JMaCS: a Java Monitoring and Control System

Rob Dickens

// Latterfrosken
software.development(limited);

Presented as poster at SPIE conference on Astronomical Instrumentation, in Marseille, France, on June 27, 2008

Introduction

JMaCS is **derived** from experimental software[1][2] originally written to permit the monitoring and control of a radar used for observing the Earth's ionosphere[3].

Of particular interest when considering that software's **requirements** were the radar's (then novel) distributed design[4], together with its remote and sometimes inaccessible location. The scope of those requirements were subsequently to grow from being user interface-related only, to system-wide.

Although JMaCS is no longer radar-specific, its design goals are otherwise essentially the same: to facilitate the local or remote interactive and programmatic monitoring and control of a distributed target. This should be achieved in soft[5] realtime, which is to say that commands and status should be delivered in a timely manner, but with the built-in assumption that delays will sometimes occur. Where necessary, therefore, external underlying hard-realtime subsystems will be assumed.

The **Java**[6] platform was chosen for its many features aimed at facilitating the development of distributed software. Of particular interest initially were those features which simplified the development of cross-platform GUI apps. Later, it was to be the ability to send and receive executable objects, using efficient native Java serialisation, and employ polymorphism in a distributed setting.

Client-server design

user interface (UI) clients → target server ← device interface (DI) clients

This was considered to be a suitable reflection of the distributed and possibly remote nature of the target, and also to have two advantages over a design involving DI servers:

- access to devices is more secure, since they may only be accessed via the target server chosen by the DI deployer;
- it is more easily extended, by the simple introduction of additional DI clients.

It was decided that the convenience of high-level **interactions**, using Java Remote Method Invocation (RMI)[7][8][9], could be afforded for delivering commands, and that the efficiency of low-level ones, using sockets, was required for the dissemination of status. Furthermore, in order to permit efficient dissemination on a local network, it was decided that the size of all device status samples should be constrained by the payload of a UDP datagram (that could then be multicast to all recipients in one go).

Each DI client requires a **'plug-in'**, consisting of classes implementing particular Java interfaces: a driver (for executing commands and generating status) together with optional monitor and controls GUIs.

Each DI client also requires a **domain.like.name**, thereby defining its position in a **logical hierarchy** representing the target as a whole.

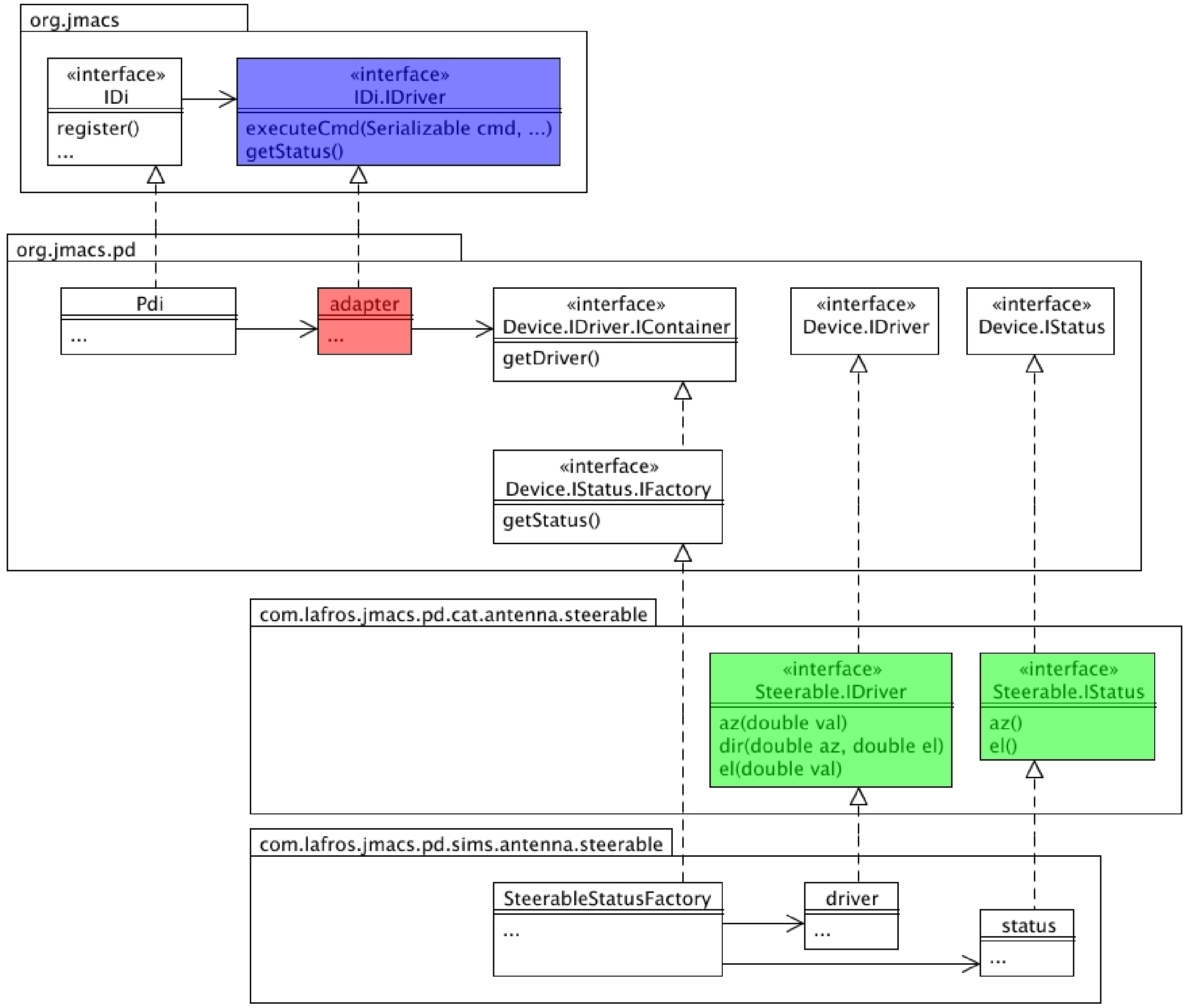
'Programmable Devices'

