ptserver, Homer, and ptdroid
Installation Guide

PETER FOLDES

ANAR HUSEYNOV

JUSTIN KILLIAN

ISHWINDER SINGH

CARNEGIE MELLON UNIVERSITY

INSTITUTE OF SOFTWARE RESEARCH

Table of Contents

Introduction

The HandSimDroid system consists of three major modules
- ptserver - responsible for distributed client/server model execution
- Homer - responsible for creation of customized layout files specifically targeted for distributed execution
- ptdroid - Android application that acts as a client to the ptserver and runs the distributed model with a customized layout created by the Homer application

ptserver (ptserver package) and Homer (ptolemy.homer package) modules are part of the ptII tree but ptdroid is separate project whose SVN repository is located here: source.eecs.berkeley.edu/home/svn/chess/ptdroid

Dependencies

Both ptdroid and Homer applications somewhat depend on ptserver. Homer needs it for generation of the layout file supporting distributed model execution. ptdroid needs it for setting up infrastructure for communication with the server. Therefore, the ptserver must be build first in the build process.

MQTT

ptserver and ptdroid communicate over MQTT protocol which is a publish subscribe kind of protocol. MQTT broker is required for passing messages between its clients which in our case are ptdroid and ptserver components. Several implementations of the broker are available but for our project we chose Mosquitto, an open source implementation of the broker. Another option is to use MQTT broker provided by IBM but it's closed sourced and commercial. Neither ptdroid nor ptserver depend on particular implementation of the broker; therefore it could be easily swapped out for another implementation. The broker runs in a separate process and either need to be launched prior to launching ptserver or ptserver can be configured to launch it when it starts.

Mosquitto

Mosquitto is an open source cross platform implementation of the MQTT broker. Binaries are available for Windows platform but it needs to be compiled from source for Mac and some Linux distributions.

The sources and binaries can be downloaded from here: http://mosquitto.org/download/

Compilation for Mac

1. Install cmake from http://www.cmake.org/cmake/resources/software.html or from http://mxcl.github.com/homebrew
2. Download and untar the mosquitto sources:

- ○ cd $PTII/vendors
- ○ wget http://mosquitto.org/files/source/mosquitto-0.10.2.tar.gz
  - i.Note: substitute the URL to the latest release version available.
- ○ tar -zxf mosquitto-0.10.2.tar.gz
- ○ cd mosquitto-0.10.2
3. Run:
   - ○ cmake .
   - ○ make
   - ○ sudo make install

Running Mosquitto

Mosquitto could be launched simply by executing its binary. By default it runs on port 1883 but both mosquitto and ptserver can be configured to run on a different port.   For details see the next section.

Building & configuring ptserver

Ptserver depends on the following jars:
- ● ptserver/lib/hessian-4.0.7.jar
  - ○ The Hessian library enabled binary synchronous communication between the server and clients over HTTP protocol.
- ● ptserver/lib/jetty-all-7.4.1.v20110513.jar
  - ○ ptserver runs embedded Jetty which is a servlet container. The servlet is needed for hessian communication protocol.
- ● ptserver/lib/servlet-api-2.5.jar
  - ○ Another library needed for the embedded Jetty server
- ● ptserver/lib/wmqtt.jar
  - ○ MQTT client library provided by IBM

In order to run the server, the following configurations need to be made:
1. Create ptserver/PtolemyServerConfig.properties file. Use PtolemyServerConfig.properties.default as a template.
2. The file has the following fields:

> #path to broker executable (if running locally)
> BROKER_PATH=
> #address of the broker
> BROKER_ADDRESS=localhost
> #port number of the broker
> BROKER_PORT=1883
> #port number to run servlet on
> SERVLET_PORT=8081
> #name to save server logs
> LOG_FILENAME=PtolemyServer.log

#absolute path to the directory where the model files are stored
MODELS_DIRECTORY=

- ○ If **BROKER_PATH** is specified which is file path to the broker executable the server would automatically try launching the broker on start up. Note: currently the server does not shut down it when the server is killed.

- ○ Alternatively **BROKER_PORT** can be specified in cases when the broker runs on a separate machine or if it does not need to be launched automatically.

