The Programmers' Playground
Application Management System User Guide

William M. Shapiro, T. Paul McCartney,
and E.F. Berkley Shands

WUCS-97-32

August 1997

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Application Management User Guide

William M. Shapiro, T. Paul McCartney, E.F. Berkley Shands

*Revised for **Application Management v2.1.0***

August 1997

Distributed Programming Environments Group
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Abstract

Application Management permits the advertising, launching, and configuring of distributed applications created using The Programmers' Playground. Applications can be documented and made available to end-users through the use of application pages on the World Wide Web. The launching and configuring of applications is performed by a brokerage system consisting of an application broker and one or more hierarchies of module launchers. This document describes how to setup and use the components of the Application Management system.

Table of Contents

Chapter 1

Introduction

This manual assumes the user is familiar with the basic concepts of The Programmers' Playground. An introduction to Playground can be found in [1], [2].

Application Management refers to a software approach for automating the process of launching and configuring complete distributed multimedia applications created using The Programmers' Playground. With such a software system, end-users can use complex distributed multimedia applications without knowing the internal details of the application. For example, an application may have the requirement that a certain component must operate on a particular file system in order to access a locally stored database. Exposing this information directly to the end-user is undesirable since it forces the end-user to be knowledgeable about the application's internal details and the current performance characteristics of the resources that may be used. Instead, it is better to leave the launching details to the Application Management system described in this manual.

1.1 Application Pages

End-users interact with the Application Management system through the use of *application pages*. Each application page is a World Wide Web document that is viewable through the use of a Java-enabled web browser (e.g., Netscape Navigator). Application Pages provide both application documentation and a end-user mechanism for launching and configuring complete distributed applications. Figure 1 contains an example application page for the Medical Image Filtering application.

1.2 Application Management System Components

Figure 2 shows the relationship between the Application Management system components. The *application directory* is a world-readable web directory tree that contains application specific information for one or more applications, including the application's web page, launch specification, and information about running application instances.

The *Application Daemon* module manages and controls access to a set of application directories. Information about launched applications is communicated to the Application Daemon, which writes the data to the appropriate application directory.

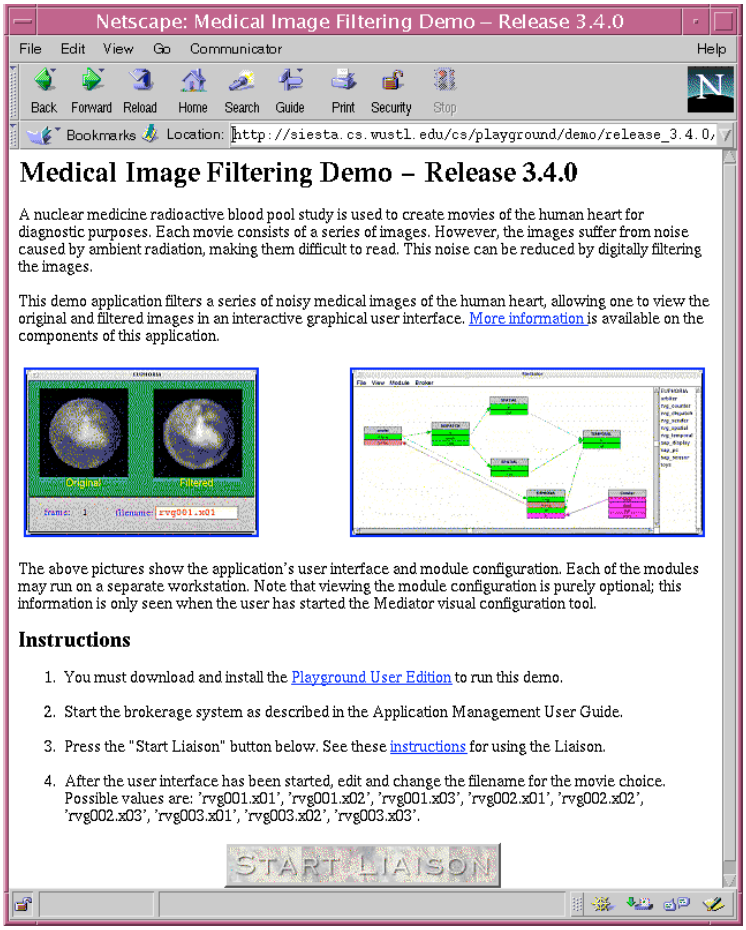

Figure 1: Example Application Page: Medical Image Filtering

When an application page is viewed with a Java-enabled Web browser, a Java program called the *Liaison* applet is run. The Liaison periodically scans the application directory and displays a list of all currently running application instances. Users can interact with the Liaison's GUI, specifying requests to be sent to the brokerage system (e.g., a "launch application" request, see Section 2.4).

The brokerage system, which consists of a *Broker* module and a hierarchy of *Launcher* modules, controls the launching and configuration of application modules. Each end-user runs their own personal copy of the Broker. In response to a Liaison request, the end-user's Broker forms connections to the appropriate Application Daemon in order to receive application information and send information about newly launched applications. Module specifications and application configuration information are read by the Application Daemon from the application directory and communicated to the Broker. The Broker delegates the task of module launching to its Launchers, gathers module IDs of newly launched modules, configures the communication among modules, and communicates the set of IDs back to the Application Daemon. In addition, the *Mediator* visual configuration tool [4] can be used to view, manipulate and launch modules in conjunction with the Broker.

