical environment for remote monitoring and control of distributed applications. Due to the fact that it uses the standard (starting with the J2SE platform 5.0) Java Management Extensions (JMX) technology in order to monitor distribution, IC2D can be deployed without requiring changes to the JVM or Java.

The programming model consists of the active objects model which offers asynchronous communication, typed group communication, and the object-oriented SPMD programming paradigm. In addition to the standard object oriented programming paradigm, ProActive also proposes a component-based programming paradigm, by providing an implementation of the Fractal component model geared at Grid computing.

The deployment layer includes a deployment framework (detailed in "ProActiveDeployment") and it allows the creation of remote active objects on various infrastructures. A scheduler is also proposed to manage the deployment of jobs. The load-balancing framework uses the migration capabilities to optimize the placement of the distributed active objects.

Non-functional features include a transparent fault-tolerance mechanism based on a communication-induced check-pointing protocol, a security framework for communications between remote active objects, migration capabilities for the mobility of active objects, a mechanism for the management of exceptions, a complete distributed garbage collector for active objects, and a mechanism for wrapping legacy code, notably as a way to control and interact with MPI applications.

In the communication layer, several protocols are provided for the communication between active objects: Java RMI as the default protocol, SSH, HTTP, tunneled RMI. It is also possible to export active objects as Web Services, which can then be accessed using the standard SOAP protocol. A file transfer mechanism is also implemented. It allows the transfer of files between active objects, for instance to send large data input files or to retrieve results files.

Chapter 2. ProActive Installation

ProActive is available for [download](#)¹ under the [AGPL 3](#)² license. ProActive requires [JDK 1.5](#)³ or later to be installed on your computer. Please note that ProActive will NOT run with any version prior to 1.5 since some features introduced in JDK 1.5 are essential.

2.1. ProActive Installation Overview

2.1.1. Testing ProActive With Examples

- Download and expand the ProActive archive
- Set the `JAVA_HOME` variable to the Java distribution you want to use
- If you have already a `PROACTIVE_HOME` and/or a `PROACTIVE` variable set, then unset them or set them to your new ProActive home. If you do not do that, scripts will try to use them.
- Launch the scripts located in the `ProActive/examples` directory.

2.1.2. Developing With ProActive

- Download and unzip the ProActive archive
- Normally, sources have already been compiled. If not, please go to the `compile` directory and type `build[.bat] deploy.all` (build script has to be executable. If not, type `chmod 744 build`). To see all the possible targets, just type `build[.bat]`.
- Include in your `CLASSPATH` the ProActive jar file `ProActive/dist/lib/ProActive.jar`. Normally, only this JAR file is needed. However, it defined some required other JAR files as indexes such as `ProActive/dist/lib/javassist.jar`, `ProActive/dist/lib/log4j.jar`, `ProActive/dist/lib/xercesImpl.jar`, `ProActive/dist/lib/fractal.jar`, `ProActive/dist/lib/asm-2.2.3.jar`, `ProActive/dist/lib/ibis-1.4.jar`, `ProActive/dist/lib/bouncycastle.jar`, `ProActive/dist/lib/servlet-api.jar`, `ProActive/dist/lib/jetty-*.jar` and `ProActive/dist/lib/virtualization-core.jar`. Thus, these files have to be placed into the same directory as `ProActive.jar`. It is a piece of information to keep in mind in particular if you want to move `ProActive.jar` toward your own lib directory. In this case, you have also to move the 7 others JAR files.
- Depending on your project needs, you might need to include other libraries located in the `ProActive/dist/lib` directory.
- Launch the JVM with a [security policy file](#)⁴ using the option `-Djava.security.policy=pathToFile`. A basic policy file can be found at `ProActive/examples/proactive.java.policy`. You can also specify a [log4j configuration file](#)⁵ with the property `-Dlog4j.configuration=file:pathToFile`. If a configuration file is not specified a default logger that logs on the console will be created.

The next sections details these different steps to install ProActive.

2.2. ProActive Detailed Installation Steps

2.2.1. Download And Expand The ProActive Archive

You can download the archive file (a zip or a tar.gz file) containing ProActive from the [download section](#)⁶ of the ProActive home page. You will be asked to accept the license agreement and provide a few details including your email address. You will be then redirected towards the download page.

Unzip this archive using your favorite program for uncompressing, such as [7Zip](#)⁷ under Windows or the [unzip](#)⁸ and [tar](#)⁹ command-line utilities under most Linux/Unix systems. Uncompressing archive creates a ProActive directory and all the files contained in this

¹ <http://www.activeeon.com/community-downloads>

² <http://www.gnu.org/licenses/agpl-3.0.txt>

³ <http://java.sun.com/j2se/1.5/>

⁴ <http://java.sun.com/j2se/1.5/docs/guide/security/permissions.html>

⁵ <http://logging.apache.org/log4j/1.2/manual.html>

⁶ <http://www.activeeon.com/community-downloads>

⁷ <http://www.7-zip.org/>

archive go into this directory and its subdirectories. As already said, ProActive should be compiled. If it is not the case, you have to compile it using the `build[.bat] deploy.all` in the compile directory.

Here is a quick overview of the directory structure of the archive:

Directory or File	Description
dist/lib/ProActive.jar	ProActive bytecode that you need to include in your CLASSPATH in order to use ProActive
dist/lib/ProActive_examples.jar	Bytecode and resources of all examples included with ProActive. This jar file needs to be included in your CLASSPATH only when trying to run examples. All examples rely on ProActive and therefore the ProActive.jar file must be included in the CLASSPATH as well. This is done automatically by the scripts (<code>env.sh</code> on Linux/Unix and <code>init.bat</code> on Windows) driving the examples. The source code is also included in the <code>src</code> directory (see below)
lib	External libraries used by ProActive
doc/built/ProActive	ProActive documentation including a HTML version, a PDF version, the API JavaDoc... If this directory does not exist, please go to the ProActive/compile directory and run <code>build doc.ProActive.docs</code> . This command will create all the ProActive documentation versions. If you just want a specific version of documentation, run <code>build</code> to see all the target possibilities and choose the most appropriate one.
examples	Unix sh scripts and Windows bat files for running examples
src	The full source code of ProActive
compile	Scripts to compile ProActive using Ant.

Table 2.1. ProActive archive contents

2.2.2. Set the CLASSPATH for using ProActive to write applications

In order to use ProActive to write your own application, you need to place the `dist/lib/ProActive.jar` file in your CLASSPATH. As said previously, it makes references to other JAR files which have to be in the same directory as ProActive.jar. If it not the case, you have to include the following JAR files to your CLASSPATH:

- `dist/lib/javassist.jar` - used to handle bytecode manipulation.
- `dist/lib/log4j.jar` - [logging](http://logging.apache.org/log4j/1.2/manual.html)¹⁰ mechanism used by ProActive.
- `dist/lib/xercesImpl.jar` - used to parse and validate XML files, like deployment descriptors, configuration files and component files (see "XML_Descriptors", "Configuration" and "intro").
- `dist/lib/fractal.jar` - component model used for ProActive Components (see "configuration_html_Controllers_and_interceptors_n").
- `dist/lib/bouncycastle.jar` - used by the ProActive security framework (see "Security").
- `dist/lib/ibis-1.4.jar` - communication library used by the ProActive components.
- `dist/lib/asm-2.2.3.jar` - used by the ProActive components for bytecode manipulations.
- `dist/lib/servlet-api.jar` and `dist/lib/jetty-*.jar` - used for dynamic class loading through the embedded Jetty server.
- `dist/lib/virtualization-core.jar` - used by ProActive deployments on virtual infrastructures.



Note

You do not need to modify your CLASSPATH permanently if you include the entries above using a Java IDE or a shell script.

⁸ <http://www.info-zip.org/pub/infozip/>

⁹ <http://unixhelp.ed.ac.uk/CGI/man-cgi?tar>

¹⁰ <http://logging.apache.org/log4j/1.2/manual.html>

Under Linux/Unix:

Open a Linux/Unix terminal and go to your ProActive directory using the `cd` command. Then, set the `CLASSPATH` as follows:

```
cd [Your ProActive install directory]
PA_PATH=$(pwd)
export CLASSPATH=.:${PA_PATH}/dist/lib/ProActive_examples.jar:${PA_PATH}/dist/lib/ProActive.jar
```

Under Windows:

Open a Windows command terminal by executing either `cmd` or `command` in **Start-> Run**

Go to the ProActive installation folder by using `cd` and set the `CLASSPATH` by executing the following commands:

```
cd [Your ProActive install directory]
set CLASSPATH=.;%CD%\dist\lib\ProActive_examples.jar;%CD%\dist\lib\ProActive.jar;
```



Note

Keep in mind that some JAR files are needed by ProActive.jar and so, they have to be in the same directory. If not, add them to the `CLASSPATH`.

In addition to the jar files above, you may want to add the following jar files. None of them are used directly by the core functionalities of ProActive but only in part of the library. They are not needed at runtime if those specific functionalities are not used but they are needed to compile all the code.

- `trilead-ssh2.jar` - used when tunneling with `rmi`.
- `cryptix.jar`, `cog-jglobus-1.2.jar`, `puretls.jar`, `cryptix32.jar`, `cryptix-asn1.jar`, `cog-ogce.jar`, `cog-jglobus.jar` - used to interface with Globus
- `fractal.jar`, `fractal-adl.jar` - used by the Fractal components
- `Proactive/lib/cxf/*` - used by the Web Services features
- `bcel-5.1.jar`, `ibis-util-1.0.jar`, `colobus-0.1.jar`, `ibis-connect-1.0.jar`, `ibis-1.4.jar`, `bcel-5.1-fixes-1.0.jar` - used by Ibis if configured as communication protocol
- `ajo.jar`, `njs_client.jar`, `scriptPlugin.jar`, `jh.jar` - used when deploying to a UNICORE site.

The jar files can be found under `Proactive/lib` and under `ProaActive/dist/lib`.

2.2.3. Create a `proactive.java.policy` File To Set Permissions

See [Permissions in the Java™ 2 SDK](http://java.sun.com/j2se/1.5/docs/guide/security/permissions.html)¹¹ to learn more about Java permissions. The option `-Djava.security.policy=pathToFile` will specify which security policy file to use within ProActive. As a first approximation, you can create a simple security policy file granting all permissions:

```
grant {
    permission java.security.AllPermission;

    // Reflect access private Members
    // Used by:
    //     -Unicore Process
    permission java.lang.RuntimePermission "accessDeclaredMembers";
    permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
};
```

¹¹ <http://java.sun.com/j2se/1.5/docs/guide/security/permissions.html>

**Note**

If you use the scripts provided with the distribution to run examples, an existing policy file named `proactive.java.policy` will be used by default. It is exactly the same as described above.

2.2.4. A simple proactive-log4j file

See for logging using the [Apache log4j](http://logging.apache.org/log4j/1.2/index.html)¹². The option `-Dlog4j.configuration=file:pathToFile` will specify which logging policy file to use within ProActive. As a first approximation, you can create a simple logging policy file:

```
# This file represents the log4j config file for non regression tests.
# It is given as property (in the proactive.xml) when starting nonregression tests
```

```
# The default logging level is INFO
# The root logger logs in the test.log file
```

```
log4j.rootLogger=INFO,CONSOLE
```

```
# If INFO is enabled Jetty is too verbose at startup
log4j.logger.org.mortbay = WARN
```

```
##### The following are existing categories in ProActive
#log4j.logger.proactive = INFO
#log4j.logger.proactive.pnp = TRACE
#log4j.logger.proactive.classloader = DEBUG
#log4j.logger.proactive.events = DEBUG
#log4j.logger.proactive.runtime = DEBUG
#log4j.logger.proactive.body = DEBUG
#log4j.logger.proactive.mop = DEBUG
#log4j.logger.proactive.groups = DEBUG
#log4j.logger.proactive.sync_call = DEBUG
#log4j.logger.proactive.deployment = DEBUG
#log4j.logger.proactive.deployment.log = DEBUG
#log4j.logger.proactive.deployment.process = DEBUG
#log4j.logger.proactive.deployment.filetransfer = DEBUG,CONSOLE
#log4j.logger.proactive.filetransfer = DEBUG
#log4j.logger.proactive.nfe = FATAL
#log4j.logger.proactive.gc = DEBUG
#log4j.logger.proactive.ft = DEBUG
#log4j.logger.proactive.ft.cic = DEBUG
#log4j.logger.proactive.ft.pml = DEBUG
#log4j.logger.proactive.communication.transport.http = DEBUG
#log4j.logger.proactive.communication.rmi = DEBUG
#log4j.logger.proactive.communication.ssh = DEBUG
#log4j.logger.proactive.communication.transport.http = DEBUG
#log4j.logger.proactive.migration = DEBUG
#log4j.logger.proactive.communication.requests = DEBUG
#log4j.logger.proactive.examples = DEBUG

#log4j.logger.proactive.components = DEBUG
#log4j.logger.proactive.components.requests = DEBUG
```

¹² <http://logging.apache.org/log4j/1.2/index.html>

```

#log4j.logger.proactive.components.activity = DEBUG
#log4j.logger.proactive.components.bytecodegeneration = DEBUG
#log4j.logger.proactive.components.adl = DEBUG
#log4j.logger.proactive.components.gui = DEBUG

#log4j.logger.proactive.security = DEBUG
#log4j.logger.proactive.security.node = DEBUG
#log4j.logger.proactive.security.session = DEBUG
#log4j.logger.proactive.security.body = DEBUG
#log4j.logger.proactive.security.manager = DEBUG
#log4j.logger.proactive.security.request = DEBUG
#log4j.logger.proactive.security.runtime = DEBUG
#log4j.logger.proactive.security.policy = DEBUG
#log4j.logger.proactive.security.policyserver = DEBUG
#log4j.logger.proactive.security.crypto = DEBUG
#log4j.logger.proactive.security.psm = DEBUG

#log4j.logger.proactive.skeletons = DEBUG
#log4j.logger.proactive.skeletons.taskpool = DEBUG
#log4j.logger.proactive.skeletons.structure = DEBUG
#log4j.logger.proactive.skeletons.environment = DEBUG
#log4j.logger.proactive.skeletons.application = DEBUG
#log4j.logger.proactive.skeletons.diagnosis = DEBUG
#log4j.logger.proactive.skeletons.system = DEBUG

#log4j.logger.proactive.masterworker = DEBUG
#log4j.logger.proactive.masterworker.workermanager = DEBUG
#log4j.logger.proactive.masterworker.pinger = DEBUG
#log4j.logger.proactive.masterworker.repository = DEBUG
#log4j.logger.proactive.masterworker.workers = DEBUG

#log4j.logger.proactive.configuration = DEBUG
#log4j.logger.proactive.remoteobject = DEBUG

#log4j.logger.proactive.jmx = DEBUG
#log4j.logger.proactive.jmx.mbean = DEBUG
#log4j.logger.proactive.jmx.notification = DEBUG

#log4j.logger.proactive.webservices = DEBUG

##### Appenders #####
#
# Appender output can be configured by using a pattern layout
# See: http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html
#
# - %c the category of the logging event
# - %d the date
# - %m the application supplied message
# - %n the platform dependent line separator character or characters
# - %p the priority of the logging event
# - %t the name of the thread that generated the logging event
# - %X{hostname} the hostname
# - %X{id@hostname} the VMID and the hostname

```

```
# - %X{shortid@hostname} the short VMID and the hostname (a collision can occur between two shortids, you should
use id@hostname)
# - %X{runtime} the ProActive runtime url (does not work very well since a different MDC is associated to each
thread)

# CONSOLE appender is used by default
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=%X{shortid@hostname} - [%d %p %20.20c{2}] %m%n

# Appender FILE writes to the file "tests.log".
# This file is recreated a file for each run
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=tests.log
log4j.appender.FILE.Append=false
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern=%5p [%t]: %m%n
```



Note

If you use the scripts provided with the distribution to run the examples, an existing log4j file named `proactive-log4j` will be used by default.

2.2.5. Run A Few Examples For Testing

You can try to run the test applications provided with ProActive. Each example comes with a script to launch the application. This script is located in `ProActive/examples`. All the example code sources can be found in the directory `ProActive/src/Examples/org/objectweb/proactive/examples/`.

2.2.5.1. Reader/Writer

This example is the ProActive version of the Readers/Writers canonical problem. To illustrate the ease-of-use of the ProActive model, different synchronization policies can be applied without even stopping the application. This example is based on a easy to use Swing GUI.

- script: `readers.sh` or `readers.bat`
- source: `examples/readers`

2.2.5.2. The Dining Philosophers

This example is one possible implementation of the well-known Dining Philosophers synchronization problem. This example is based on a easy to use Swing GUI.

- script: `philosophers.sh` or `philosophers.bat`
- source: `examples/philosophers`

2.2.5.3. The N-Body Simulation

This example can be used later on as a deployment example for several machines, and also for the fault-tolerance features.

- script: `nbody.sh` or `nbody.bat`
- source: `examples/nbody`

Chapter 3. ProActive Troubleshooting

In this section, we present common problems encountered while trying to use ProActive. For further assistance, or if you cannot solve a problem, feel free to post your question on the ProActive mailing list proactive@ow2.org. Make sure that you include a precise description of your problem along with a full copy of your error message.

3.1. Common problems

If you encounter any problem with installing ProActive and running examples, please make sure you have correctly followed all the steps described in the previous chapter. If it doesn't help, here is a list of the most common mistakes:

- **Permission denied when trying to launch scripts under Linux** - Files have the wrong permissions set. You have to change permissions by executing `chmod -R 744 *.sh` in the `ProActive/examples` directory. The command will set the files to executable and readable for all users and also writeable for the current user.
- **Java complains about not being able to find the ProActive classes** - Your `CLASSPATH` environment variable does not contain the entry for classes belonging to ProActive, ASM, Log4, Xerces, Fractal, Ibis or BouncyCastle. Normally, only `ProActive.jar` must be in your `CLASSPATH` since the other libraries are indexed into the `ProActive.jar` file. However, if these others libraries are not in the same directory than `ProActive.jar`, then you have to insert them into your `CLASSPATH`. In any case, you can try to include them to your `CLASSPATH` but it might lead to some problems under Windows since the command line to launch applications will be very long and Windows will probably reject it.
- **Java complains about denial of access** - If you get the following exceptions, you probably didn't change the file `java.policy` as described in [Section 2.2.3, "Create a proactive.java.policy File To Set Permissions"](#).

```
org.objectweb.proactive.NodeException:
java.security.AccessControlException: access denied
(java.net.SocketPermission 127.0.0.1:1099 connect,resolve)
    at org.objectweb.proactive.core.node.rmi.RemoteNodeImpl.<init>(RmiNode.java:17)
    at org.objectweb.proactive.core.node.rmi.RemoteNodeFactory._createDefaultNode
        (RmiNodeFactory.java, Compiled Code)
    at org.objectweb.proactive.core.node.NodeFactory.createDefaultNode(NodeFactory.java:127)
    at org.objectweb.proactive.core.node.NodeFactory.getDefaultNode(NodeFactory.java:57)
    at org.objectweb.proactive.ProActive.newActive(ProActive.java:315)
    ...
Exception in thread "main" java.lang.ExceptionInInitializerError:
java.security.AccessControlException: access denied
(java.util.PropertyPermission user.home read)
    at java.security.AccessControlContext.checkPermission (AccessControlContext.java, Compiled Code)
    at java.security.AccessController.checkPermission(AccessController.java:403)
    at java.lang.SecurityManager.checkPermission(SecurityManager.java:549)
    at java.lang.SecurityManager.checkPropertyAccess(SecurityManager.java:1243)
    at java.lang.System.getProperty(System.java:539)
    at org.objectweb.proactive.mop.MOPProperties.createDefaultProperties (MOPProperties.java:190)
    ...
```

- **Java complains on log4j initialization** - If you get the following message, you probably made a mistake when giving the `Dlog4j.configuration` property to the `java` command. Be sure that the given path is right, try also to add `file:` before the path.

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

- **Examples and compilation do not work at all under Windows system** - Check whether your `java` installation is not in a path containing spaces like `C:\Program Files\java` or `C:\Documents and Settings\java`. Batch scripts do not run properly when `JAVA_HOME` is set to such a directory. The solution is to install the `JDK` under a space-free path (e.g. `C:\java\jdk...` or `D:\java\jdk...`) and then, set the `JAVA_HOME` environment variable accordingly.

- **Java complains about unsupported classes** - Check that you are using a JRE that is at least version 1.5 since ProActive needs certain features which are only available starting from Java 1.5.

```
Exception in thread "main" java.lang.UnsupportedClassVersionError: org/objectweb/proactive/core/util/wrapper/
StringWrapper (Unsupported major.minor version 49.0)
  at java.lang.ClassLoader.defineClass0(Native Method)
  at java.lang.ClassLoader.defineClass(Unknown Source)
  at java.security.SecureClassLoader.defineClass(Unknown Source)
  at java.net.URLClassLoader.defineClass(Unknown Source)
  at java.net.URLClassLoader.access$100(Unknown Source)
  at java.net.URLClassLoader$1.run(Unknown Source)
  at java.security.AccessController.doPrivileged(Native Method)
  at java.net.URLClassLoader.findClass(Unknown Source)
  at java.lang.ClassLoader.loadClass(Unknown Source)
  at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
  at java.lang.ClassLoader.loadClass(Unknown Source)
  at java.lang.ClassLoader.loadClassInternal(Unknown Source)
```

3.2. Enabling the loggers

To enable the debugging loggers, the following log file can be used:

```
-Dlog4j.configuration=file:ProActive/examples/proactive-log4j
```

In this file, the relevant loggers can be uncommented (by removing the leading #). For example, the deployment loggers are activated with the following lines:

```
log4j.logger.proactive.deployment = DEBUG, CONSOLE
log4j.logger.proactive.deployment.log = DEBUG, CONSOLE
log4j.logger.proactive.deployment.process = DEBUG, CONSOLE
```

3.3. Domain name resolution problems

To work properly, ProActive requires machines to have a correctly configured host and domain name. If the name of a machine is not properly configured, then remote nodes will be unable to locate the machine. To work around bad-configured domain names, ProActive can be activated to use IP addresses through the following java property:

```
-Dproactive.useIPAddress=true
```

This property should be given as parameter to Java virtual machines deployed on computers which have names that do not resolve properly.

3.4. RMI Tunneling

ProActive provides RMI tunneling through ssh for crossing firewalls that only allow ssh connections. Here is a check-list for using rmissh tunneling:

- ProActive/lib/trilead-ssh2.jar must be included in the CLASSPATH of the concerned machines.
- JVMs that are only accessible with ssh must be started using: `-Dproactive.communication.protocol=rmissh`
- A key without a passphrase must be installed on the machine accepting ssh connections. It should be possible to log into the site without using an ssh-agent and without providing a password.

Part II. Guided Tour and Tutorial

Table of Contents

Chapter 4. Introduction	17
4.1. Overview	17
4.2. Installation and setup	17
4.3. ProActive and IDEs (Eclipse, ...)	17
Chapter 5. ProActive Example Applications	21
5.1. C3D: A distributed 3D renderer	21
5.1.1. How to use C3D	22
5.2. Readers/Writers Application	25
5.2.1. How to use the Readers/Writers	26
5.3. The dining philosophers	27
5.3.1. How to use the philosophers application	28
5.4. The migratory penguins	30
5.4.1. How to use the penguin application	30
5.4.2. How to use the Penguin Controller	32
5.5. Chat example	32
5.5.1. How to run the Chat application	32
5.5.2. Chat migration	34
5.6. Integral Pi	34
5.6.1. Introduction	34
5.6.2. Initialization	35
5.6.3. Communication primitives	36
5.6.4. Running the PI example	37
5.7. The nbody example	38
5.7.1. How to run the n-body example	39
5.7.2. Barnes-Hut	40
5.8. Conclusion	42
Chapter 6. Active Object Tutorial	43
6.1. Simple Computation And Monitoring Agent	43
6.1.1. Classes Used	44
6.1.2. CMA Architecture and Skeleton Code	44
6.1.3. Proposed Work	46
6.1.4. Solutions and Full Code	46
6.2. Active Objects Lifecycle: Using InitActive, RunActive and EndActive	47
6.2.1. Classes Used	48
6.2.2. Initialized CMA Architecture and Skeleton Code	48
6.2.3. Proposed Work	49
6.2.4. Solution and Full Code	50
6.3. Remote Monitoring Agent	51
6.3.1. Classes Used	51
6.3.2. Deployed CMA Architecture and Skeleton Code	52
6.3.3. Proposed Work	53
6.3.4. Solution and Full Code	53
6.4. Agent synchronization	56
6.4.1. Classes Used	56

6.4.2. Architecture and Skeleton Code	57
6.4.3. Proposed Work	59
6.4.4. Solution And Full Code	59
6.5. Monitoring Several Computers Using Migration	61
6.5.1. Classes Used	61
6.5.2. Design of Migratable Monitoring Agent and Skeleton Code	62
6.5.3. Proposed Work	64
6.5.4. Solution and Full Code	65
6.6. Groups of Monitoring Agents	67
6.6.1. Classes Used	67
6.6.2. Architecture and Skeleton Code	68
6.6.3. Proposed Work	70
6.6.4. Solution And Full Code	71
6.7. Monitoring Agent As A Web Service	73
6.7.1. Classes Used	73
6.7.2. Architecture and Skeleton Code	73
6.7.3. Proposed Work	76
6.7.4. Solution And Code	76
6.8. Primality test tutorial	76
6.8.1. A Sequential Version of the Primality Test	77
6.8.2. A Distributed Version of the Primality Test	78
6.8.3. A Distributed Version of the Primality Test Using the Master-Worker API	89

Chapter 4. Introduction

4.1. Overview

This tour is a practical introduction to ProActive giving an illustrated introduction to some of the functionality and facilities offered by the library.

We will introduce several features of the library through examples (see [Chapter 4, Introduction](#)). If after reading this chapter you need further details on how the examples work, visit the [ProActive applications page](#)¹.

[Chapter 6, Active Object Tutorial](#) will provide you with practical experiences on how to program using ProActive by showing how to write your own client-server monitoring agent with active objects. This chapter will show different basic features of ProActive through examples that increase in complexity.

The second part of the tutorial will point out how to program using the ProActive high level APIs. It will introduce the MasterWorker API, programming with components and the ProActive Scheduler.

4.2. Installation and setup

To get started, please follow instructions for downloading and installing ProActive (see [Chapter 2, ProActive Installation](#)).

Example applications in [Chapter 4, Introduction](#) can be run by going to `ProActive/examples/` directory and starting the corresponding script. You do not need to set up any environment variables as these are set up automatically by the scripts. The scripts are platform dependant: use the `.sh` files on Unix/Linux and the `.bat` files on Windows.

For programming examples, you will need to use a policy file, such as `ProActive/examples/proactive.java.policy`, with the JVM option `-Djava.security.policy=filelocation/proactive.java.policy`. You will also need a `log4j` configuration file. An example file you can use is `ProActive/examples/proactive-log4j`. The JVM option for the `log4j` configuration file is `-Dlog4j.configuration=file:/filelocation/proactive-log4j`.

