An open integration framework
for high performance middleware
May 22, 2012 — Version 0.5.0

Installation guide

&

User's manual

Alexandre Denis — Alexandre.Denis@inria.fr
Christian Pérez — Christian.Perez@inria.fr
Christophe Frézier — christophe.frezier@labri.fr

$Revision: 4138 $ $Date: 2011-07-04 14:01:39 +0200 (lun. 04 juil. 2011) $

Copyright ©2002 INRIA and the University of Rennes 1.
Written by Alexandre Denis, Christian Pérez and Christophe Frézier.

Padico is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program 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 General Public License for more details, given in appendix A page 57 of this manual.

This Padico package contains external software packages subject to copyright. PM^2 and Expat are required by PadicoTM; omniORB and MPICH have been modified for use within PadicoTM. They are distributed in the same bundle as Padico for convenience; they remain the copyright property of their respective authors. These software packages are stored in `PadicoTM-0.5.0/opt/`.

PM^2 — PM^2 is distributed under the terms of the GNU General Public License version 2. The list of authors can be found in the file `PadicoTM-0.5.0/opt/pm2/AUTHORS`. The PM^2 version supplied with Padico has been tweaked for special use in PadicoTM.

Expat 2.0 — Copyright Thai Open Source Software Center Ltd and Clark Cooper. Expat is distributed under the term of its own license as found in `PadicoTM-0.5.0/opt/libexpat.org/COPYING`

omniORB 4 — Copyright AT&T Cambridge. omniORB programs are distributed under the terms of the GNU General Public License version 2, omniORB libraries are distributed under the terms of the GNU Library General Public License. The list of authors can be found in the file `MiddlewareServices/omniORB-4.0/package/omniORB-4.0.6.tar.gz (./CREDITS)`. The omniORB version supplied with Padico has been modified for use within PadicoTM.

MPICH 1.2 — Copyright University of Chicago and the Mississippi State University. The license and authors are given in `MiddlewareServices/mpich-1.1.2/package/mpich-1.2.4-padico-svn20060411.tar.gz (./COPYRIGHT)`. Additionally, we provide a Madeleine driver for MPICH (`ch_mad`). Original `ch_mad` for MPICH by Guillaume Mercier and Loïc Prylli, modified by Alexandre Denis for use within PadicoTM.

Warning

This document is based on an old version of PadicoTM and is mostly outdated. For up-to-date information, please see `README` files in PadicoTM directories from `svn` trunk.

Contents

Chapter 1

What is Padico?

 adico is an environment designed for high performance parallel computing, distributed computing, and software components. It is composed of four entities:

PadicoTM is the nickname for Padico Task Manager, the Padico high performance runtime layer. PadicoTM provides the user with fast middleware systems such as CORBA, MPI and Java ™ implementations over high performance networks such as Myrinet, SCI or VIA and user-level multi-threading. Padico version 0.5.0 comprises CORBA and MPI implementations. Java support is experimental.

PadicoControl is a set of graphic and command-line tools for deploying PadicoTM, and remotely loading, running, and monitoring Padico applications.

***my*CORBA** is a set of tools which hides the differences between ORB vendors for compiling and running CORBA applications.

PaCO++ is the new generation of parallel CORBA objects framework. It adds portability to PaCO. PaCO++ is not included in Padico version 0.5.0 .

1.1 Philosophy of PadicoTM

PadicoTM is the runtime environment for Padico. It is composed of a *core* which provides a high-performance framework for networking and multi-threading, and *services* plugged into the core. The PadicoTM core aims at making the different services running at the same time run in a *cooperative* way rather than *competitive*. Basically, every piece of code in Padico is embedded into a *module*. For example, binary code is embedded into binary libraries ("`.so`" files) and a description file (written in XML). Modules may be loaded, run and unloaded dynamically, on one node, on all nodes, or on a *group* of nodes.

Services are modules *plugged* into the core. Usually, applications do not directly use the PadicoTM core specific interface. Instead, they use a standard API provided as a Padico service. The PadicoTM core brings many features to the services. For example, thanks to the PadicoTM core, services:

- take benefit from high-performance networks (Myrinet, SCI, VIA, INFINIBAND) where they are available;
- use the Marcel high-performance multi-threading system;

- multiplex their accesses to the network, without lowering the performance;
- are usable at the same time, in the same process;
- are dynamically loadable and unloadable.

The core itself is composed of three modules: Puk (a nickname for *Padico μ-kernel*) is the foundation module. Its task is to manage modules (loading, running and unloading). The ThreadManager manages multi-threading in a coherent way, it provides hooks for periodic operations, and manages queues of I/O operations so that they do not block the whole process. The third module of the PadicoTM core is called NetAccess. It multiplexes the network accesses so that several modules can use networks that require exclusive access otherwise. Hence different middlewares (CORBA, MPI) can efficiently share the same process and the same network without disturbing each other.

1.2 Understanding PadicoTM

In order to reach high performance, PadicoTM redefines some known concepts. The steps involved in using PadicoTM are the following:

Framework installation — The first step is the installation of the framework. It is usually done once for a given platform.

Core & services setup — This step aims at building a specific flavor of the runtime layer. A flavor is composed of: SMP/mono, network type, debug/optimize flags. It should be done once for each flavor, and in particular once for each network type. As a flavor is determined by the included core, it is often referred to as a "core flavor".

User module setup — User applications are stored in modules. Modules are built through a "linker-like" command.

Boot — To be able to load and run user modules, you should first launch the processes. This step is called "boot" in PadicoTM. It starts processes on every nodes which are about to be used by user modules.

Execution steering — When the processes are started, you can then load, run and unload modules in them. This is usually remotely done through the use of a steering tool of PadicoControl.

1.3 Getting help.

If you have questions about Padico or have found a bug, please send an electronic mail to `<padico-users@listes.irisa.fr>`. Users are encouraged to subscribe to the Padico users mailing list at: `https://www.irisa.fr/wws/info/padico-users/` (notice the the `https`, not `http`).

Padico is a research prototype, thus is subject to quick changes. For up-to-date information about Padico, please visit the Padico web site at `http://runtime.bordeaux.inria.fr/PadicoTM/`.

Chapter 2

Installation and configuration

2.1 System requirements

The supported systems are PC/Linux, Sparc/Solaris, SGI/Irix and PowerPC/Mac OS X systems. The supported networks are Myrinet, Myrinet-2000, SCI, VIA, Quadrics, INFINIBAND, and all networks for which the OS supplies TCP/IP. Padico has been tested successfully on:

- x86 and x86_64, Debian GNU/Linux 2.6.x — random bugs in dynamic library operations have been reported with glibc 2.2; it is fixed in glibc 2.3 and later;
- Sun Ultra-1 (sparcv9), Solaris 5.6 & 5.7; Sun Ultra-2, Solaris 5.8;
- SGI Onyx 2 (r10000), IRIX64 6.5;
- Apple Xserve G4, Mac OS X 10.2 (Darwin 6.6).

It should work on any other Unix-like system supporting dynamic libraries in ELF format, provided that the software requirements are fullfilled. There is no Windows support.

2.2 Software requirements

Compiler. PadicoTM is based on PM^2 which requires the GNU C compiler. Therefore, only `gcc` is supported. It has been successfully tested with `gcc 3.2` through `4.4`. The versions priori to `3.1` are not supported anymore.

The default behavior of Padico compilation is to use `gcc` and `g++` command lines to invoke the C and C++ compilers. To force another command line, set the `CC` and `CXX` shell environment variables before installing Padico or use the interactive installation procedure as described in Section 2.3.1

Make. Padico requires GNU `make 3.81` or later. It may be required to set the environment variable `MAKE` to the correct value if the command is not `make` (e.g. on some systems, GNU make is `gmake`), or use the interactive installation procedure.

Network drivers. The supported high-speed networks are Myrinet, Myrinet-2000, Quadrics QsNet-II, INFINIBAND and SCI. Myrinet and Myrinet-2000 require MX or GM. QsNet requires Elan 4. Infiniband requires OpenFabrics or OpenIB verbs (1.0 or later– 1.0-rc are not supported). SCI networks require SISCI. You must provide the path to MX and SISCI drivers through the variables `MX_DIR` and `SISCI_PATH` – the defaults are `MX_DIR=/opt/mx` and `SISCI_PATH=/usr/local/sisci` and can be overridden by setting the environment variable, or interactively as described in Section 2.3.1.

Other. The PadicoTM core uses PM^2 and the Expat XML parser. Both are supplied in the Padico package and are automatically built with PadicoTM. If you get weird errors with PM^2 makefiles and shell scripts, try to upgrade GNU textutils to version 2.0 or later.

Optional. Some Padico parts have specific software requirements but can be deactivated. Their building is optional. Check below if you plan to use these services.

CORBA service requires `python 1.5.2` or later. Previous versions won't work. You need the runtime environment *and* the developers package (including for example the `python.h` file).

`padico-run` is a CORBA remote-control tool used to steer the processes. It requires a *my*CORBA-supported ORB. See the details in Section 4 about *my*CORBA and Section 3.1.2 about `padico-run` and the CORBA gatekeeper.

`padico-control` is a graphical remote-control tool used to steer the processes. It requires a Java virtual machine with *Swing* (Java 1.5 or later).

`padico-nsconfig` is a graphical configuration tool used to assemble communication components. It requires a Java virtual machine with *Swing* and templates support (Java 1.5 or later).

`G5k` is a Padico module which captures the Grid5000 Topology. It requires two libraries, libcurl and libjansson.

`Topo-Print` is a Padico module which displays the current known topology. It requires graphviz libraries to display topology in dot format.

`Control-SSH2` is a Padico module which establishes control channel and can cross gateway with sshd service. It requires libssh library.

Summary

Software	Padico package	Requirement
C compiler	PadicoTM	Ok: gcc-3.2, gcc-3.3, gcc-3.4, gcc-4.0; doesn't work: gcc-2.95, gcc-3.0, gcc-3.1, non-GNU
C++ compiler	CORBA for PadicoTM	Ok: g++-3.2, g++-3.3, g++-3.4, g++-4.0; doesn't work: g++-2.96, g++-3.0, g++-3.1
Python	CORBA for PadicoTM	python 1.5.2 or later (runtime *and* developers package)
make	all	GNU make 3.81 or later
Java	PadicoControl GUI	Sun JDK 1.5 or later; should compile with `jikes`; won't work with non-Sun virtual machine
ORB	PadicoControl command-line	MICO 2.3.x, ORBacus 4.0.x, OmniORB 3.0.x or OmniORB 4.0.x (see section 4.1)

2.3 Installation and configuration

Files. The Padico packages are contained in a single archive named `PadicoTM-0.5.0.tar.gz`. Un-gzip and un-tar the archive. You should get a directory named `PadicoTM-0.5.0` with the following files and directories:
`README` — a short guide for Padico installation
`RELEASE_NOTE` — a short note about what is contained in the package
`COPYING` — the license for Padico
`VERSION` — Padico version number, namely `0.5.0`
`doc/` — this documentation
`padico-install` — the quick install wizard
`Makefile` — admin makefile, normally not needed by regular users
`admin/` — common parts used by each Padico package
`myCORBA/` — the *my*CORBA package; see Section 4
`PadicoTM/` — the PadicoTM package; see Section 3
`PadicoControl/` — the PadicoControl package; see Section 3.1.3

This section describes the quick installation method. An assistant guides you through the installation and configuration steps. This method is recommended for most users. Advanced users may prefer to be able to configure and install each package separately, or to have fine control over the installation process. Advanced installation and setup is described in Chapter B.1. If you encounter an error while installing Padico, please see Section 2.4.

2.3.1 Framework installation

In the Padico source directory `PadicoTM-0.5.0`, first type `make config` then type `./padico-install` and answer the questions. You will be asked:

- for the installation directory. Type the installation directory path where to install Padico, or simply accept the default path by leaving the answer empty;

- if you want the whiptail or text installation. If you have any problems with whiptail, then choose the text installation;
- for some environment variables needed for setup: C/C++/Java compilers, Myrinet, BIP and SCI driver location. To accept the default choice (shown in square brackets), simply type *Return*;
- for build options. Regular users will want option 1 ("optimize") which is the default. Developers may want option 2 (with debug symbols and trace messages);
- for the Padico packages to build. All the packages you want to use must have to be build. If unsure, accept the default choice.

Then depending on your input, *my*CORBA, PadicoTM and PadicoControl are automatically configured, built and installed. The output messages of all commands are stored in log files.

If you encounter a problem, have a look at Section 2.4. The most common problem is an error in *my*CORBA setup; in this case, carefully read Section 4.1.

2.3.2 User environment

Before going further, Padico needs some customization in the user's environment. For C-style shells (`csh`, `tcsh`), add the following line into your `.cshrc` file:
`source <padico_root>/etc/padico-setup.csh`
For Bourne shells (`sh`, `bash`, `ksh`), add the following line into your `.profile` or `.bashrc` file:
`. <padico_root>/etc/padico-setup.sh`
where <*padico_root*> is the directory where you installed Padico.

Please see the manual of your shell if you are not sure in which file you should add one of these lines. It *must* be in an initialization file and cannot be setup by hand in each terminal. In particular, it should be automatically executed upon non-interactive `rsh`. Do not forget to restart your shell before the core and service setup.

2.3.3 Core & services setup

Core flavors. PadicoTM core & services setup consists in creating cores for specific network- and processors- configurations. To create a new flavor which contains a PadicoTM core and the associated services, the quick and easy way is to use `padico-core-assistant`. Advanced core configuration may require custom setup as described in Section B.2.