The Launcher is responsible for launching a collection of modules and optimizing module allocation to processors according to heuristics based on module performance requirements, hardware configurations, and other factors. Each Launcher controls access to some portion of the total computing resource,

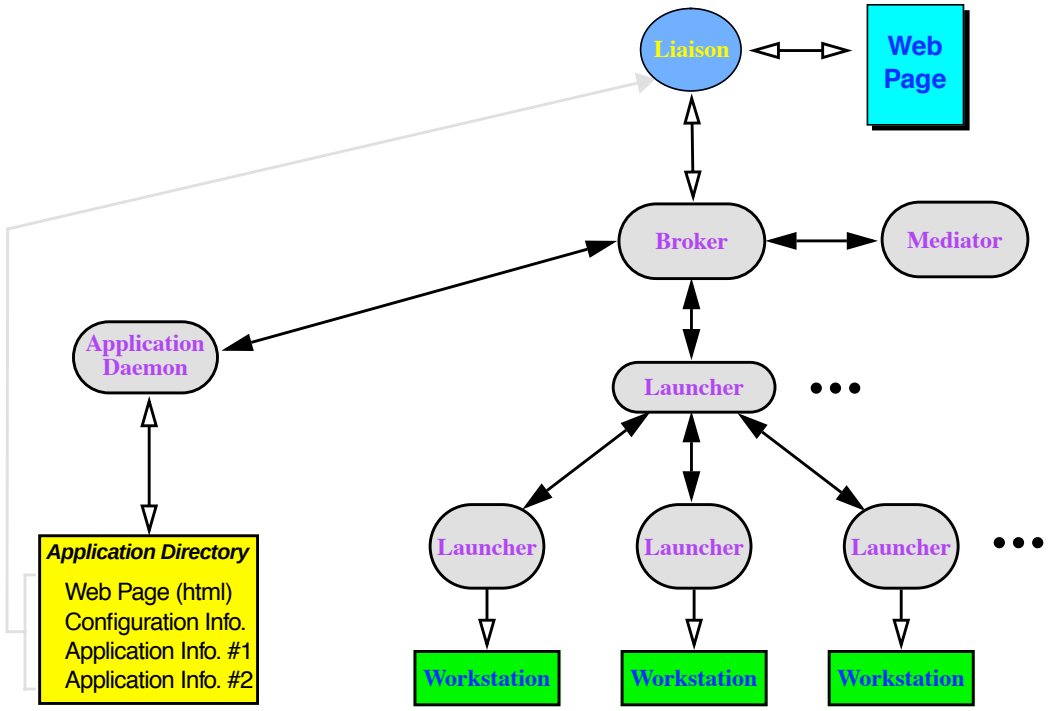

Figure 2: The Application Launch architecture.

including other Launchers and physical CPUs. Launchers may form a hierarchy, as seen in Figure 2. "Leaf" Launchers gather information about a single workstation (e.g., its current load, list of supported modules) and launch modules on that workstation. Other Launchers manage a collection of child Launchers, summarizing child information to parents and delegating the launching task to children. A Launcher may conceal the details of resources it controls, acting as a firewall to those resources.

1.3 Overview

The remainder of the manual is organized as follows. Chapter 2 provides quick-start instructions for using Application Management. Chapter 3 describes the setup and customization of the brokerage system. Chapter 4 explains how to create, and allow others to use, application pages on the World Wide Web.

Throughout this manual, the directory of "/home/fred/PG" is used for demonstration purposes as the base directory of the Playground User Edition installation. Additionally, the UNIX file separator "/" is used throughout the manual. For Windows NT, the file separator "\" should be used instead.

Also, this manual refers to a directory called ".pgdir." This directory and its contents are created in the user's home directory the first time that the Application Management system is used.

Chapter 2

Using Application Management with the World Wide Web

This chapter provides step-by-step instructions for starting the Application Management system for use with the World Wide Web, assuming that the Playground User Edition has been installed.

Starting the Application Management system can be divided into four steps:

1. Start one or more Launchers.
2. Start a Broker on the local machine
3. Start a Mediator (optional).
4. Using Netscape Navigator version 4, go to an application page and press the "Start Liaison" button.

2.1 Starting a Launcher

At least one Launcher must be running in order to launch modules. The Launchers may be run on either remote or local computers; however, you must run at least one local Launcher. The performance of graphical modules (e.g., EUPHORIA) is superior when running locally. A Launcher should be running on each computer that you intend to launch modules. This may include "remote launchers" managed by other users (see Section 2.5).

To start a Launcher, you must know the root of a launch tree of available modules (see Section 3.1). In the Programmers' Playground user edition, a "launchTree" directory is created upon installation. You may provide this directory as a command-line argument to the Launcher:

UNIX
```
PGlauncher -base /home/fred/PG/launchTree
```

NT
```
start PGlauncher -base  C:\home\fred\PG\launchTree
```

or, alternatively, you may specify the root of the launch tree in *Launcher.Policy* file (see Section 3.4) and start the Launcher without arguments.

2.2 Starting the Broker

In order to use Application Management, the user must have a Broker running on the local machine. The user may start the Broker with the command:

UNIX
```
PGbroker
```

NT
```
start PGbroker
```

2.3 Starting the Mediator

The Mediator may be used to view, launch and visually configure Playground modules. Application modules initiated by your Broker will appear in your Mediator GUI. The Mediator may be started with the command:

UNIX
```
PGmediator
```

NT
```
start PGmediator
```

A Mediator containing the modules for a medical filtering application is shown in Figure 3.