4.3. ProActive and IDEs (Eclipse, ...)

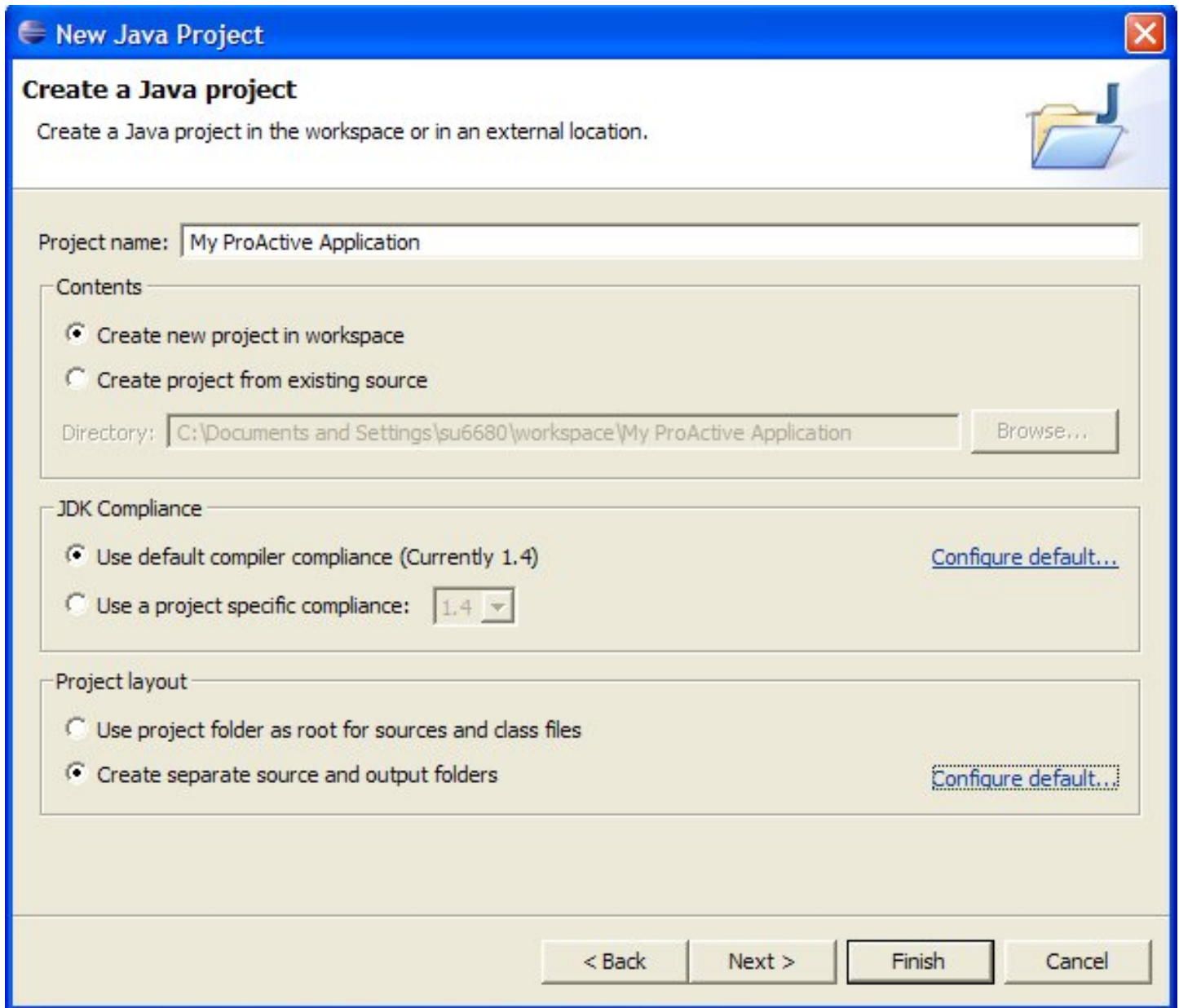
We recommend you to use the Eclipse IDE to develop your ProActive applications. You can get this tool on the [Eclipse website](#)². Just unzip and launch the `eclipse` executable. In order to develop your own ProActive application, you will need to create an Eclipse project:

File -> New ... -> Project

Then, choose **Java Project**. A wizard will appear and ask you to enter the project name:

¹ <http://proactive.inria.fr/applications.htm>

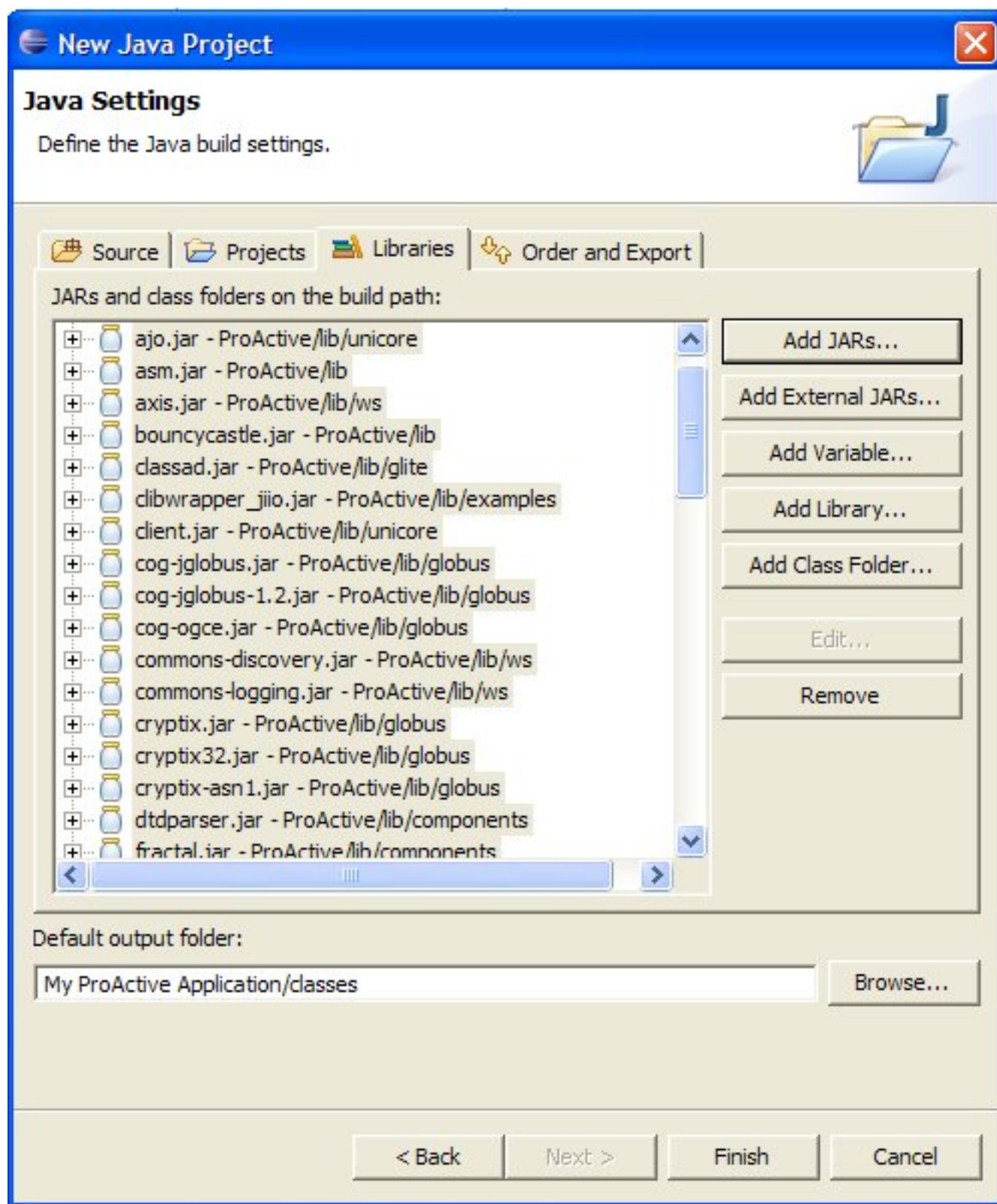
² <http://www.eclipse.org>



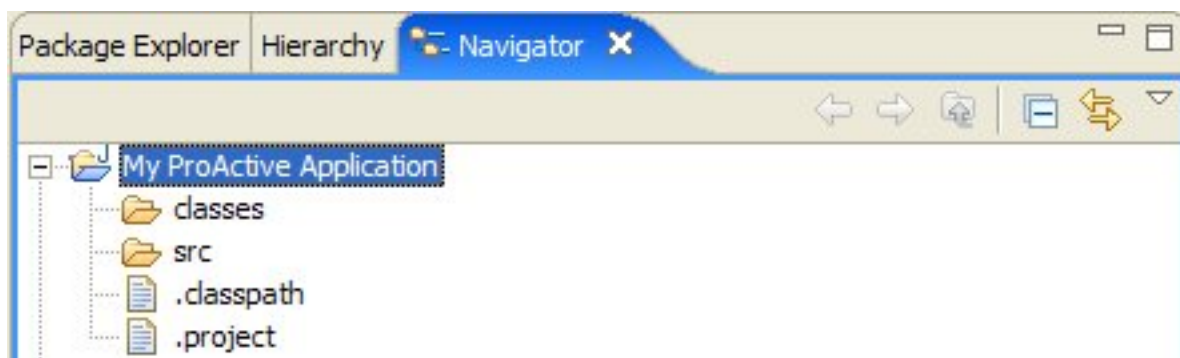
In order to separate class files from source files (it can be useful if you write scripts which refer to these classes), you can check the **Create separate source and output folders** in the **Project Layout** Frame, and click on **Configure Default...** to choose the folders names. Once you are done with page, click on **Next**.

You have to specify some Java settings in order to set the application classpath. Select the **Libraries** tab and click on the **Add External Jar...** button. Add the ProActive.jar and the libraries contained in the ProActive/dist/lib/ directory of the ProActive distribution.

The **Libraries** tab should look like this:



You can now see on the navigator tab on the left side that there is a new Project with the source and output folders you have just created:



You are now able to create classes and packages that use the ProActive library.

Everything is now configured to create your ProActive application. Click on the **Finish** button to terminate.

We have developed an Eclipse application that will help developers to easily monitor ProActive applications. To learn how to monitor ProActive applications read the ProActive IC2D documentation.

Chapter 5. ProActive Example Applications

5.1. C3D: A distributed 3D renderer

Distribution is often used for CPU-intensive applications where parallelism is a key for performance. The parallelization of programs can be facilitated with ProActive, thanks to asynchronous method calls (see [Asynchronous calls and futures](#)¹), as well as group communications (see [Typed Group Communication](#)²).

To illustrate how parallelization can be used for computationally intensive tasks, we have built the C3D application. [C3D](#)³ is a Java benchmark application that measures the performance of a 3D raytracer renderer distributed over several Java virtual machines using Java RMI. It shows some of the benefits of ProActive, in particular the ease of distributed programming and the speedup through distributed parallel calculation. This benchmark gives indication of the performance of the serialization process and Java RMI itself. The benchmark is an automated version of C3D which is both a collaborative application and a distributed raytracer: users can interact through messaging and voting facilities in order to choose a scene that is rendered using a set of distributed rendering engines working in parallel. Near-linear speedup is achieved with up to half a dozen rendering engines running in parallel that communicate using Java RMI and controlled by a central dispatcher.

¹ file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/./../ReferenceManual/multiple_html/ActiveObjectCreation.html#FutureObjectCreation

² file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/./../ReferenceManual/multiple_html/TypedGroupCommunication.html

³ <http://proactive.inria.fr/c3d.htm>

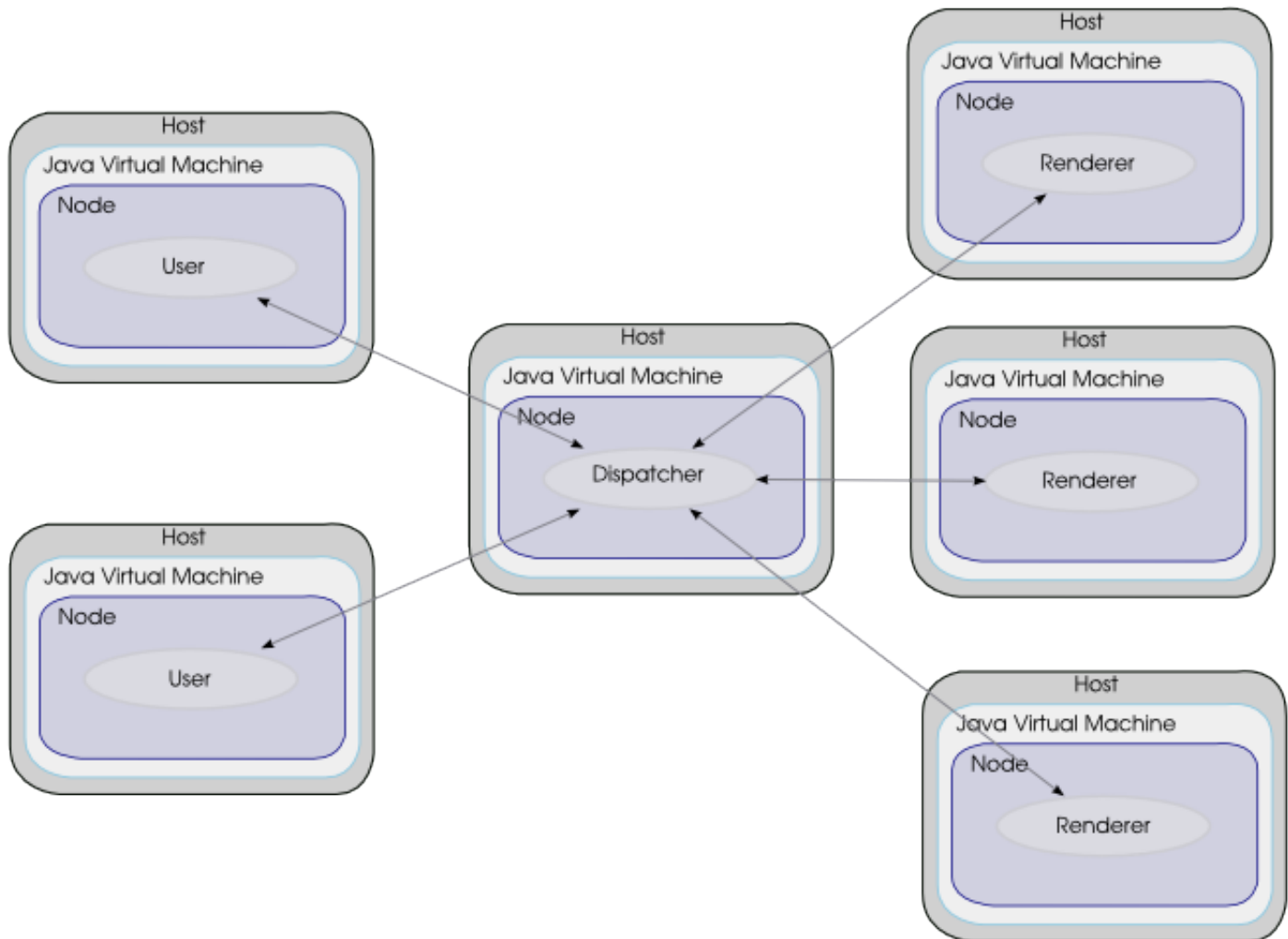


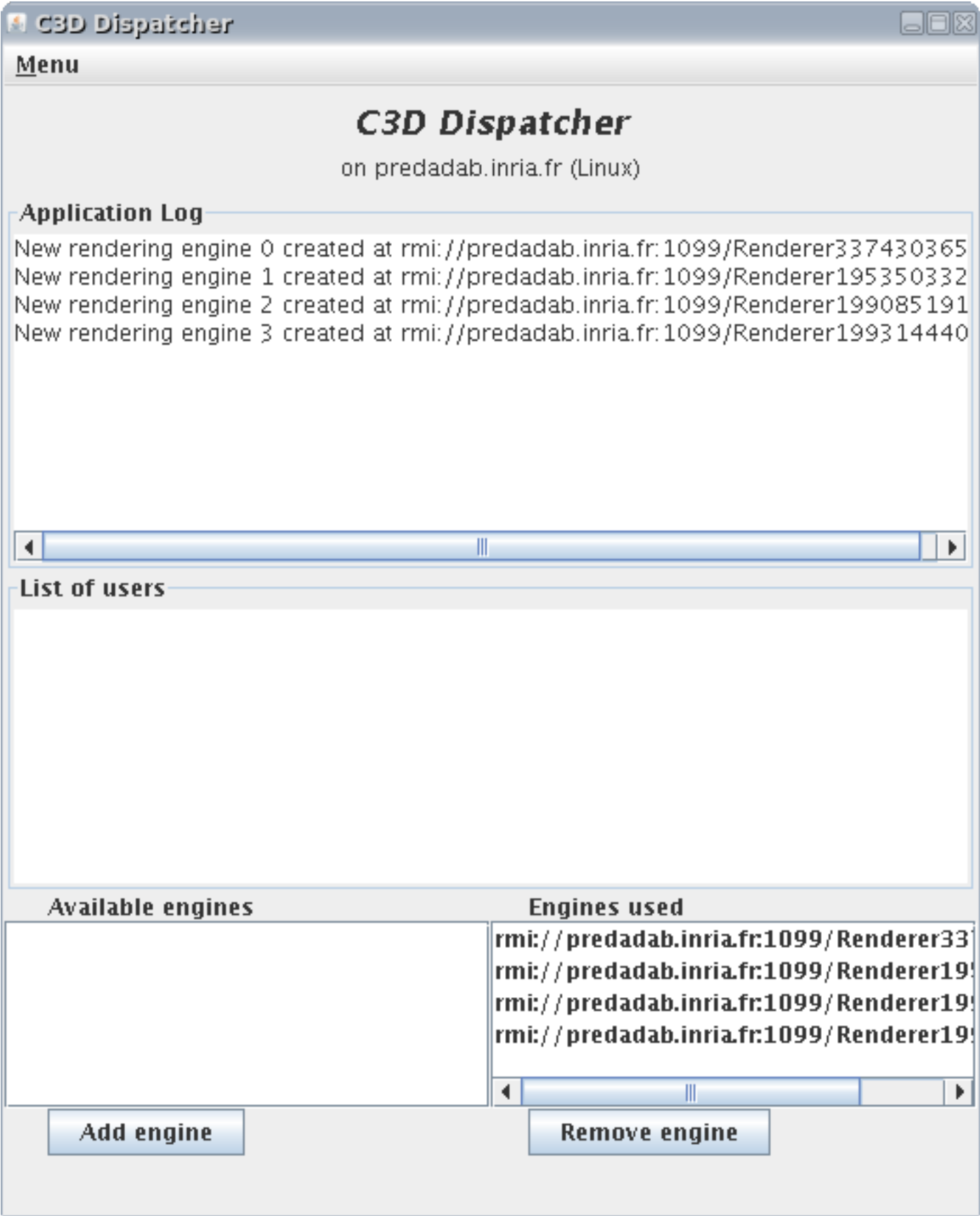
Figure 5.1. Active objects in the c3d application

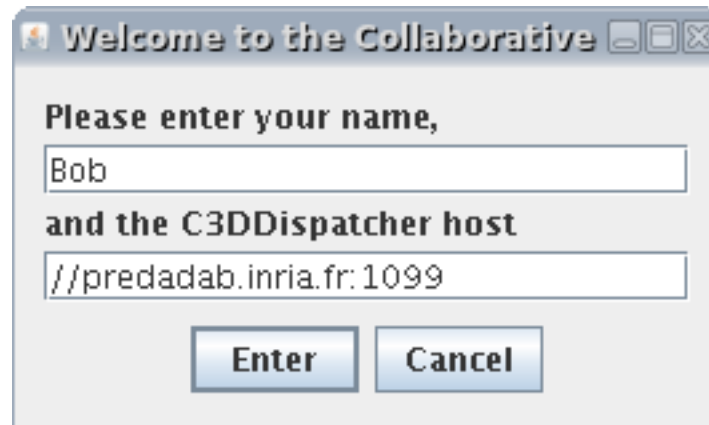
5.1.1. How to use C3D

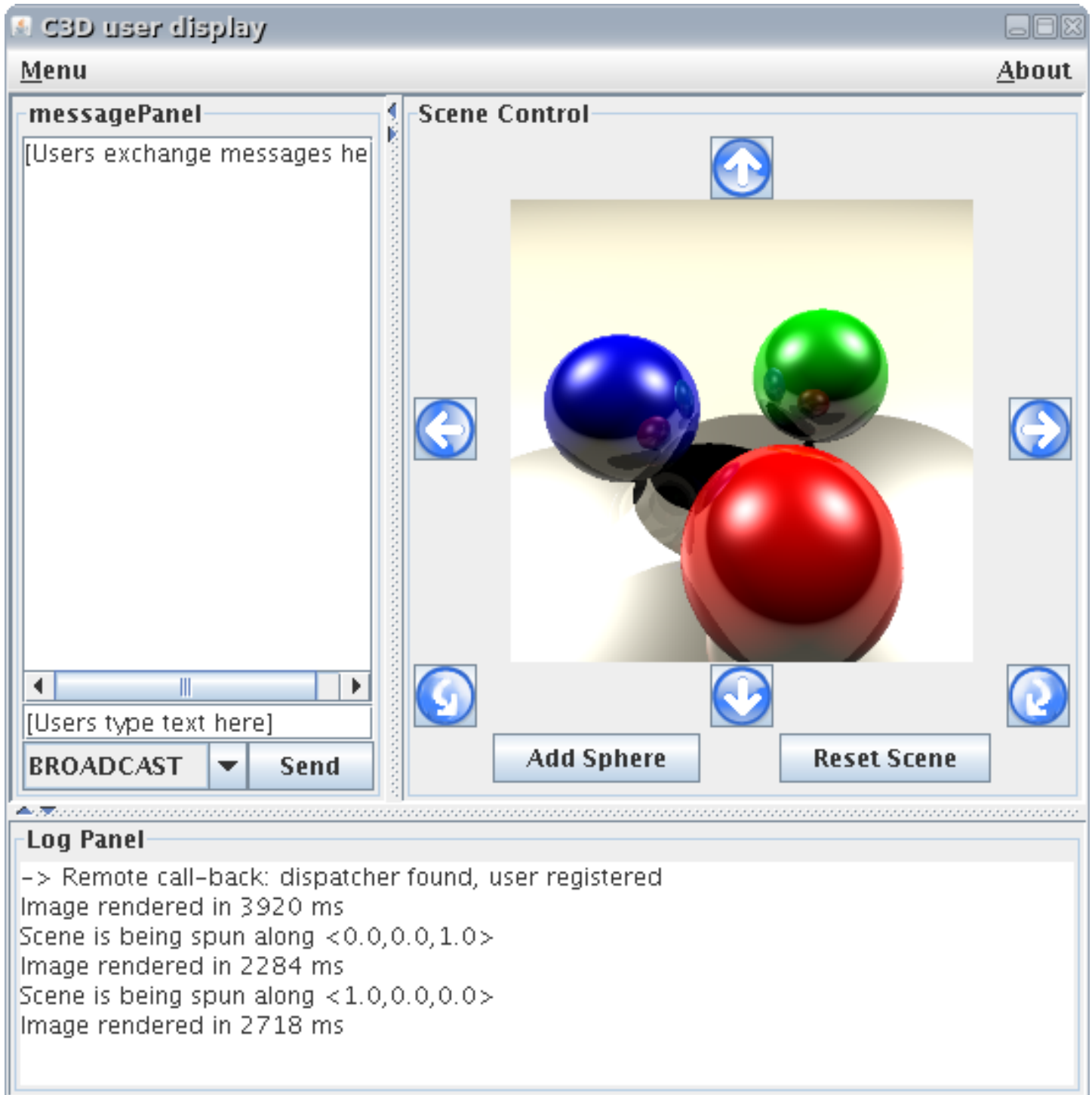
The C3D application uses three scripts that can be found under the `ProActive/examples/c3d` directory:

- `c3d_one_user.sh` or `c3d_one_user.bat` launches the C3D Dispatcher with four rendering engines and one user.
- `c3d_no_user.sh` or `c3d_no_user.bat` launches the C3D Dispatcher with four rendering engines and no user
- `c3d_add_user.sh` or `c3d_add_user.bat` adds another users

To use the application first start the dispatcher using either `c3d_one_user.[sh, bat]` or `c3d_no_user.[sh, bat]` and then add users with `c3d_add_user.[sh, bat]`.







5.2. Readers/Writers Application

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

The readers/writers application shows the synchronization capabilities of ProActive. In order to allow concurrency while ensuring the consistency of the readings, data accesses have to be synchronized with a specified policy.

The [implementation with ProActive](http://proactive.inria.fr/reader_writers.htm)⁴ uses 3 active objects: Reader, Writer, and the controller class (ReaderWriter).

5.2.1. How to use the Readers/Writers

To start application, use `examples/readers`.

ProActive starts a node (on an already existing JVM or on a new one) on the current machine and creates 3 Writers, 3 Readers, a ReaderWrite (the application controller) and a ReaderDisplay, which are all active objects.

The examples has three synchronization modes "Priority to Writers", "Priority to Readers", and "Even Policy". The application is able to assign different priority to readers or writers without blocking on reading or writing.