Creating a core with the core assistant: `padico-core-assistant`. Before you start, don't forget that you should have sourced <*padico_root*>`/etc/padico-setup.csh`. Start the script `padico-core-assistant`. You will be asked for core parameters (mono- or multi-processor, the PM^2 flags, the NetSelector and Controllers, network type), a symbolic name for the core, and whether you want to build optional services such as CORBA and MPI. Building CORBA and MPI can be very long (up to 30 minutes) so be patient!

Creating a CORBA startup module: `padico-corba-startup`. When you *boot* PadicoTM (see Section 1.2 "Understanding PadicoTM" for our definition of *boot*), you actually start processes with some *pre-loaded* modules in them. This bundle of modules must at least contain a core. It should contain a module to remotely steer the processes. As CORBA remote control is heavily used in PadicoTM, we suggest you build a startup package that contains a PadicoTM core, an ORB, and some configuration attributes for CORBA (e.g. a reference to the CORBA NameService). It may be automatically done through the use of `padico-corba-startup`. This script builds a startup package suitable for CORBA use and starts a CORBA NameService if none is detected by *my*CORBA. For example, if you have built a core called "mono-mx", then type:

```
padico-corba-startup --padico-core mono-mx -iloadCORBA
```

Ready for the tutorial! The following sections are useful for troubleshooting, for advanced installation and configuration mechanisms, and custom core and services setup. Users who have chosen the quick install as recommended (i.e. `padico-install`, `padico-core-assistant` and `padico-corba-startup`) can skip these sections and go directly to the PadicoTM tutorial in Chapter 3.

2.4 Troubleshooting

This section describes the solution to the most common problems that may arise when installing and configuring PadicoTM.

"No ORB detected. Installation stopped." `padico-run` requires an *my*CORBA-supported ORB. See Section 4.1 for more information on *my*CORBA setup and requirements.

Error while compiling PadicoTM code. This is a well-known symptom of a non-supported compiler. See the software requirements section in this document. Check your `gcc` version with:

```
gcc --version
```

Syntax errors in makefiles. Padico makefiles require GNU `make` version 3.81 or later (`http://www.gnu.org/software/make/`), and GNU textutils version 2.0 or later (`http://www.gnu.org/software/textutils/`). Check with:

```
make --version
# should be at least: GNU Make version 3.81
sort --version
# should be at least: sort (GNU textutils) 2.0
```

SISCI not found. If SISCI is not installed in `/usr/local/sisci/`, you must specify where it is installed through the use of the environment variable `SISCI_PATH` *before* any other installation step since PadicoTM is statically linked against it. If it does not work, it may be required to adapt PM^2 scripts in `PadicoTM-0.5.0`

`/PadicoTM/opt/pm2/mad2/config/options/00sisci.sh`. SISCI drivers are likely to have a different (incompatible) source tree.

Python-related error while compiling CORBA. Python version 1.5.2 or later is required for the CORBA service. You need the runtime environment *and* the development kit. The environment variable `PYTHONHOME` should point to where Python is installed. Unless you know what you are doing, we advise no to set the `PYTHONPATH` variable. There is a bug in the way OmniORB uses Python. It assumes `prefix` and `exec_prefix` to be the same, thus a multi-platform installation will not work.

"*xxxxx* requires PADICO_CORE" or other message relative to PADICO_CORE. Some Padico commands are core-specific and thus must know to what core flavor the operation is related. Either set the `PADICO_CORE` environment variable to give the default core name, or give a `--padico-core <core>` argument to the command. More details in Chapter B.2.

Error while building the "ORB" service on Redhat Linux 7.0 or 7.1. Some Redhat distribution come with a non-standard version of the GNU compiler, namely 2.96. This compiler is not able to build PM^2 and should not be used. Even if you specify an alternate (valid) compiler, it is still possible that OmniORB forces `/usr/include` in the include path because of Python, thus using wrong include files. If this would happen, a possible fix (but not recommended) consists in editing the `PadicoTM/opt/omniORB-3.0.2/src/tool/omniidl/cxx/dir.mk` file and control "by hand" the lines dealing with `PYPREFIX`. A better solution is to upgrade to gcc 3.4 or later, and to completely get rid of the infamous gcc 2.96.

padico-core-assistant command not found. After the framework installation (`padico-install`), you have to customize your environment (see Section 2.3.2). You must *restart your shell* or open a new shell for this modification to be taken into account. To check that everything is ok, you may type:

```
echo $PADICO_ROOT
# should print where Padico is installed
```

pm2-config not found in PATH – please add $PM2_ROOT/bin in your PATH Either you are trying to use your existing PM^2 installation (and then you should know what you are doing), or you are re-installing Padico over an existing installation of Padico. In this later case, delete your old Padico installation, and restart a new installation in a clean shell (i.e. no `PADICO_ROOT` nor `PM2_ROOT`).

Problem not listed here? If the problem you encounter is not listed here, then you can get help from the Padico users mailing-list. See `https://www.irisa.fr/wws/info/padico-users/` to subscribe to the `<padico-users@listes.irisa.fr>` mailing-list. Do not forget to include in your e-mail the exact error message, your hardware description, your OS name and version, and the Padico version number.

2.5 Middleware services setup

How to build a middleware service with Padico Padico can support some external middleware services, in order to provide their services, using the benefits of Padico. These services cannot be used with a normal compilation. They have to be builded with Padico compiler and linker.

This services are all listed in *$PADICO_ROOT*/MiddlewareServices/. This folder contains a directory for each middleware, and some additional scripts designed for the download and the installation of this middlewares.

In order to install a service, you first have to download and unpack it. This can be done using the command:

```
./package-admin download
```

Then you can install the service in a dedicated directory with the command:

```
./package-admin import
```

The last step, the compilation of the service, is done from the dedicated-service directory using the command:

```
make
```

Make sure your `PADICO_CORE` environment variable is set before compiling the service.

The available middleware services Here is a list of the middleware services available with Padico:

- certi-3.0. CERTI is a runtime infrastructure for distributed discrete event simulations developed at ONERA. It provides a set of services used by simulators to interoperate (such as object management, time management, optimization services, etc.)
- juxmem-c-0.2 This software prototype is based on the concept of data sharing service for grid computing, as a compromise between DSM systems and P2P systems. The main contribution of this service is to decouple data management from grid computation, by providing location transparency as well as data persistence in a dynamic environment.
- mome The MOME database provides a knowledge exchange platform about measurement tools and measurement data.
- mpich-1.2.x MPICH2 is an implementation of the Message-Passing Interface (MPI). The goals of MPICH2 are to provide an MPI implementation for important platforms, including clusters, SMPs, and massively parallel processors. It also provides a vehicle for MPI implementation research and for developing new and better parallel programming environments
- omniORB-4.0 omniORB is a robust high performance CORBA ORB for C++ and Python.

- apr-1.2.1 The mission of the Apache Portable Runtime (APR) project is to create and maintain software libraries that provide a predictable and consistent interface to underlying platform-specific implementations.
- gSOAP-2.x The gSOAP Web services development toolkit offers an XML to C/C++ language binding to ease the development of SOAP/XML Web services in C and C/C++.
- jxta-c-2.3 JXTA technology is a set of open protocols that allow any connected device on the network ranging from cell phones and wireless PDAs to PCs and servers to communicate and collaborate in a P2P manner. JXTA peers create a virtual network where any peer can interact with other peers and resources directly even when some of the peers and resources are behind firewalls and NATs or are on different network transports.
- mico-2.3 MICO implements the CORBA standard. It only relies on C++, the standard Unix API and non-proprietary libraries.
- yampii-0.3 YAMPII is an implementation of the Message-Passing Interface (MPI).

Chapter 3

PadicoTM tutorial

3.1 Getting started: a step by step example

3.1.1 A PadicoTM-flavored "Hello world!"

Let us write step by step a basic "Hello world" example. We create a file named `HelloWorld.c`. We want to declare that this file is a Padico module. This is "automatically" done by the following two lines:

```
#include <Padico/Module.h>
PADICO_MODULE_DECLARE(HelloWorld);
```

Then we define the actions of the module. A module may declare an "init" function, a "run" function, and a "finalize" function, called respectively when the module would be loaded, run, and unloaded. These functions are declared with macros defined in `Padico/Module.h`. For example, we declare here that the "run" function for our Hello world is `hello_main`:

```
PADICO_MODULE_RUN(hello_main);
```

Finally, we implement the `hello_main` function. It should have a `main`-like header.

```
int hello_main(int argc, char**argv)
{
  padico_print("Hello world!\n");
  return 0;
}
```

We use the `padico_print` function which is like the well-known `printf` function except that its output is captured by Padico console. This function, like most of the Padico basic function, is defined in `Puk.h`, thus we add this line in the headers:

```
#include <Padico/Puk.h>
```

We put it altogether on Figure 3.1. You can find the examples of this tutorial in the directory `PadicoTM-0.5.0/PadicoTM/Examples/Tutorial/`.

Now, let us compile this file. We suppose that we compile the example for a core (strictly speaking: a PadicoTM core flavor) named `mono-mx`:

```
/* Padico tutorial 1: "Hello world"
 * available in: PadicoTM/Examples/Tutorial/01-HelloWorld/
 * author: Alexandre Denis
 */

#include <Padico/Puk.h>
#include <Padico/Module.h>

PADICO_MODULE_DECLARE(HelloWorld);
PADICO_MODULE_RUN(hello_main);

static int hello_world_count = 0;

int hello_main(int argc, char**argv)
{
  padico_print("Hello world! [%d]\n", hello_world_count);
  hello_world_count++;
  return 0;
}
```

Figure 3.1: The whole `HelloWorld.c` example

```
setenv PADICO_CORE mono-mx
padico-cc -c HelloWorld.c
padico-mkmod HelloWorld.o -o HelloWorld
```

The on-screen output should looks like this:

```
-----------------------------------------
<!-- Padico module description. This file is HelloWorld.xml
     Automaticaly generated by padico-mkmod - do not edit!
     Built on: Linux paraski 2.2.18 #1 SMP Fri Aug 24 10:12:15 CEST 2001 i686 unknown
  -->
<defmod name="HelloWorld" driver="binary">
  <unit>libHelloWorld.so</unit>
</defmod>
-----------------------------------------
```

Don't worry about this for the moment. It is the XML file which describes the module.

3.1.2 Managing PadicoTM in command-line: `padico-boot` and `padico-run`

Boot PadicoTM: `padico-boot`

The core and the services are now ready for startup. The "HelloWorld" module is built. Building a core, additional services, and startup module has to be done only once for each core flavor. Let us assume that we want to boot the startup module `mono-mx` previously built in section 2.3.3 on 2 machines called `paraski43` and `paraski44`. We want each machine to open a new console. The command line is:

```
padico-boot -c mono-mx paraski43 paraski44
```

If you have enabled trace at the configure step, then you probably want to turn trace off. The command line without trace is:

```
padico-boot -c mono-mx paraski43 paraski44 -- --padico-trace-off
```

All the parameters after the -- are passed to Puk, not to the user's modules.

If we do not need one console per node, we can type:

```
padico-boot mono-mx paraski43 paraski44
```

If we need to specify different init sequences on different nodes, we can type the -i<init> argument. This add file <init> to startup sequence. <init> must be an absolute file name or a name without path that will be search for in $PADICO_ROOT/etc/init.d/. This argument can be given several times:

```
padico-boot mono-mx-with-CORBA paraski43 -iNetSelector-preset-multi-cluster
-i$PADICO_ROOT/etc/init.d/load-InfinibandVerbs.xml
%$
```

Be aware that the order of the differernt -i arguments can be important; for example, we have to load the NetSelector before loading something else that uses networks. Here is an example of <init> file (load-InfiniBandVerbs.xml):

```
<?xml version="1.0"?>
<sequence>
  <load>InfinibandVerbs</load>
</sequence>
```

This -i option can be limited to an host, or a list of hosts, by typing:

```
padico-boot [...] --init:paraski43=<init>
padico-boot [...] --init:paraski43,paraski44=<init>
```

Many other options can be limited using this method, as shown below:

```
padico-boot [...] --console:paraski43
padico-boot [...] --console:paraski43=xterm
padico-boot [...] --debug:paraski43
padico-boot [...] --log:paraski43
padico-boot [...] --log:paraski43=/tmp/padico-log/
padico-boot [...] --trace:paraski43
padico-boot [...] --trace:paraski43=/home/user/padico-trace-file.xml
```

Hint: this "machine" specification is applicable to -D. For instance it's possible to launch:

```
padico-boot [...] -DPADICO_LAUNCH:paraski43=/bin/echo
```

It works with all -D specifications.

**** Warning ** : in this exemple, we used paraski43 and paraski44. These machine names is actually compared to the answer of the machine to the command "hostname". Be sure the value returned by the system is the same as yours.**

Tip: `padico-boot` with a job manager. If your cluster is managed by a job manager (batch), you have to submit `padico-boot` to the job manager and then get the hosts list form the job manager. For example, to boot up 4 PadicoTM nodes with LSF, a command line could look like this:

```
bsub -I -n 4 padico-boot -c mono-tcp '$LSB_HOSTS'
```

Load, run and unload a module with the CORBA remote-control: `padico-run`

After PadicoTM has been booted, an user's module, such as HelloWorld in the previous example, can be loaded. In this example, we use `padico-run` from the PadicoControl package. The following command line will load `HelloWorld` on all booted nodes:

```
padico-run -l HelloWorld
```

Starting the execution of the user's module, that have been previously loaded, is performed thanks to the following command line:

```
padico-run -r HelloWorld
```

If we want to restart `HelloWorld` only on `paraski43`:

```
padico-run -r HelloWorld paraski43
```

The module is restarted (i.e. its "run" method is called in a new thread) and remains loaded in memory.