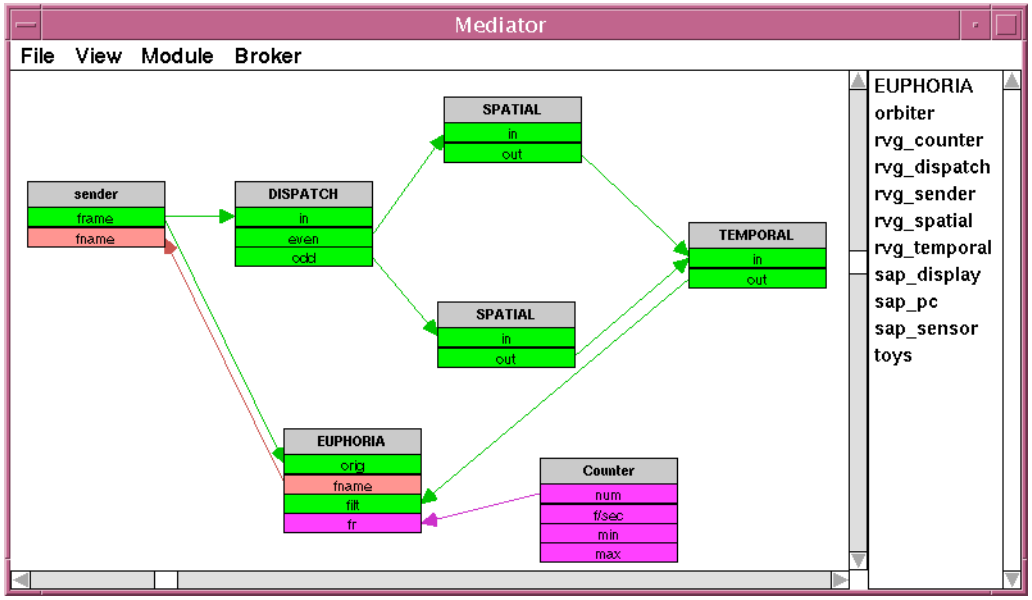

Figure 3: Mediator Containing Modules for a Medical Image Filtering Application

2.4 Using the Liaison

⚠ Currently, you must be running Netscape Communicator version **4.0** or above to use the Liaison applet.

Each application page contains a button to expose the Liaison applet window (see Figure 1). Figure 4 shows an example Liaison window. The Liaison contains a list of running application instances and several buttons.

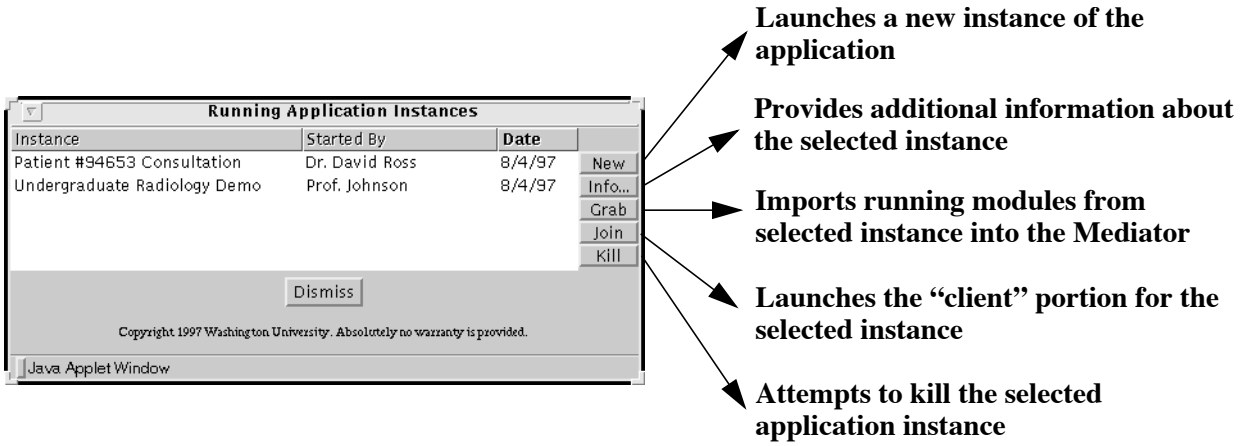

Figure 4: Liaison Window

The NEW option launches a new instance of the given application. When the NEW button is pressed, the box shown in Figure 5 pops up, requesting information from the user. The user must provide a name for the instance. Additionally, the user may provide the name of the person or group that started the instance and documentation information. The documentation information may either consist of text or a URL with further information. Finally, the user may specify whether the instance should be made public (i.e., viewable by other users in the Liaison window).

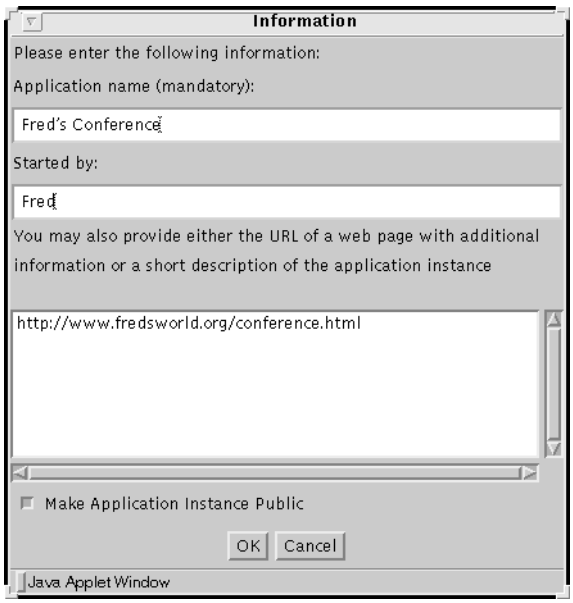

Figure 5: Information Requested for a New Application Instance

To use any button other than NEW, the desired application instance in the list must first be selected. The INFO option provides additional information about the selected application instance by either displaying text or a web page (supplied previously when the NEW command was executed). The GRAB option imports the already running modules of the selected application into the Mediator (see Figure 2.3). The JOIN option launches the "client" portion of the selected "server" application. One may wish to "join" a teleconferencing application, for example. Not all applications are set up to be joined. Finally, the KILL option attempts to shut down the selected application instance.

Due to performance issues, the Liaison's table of running applications is only updated every five minutes. That is, applications that are exited in some other way than using KILL may continue to appear in the Liaison table for several minutes.

Applet Security

The Liaison is a signed applet (signed by the authors of this manual), which needs to interact with the user's computer. The first time the NEW, GRAB, JOIN or KILL button is pushed, several security dialog boxes will pop up the screen, requesting permission to read files on the user's file system, read system properties and connect to other computers. If all security permissions are not granted, the Liaison will not able to send requests to the Broker. A description of each permission requested is provided in Table 1.