⁴ http://proactive.inria.fr/reader_writers.htm

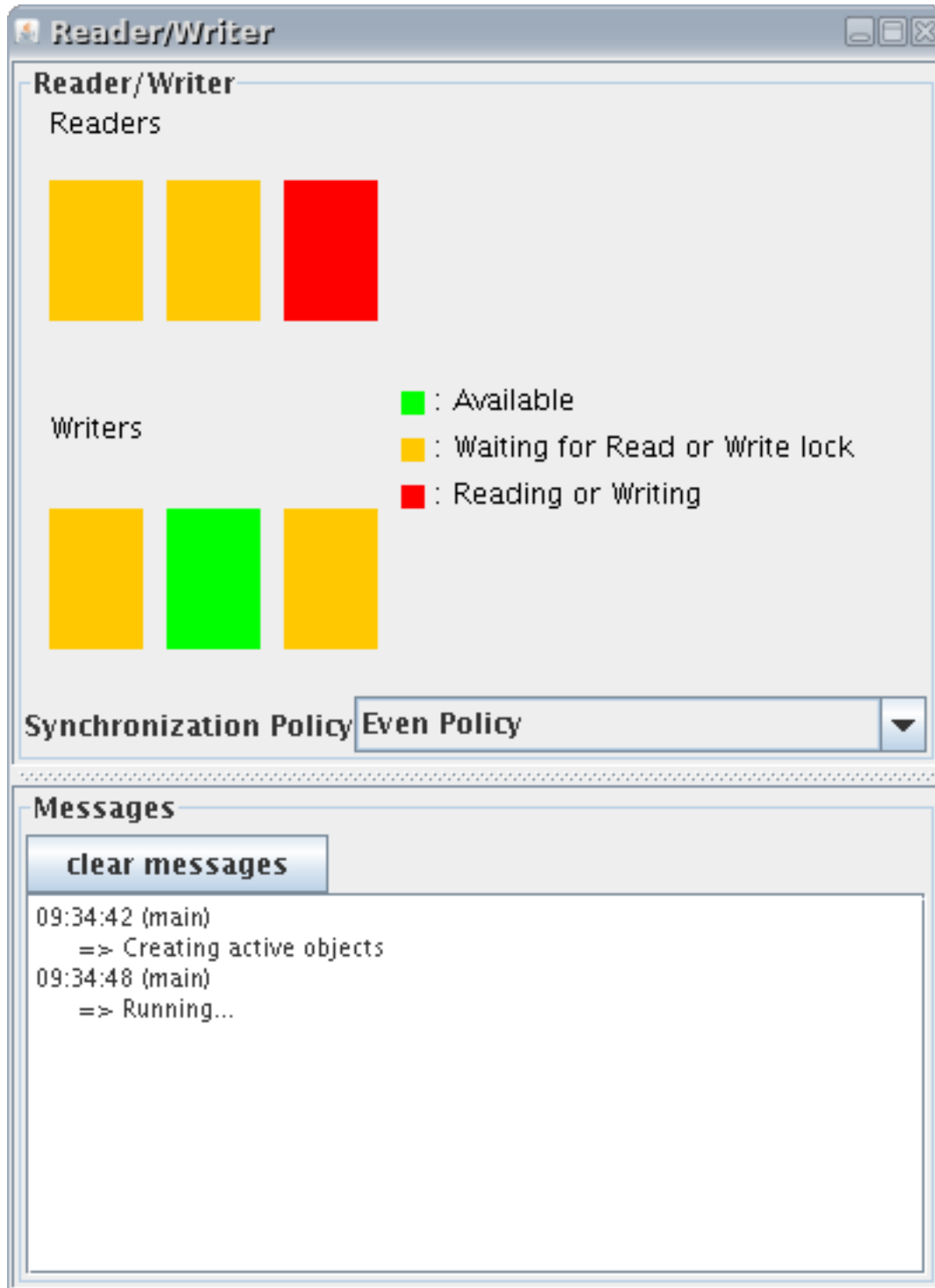


Figure 5.2. Started GUI illustrating the activities of Reader and Writer objects.

5.3. The dining philosophers

The 'dining philosophers' problem is a classic exercise in concurrent programming with the goal of avoiding deadlocks. In this problem, there are five philosophers sitting around a dinner table and sharing five forks. The philosophers alternate between eating and thinking.

When eating, a philosopher must use the two forks adjacent to him. As there are not enough forks for all the philosophers to eat at the same time, the possibility of deadlock arises.

We have provided [an illustration of the solution](http://proactive.inria.fr/dining_philosophers.htm)⁵ using ProActive, where all the philosophers are active objects, as well as the table (controller) and the dinner frame (user interface).

5.3.1. How to use the philosophers application

To start the application use the `examples/philosophers/philosophers.sh` or `examples\philosophers\philosophers.bat` script depending on your operating system.

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

⁵ http://proactive.inria.fr/dining_philosophers.htm

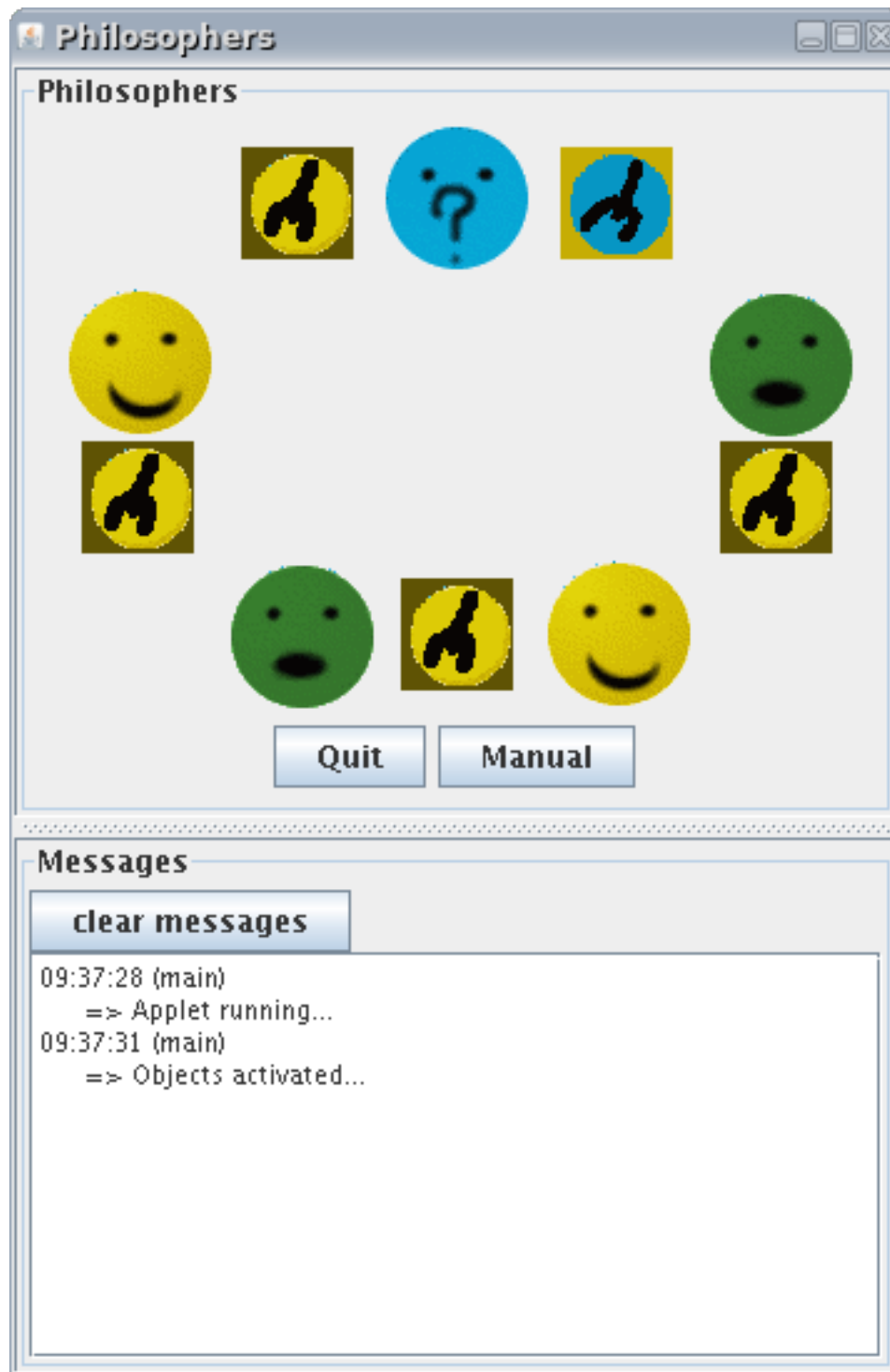







Figure 5.3. The Dining Philosophers Example

The pictures represent the state of the philosophers. They can be:

- 
philosophizing
- 
hungry, wants the fork!
- 
eating

The forks can have two states:

- 
taken
- 
free

You can either run the application in autopilot mode running it on its own without encountering a deadlock or in manual mode where you click on the philosophers' heads to switch their modes. With the second mode, you can, for instance, make one of the philosophers starve, making his two closer neighbours alternatively eat and think.

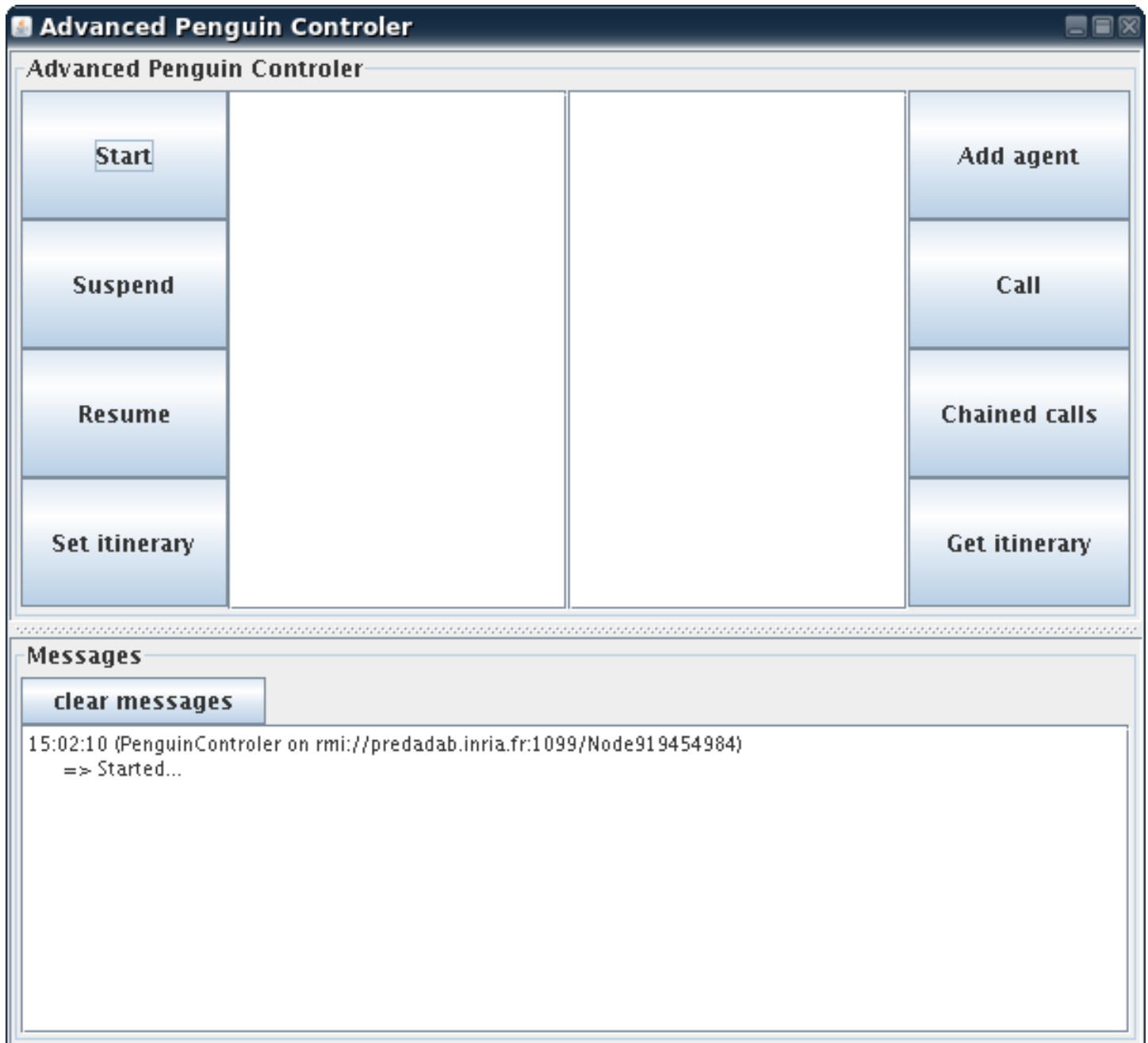
5.4. The migratory penguins

ProActive allows the transparent migration of active objects between virtual machines. The migration happens while the application containing the active objects is running and without interruptions in the application.

The penguin migration example shows how a set of mobile agents can move around from machine to machine while still communicating with their base and with each other. It also features the ability to move a swing window between screens while moving an agent from one JVM to the other.

5.4.1. How to use the penguin application

Use the `examples/penguin/penguin.sh` or `examples/penguin/penguin.bat` script to start the penguin controller. The controller allows you to add a new agent (penguin), and control its route.





The active object is moving between machines (specified in the deployment descriptor) and the penguin window disappears and reappears on the screen associated with the new JVM.

5.4.2. How to use the Penguin Controller

After selecting them, use buttons to:

- Add agents - Start Agent
- Communicate with them ('chained calls')
- Send the agent on the itinerary - Start, Suspend, Resume
- Trigger a communication between them ('call another agent')

5.5. Chat example

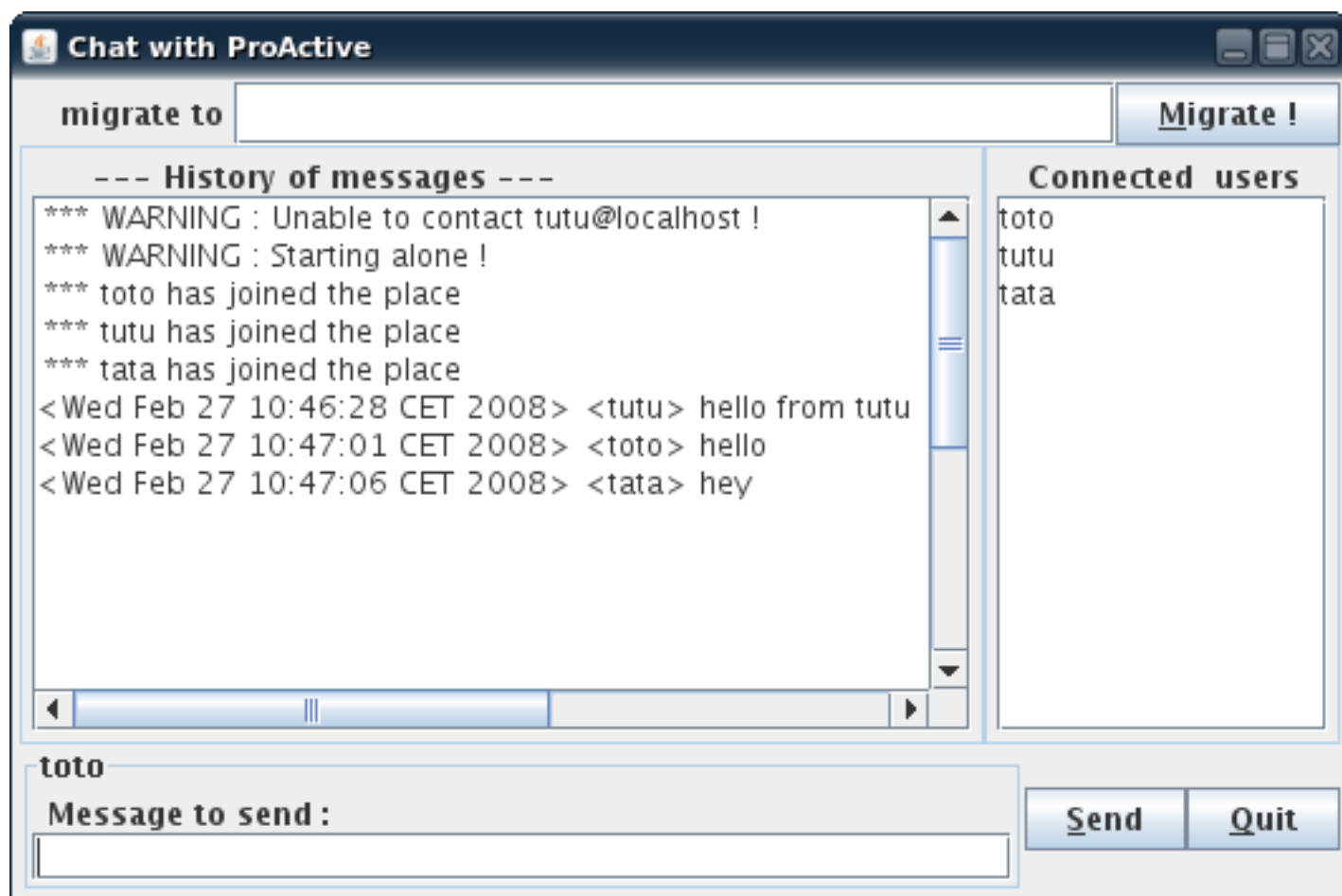
ProActive has support for groups and migration. To show this, we have programmed a simple chat application using the groups framework. The application clients can communicate with each other in a decentralized manner and can migrate from computer to computer.

5.5.1. How to run the Chat application

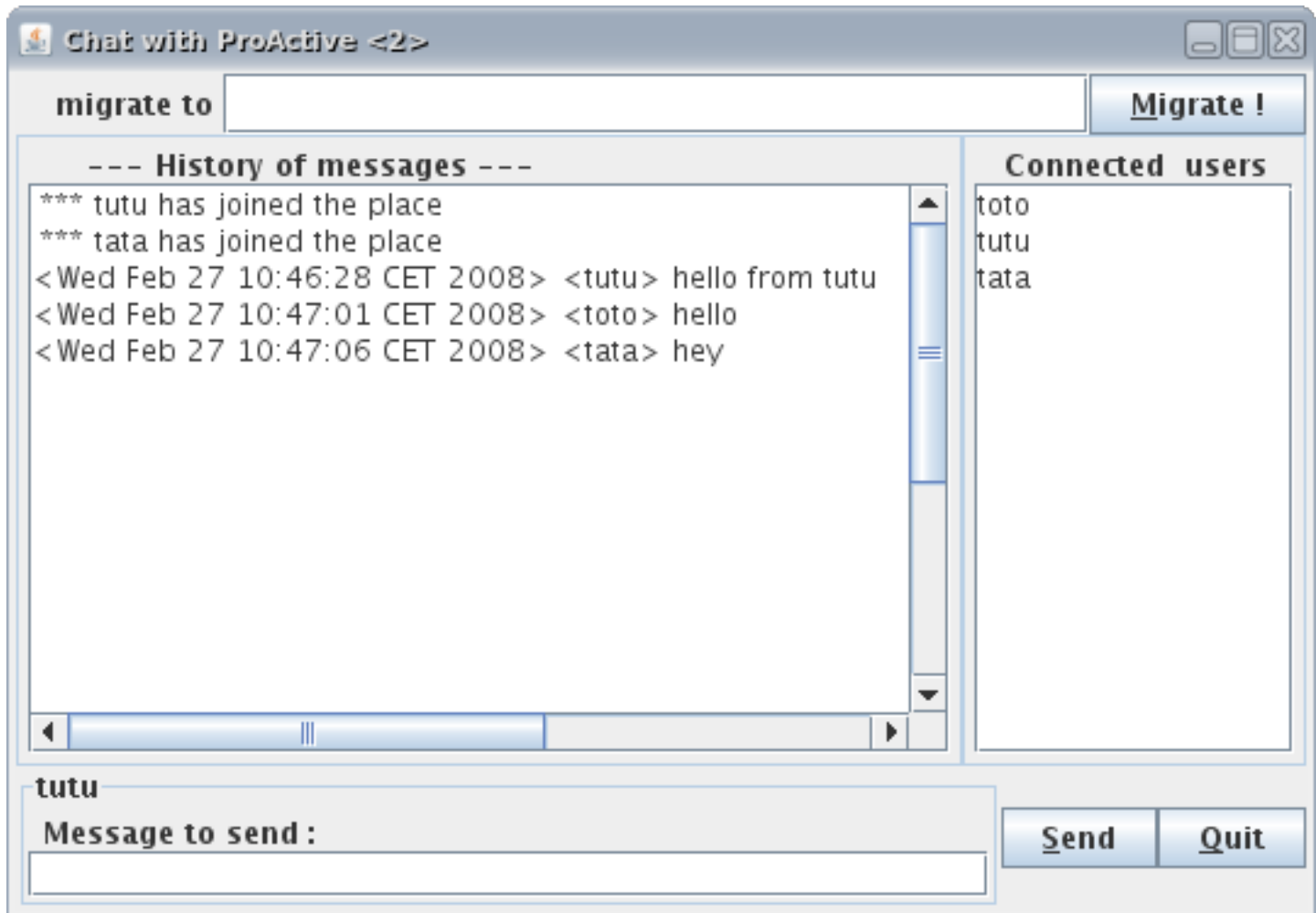
To start a chat client, run `chat.sh` or `chat.bat` from the `examples/chat/` folder. The syntax is `chat.[sh|bat] user_name host_to_connect_to user_to_connect_to`.

When you start the first client the last argument can have any value. However if you want to connect to another chat you have to supply a valid user name and host. The communication is not dependent on any particular user. The first user that created the chat can leave and the other users can still communicate.

Start the first user by running `chat.[sh|bat] toto localhost no_user`. The following window appears:



To connect the user "tutu", run `chat.[sh|bat] tutu localhost toto`.



You can later connect other users in the same manner. Run `chat.[sh|bat] tata localhost tutu`. You can connect to any user in the chat and you will have access to all the users.

5.5.2. Chat migration

The chat application can be migrated to another computer. All we have to do in order to fully migrate one chat is to start a node on the remote computer using `startNode.sh` or `startNode.bat` from the `ProActive/bin/` folder. We have to use a parameter in the form of `rmi://hostname:port/node_name` to start the node. Port number is not mandatory. If you do not precise it, the node will be started on the default port, that is 1099. After starting the node, we can migrate the chat application by putting the URL `//hostname:port/node_name` in the "migrate to" textbox and clicking on the Migrate button. For the migration, port number is mandatory. The application will be recreated on the remote computer with its state intact and ready to communicate to the other users.

5.6. Integral Pi

5.6.1. Introduction

In this chapter, we are going to see a simple example of an MPI written program ported to ProActive.

First let's introduce what we are going to compute.

The MPI PI program approximates π ⁶ by computing:

⁶ <http://en.wikipedia.org/wiki/Pi>

$$\Pi = \int_0^1 \frac{4}{1+x^2} dx$$

Which is approximated by:

$$\Pi \simeq \sum_{k=1}^N \frac{4}{1 + \left(k - \frac{1}{2}\right)^2}$$

The only input data required is therefore N , the number of iterations.

Involved files :

- ProActive/src/Examples/org/objectweb/proactive/examples/documentation/integralpi/int_pi2.c: the original MPI implementation
- ProActive/src/Examples/org/objectweb/proactive/examples/integralpi/Launcher.java: the main class
- ProActive/src/Examples/org/objectweb/proactive/examples/integralpi/Worker.java: the class implementing the SPMD code

5.6.2. Initialization

5.6.2.1. MPI Initialization primitives

Some basic primitives are used. Notice that MPI provides a rank to each process and the group size (the number of involved processes).

```
/* All instances call startup routine to get their instance number (mynum) */
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &mynum);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

/* Step (1): get a value for N */
solicit (&N, &nprocs, mynum);
```

5.6.2.2. ProActive Initialization primitives

First, we need to create the group of workers (MPI processes represented by active objects). Notice that the creation of active objects is done in Launcher.java.

The group of active objects is created using given parameters and nodes specified in the deployment descriptor.

```
// Group creation
Worker workers = (Worker) PAsPMD.newSPMDGroup(Worker.class.getName(), params, nodesArray);
DoubleWrapper wrappedResult;

while (numOfIterations > 0) {
    // Send the number of iterations to the first worker
    Worker firstWorker = (Worker) PAGroup.getGroup(workers).get(0);
    wrappedResult = firstWorker.start(numOfIterations);
}
```

As with MPI, the ProActive's SPMD layer provides some initialization primitives. In Worker.java you can identify this initialization. Note that one-to-one communications will be done thanks to an array view on the created group.

```
// ProActive initialization
rank = PAsPMD.getMyRank();
groupSize = PAsPMD.getMySPMDGroupSize();

// Get all workers references
workersArray = (Worker[]) PAGroup.getGroup(PAsPMD.getSPMDGroup()).toArray(new Worker[0]);
```

5.6.3. Communication primitives

5.6.3.1. Communication pattern

The communication pattern is very simple. It is done in two steps: first, the process 0 broadcasts N processes and then waits for the result from each other process and secondly, sums the received values.