Rather than have to write a new DI plug-in every time, it is useful to write '**adapter**'[10] ones, which fill-in some of the 'blanks', and often introduce new ones of their own. This technique may be used to write an adapter which **accepts a definition of an actual named device** (e.g. steerable antenna), having its own API.

Furthermore, this adapter can be endowed with the means to receive and run programs, exposing the device's API to those programs.

The definition accepted by such an adapter is therefore referred to as a 'programmable device' (PD).

New PDs may be written in terms of existing ones using the object-oriented techniques of composition and inheritance[11], and catalogues then accumulated.


org.jmacs
«interface» IDi
register()
...
«interface» IDi.IDriver
executeCmd(Serializable cmd, ...)
getStatus()
org.jmacs.pd
Pdi
...
adapter
...
«interface» Device.IDriver.IContainer
getDriver()
«interface» Device.IDriver
«interface» Device.IStatus
«interface» Device.IStatus.IFactory
getStatus()
com.lafros.jmacs.pd.cat.antenna.steerable
«interface» Steerable.IDriver
az(double val)
dir(double az, double el)
el(double val)
«interface» Steerable.IStatus
az()
el()
com.lafros.jmacs.pd.sims.antenna.steerable
SteerableStatusFactory
...
driver
...
status
...

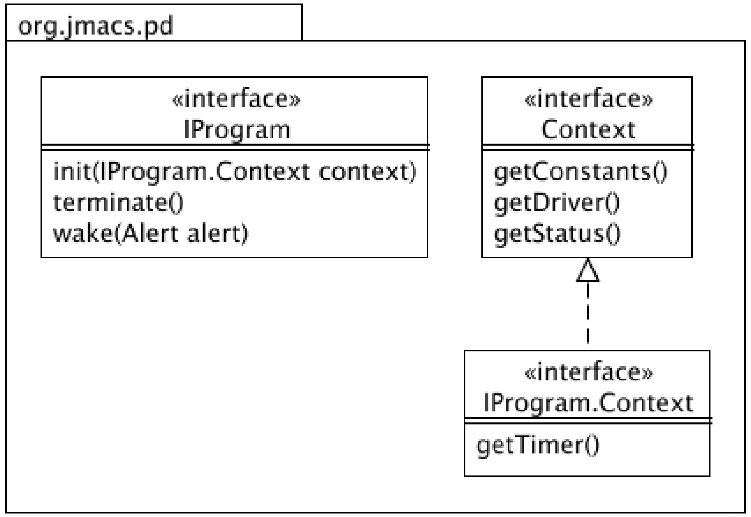
Implementation

Thanks to the **org.jmacs** Java package, DI plug-ins are developed independently of the actual JMaCS implementation, which is supplied by a third party. Thus, plug-ins implement `org.jmacs.IDi.IDriver`, while the JMaCS implementation implements `org.jmacs.IDi.AbstractFactory`. DI deployment involves first instantiating the latter via its static getInstance() method, where the implementation class must be in the Java class-path, and is specified using a Java system-property.

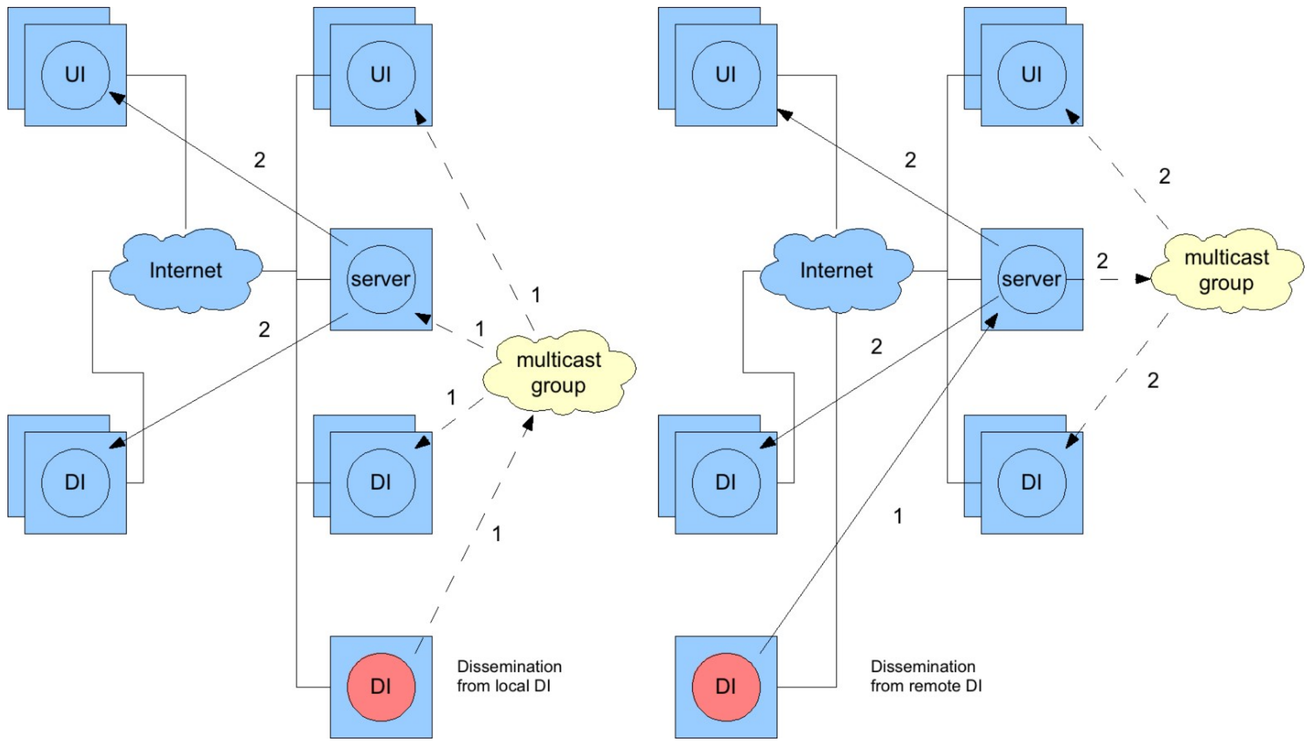
JMaCS implementation packages	
org.jmacs	
DI plug-ins	**org.jmacs.pd**
	Fully-implemented PDs

The **org.jmacs.pd** package provides the DI plug-in adapter described earlier. All PD definitions must include a Java interface which extends `org.jmacs.pd.Device.IDriver`. An implementation of this (which might or might not be part of the definition) is then instantiated and made available locally to any command-`org.jmacs.pd.IInterpreter` provided, and to any `org.jmacs.pd.IProgram`s run, and also remotely via a dynamic proxy.