Permission	Description	Why Needed
UniversalPropertyRead	Reading information stored in your computer that is normally kept private, such as your user name and the current directory.	Required to find your home directory, where the file containing the Broker's location is stored.
UniversalFileRead	Reading any files stored on hard disks or other storage media connected to your computer.	Required to read the file that stores the location of your Broker.
UniversalConnect	Contacting and connecting with other computers over a network.	Required to connect to your Broker

Table 1: Security Permissions Requested by the Liaison Applet

2.5 Using Remote Launchers

Sometimes users may wish to use launchers that are running on remote systems. Application Management allows users to specify remote launchers in a "RemoteLaunchers" directory. To specify a remote launcher, the user must save a remote launcher file (created by the remote launcher provider, see Section 3.5) into their ~/.pgdir/RemoteLaunchers directory. When the Broker is started, it searches this directory for available launchers, and connects to those launchers. To remove a remote launcher, simply remove the remote launcher's file from the directory. Each remote launcher file will contain instructions on how to add it to your list of remote launchers. An example remote launcher file is provided below:

```
# Example Remote Launcher file: example.url
# To use this launcher save this file in your
# ~/.pgdir/RemoteLauncher directory

http://www.fredsworld.org/fred/my_launcher.pgmid
```

Chapter 3

Customizing the Launch System

The brokerage system is responsible for launching and configuring Playground applications. It consists of a Broker module, which is run on the user's local machine, and one or more Launcher modules (see Figure 2).

Customizing the launch system for your computers/filesystem involves:

1. Creating and configuring one or more launch trees
2. Creating a policy file for the Broker
3. Creating a policy file for the Launcher

3.1 Creating and Configuring the Launch Tree

A launch tree is a directory that contains information about module executables, module public names, filesystem specific settings and other information. The launch tree consists of a root launcher directory that may contain any number of module directories and sublauncher directories. Each module directory contains information describing where the executable for the module is located and additional module properties. Each sublauncher directory represents the root of a separate launch tree. A sublauncher may contain additional module directories and sublauncher directories.

The Playground User Edition release will install a launch tree containing the EUPHORIA module as shown in Figure 6.

3.1.1 Module Directories

Each module that will be used with the Application Management system must have a module directory. The name of each module directory is arbitrary except that it must end with the suffix *.module* (e.g., foo.module). Additionally, the module directory must contain a *.info* file with the same name as the directory (e.g., foo.info).

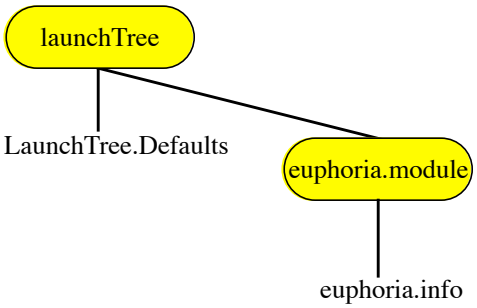

Figure 6: Launch Tree Installed in the Playground User Edition Release

3.1.2 Info File

Currently, the designer must write each <module>.info file. A graphical interface for writing this file will be provided in a future release.

The *<module>.info* file provides machine and/or filesystem-specific information needed to launch a module. A *<module>.info* file must be provided for each module represented in the launch tree. An example *<module>.info* file is provided below:

```
PG_MOD_SPEC 1 0 # This is required!

# mandatory data
#
map EUPHORIA.dpe.cs.wustl.edu /home/fred/PG/bin/PGeuphoria

# optional data
#
launch fork
directory /home/fred/

# logging info
#
logto EUPHORIA.log
```

Each *<module>.info* file must contain a file header and the mapping between the module's public name and the module executable location. Additionally, each file may include additional options (explained below) and comments beginning with "#".

Public Name

A module is identified by its unique public name, a string that should identify the module, its publisher, and something about its purpose. A suggested convention is <ModuleName>.<Group>.<Host>. For example, EUPHORIA's public name is "EUPHORIA.dpe.cs.wustl.edu". The public name cannot contain whitespace and is not case sensitive.

The launch system is responsible for finding an appropriate copy (i.e., a machine specific copy) of each module executable at launch time. On UNIX systems, the public name may contain wild card characters which will be matched with the available modules from the launch tree. It is then possible to code the application specification (see Section 4.4) to request a generic public name and let the launch system find an appropriate match. For example, "MathFunctions.*.cs.wustl.edu" could match

"MathFunctions.Generic.dpe.cs.wustl.edu" or "MathFunctions.DEC-ALPHA.dpe.cs.wustl.edu". If there is more than one match, the first match will be chosen.

The mapping from the module public name to the executable location is required. The executable location is specific to the filesystem of the running module. For example, if the public name of a module is "mmx.dpe.cs.wustl.edu" and the path of the executable is "home/vaudeville/bin/mmx," then the mapping would be:

```
map mmx.dpe.cs.wustl.edu /home/vaudeville/bin/mmx
```

The path of the executable may also be local (e.g., mmx instead of /home/vaudeville/bin/mmx) if the executable is stored in the same directory as the <module>.info file and the "directory" option (see Options below) is not set to different directory.

Options

The options section allows the designer to specify additional properties of the module that may not have been specified elsewhere. Options may be specified in arbitrary order.

- **args** - Allows the user to specify command-line arguments to send to the launched modules.

  ```
  args <string>
  ```

- **env** - specifies the UNIX shell dependent command to use to set environment variables (on Windows NT this option is ignored).

  ```
  env ( setenv | set )
  ```

- **launch** - Specifies whether the module should be launched via fork()/exec() or by a system() call (On Windows NT, all calls are done with CreateProcess()). By default modules are launched via fork()/exec().

  ```
  launch ( fork | system )
  ```

- **var** - Allows the user to specify environment variable/value pairs needed by the module

  ```
  var <variable1> <value1>
  var <variable2> <value2>
  ...
  ```

- **notpg** - Specifies the module is not a Playground module, preventing the module ID from being reported to the Application Daemon.

  ```
  notpg
  ```

- **parent** - Specifies the module is a "parent," causing subsequent modules to bootstrap to it.

  ```
  parent
  ```

- **directory** - Specifies the module's working directory.

  ```
  directory <directory path>
  ```