User's modules can be unloaded from PadicoTM using the following command line:

```
padico-run -u HelloWorld
```

Finally, PadicoTM processes can be killed thanks to the following command line:

```
padico-run -k
```

`padico-launch`

This command is used when we don't want to modify our program to execute it with the benefits of PadicoTM. It allows the execution of the program, with PadicoTM as preload. Then, in order to lauch the "foo" program:

```
padico-launch foo arg1 arg2 ...
```

3.1.3 Managing PadicoTM with a GUI: `padico-control`

The PadicoControl GUI is a user-friendly alternative to the command-line tools. To start the PadicoControl GUI, type: `padico-control`. You should get something like Figure 3.2 (but with all fields empty).

Booting processes: the "Boot" panel — The "Boot" panel (the left part of the `padico-control` interface) aims at managing `padico-boot` commands. Use the upper part to build a machine list on which to boot PadicoTM processes. The "Proxy Command" aims at launching a `padico-boot` command remotely, for example on a cluster front-end while `padico-control` is running on your office workstation; leave it blank if you want `padico-boot` to be executed on the local machine. The "Core" field is mandatory. It is the name of the core that will be passed to `padico-boot`. Puk parameters, other padico-boot parameters can also be added. The "Rdv Client", "Rdv Server" and "Contact padico-control" are used to modify the way `padico-boot` is connected to the other nodes (see Section 3.5). You can also choose to launch the different nodes into a GDB process with the

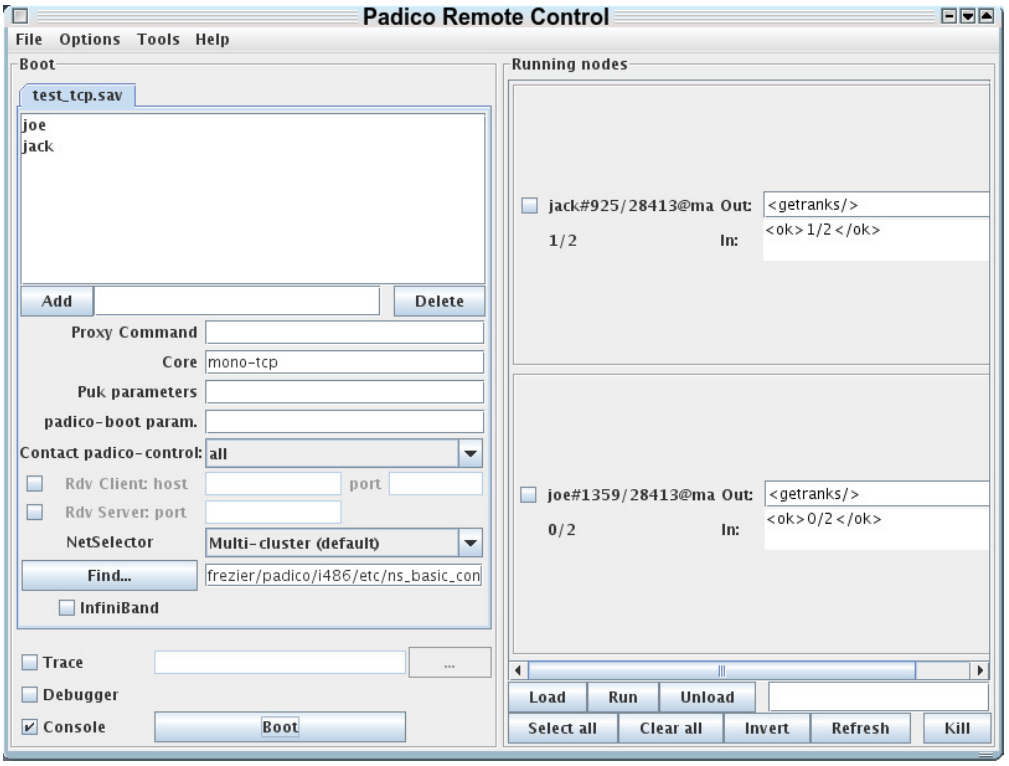

Figure 3.2: Panel of `padico-control`

"Debugger" box, or ask for traces if your Padico and/or core is built as a debug version (see Section 5.1.6 for help on traces). You can load and save such configurations with the "Load" and "Save" entries of the "File" menu. To actually boot, press the "Boot" button. The standard input/output are captured by the `padico-control` console. You also have to choose the NetSelector to use, see Section 3.2 for more details.

Build the list of nodes you want PadicoTM to run on, type `mono-mx` (or any suitable core name) in the "Startup Module" field, select the "Console" mode, and press "Boot". One console per node should appear; when the message `PadicoTM: ***** Padico Task Manager ready *****` appears in the consoles, the processes are ready to accept commands.

Execution steering: the "Running nodes" panel — Once the processes are started, they contact the remote control that started them. When they subscribe for remote control, `padico-control` automatically show them into the "Running nodes" panel, the right part of the `padico-control` interface (Figure 3.2). There is one sub-panel per running node. The bottom right text field, next to the "Unload" button aims at sending commands to the nodes: select the nodes which must receive the command (using the checkboxes on the left, the "Select all", "Clear all" and "Invert" buttons), then press the "Kill" button to kill the processes, or type a module name in the text field and press "Load", "Run" or "Unload" to load, run or unload this module on all checked nodes. Input and output for each node is monitored in text fields; advanced users can directly send XML commands to the nodes.

Type `HelloWorld` in the bottom-right text field, select the left boxes of the nodes you want to run the "Hello world" example on, and press "Load". They should answer `<ok></ok>`. This XML messages means that the operation is successfull. Try to press "Load" again: there should be an error; no harm is done, PadicoTM detects that this module is already loaded. Try to press "Run": the nodes' windows should display "Hello world!". Press "Unload" then check all processes and press "Kill". It terminates the PadicoTM session. A final dialog box sums up the result of the `padico-boot` command; it may help to diagnose boot errors.

Tip: the return code may seem erroneous if you use a non-empty proxy command. Actually, it is the return code for the contact command itself, not for the remote `padico-boot` command. Typically, `ssh` forwards the return code from the remote process, whereas `rsh` does not report anything about the remote process.

The "Active sessions" panel is not used yet. It is reserved for future use.

3.2 NetSelector

Choosing how to assemble adapters, based on the required interfaces and some set of preferences, can be done easily using the NetSelector module.

3.2.1 NetSelector configuration

It is possible to control, or even to force, the decisions made by the NetSelector. This is done by providing instructions through an XML file. The NetSelector configuration provides two modes: the NetSelector-basic which reads an XML file and chooses what is described in this

file, and the NetSelector-besteffort which tries to resolve the requests the best possible way, and does not need an XML file.

3.2.2 NetSelector and `padico-boot`

You can select the type of NetSelector you want to run by adding an argument to `padico-boot`. The differents modes are the following:

- Multi-cluster (default) : `-iNetSelector-preset-multi-cluster` This option is chosen when you want to use Madeleine and inter-cluster capabilities for this set of nodes. As it only uses NetSelector-inet-default and NetSelector-Besteffort, it does not need a configuration file. The module Control-Router is loaded, in order to route the inter-cluster communications.
- Grid : `-iNetSelector-preset-grid` The grid option provides full-routing options. Madeleine is enabled. It uses the NetSelector-basic prior to NetSelector-inet-default and NetSelector-besteffort. A XML configuration file for the NetSelector can be provided.
- Single-cluster : `-iNetSelector-preset-single-cluster` This option is chosen when you do not want any routing options, only Madeleine can be used here. As it only uses NetSelector-inet-default and NetSelector-Besteffort, it does not need a configuration file.
- Router (single node) : `-iNetSelector-preset-router` This option is useful when you want to create a full routing capable node, which is not in a set of Madeleine nodes. A XML configuration file can be provided.

To specify a XML NetSelector file, use the `-DPADICO_NSCONFIG="path"` argument.

NetSelector-basic

The NetSelector-basic uses an XML file to choose how to communicate on the networks. This configuration file describes all the rules that can be used to perform communication, depending on the topology, and so on... This file can be written through the `padico-nsconfig` interface, but we'll describe the structure of this file just below: The document contains 3 parts: first is the description of the topology, second is the description of the adapters that we want to use, and the third part is the description of what adapter we have to use in a particular situation.

```
<?xml version="1.0"?>
<!DOCTYPE NS:rule-set SYSTEM ".../<arch>/etc/NetSelector.dtd">
<NS:rule-set xmlns:NS="http://padico.org/PadicoTM/NetSelector">
    <NS:group id="GROUP_ID">
        <NS:TYPE ID="ID_MACHINES"/>
    </NS:group>
    [other groups definitions...]
    <NS:assembly id="ASSEMBLY_ID">
        <NS:adapter id="ADAPTER_ID" name="ADAPTER_NAME"/>
```

```
        <NS:adapter id="ADAPTER_ID_2" name="ADAPTER_NAME_2">
            <NS:uses adapter-id="ADAPTER_ID" iface="IFACE_USED"/>
        </NS:adapter>
        <NS:assembly-provides adapter-id="ADAPTER_ID_2" iface="IFACE_PROVIDED"/
    </NS:assembly>
    [other assemblies definitions...]
    <NS:rule>
        <NS:assembly ref="ASSEMBLY_ID"/>
        <NS:target kind="TYPE" profile="PROFILE">
            <NS:group ref="GROUP_ID"/>
        </NS:target>
    </NS:rule>
    [other rules...]
</NS:rule-set>
```

First we find the topology definitions. This describes some groups that can be used later in the rule definition. Here we describe a set of machines that will be named GROUP_ID. We can describe them through some different ways: TYPE can be `node` or `host`. ID_MACHINES is a description of the nodes/hosts contained in the group, as machines names. After defining all the groups that we need, we describes the assemblies (adapters) that we want to use. Each assembly has an ASSEMBLY_ID, which can be used for rules definitions, and is composed of many adapters. Each adapter is designed by his name and an ADAPTER_ID, using a "<NS:adapter [...]>" line. Some adapters needs to be connected to another (because they use an interface to provides its service) and that has to be described: then we have to use "<NS:uses [...]>" line. This need the ADAPTER_ID of the connected adapter and the interface (IFACE_USED) that connects both adapters. Third type of assembly description, the "<NS:assembly-provides [...]>" line let the NetSelector know what interface is provided by this assembly. Last, we have to describe the usage of the assemblies. That's a rule definition. It contains an assembly (referred by its ADAPTER_ID) and some target definitions. A target is defined by a type (clique, multi-partite, or arrow), a profile, and some nodes/hosts. Here we can use the topology definitions.

Here is a real example:

```
<?xml version="1.0"?>
<!DOCTYPE NS:rule-set SYSTEM ".../<arch>/etc/NetSelector.dtd">
<NS:rule-set xmlns:NS="http://padico.org/PadicoTM/NetSelector">
    <NS:group id="paravent-IB">
        <NS:host name="paravent[0-6]*.rennes.grid5000.fr"/>
    </NS:group>
    <NS:group id="paravent">
        <NS:group ref="paravent-IB"/>
        <NS:host name="paravent*.irisa.fr"/>
    </NS:group>
    <NS:assembly id="s-0">
        <NS:adapter id="0" name="InfinibandVerbs"/>
        <NS:assembly-provides adapter-id="0" iface="PadicoSimplePackets"/>
    </NS:assembly>
    <NS:rule>
```

```
        <NS:assembly ref="s-0"/>
        <NS:target kind="clique" profile="default">
            <NS:group ref="paravent-IB"/>
        </NS:target>
    </NS:rule>
</NS:rule-set>
```

This XML document contains several elements or definitions.

The first type of definition is a group definition: it defines a group named *paravent-IB* containing a set of hosts whose names are defined by the expression `paravent[0-6]*.rennes.grid5000.fr`. A second group is defined, *paravent*, containing the first group, and a set of hosts also defined through an expression. These groups can then be used in the following definitions.

The second type of definition is an assembly definition: it defines the assembly *s-0* using the adapter named *Infiniband*, and providing the interface *PadicoSimplePackets*.

The third type of definition is a rule definition: it defines a link between an assembly and a target. A target is defined through a type and a profile. Several types are available: "clique", "multi-partite", or "arrow" (only one-way link, like ssh). The "default" profile is used (see below). The profile used is "default" for this rule (see below).

NetSelector-besteffort

The second type of NetSelector is the NetSelector-besteffort. This one doesn't need a XML configuration file. The goal of this NetSelector is to provide answer for all possible communications. It has a large set of rules that he can use to provide communications. It chooses automatically an assembly that enables the communication that the module is asking for, depending on the topology and possibilities. It can be used as a "default" set of rules.

Profiles

Many applications need to provide different communication channels for the same set of nodes. For example, these nodes might need to exchange data, important messages (control), ... In order to select different methods of communication, different profiles can be used.

The main profiles in Padico are:

- control: is useful in order to build Padico control links.
- default: is the default choice when we talk to a standard Padico node.
- inet: use a common socket connection. This is in order to respect interoperability with others applications.

`padico-nsconfig`

The NetSelector XML configuration files can easily be created and modified using the Padico application `padico-nsconfig`.