- ○ **BROKER_PORT** specifies the port the broker run on. Mosquitto can run on a different port via -p command line option or by changing its configuration file.

- ○ **SERVLET_PORT** is the port the Jetty, embedded servlet container would use. This port must be specified in the ptdroid when adding new server to it.

- ○ **LOG_FILENAME** - the file name of the log file. The file would be created relative to the execution directory of the ptserver

- ○ **MODELS_DIRECTORY** - directory containing both model and layout files. For details see Homer user documentation. The directory must be specified in order to remotely load models from the ptdroid.

To run the server, launch ptserver/PtolemyServer.class file.

ptserver supports basic authentication via Hessian protocol. In order to add or remove users, modify ptserver/PtolemyServerUsers.properties file. The PtolemyServerUsers.properties.default is provided as a template.

Building & configuring Homer

Homer depends on the following libraries:
- ● lib/org-netbeans-api-visual.jar
- ● lib/org-openide-util-lookup.jar
- ● lib/org-openide-util.jar

All three jars part of Netbeans Visual library which is used within Homer extensively to enable WYSIWYG kind of layout editor.

In order launch Homer, execute ptolemy/homer/HomerApplication.class file.

Homer supports pluggable design for adding support of new widgets for different actors or attributes. There are 3 different ways of visualizing a given actor/attribute with a widget:

1. By creating a new custom widget that extends NamedObjectWidget and adding a mapping from the actor class to the widget class to ptolemy/homer/widgets/ObjectWidgets.properties file.
2. By adding an image to ptolemy/homer/images folder and adding mapping from the named object to the image file name to ptolemy/homer/images/ImageWidgets.properties file.
3. If above two methods fail, Homer would use named object's icon as a widget.

Homer could also fallback to the mapping of a named object's super classes if it's available.

If an actor implements PortablePlaceable interface, the easiest way to visualize it is to map the actor to ptolemy.homer.widgets.PortablePlaceableWidget which would visualize it as it is seen when it's running within Vergil.

Building & configuring ptdroid

ptdroid project depends on the following jars:
- lib/hessdroid.jar - the Hessian library compiled specifically for Android
- wmqtt.jar - IBM MQTT client library

Since Android Eclipse plugin has bug, Android projects can't reference projects that have project structure used by ptII - same folder for source and class files. As a result, we had to write a custom build script that compiles required classes from ptII and puts them into ptbin folder within ptdroid. ptbin folder is configured to be a class folder.
In order to automatically run the build script, a custom builder is included along with the project called Ptolemy Builder. One could find it from the properties of the project in the Builders pane. The builder must be configured to point to the build file - build.xml file - within the project. Currently the trunk version is pointing to ${workspace_loc:/ptdroid/build.xml} but this might need to be adjusted if the project is named differently (i.e. if a branch is checked out). Also, an argument to the build file must be supplied in the from of -DptIIsrcdir=path to the ptII source directory. Currently it points to ../ptII.

The builder must be configured to perform ptbinclean for all cases but auto build when ptbinbuild needs to be performed. Also sometimes when files within ptII change, a clean build need to be made to ensure that ptdroid has a latest version. It's unclear why ant won't detect file changes and recompile them.

Continuous Integration

Both projects contain build scripts to automatically build them, run findbugs and checkstyle checks. ptserver project also runs all JUnit tests along and collects test coverage metrics using cobertura. ptdroid does not include many unit test cases because it's largely UI driven

application and because it's hard to integrate the test cases with continuous integration since Android does not produce junit test results report reliably.