- **private** - Specifies the module is launcher PRIVATE, meaning only the local launcher will know about the module (it will not be propagated to other modules). This option is useful for debugging modules.
  `private`
- **join only** - Specifies that the module is long-lived (e.g., a server) and should not be killed by the Application Daemon.
  `joinonly`
- **log info** - Specifies a log file for the module
  `logto <filename>`
- **remote host** - specifies the host the module should be launched on.
  `host <host name>`

3.1.3 LaunchTree.Defaults File

A *Launcher.Defaults* file may be placed in the root directory of a launch tree (see Figure 6). The file contains default values for all modules in the launch tree. The format of the Launcher.Defaults file is the same as the <module>.info file and may contain the same options. Values specified in the <module.info> file overrule those in the Launcher.Defaults file. An example file is presented below:

```
PG_MOD_SPEC 1 0

# mandatory data - LaunchTree.Defaults
#
map Mandatory Mandatory-but-unused

# optional data
#
var LD_LIBRARY_PATH /home/fred/PG/lib
```

In this example, because the dynamically linked library path is the same for all modules, the environment variable for the library path is set in the defaults file.

3.1.4 Sublauncher Directories

Each sublauncher directory is named for the machine it is to exist on. For example, if we wish to create a sublauncher directory for the machine "trampoline," we would create a subdirectory named "trampoline.sublauncher." The directory would then contain directories for any modules that are located on trampoline and possibly other sublauncher directories. When a launcher starts, it checks for sublauncher directories in its launch tree. For each sublauncher directory without a running launcher, the root launcher attempts to start an appropriate launcher. The root launcher must be able to do a remote login to the sublauncher's computer without the need to enter a password.

3.2 Configuring the Broker

The Broker is responsible for delegating the launching of an application to one or more Launchers. It receives requests from either the Liaison, the Mediator, or its Playground interface.

Each user must launch his or her own Broker on the local machine. If more than one Broker exists for a given user, each Broker must have a different "DISPLAY" value, which the Broker determines by reading the "DISPLAY" environment variable. If the "DISPLAY" environment variable is not defined, the Broker will create one based on the current host name.

All configuration information for the Broker is stored in the file *Broker.Policy*, which is located in the users *.pgdir* directory (~/.pgdir/Broker.Policy). The *Broker.Policy* file is free-form meaning that the order of each item does not matter. Additionally, comments beginning with "#" and extending to the end of the line are permitted throughout the file. An example policy file is provided below:

```
#  Example Broker Policy File

launcherbase /home/fred/PG/launchTree

overrideretry false
retry softfail
launchpolicy complex
socket 32145
```

The user may specify the following properties (in any order) in the *Broker.Policy* file:

- **launcherbase** - Specifies the base directory of all auto-launched launchers the Broker initiates:

  ```
  launcherbase <root directory>
  ```

  where <root directory> is the root of the launch tree (see Section 3.1). The launcherbase may also be specified by setting the environment variable "PGLAUNCHERDIR" or with a command-line-argument (see Section A.1).

- **socket** - Specifies the socket to which Broker should listen for requests from the Liaison. The form is:

  ```
  socket <integer>
  ```

  where <integer> is a valid socket port (between 1024 and 65535). If <integer> is specified to be 0, the Liaison interface will be disabled.

- **retry** - The Broker's default retry policy:

  ```
  retry ( none | abort | rotate | broker | softfail )
  ```

  A retry occurs when a module cannot be launched given the current constraints (e.g., there is excess load on the desired launcher). Several retry options are available if module placement fails. "Softfail" will drop performance requirements and retry the launch. "Abort" simply stops and does not retry. "Rotate" tries the next available Launcher until its retries are used up. "Broker" returns the request back to the Broker (used with a Liaison). The retry option specified will be used if either the application designer did not specify a retry policy or the "overrideretry" option is set.

- **overrideretry** - Specifies whether the Broker should ignore the designer's retry policy and, instead, use the retry option specified in *Broker.Policy*:

```
overrideretry ( true | false )
```

- **logging** - Specifies the file where the Broker will log all errors and requests:

```
logging <file path>
```

- **launchpolicy** - Specifies the type of launch policy for the Broker:

```
launchpolicy ( simple | complex )
```

The default launch policy, "simple", chooses the best machine for each individual module. If "complex" is specified, the Broker attempts to balance the group of modules based on performance and load specifications.

- **fixedpid** and **fixedsocket** - Specifies default values for the process ID (pid) and socket for used for Playground communication. For example,

```
fixedsocket 8765
fixedpid 555
```

The user may choose a socket value between 1024 and 65535 for the underlying Playground veneer to use for communication. Similarly, the user may specify a process ID (pid) between 1 and 1024 the Broker should use. Specifying both a fixedsocket and a fixedpid provides a fixed address that can be published.

- **launchwait** - Specifies the time the launch system should wait for a module to be launched. The default value is 20 seconds. The maximum value is 60 seconds. If a module takes a long time to start, the user may wish to increase the launchwait. For example,

```
launchwait 60
```

Finally, the *Broker.Policy* file may contain a rejection section that allows the user to specify which hosts or users should be blocked and the rejection messages that should be sent to these users. For example:

```
SECTION rejections
host flimflam.edu user JoeUser reject message Go Away!!!!!
host nowhere.com user fred accept message required
host nowhere.com user * reject message you're not fred!!
user bully reject message bullies are not welcome here
END_SECTION
```

The designer may specify an arbitrary number of rejections. Each rejection optionally begins with the host to be rejected then specifies the user to be rejected and the rejection message. The "*" denotes wildcard. Full file name wildcarding is supported under UNIX.

3.3 Running the Benchmarker

The benchmarker uses a set of benchmarks to generate statistics describing the performance of a given host. The statistics are used by the launch system to optimize module placement across computers. The benchmarker need only be run once for each computer used by the Application Management system. To run the benchmarker, go to the "bin" directory of the release (e.g., /home/fred/PG/bin) and execute the command:

UNIX
```
PGbenchmarker
```

NT
```
start PGbenchmarker
```

The benchmarker will create a file <host>.hwinfo in the directory from which it is run. Launchers look for this file in the "bin" directory.