A snapshot is shown on Figure 3.3. The first bottom panel shows the groups. Here a group "daltons" is created. To create a new group, click on the "New" button at the left bottom of the panel. A new panel as shown on Figure 3.4 will appear. You need to provide

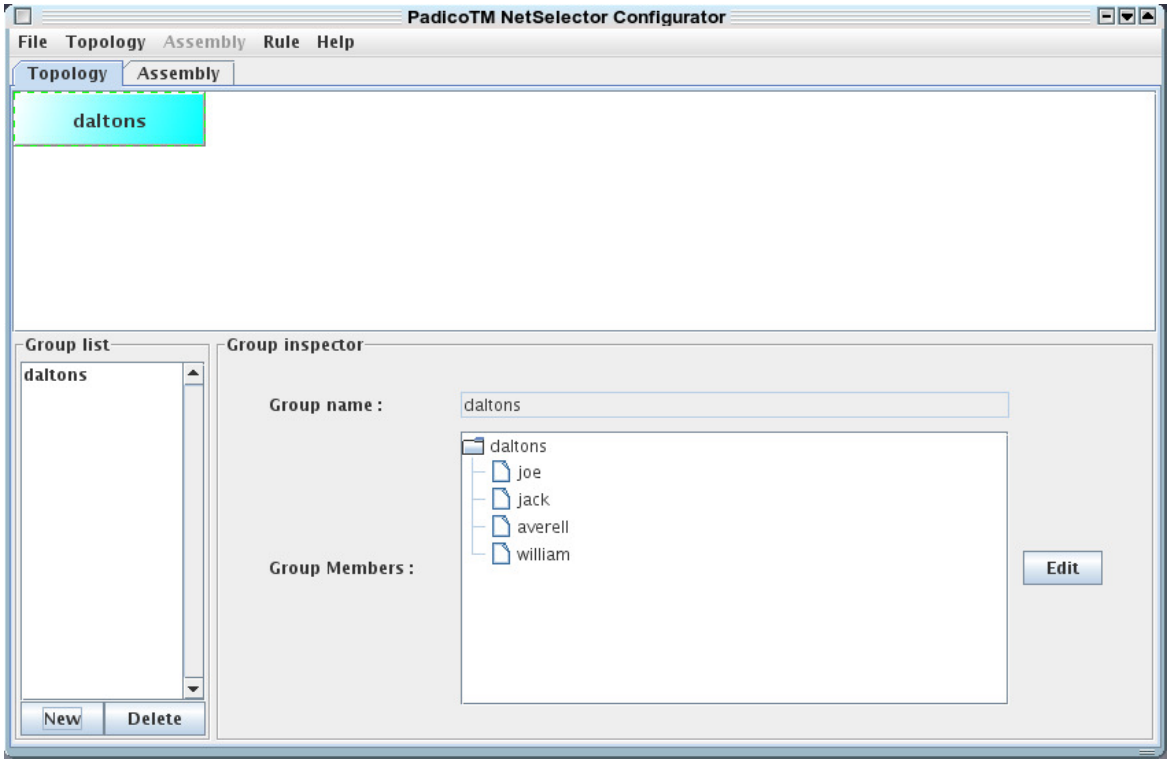

Figure 3.3: "Topology" panel of `padico-nsconfig`

the name of the new group, the other groups and others hosts that it contains, and click "OK" to confirm the creation of the group.

When all your groups are defined, switch to the "Assembly" panel, shown on Figure 3.5. To create a new assembly, click on the "New" button. You need to provide in a new panel the list of interfaces provided by this new assembly, and click on "OK" to confirm. The chosed interfaces will appear in the graph in the middle of the "Assembly" panel. To complete the creation of the assembly, you can add adapters by first selecting them in the Assembly Builder box and then clicking on the "Add" button. Choosing and finding adapters can be done by applying filters on their interfaces. The chosen adapter is shown at the right bottom of the panel. After adding all the needed adapters, you need to connect their interfaces using the appropriate "Current Tool" in the list at the right top corner of the panel.

After defining the assemblies and the groups, you need to bind them by defining some rules. To do so, choose an assembly, click on "New" in the "Targets" panel, add some groups, choose a profile and a type of link.

Once you are done with all these steps, save your new NetSelector configuration into a file. The created XML file can then be given to the NetSelector configuration module.

3.2.3 NetSelector and `padico-control`

The choice of the NetSelector can also be done using the `padico-control` interface. You can choose between different combinations of the allowed NetSelectors. The choices are the same as presented just above, plus a None (in-core) choice, that supposes that the NetSelector

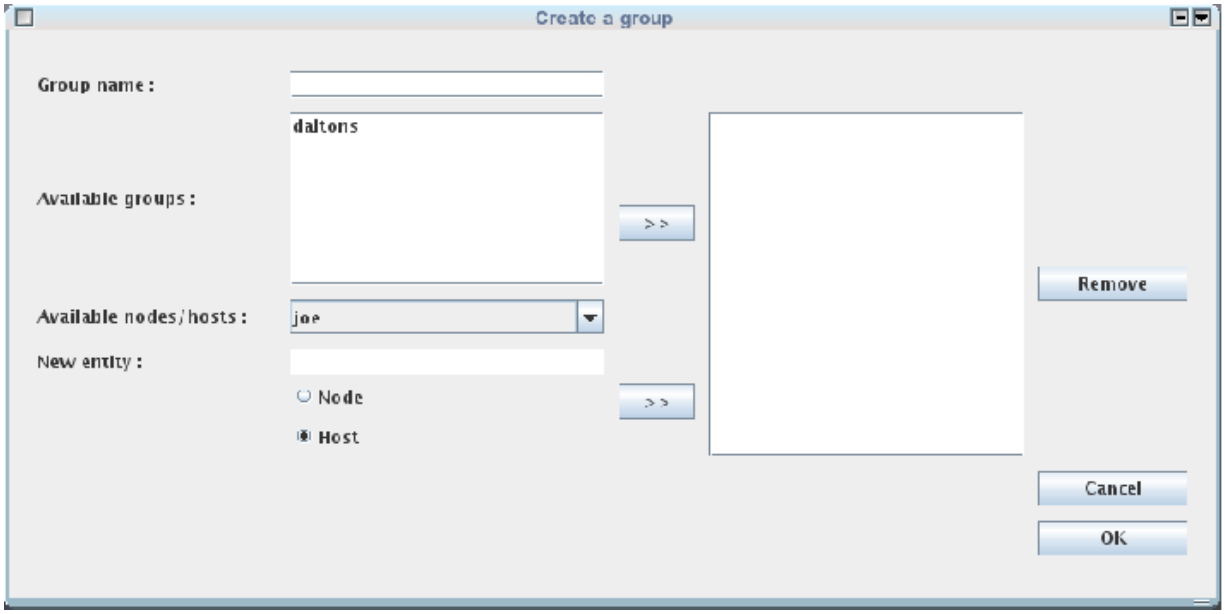

Figure 3.4: Group creation in `padico-nsconfig`

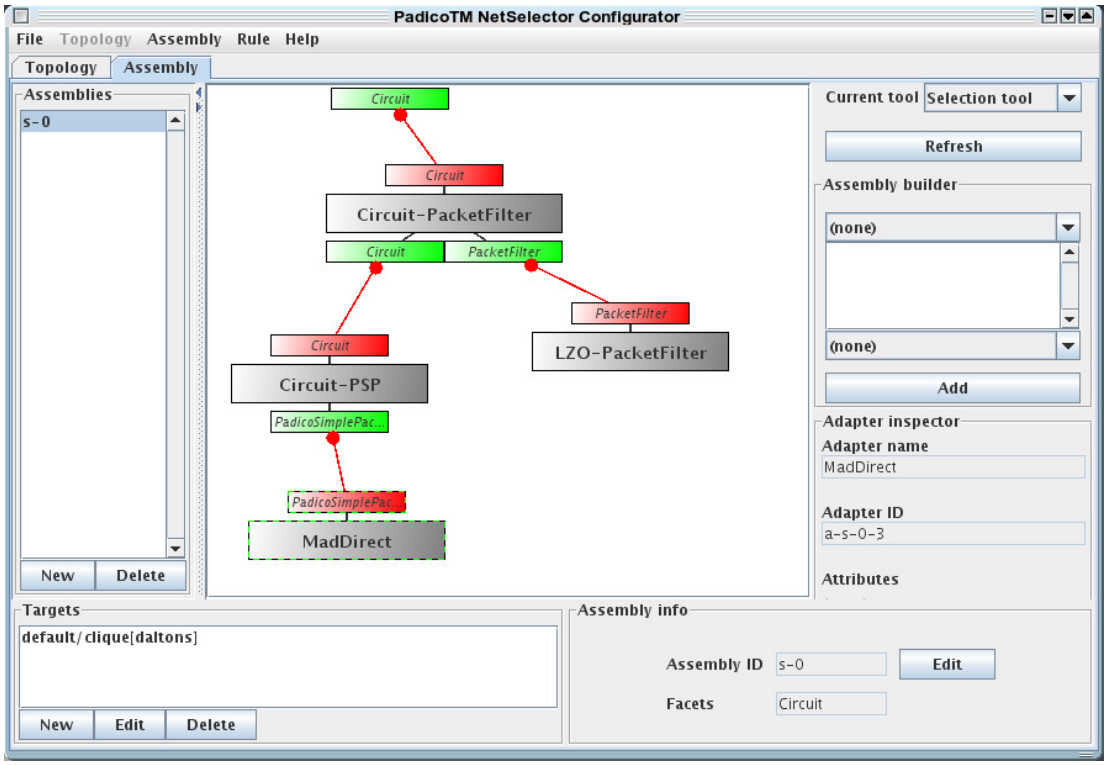

Figure 3.5: "Assembly" panel of `padico-nsconfig`

was given at core-making time. This functionnality will be deleted in the near future. A text box allows to give the name of the XML file in the case where the NetSelector-basic is used."Find" and "Edit" buttons allow to find an XML NetSelector file and to edit the selected file by hand (using emacs).

3.3 MPI with PadicoTM

3.3.1 Example

This section is not a MPI programming guide. It only gives the differences between MPI programming in Padico and "regular" MPI programming.

HelloMPI. Let us consider a "Hello world" written with MPI. The source code for an MPI code with PadicoTM is exactly the same than with any other MPI implementation. The `C` include file is `<mpi.h>`. The example for this tutorial is in `PadicoTM-0.5.0/PadicoTM/Examples/Tutorial/02-HelloMPI/`.

Compiling You should have compiled the MPI middleware service (as mpich 1.2.X). See Section 2.5 on how to build external middlewares services. For our HelloMPI example, we first declare what core to use (`mono-mx` in our example):

```
setenv PADICO_CORE mono-mx
```

Then we compile `HelloMPI.c`:

```
padico-mpicc -c HelloMPI.c
```

Then, we link:

```
padico-mpicc HelloMPI.o -o HelloMPI
```

Compiling and linking in one pass is not supported. The link operation does not build a `HelloMPI` binary file. Instead, it creates a Padico module called HelloMPI. If you do not want to use an environment variable, you can specify the core for each operation:

```
padico-mpicc --padico-core mono-mx -c HelloMPI.c
padico-mpicc --padico-core mono-mx HelloMPI.o -o HelloMPI
```

Running HelloMPI. As with any Padico application, we first boot Padico with a `padico-boot` command (see the example in Section 3.1.2) or with `padico-control`. MPI uses the CORBA gatekeeper, so it is required to use a startup module which includes `CORBA-Gatekeeper`. Our example `mono-mx` startup module fullfills this condition.

Instead of `mpirun`, you will run your application with `padico-mpirun`. It accepts the usual "`-np`" and `-machinefile` flags. We added the non-standard `-nodelist` flag to explicitly give a list of nodes on the command line. Once Padico has booted, we run HelloMPI on all available nodes with the following command line:

```
padico-mpirun HelloMPI
```

You can also use GUI for `padico-mpirun`. Open the "MPI run..." panel in the "Tools" menu, enter the module that you want to launch, the MPI module you want to use and press "MPIRun".

```
interface Hello_server
{
  void Hello();
};
```

Figure 3.6: The `Hello.idl` interface declaration

Summary for MPI in PadicoTM. MPI in PadicoTM works like any other MPI implementation except that it uses `padico-mpicc` instead of `mpicc`, and `padico-mpirun` instead of `mpirun`. It is possible to make links so that the command names are `mpicc` and `mpirun`, but it may conflict with an already installed MPI implementation on your system. The only peculiarity is that you must not forget to boot PadicoTM before running `padico-mpirun`, or use the `padico-control` MPI panel.

3.4 CORBA with PadicoTM

3.4.1 Example

We will write a "Hello world!" client-server application. In this section, we suppose you are familiar with the CORBA architecture and the "Hello world" PadicoTM example described in Section 3.1.1. Let us consider the IDL declaration shown on Figure 3.6. The source code for this example is in `PadicoTM-0.5.0/PadicoTM/Examples/Tutorial/03-HelloCORBA/`.

First, we choose the CORBA implementation that *my*CORBA will use:

```
setenv MYCORBA omniORB-3.0.2
```

Then, we invoke the IDL compiler through *my*CORBA with the following command line:

```
mycorba-idl Hello.idl
```

It generates two files: `Hello.cc` and `Hello.h`. Compile `Hello.c` with the following command line:

```
mycorba-c++ Hello.cc
```

The source code for a Padico CORBA server is very similar to a regular CORBA code. It is given on Appendix C.2.