Adding new library

When a new library is added to either project classpath must be updated with the build file. In ptserver it's ptII.classpath. In ptdtoid, it's "jar.libs.ref" path id.
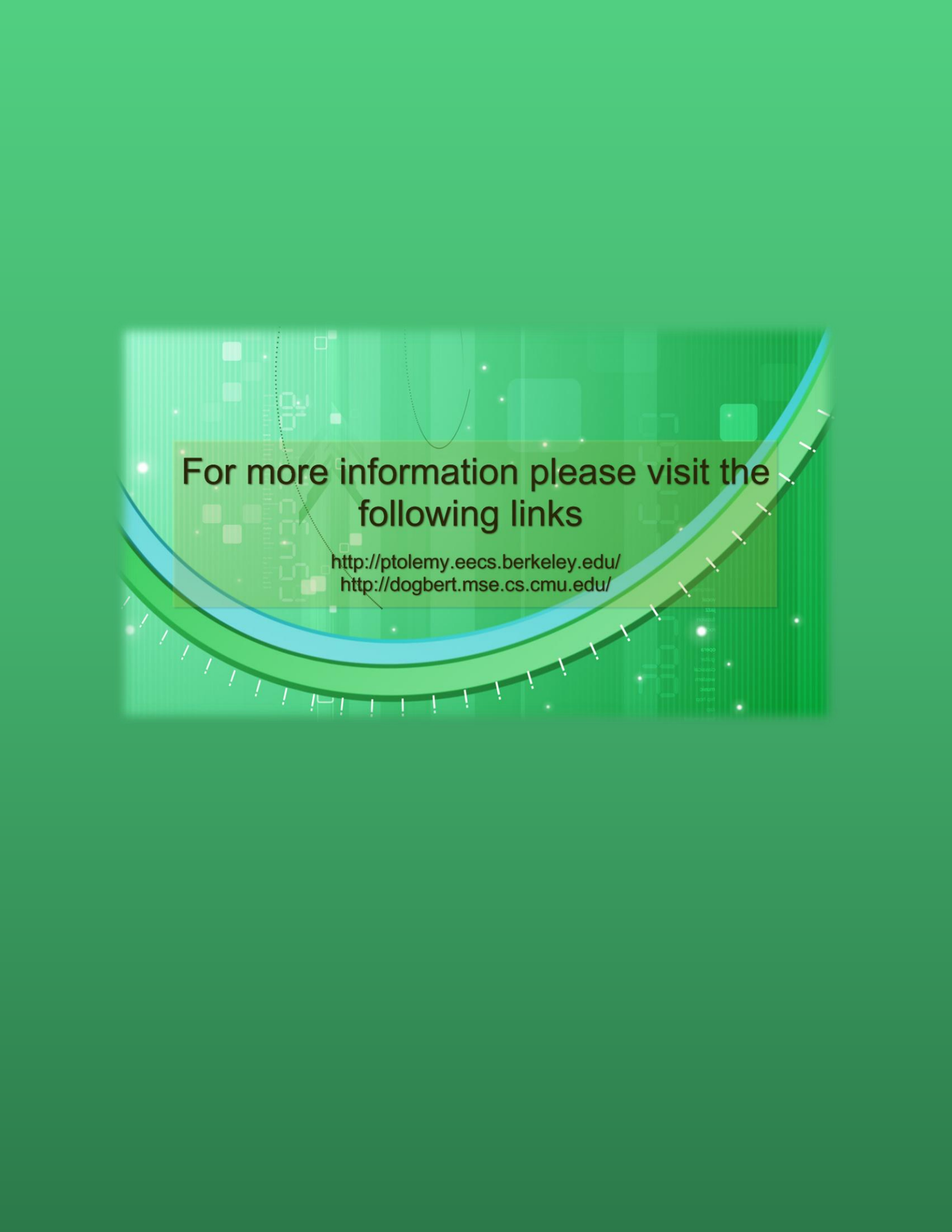
Changing classpath

Both projects are configured to exclude certain packages from the classpath. It's done mostly for performance reasons because conversion from .class files to dex files are slow. Also the large class path would likely impact load time of the ptdroid application.

Feeding output to Continuous Integration

ptserver/homer build file produces the following output files:
- JUnit test result - tempoutput folder
- Cobertura test coverage - tempoutput/coveragereport folder
- Checkstyle report - tempoutput/checkstyle_errors.xml
- Findbugs - tempoutput/ptserver-fb.xml

All tools are configured to only look at a portion of the codebase. See the build file for details.

For more information please visit the following links

http://ptolemy.eecs.berkeley.edu/
http://dogbert.mse.cs.cmu.edu/