5.6.3.2. MPI Approach

The MPI implementation involves 3 communication primitives:

- **MPI_Send** (Sends data to one process)
- **MPI_Recv** (Receives data from a sending process)
- **MPI_Bcast** (Broadcast a data to all processes)

Please note that MPI_Bcast, MPI_Send and MPI_Recv primitives are blocking.

```
/* Step (4): print the results
 * Parallel version: collect partial results and let master instance
 * print it.
 */
if (mynum==0) {
    printf ("host calculated x = %7.5f\n", sum);
    for (i=1; i<nprocs; i++) {
        source = i;
        info = MPI_Recv(&x, 1, MPI_FLOAT, source, type, MPI_COMM_WORLD, &status);
        printf ("host got x = %7.5f\n", x);
        sum=sum+x;
    }
    err = sum - pi;
    printf ("sum, err = %7.5f, %10e\n", sum, err);
    fflush(stdout);
}
/* Other instances just send their sum and wait for more input */
else {
    info = MPI_Send(&sum, 1, MPI_FLOAT, dest, type, MPI_COMM_WORLD);
    if (info != 0) {
        printf ("instance no, %d failed to send\n", mynum);
        exit(0);
    }
    printf ("inst %d sent partial sum %7.2f to inst 0\n", mynum, sum);
    fflush(stdout);
}
/* get a value of N for the next run */
solicit (&N, &nprocs, mynum);
}
```

Where the solicit method looks like this:

```

void solicit (pN, pnprocs, mynum) int *pN, *pnprocs, mynum; {
    /* Get a value for N, the number of intervals in the approximation.
     * (Parallel versions: master instance reads in N and then
     * broadcasts N to all the other instances of the program.)
     */
    int source = 0;

    if (mynum == 0) {
        printf ("Enter number of approximation intervals:(0 to exit)\n");
        scanf ("%d", pN);
    }
    MPI_Bcast(pN, 1, MPI_INT, source, MPI_COMM_WORLD);
}

```

5.6.3.3. ProActive Approach

The ProActive implementation is quite similar to MPI one. The fact is that all communications in ProActive are asynchronous (non-blocking) by default, therefore we need to specify explicitly to block until a specific request.

```

// The leader collects partial results.
// Others just send their computed data to the rank 0.
if (rank == 0) {
    for (i = 1; i < groupSize; i++) {
        body.serve(body.getRequestQueue().blockingRemoveOldest("updateX")); // block until an updateX call
        sum += x;
    }
} else {
    workersArray[0].updateX(sum);
}

```

The leader blocks his request queue until another worker will do a distant call on the leader's **updateX** method which is:

```

/**
 * This method will be called remotely by a worker to send its value.
 *
 * @param value The value to remotely update.
 */
public void updateX(double value) {
    this.x = value;
}

```

5.6.4. Running the PI example

In examples/integralPi, run `integralpi.[sh|bat]`. You can specify the number of workers from the command line. Feel free to edit scripts to specify another deployment descriptor.

```

bash-3.00$ ./integralpi.sh

--- IntegralPi -----
The number of workers is 4
--> This ClassFileServer is reading resources from classpath 2011
Created a new registry on port 1099
ProActive Security Policy (proactive.runtime.security) not set. Runtime Security disabled
***** Reading deployment descriptor: file:./../descriptors/Matrix.xml *****
created VirtualNode name=matrixNode

```

```

**** Starting jvm on amda.inria.fr
**** Starting jvm on amda.inria.fr
**** Starting jvm on amda.inria.fr
ProActive Security Policy (proactive.runtime.security) not set. Runtime Security disabled
--> This ClassFileServer is reading resources from classpath 2012
ProActive Security Policy (proactive.runtime.security) not set. Runtime Security disabled
ProActive Security Policy (proactive.runtime.security) not set. Runtime Security disabled
--> This ClassFileServer is reading resources from classpath 2013
--> This ClassFileServer is reading resources from classpath 2014
**** Starting jvm on amda.inria.fr
Detected an existing RMI Registry on port 1099
Detected an existing RMI Registry on port 1099
Detected an existing RMI Registry on port 1099
ProActive Security Policy (proactive.runtime.security) not set. Runtime Security disabled
--> This ClassFileServer is reading resources from classpath 2015
//amda.inria.fr/matrixNode2048238867 successfully bound in registry at //amda.inria.fr/matrixNode2048238867
**** Mapping VirtualNode matrixNode with Node: //amda.inria.fr/matrixNode2048238867 done
//amda.inria.fr/matrixNode690267632 successfully bound in registry at //amda.inria.fr/matrixNode690267632
**** Mapping VirtualNode matrixNode with Node: //amda.inria.fr/matrixNode690267632 done
//amda.inria.fr/matrixNode1157915128 successfully bound in registry at //amda.inria.fr/matrixNode1157915128
**** Mapping VirtualNode matrixNode with Node: //amda.inria.fr/matrixNode1157915128 done
Detected an existing RMI Registry on port 1099
//amda.inria.fr/matrixNode-814241328 successfully bound in registry at //amda.inria.fr/matrixNode-814241328
**** Mapping VirtualNode matrixNode with Node: //amda.inria.fr/matrixNode-814241328 done
4 nodes found
Generating class : pa.stub.org.objectweb.proactive.examples.integralpi.Stub_Worker

Enter the number of iterations (0 to exit) : 100000
Generating class : pa.stub.org.objectweb.proactive.examples.integralpi.Stub_Worker
Generating class : pa.stub.org.objectweb.proactive.examples.integralpi.Stub_Worker
Generating class : pa.stub.org.objectweb.proactive.examples.integralpi.Stub_Worker
Generating class : pa.stub.org.objectweb.proactive.examples.integralpi.Stub_Worker

Worker 2 Calculated x = 0.7853956634245252 in 43 ms

Worker 3 Calculated x = 0.7853906633745299 in 30 ms

Worker 1 Calculated x = 0.7854006634245316 in 99 ms

Worker 0 Calculated x = 3.141592653598117 in 12 ms

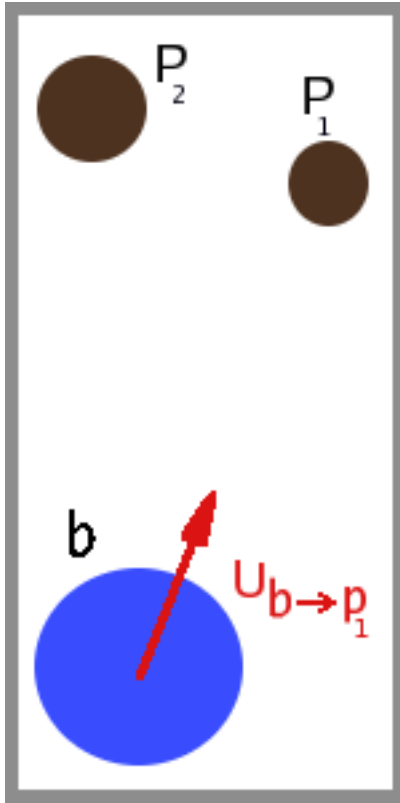
Calculated PI is 3.141592653598117 error is 8.324008149429574E-12

Enter the number of iterations (0 to exit) :

```

5.7. The nbody example

n-body is a classic problem consisting of calculating the position of bodies in space. The position depends only on the gravitational forces existing between themselves. In short, we consider several bodies (sometimes called particles) in space, where the only force is due to gravity. When only two bodies are at hand, this force is expressed as



$$F_{p \rightarrow b} = \frac{-Gm_p m_b}{r^2} \vec{u}_{p \rightarrow b}$$

$F_{p \rightarrow b}$ is the force that p applies on b , G is the gravitational constant, m_p , m_b describe the mass of the bodies, r is the distance between p and b and u is a unit vector in the direction going from p to b . When we consider all the forces that apply to one given body, we have to sum up the contribution of all the other bodies:

$$F_b = \sum_{p \in Planets} F_{p \rightarrow b}$$

This should be read as: the total force on the body b is the sum of all the forces applied to b , generated by all the other bodies in the system.

This is the force that has to be computed for every body in the system. With this force, using the usual physics formulae, (Newton's second Law)

$$F_b = ma$$

We can now compute the movement of a particle for a given time step (a the acceleration, v the velocity, x the position, t the time):

$$x(t + dt) = x(t) + v(t)dt$$

$$v(t + dt) = v(t) + a(t)dt$$

5.7.1. How to run the n-body example

In the folder ProActive/examples/, run:

```
nbody.[bat|sh] [-nodisplay | -displayft | -3d | -3dft] totalNbBodies maxIter
```

- **No parameter** starting in default mode (2D).
- **-nodisplay** starting in console mode.
- **-displayft** starting with fault-tolerance configuration.
- **-3d** starting GUI in 3D, must have [Java3d](http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html)⁷ (≥ 1.4) installed and also must have ProActive compiled with it.
- **-3dft** same as above with fault-tolerance configuration.
- **totalNbBodies** is the total number of bodies, default is 4 bodies.
- **maxIter** is the maximum number of iterations, default is 10,000 iterations.

Right after starting the application, users have to choose one algorithm for computing amongst the following ones:

- Simplest version, one-to-one communication and master.
- Group communication and master.

⁷ <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html>

- Group communication, odd-even-synchronization.
- Group communication, oospmd synchronization.
- Barnes-Hut.

Mouse controls with the 3D GUI:

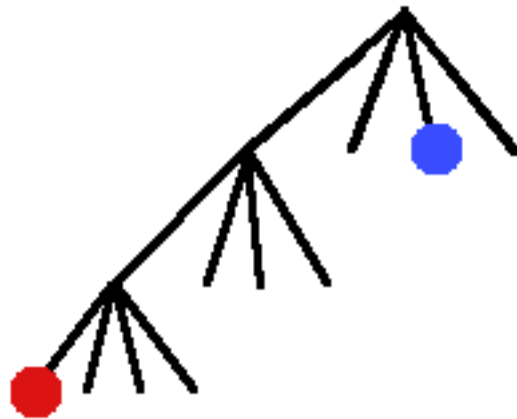
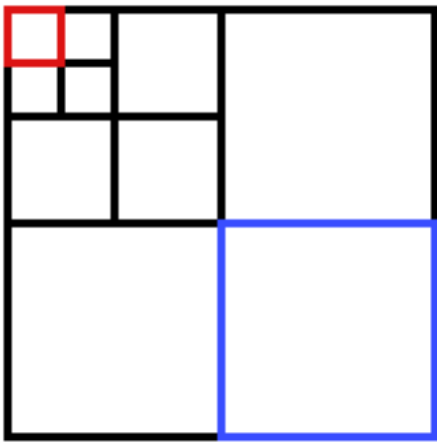
- Left click: rotating.
- Right click: moving the scene.
- Scroll wheel: zoom in/out

5.7.2. Barnes-Hut

This way to construct the nbody simulation is based on a very different algorithm. This is described here to show how one can express this algorithm in ProActive, but having a different approach to solve the problem. Here's how it works:

To avoid broadcasting the new position of every particle to every active object, a tree implementation can simplify the problem by agglomerating sets of particles as a single particle, with a mass equal to the sum of masses of all the particles. This is the core of the Barnes-Hut algorithm. This method allows us to have a complexity brought down to $O(N \log N)$.

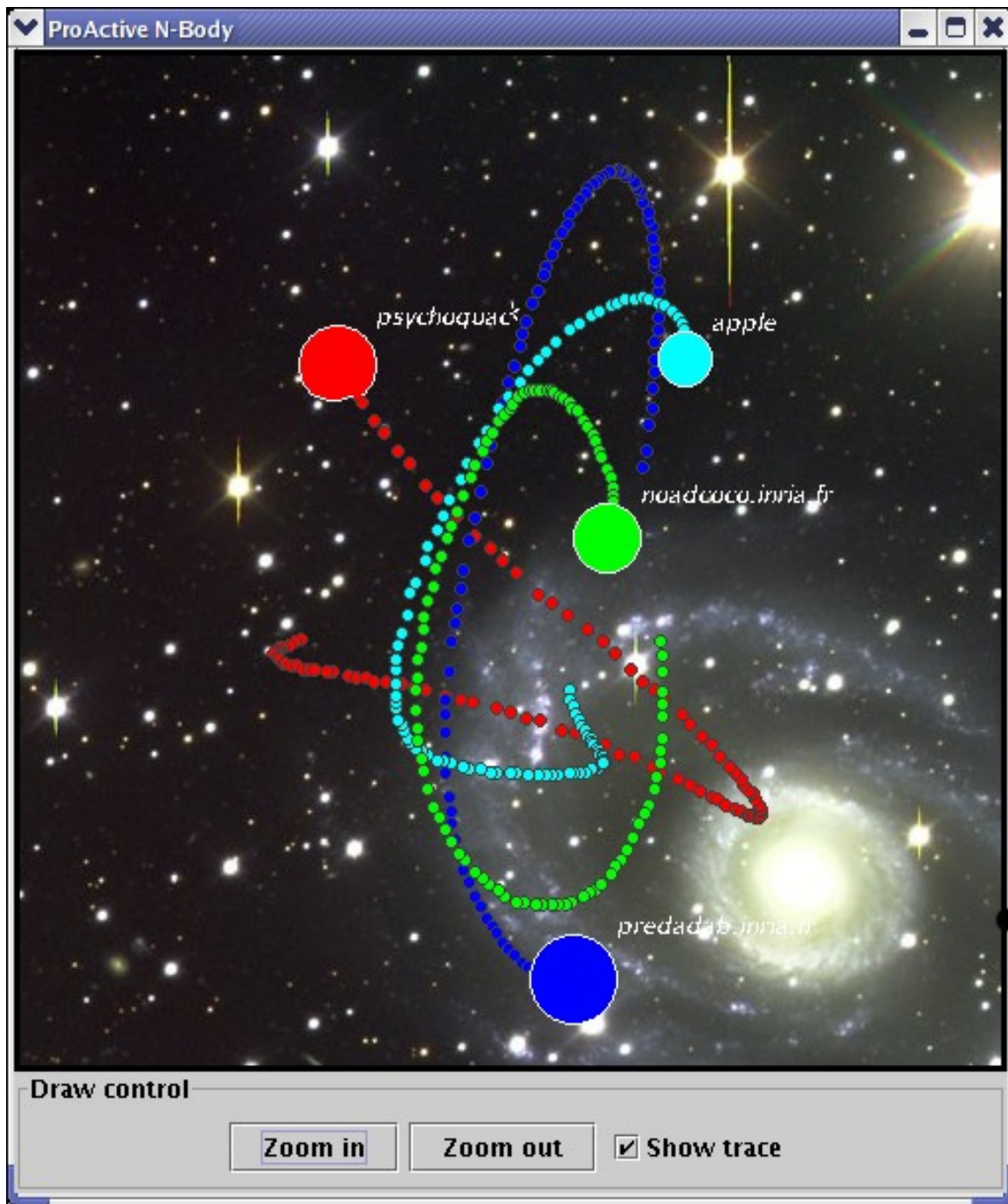
In our parallel implementation, we have defined an **Active Object** called **Domain**, which represents a volume in space, and which contains **Planets**. It is either subdivided into smaller **Domains**, or is a leaf of the total tree, and then only contains **Planets**. A **Planet** is still an **Object** with mass, velocity and position, but is no longer on a one-to-one connection with a **Domain**. We have cut down communications to the biggest **Domains** possible: when a **Planet** is distant enough, its interactions are not computed, but it is grouped with its local neighbours to a bigger particle. Here is an example of the **Domains** which would be known by the **Domain** drawn in red:



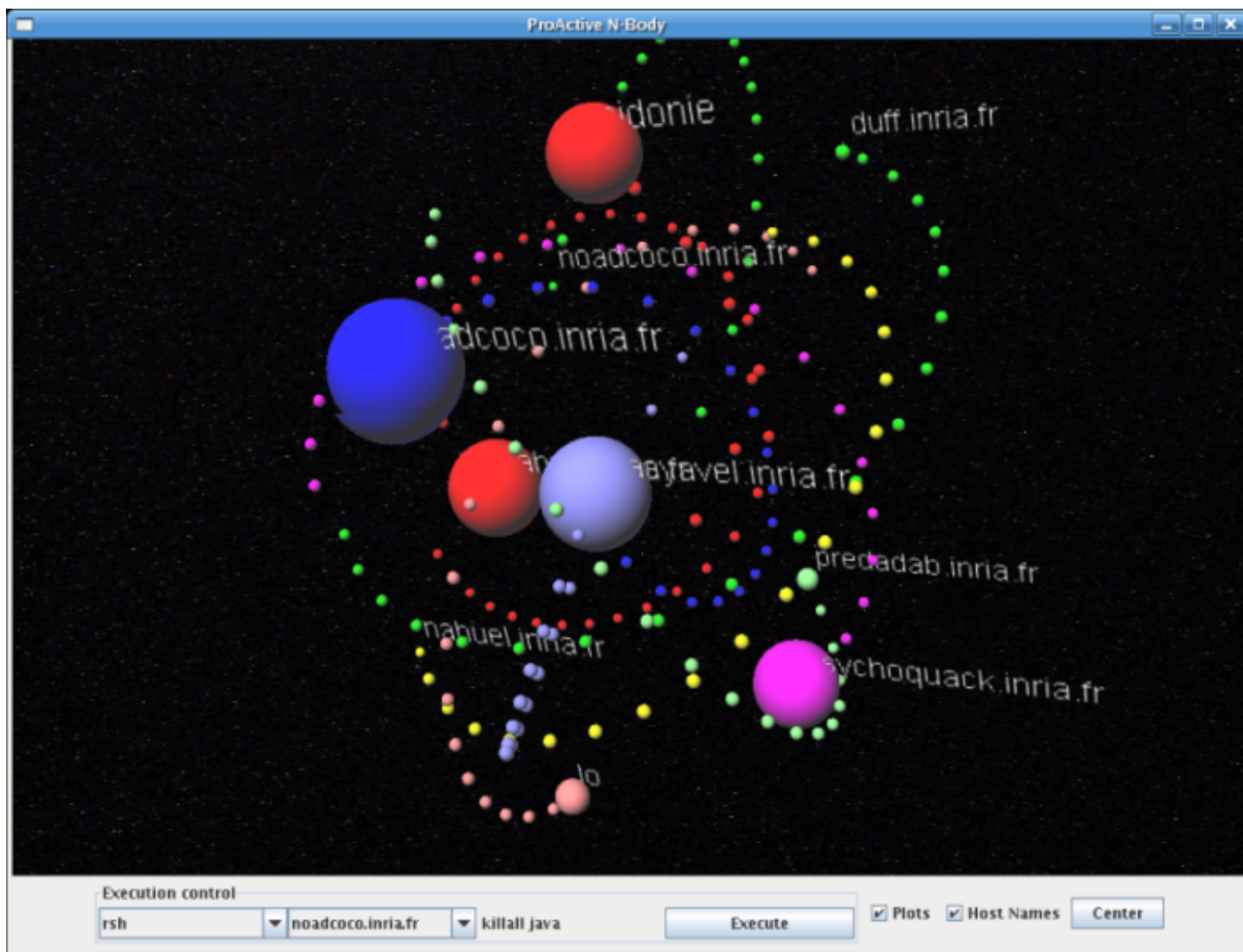
The Domain in the lower right hand-corner, drawn in blue, is also divided into sub-Domains, but that does not need to be known by the Domain in red: it assumes all the particles in the blue Domain are only one big one, centered at the center of mass of all the particles within the blue.

In this version, **Domains** communicate with a reduced set of other **Domains**, spanning on volumes of different sizes. Synchronization is achieved by sending explicitly iteration numbers, and returning when needed older positions. You may notice that some **Domains** seem desynchronized with other ones, having several iterations in-between. That is not a problem because if they need to be synchronized and send each other information, a mechanism saving the older positions permits to send them when needed.

This is a snapshot of the ProActive n-body example running on 3 hosts with 8 bodies:



And here is a n-body screenshot, with the application GUI and Java3D installed:



5.8. Conclusion

These are some examples amongst all the ProActive examples present in the ProActive distribution. To see a full list of examples, please refer to the [application](http://proactive.inria.fr/index.php?page=applications)⁸ web page.

⁸ <http://proactive.inria.fr/index.php?page=applications>

Chapter 6. Active Object Tutorial

In this chapter, we present a step by step introduction to programming with ProActive.

The program that we will develop is a remote computation and monitoring agent. As we progress through the example, we will increase the complexity of the agent by using different features of ProActive.

- In part one, we will code a 'client-server' application, the server being an active object that acts as a monitoring agent.
- In part two, we will see how we can control the activity of an active object.
- In part four, this exercise will explain message synchronization and synchronous and asynchronous method calls using a monitoring agent that makes chained calls.
- In part five, we will add mobility to this active object and make the agent migrate to another computer and report on the status of the JVM on the machine.
- In part six, we will show how to use ProActive groups to monitor several machines at the same time.
- In part seven, we will show how to expose an active object as a web service.
- The next parts aim at showing how we can transform a sequential primality test into a distributed one. In this parts, we will also use the ProActive Master-Worker API.

In order to get the tutorials, that is to say, code sources with missing lines, go to the `compile/` directory and type `build tutorials`. This command will create a new directory in the ProActive home directory, called `tutorials`. This directory will be composed of four directories:

- `src` - source directory. You will find in this directory all sources to fill in.
- `compile` - compilation directory. In this directory, you will be able to compile your code. Type `build` to know all the available targets. Normally, you will see one target per tutorial example. Targets will be described after each example throughout this chapter.
- `scripts` - launch scripts directory. You will find in this directory all scripts to launch your compiled code. Scripts will be described after each example throughout this chapter.
- `dist` - library directory. You will find in this directory all libraries needed to compile your code. Normally, you will not have to deal with this directory.

To further learn what is needed for the running ProActive applications, please refer to [Chapter 2, ProActive Installation](#).

6.1. Simple Computation And Monitoring Agent

Welcome to your first ProActive program! This is one of the simplest applications that can be written using ProActive. We will create an active object locally and get the state of the JVM through it. Our application is composed of three classes with a client-server structure.

The example illustrates the creation of an active object from the `CMAgent` class that will be used by the `Main` class to retrieve the JVM state for a machine and print it to the standard output. The `Main` class corresponds to the client, and is only a container for the `main()` method, while the `CMAgent` class corresponds to the server and its instance is an active object which provides a `getCurrentState()` method as a remote service.

To safely use the `CMAgent` class as an active object, we have to meet three requirements:

- **no direct access to field variables** - If public variables are used, then the stub class generated from the original class may become decoupled from the original class. If a change is affected on the public field variable in the stub instance, the change will not be propagated to the class instance from which the stub was generated. The safe way to change variables is to set them as `private` and access them through public getter/setter methods.
- **provide a no-argument and preferably an empty constructor** - A no-argument constructor is necessary to create the stub class needed for communication. A stub cannot be created if there are only constructors with arguments since the stub is only meant to abstract the communication from the active objects. If there is no constructor defined, the Java compiler will automatically create a no-argument constructor that initializes all instance variables to the default value. However, if there is an already defined constructor with arguments then no default no-argument constructor will be created by the compiler. In that case, the definition of

a no-argument constructor is mandatory for stub creation. The safest way is to always define a no-argument constructor. Also, the constructor should be empty so that on stub creation no initialization is done on the stub.

- **provide remote functionalities as public methods with return types that can be subclassed and are serializable** - Since the stub is created through inheritance, the only methods it can use for communication are the inherited public methods from the superclass. The return types of the methods have to be subclassable and therefore not final. ProActive provides several wrappers for Java types that are final. The example uses the `StringWrapper` class in order to provide a wrapper for the final `String` class. Since ProActive uses a proxy mechanism and the `String` class is final, it is not possible to subclass a `String` and to perform asynchronous calls. ProActive provides several wrappers for final classes: `StringWrapper`, `BooleanWrapper`, `IntWrapper`, `DoubleWrapper` and `FloatWrapper`. These have to be used in replacement of `String`, `Boolean`, `Integer`, `Double` and `Float` in classes which will be active objects. If you do not use wrappers, method calls will be synchronous. In our case, the return type is `State` which is not final.

6.1.1. Classes Used

ProActive

- `org.objectweb.proactive.api.PAActiveObject` - used to create an instance of an active object
- `org.objectweb.proactive.core.node.NodeException` - used to catch the exceptions that the creation of the Node might throw
- `org.objectweb.proactive.ActiveObjectCreationException` - used to catch the exceptions that the creation of the active object might throw

Other

- `java.io.Serializable` - used to make the `State` object serializable so it can be sent across the network
- `java.lang.management.ManagementFactory` - used to get various information about the machine the active object is running on
- `java.net.InetAddress` - used to get the address of the host where the active object is running on
- `java.net.UnknownHostException` - used to catch the exceptions that might be thrown when requesting host information
- `java.util.Date` - used to get the time for the requested state

6.1.2. CMA Architecture and Skeleton Code

For our Monitoring agent, we use three classes. The first two classes are regular Java objects. The first class is `State` which we use to get some information on the JVM the object is located on. This object will be use as return value for the `getCurrentState()` method in the `CMAgent` class.

```
package org.objectweb.proactive.examples.userguide.cmagent.simple;

import java.io.Serializable;
import java.lang.management.ManagementFactory;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Date;

//TODO 4. remove Serializable and run the agent
public class State implements Serializable {
    private long committedMemory = ManagementFactory.getMemoryMXBean().getHeapMemoryUsage().getCommitted();
    private long initMemory = ManagementFactory.getMemoryMXBean().getHeapMemoryUsage().getInit();
    private long maxMemory = ManagementFactory.getMemoryMXBean().getHeapMemoryUsage().getMax();
    private long usedMemory = ManagementFactory.getMemoryMXBean().getHeapMemoryUsage().getUsed();
    private String osArch = ManagementFactory.getOperatingSystemMXBean().getArch();
    private String osName = ManagementFactory.getOperatingSystemMXBean().getName();
    private String osVersion = ManagementFactory.getOperatingSystemMXBean().getVersion();
    private int osProcs = ManagementFactory.getOperatingSystemMXBean().getAvailableProcessors();
}
```

```

private int liveThreads = ManagementFactory.getThreadMXBean().getThreadCount();
private long startedThreads = ManagementFactory.getThreadMXBean().getTotalStartedThreadCount();
private int peakThreads = ManagementFactory.getThreadMXBean().getPeakThreadCount();
private int daemonThreads = ManagementFactory.getThreadMXBean().getDaemonThreadCount();
private Date timePoint = new Date();
private String hostname;
{
    try {
        hostname = InetAddress.getLocalHost().toString();
    } catch (UnknownHostException e) {
        e.printStackTrace();
    }
}

public State() {
}

public String toString() {

    return new String("\n===== [" + "State at " + timePoint + " on " + hostname + "] =====" +
        "\nCommitted memory: " + committedMemory + " bytes\nInitial memory requested: " + initMemory +
        " bytes\nMaximum memory available: " + maxMemory + " bytes\nUsed memory: " + usedMemory +
        " bytes\nOperating System: " + osName + " " + osVersion + " " + osArch + "\nProcessors: " +
        osProcs + "\nCurrent live threads: " + liveThreads + "\nTotal started threads: " +
        startedThreads + "\nPeak number of live threads: " + peakThreads + "\nCurrent daemon threads: " +
        daemonThreads +
        "\n=====");
}
}

```

The Monitoring agent class is a regular Java class with only one method:

```

package org.objectweb.proactive.examples.userguide.cmagent.simple;

import org.objectweb.proactive.extensions.annotation.ActiveObject;

@ActiveObject
public class CMAgent {
    // empty constructor is required by Proactive
    public CMAgent() {
    }

    public State getCurrentState() {
        return new State();
    }
}

```

For this simple exercise, we only need to add ProActive code in the Main class where we instantiate the active object.

```

package org.objectweb.proactive.examples.userguide.cmagent.simple;

import org.objectweb.proactive.ActiveObjectCreationException;

```

```

import org.objectweb.proactive.api.PAActiveObject;
import org.objectweb.proactive.api.PALifeCycle;
import org.objectweb.proactive.core.node.NodeException;

public class Main {
    public static void main(String args[]) {
        try {
            String currentState = new String();
            //TODO 1. Create the active object
            //TODO 2. Get the current state
            //TODO 3. Print the state
            //TODO 4. Stop the active object and
            //      terminate the application
        } catch (NodeException nodeExcep) {
            System.err.println(nodeExcep.getMessage());
        } catch (ActiveObjectCreationException aoExcep) {
            System.err.println(aoExcep.getMessage());
        }
    }
}

```

6.1.3. Proposed Work

You can find the skeleton codes for this exercise into the `tutorials/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/simple/` directory. Full code is also available into the directory `ProActive/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/simple/`.

1. Create an active object using the `org.objectweb.proactive.api.PAActiveObject.newActive(...)` static method.
2. Call the `getCurrentState()` method and display the results.
3. Since the active object has a never ending thread we need to make an explicit call in order to stop it. Use `PAActiveObject.terminateActiveObject(...)` to stop the active object and use the `PALifeCycle.exitSuccess()` method to terminate the application.
4. Remove the `Serializable` from the class `State` and explain the results when running the example.

Once filled in, go to the `tutorials/compile` directory and type `build[.bat] cmagent.simple` to compile your code. Then, after a successful compilation, go to the `scripts/CMAgent` directory and launch the `simpleCMA.[sh|bat]` script to execute your code.

6.1.4. Solutions and Full Code

1. We will now show how to create the server object. For now, we want the `CMAgent` active object to be created on the current Node (we will see later how to distribute the program). To create an instance of a remotely accessible object, we must use the `PAActiveObject.newActive(...)` static method. We pass as an argument the name of the class to be instantiated and arguments for the constructor of the class. In our case, `CMAgent` does not need any arguments for the constructor and therefore we use `null`.

```

//TODO 1. Create the active object
CMAgent ao = (CMAgent) PAActiveObject.newActive(CMAgent.class.getName(), null);

```

2. Invoking a method on a remote active object is transparent and is similar to invoking a method on a local object of the same type. The user does not have to deal with catching exceptions related to the remote communication. The only modification brought to the code by ProActive is during the active objects creation. All the rest of the code can remain unmodified, fostering software reuse.

To invoke the `getCurrentState()` method and display the result, we execute:

```

//TODO 2. Get the current state
currentState = ao.getCurrentState().toString();

```



```
//TODO 3. Print the state
```

```
System.out.println(currentState);
```

3. To stop the active object, we call the `PAActiveObject.terminateActiveObject(ao, false)` method. The `false` argument is used in order to call the terminate method as a regular request. If the argument was `true`, the method call would be served as an immediate service (executed as soon as the current executing request is completed regardless of how many other requests might be waiting) and synchronously (the caller thread blocks until it receives the results).

```
//TODO 4. Stop the active object and
```

```
// terminate the application
```

```
PAActiveObject.terminateActiveObject(ao, true);
```

```
PALifeCycle.exitSuccess();
```

4. Passive objects in ProActive are always passed by deep copy when returned as a method results. If the object `State` does not implement the `Serializable` interface, ProActive will not be able to make a deep copy of the object and will throw an exception.