Notice the differences:

- the header file for CORBA is `Padico/CORBA.h`, the CORBA naming service is `Padico/COSNaming.h`;
- the type of the ORB is `Padico::ORB_ptr` instead of `CORBA::ORB_ptr`;
- ORB initialization is done through the function `Padico::ORB_init()` instead of `CORBA::ORB_init(argc, argv, orbname)`.
- `orb->run()` and `orb->shutdown()` are not needed, though they are allowed.

We build the server with the following command lines:

```
mycorba-c++ Hello-server.cc
padico-mkmod Hello.o Hello-server.o -o Hello-server \
  --padico-core <your core name>
```

The client-side code source follows exactly the same rules concerning headers files, ORB variable type and ORB initialization. It is given on Appendix C.1.

Build the client with the following command lines:

```
mycorba-c++ Hello-client.cc
padico-mkmod Hello.o Hello-client.o -o Hello-client \
  --padico-core <your core name>
```

3.4.2 More about CORBA in PadicoTM

Files and services

Padico permits the use of any ORB implementation. The ORB are available through the use of the appropriate middleware service. Look at Section 2.5 to see how to compile it. A user code that uses CORBA must include `Padico/CORBA.h`. No other `CORBA.h` may be included within a Padico module.

The CORBA naming service can be accessed with the include file `Padico/COSNaming.h`. Padico CORBA does not provide other CORBA services. However, services from other CORBA implementations (MICO for example) should work with Padico CORBA.

PadicoTM CORBA in the source code

The ORB should be stored into a variable with type `Padico::ORB_ptr`, initialized with `Padico::ORB_init()`. **Do not use `CORBA::ORB_ptr`**. `Padico::ORB_ptr` provides a CORBA-like interface. It defines almost all the standard ORB methods. Keep in mind that, though it looks like a regular CORBA ORB, it is not a real ORB. It is only an interface that provides access to a *shared* ORB. The real ORB is owned, initialized, and run by Padico. It runs in a background thread managed by Padico. It is able to serve several modules at the same time; each module has its own `Padico::ORB_ptr` variable.

For compatibility with regular CORBA programs, methods such as `run()`, `perform_work()` and `shutdown()` are defined and may be called but do nothing in `Padico::ORB_ptr`. Advanced features such as typecode-related functions are not defined in the Padico ORB. If you really need them, you can get the actual ORB in `Padico::the_orb`. Do not touch `Padico::the_orb` unless you really know what you are doing. In particular, it is strictly forbidden to call `run()` or `shutdown()` on this object.

This limitation should be removed in a future version of PadicoTM.

Build

Compile the IDL files with `mycorba-idl`. There is an optional flag `--any` for generating code to handle the CORBA type `Any`. Compile the `.cc` files with `mycorba-c++` which behaves like any C++ compiler. See the section 4.3 for more details on *my*CORBA. `mycorba-c++` is only a compiler; it is *not a linker*. Then link as usually with `padico-mkmod`.

A module that uses CORBA must be compiled with the same C++ compiler than the PadicoTM service "ORB" itself. Using another compiler would most probably prevent the file from being linked properly since C++ mangled names vary from one compiler to another.

Execute

In order to load all we need to execute CORBA applications, we know have to use the -iloadCORBA option for `padico-boot`.

3.5 Padico-control and Firewalls

In some specific cases, you might want to run the `padico-control` panel on a computer which is away from the "compute" nodes. For example, you might want to use nodes from a cluster, and to have your `padico-control` panel on your local machine. It is then very likely to have a firewall between the cluster and your local machine. We will see how now to deal with these cases.

3.5.1 A first example

Let's take the most common example: you want to run the `padico-control` panel on your local machine. You are behind a firewall. You also want to run the nodes on the machines of a cluster, which is also protected by a firewall.

If you choose the option "all" in the "Contact padico control" box of the `padico-control` panel, then all the compute nodes will try to connect to the control node running on your local machine. That will probably fail because of the firewall. You could allow these connections to go through by changing the rules of your firewall, but this is not a good (and feasible) solution, you should instead use the routing capabilities of Padico.

As shown on Figure 3.7, launch the `padico-control` on your local machine. Then you need to start the routing node `node0`. This is done by choosing the option "one" or "all" in the "Contact padico control" box, precise that the node will be a "Rdv Server" and give the entry routing port in the appropriate field. This node will then contact the machine that runs `padico-control` through a SSH tunnel (you need to use a NetSelector XML file as below). This is the only connection which needs to go through the firewall in order to connect all the nodes. In the NetSelector XML file given just below; the s-1 assembly forces the SSH-tunneling.

```
<?xml version="1.0"?>
<!DOCTYPE NS:rule-set SYSTEM ".../<arch>/etc/NetSelector.dtd">
<NS:rule-set xmlns:NS="http://padico.org/PadicoTM/NetSelector">

    <NS:group id="paravent-IB">
        <NS:host name="paravent[0-6]*.rennes.grid5000.fr"/>
    </NS:group>
    <NS:group id="paravent">
        <NS:group ref="paravent-IB"/>
        <NS:host name="paravent*.irisa.fr"/>
    </NS:group>
```

```
    <NS:group id="portable">
      <NS:host addr="0.0.0.0"/>
    </NS:group>

    <NS:assembly id="s-0">
        <NS:adapter id="0" name="InfinibandVerbs"/>
        <NS:assembly-provides adapter-id="0" iface="PadicoSimplePackets"/>
    </NS:assembly>
    <NS:rule>
        <NS:assembly ref="s-0"/>
        <NS:target kind="clique" profile="default">
            <NS:group ref="paravent-IB"/>
        </NS:target>
    </NS:rule>

    <NS:assembly id="s-1">
        <NS:adapter id="0" name="SocketFactory-SSH-Tunnel">
          <NS:attr label="ssh_host">cocalight.labri.fr</NS:attr>
          <NS:attr label="ssh_port">21436</NS:attr>
        </NS:adapter>
        <NS:assembly-provides adapter-id="0" iface="SocketFactory"/>
    </NS:assembly>
    <NS:rule>
        <NS:assembly ref="s-1"/>
        <NS:target kind="arrow" profile="inet">
            <NS:group ref="paravent"/>
            <NS:group ref="portable"/>
        </NS:target>
    </NS:rule>

</NS:rule-set>
```

You can launch the other nodes by choosing the option “none” in the “Contact padico-control” box, and entering the routing host and port after precising that the nodes will be clients. All these nodes will contact the `padico-control` through the routing node.

Note: You can also start all the nodes together without precising the server, by choosing the “one” mode for all the nodes. The 0-node will be the routing node.

3.5.2 Multi-site example

The example on Figure 3.8 shows how we can connect several sites with only one connection between the machine running `padico-control` and the “outside world” defined by the machines from the other side of the “local” firewall.

The principle is to start a node that will be the routing node like in the first example. The nodes on the same cluster are “Rdv Clients” of this node. But when we want to start some nodes on another cluster, and do not want to allow connections between this cluster and the

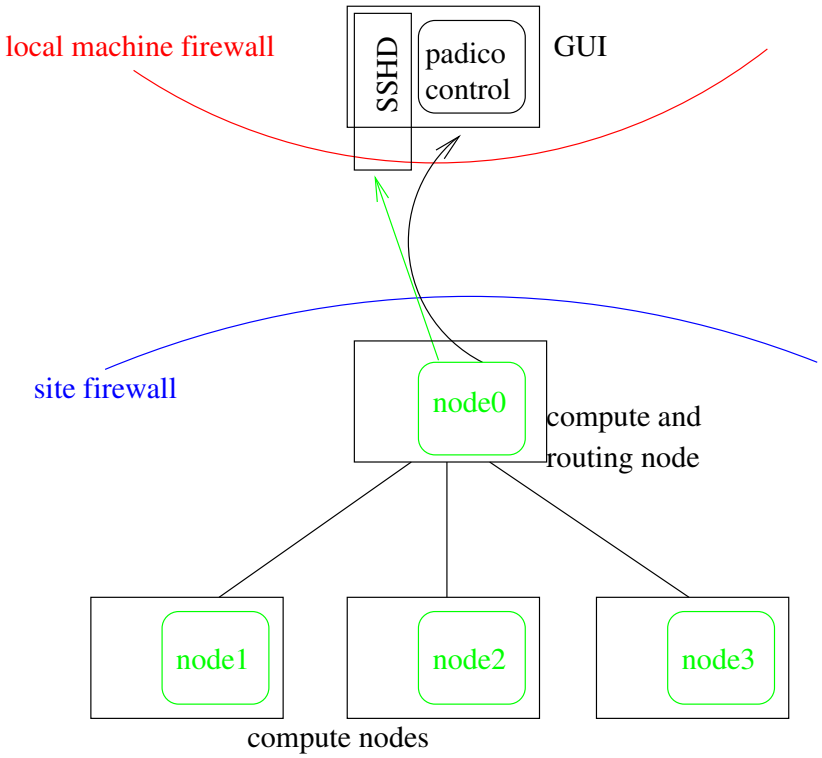

Figure 3.7: Simple connection through a firewall

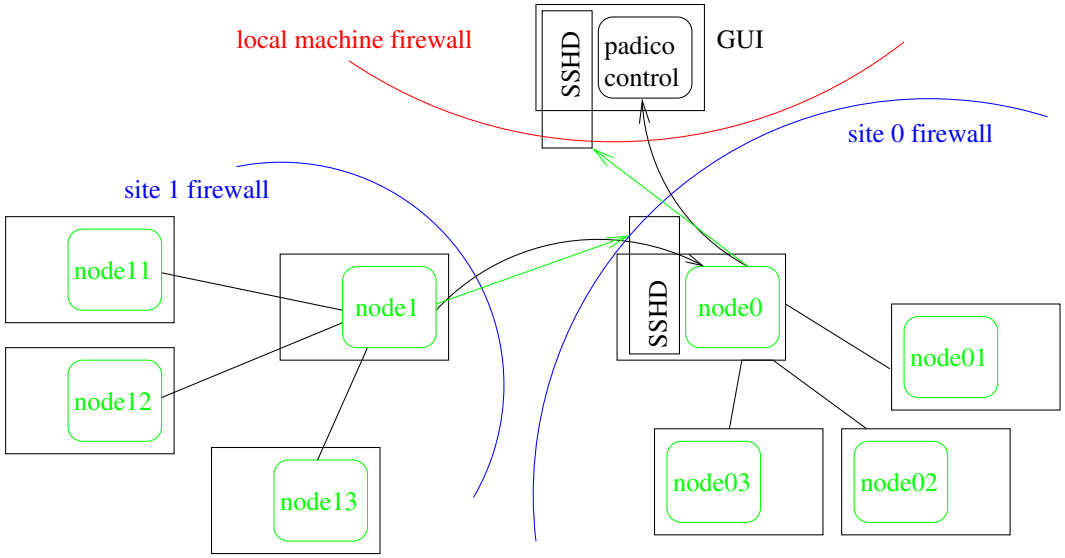

Figure 3.8: Multi-site example

local machine, the routing node (here "node0") can also route the connections. But in this case, we also want to limit the number of connections between "site1" and "site0". Then there will be another routing node (here "node1"). This node must be started as a client (it will connect to the "node0" for routing its messages) but also as a server, in order to route the others nodes from the "site2" machines. The "Contact padico-control" choice is here "none" as no node will connect directly to `padico-control`. The other nodes of the cluster must be started as "Rdv Client" of the "node1" routing node.

3.6 Topology manipulation

To choose best assembly the NetSelector-besteffort must know the topology between two nodes. A node can automatically discover the neighborhood topology, but it can't guess network topology beyond a firewall for example. In this case we must export and import topology between nodes. In PadicoTM we have a padico service called "Topo-print" which can print and export topology in PadicoTM native format.