The wake() method implemented by PD programs (right) is called periodically, as controlled by the timer.



Negotiating the Internet



Initially, device status samples were disseminated using connectionless transport[12] exclusively. However, it was later discovered that there was effectively a limit of only a few kilobytes to the size of UDP datagrams which could be sent over the internet. This necessitated making significant **changes to the JMaCS implementation**, to enable connection-oriented transport[12] to be used for sending samples over the internet. Their dissemination from local and remote DIs is depicted above.

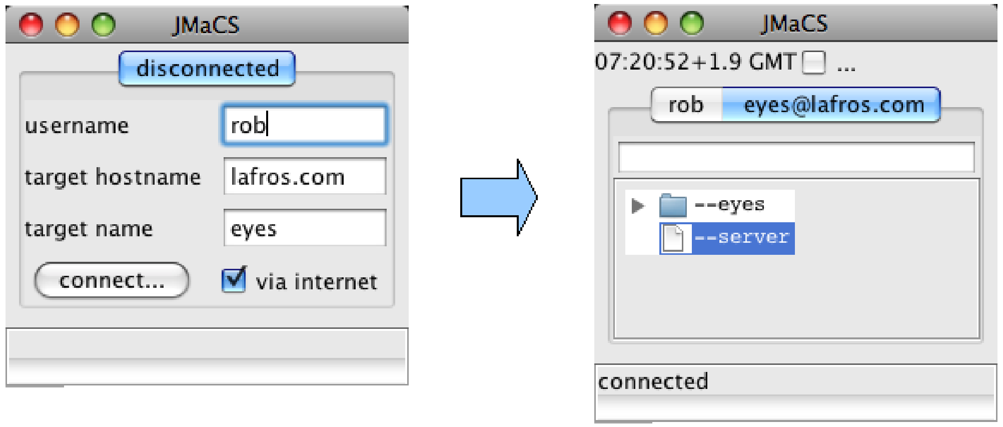
A severe bottleneck was subsequently encountered when sending status samples to these remote subscribers. This was addressed by utilising a separate thread (in the target server) for each DI being subscribed to, together with a separate connection for each remote subscription, thus allowing different samples to be sent concurrently.

Changes were also required in order to accommodate port-number indirection security-measures.

User interface

The AWT-based GUI of the original experimental software has evolved into a **Swing-based** one, utilising a set of JUICe (http://lafros.com/juice) libraries which have been developed in parallel with JMaCS. Among these libraries is the JUICe.app application framework, which allows the UI client to be run either as a stand-alone application (using **Java Web Start**) or as an **applet** (embedded in the browser window). The following features are implemented:

- compensation for any clock lag or drift (relative to the server host's clock);
- synchronized refreshing of monitor windows (after a set delay), where the titles of any which could not be refreshed are highlighted;



username: rob
target hostname: lafros.com
target name: eyes
[✓] via internet

07:20:52+1.9 GMT ☐ ...
rob | eyes@lafros.com
--eyes
--server
connected

- intrusive alerts (issued by the UI Client itself, together with any written using the JUICe.alerts library);
- 'hot deployment' of downloaded classes where appropriate;
- interactive configuration of PD program properties.

Connecting to the target (above right) creates a viewer pane, containing a DI navigator and scrollable area in which to display the monitor windows and control panels (optionally) supplied by each DI, together with a command line (whose commands will be sent to the selected DI).