The full code of the `Main` class is the following:

```
package org.objectweb.proactive.examples.userguide.cmagent.simple;

import org.objectweb.proactive.ActiveObjectCreationException;
import org.objectweb.proactive.api.PAActiveObject;
import org.objectweb.proactive.api.PALifeCycle;
import org.objectweb.proactive.core.node.NodeException;

public class Main {
    public static void main(String args[]) {
        try {
            String currentState = new String();
            //TODO 1. Create the active object
            CMAgent ao = (CMAgent) PAActiveObject.newActive(CMAgent.class.getName(), null);
            //TODO 2. Get the current state
            currentState = ao.getCurrentState().toString();
            //TODO 3. Print the state
            System.out.println(currentState);
            //TODO 4. Stop the active object and
            // terminate the application
            PAActiveObject.terminateActiveObject(ao, true);
            PALifeCycle.exitSuccess();
        } catch (NodeException nodeExcep) {
            System.err.println(nodeExcep.getMessage());
        } catch (ActiveObjectCreationException aoExcep) {
            System.err.println(aoExcep.getMessage());
        }
    }
}
```

6.2. Active Objects Lifecycle: Using `InitActive`, `RunActive` and `EndActive`

Active objects, as their name indicates, have an activity of their own (an internal thread). By default, the active object steps through the constructor, the `initActivity`, the `runActivity`, and when the `terminate` method is called on the `Body` of the active object through the `endActivity` method. It is possible to control the initialization, running, and ending phase of this thread by implementing three interfaces: `InitActive`, `RunActive`, and `EndActive`. These interfaces define respectively the `initActivity`, `runActivity` and `endActivity` methods. One of the reasons for using `initActivity` method is the presence of the empty constructor in an active object. The `initActivity` method is automatically called on the creation of an active object in order to set up the object without using the constructor. The `runActivity`

method allows the user to control the active object request queue. By implementing the `EndActive` interface, it is also possible to clean up before the active object thread is stopped.

The following example will help you to understand how and when you can initialize and clean the activity. The example will implement the `InitActive`, `RunActive`, and `EndActive` interfaces. However `RunActive` has a more complex structure than it is presented here. To understand how to use `EndActive` `RunActive` interface, please refer to [Active Objects: Creation And Advanced Concepts](#)¹.

6.2.1. Classes Used

New classes used

- `org.objectweb.proactive.Body` - used to access the body of the active object
- `org.objectweb.proactive.Service` - used to access the queue of the active object
- `org.objectweb.proactive.InitActive` - used to define the `initActivity(Body body)` method, which is run at active object initialization
- `org.objectweb.proactive.EndActive` - used to define the `endActivity(Body body)` method, which is run at active object destruction
- `org.objectweb.proactive.RunActive` - used to define the `runActivity(Body body)` method, which manages the queue of requests
- `org.objectweb.proactive.core.util.wrapper.LongWrapper` - used to wrap the `Long` return type

Previously used classes

- `org.objectweb.proactive.api.PAActiveObject` - used to create an instance of an active object
- `org.objectweb.proactive.core.node.NodeException` - used to catch the exceptions that the creation of the Node might throw
- `org.objectweb.proactive.ActiveObjectCreationException` - used to catch the exceptions that the creation of the active object might throw
- `java.io.Serializable` - used to make the `State` object serializable so it can be sent across the network
- `java.lang.management.ManagementFactory` - used to get various information about the machine the active object is running on
- `java.net.InetAddress` - used to get the address of host where the active object is running on
- `java.net.UnknownHostException` - used to catch the exceptions that might be thrown when requesting host information
- `java.util.Date` - used to get the time for the requested state

6.2.2. Initialized CMA Architecture and Skeleton Code

The `CMAgentInitialized` class extends the `CMAgent` class from the previous example, and implements the interfaces `InitActive` and `EndActive`. It acts as a server for the `Main` class.

To implement the application we will create a class that inherits from the `CMAgent` class and implements the `InitActive`, `RunActive`, and `EndActive` interfaces.

```
package org.objectweb.proactive.examples.userguide.cmagent.initialized;

import org.objectweb.proactive.Body;
import org.objectweb.proactive.EndActive;
import org.objectweb.proactive.InitActive;
import org.objectweb.proactive.RunActive;
import org.objectweb.proactive.Service;
import org.objectweb.proactive.core.util.wrapper.LongWrapper;
import org.objectweb.proactive.examples.userguide.cmagent.simple.CMAgent;
import org.objectweb.proactive.extensions.annotation.ActiveObject;
```

¹ file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/./../ReferenceManual/multiple_html/ActiveObjectCreation.html

```

@ActiveObject
public class CMAgentInitialized extends CMAgent implements InitActive, RunActive, EndActive {
    private long lastRequestDuration;
    private long startTime;
    private long requestsServed = 0;

    public void initActivity(Body body) {
        //TODO 1. Print start information

        //TODO 2. Record start time
    }

    public void runActivity(Body body) {
        Service service = new Service(body);
        long currentRequestDuration = 0;
        while (body.isActive()) {
            //TODO 3. wait for a request

            //TODO 4. Record time

            //TODO 5. Serve request

            //TODO 6. Calculate request duration

            //TODO 7. Increment the number of requests served
        }
    }

    public void endActivity(Body body) {
        //TODO 8. Calculate the running time of the active object using the start time recorded in initActivity()

        //TODO 9. Print various stop information
    }

    public LongWrapper getLastRequestServeTime() {
        //TODO 10. Use wrappers for primitive types so the calls are asynchronous
    }
}

```

By default an active object has a never-ending thread that should be stopped when the object is not needed anymore. The method `terminate()` serves the purpose of destroying the object. However, if an explicit call to terminate the object is not made, ProActive has its own distributed garbage collection system that is able to decide when an active object can be destroyed.

The Main is similar to the one in the previous example. We will change the object created from `CMAAgent` to `CMAAgentInitialized` and also we will call the `terminate()` method to destroy the object.

6.2.3. Proposed Work

You can find the skeleton codes for this exercise into the `tutorials/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/initialized/` directory. Full code is also available into the directory `ProActive/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/initialized/`.

1. Use `initActivity(Body body)` to print information about the start location of the active object and record the start time.

2. Use `endActivity(Body body)` to print information about the stop location of the active object, calculate the running time, and print the number of requests served.
3. Calculate the last request duration and count the requests using `org.objectweb.proactive.Service.waitForRequest()` and `org.objectweb.proactive.Service.serveOldest()`
4. Use `org.objectweb.proactive.core.util.wrapper.LongWrapper` to return a wrapped Long value.

Once filled in, go to the `tutorials/compile` directory and type `build[.bat] cmagent.initialized` to compile your code. Then, after a successful compilation, go to the `scripts/CMAgent` directory and launch the `initializedCMA.[sh|bat]` script to execute your code.

6.2.4. Solution and Full Code

We only need to extend the `CMAgent` class and implement the interfaces. In `initActivity(Body body)`, we use `body.getName()` to get the name of the active object and `body.getNodeUrl()` to get the location. In `endActivity(Body body)`, we also use `body.getNodeUrl()` to get the location.

```
public void initActivity(Body body) {
    //TODO 1. Print start information
    System.out.println("### Started Active object " + body.getMBean().getName() + " on " +
        body.getMBean().getNodeUrl());

    //TODO 2. Record start time
    startTime = System.currentTimeMillis();
}

public void runActivity(Body body) {
    Service service = new Service(body);
    long currentRequestDuration = 0;
    while (body.isActive()) {
        //TODO 3. wait for a request
        service.waitForRequest(); // block until a request is received

        //TODO 4. Record time
        currentRequestDuration = System.currentTimeMillis();

        //TODO 5. Serve request
        service.serveOldest(); //server the requests in a FIFO manner

        //TODO 6. Calculate request duration
        currentRequestDuration = System.currentTimeMillis() - currentRequestDuration;

        // an intermediary variable is used so
        // when calling getLastRequestServeTime()
        // we get the first value before the last request
        // i.e when calling getLastRequestServeTime
        // the lastRequestDuration is update with the
        // value of the getLastRequestServeTime call
        // AFTER the previous calculated value has been returned
        lastRequestDuration = currentRequestDuration;

        //TODO 7. Increment the number of requests served
        requestsServed++;
    }
}
```

```

public void endActivity(Body body) {
    //TODO 8. Calculate the running time of the active object using the start time recorded in initActivity()
    long runningTime = System.currentTimeMillis() - startTime;

    //TODO 9. Print various stop information
    System.out.println("### You have killed the active object. The final " + " resting place is on " +
        body.getNodeURL() + "\n### It has faithfully served " + requestsServed + " requests " +
        "and has been an upstanding active object for " + runningTime + " ms ");
}

public LongWrapper getLastRequestServeTime() {
    //TODO 10. Use wrappers for primitive types so the calls are asynchronous
    return new LongWrapper(lastRequestDuration);
}

```

6.3. Remote Monitoring Agent

In this part of the tutorial, we will see how to start a monitoring agent using the deployment methods. In the previous example, the application were deployed inside the same JVM. This section will focus on showing how to deploy the application on different nodes using deployment descriptors. To be able to deploy on remote machines, we just have to use the deployment file, add a method that tells ProActive to activate the nodes used and tell the active object to start on the remote node. This section does not aim at explaining how to write deployment files but rather at describing how to use them. If you want more information on GCM deployment, please refer to [ProActive Grid Component Model Deployment](#)² Normally, only GCM concepts as well as the provided examples will be useful to successful achieve this tutorial. Anyway, launch scripts use a deployment descriptor which will use local JVMs.

6.3.1. Classes Used

New classes

- org.objectweb.proactive.extensions.gcmdeployment.PAGCMDeployment - used to create a GCMApplication from an Application Descriptor
- org.objectweb.proactive.gcmdeployment.GCMApplication - represents the application which is being deployed
- org.objectweb.proactive.core.ProActiveException - used to catch exception
- org.objectweb.proactive.gcmdeployment.GCMVirtualNode - used to control and instantiate virtual node objects

Previously used classes

- org.objectweb.proactive.Body - used to access the body of the active object
- org.objectweb.proactive.PALifeCycle - controls the lifecycle of the ProActive application
- org.objectweb.proactive.Service - used to access the queue of the active object
- org.objectweb.proactive.InitActive - used for defining the initActivity(Body body) method, which is run at active object initialization
- org.objectweb.proactive.EndActive - used for defining the endActivity(Body body) method, which is run at active object destruction
- org.objectweb.proactive.RunActive - used for defining the runActivity(Body body) method, which manages the queue of requests
- org.objectweb.proactive.core.util.wrapper.LongWrapper - used to wrap the Long return type
- org.objectweb.proactive.api.PAActiveObject - used to create an instance of an active object
- org.objectweb.proactive.core.node.NodeException - used to catch the exceptions that the creation of the Node might throw

² file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/./../ReferenceManual/multiple_html/GCMDeployment.html

- `org.objectweb.proactive.ActiveObjectCreationException` - used to catch the exceptions that the creation of the active object might throw
- `java.io.Serializable` - used to make the `State` object serializable so it can be sent across the network
- `java.lang.management.ManagementFactory` - used to get various information about the machine the active object is running on
- `java.net.InetAddress` - used to get the address of the host where the active object is running on
- `java.net.UnknownHostException` - used to catch the exceptions that might be thrown when requesting host information
- `java.util.Date` - used to get the time for the requested state

6.3.2. Deployed CMA Architecture and Skeleton Code

We will change the `Main` class to declare and load the deployment descriptors to be used. For this, we will use a `deploy()` method that returns a `Virtual Node` which has several nodes (as specified in the deployment file) that we can deploy on. First, the method creates an object representation of the deployment file, then activates all the nodes, and then returns the first available node. We also have to change the deployment descriptor files to fit our local settings.

```
package org.objectweb.proactive.examples.userguide.cmagent.deployed;

import java.io.File;

import org.objectweb.proactive.ActiveObjectCreationException;
import org.objectweb.proactive.api.PAActiveObject;
import org.objectweb.proactive.api.PALifeCycle;
import org.objectweb.proactive.core.ProActiveException;
import org.objectweb.proactive.core.node.NodeException;
import org.objectweb.proactive.examples.userguide.cmagent.initialized.CMAgentInitialized;
import org.objectweb.proactive.extensions.gcmdeployment.PAGCMDeployment;
import org.objectweb.proactive.gcmdeployment.GCMApplication;
import org.objectweb.proactive.gcmdeployment.GCMVirtualNode;

public class Main {
    private static GCMApplication pad;

    //deployment method
    private static GCMVirtualNode deploy(String descriptor) throws NodeException, ProActiveException {
        //TODO 1. Create object representation of the deployment file
        //TODO 2. Activate all Virtual Nodes
        //TODO 3. Wait for all the virtual nodes to become ready
        //TODO 4. Get the first Virtual Node specified in the descriptor file
        //TODO 5. Return the virtual node
    }

    public static void main(String args[]) {
        try {
            //TODO 6. Get the virtual node through the deploy method
            //TODO 7. Create the active object using a node on the virtual node
            //TODO 8. Get the current state from the active object
            //TODO 9. Print the state
            //TODO 10. Stop the active object
        } catch (NodeException nodeExcep) {
            System.err.println(nodeExcep.getMessage());
        }
    }
}
```



```

    } catch (ActiveObjectCreationException aoExcep) {
        System.err.println(aoExcep.getMessage());
    } catch (ProActiveException poExcep) {
        System.err.println(poExcep.getMessage());
    } finally {
        //TODO 11. Stop the virtual node
    }
}
}

```

6.3.3. Proposed Work

You can find the skeleton codes for this exercise into the `tutorials/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/deployed/` directory. Full code is also available into the directory `ProActive/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/deployed/`.

1. Create `GCMApplication` from the application descriptor file using `PAGCMDeployment.loadApplicationDescriptor(String descriptor)`
2. Start the deployment of all the virtual nodes using `GCMApplication.startDeployment()`
3. Wait for all virtual nodes to get ready using `GCMApplication.waitReady()`
4. Get a virtual node using `GCMApplication.getVirtualNodes()`. You can also use `GCMApplication.getVirtualNode(String virtualNodeName)`. Still, this latter method requires to know the name of the virtual node and so, it forces us to give the same virtual node id into every application descriptor we want to pass as argument.
5. Return this virtual node
6. Retrieve a virtual node using the previous `deploy()` method.
7. Create the active object using a node from the virtual node
8. Get the state from the active object
9. Print this state
10. Stop the active object
11. Stop the virtual node using `GCMApplication.kill()`
12. Change the `State` class so that the initialization of variables takes place in the `toString()` method. Run the deployed application again and explain the different results. Hint: try to deploy the application on a remote host.

Once filled in, go to the `tutorials/compile` directory and type `build[.bat]` `cmagent.deployed` to compile your code. Then, after a successful compilation, go to the `scripts/CMAgent` directory and launch the `deployedCMA.[sh|bat]` script to execute your code.

6.3.4. Solution and Full Code

In the `deploy()` method, we return the first virtual node found in the deployment descriptor:

```

//deployment method
private static GCMVirtualNode deploy(String descriptor) throws NodeException, ProActiveException {
    //TODO 1. Create object representation of the deployment file
    pad = PAGCMDeployment.loadApplicationDescriptor(new File(descriptor));
    //TODO 2. Activate all Virtual Nodes
    pad.startDeployment();
    //TODO 3. Wait for all the virtual nodes to become ready
    pad.waitReady();
    //TODO 4. Get the first Virtual Node specified in the descriptor file
    GCMVirtualNode vn = pad.getVirtualNodes().values().iterator().next();
    //TODO 5. Return the virtual node
    return vn;
}

```

The active object is created by using the first node on the virtual node:

//TODO 7. Create the active object using a node on the virtual node

```
CMAgentInitialized ao = (CMAgentInitialized) PAActiveObject.newActive(CMAgentInitialized.class
    .getName(), new Object[] {}, vn.getANode());
```

The full Main class:

```
package org.objectweb.proactive.examples.userguide.cmagent.deployed;

import java.io.File;

import org.objectweb.proactive.ActiveObjectCreationException;
import org.objectweb.proactive.api.PAActiveObject;
import org.objectweb.proactive.api.PALifeCycle;
import org.objectweb.proactive.core.ProActiveException;
import org.objectweb.proactive.core.node.NodeException;
import org.objectweb.proactive.examples.userguide.cmagent.initialized.CMAgentInitialized;
import org.objectweb.proactive.extensions.gcmdeployment.PAGCMDeployment;
import org.objectweb.proactive.gcmdeployment.GCMApplication;
import org.objectweb.proactive.gcmdeployment.GCMVirtualNode;

public class Main {
    private static GCMApplication pad;

    //deployment method
    private static GCMVirtualNode deploy(String descriptor) throws NodeException, ProActiveException {
        //TODO 1. Create object representation of the deployment file
        pad = PAGCMDeployment.loadApplicationDescriptor(new File(descriptor));
        //TODO 2. Activate all Virtual Nodes
        pad.startDeployment();
        //TODO 3. Wait for all the virtual nodes to become ready
        pad.waitReady();
        //TODO 4. Get the first Virtual Node specified in the descriptor file
        GCMVirtualNode vn = pad.getVirtualNodes().values().iterator().next();
        //TODO 5. Return the virtual node
        return vn;
    }

    public static void main(String args[]) {
        try {
            //TODO 6. Get the virtual node through the deploy method
            GCMVirtualNode vn = deploy(args[0]);
            //TODO 7. Create the active object using a node on the virtual node
            CMAgentInitialized ao = (CMAgentInitialized) PAActiveObject.newActive(CMAgentInitialized.class
                .getName(), new Object[] {}, vn.getANode());
            //TODO 8. Get the current state from the active object
            String currentState = ao.getCurrentState().toString();
            //TODO 9. Print the state
            System.out.println(currentState);
            //TODO 10. Stop the active object
```



```

    PAActiveObject.terminateActiveObject(ao, false);

} catch (NodeException nodeExcep) {
    System.err.println(nodeExcep.getMessage());
} catch (ActiveObjectCreationException aoExcep) {
    System.err.println(aoExcep.getMessage());
} catch (ProActiveException poExcep) {
    System.err.println(poExcep.getMessage());
} finally {
    //TODO 11. Stop the virtual node
    if (pad != null)
        pad.kill();
    PALifeCycle.exitSuccess();
}
}
}

```

In order to answer to our last question, we follow the given hint. To do that, the simplest way is to remove the local host from the host list in the nodeProvider tag. In our case, the local host is kisscool. Try to do same thing and launch the application with the two different variable initializations.



Note

the remote machine needs to have the rsh service running since we are using rsh deployment.

If you do not change the State class, you will get a display looking like the following one:

```

===== [State at Wed May 27 14:45:05 CEST 2009 on paquito.inria.fr/138.96.218.106] =====
Committed memory: 63111168 bytes
Initial memory requested: 63961152 bytes
Maximum memory available: 985530368 bytes
Used memory: 7939936 bytes
Operating System: Linux 2.6.23.1-21.fc7 i386
Processors: 2
Current live threads: 27
Total started threads: 27
Peak number of live threads: 27
Current daemon threads: 21
=====

```

Whereas with the variable initializations done in the toString() method, you will get something like this:

```

===== [State at Wed May 27 14:46:14 CEST 2009 on kisscool/138.96.218.126] =====
Committed memory: 161021952 bytes
Initial memory requested: 130110784 bytes
Maximum memory available: 1034027008 bytes
Used memory: 45900640 bytes
Operating System: Linux 2.6.27.9-159.fc10.x86_64 i386
Processors: 4
Current live threads: 41
Total started threads: 41

```

```
Peak number of live threads: 41
```

```
Current daemon threads: 37
```

```
=====
```

You can therefore realise that information returned in the second display are information about the local host but not about the host where the active object was running on. This is totally normal since when you call the `toString()` method, you will implicitly get the active object from the remote host and call this method next. So, if variable initializations are done in the `toString()` method, returned information will be information on the local host, whereas if variable initializations are done at the active object creation, returned information will be information on the host where the active object has been created that is to say, the remote host.

6.4. Agent synchronization

In this example, we will use a chained call between agents deployed on several node in order to retrieve information about the nodes. We have several agents that know their previous and next neighbour. When an agent receives a request for a state returns its state and asks the its respective neighbour for the state. If it doesn't have a neighbour then it just returns its state. We will use this example to show how `PAActiveObject.getStubOnThis()` method call is employed. Since an active object is actually a composite of two objects, the method call `PAActiveObject.getStubOnThis()` is the equivalent of `this` in a regular java object. If we used `this`, we would get a reference to the passive object and not to the stub of the active object.

6.4.1. Classes Used

There are no new classes used in this exercise. Instead we will show how to use the `PAActiveObject.getStubOnThis()` method.

Previously used classes

- `org.objectweb.proactive.extensions.gcmdeployment.PAGCMDDeployment` - used to create a `GCMApplication` from an Application Descriptor
- `org.objectweb.proactive.core.ProActiveException` - used to catch exception
- `org.objectweb.proactive.gcmdeployment.GCMApplication` - represents the application which is being deployed
- `org.objectweb.proactive.gcmdeployment.GCMVirtualNode` - used to control and instantiate virtual node objects
- `org.objectweb.proactive.core.node.Node` - used to instantiate and control Node objects
- `org.objectweb.proactive.Body` - used to access the body of the active object
- `org.objectweb.proactive.PALifeCycle` - controls the lifecycle of the ProActive application
- `org.objectweb.proactive.Service` - used to access the queue of the active object
- `org.objectweb.proactive.InitActive` - used to define the `initActivity(Body body)` method, which is run at active object initialization
- `org.objectweb.proactive.EndActive` - used to define the `endActivity(Body body)` method, which is run at active object destruction
- `org.objectweb.proactive.RunActive` - used to define the `runActivity(Body body)` method, which manages the queue of requests
- `org.objectweb.proactive.core.util.wrapper.LongWrapper` - used to wrap the `Long` return type
- `org.objectweb.proactive.api.PAActiveObject` - used to create an instance of an active object
- `org.objectweb.proactive.core.node.NodeException` - used to catch the exceptions that the creation of the Node might throw
- `org.objectweb.proactive.ActiveObjectCreationException` - used to catch the exceptions that the creation of the active object might throw
- `java.io.Serializable` - used to make the `State` object serializable so it can be sent across the network
- `java.lang.management.ManagementFactory` - used to get various information about the machine the active object is running on
- `java.net.InetAddress` - used to get the address of the active object is running on
- `java.net.UnknownHostException` - used to catch the exceptions that might be thrown when requesting host information
- `java.util.Date` - used to get the time for the requested state

6.4.2. Architecture and Skeleton Code

There are no new concepts present in the Main class. We just create the active objects and make the method calls on one of the active objects.

```
public static void main(String args[]) {
    try {
        GCMVirtualNode vn = deploy(args[0]);
        Vector<CMAgentChained> agents = new Vector<CMAgentChained>();
        //create the active objects
        //create a collection of active objects
        for (Node node : vn.getCurrentNodes()) {
            CMAgentChained ao = (CMAgentChained) PAActiveObject.newActive(CMAgentChained.class.getName(),
                null, node);
            agents.add(ao);

            //connect to the neighbour
            int size = agents.size();
            if (size > 1) {
                CMAgentChained lastAgent = agents.get(size - 1);
                CMAgentChained previousAgent = agents.get(size - 2);
                lastAgent.setPreviousNeighbour(previousAgent);
            }
        }
        //start chained call
        Vector<State> states = agents.get(agents.size() / 2).getAllPreviousStates();
        for (State s : states) {
            System.out.println(s.toString());
        }

        states = agents.get(agents.size() / 2).getAllNextStates();
        for (State s : states) {
            System.out.println(s.toString());
        }
    } catch (ActiveObjectCreationException e) {
        System.err.println(e.getMessage());
    } catch (NodeException nodeExcep) {
        System.err.println(nodeExcep.getMessage());
    } catch (ProActiveException e) {
        System.err.println(e.getMessage());
    } finally {
        //stopping all the objects and JVMs
        if (pad != null)
            pad.kill();
        PALifeCycle.exitSuccess();
    }
}
```

The agent class inherits from CMAgentInitialized:

```
package org.objectweb.proactive.examples.userguide.cmagent.synch;

import java.io.Serializable;
import java.util.Vector;
```

```

import org.objectweb.proactive.api.PAActiveObject;
import org.objectweb.proactive.examples.userguide.cmagent.initialized.CMAgentInitialized;
import org.objectweb.proactive.examples.userguide.cmagent.simple.State;
import org.objectweb.proactive.extensions.annotation.ActiveObject;

@ActiveObject
public class CMAgentChained extends CMAgentInitialized implements Serializable {
    private CMAgentChained previousNeighbour;
    private CMAgentChained nextNeighbour;

    public void setPreviousNeighbour(CMAgentChained neighbour) {
        this.previousNeighbour = neighbour;
        //TODO 1. Pass a remote reference of this object to the neighbour
        // Hint: This object is "nextNeighbour" for previous neighbour if not null
    }

    public void setNextNeighbour(CMAgentChained neighbour) {
        this.nextNeighbour = neighbour;
        //TODO 2. Pass a remote reference of this object to the neighbour
        // Hint: This object is "previousNeighbour" for next neighbour if not null
    }

    public CMAgentChained getPreviousNeighbour() {
        return previousNeighbour;
    }

    public CMAgentChained getNextNeighbour() {
        return nextNeighbour;
    }

    public Vector<State> getAllPreviousStates() {
        System.out.println(PAActiveObject.getStubOnThis());

        if (this.previousNeighbour != null) {
            System.out.println("Passing the call to the previous neighbour...");
            // wait-by-necessity
            Vector<State> states = this.previousNeighbour.getAllPreviousStates();
            // states is a future

            // TODO 3. Is this explicit synchronization mandatory ? (NO the wait was removed)

            return states;
        } else {
            System.out.println("No more previous neighbours..");
            Vector<State> states = new Vector<State>();
            states.add(this.getCurrentState());
            return states;
        }
    }

    public Vector<State> getAllNextStates() {
        System.out.println(PAActiveObject.getStubOnThis());
    }
}

```

```

if (this.nextNeighbour != null) {
    // wait-by-necessity
    System.out.println("Passing the call to the next neighbour..");
    Vector<State> states = this.nextNeighbour.getAllNextStates();
    // states is a future

    // TODO 4. Is this explicit synchronization mandatory ? (NO the wait was removed)

    return states;
} else {
    System.out.println("No more next neighbours");
    Vector<State> states = new Vector<State>();
    states.add(this.getCurrentState());
    return states;
}
}

```

6.4.3. Proposed Work

You can find the skeleton codes for this exercise into the `tutorials/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/synch/` directory. Full code is also available into the directory `ProActive/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/synch/`.

1. Write the code for connecting to the previous neighbour
2. Write the code for connecting to the next neighbour
3. Add and remove explicit synchronization between the agents when retrieving the states

Once filled in, go to the `tutorials/compile` directory and type `build[.bat] cmagent.synch` to compile your code. Then, after a successful compilation, go to the `scripts/CMAgent` directory and launch the `synchCMA.[sh|bat]` script to execute your code.