3.6.1 Export and import topology example

In this section we explain how to transfert topology between 2 nodes. We want export topology from node A to the node B. On the node A we use the `Topo-Print` module to export the topology in PadicoTM format in a file called mytopo.xml:

```
padico-launch Topo-Print padico mytopo.xml
```

Then on the other side, on the node B we use the "-i" `padico-launch option` to load the file mytopo.xml which contains the topology previously saved:

```
padico-launch -i<path_to_file_mytopo.xml> ...
```

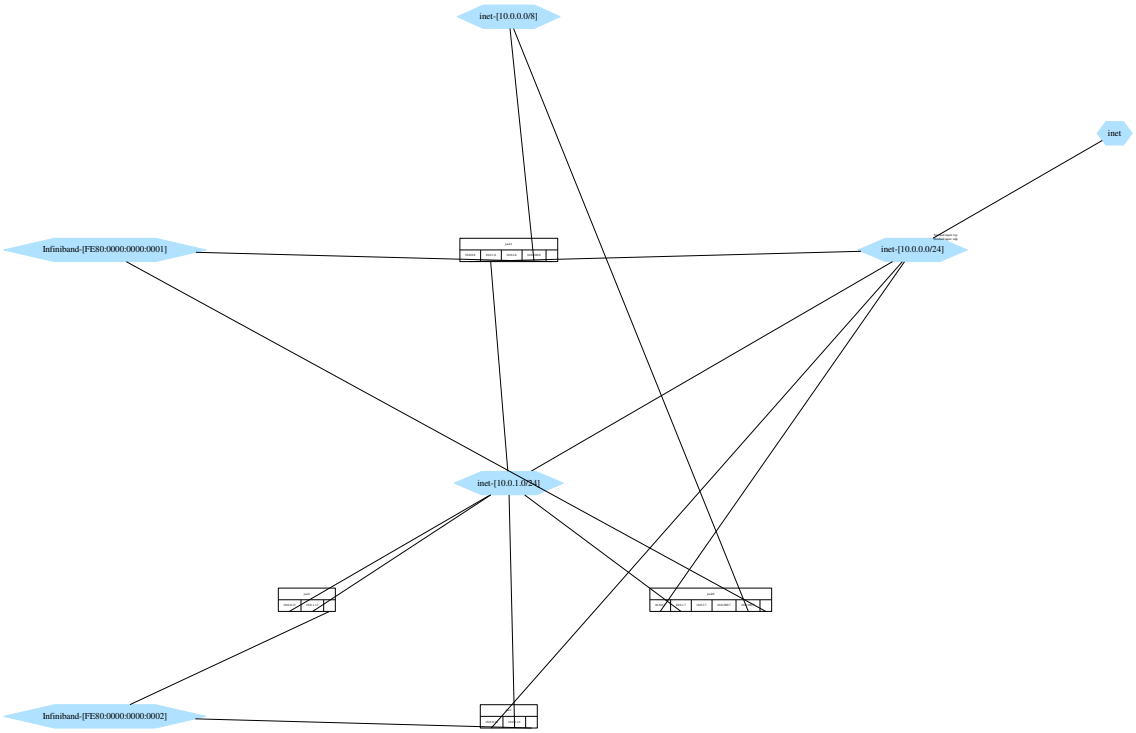

Figure 3.9: example of graphic form topology

3.6.2 Draw topology

Sometimes NetSelector-besteffort do not find a path between two nodes, and the result is it can't compute a correct assembly. In this case, it is pleasant to show the topology in a graphic form for debbuging and watch if a path exist between two nodes. If Topo-Print was compiled with graphviz libraries, it is possible to export topology in a dot format.

```
padico-launch -i<path_to_file_mytopo.xml> Topo-Print dot mytopo.dot
```

This command transform the topology of the previous example in a graphviz dot language. Then this file can be use by graphviz's tools to generate the graph in a graphic form. The example on Figure 3.9 show topology graph generated by graphiz's tool.

Chapter 4

*my*CORBA reference manual

*my*CORBA is a set of wrappers designed to hide the differences between ORB-vendors. Thanks to *my*CORBA, the source code and the makefiles do not see different ORB vendors, they see *my*CORBA instead. For now, *my*CORBA is C++ only.

4.1 Setting up *my*CORBA: `mycorba-admin`

When you install Padico, *my*CORBA is automatically installed. You must then *declare* to *my*CORBA what ORB are installed on your system. The setup command for *my*CORBA is `mycorba-admin`.

Supported ORB Currently, supported CORBA implementations are: MICO, OmniORB 3, OmniORB 4, ORBacus 4 and PadicoTM-enabled OmniORB.

ORB name	ORB identifier string	Name Service	auto-detection
MICO 2.3.x (tested with 2.3.7)	`MICO`	yes	`MICODIR`
Iona ORBacus 4.0.x (tested with 4.0.5)	`ORBACUS4`	yes	`OB_ROOT`
OmniORB 3.0.x (tested with 3.0.2)	`OMNIORB3`	yes	`OMNIORB3_DIR` and `OMNIORB3_PLATFORM`
OmniORB 4.0.x (tested with 4.0.0)	`OMNIORB4`	yes	`OMNIORB4_TOP`
TAO— *not supported yet*	`TAO`	–	–
PadicoTM-enabled OmniORB 3 (3.0.2 provided in the PadicoTM distribution)	`PADICO_OMNIORB3`	no	*automatically declared by PadicoTM*
PadicoTM-enabled OmniORB 4	`PADICO_OMNIORB4`	no	*automatically declared by PadicoTM*
PadicoTM-enabled MICO, MICOMT, MICOCCM or MICO-GUP — *experimental — not supported yet*	`PADICO_MICO`	no	–

***my*CORBA ORB identifier string** In order to use an ORB with *my*CORBA, you should set an environment variable with the appropriate ORB identifier. For example to set OmniORB 3 as your default ORB in *my*CORBA :

```
setenv MYCORBA omniORB-3.0.2
```

Additionally, every *my*CORBA command accept the `--mycorba=` option to override the default choice.

Scanning ORB. DEPRECATED — *my*CORBA cannot scan for ORBs!
*my*CORBA can automatically detect what ORB are installed. Type the following command:

```
ugo-orb --scan
```

Auto-detection relies on environment variables to detect what ORB are installed. For MICO, the variable `MICODIR` must be set; it is done if MICO is correctly installed and `mico-setup.csh` is sourced. For OmniORB 3, the variable `OMNIORB3_DIR` should point to the root of the OmniORB installation. For ORBacus 4, the variable `OB_ROOT` should point to the root of the ORBacus installation. The PadicoTM-enabled OmniORB is automatically installed by PadicoTM.

List known ORB. To display what ORB are recognized by *my*CORBA on your system, type:

```
mycorba-admin --list
```

Adding an ORB. To add a specific ORB, you can use the `--add` flag. The general syntax is:

```
mycorba-admin --add <orb_file> <orb_installation_path>
```

For example:

```
mycorba-admin --add omniORB-4.0.x $HOME/omniORB-4.0.3
```

This example declares *omniORB 4.0.3* to *my*CORBA, based on the `omniORB-4.0.x` template file and *omniORB* being installed in `$HOME/omniORB-4.0.3`.

4.2 Managing a CORBA name server with *my*CORBA: `mycorba-ns`

If an ORB implements a CORBA NameService, then *my*CORBA can start and stop the server, and get the reference of the running server, in a portable fashion. The PadicoTM-enabled CORBA implementations do not provide a name server implementation.

Start a name server.

```
mycorba-ns --start
```

Stop the name server.

```
mycorba-ns --stop
```

Declare a name server. You can declare to *my*CORBA an already running name server in order to use your own name server. Example:

```
mycorba-ns --register corbaname::paraski.irisa.fr:10000
```

Get the name server status.

```
mycorba-ns --status
```

Get the name server reference.

```
mycorba-ns --getref
```

This feature is very useful in wrapper scripts for CORBA clients. You will usually add the following flag on the command line:

```
 -ORBInitRef NameService=`mycorba-ns --getref`
```

Additionally, *my*CORBA provides for a generic wrapper which automatically adds an `-ORBInitRef` argument pointing to the currently registered name server. For example, to run `nsadmin` with the currently registered name server, type:

```
mycorba-ns --wrap nsadmin
```

4.3 *my*CORBA in your makefiles

The usual way is to use `mycorba-idl` as an IDL compiler, and `mycorba-c++` as a C++ compiler. `mycorba-idl` and `mycorba-c++` should be enough for average users.

Build for advanced users. Advanced users may not want to use `mycorba-idl` or `mycorba-c++`. These users may be interested in `$PADICO_ROOT/include/myCORBA.sh` and `$PADICO_ROOT/include/myCORBA.mk` which can be sourced respectively in `sh` shell and GNU Makefiles. The names are (hopefully!) self-explanatory. `myCORBA.sh` and `myCORBA.mk` are installation dependent. It is unsafe to copy/paste their content. You should rather source them, then use the variables defined. For the advanced way of using *my*CORBA in the makefiles, see the makefile of the CORBA example in PadicoTM.

Chapter 5

PadicoTM reference

5.1 PadicoTM Module Specification

5.1.1 Module description

In PadicoTM, all modules are described in a XML dialect, called *XML Padico Module Description* or `XPMD`. Its specification is presented in Figure 5.2 and a simple example is listed in Figure 5.1. A module is a static object. A node can at most load one instance of a module. A module definition contains a name, a driver name and a list of units. It may also contain some attributes. Here is a description of theses objects:

Module name The name of the module is a string which identifies the module on disk and in memory. It must be unique.

Driver Each module is associated to a driver. The driver is responsible for loading, starting and unloading the units of the module.

Dependency A module may require some others modules. The required modules will be automatically loaded if necessary.

Unit Components of modules are called "units". The type of the units is determined by the driver, so all units of a module are of the same type. For example, units may be binary libraries, Java classes, other modules. There must be at least one unit in a module.

Attribute Attributes are static values used for configuring modules. Attributes are composed of a *label* and a *value*. Labels and values are strings.

File Additional files that compose the module but are not explicitly loaded are referenced.

All the units of a module must have the same type: they are all managed by the same module driver. The module driver specifies how the units of the module have to be loaded, unloaded and started. Section 5.1.2 explains the driver execution model in depth.

```
<defmod name="Madico" driver="binary">
<attr label="ENABLE_OPTIMIZE">no</attr>
<attr label="ENABLE_DEBUG"></attr>
<requires>Circuit</requires>
<unit>Madico.so</unit>
</defmod>
```

Figure 5.1: The XPMD of the Madico service. The *name* is "Madico". The *driver* is "binary", which means that units are shared objects. As the service is based on the circuit module API, it *requires* the `Circuit` module. The Madico service needs to know if the core is in debug or optimized mode; this is done through an *attribute*. The module contains one *unit*, namely the binary library for the module. A module can also include some other *files* which are not explicitely loaded, but are considered as a part of the module; it is useful to store these files if we want to move the module accross systems.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:annotation>
  <xsd:documentation xml:lang="en">
  Padico Module Description V1.0
  Based on a maybe too recent version of XMLSchema.
  </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="mod" type="modType"/>

 <xsd:complexType name="modType">
  <xsd:attribute name="name"   type="xsd:string" use="required"/>
  <xsd:attribute name="driver" type="xsd:string" use="required"/>
  <xsd:sequence>
   <xsd:element name="requires" type="xsd:string" minOccurs="0"
                                                  maxOccurs="unbounded" />
   <xsd:element name="attr" type="attrType"      minOccurs="0"
                                                  maxOccurs="unbounded" />
   <xsd:element name="unit" type="xsd:string" minOccurs="1"
                                                  maxOccurs="unbounded" />
   <xsd:element name="file" type="xsd:string" minOccurs="0"
                                                  maxOccurs="unbounded" />
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="attrType">
  <xsd:simpleContent>
   <xsd:extension base="xsd:string">
    <xsd:attribute name="label"  type="xsd:string" use="required"/>
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>