3.4 The Launcher's Policy File

Like the Broker, the Launcher also has a policy file that is located in the user's Playground directory (~/.pgdir/Launcher.Policy). An example *Launcher.Policy* files is provided below:

```
# Example Launcher Policy File

launcherdir /home/fred/PG/launchTree

overrideretry false
hiding false
retry softfail
launchpolicy simple
```

The *Launcher.Policy* file may contain the root of the Launcher hierarchy which is specified as:

```
launcherdir <path>
```

Additionally, the designer may specify that the Launcher "hide" all host information about the module by stripping the host name from published variables of the outgoing module. To specify that "hiding" should take place, the following line should be added:

```
hiding true
```

Finally, as in the *Broker.Policy* file (see Section 3.2), the designer may specify the Launcher's:

- overrideretry policy
- retry policy
- launchpolicy
- logging
- fixedpid and fixedsocket
- rejection section

3.5 Advertising Remote Launchers

Computers may be used as launch service providers, allowing remote users to launch modules on the designated machines. The Launcher ID of each Launcher serving as a provider must then be made available to remote users (see Section 2.5). Application Management provides two mechanisms for advertising Launcher IDs: directly, by creating a file with the suffix ".pgmid" containing the Launcher's Playground module ID [1] and indirectly, by publishing the URL of the file with the suffix ".url."

Launch service providers can put their .pgmid and/or .url files in a well known location, such as the World Wide Web, where users can easily locate and download the files. The .pgmid option is more efficient, giving direct access to the launcher's location. The second option is more flexible because it allows the Launcher's ID to change by providing an extra level of indirection. In general, if the Launcher is a "system" launcher (i.e., it has a fixed address), then the first option is preferable.

An example .pgmid file is provided below:

```
# Example *.pgmid file: foo.pgmid
# To use this Launcher, save this file
# in your ~/.pgdir/RemoteLaunchers directory

128.252.165.86,26091:SCK55549
```

The Playground module ID of a launcher is printed when it is started. Creating the .pgmid file simply involves copying this ID into a file and (optionally) adding comments.

An example .url file is provided below. This file should give the URL of a .pgmid file stored in a World Wide Web readable location.

```
# Example *.url file: foo.url
# To use this Launcher, save this file
# in your ~/.pgdir/RemoteLaunchers directory

http://www.fredsworld.org/fred/foo.pgmid
```

Chapter 4
Creating and Serving Application Pages

The Application Daemon and Liaison allow end-users to initiate launching of distributed applications from the World Wide Web. Users who solely plan to use either the Mediator or the Playground interfaces of the brokerage system to launch applications may skip this chapter.

Making your application available on the World Wide Web for others to use requires:

- Creating an application directory
- Creating an application description file that contains module and configuration information
- Creating the Application Daemon's policy file
- Running the Application Daemon to manage the application directory

4.1 Creating the Application Directory Structure

The application directory hierarchy must reside in a world-readable directory (WWW directory) containing all information specific to each application. The application directory structure consists of a root directory and an arbitrary number of application subdirectories. Each application subdirectory must contain an application description file called *new.spec*. An example application directory is shown in Figure 7.

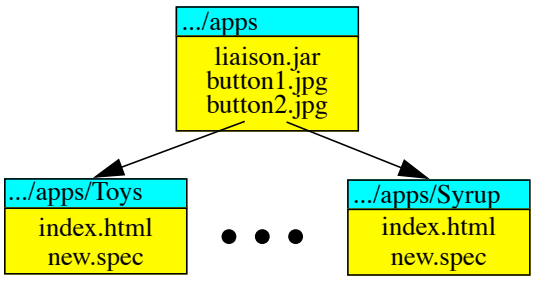

Figure 7: Example Application Directory Structure

Publishing a new application on the web for use with the Application Management system requires creating a new application subdirectory containing an application page (HTML), a new.spec file, and whatever other files are associated with the application (e.g., EUPHORIA files [3]).

4.2 Creating the Application Page Root Directory

The root application page directory contains several files and all application page subdirectories. To create the root directory:

1. Create a directory in your local World Wide Web directory and set the directory's permissions to be world-viewable.
2. Copy all files provided in the "liaison" directory of the distribution (e.g., /home/fred/PG/liaison) to the directory you just created and set the permissions on all of these files to be world-readable.

4.3 Creating Application Pages

Creating an application page requires creating a HTML web page and adding the Liaison applet tag to the page. The application page would likely contain information about the application and possibly documentation for using the application (see Figure 1).

The applet tag to be added to an application page is provided below:

```
<APPLET code="Liaison.class" archive="liaison.jar" codebase=".." WIDTH=240 HEIGHT=40>
        <PARAM NAME=buttonImage VALUE="button1.jpg">
        <PARAM NAME=pressedButtonImage VALUE="button2.jpg">
        <PARAM NAME=showRunningApps VALUE=”true”>
</APPLET>
```

The user can customize the following attributes of the applet tag:

- **buttons** - The user may optionally add the parameters "buttonImage" and "pressedButtonImage" to specify the button images that appear in the Liaison window. The images must then be stored in the root application directory and must be world-readable.

  ```
  <PARAM NAME=buttonImage VALUE="<FIRST IMAGE>">
  <PARAM NAME=pressedButtonImage VALUE="<IMAGE WHEN PRESSED>">
  ```

- **showRunningApps** - The user may specify whether application instances should be public (i.e., shown in the list of running applications). By default application instances are shown.

  ```
  <PARAM NAME=showRunningApps VALUE=”<true or false>”>
  ```

- **WIDTH** and **HEIGHT** - The user may specify applet button's the width and height.

4.4 Creating the Application Description

Each application directory must contain a file named *new.spec* that specifies the modules that comprise the application and how the modules are configured. The user creates a *new.spec* file for an application by launching and configuring the application in the Mediator [2] and saving the configuration to a file called new.spec. The file grammar for an application description is provided in Appendix C for developers who would like to generate the new.spec file some other way.