6.4.4. Solution And Full Code

To pass a remote reference to the active object we are currently in, we use `PAActiveObject.getStubOnThis()`. A simple use of the `this` keyword will return a reference to the passive object components of the active object and not a reference to the stub.

```

public void setPreviousNeighbour(CMAgentChained neighbour) {
    this.previousNeighbour = neighbour;
    //TODO 1. Pass a remote reference of this object to the neighbour
    // Hint: This object is "nextNeighbour" for previous neighbour if not null
    if (neighbour.getNextNeighbour() == null)
        neighbour.setNextNeighbour((CMAgentChained) PAActiveObject.getStubOnThis());
}

public void setNextNeighbour(CMAgentChained neighbour) {
    this.nextNeighbour = neighbour;
    //TODO 2. Pass a remote reference of this object to the neighbour
    // Hint: This object is "previousNeighbour" for next neighbour if not null
    if (neighbour.getPreviousNeighbour() == null)
        neighbour.setPreviousNeighbour((CMAgentChained) PAActiveObject.getStubOnThis());
}

```

The full code for the `CMAgentChained` class is:

```

package org.objectweb.proactive.examples.userguide.cmagent.synch;

```

```

import java.io.Serializable;
import java.util.Vector;

import org.objectweb.proactive.api.PAActiveObject;
import org.objectweb.proactive.examples.userguide.cmagent.initialized.CMAgentInitialized;
import org.objectweb.proactive.examples.userguide.cmagent.simple.State;
import org.objectweb.proactive.extensions.annotation.ActiveObject;

@ActiveObject
public class CMAgentChained extends CMAgentInitialized implements Serializable {
    private CMAgentChained previousNeighbour;
    private CMAgentChained nextNeighbour;

    public void setPreviousNeighbour(CMAgentChained neighbour) {
        this.previousNeighbour = neighbour;
        //TODO 1. Pass a remote reference of this object to the neighbour
        // Hint: This object is "nextNeighbour" for previous neighbour if not null
        if (neighbour.getNextNeighbour() == null)
            neighbour.setNextNeighbour((CMAgentChained) PAActiveObject.getStubOnThis());
    }

    public void setNextNeighbour(CMAgentChained neighbour) {
        this.nextNeighbour = neighbour;
        //TODO 2. Pass a remote reference of this object to the neighbour
        // Hint: This object is "previousNeighbour" for next neighbour if not null
        if (neighbour.getPreviousNeighbour() == null)
            neighbour.setPreviousNeighbour((CMAgentChained) PAActiveObject.getStubOnThis());
    }

    public CMAgentChained getPreviousNeighbour() {
        return previousNeighbour;
    }

    public CMAgentChained getNextNeighbour() {
        return nextNeighbour;
    }

    public Vector<State> getAllPreviousStates() {
        System.out.println(PAActiveObject.getStubOnThis());

        if (this.previousNeighbour != null) {
            System.out.println("Passing the call to the previous neighbour...");
            // wait-by-necessity
            Vector<State> states = this.previousNeighbour.getAllPreviousStates();
            // states is a future

            // TODO 3. Is this explicit synchronization mandatory ? (NO the wait was removed)
            states.add(this.getCurrentState());

            return states;
        } else {

```

```

        System.out.println("No more previous neighbours..");
        Vector<State> states = new Vector<State>();
        states.add(this.getCurrentState());
        return states;
    }
}

public Vector<State> getAllNextStates() {
    System.out.println(PAActiveObject.getStubOnThis());
    if (this.nextNeighbour != null) {
        // wait-by-necessity
        System.out.println("Passing the call to the next neighbour..");
        Vector<State> states = this.nextNeighbour.getAllNextStates();
        // states is a future

        // TODO 4. Is this explicit synchronization mandatory ? (NO the wait was removed)
        states.add(this.getCurrentState());

        return states;
    } else {
        System.out.println("No more next neighbours");
        Vector<State> states = new Vector<State>();
        states.add(this.getCurrentState());
        return states;
    }
}
}

```

In the previous example, we have some parts of the code that may trigger a wait-by-necessity. A wait-by-necessity happens when we try to use a future that has not been updated yet. When a future in this state is used, the thread trying to use the future blocks until the value of the future is updated.

6.5. Monitoring Several Computers Using Migration

Our next example deals with migrating the Monitoring Agent between remote nodes. We will start the monitoring agent on one machine and then move it to other machines and report on the state of each JMV on the machines. To do that, we will need to change the descriptor file to specify the nodes and the machines the nodes are mapped to. We also have to add a method that enables us to tell the active object to migrate.

An active object has to implement the Serializable interface (as it will be transferred through the network) in order to be able to migrate. For more information on the topic of object migration, check [Mobile Agents And Migration](#)³.

6.5.1. Classes Used

New classes

- org.objectweb.proactive.api.PAMobileAgent - used to tell the active object to migrate
- org.objectweb.proactive.core.body.migration.MigrationException - used to catch migration related exceptions

Previously used classes

- org.objectweb.proactive.extensions.gcmdeployment.PAGCMDeployment - used to create a GCMAApplication from an Application Descriptor

³ file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/./doc/built/Programming/GetStarted/pdf/./../ReferenceManual/multiple_html/Migration.html

- `org.objectweb.proactive.core.ProActiveException` - used to catch exception
- `org.objectweb.proactive.gcmdeployment.GCMApplication` - represents the application which is being deployed
- `org.objectweb.proactive.gcmdeployment.GCMVirtualNode` - used to control and instantiate virtual node objects
- `org.objectweb.proactive.core.node.Node` - used to instantiate and control Node objects
- `org.objectweb.proactive.Body` - used to access the body of the active object
- `org.objectweb.proactive.PALifeCycle` - controls the lifecycle of the ProActive application
- `org.objectweb.proactive.Service` - used to access the queue of the active object
- `org.objectweb.proactive.InitActive` - used for defining the `initActivity(Body body)` method, which is run at active object initialization
- `org.objectweb.proactive.EndActive` - used for defining the `endActivity(Body body)` method, which is run at active object destruction
- `org.objectweb.proactive.RunActive` - used for defining the `runActivity(Body body)` method, which manages the queue of requests
- `org.objectweb.proactive.core.util.wrapper.LongWrapper` - used to wrap the `Long` return type
- `org.objectweb.proactive.api.PAActiveObject` - used to create an instance of an active object
- `org.objectweb.proactive.core.node.NodeException` - used to catch the exceptions that the creation of the Node might throw
- `org.objectweb.proactive.ActiveObjectCreationException` - used to catch the exceptions that the creation of the active object might throw
- `java.io.Serializable` - used to make the `State` object serializable so it can be sent across the network
- `java.lang.management.ManagementFactory` - used to get various information about the machine the active object is running on
- `java.net.InetAddress` - used to get the address of the active object is running on
- `java.net.UnknownHostException` - used to catch the exceptions that might be thrown when requesting host information
- `java.util.Date` - used to get the time for the requested state

6.5.2. Design of Migratable Monitoring Agent and Skeleton Code

We create a `CMAgentMigrator` class, that inherits from `CMAgentInitialized`. This class will implement all the non-functional behavior concerning the migration, for which this example is created.

The migration has to be initiated by the active object itself. We will have to write the `migrate` method in the code of `CMAgentMigrator` - i.e. a method that contains an explicit call to the migration primitive.

`CMAgentMigrator` skeleton class:

```
package org.objectweb.proactive.examples.userguide.cmagent.migration;

import java.io.Serializable;

import org.objectweb.proactive.api.PAMobileAgent;
import org.objectweb.proactive.core.ProActiveException;
import org.objectweb.proactive.core.node.Node;
import org.objectweb.proactive.examples.userguide.cmagent.initialized.CMAgentInitialized;
import org.objectweb.proactive.extensions.annotation.ActiveObject;
import org.objectweb.proactive.extensions.annotation.MigrationSignal;

@ActiveObject
public class CMAgentMigrator extends CMAgentInitialized implements Serializable {

    @MigrationSignal
    public void migrateTo(Node whereTo) {
        try {
            //TODO 1. Migrate the active object to the Node received as parameter
        }
    }
}
```



```

    } catch (ProActiveException moveExcep) {
        System.err.println(moveExcep.getMessage());
    }
}
}

```

In the Main class, we only have to change a few lines of code to migrate the agent. As you may notice there is more code for creating the textual menu than for controlling the agents.

```

package org.objectweb.proactive.examples.userguide.cmagent.migration;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;

import org.objectweb.proactive.ActiveObjectCreationException;
import org.objectweb.proactive.api.PAActiveObject;
import org.objectweb.proactive.api.PALifeCycle;
import org.objectweb.proactive.core.ProActiveException;
import org.objectweb.proactive.core.node.Node;
import org.objectweb.proactive.core.node.NodeException;
import org.objectweb.proactive.extensions.gcmdployment.PAGCMDDeployment;
import org.objectweb.proactive.gcmdployment.GCMApplication;
import org.objectweb.proactive.gcmdployment.GCMVirtualNode;

public class Main {
    private static GCMApplication pad;

    private static GCMVirtualNode deploy(String descriptor) throws NodeException, ProActiveException {

        //Create object representation of the deployment file
        pad = PAGCMDDeployment.loadApplicationDescriptor(new File(descriptor));
        //Activate all Virtual Nodes
        pad.startDeployment();
        //Wait for all the virtual nodes to become ready
        pad.waitReady();

        //Get the first Virtual Node specified in the descriptor file
        GCMVirtualNode vn = pad.getVirtualNodes().values().iterator().next();

        return vn;
    }

    public static void main(String args[]) {
        try {
            BufferedReader inputBuffer = new BufferedReader(new InputStreamReader(System.in));
            GCMVirtualNode vn = deploy(args[0]);

            //create the active object
            CMAgentMigrator ao = (CMAgentMigrator) PAActiveObject.newActive(CMAgentMigrator.class.getName(),
                new Object[] {}, vn.getANode());

```

```

int k = 1;
int choice;
while (k != 0) {

    //display the menu with the available nodes
    k = 1;
    for (Node node : vn.getCurrentNodes()) {
        //TODO 2. Add the node URL to the menu
        k++;
    }
    System.out.println("0. Exit");

    Node[] nodeArray = vn.getCurrentNodes().toArray(new Node[] {});

    //select a node
    do {
        System.out.print("Choose a node :- ");
        try {
            // Read an option from keyboard
            choice = Integer.parseInt(inputBuffer.readLine().trim());
        } catch (NumberFormatException noExcep) {
            choice = -1;
        }
    } while (!(choice >= 1 && choice < k || choice == 0));
    if (choice == 0)
        break;

    //TODO 3. Migrate the active object to the selected node: choice-1
    //TODO 4. Get the state and the last request time and print them out
    //TODO 5. Print the execution time of the last request
}
} catch (NodeException nodeExcep) {
    System.err.println(nodeExcep.getMessage());
} catch (ActiveObjectCreationException aoExcep) {
    System.err.println(aoExcep.getMessage());
} catch (IOException e) {
    System.err.println(e.getMessage());
} catch (ProActiveException e) {
    System.err.println(e.getMessage());
} finally {
    //TODO 6. Stop all the objects and JVM
}
}
}

```

6.5.3. Proposed Work

You can find the skeleton codes for this exercise into the `tutorials/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/migration/` directory. Full code is also available into the directory `ProActive/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/migration/`.

1. Write the code for the migration method using `org.objectweb.proactive.api.PAMobileAgent.migrateTo(Node URL)`
2. Create the menu using the node URLs - `org.objectweb.proactive.api.Node.getNodeInformation().getURL()`
3. Migrate the node

4. Get and print the state of the node
5. Print the execution time of the last request
6. Stop all the objects and JVM

Once filled in, go to the tutorials/compile directory and type `build[.bat] cmagent.migration` to compile your code. Then, after a successful compilation, go to the scripts/CMAgent directory and launch the `migrateCMA.[sh|bat]` script to execute your code.

6.5.4. Solution and Full Code

This first question is straightforward since you just have to call the `PAMobileAgent.migrateTo` method. The `CMAgentMigrator` class is therefore as follows:

```
package org.objectweb.proactive.examples.userguide.cmagent.migration;

import java.io.Serializable;

import org.objectweb.proactive.api.PAMobileAgent;
import org.objectweb.proactive.core.ProActiveException;
import org.objectweb.proactive.core.node.Node;
import org.objectweb.proactive.examples.userguide.cmagent.initialized.CMAgentInitialized;
import org.objectweb.proactive.extensions.annotation.ActiveObject;
import org.objectweb.proactive.extensions.annotation.MigrationSignal;

@ActiveObject
public class CMAgentMigrator extends CMAgentInitialized implements Serializable {

    @MigrationSignal
    public void migrateTo(Node whereTo) {
        try {
            //TODO 1. Migrate the active object to the Node received as parameter
            //should be the last call in this method
            //instructions after a call to PAMobileAgent.migrateTo are NOT executed
            PAMobileAgent.migrateTo(whereTo);
        } catch (ProActiveException moveExcep) {
            System.err.println(moveExcep.getMessage());
        }
    }
}
```



Warning

As it said in comments, codes after the `migrateTo` method will not be executed.

Other questions are not more difficult and here is the `CMAAgentMigrator` class:

```
package org.objectweb.proactive.examples.userguide.cmagent.migration;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;

import org.objectweb.proactive.ActiveObjectCreationException;
```

```

import org.objectweb.proactive.api.PAActiveObject;
import org.objectweb.proactive.api.PALifeCycle;
import org.objectweb.proactive.core.ProActiveException;
import org.objectweb.proactive.core.node.Node;
import org.objectweb.proactive.core.node.NodeException;
import org.objectweb.proactive.extensions.gcmdeployment.PAGCMDeployment;
import org.objectweb.proactive.gcmdeployment.GCMApplication;
import org.objectweb.proactive.gcmdeployment.GCMVirtualNode;

public class Main {
    private static GCMApplication pad;

    private static GCMVirtualNode deploy(String descriptor) throws NodeException, ProActiveException {

        //Create object representation of the deployment file
        pad = PAGCMDeployment.loadApplicationDescriptor(new File(descriptor));
        //Activate all Virtual Nodes
        pad.startDeployment();
        //Wait for all the virtual nodes to become ready
        pad.waitReady();

        //Get the first Virtual Node specified in the descriptor file
        GCMVirtualNode vn = pad.getVirtualNodes().values().iterator().next();

        return vn;
    }

    public static void main(String args[]) {
        try {
            BufferedReader inputBuffer = new BufferedReader(new InputStreamReader(System.in));
            GCMVirtualNode vn = deploy(args[0]);

            //create the active object
            CMAgentMigrator ao = (CMAgentMigrator) PAActiveObject.newActive(CMAgentMigrator.class.getName(),
                new Object[] {}, vn.getANode());

            int k = 1;
            int choice;
            while (k != 0) {

                //display the menu with the available nodes
                k = 1;
                for (Node node : vn.getCurrentNodes()) {
                    //TODO 2. Add the node URL to the menu
                    System.out.println(k + ". Statistics for node :" + node.getNodeInformation().getURL());
                    k++;
                }
                System.out.println("0. Exit");

                Node[] nodeArray = vn.getCurrentNodes().toArray(new Node[] {});

                //select a node
                do {

```

```

        System.out.print("Choose a node :> ");
        try {
            // Read an option from keyboard
            choice = Integer.parseInt(inputBuffer.readLine().trim());
        } catch (NumberFormatException noExcep) {
            choice = -1;
        }
        while (!(choice >= 1 && choice < k || choice == 0));
        if (choice == 0)
            break;

        //TODO 3. Migrate the active object to the selected node: choice-1
        ao.migrateTo(nodeArray[choice - 1]); //migrate
        //TODO 4. Get the state and the last request time and print them out
        String currentState = ao.getCurrentState().toString(); //get the state
        System.out.println("\n" + currentState);
        //TODO 5. Print the execution time of the last request
        System.out.println("Calculating the statistics took " +
            ao.getLastRequestServeTime().getLongValue() + "ms\n");
    }
    catch (NodeException nodeExcep) {
        System.err.println(nodeExcep.getMessage());
    }
    catch (ActiveObjectCreationException aoExcep) {
        System.err.println(aoExcep.getMessage());
    }
    catch (IOException e) {
        System.err.println(e.getMessage());
    }
    catch (ProActiveException e) {
        System.err.println(e.getMessage());
    }
    finally {
        //TODO 6. Stop all the objects and JVM
        if (pad != null)
            pad.kill();
        PALifeCycle.exitSuccess();
    }
}
}

```

6.6. Groups of Monitoring Agents

In this part of the tutorial, we will show how to use groups of active objects. We will create several active objects that we add and remove from a group. The group will be used to retrieve the **State** object from all the active objects in the group. In order to ease the use of the group communication, ProActive provides a set of static methods in the **PAGroup** class and a set of methods in the **Group** interface. ProActive also provides typed group communication, meaning that only methods defined on classes or interfaces implemented by members of the group can be called. There are two ways to create groups of active objects: using the **PAGroup.newGroup(...)** static method or using the **PAGroup.newGroupInParallel(...)** one. Once the group created, it is possible to turn it into active using the **PAGroup.turnActiveGroup(...)** static method. Group creation takes several parameters similar to those of active object creation. To understand parameters that are not explained here, please refer to Chapter 9, Active Objects: Creation And Advanced Concepts as it contains a detailed explanation of every active object creation parameter.

6.6.1. Classes Used

New Classes

- `org.objectweb.proactive.api.PAGroup` - used to create a group of active objects

- `org.objectweb.proactive.core.group.Group` - used to control the group of objects

Previously used classes

- `org.objectweb.proactive.extensions.gcmdployment.PAGCMDDeployment` - used to create a object version of the deployment descriptor
- `org.objectweb.proactive.core.ProActiveException` - used to handle ProActive exceptions
- `org.objectweb.proactive.gcmdployment.GCMApplication` - represents the application which is being deployed
- `org.objectweb.proactive.gcmdployment.GCMVirtualNode` - used to control and instantiate virtual node objects
- `org.objectweb.proactive.core.node.Node` - used to instantiate and control Node objects
- `org.objectweb.proactive.Body` - used to access the body of the active object
- `org.objectweb.proactive.PALifeCycle` - controls the lifecycle of the ProActive application
- `org.objectweb.proactive.Service` - used to access the queue of the active object
- `org.objectweb.proactive.InitActive` - used for defining the `initActivity(Body body)` method, which is run at active object initialization
- `org.objectweb.proactive.EndActive` - used for defining the `endActivity(Body body)` method, which is run at active object destruction
- `org.objectweb.proactive.RunActive` - used for defining the `runActivity(Body body)` method, which manages the queue of requests
- `org.objectweb.proactive.core.util.wrapper.LongWrapper` - used to wrap the Long return type
- `org.objectweb.proactive.api.PAActiveObject` - used to create an instance of an active object
- `org.objectweb.proactive.core.node.NodeException` - used to catch the exceptions that the creation of the Node might throw
- `org.objectweb.proactive.ActiveObjectCreationException` - used to catch the exceptions that the creation of the active object might throw
- `java.io.Serializable` - used to make the State object serializable so it can be sent across the network
- `java.lang.management.ManagementFactory` - used to get various information about the machine the active object is running on
- `java.net.InetAddress` - used to get the address of the active object is running on
- `java.net.UnknownHostException` - used to catch the exceptions that might be thrown when requesting host information
- `java.util.Date` - used to get the time for the requested state

6.6.2. Architecture and Skeleton Code

In this example, we only need to modify the Main class to create the group of objects. To instantiate the active object, we will use the `CMAgentInitialized` class that we have defined previously.

Main skeleton class:

```
package org.objectweb.proactive.examples.userguide.cmagent.groups;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Vector;

import org.objectweb.proactive.ActiveObjectCreationException;
import org.objectweb.proactive.api.PAActiveObject;
import org.objectweb.proactive.api.PAGroup;
import org.objectweb.proactive.api.PALifeCycle;
import org.objectweb.proactive.core.ProActiveException;
import org.objectweb.proactive.core.group.Group;
import org.objectweb.proactive.core.mop.ClassNotReifiableException;
```

```

import org.objectweb.proactive.core.node.Node;
import org.objectweb.proactive.core.node.NodeException;
import org.objectweb.proactive.examples.userguide.cmagent.migration.CMAgentMigrator;
import org.objectweb.proactive.examples.userguide.cmagent.simple.State;
import org.objectweb.proactive.extensions.gcmdeployment.PAGCMDDeployment;
import org.objectweb.proactive.gcmdeployment.GCMApplication;
import org.objectweb.proactive.gcmdeployment.GCMVirtualNode;

public class Main {
    private static GCMApplication pad;

    private static GCMVirtualNode deploy(String descriptor) throws NodeException, ProActiveException {

        //Create object representation of the deployment file
        pad = PAGCMDDeployment.loadApplicationDescriptor(new File(descriptor));
        //Activate all Virtual Nodes
        pad.startDeployment();
        //Wait for all the virtual nodes to become ready
        pad.waitReady();

        //Get the first Virtual Node specified in the descriptor file
        GCMVirtualNode vn = pad.getVirtualNodes().values().iterator().next();

        return vn;
    }

    public static void main(String args[]) {
        try {
            Vector<CMAgentMigrator> agents = new Vector<CMAgentMigrator>();
            BufferedReader inputBuffer = new BufferedReader(new InputStreamReader(System.in));
            GCMVirtualNode vn = deploy(args[0]);
            //TODO 1. Create a new empty group
            //TODO 2. Create a collection of active objects with on object on each node
            //TODO 3. Get a management representation of the monitors group
            //ask for adding or removing nodes
            //get statistics
            int k = 1;
            int choice;
            while (k != 0) {
                //display the menu
                k = 1;
                System.out.println("Toggle monitored nodes (*) or display statistics: ");
                for (CMAgentMigrator agent : agents) {
                    //TODO 4. Print the node URL
                    if (gA.contains(agent)) {
                        } else {
                        }
                    }
                k++;
            }
            System.out.println("-1. Display statistics for monitored nodes");
            System.out.println(" 0. Exit");

            //select a node

```



```

do {
    System.out.print("Choose a node to add or remove :- ");
    try {
        // Read an option from keyboard
        choice = Integer.parseInt(inputBuffer.readLine().trim());
    } catch (NumberFormatException noExcep) {
        choice = -1;
    }
    while (!(choice >= 1 && choice < k || choice == 0 || choice == -1));
    if (choice == 0)
        break;
    if (choice == -1) {

        State resultsGroup = monitorsGroup.getCurrentState();
        while (PAGroup.size(resultsGroup) > 0) {
            //TODO 5. Use PAGroup.waitAndGetOneThenRemoveIt() to control the list of State futures
            System.out.println(statistic.toString());
        }
    } else {
        //TODO 6. Use the management representation to add or remove
        // the agent (choice-1) to/from the group.
    }
}

} catch (NodeException nodeExcep) {
    System.err.println(nodeExcep.getMessage());
} catch (ActiveObjectCreationException aoExcep) {
    System.err.println(aoExcep.getMessage());
} catch (IOException e) {
    System.err.println(e.getMessage());
} catch (ClassNotReifiableException e) {
    System.err.println(e.getMessage());
} catch (ClassNotFoundException e) {
    System.err.println(e.getMessage());
} catch (ProActiveException e) {
    System.err.println(e.getMessage());
} finally {
    //stopping all the objects and JVMs
    if (pad != null)
        pad.kill();
    PALifeCycle.exitSuccess();
}
}
}

```

6.6.3. Proposed Work

You can find the skeleton codes for this exercise into the `tutorials/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/groups/` directory. Full code is also available into the directory `ProActive/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/groups/`.

1. Create a new empty group using `PAGroup.newGroup(..)`
2. Create a collection of active objects with one object on each node
3. Get a management representation of the monitors group using the `Group` interface

4. Print the Node URL using `PAActiveObject.getActiveObjectNodeId(...)`
5. Use `PAGroup.waitForOneThenRemoveIt()` to control the list of `State` futures
6. Use the management representation to add or remove the agent (choice-1) to/from the group.

Once filled in, go to the `tutorials/compile` directory and type `build[.bat] cmagent.groups` to compile your code. Then, after a successful compilation, go to the `scripts/CMAgent` directory and launch the `groups.[sh|bat]` script to execute your code.

6.6.4. Solution And Full Code

To create the group, we need to specify the type of the group. In our case, we use the previously defined `CMAgentMigrator`. We first create the group and then get a management representation of the group through the `Group` interface. The `Group` interface has the necessary methods for adding and removing members of the group.

```
//TODO 1. Create a new empty group
CMAgentMigrator monitorsGroup = (CMAgentMigrator) PAGroup.newGroup(CMAgentMigrator.class
    .getName());
//TODO 2. Create a collection of active objects with on object on each node
for (Node node : vn.getCurrentNodes()) {
    CMAgentMigrator ao = (CMAgentMigrator) PAActiveObject.newActive(CMAgentMigrator.class
        .getName(), new Object[] {}, node);
    agents.add(ao);
}
//TODO 3. Get a management representation of the monitors group
Group<CMAgentMigrator> gA = PAGroup.getGroup(monitorsGroup);
```

We use synchronization to wait for all the agents to send the states. As an agent returns a `State`, we remove it from the list of futures.

```
//TODO 5. Use PAGroup.waitForOneThenRemoveIt() to control the list of State futures
State statistic = (State) PAGroup.waitForOneThenRemoveIt(resultsGroup);
```

To test if an agent is into the group and then to add or remove it, we use the same methods as for the classical containers, that is to say, the methods `contains`, `add` and `remove`:

```
if (gA.contains(agents.elementAt(choice - 1))) {
    gA.remove(agents.elementAt(choice - 1));
} else {
    gA.add(agents.elementAt(choice - 1));
}
```

The full `Main` method:

```
public static void main(String args[]) {
    try {
        Vector<CMAgentMigrator> agents = new Vector<CMAgentMigrator>();
        BufferedReader inputBuffer = new BufferedReader(new InputStreamReader(System.in));
        GCMVirtualNode vn = deploy(args[0]);
        //TODO 1. Create a new empty group
        CMAgentMigrator monitorsGroup = (CMAgentMigrator) PAGroup.newGroup(CMAgentMigrator.class
            .getName());
        //TODO 2. Create a collection of active objects with on object on each node
        for (Node node : vn.getCurrentNodes()) {
            CMAgentMigrator ao = (CMAgentMigrator) PAActiveObject.newActive(CMAgentMigrator.class
                .getName(), new Object[] {}, node);
            agents.add(ao);
        }
    }
}
```

```

//TODO 3. Get a management representation of the monitors group
Group<CMAgentMigrator> gA = PAGroup.getGroup(monitorsGroup);
//ask for adding or removing nodes
//get statistics
int k = 1;
int choice;
while (k != 0) {
    //display the menu
    k = 1;
    System.out.println("Toggle monitored nodes (*) or display statistics: ");
    for (CMAgentMigrator agent : agents) {
        //TODO 4. Print the node URL
        if (gA.contains(agent)) {
            System.out.println(" " + k + ".*" + PAActiveObject.getActiveObjectNodeUrl(agent));
        } else {
            System.out.println(" " + k + "." + PAActiveObject.getActiveObjectNodeUrl(agent));
        }
        k++;
    }
    System.out.println("-1. Display statistics for monitored nodes");
    System.out.println("0. Exit");

    //select a node
    do {
        System.out.print("Choose a node to add or remove :> ");
        try {
            // Read an option from keyboard
            choice = Integer.parseInt(inputBuffer.readLine().trim());
        } catch (NumberFormatException noExcep) {
            choice = -1;
        }
    } while (!(choice >= 1 && choice < k || choice == 0 || choice == -1));
    if (choice == 0)
        break;
    if (choice == -1) {

        State resultsGroup = monitorsGroup.getCurrentState();
        while (PAGroup.size(resultsGroup) > 0) {
            //TODO 5. Use PAGroup.waitAndGetOneThenRemoveIt() to control the list of State futures
            State statistic = (State) PAGroup.waitAndGetOneThenRemoveIt(resultsGroup);
            System.out.println(statistic.toString());
        }
    } else {
        //TODO 6. Use the management representation to add or remove
        // the agent (choice-1) to/from the group.
        if (gA.contains(agents.elementAt(choice - 1))) {
            gA.remove(agents.elementAt(choice - 1));
        } else {
            gA.add(agents.elementAt(choice - 1));
        }
    }
}

} catch (NodeException nodeExcep) {

```

```

    System.err.println(nodeExcep.getMessage());
} catch (ActiveObjectCreationException aoExcep) {
    System.err.println(aoExcep.getMessage());
} catch (IOException e) {
    System.err.println(e.getMessage());
} catch (ClassNotReifiableException e) {
    System.err.println(e.getMessage());
} catch (ClassNotFoundException e) {
    System.err.println(e.getMessage());
} catch (ProActiveException e) {
    System.err.println(e.getMessage());
} finally {
    //stopping all the objects and JVMs
    if (pad != null)
        pad.kill();
    PALifeCycle.exitSuccess();
}
}

```

6.7. Monitoring Agent As A Web Service

In this example, we will show how to expose an active object as a web service. We will use the same monitoring agent as in the previous examples and expose it as a web service. You can expose it either to the local Jetty server or to a Tomcat server (or an other application server). For the latter, you first have to generate the `proactive.war` archive, using the command `build proactiveWar` into the `compile` directory and to put it in your application server.