</schema>
```

Figure 5.2: XML Padico Module Description specifications in XML Schema.

5.1.2 PadicoTM Execution Model

Module. The basic operations allowed on modules are: open, load, start, unload. These operations are provided by Puk, which parses the XPMD files, resolves any dependencies, and delegates units operations to drivers. *Open* a module is the action of opening the XML file, parsing it, and storing its description in memory; nothing is read from disk except the XML file; units are not loaded. *Load* a module is the action of resolving module dependencies, loading and initializing the module units. This operation is performed on each embedded unit. *Start* a module is the action of starting a new execution thread. The *start* operation is not supposed to block. A module may be started several times, in sequence or even in parallel. *Unload* is the operation of finalizing the module execution, i.e. stop the threads, free memory, close files, and then unloading the module from memory.

Units. To perform operation on modules, Puk delegates operation on the units to drivers. Every driver has to support three basic operations on units: **load**, **unload** and **start**. When a module contains several units, the operations are always performed one unit after the other in the order of appearance in the module definition.

5.1.3 Available drivers

This section presents the drivers currently available in Padico. Figure 5.3 sums up available standard drivers.

Driver name	Implemented by module...	Units type
`binary`	`PadicoTM`	Shared object libraries (binary code)
`pkg`	`Puk`	Modules
`multi`	`NetAccess`	Modules to deploy on a group of nodes in an SPMD way
`java/class`	`Kaffe`	Java classes
`bin/sh`	`Sh-driver`	`/bin/sh` scripts
`boot`	`Puk`	An expanded "binary" module used to manage a node. Users are not supposed to create such modules

Figure 5.3: List of standard drivers.

Driver	Description

`binary` Implemented by: **`PadicoTM`**

The units are shared object file. All three operations are supported, all of them are optional.

Load When the module is loaded, the function `int padico_module_init(void)` is called if it exists for each unit of the module. This function must not block. It must return 0 upon initialization success, any non-zero value in the case of failure.

Unload When the module is being unloaded, the function `void padico_module_finalize(void)` is called if it exists for each unit of the module. This function must not block. It must stop any activity started by calls to `start`.

Start When a module is started the function `int padico_module_run(int argc, char**argv);` is called if it exists for each unit of the module. If it does not exist, the function `void main(int, char**);` is looked up instead. The `argc` and `argv` parameters are supplied by the user at run-time and given directly to the user code. `argc` is the size of `argv`; `argv[0]` is the module name, and `argv[argc]==NULL`. The driver creates a new thread for the execution, thus the function may block.

Note: all the prototypes are in C. In C++, do not forget the `extern "C"`. Usually, the user does not declare explicitly these functions, but uses `PADICO_MODULE_INIT`, `PADICO_MODULE_RUN` and `PADICO_MODULE_FINALIZE` instead.

`pkg` Implemented by: **`Puk`**

The units are modules. The three operations are supported, all of them are optional. Each operation is applied sequentially to all the units of the module. These modules may be used for grouping several modules in one, or to add an attribute to another module. At most one included module should implement the "Start" operation; if more than one module implements a "Start", the behavior is not specified.

`multi` Implemented by: **`NetAccess`**

Attribute: `GroupID` (mandatory).

The units are modules. This driver is very similar to the `pkg` driver except that the units are managed at a *group of nodes* level. The group is designed by the **GroupID** attribute (**required**). The attribute is searched in the environment of the module. If the search fails in this environment, the search goes on in the father module and so on until the root is reached. It is an error not to provide such an attribute. There is SPMD synchronization between the execution of the **load**, **unload** and **start** operations on the units.

continued on next page

Driver	Description
`bin/sh`	Implemented by: **`Sh-driver`** Attribute: `Dir` (mandatory), `Sync` (optional, default="Yes"). The units are `/bin/sh` scripts. Supported operation: **Start**. The `Dir` attribute specify the current directory when the script will be started. The `Sync` attribute can be "Yes" or "No"; in the case of "Yes", the parent process is notified upon script completion; in the case of "No", the script runs asynchronously without completion notification. Note: unlike other module types, scripts are started in their own process.
`java/class`	Implemented by: **`Kaffe`** Attribute: `Code` (optional), `Stdmain` (optional, depends on Code). The units are Java classes. The three operations (**load**, **unload** and **start**) are recognized. All of them are optional and only apply to the Java class defined by the `Code` attribute. If this attribute is not set, none of this operation can occur. **Load** When the module is loaded, the method of prototype `public static void padico_module_init(JModule mod)` of the class specified by the `Code` attribute is called if it exists. This method is called after static initializers. It must not block. **Unload** When the module is being unloaded, the method of prototype `public static void padico_module_finalize(JModule mod)` of the class specified by the `Code` attribute is called if it exists. This method must not block. **Start** When a module is started and the `StdMain` attribute is not set, the method of prototype `public static void padico_module_run(String[])` of the class specified by the `Code` attribute is called whether is exists. If the attribute `StdMain` is set, the function of prototype `public static void main(String[])` of the class specified by the `Code` attribute is called whether is exists. As a thread is created for this execution, the `padico_module_run` and the `main` method are allowed to block. Note: Java support is not included in the standard PadicoTM-0.5.0release. It is supplied as an experimental add-on.

5.1.4 Execution environment

Each PadicoTM module is associated with a run-time environment. This environment is a tuple-space composed of labels associated to values. Labels and values are case-sensitive strings.

The C mapping of the basic operations are:

Function name	Description
`char* padico_getattr(char* label);`	Return the value associated with the variable of name described by `label`. The variable is first searched for in the module environment, and, if the search fails, the variable is search recursively in the parent module environment. Return `NULL` if the variable does not exist or is the null string.
`char* padico_getlocalattr(char* label);`	Return the value associated with the variable of name described by `label`. The variable is only searched in the module environment. Return `NULL` if the variable does not exist or is the null string.
`void padico_setattr(char* label, char* value);`	Insert the variable designed by label and of content `value` in the environment of the module. Any previous value of the variable is overwritten.

There is a Java mapping available. These methods belongs to the `JModule` class. An instance of such a class is pass to `padico_module_init(JModule mod)` and to `padico_module_finalize(JModule mod)`.

Function name	Change from C mapping
`public String getAttr(String label);`	Return `null` instead of `NULL`.
`public String getLocalAttr(String label);`	Return `null` instead of `NULL`.
`public void setAttr(String label, String value);`	

To remove a variable from the environment, set it to `NULL`. It is impossible to distinguish the case of a non-existent variable from the case of a variable set to `NULL`.

5.1.5 How to

- Load the modules A and B on group `MyGroup` without synchronization between A and B:

  Let's create a module C that embeds modules A and B. The command line is:

  ```
  padico-mkmod --driver=multi -DGroupID=MyGroup -o C A B
  ```

The generated XPMD is:

```
<mod name="C" driver="multi">
  <attr label="GroupID">MyGroup</attr>
  <unit>A</unit>
  <unit>B</unit>
</mod>
```

- Load the modules A and B on a group with a synchronization between A and B.

  Let's create modules C, group-A and group-B. group-A embeds module A and group-B embeds module B. The command lines are:

```
padico-mkmod --driver=multi -o group-A A
padico-mkmod --driver=multi -o group-B B
padico-mkmod --driver=pkg -DGroupID=MyGroup -o C group-A group-B
```

  The generated XPMD are:

```
<mod name="C" driver="pkg">
  <unit>group-A</unit>
  <unit>group-B</unit>
</mod>
```

```
<mod name="group-A" driver="multi"><mod name="group-B" driver="multi">
  <unit>A</unit>                      <unit>B</unit>
</mod>                              </mod>
```

5.1.6 Traces usage

In order to debug some code in Padico, it contains an option that enables traces. We'll see here what we need to know in order to use them.

First your core needs to be build with the –padico-trace-on option, or, if you used to use the padico-core-assistant, in "debug" mode.

Then you can enable the traces on the GUI, by choosing the "Trace" box. If you only do that, traces of all modules will be enabled. But there are many traces in Padico code. That would be a little hard to find the interresting messages. You can give an XML file that describes a policy. It looks like that:

```
<!-- Your comments ! -->
<tracepolicy defaultlevel="5">
  <info mod="yes" level="yes" func="yes" />
  <show mod="Circuit-PacketFilter" />
  <show mod="Circuit" />
  <show mod="bench-Circuit" />
</tracepolicy>
```

In this example, we ask for Circuit, Circuit-PacketFilter and bench-Circuit traces. The default level of trace in the code is also set to 5. We wants that the traces show the module, the level of the trace and the name of the function that called the trace function. The trace level is given when you write the function: `padico_out(trace_level, printf_args)`. Then all messages with a higher trace level won't appear. Messages of lower trace level of all modules will appear.

5.2 PadicoTM Services Reference

5.2.1 VIO

VIO is a virtualization of an I/O system. VSock virtualizes the sockets and only provides a minimal subset of the standard BSD socket programming interface. VIO goes further by providing a full socket API support — packets and stream, blocking and non-blocking, `select()` primitives — and by handling primitives for other I/O operations such as files and stdio. The "socket" part of VIO may be seen as VSock 2.0.

Used by: RemoteControl, Sh-driver, VLink, SocketFactory

Requires: NetAccess

Status: stable (except non-blocking I/O: beta).

API: C/C++ header file is `<Padico/VIO.h>`.

Note: implicit use possible without touching the code through SysW (see below).

5.2.2 Circuit

Circuit provides an abstraction of the Madeleine channels. Basically, it handles sub-groups and multiplexing over Madeleine. Reception is handled through callbacks.

Used by: Madico, MPI, MPCircuit

Status: stable

API: C/C++ header file is `<Padico/Circuit.h>`.

5.2.3 VRP

VRP stands for Variable Reliability Protocol. It is a module which implements a protocol with a tunable loss tolerance for high performance over a WAN. VRP is not built by default.

Used by: (none)

Requires: NetAccess

Status: experimental

API: C/C++ header file is `<Padico/VRP.h>`.

5.2.4 Console

Console enables users to interactively steer each process on its own console for debugging purposes.

Used by: none (end-user interface)

Status: stable

API: interactive use, type `help` for help.

Note: the PadicoTM processes should be launched in *console* mode to be able to use the Console service.

5.2.5 Sh-driver

The Sh-driver modules provides the `bin/sh` driver (see drivers in Section 5.1.3).

Used by: none (end-user interface)

Status: stable

API: driver `bin/sh` for `padico-mkmod`

5.2.6 RemoteControl

RemoteControl is the server-side for remote steering through `padico-control` (see section 3.1.3).

Used by: `padico-control` (remotely)

Requires: VIO

Status: stable

API: XML schema for XML Padico Command Language (TODO)

Attributes: `ControlHost`, `ControlPort`, `ControlKey` (mandatory)

Three attributes (`ControlHost`, `ControlPort` and `ControlKey`) are set by `padico-control` for RemoteControl to be able to callback `padico-control`.

5.2.7 CORBA-Gatekeeper

CORBA-Gatekeeper is the server-side for remote steering through CORBA (e.g. by `padico-run`, see Section 3.1.2).

Used by: `padico-run` (remotely), `padico-mpirun` (remotely)

Requires: CORBA-orb (CORBA-omniORB-4.0.6 for ex.), shared-CORBA

Status: stable

API: IDL file is `$PADICO_ROOT/include/Padico/gatekeeper.idl`

5.2.8 Madico

Provides a Madeleine interface over Padico. The Madico service is usually loaded by `padico-mpirun`, not directly by the end-user (see the MPI Tutorial).

Used by: MPI, user code

Requires: Circuit

Status: stable

API: C/C++ header file is `<Padico/Madico.h>`

5.2.9 shared-CORBA

Provides a high performance, shareable CORBA implementation.

Used by: CORBA-Gatekeeper, user code

Requires: VIO.

Status: stable

API: C++ header file is `<Padico/CORBA.h>`; use `Padico::ORB` wherever you would have used `CORBA::ORB`. See CORBA tutorial in Section 3.4.

Attributes: `NameService` (mandatory), `ORBgiopMaxMsgSize` (optional)

External package: `omniORB-4.0.6` (included in the standard distribution). See Section **??** for help.

5.3 PadicoTM Core Interface

5.3.1 Puk

Puk stands for "Padico micro-kernel" (or Padico μ-kernel). It is not a regular module like the others. It is responsible for module management and provides basic operations. When a PadicoTM process is started, it automatically loads Puk (hard-coded). Then Puk loads the startup module it was given; for the moment, the startup module must contain a PadicoTM core.

Used by: PadicoTM, NetAccess

Status: stable

API: C header is `<Padico/Puk.h>`. Puk functions begin with `padico_*` or `puk_*`. Puk is not designed to be used directly by the end-user. It aims at being used by core modules. However, Puk `padico_*` function may be safely used by the user; on the other side, `puk_*` function should be considered as low level interface – neither multi-threaded nor thread-aware – and should be used with extreme care. It is better to use the PadicoTM core module (see below).

Attributes: configured through command-line flags at boot time; recognized flags: `--padico-core <core>`, `--padico-startup <module>`, `--padico-session <module>`, `--padico-trace-{on|off}`.

5.3.2 PadicoTM

Used by: NetAccess, all services modules (VIO, Circuit, etc.)

Status: stable

API: C/C++ header for PadicoTM is `<Padico/PadicoTM.h>` (public API), low level Marcel/Madeleine C header available as `<Padico/PM2.h>` (internal use).

Attributes: `PADICO_STARTUP`, `PADICO_SESSION` (automatically set by Puk in regard to the command-line flags).

5.3.3 NetAccess

NetAccess is the PadicoTM network access manager. It provides callback facilities for efficient polling strategies, topology management, collective operations/synchronization.

Used by: Circuit, VIO

Status: stable

API: C/C++ header is `<Padico/NetAccess.h>`.

Attributes: (none)

NetAccess provides access to the network topology. See `<Padico/na-Cluster.h>` for more information. NetAccess functions begin with `padico_na_*`. Specialized NetAccess functions for system I/O (system sockets, files, etc.) are `padico_io_*`. Functions for extended Madeleine channels (multiplexed, callback-driven) are `padico_xchn_*`.

5.4 Commands manual

Each command supports (or *should* support!) the `--help` flag.

`padico-mkmod`

```
Usage: padico-mkmod [options] [files] [driver specific flags]
Options:
  -o <module>           Place the output into module <module>
  -D<label>=<value>     Defines an attribute <label> with the value <value>
  -Xlinker              Ignored (for compatibility with other linkers)
  --padico-core <core>  Use the core flavor <core>
  --driver=<driver>     Use Puk driver <driver> for the module
  --option=<opt>        Driver-specific option <opt> given to driver
  --requires <module>   Set a dependency on module <module>
  --bootable            Makes the module bootable
  --quiet               Do not show progress information
```

```
  -h|--help             Display this information
Files:
  *.o *.a *.so          For binary driver
  *.class               For java/class driver
  *.sh                  For bin/sh driver
  <module>              For pkg and multi drivers
Driver specific flags:
  -L... -l...           For binary driver
  -cp                   For java/class driver

Note:
  -o is mandatory.
  Unless the $PADICO_CORE environment variable is set, --padico-core is mandatory
```

5.4.1 padico-launch

```
Usage: padico-lauch <binary> [<arg>...]
```

padico-mkcore

```
Usage: padico-mkcore <core> [options] [ -- <args> ]

  <core>                     Name of the output core flavor

Options:
  --mad-{tcp|bip|gm|mx|sisci|elan}  Chose Madeleine network drivers
  --madico                          Enable Madico driver in Madeleine
  --marcel-{mono|smp|numa}          Chose Marcel flavor
  --marcel-trace                    Enable Marcel traces
  --marcel-pthread                  (experimental)
  --marcel-profile                  Enable Marcel profiling (experimental)
  --pm2-debug                       Switch PM2 in debug mode
  --pm2-verbose                     Verbose mode for compiling PM2
  --enable-debug                    Enable debugging symbols flag
  --enable-optimize                 Enable optimize flag
  --modules=<startup_modules>
Remaining args are passed directly to pm2-flavor.
```

padico-boot

```
Usage: padico-boot [options] <core> <host>... [ -- <arg>... ]
Options:
  -D<label>=<value> Define attribute <label> with value <value> in
                    the base environnment.
  --init[:<host>[,<host>...]]=<init>
  -i<init>          Add file <init> to startup sequence. <init> must
                    be an absolute file name or a name without path
                    (will be searched for in $PADICO_ROOT/etc/init.d/)
                    Argument -i may be given multiple times.
                    (only on host <host>)
  -c[:<host>[,<host>...]][=<xterm>]
  --console[:<host>[,<host>...]][=<xterm>]
```