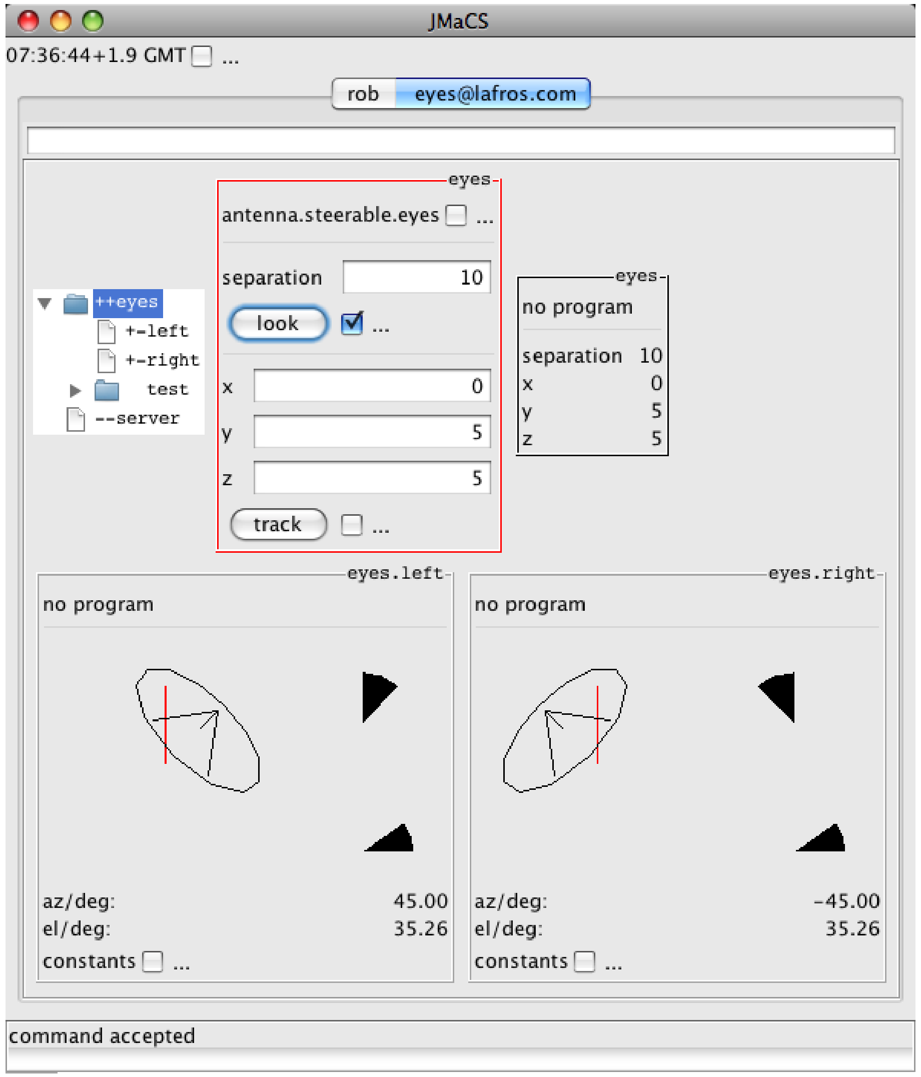
The demonstration target

This is an easily understood 'installation' that demonstrates, amongst other things, how one DI may control others, in this case simply the root controlling its two children. Thus, we have **a PD representing an eyes 'device', together with two steerable antenna ones named 'eyes.left' and 'eyes.right'**.

The user first submits a command to the root, via its control panel (or the UI's built-in command line). The root then sends corresponding commands to its children. Two 'eyes' commands are available—one to 'look' at a particular point in space (right), and another to track a projectile from a specified launch site to a specified target one.

N.B. None of these DIs interfaces to any hardware device. The steerable antenna ones are instances of a simulator implementation of an abstract PD defined for use with the original radar.