6.7.1. Classes Used

New Classes

- `org.objectweb.proactive.extensions.webservices.AbstractWebServicesFactory` - used to get a `WebServicesFactory` depending on the CXF framework
- `org.objectweb.proactive.extensions.webservices.WebServicesFactory` - used to get a `WebServices` corresponding to a specified url
- `org.objectweb.proactive.extensions.webservices.WebServices` - used to transform the active object into a web service
- `org.objectweb.proactive.extensions.webservices.exceptions.WebServicesException` - used to report web service exceptions

Previously used classes

- `org.objectweb.proactive.api.PAActiveObject` - used to create an instance of an active object
- `org.objectweb.proactive.core.node.NodeException` - used to catch the exceptions that the creation of the Node might throw
- `org.objectweb.proactive.ActiveObjectCreationException` - used to catch the exceptions that the creation of the active object might throw

6.7.2. Architecture and Skeleton Code

In this example, we extend `CMAgentInitialized` and create a `CMAgentWebService` that we expose through the main method.

Since the usage of the webservice is independent of the underlying implementation, we give here only a Java example of how to access the webservice. However, you can simply use your favorite browser.

```

public class CMAgentWebServiceClient {

```

```

public static void main(String[] args) {

    try {
        String url = "";
        String wsFrameWork = "";
        if (args.length == 0) {
            url = "http://localhost:8080/";
            wsFrameWork = CentralPAPropertyRepository.PA_WEBSERVICES_FRAMEWORK.getValue();
        } else if (args.length == 1) {
            try {
                new URL(args[0]);
                url = args[0];
                wsFrameWork = CentralPAPropertyRepository.PA_WEBSERVICES_FRAMEWORK.getValue();
            } catch (MalformedURLException me) {
                // Given argument is not the URL to use to expose the web service, it should be the web service framework
                to use
                url = "http://localhost:8080/";
                wsFrameWork = args[0];
            }
        } else if (args.length == 2) {
            url = args[0];
            wsFrameWork = args[1];
        } else {
            System.out.println("Wrong number of arguments:");
            System.out.println("Usage: java CMAgentWebServiceClient [url] [wsFrameWork]");
            System.out.println("where wsFrameWork should be 'cxf'");
            return;
        }

        ClientFactory cf = AbstractClientFactory.getClientFactory(wsFrameWork);
        Client client = cf.getClient(url, "cmAgentService", CMAgentService.class);

        Object[] response = client.call("getCurrentState", null, State.class);

        System.out.println("Current state is:\n" + ((State) response[0]).toString());

        response = client.call("waitLastRequestServeTime", null, long.class);

        System.out.println("Last request serve time = " + response[0]);

    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

In the CMAgentWebService, we need to write 3 lines of code to use the active object as an webservice.

```
package org.objectweb.proactive.examples.userguide.cmagent.webservice;
```

```
import java.net.MalformedURLException;
```

```
import java.net.URL;
```

```

import org.objectweb.proactive.ActiveObjectCreationException;
import org.objectweb.proactive.api.PAActiveObject;
import org.objectweb.proactive.core.config.CentralPAPropertyRepository;
import org.objectweb.proactive.core.node.NodeException;
import org.objectweb.proactive.examples.userguide.cmagent.initialized.CMAgentInitialized;
import org.objectweb.proactive.extensions.annotation.ActiveObject;
import org.objectweb.proactive.extensions.webservices.AbstractWebServicesFactory;
import org.objectweb.proactive.extensions.webservices.WebServices;
import org.objectweb.proactive.extensions.webservices.WebServicesFactory;
import org.objectweb.proactive.extensions.webservices.exceptions.WebServicesException;

@ActiveObject
public class CMAgentService extends CMAgentInitialized {

    // we cannot use the getLastRequestServeTime() method
    // directly since LongWrapper is not recognize
    // TODO: See why
    public long waitLastRequestServeTime() {
        return this.getLastRequestServeTime().getLongValue();
    }

    public static void main(String[] args) {

        String url = "";
        String wsFrameWork = "";
        if (args.length == 0) {
            url = AbstractWebServicesFactory.getLocalUrl();
            wsFrameWork = CentralPAPropertyRepository.PA_WEBSERVICES_FRAMEWORK.getValue();
        } else if (args.length == 1) {
            try {
                new URL(args[0]);
                url = args[0];
                wsFrameWork = CentralPAPropertyRepository.PA_WEBSERVICES_FRAMEWORK.getValue();
            } catch (MalformedURLException me) {
                // Given argument is not the URL to use to expose the web service, it should be the web service framework to
                use
                url = AbstractWebServicesFactory.getLocalUrl();
                wsFrameWork = args[0];
            }
        } else if (args.length == 2) {
            url = args[0];
            wsFrameWork = args[1];
        } else {
            System.out.println("Wrong number of arguments:");
            System.out.println("Usage: java CMAgentService [url] [wsFrameWork]");
            System.out.println("where wsFrameWork should be 'cxf'");
            return;
        }

        System.out.println("Started a monitoring agent on : " + url);

        CMAgentService hw;
        try {

```

```

hw = (CMAgentService) PAActiveObject.newActive(
    "org.objectweb.proactive.examples.userguide.cmagent.webservice.CMAgentService",
    new Object[] {});

// TODO 1.Expose as web service (on URL 'url') the methods
// "getLastRequestServeTime" and "getCurrentState"
// of 'hw' CMAgentService.
// Name your service "cmAgentService" and use the web service framework given
// in argument.

} catch (ActiveObjectCreationException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (NodeException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (WebServicesException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}
}

```

6.7.3. Proposed Work

You can find the skeleton codes for this exercise into the `tutorials/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/webservice/` directory. Full code is also available into the directory `ProActive/src/Examples/org/objectweb/proactive/examples/userguide/cmagent/webservice/`.

1. Get a `WebServicesFactory` using a static method of the `AbstractWebServicesFactory` class
2. Get a `WebServices` using the `WebServicesFactory` instance you have just got
3. Expose your active object using this `WebServices` instance

Once filled in, go to the `tutorials/compile` directory and type `build[.bat] cmagent.webservice` to compile your code. Then, after a successful compilation, go to the `scripts/CMAgent` directory and launch the `exposeCMA.[sh|bat] [url]` script to execute your code. The url is only needed when you want to deploy the agent on an external application server.

6.7.4. Solution And Code

```

WebServicesFactory wsf = AbstractWebServicesFactory.getWebServicesFactory(wsFrameWork);
WebServices ws = wsf.getWebServices(url);
ws.exposeAsWebService(hw, "cmAgentService", new String[] { "waitLastRequestServeTime",
    "getCurrentState" });

```

Refer to [Exporting active objects and components as Web Services](#)⁴ to get more information about web services exposition.

6.8. Primality test tutorial

This chapter presents a step by step tutorial whose the purpose is to develop a distributed application to test if a number is prime using a self-made master/worker. The program to be created is a distributed version of a sequential primality test. The parallelism shown here is called data-level parallelism because the same operation can be applied simultaneously to multiples piece of data.

- The first step is to study the sequential version of the application and to determinate what could be parallelized.

⁴ file:///home/hudson/workdir/workspace/ProActive_Documentation/compile/..doc/built/Programming/GetStarted/pdf/../../AdvancedFeatures/multiple_html/WebServices.html

- The second part introduces the distribution of the computation using the Computing and Monitoring agents from the previous tutorial. This part highlights the use of asynchronous communications and future monitoring as well as a straightforward round-robin load-balancer.
- Finally, the last part of the tutorial presents the Master-Worker API featured by ProActive. This API removes all the complexity introduced by the distribution simply by handling it internally.

6.8.1. A Sequential Version of the Primality Test

In this section, we will look at a simple sequential version of the application to find if a candidate number is prime.

6.8.1.1. Architecture of the Sequential Version

The provided application uses a straightforward algorithm: Given a candidate number m , the algorithm loops over all the integers from 2 to square root of m (rather than to $m - 1$). At each step, it divides the candidate number m by the current integer (the running divisor). The loop ends when either:

- the running divisor divides the candidate number and then, the candidate number is not a prime number.
- the running divisor is greater than the square root of the candidate number and so, the candidate number is prime.

You can find code for this version into the `tutorials/src/Examples/org/objectweb/proactive/examples/userguide/primes/sequential/` directory.

6.8.1.2. Implementation of the Sequential Version

The `main` method selects a range between 2 and the square root of the candidate and calls `isPrime` method that tries to divide the candidate by a divider in a given range. For this example a prime number is given as candidate by default.

```
package org.objectweb.proactive.examples.userguide.primes.sequential;

/**
 * This class illustrates a sequential algorithm for primality test.
 * <p>
 * Some primes : 43980423167991, 63018038201, 2147483647
 *
 * @author The ProActive Team
 */
public class Main {

    public static void main(String[] args) {
        // The default value for the candidate to test (is prime)
        long candidate = 30932158813330571;
        // Parse the number from args if there is some
        if (args.length > 0) {
            try {
                candidate = Long.parseLong(args[0]);
            } catch (NumberFormatException numberException) {
                System.err.println(numberException.getMessage());
                System.err.println("Usage: Main <candidate>");
            }
        }
        // We don't need to check numbers greater than the square-root of the
        // candidate in this algorithm
        long squareRootOfCandidate = (long) Math.ceil(Math.sqrt(candidate));
        // Begin from 2 the first known prime number
```



```

long begin = 2;
// Until the end of the range
long end = squareRootOfCandidate;
// Check the primality
boolean isPrime = Main.isPrime(candidate, begin, end);
// Display the result
System.out.println("\n" + candidate + (isPrime ? " is prime." : " is not prime.") + "\n");
}

/**
 * Tests a primality of a specified number in a specified range.
 *
 * @param candidate
 *      the candidate number to check
 * @param begin
 *      starts check from this value
 * @param end
 *      checks until this value
 * @return <code>true</code> if is prime; <code>false</code> otherwise
 */
public static Boolean isPrime(long candidate, long begin, long end) {
    for (long divider = begin; divider < end; divider++) {
        if ((candidate % divider) == 0) {
            return false;
        }
    }
    return true;
}

```

6.8.1.3. Running the Sequential Version

Go to the tutorials/compile directory and type `build[.bat] primes.sequential` to compile your code. Then, after a successful compilation, go to the scripts/Primes directory and launch the `sequential.[sh|bat]` script with the integer you want to test.

```
sequential.[sh|bat] 2147483647
```

6.8.2. A Distributed Version of the Primality Test

In this section, we will distribute the computation using agents from the precedent tutorial.

6.8.2.1. Classes Used

ProActive

- `org.objectweb.proactive.api.PAActiveObject` - used to create an instance of an active object
- `org.objectweb.proactive.extensions.gcmdeployment.PAGCMDeployment` - used for the deployment
- `org.objectweb.proactive.api.PAFuture` - used to catch the exceptions that the creation of the active object might throw
- `org.objectweb.proactive.core.util.wrapper.BooleanWrapper` - used to wrap the primitive boolean type
- `org.objectweb.proactive.gcmdeployment.GCMApplication` - represents the application which is being deployed
- `org.objectweb.proactive.gcmdeployment.GCMVirtualNode` - used to control and instantiate virtual node objects

Other

- `org.objectweb.proactive.examples.userguide.cmagent.simple.CMAgent` - parent class of the workers

- `java.util.Vector` - used by the manager to store future results

6.8.2.2. Architecture of the Distributed Version

The test range from 2 to \sqrt{n} is divided into a number of intervals that will be sent asynchronously to workers by the manager using a round-robin algorithm. Once all intervals have been sent, the manager waits for any answers and checks the result.

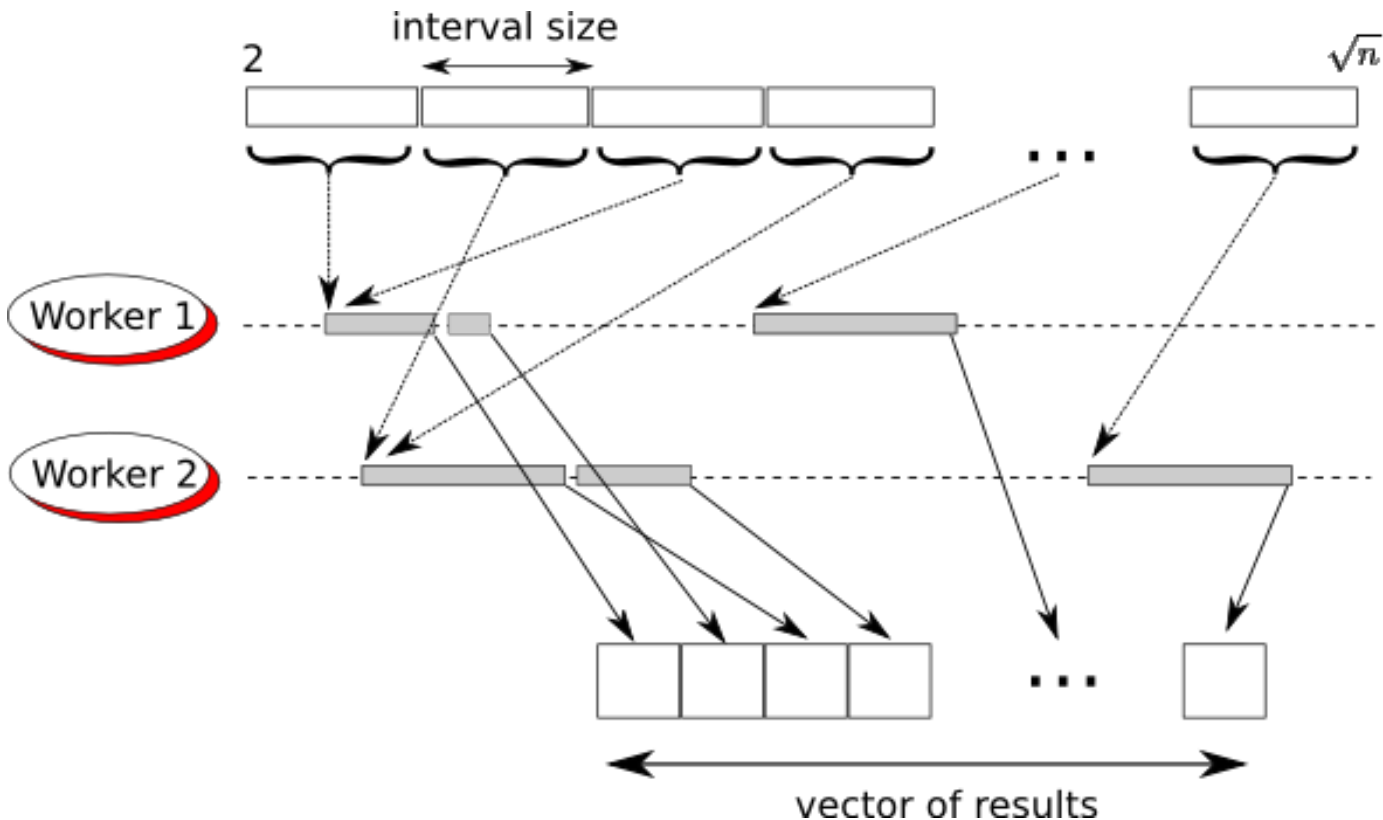


Figure 6.1. Distribution of the Test Range (Example with 2 Workers)

Each worker serves requests in a FIFO order, meanwhile the order of result reception by the manager is nondeterministic, it depends on various factors like workers load, network latency, etc.

`CMAgentPrimeWorker` and `CMAgentPrimeManager` classes have several methods implementing the mechanism described above. The manager class has a method for adding workers to its list (`addWorker(CMAgentPrimeWorker)`), and a method that distributes the workload by performing method calls on workers (`isPrime(long number)`). The worker class has a method that checks if a number is prime (`isPrime(long number, long begin, long end)`). A main class is used to deploy the manager and its workers on a specified deployment descriptor. For this example, the manager and worker classes without code look like hereafter. Try to fill in the code for the methods. An example implementation is also provided below.

```
package org.objectweb.proactive.examples.userguide.primes.distributed;

import java.util.Vector;

import org.objectweb.proactive.api.PAFuture;
import org.objectweb.proactive.core.util.wrapper.BooleanWrapper;
import org.objectweb.proactive.extensions.annotation.ActiveObject;
```

```

/**
 * @author The ProActive Team
 */
@ActiveObject
public class CMAgentPrimeManager {
    /**
     * A vector of references on workers
     */
    private Vector<CMAgentPrimeWorker> workers = new Vector<CMAgentPrimeWorker>();
    /**
     * Default interval size
     */
    public static final int INTERVAL_SIZE = 100;

    /**
     * Empty no-arg constructor needed by ProActive
     */
    public CMAgentPrimeManager() {
    }

    /**
     * Tests a primality of a specified number. Synchronous !
     *
     * @param number
     *      The number to test
     * @return <code>true</code> if is prime; <code>false</code> otherwise
     */
    public boolean isPrime(long number) {
        // We don't need to check numbers greater than the square-root of the
        // candidate in this algorithm
        long squareRootOfCandidate = (long) Math.ceil(Math.sqrt(number));

        // Begin from 2 the first known prime number
        long begin = 2;

        // The number of intervals
        long nbOfIntervals = (long) Math.ceil(squareRootOfCandidate / INTERVAL_SIZE);

        // Until the end of the first interval
        long end = INTERVAL_SIZE;

        // The vector of futures
        final Vector<BooleanWrapper> answers = new Vector<BooleanWrapper>();

        // Non blocking (asynchronous method call)
        for (int i = 0; i <= nbOfIntervals; i++) {

            // Use round robin selection of worker
            int workerIndex = i % workers.size();
            CMAgentPrimeWorker worker = workers.get(workerIndex);

            //TODO 1. Send asynchronous method call to the worker

            //TODO 2. Add the future result to the vector of answers

```

```

    // Update the begin and the end of the interval
    begin = end + 1;
    end += INTERVAL_SIZE;
}
// Once all requests was sent
boolean prime = true;
int intervalNumber = 0;
// Loop until a worker returns false or vector is empty (all results have been checked)
while (!answers.isEmpty() && prime) {

    // TODO 3. Block until a new response is available
    // by using a static method from org.objectweb.proactive.api.PAFuture

    // Check the answer
    prime = answers.get(intervalNumber).getBooleanValue();

    // Remove the actualized future
    answers.remove(intervalNumber);

}
return prime;
}

/**
 * Adds a worker to the local vector
 *
 * @param worker
 *     The worker to add to the vector
 */
public void addWorker(CMAgentPrimeWorker worker) {
    this.workers.add(worker);
}
}

```

```

package org.objectweb.proactive.examples.userguide.primes.distributed;

import org.objectweb.proactive.core.util.wrapper.BooleanWrapper;
import org.objectweb.proactive.examples.userguide.cmaget.simple.CMAgent;
import org.objectweb.proactive.extensions.annotation.ActiveObject;

/**
 * @author The ProActive Team
 */
@ActiveObject
public class CMAgentPrimeWorker extends CMAgent {

    /**
     * Tests a primality of a specified number in a specified range.
     *
     * @param candidate
     *     the candidate number to check
     * @param begin
     *
     */
}

```

```

*      starts check from this value
* @param end
*      checks until this value
* @return <code>true</code> if is prime; <code>false</code> otherwise
*/
public BooleanWrapper isPrime(final long candidate, final long begin, final long end) {
    try {
        //Used for slowing down the application for in order
        //to let one stop it for checking fault tolerance behavior
        Thread.sleep(300);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    //TODO 4. Return a reifiable wrapper for the Boolean type
    // for asynchronous calls.
}
}

```

```

package org.objectweb.proactive.examples.userguide.primes.distributed;

import java.io.File;
import java.util.Collection;
import java.util.Iterator;

import org.objectweb.proactive.api.PAActiveObject;
import org.objectweb.proactive.core.ProActiveException;
import org.objectweb.proactive.core.node.Node;
import org.objectweb.proactive.core.node.NodeException;
import org.objectweb.proactive.examples.userguide.cmagent.simple.CMAgent;
import org.objectweb.proactive.extensions.gcmdeployment.PAGCMDeployment;
import org.objectweb.proactive.gcmdeployment.GCMApplication;
import org.objectweb.proactive.gcmdeployment.GCMVirtualNode;

/**
 * This class illustrates a distributed version of the sequential algorithm for
 * primality test based on the {@link CMAgent}.
 * <p>
 * Some primes : 3093215881333057, 4398042316799, 63018038201, 2147483647
 *
 * @author The ProActive Team
 */
public class Main {

    private static GCMApplication pad;

    private static Collection<GCMVirtualNode> deploy(String descriptor) {
        try {
            pad = PAGCMDeployment.loadApplicationDescriptor(new File(descriptor));
            //active all Virtual Nodes
            pad.startDeployment();
        }
    }
}

```

```

        pad.waitReady();

        return pad.getVirtualNodes().values();
    } catch (NodeException nodeExcep) {
        System.err.println(nodeExcep.getMessage());
    } catch (ProActiveException proExcep) {
        System.err.println(proExcep.getMessage());
    }
    return null;
}

public static void main(String[] args) {
    // The default value for the candidate to test (is prime)
    long candidate = 2147483647L;
    // Parse the number from args if there is some
    if (args.length > 1) {
        try {
            candidate = Long.parseLong(args[1]);
        } catch (NumberFormatException numberException) {
            System.err.println("Usage: Main <candidate>");
            System.err.println(numberException.getMessage());
        }
    }

    try {
        Collection<GCMVirtualNode> vNodes = deploy(args[0]);
        GCMVirtualNode vNode = vNodes.iterator().next();

        // create the active object on the first node on
        // the first virtual node available
        // start the master
        CMAgentPrimeManager manager = (CMAgentPrimeManager) PAActiveObject.newActive(
            CMAgentPrimeManager.class.getName(), new Object[] {}, vNode.getANode());

        //TODO 5: iterate through all nodes, deploy
        // a worker per node and add it to the manager

        // Check the primality (Send a synchronous method call to the manager)
        boolean isPrime = manager.isPrime(candidate);
        // Display the result
        System.out.println("\n" + candidate + (isPrime ? " is prime." : " is not prime.") + "\n");
        // Free all resources
        pad.kill();
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        System.exit(0);
    }
}
}

```

You can find the skeleton codes for this exercise into the `tutorials/src/Examples/org/objectweb/proactive/examples/userguide/primes/distributed/` directory. Full code is also available into the directory `ProActive/src/Examples/org/objectweb/proactive/examples/userguide/primes/distributed/`.

Once the code filled and the application run successfully, let's simulate a crash on a worker JVM to know if this application is fault-tolerant.

For this purpose add a `Thread.sleep()` in the `isPrime` method of the `CMAgentPrimeWorker` class in order to have time to monitor this application with IC2D.

From IC2D right-click on the JVM that contains a worker and hit the "Kill This JVM" option. Then Expect The Unexpected !

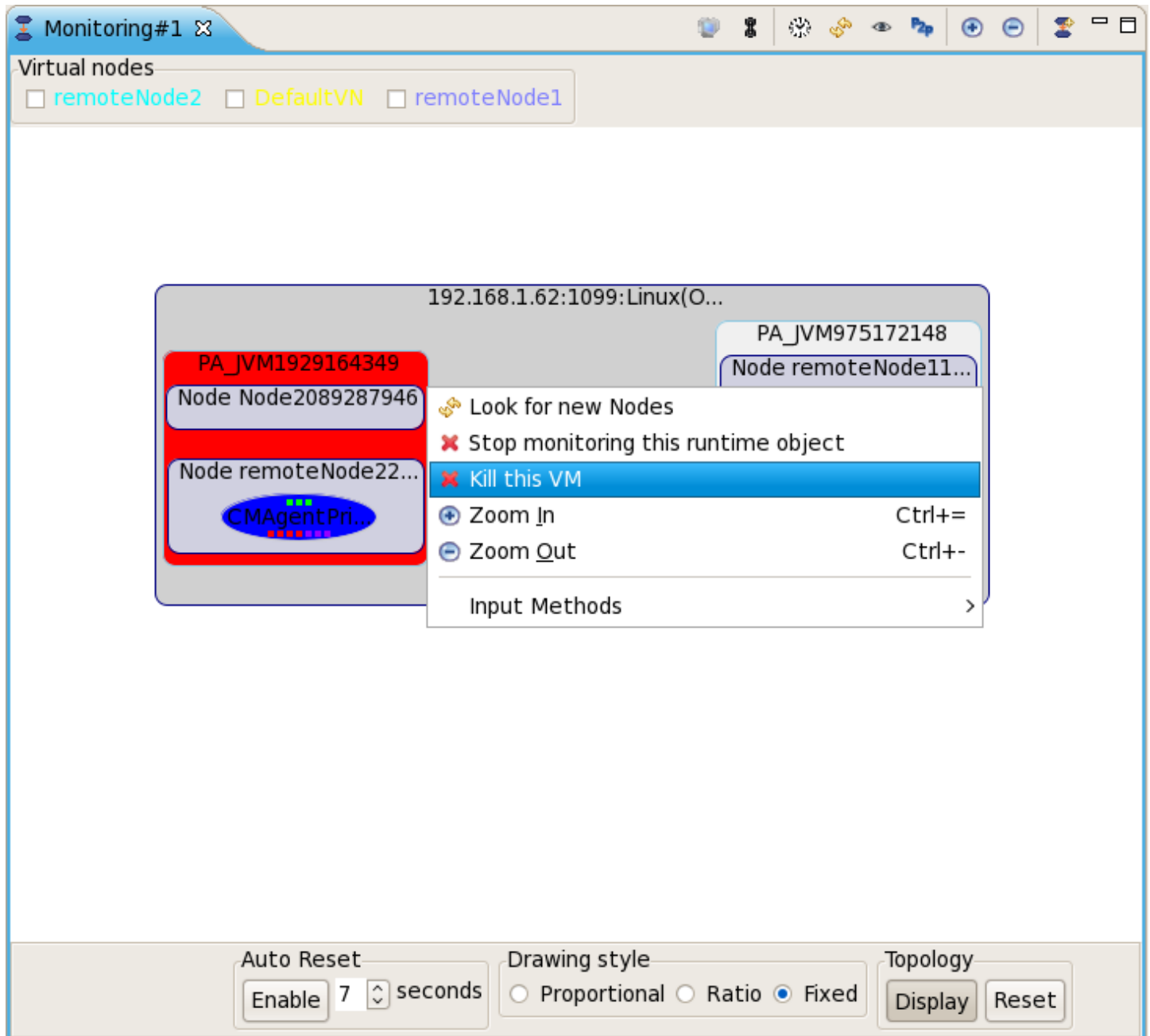


Figure 6.2. Killing a JVM from IC2D

Typically, the user will need to catch a `org.objectweb.proactive.core.body.exceptions.FutureMonitoringPingFailureException` exception manually to handle such failures.

Moreover, the application will never end or return an incorrect result since the manager will wait infinitely for an answer that never comes from a crashed worker.

6.8.2.3. Solution/Full Code

The Main class that deploys the manager and the workers.