In addition, an application directory may also (optionally) contain a file named *join.spec* that specifies the modules that comprise the application client portion and the connections to the server portion (i.e., modules specified in the new.spec file). Currently, join.spec files cannot be created directly by the Mediator. In the future, the Mediator will provide such a mechanism.

For client-server applications, join.spec and its associated new.spec can be created in the following way:

1. In the Mediator, launch and configure both the server portion and one instance of the client portion.
2. Save the configuration as “new.spec”
3. Create a copy of the new.spec; call the copy “join.spec”
4. In the new.spec file, delete each “MODULE” section (see Appendix C) of client modules.
5. In the new.spec file, delete each connection in the “CONNECTIONS” section that refers to client modules. These can be identified by the fromID and toID of the associated modules.
6. In the join.spec file, delete each “MODULE” section of server modules.
7. In the join.spec file, delete each connection in the “CONNECTIONS” section that refers to strictly server connections (i.e., both from and to server modules).

4.5 Customizing the Application Daemon

The *AppDaemon.Policy* file is read by the Application Daemon in the user's Playground settings directory (~/.pgdir/AppDaemon.Policy). If the file does not already exist, the Application Daemon will create it. The *AppDaemon.Policy* file may contain the web directory root where the application pages reside and administrative information. Additionally, comments beginning with "#" and extending to the end of the line are permitted throughout the file. An example policy file is provided below:

```
# Example AppDaemon Policy File

weblocation /home/fred/www/apps

fixedsocket 7654
fixedpid 555
```

The attributes specified in the policy file are explained below:

- **weblocation** - The user may specify the root of the application directory:

  ```
  weblocation <directory path>
  ```

- **fixedsocket** and **fixedpid** - The user may choose a socket value between 1024 and 65535 for the underlying Playground Veneer to use for communication. Similarly, the user may specify a process ID (pid) between 1 and 1024 that the Application Daemon should use. Specifying both a fixedsocket and a fixedpid provides a fixed location that can be shared. Example specifications are provided below:

  ```
  fixedsocket 8765
  fixedpid 555
  ```

Finally, the *AppDaemon.Policy* file may contain a rejection section as described in Section 3.2.

4.6 Starting the Application Daemon

To enable users to start an application, an Application Daemon must be running to serve the information in the application directory tree. The user may either specify the application page root directory as a command line argument when starting the Application Daemon:

UNIX
```
PGappDaemon -base /home/fred/www/apps &
```

NT
```
start PGappDaemon -base C:\home\fred\www\apps
```

or, alternatively, the root directory may be specified in your *AppDaemon.Policy* file (see Section 4.5) and the Application Daemon may be started without arguments.

The Application Daemon **must** have write access to the application directory.

Appendix A

Command-Line Arguments

Command-line arguments provide a means of either specifying or overriding the options described in the *.Policy files. This appendix describes the command-line arguments for the Broker, Application Daemon and Launcher.

A.1 Broker Command-Line Arguments

Argument	Description
-socket <N>	Sets the socket the Broker should listen on for requests from the Liaison. A value of 0 will disable the socket interface.
-fixedsocket <N>	Sets the out-of-band communication socket for the Playground interface.
-fixedpid <N>	Sets the alias process id.
-system	Sets fixedsocket and fixedpid to known default values (3 and 5003).
-master <PGID>	Permits the given PGID to kill the launched Broker. Normally, only a PGDAD or PGMOM [1] is allowed to kill a module.
-pgdir <path>	Specifies that <path> should be used instead of ~/.pgdir. This option should be used for a Broker running as nobody.
-nobodyuid <uid>	If run as root, this option runs the Broker as the provided “uid” (e.g., nobody).

Table 2: Broker Command-Line Arguments

A.2 Application Daemon Command-Line Arguments

Argument	Description
-base <dir>	Specifies the path of the root of the application directory.
-fixedsocket <N>	Sets the out-of-band communication socket to N for the Playground interface.
-fixedpid <N>	Sets the alias process id.
-system	Sets fixedsocket and fixedpid to known default values (3 and 5003).
-pgdir <path>	Specifies that <path> should be used instead of ~/.pgdir. This option should be used for an Application Daemon running as nobody.
-nobodyuid <uid>	If run as root, this option runs the Application Daemon as the provided "uid" (e.g., nobody).

Table 3: Application Daemon Command-Line Arguments

A.3 Launcher Command-Line Arguments

Argument	Description
-fixedsocket N	Sets the out-of-band communication socket to N for the Playground interface.
-fixedpid N	Sets the alias process id to N.
-system	Sets fixedsocket and fixedpid to known default values (3 and 5003).
-base <dir>	Sets the top-level directory where the Launcher recursively searches for modules and sublaunchers.
-bootstrap	Causes the Launcher to recursively bootstrap all sublaunchers in the directory tree (using rsh).
-fork	Causes the Launcher to fork() and exec() an new process to free the shell for other use.
-hide_modules	Prevents the launcher's ID from being propagated up the launch tree to other Launchers or Brokers.
-pgdir <path>	Specifies that <path> should be used instead of ~/.pgdir. This option should be used for a Launcher running as nobody.
-nobodyuid <uid>	If run as root, this option runs the Launcher as the provided "uid" (e.g., nobody).

Table 4: Launcher Command-Line Arguments

Appendix B

Launch Tree Information File Grammar

The grammar for the <module>.info file is provided for the user that wishes to either use advanced options in the <module>.info file that are not described elsewhere or wish to write their own configuration tool for the file format.