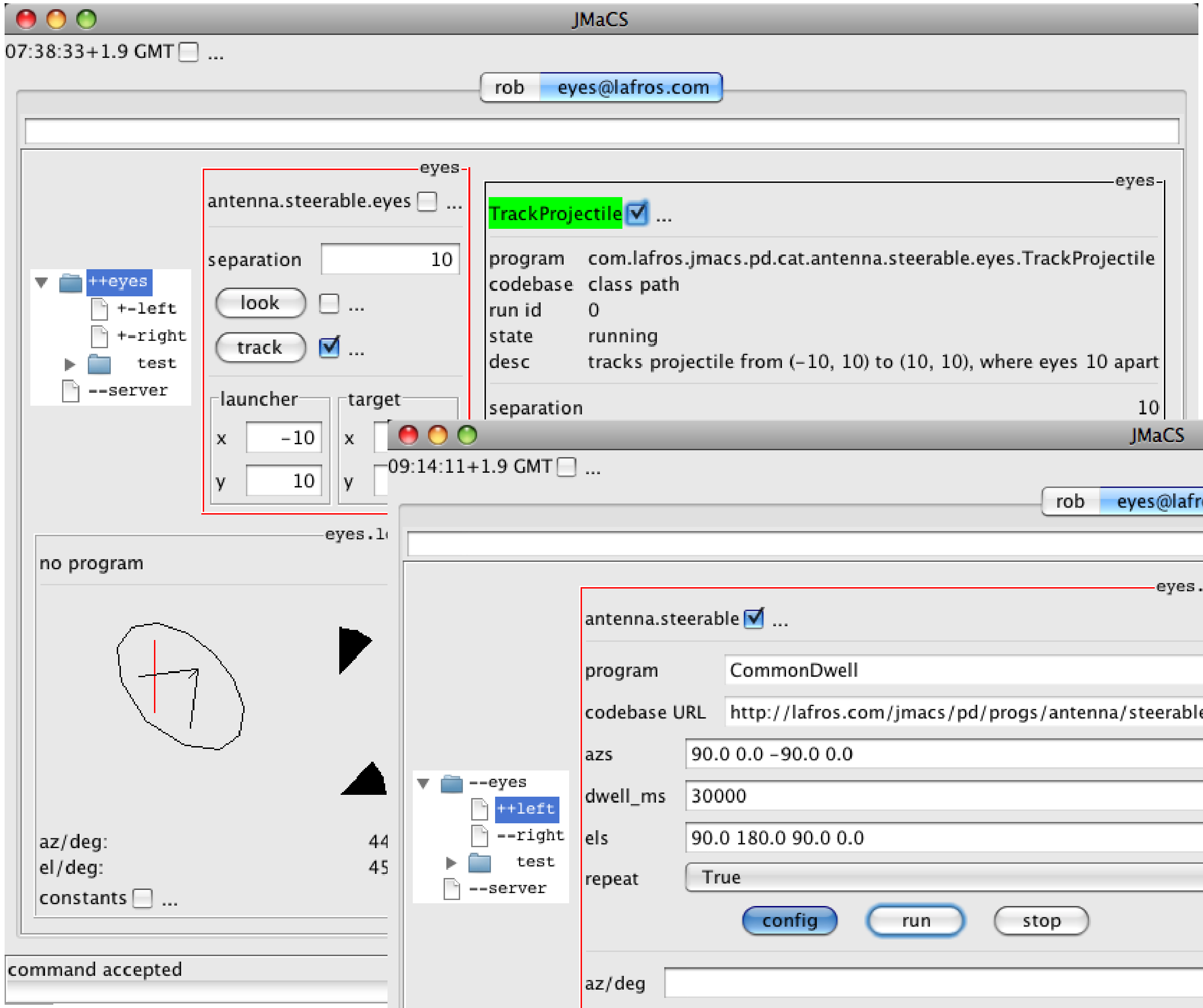
N.B. The target is not strictly distributed, since all the DIs are currently running on the server host.



```
07:36:44+1.9 GMT ☐ ...

eyes
antenna.steerable.eyes ☐ ...
separation   10
look ☑ ...
x   0
y   5
z   5
track ☐ ...

eyes
no program
separation  10
x   0
y   5
z   5

eyes.left
no program
az/deg:   45.00
el/deg:   35.26
constants ☐ ...

eyes.right
no program
az/deg:   -45.00
el/deg:   35.26
constants ☐ ...

command accepted
```

Demonstrating programmable control

Programs may be instantiated and configured either programmatically (left) or interactively (below).



JMaCS

07:38:33+1.9 GMT ☐ ...

rob | eyes@lafros.com

- ++eyes
  - +-left
  - +-right
  - test
- --server

eyes

antenna.steerable.eyes ☐ ...

separation 10

look ☐ ...

track ☑ ...

launcher: x -10, y 10

target: x, y

eyes

TrackProjectile ☑ ...