```
package org.objectweb.proactive.examples.userguide.primes.distributed;

import java.io.File;
import java.util.Collection;
import java.util.Iterator;

import org.objectweb.proactive.api.PAActiveObject;
import org.objectweb.proactive.core.ProActiveException;
import org.objectweb.proactive.core.node.Node;
import org.objectweb.proactive.core.node.NodeException;
import org.objectweb.proactive.examples.userguide.cmagent.simple.CMAgent;
import org.objectweb.proactive.extensions.gcmdeployment.PAGCMDeployment;
import org.objectweb.proactive.gcmdeployment.GCMApplication;
import org.objectweb.proactive.gcmdeployment.GCMVirtualNode;

/**
 * This class illustrates a distributed version of the sequential algorithm for
 * primality test based on the {@link CMAgent}.
 * <p>
 * Some primes : 3093215881333057, 4398042316799, 63018038201, 2147483647
 *
 * @author The ProActive Team
 */
public class Main {

    private static GCMApplication pad;

    private static Collection<GCMVirtualNode> deploy(String descriptor) {
        try {
            pad = PAGCMDeployment.loadApplicationDescriptor(new File(descriptor));
            //active all Virtual Nodes
            pad.startDeployment();
            pad.waitReady();

            return pad.getVirtualNodes().values();
        } catch (NodeException nodeExcep) {
            System.err.println(nodeExcep.getMessage());
        } catch (ProActiveException proExcep) {
            System.err.println(proExcep.getMessage());
        }
        return null;
    }

    public static void main(String[] args) {
        // The default value for the candidate to test (is prime)
    }
}
```

```

long candidate = 2147483647L;
// Parse the number from args if there is some
if (args.length > 1) {
    try {
        candidate = Long.parseLong(args[1]);
    } catch (NumberFormatException numberException) {
        System.err.println("Usage: Main <candidate>");
        System.err.println(numberException.getMessage());
    }
}

try {
    Collection<GCMVirtualNode> vNodes = deploy(args[0]);
    GCMVirtualNode vNode = vNodes.iterator().next();

    // create the active object on the first node on
    // the first virtual node available
    // start the master
    CMAgentPrimeManager manager = (CMAgentPrimeManager) PAActiveObject.newActive(
        CMAgentPrimeManager.class.getName(), new Object[] {}, vNode.getANode());

    //TODO 5: iterate through all nodes, deploy
    // a worker per node and add it to the manager
    Iterator<Node> nodesIt = vNode.getCurrentNodes().iterator();
    while (nodesIt.hasNext()) {
        Node node = nodesIt.next();
        CMAgentPrimeWorker worker = (CMAgentPrimeWorker) PAActiveObject.newActive(
            CMAgentPrimeWorker.class.getName(), new Object[] {}, node);
        manager.addWorker(worker);
    }

    // Check the primality (Send a synchronous method call to the manager)
    boolean isPrime = manager.isPrime(candidate);
    // Display the result
    System.out.println("\n" + candidate + (isPrime ? " is prime." : " is not prime.") + "\n");
    // Free all resources
    pad.kill();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    System.exit(0);
}
}
}

```

The CMAgentPrimeManager class that distributes the intervals to workers.

```

package org.objectweb.proactive.examples.userguide.primes.distributed;

import java.util.Vector;

import org.objectweb.proactive.api.PAFuture;
import org.objectweb.proactive.core.util.wrapper.BooleanWrapper;
import org.objectweb.proactive.extensions.annotation.ActiveObject;

```



```

/**
 * @author The ProActive Team
 */
@ActiveObject
public class CMAgentPrimeManager {
    /**
     * A vector of references on workers
     */
    private Vector<CMAgentPrimeWorker> workers = new Vector<CMAgentPrimeWorker>();
    /**
     * Default interval size
     */
    public static final int INTERVAL_SIZE = 100;

    /**
     * Empty no-arg constructor needed by ProActive
     */
    public CMAgentPrimeManager() {
    }

    /**
     * Tests a primality of a specified number. Synchronous !
     *
     * @param number
     *     The number to test
     * @return <code>true</code> if is prime; <code>false</code> otherwise
     */
    public boolean isPrime(long number) {
        // We don't need to check numbers greater than the square-root of the
        // candidate in this algorithm
        long squareRootOfCandidate = (long) Math.ceil(Math.sqrt(number));

        // Begin from 2 the first known prime number
        long begin = 2;

        // The number of intervals
        long nbOfIntervals = (long) Math.ceil(squareRootOfCandidate / INTERVAL_SIZE);

        // Until the end of the first interval
        long end = INTERVAL_SIZE;

        // The vector of futures
        final Vector<BooleanWrapper> answers = new Vector<BooleanWrapper>();

        // Non blocking (asynchronous method call)
        for (int i = 0; i <= nbOfIntervals; i++) {

            // Use round robin selection of worker
            int workerIndex = i % workers.size();
            CMAgentPrimeWorker worker = workers.get(workerIndex);

            //TODO 1. Send asynchronous method call to the worker

```

```

    // Send asynchronous method call to the worker
    BooleanWrapper res = worker.isPrime(number, begin, end);

    //TODO 2. Add the future result to the vector of answers
    // Adds the future to the vector
    answers.add(res);

    // Update the begin and the end of the interval
    begin = end + 1;
    end += INTERVAL_SIZE;
}
// Once all requests was sent
boolean prime = true;
int intervalNumber = 0;
// Loop until a worker returns false or vector is empty (all results have been checked)
while (!answers.isEmpty() && prime) {

    // TODO 3. Block until a new response is available
    // by using a static method from org.objectweb.proactive.api.PAFuture
    // Will block until a new response is available
    intervalNumber = PAFuture.waitForAny(answers);

    // Check the answer
    prime = answers.get(intervalNumber).getBooleanValue();

    // Remove the actualized future
    answers.remove(intervalNumber);

}
return prime;
}

/**
 * Adds a worker to the local vector
 *
 * @param worker
 *     The worker to add to the vector
 */
public void addWorker(CMAgentPrimeWorker worker) {
    this.workers.add(worker);
}
}

```

The CMAgentPrimeWorker class that tests whether a candidate is prime.

```

package org.objectweb.proactive.examples.userguide.primes.distributed;

import org.objectweb.proactive.core.util.wrapper.BooleanWrapper;
import org.objectweb.proactive.examples.userguide.cmagent.simple.CMAgent;
import org.objectweb.proactive.extensions.annotation.ActiveObject;

/**
 * @author The ProActive Team

```

```

*/
@ActiveObject
public class CMAgentPrimeWorker extends CMAgent {

    /**
     * Tests a primality of a specified number in a specified range.
     *
     * @param candidate
     *         the candidate number to check
     * @param begin
     *         starts check from this value
     * @param end
     *         checks until this value
     * @return <code>true</code> if is prime; <code>false</code> otherwise
     */
    public BooleanWrapper isPrime(final long candidate, final long begin, final long end) {
        try {
            //Used for slowing down the application for in order
            //to let one stop it for checking fault tolerance behavior
            Thread.sleep(300);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        //TODO 4. Return a reifiable wrapper for the Boolean type
        // for asynchronous calls.
        for (long divider = begin; divider < end; divider++) {
            if ((candidate % divider) == 0) {
                return new BooleanWrapper(false);
            }
        }
        return new BooleanWrapper(true);
    }
}

```

6.8.2.4. Running the Distributed Version

Go to the tutorials/compile directory and type `build[.bat] primes.distributed` to compile your code. Then, after a successful compilation, go to the scripts/Primes directory and launch the `distributed.[sh|bat]` script with the integer you want to test.

```
distributed.[sh|bat] 2147483647
```

6.8.3. A Distributed Version of the Primality Test Using the Master-Worker API

In this section, we will distribute the computation using the Master-Worker API.

6.8.3.1. Classes Used

ProActive

- `org.objectweb.proactive.extensions.gcmdeployment.PAGCMDeployment` - used for the deployment
- `org.objectweb.proactive.gcmdeployment.GCMApplication` - represents the application which is being deployed
- `org.objectweb.proactive.gcmdeployment.GCMVirtualNode` - used to control and instantiate virtual node objects

- `org.objectweb.proactive.extensions.masterworker.ProActiveMaster` - used to create a master
- `org.objectweb.proactive.extensions.masterworker.interfaces.Task` - used to submit tasks to the master

Other

- `java.util.List` - used by to store the prime computation tasks

6.8.3.2. Architecture of the Master-Worker Version

For this example, the master and worker classes without code look like hereafter. Try to fill in the code for the methods. An example implementation is also provided below.



Note

With the Master-Worker API, there is no need to explicitly deal with primitive type wrappers, these kind of pitfalls are simply handled internally.

```
package org.objectweb.proactive.examples.userguide.primes.distributedmw;

import org.objectweb.proactive.extensions.masterworker.interfaces.Task;
import org.objectweb.proactive.extensions.masterworker.interfaces.WorkerMemory;

/**
 * Task to find if any number in a specified interval divides the given
 * candidate
 *
 * @author The ProActive Team
 */
public class FindPrimeTask implements Task<Boolean> {

    private long begin;
    private long end;
    private long taskCandidate;

    //TODO 1. Write the constructor for this task
    public FindPrimeTask(long taskCandidate, long begin, long end) {
    }

    //TOOD 2. Fill the code that checks if the taskCandidate
    // is prime. Note that no wrappers are needed !
    public Boolean run(WorkerMemory memory) {
        return Boolean.valueOf(true);
    }
}
```

```
package org.objectweb.proactive.examples.userguide.primes.distributedmw;

import java.net.URL;
import java.util.ArrayList;
import java.util.List;

import org.objectweb.proactive.extensions.masterworker.ProActiveMaster;
```

```

/**
 *
 * Some primes : 30932158813330571, 43980423167991, 63018038201, 2147483647
 *
 * @author The ProActive Team
 */
public class PrimeExampleMW {
    /**
     * Default interval size
     */
    public static final int INTERVAL_SIZE = 100;

    public static void main(String[] args) {
        // The default value for the candidate to test (is prime)
        long candidate = 2147483647L;
        // Parse the number from args if there is some
        if (args.length > 1) {
            try {
                candidate = Long.parseLong(args[1]);
            } catch (NumberFormatException numberException) {
                System.err.println("Usage: PrimeExampleMW <candidate>");
                System.err.println(numberException.getMessage());
            }
        }
        try {
            // Create the Master
            ProActiveMaster<FindPrimeTask, Boolean> master = new ProActiveMaster<FindPrimeTask, Boolean>();
            // Deploy resources
            master.addResources(new URL(args[0]));
            // Create and submit the tasks
            master.solve(createTasks(candidate));

            // TODO 3. Wait all results from master

            // Test the primality
            boolean isPrime = true;
            for (Boolean result : results) {
                isPrime = isPrime && result;
            }
            // Display the result
            System.out.println("\n" + candidate + (isPrime ? " is prime." : " is not prime.") + "\n");
            // Terminate the master and free all resources
            master.terminate(true);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            System.exit(0);
        }
    }
}
/**

```

```

* Creates the prime computation tasks to be solved
*
* @return A list of prime computation tasks
*/
public static List<FindPrimeTask> createTasks(long number) {
    List<FindPrimeTask> tasks = new ArrayList<FindPrimeTask>();

    // We don't need to check numbers greater than the square-root of the
    // candidate in this algorithm
    long squareRootOfCandidate = (long) Math.ceil(Math.sqrt(number));

    // Begin from 2 the first known prime number
    long begin = 2;

    // The number of intervals
    long nbOfIntervals = (long) Math.ceil(squareRootOfCandidate / INTERVAL_SIZE);

    // Until the end of the first interval
    long end = INTERVAL_SIZE;

    for (int i = 0; i <= nbOfIntervals; i++) {

        //TODO 4. Create a new task for the current interval and
        // add it to the list of tasks

        // Update the begin and the end of the interval
        begin = end + 1;
        end += INTERVAL_SIZE;
    }

    return tasks;
}

```

You can find the skeleton codes for this exercise into the `tutorials/src/Examples/org/objectweb/proactive/examples/userguide/primes/distributedmw/` directory. Full code is also available into the directory `ProActive/src/Examples/org/objectweb/proactive/examples/userguide/primes/distributedmw/`.

Once the code filled in and the application successfully run, let's crash a worker to see the difference with the precedent application.

For this purpose add a `Thread.sleep()` in the `run()` method of the `FindPrimeTask` class.

Like in the precedent application, from IC2D, kill a JVM that contains a worker. Then expect no exceptions or errors!

The main difference with the precedent application is that the master does not send any tasks to the workers. In fact, it is the workers that ask the master for tasks. Such "work-stealing" pattern proves to be fault-tolerant and it is one the benefits from using a High-Level ProActive API.

6.8.3.3. Solution/Full Code

The `FindPrimeTask` class that performs the computation is as follows:

```

package org.objectweb.proactive.examples.userguide.primes.distributedmw;

import org.objectweb.proactive.extensions.masterworker.interfaces.Task;

```

```

import org.objectweb.proactive.extensions.masterworker.interfaces.WorkerMemory;

/**
 * Task to find if any number in a specified interval divides the given
 * candidate
 *
 * @author The ProActive Team
 */
public class FindPrimeTask implements Task<Boolean> {

    private long begin;
    private long end;
    private long taskCandidate;

    //TODO 1. Write the constructor for this task
    public FindPrimeTask(long taskCandidate, long begin, long end) {
        this.begin = begin;
        this.end = end;
        this.taskCandidate = taskCandidate;
    }

    //TOOD 2. Fill the code that checks if the taskCandidate
    // is prime. Note that no wrappers are needed !
    public Boolean run(WorkerMemory memory) {
        try {
            Thread.sleep(300);
        } catch (Exception e) {
            e.printStackTrace();
        }
        for (long divider = begin; divider < end; divider++) {
            if ((taskCandidate % divider) == 0) {
                return Boolean.valueOf(false);
            }
        }
        return Boolean.valueOf(true);
    }
}

```

The PrimeExampleMW class that deploys the master and the workers looks like that:

```

package org.objectweb.proactive.examples.userguide.primes.distributedmw;

import java.net.URL;
import java.util.ArrayList;
import java.util.List;

import org.objectweb.proactive.extensions.masterworker.ProActiveMaster;

/**
 *
 * Some primes : 30932158813330571, 43980423167991, 63018038201, 2147483647
 */

```

```

*
* @author The ProActive Team
*
*/
public class PrimeExampleMW {
    /**
     * Default interval size
     */
    public static final int INTERVAL_SIZE = 100;

    public static void main(String[] args) {
        // The default value for the candidate to test (is prime)
        long candidate = 2147483647L;
        // Parse the number from args if there is some
        if (args.length > 1) {
            try {
                candidate = Long.parseLong(args[1]);
            } catch (NumberFormatException numberException) {
                System.err.println("Usage: PrimeExampleMW <candidate>");
                System.err.println(numberException.getMessage());
            }
        }
        try {
            // Create the Master
            ProActiveMaster<FindPrimeTask, Boolean> master = new ProActiveMaster<FindPrimeTask, Boolean>();
            // Deploy resources
            master.addResources(new URL(args[0]));
            // Create and submit the tasks
            master.solve(createTasks(candidate));

            //TODO 3. Wait all results from master
            // Collect results
            List<Boolean> results = master.waitAllResults();

            // Test the primality
            boolean isPrime = true;
            for (Boolean result : results) {
                isPrime = isPrime && result;
            }
            // Display the result
            System.out.println("\n" + candidate + (isPrime ? " is prime." : " is not prime.") + "\n");
            // Terminate the master and free all resources
            master.terminate(true);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            System.exit(0);
        }
    }

    /**
     * Creates the prime computation tasks to be solved
     *
     * @return A list of prime computation tasks

```



```
*/
public static List<FindPrimeTask> createTasks(long number) {
    List<FindPrimeTask> tasks = new ArrayList<FindPrimeTask>();

    // We don't need to check numbers greater than the square-root of the
    // candidate in this algorithm
    long squareRootOfCandidate = (long) Math.ceil(Math.sqrt(number));

    // Begin from 2 the first known prime number
    long begin = 2;

    // The number of intervals
    long nbOfIntervals = (long) Math.ceil(squareRootOfCandidate / INTERVAL_SIZE);

    // Until the end of the first interval
    long end = INTERVAL_SIZE;

    for (int i = 0; i <= nbOfIntervals; i++) {

        //TODO 4. Create a new task for the current interval and
        // add it to the list of tasks
        // Adds the task for the current interval to the list of tasks
        tasks.add(new FindPrimeTask(number, begin, end));

        // Update the begin and the end of the interval
        begin = end + 1;
        end += INTERVAL_SIZE;
    }

    return tasks;
}
```

6.8.3.4. Running the Master-Worker Version

Go to the tutorials/compile directory and type `build[.bat] primes.distributedmw` to compile your code. Then, after a successful compilation, go to the scripts/Primes directory and launch the `distributedMW.[sh|bat]` script with the integer you want to test.

```
distributedMW.[sh|bat] 2147483647
```

Bibliography

- [ACC05] Isabelle Attali, Denis Caromel, and Arnaud Contes. *Deployment-based security for grid applications*. The International Conference on Computational Science (ICCS 2005), Atlanta, USA, May 22-25. . LNCS. 2005. Springer Verlag.
- [BBC02] Laurent Baduel, Francoise Baude, and Denis Caromel. *Efficient, Flexible, and Typed Group Communications in Java*. 28--36. Joint ACM Java Grande - ISCOPE 2002 Conference. Seattle. . 2002. ACM Press. ISBN 1-58113-559-8.
- [BBC05] Laurent Baduel, Francoise Baude, and Denis Caromel. *Object-Oriented SPMD*. Proceedings of Cluster Computing and Grid. Cardiff, United Kingdom. . May 2005.
- [BCDH05] Francoise Baude, Denis Caromel, Christian Delbe, and Ludovic Henrio. *A hybrid message logging-cic protocol for constrained checkpointability*. 644--653. Proceedings of EuroPar2005. Lisbon, Portugal. . LNCS. August-September 2005. Springer Verlag.
- [BCHV00] Francoise Baude, Denis Caromel, Fabrice Huet, and Julien Vayssiere. *Communicating mobile active objects in java*. 633--643. <http://www-sop.inria.fr/oasis/Julien.Vayssiere/publications/18230633.pdf>. Proceedings of HPCN Europe 2000. . LNCS 1823. May 2000. Springer Verlag.
- [BCM+02] Francoise Baude, Denis Caromel, Lionel Mestre, Fabrice Huet, and Julien Vayssiere. *Interactive and descriptor-based deployment of object-oriented grid applications*. 93--102. http://www-sop.inria.fr/oasis/personnel/Julien.Vayssiere/publications/hpdc2002_vayssiere.pdf. Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing. Edinburgh, Scotland. . July 2002. IEEE Computer Society.
- [BCM03] Francoise Baude, Denis Caromel, and Matthieu Morel. *From distributed objects to hierarchical grid components*. <http://proactive.activeeon.com/userfiles/file/papers/HierarchicalGridComponents.pdf>. International Symposium on Distributed Objects and Applications (DOA), Catania, Sicily, Italy, 3-7 November. Springer Verlag. . 2003. Lecture Notes in Computer Science, LNCS. ISBN ??.
- [Car93] Denis Caromel. *Toward a method of object-oriented concurrent programming*. 90--102. <http://citeseer.ist.psu.edu/300829.html>. *Communications of the ACM*. 36. 9. 1993.
- [CH05] Denis Caromel and Ludovic Henrio. *A Theory of Distributed Object*. Springer Verlag. 2005.
- [CHS04] Denis Caromel, Ludovic Henrio, and Bernard Serpette. *Asynchronous and deterministic objects*. 123--134. <http://doi.acm.org/10.1145/964001.964012>. Proceedings of the 31st ACM Symposium on Principles of Programming Languages. . 2004. ACM Press.
- [CKV98a] Denis Caromel, W. Klauser, and Julien Vayssiere. *Towards seamless computing and metacomputing in java*. 1043--1061. <http://proactive.inria.fr/doc/javallCPE.ps>. *Concurrency Practice and Experience*. . Geoffrey C. Fox. 10, (11--13). September-November 1998. Wiley and Sons, Ltd..
- [HCB04] Fabrice Huet, Denis Caromel, and Henri E. Bal. *A High Performance Java Middleware with a Real Application*. <http://proactive.inria.fr/doc/sc2004.pdf>. Proceedings of the Supercomputing conference. Pittsburgh, Pennsylvania, USA. . November 2004.
- [BCDH04] F. Baude, D. Caromel, C. Delbe, and L. Henrio. *A fault tolerance protocol for asp calculus : Design and proof*. <http://www-sop.inria.fr/oasis/personnel/Christian.Delbe/publis/rr5246.pdf>. Technical ReportRR-5246. INRIA. 2004.
- [FKTT98] Ian T. Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. *A security architecture for computational grids*. 83--92. <http://citeseer.ist.psu.edu/foster98security.html>. ACM Conference on Computer and Communications Security. . 1998.
- [CDD06c] Denis Caromel, Christian Delbe, and Alexandre di Costanzo. *Peer-to-Peer and Fault-Tolerance: Towards Deployment Based Technical Services*. Second CoreGRID Workshop on Grid and Peer to Peer Systems Architecture . Paris, France. . January 2006.

- [CCDMCompFrame06] Denis Caromel, Alexandre di Costanzo, Christian Delbe, and Matthieu Morel. *Dynamically-Fulfilled Application Constraints through Technical Services - Towards Flexible Component Deployments*. Proceedings of HPC-GECCO/CompFrame 2006, HPC Grid programming Environments and Components - Component and Framework Technology in High-Performance and Scientific Computing. Paris, France. June 2006. IEEE.
- [CCMPARCO07] Denis Caromel, Alexandre di Costanzo, and Clement Mathieu. *Peer-to-Peer for Computational Grids: Mixing Clusters and Desktop Machines*. *Parallel Computing Journal on Large Scale Grid*. 2007.
- [PhD-Morel] Matthieu Morel. *Components for Grid Computing*. http://www-sop.inria.fr/oasis/personnel/Matthieu.Morel/publis/phd_thesis_matthieu_morel.pdf. PhD thesis. University of Nice Sophia-Antipolis. 2006.
- [FACS-06] I. \vCern\,a, P. Va\vrekov\,a, and B. Zimmerova. "Component Substitutability via Equivalencies of Component-Interaction Automata". To appear in ENTCS. 2006.
- [JavaA05] H. Baumeister, F. Hacklinger, R. Hennicker, A. Knapp, and M. Wirsing. "A Component Model for Architectural Programming". 2005.
- [Fractal04] E. Bruneton, T. Coupaye, M. Leclercp, V. Quema, and J. Stefani. "An Open Component Model and Its Support in Java.". 2004.
- [ifip05] Alessandro Coglio and Cordell Green. "A Constructive Approach to Correctness, Exemplified by a Generator for Certified Java Card Applets". 2005.
- [STSLib07] Fabricio Fernandes and Jean-Claude Royer. "The STSLIB Project: Towards a Formal Component Model Based on STS". To appear in ENTCS. 2007.
- [InterfaceAutomata2001] Luca de Alfaro, Tom Henzinger. "Interface automata". 2001.
- [CCM] OMG. "CORBA components, version 3". 2002.
- [Plasil02] F. Plasil and S. Visnovsky. "Behavior Protocols for Software Components". *IEEE Transactions on Software Engineering*. 28. 2002.
- [Reussnrm] Reussner, Ralf H.. "Enhanced Component Interfaces to Support Dynamic Adaption and Extension". IEEE. 2001.
- [JKP05] P. Jezek, J. Kofron, F. Plasil. "Model Checking of Component Behavior Specification: A Real Life Experience". *Electronic Notes in Theoretical Computer Science (ENTCS)*. 2005.
- [MB01] V. Mencl and T. Bures. "Microcomponent-based component controllers: A foundation for component aspects". APSEC. Dec. 2005. IEEE Computer Society.
- [SPC01] L. Seinturier, N. Pessemier, and T. Coupaye. "AOKell: an Aspect-Oriented Implementation of the Fractal Specifications". 2005.

Index

U

Use cases, 21

A

Active Object
Tutorial, 43

C

CLASSPATH
Libraries, 8

E

Eclipse, 17
Examples, 21
 C3D: A distributed 3D renderer, 21
 Chat application, 32
 Intregal Pi, 34
 Readers/Writers application, 25
 The dining philosophers, 27
 The migratory penguins, 30
 The NBody example, 38

G

Guided Tour, 17

I

IDE, 17
Installation, 7
 Download, 7
 Steps, 7

J

Java Policy File
Simple example, 9

L

Log4j
Simple example, 10

T

Troubleshooting, 13
Tutorial, 43
 Active Objects Lifecycle, 47
 Agent Synchronisation, 56
 Groups of Monitoring Agents, 67
 Monitoring Agent As A Web Service, 73
 Monitoring Several Computers Using Migration, 61
 Primality test, 76
 Using Master-Worker API, 89
 Remote Monitoring Agent, 51
 Simple monitoring agent, 43