The grammar for the <*module*>.*info* file is as follows:

```
        <header>
        <mandatory>
        <optional>*

<header>      →     PG_MOD_SPEC <version> <revision>

<mandatory>   →     map <publicName> <executable>

<publicName>  →     Advertised name of module

<executable>  →     Absolute path to executable or just the executable name if stored locally.

<optional>    →     args <string to be sent to launched modules (command line options)> |
                    env ( setenv | set ) |
                    var <environment variable pairs> |
                    launch ( fork | system ) |
                    flags <launchCost> <runningCost> |
                    notpg <Not A Playground Module> |
                    parent <Module is bootstrapped to (I.E. a configurer)> |
                    directory <Module's connected directory on execution> |
                    private <Module is launcher PRIVATE> |
                    joinonly <Module is long lived, do not kill it> |
                    logto <filename of log data> |
                    host <host name of rsh launched module or sublauncher> |
                    launchwait <seconds to wait>
```

Appendix C

Application Specification File Grammar

The grammar for the new.spec and join.spec files is provided for the user that wishes to either use advanced options that cannot be specified through the Mediator or wishes to write his or her own configuration tool for creating new.spec and join.spec files.

C.1 Grammar

The grammar for the *new.spec* file is as follows:

```
        PGMEDIATOR <version>
        <section>*
        END_DOCUMENT

<section>     →     SECTION <secname> <version>
                    <secbody>
                    END_SECTION

<secname>     →     APPSPACE | MODULE | CONNECTIONS | APPDAEMON

<secbody>     →     <appbody>* | <modbody>* | <connectbody>* | <adbody>*

<appbody>     →     name <cstr> |
                    dimensions <l> <t> <w> <h> |
                    zoom <factor> <px> <py> |
                    module <intID> <x> <y>

<modbody>     →     id <intID> |
                    name <counted-string> |
                    publicName <string> |
                    location <location> |
                    optimize <optimize> |
                    launch (true | false) |
                    env <variable> <value> |
                    performance <performance> |
                    pgid <PGID> |
                    retry <retry> |
                    flags <flags> |
                    args <command-line arguments>
```

```
<intID>        →    a unique non-negative integer (an index)

<location>     →    local | preferred |
                    launcher <launcherName> | host <HostName>

<retry>        →    abort | rotate | broker | softfail

<flags>        →    joinonly | notpg | parent

<optimize>     →    flops | vmips | mcpys | load

<performance>  →    flops <float> | vmips <float> | mcpys <float>

<args>         →    string to be sent to launched modules  (command line args)

<env-pair>     →    <varname> <var-value>

<varname>      →    string of characters without whitespace - var name

<var-value>    →    string of characters without whitespace - var translation

<string>       →    string of characters without whitespace

<launcherName> →    <PGID> - fixed string of Launcher ID

<PGID>         →    string of encoded playground identifier

<cstr>         →    <count> string of characters - may have whitespace

<adbody>       →    demo <minutes>

<connectbody>  →    connect <conID> <fromID> <cstr> <toID> <cstr> <properties>

<float>        →    floating point number (decimal)

<minutes>      →    integer, 0 < X < 60

<properties>   →    <bit-vector>

<bit-vector>   →    1  : unidirectional
                    2  : bidirectional
                    4  : elt-to-aggr
                    8  : elt-from-aggr
                    16 : send on connect
```

C.2 Option Descriptions

- location - The <location> specifier allows the designer to specify WHERE a module should be launched from. A "local" launch goes to a Launcher on the same machine as the user's Broker. A "preferred" launch heads to a work group Launcher (if any). Specifying "launcher X" or "host X" sends the modules directly to the named Launcher. Launcher names may be fixed in advance using the fixedpid and fixedsocket options in the launcher.
- flags - The <flags> specifier allows the designer to specify characteristics of the module. A "notpg" indicates that the module is NOT a Playground compatible image, and should not expect a veneer handshake. A "joinonly" indicates the module (if it already exists) is not to be killed should a launch fail. This type of module would be typified by a video server, or a conference management module. A "parent" indicates that the module is to be launched before ALL other non-parent modules. The resulting modules are then passed to the remaining modules as PGMOM modules. This allows a GUI or other controlling module to be started as part of a launch.
- optimize - The <optimize> specifier allows the designer to choose the processor characteristics of the host the launch system selects for execution. A keyword of "flops" specifies that the fastest floating point hardware available is to be used for launching this module, based on current loading. A keyword of "vmips" selects the fastest scalar hardware available is to be used for launching this module, based on current loading. A keyword of "mcpys" specifies the greatest memory bandwidth (or size) hardware available is to be used for launching this module, based on current loading. A keyword of "load" selects the least loaded machine available (beware, a 386 might have a load of zero!)
- performance - The <performance> specifier allows the designer to choose the processor characteristics of the host the launch system selects for execution. A keyword of "flops" specifies that at least that many flops should be currently available on the target machine before a launch is attempted. The target computes the available flops based on the BenchMarker output and its current load. Windows-NT machines have a fixed load of 5. Flops are double precision operations. A keyword "vmips" indicates a specified minimum scalar performance. A keyword of "mcpys" indicates a specific memory characteristic.
- args - The <args> specifier allows the designer to pass command line arguments to the launched module. These arguments are the last items on the launched module's command line. Locally specified arguments (i.e., those from the launching agent's machine) are appended first.
- pgid - The PGID of a module may be fixed in advance, say for a video server at a know site. A typical PGID is of the form <inet-addr>,<pid>:<commtype><int>. e.g., 128.252.137.1,1:SCK5001. The fixed address must be coded into the module at the other end.
- properties - The connection properties are specific to the type of data and dataflow directions needed for this connection. For a simple connection, the constant 17 is typical.
- demo - The "demo N" entry in the APPDAEMON section specifies to the AD that all modules (and this application) should be killed on the next update after N minutes have passed. This will kill applications with "joinonly" set.

References

[1] Kenneth J. Goldman, Joe Hoffert, T. Paul McCartney, Jerome Plun, Todd Rodgers. “The Playground Veneer Reference Manual.”

[2] Goldman, K.J., et. al. “Welcome to the Programmer’s Playground!” http://www.cs.wustl.edu/cs/playground/

[3] T. Paul McCartney, Kenneth J. Goldman. “EUPHORIA Reference Manual.” Washington UniversityDepartment of Computer Science WUCS-97-13, February 1997.

[4] T. Paul McCartney. “Mediator Reference Manual.” In preparation.