```
                    Start each process in its own console
                    (only on host <host>, using <xterm> as console)
  -d[:<host>[,<host>...]]
  --debug[:<host>[,<host>...]]
                    Start processes in gdb (only on host <host>)
  -l[:<host>[,<host>...]][=<logdir>]
  --log[:<host>[,<host>...]][=<logdir>]
                    Log stdout and stderr (into directory <logdir>)
  -t[:<host>[,<host>...]][=<trace>]
  --trace[:<host>[,<host>...]][=<trace>]
                    Activate traces, using <trace> file as trace policy
  --rsh=<rsh>       Use <rsh> to start remote jobs
  -s, --sync-start  Starts processes sequentially. $LEO_RSH must be "ssh -f"
  -x, --xauth       Forward X authority (experimental)
  -f <file>         Add hotnames given in <file>
  -m <mod>          Defines a session module "inline"
  --ld[=<loader>]   Use an alternate dynamic linker/loader (expert only)
  --verbose         Show environment content while booting
  -h, --help        This help

 <core>             Name of a PadicoTM core
 <host>             Host list to start Padico on (at least one host)
 <arg>              Arguments given to Puk on each process

 -c, -d and -l are mutually exclusive.
 Several -d:<host> and -c:<host> may be given.
```

padico-run

```
usage:
/home/cfrezier/padico/i486/bin/padico-run <method> <args>...
With <method>:
  --soap
  --corba
  --get-default
  --set-default <method>

padico-run -corba:

Options:
  -c|--cleanup                   Deletes the node fromn the database if it does not a
  -a|--async                     Does not wait the command completion
  -q|--quiet                     Does not output any information
Commands:
  -k|--kill                      Kills PadicoTM
  -l|--loadmod <mod>             Loads module <mod>
  -r|--runmod  <mod>             Runs module <mod>
  -lr|--loadrunmod  <mod>        Loads and runs module <mod>
  -u|--unload  <mod>             Unload module <mod>
  -m|--lsmod                     Lists the module loaded
  -g|--group <name> "<node>,..."  Creates a group named <name>
  -n|--lsnodes                   Lists the running nodes
  Commands are sent to the given hosts, or to every known host if no host is explicitl
```

```
-r and -lr take additionnal parameters given to the module.
```

Chapter 6

Questions & Answers

- I get an abort on the last instruction of a C++ code.
  Tip: Sure there is a throw in the C++ code. As `padico_module_run` is called by a C code, the abort occurs when the unwrapping of the stack can no longer go on.
- My C++ code does not seem to load/start/unload.
  Tip: Do not forget to declare `padico_module_init` and the likes as `extern "C"`!
- Could I share the CORBA NameService between 2 sites running PadicoTM?
  Tip: `padico-run` lists the known running nodes through the use of a CORBA NamingService managed by Ugo. Chose which is site 1 and which is site 2. Each command given here should be executed on the appropriate site. On site 1, start a name server if none is running: `mycorba-ns --start`. On site 2, stop the name server if it is running: `mycorba-ns --stop`. On site 1, get the reference of the name server: `mycorba-ns --get-ref`. On site 2, register this reference: `mycorba-ns --register <copy/paste ref of site 1 here>`.

Appendix A

License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in

whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for

making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other

circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program 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 General Public License for more details.

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

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your

school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  'Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendix B

Advanced configuration

B.1 PadicoTM framework advanced configuration

This section describes the custom installation and advanced configuration mechanisms; it is an alternate to the quick install described in Section 2.3.1. Most users may safely skip this section.

Directories. Each Padico package uses three directories:

the source directory is where you unpacked the Padico packages. It is referred to as `PadicoTM-0.5.0/` in this manual. For example, the source directory for PadicoTM is `PadicoTM-0.5.0/PadicoTM/`;

the build directory is where you build Padico packages. It is a user-created directory where you run the `configure` command. Each Padico package should have its own build directory. Padico does not allow the build directory to be the same as the source directory, i.e. `cd PadicoTM-0.5.0/PadicoTM; ./configure` is not allowed;

the install directory is where Padico is installed. It is chosen through the `--prefix=` flag for `configure`. It is referred to as <*padico_root*> in this manual. We require that all Padico packages have the same install directory, i.e. you should give the same `--prefix=` configuration flag to all Padico packages. In order to use Padico, the environment variable `$PADICO_ROOT` must be set to point to the directory where Padico is installed. It is usually done by the `etc/padico-setup.{csh|sh}` script (see Section 2.3.2).

The source and build directories are used to compile Padico and its services. It is safe to delete them only if you do not plan to build any new service.

Using an existing PM^2 installation. the Padico package contains its own version of PM^2. However, it is possible to use your own PM^2 installation. Make sure that the environment variable `PM2_ROOT` is set and points to a valid PM^2 installation before running `padico-install` or configuring PadicoTM. Padico will generate on-the-fly PM^2 flavors when you setup a new PadicoTM core. PM^2 flavors for Padico are stored in <*build-dir*>`/PadicoTM/opt/pm2/`.

Step by step installation of Padico packages. The first step consists in installing the common parts. For example, if the sources are in `$HOME/PadicoTM-0.5.0`, to be build in `$HOME/PadicoTM-0.5.0/build`, and to install Padico in `$HOME/Padico`, type:

```
cd PadicoTM-0.5.0

admin/padico-init $HOME/Padico
```

to create the Padico installation directory and to copy the installation tools. You may need to tweak variables stored into `$PADICO_ROOT/etc/padico-install.conf`. Then we configure, build and install *my*CORBA:

```
mkdir build/<arch>/myCORBA
cd build/<arch>/myCORBA
make install
```

Notice that we provide the same prefix as in the first step. *my*CORBA has to be installed first. Then we setup *my*CORBA. The setup assistant is in `PadicoTM-0.5.0/myCORBA/package-setup`. See Section 4 if you want to manually setup *my*CORBA with the `mycorba-admin` command.

The `configure` script for PadicoTM takes the well-known flags:

`--prefix=PREFIX` install Padico in `PREFIX` directory. `PREFIX` *must be an absolute path*;

`--{enable|disable}-optimize`

`--{enable|disable}-debug`

`--{enable|disable}-trace`

plus the PadicoTM-specific flag to statically link the BIP Myrinet driver

`--{enable|disable}-bip`

Using alternative compilers. It is possible to explicitly specify which compiler to use. For example, we can type:

```
setenv CC gcc-3.0.4
setenv CXX g++-3.0.4
setenv JAVAC $HOME/bin/jikes
setenv JAVA /soft/jdk/bin/java
setenv JDK /soft/jdk/
```

and then quick install with `./padico-install` or custom install by beginning with `admin/padico-init`.

B.2 Custom core & services setup

The core assistant takes some decisions in order to simplify the core & services setup. For a full customized setup, you should build separately each part. Custom setup is often the preferred way when you want to create several core flavors. If you do not know whether to use the core assistant or the custom core setup, then you probably want the core assistant described in Section 2.3.3! This section describes how to do by hand what is usually done by `padico-core-assistant` and `padico-corba-startup`.

Build a Padico core. First, we build a PadicoTM core. For example, we choose here a flavor tuned for mono-processor Myrinet machines including full optimization, without debugging symbols. We chose to call it `mono-bip`. The command line is:

```
padico-mkcore mono-mx --mad-mx --marcel-mono --pm2-opt
```

`--mad-mx` tells Padico to use the BIP/Myrinet driver. Other available choices are `--mad-sci` for SISCI/SCI, `--mad-tcp` for TCP/IP.

`--marcel-mono` tells Padico to use the mono-processor flavor of Marcel. Use `--marcel-smp` for multi-processor machines.

`--marcel-smp` is not fully supported yet.

`--pm2-opt` tells Padico to use the optimized flavor of PM^2 (i.e. with the `-O4` compiler flag). Other available choices are `--pm2-g` to include debug symbols (i.e. `-g` compiler flag) and `--pm2-inherit` for the same debug/optimize flags as global Padico configuration. The core assistant defaults to `--pm2-inherit`.

Build additional services. We have now a PadicoTM core named `mono-bip`. The core by itself is pretty useless without other services such as CORBA, MPI, etc. In this example, we will build the CORBA service. The name of the service is `ORB`. It requires the `VIOk` service. The basic remote control is `CORBA-Gatekeeper`. The command line looks like this:

```
padico-mkservice --padico-core mono-bip --full-build VSock ORB CORBA-Gatekeeper
```

It can take a long time (e.g. 20 minutes) to build the `ORB` service.

The services, like any PadicoTM module, are core-specific, i.e. services are built for a given PadicoTM core flavor. Almost each command takes a `--padico-core <core>` flag to specify which core is concerned. It may be omitted if you set the environment variable `PADICO_CORE`.

Make a startup module We now build a startup module which contains all the modules to automatically load at boot time. The syntax is a little bit tricky :-) This command lines builds a module named `mono-mx-with-CORBA` which contains the PadicoTM core `mono-bip` – it contains support for Myrinet, mono-processor threads, and is an optimized flavor –, the `ORB` module with its required module `VSock`, and a basic CORBA gatekeeper named `CORBA-Gatekeeper`. It contains a reference to the already-running CORBA name service, assuming it has been started with `mycorba-ns` (see section 4.2) or referenced by *my*CORBA.

```
padico-mkmod --padico-core mono-mx --driver=pkg --bootable \
  mono-bip VSock ORB CORBA-Gatekeeper -o mono-bip \
  -DNameService=`mycorba-ns --getref`
```

Do not forget the `--bootable` flag, otherwise the module would not be bootable and would not be suitable to play the role of a startup module! The `--driver=pkg` flag tells Padico that this module is only made out of other modules and contains no code. The `NameService` attribute is mandatory for CORBA.

Appendix C

Hello CORBA Examples

C.1 The Hello-CORBA client (`Hello-client.cc`)

```
/* Padico tutorial 3: "CORBA client"
 * PadicoTM/Examples/Tutorial/03-HelloCORBA/Hello-client.cc
 */

#include <iostream>
#include <Padico/CORBA.h>
#include <Padico/COSNaming.h>
#include <Padico/Module.h>

#include "Hello.h"

PADICO_MODULE_DECLARE(Hello_client);
PADICO_MODULE_INIT(client_init);
PADICO_MODULE_RUN(say_hello);

static Padico::ORB_ptr                orb;
static CosNaming::NamingContext_ptr   naming_context;
static Hello_server_ptr               server;
static CosNaming::Name                server_name;

int client_init(void)
{
  orb = Padico::ORB_init();

  try
    {
      CORBA::Object_var ns_ref =
orb->resolve_initial_references ("NameService");
      naming_context = CosNaming::NamingContext::_narrow (ns_ref);
    }
  catch(...)
```

```
    {
      padico_print("Hello-client: cannot resolve NameService.\n");
      return -1;
    }

  server_name.length (1);
  server_name[0].id = CORBA::string_dup("default");
  server_name[0].kind = CORBA::string_dup("HelloServer");

  try
    {
      CORBA::Object_var server_obj = naming_context->resolve(server_name);
      server = Hello_server::_narrow(server_obj);
    }
  catch(...)
    {
      padico_print("Hello-client: cannot resolve HelloServer in NameService.\n"
      return -1;
    }

  return 0;
}

int say_hello(int argc, char**argv)
{
  try {
    server->Hello();
  }
  catch(...)
    {
      padico_print("Hello-client: exception while invoking Hello server\n");
      return -1;
    }
  return 0;
}
```

C.2 The Hello-CORBA server (`Hello-server.cc`)

```
/* Padico tutorial 3: "CORBA server"
 * PadicoTM/Examples/Tutorial/03-HelloCORBA/Hello-server.cc
 */

#include <iostream>
#include <Padico/CORBA.h>
```

```
#include <Padico/COSNaming.h>
#include <Padico/Module.h>
#include "Hello.h"

PADICO_MODULE_DECLARE(Hello_server);
PADICO_MODULE_INIT(server_init);
PADICO_MODULE_FINALIZE(server_finalize);

class Hello_server_impl : virtual public POA_Hello_server
{
public:
  void Hello(void)
  {
    ::std::cout << "**********************************" << ::std::endl;
    ::std::cout << "********** Hello world! **********" << ::std::endl;
    ::std::cout << "**********************************" << ::std::endl;
  }
};


static Padico::ORB_ptr                orb;
static PortableServer::POA_var        poa;
static PortableServer::POAManager_var poa_manager;
static CosNaming::NamingContext_var   naming_context;
static Hello_server_impl*             server;
static PortableServer::ObjectId_var   server_oid;
static CosNaming::Name                server_name;

int server_init(void)
{
  orb = Padico::ORB_init();

  CORBA::Object_var poa_ref = orb->resolve_initial_references ("RootPOA");
  poa = PortableServer::POA::_narrow (poa_ref);
  poa_manager = poa->the_POAManager();
  poa_manager->activate();

  CORBA::Object_var ns_ref =
    orb->resolve_initial_references ("NameService");
  naming_context = CosNaming::NamingContext::_narrow (ns_ref);

  server = new Hello_server_impl;
  server_oid = poa->activate_object(server);

  server_name.length (1);
  server_name[0].id = CORBA::string_dup("default");
  server_name[0].kind = CORBA::string_dup("HelloServer");
```

```
  naming_context->rebind(server_name, server->_this());
  return 0;
}

void server_finalize()
{
  naming_context->unbind(server_name);
  //    poa->deactivate_object(server_oid);
  delete server;
}
```