program	com.lafros.jmacs.pd.cat.antenna.steerable.eyes.TrackProjectile
codebase	class path
run id	0
state	running
desc	tracks projectile from (-10, 10) to (10, 10), where eyes 10 apart

separation 10

eyes.l

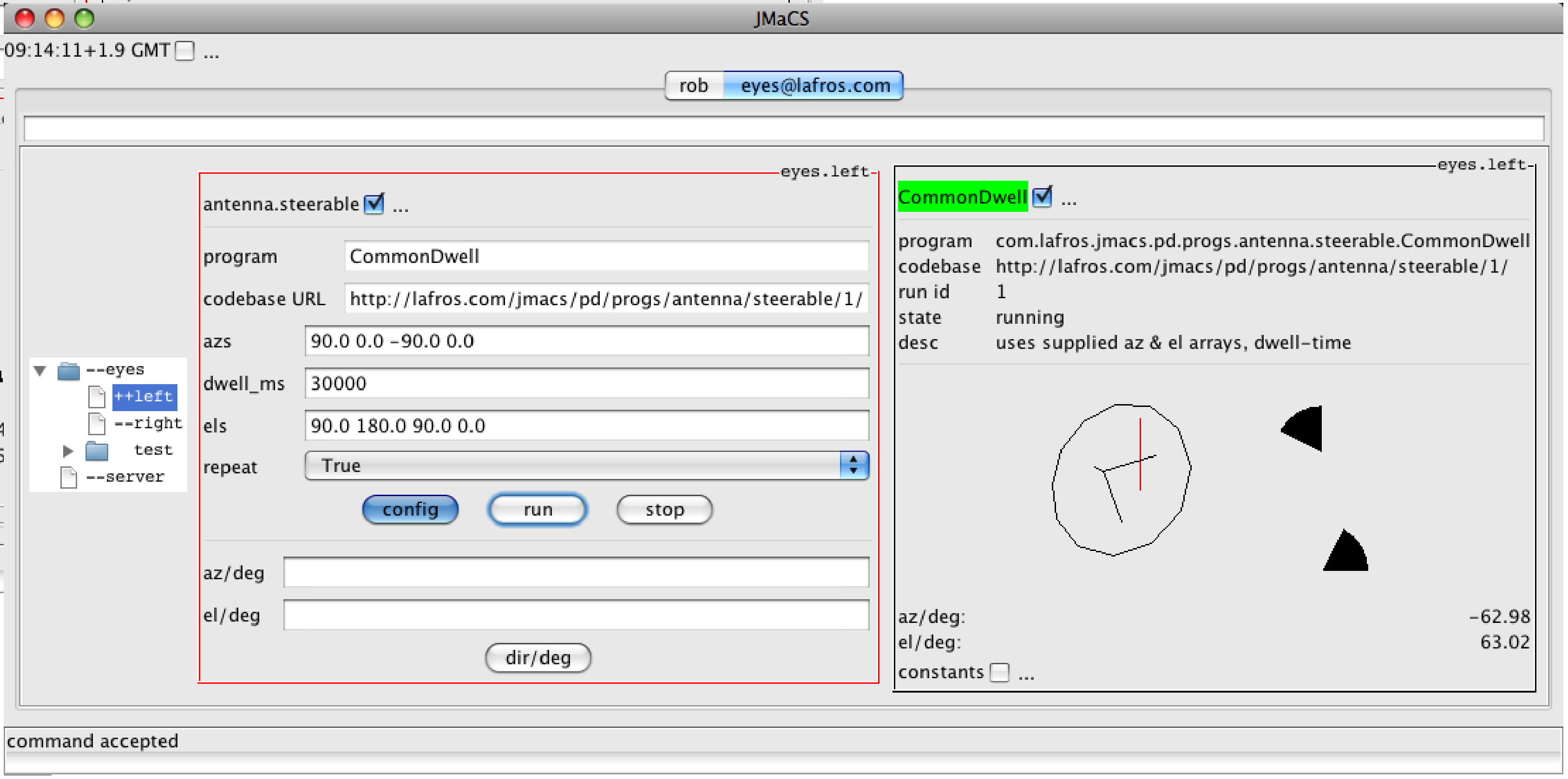
no program

az/deg: 44

el/deg: 45

constants ☐ ...

command accepted



JMaCS

09:14:11+1.9 GMT ☐ ...

rob | eyes@lafros.com

- --eyes
  - ++left
  - --right
  - test
- --server

eyes.left

antenna.steerable ☑ ...

program	CommonDwell
codebase URL	http://lafros.com/jmacs/pd/progs/antenna/steerable/1/
azs	90.0 0.0 -90.0 0.0
dwell_ms	30000
els	90.0 180.0 90.0 0.0
repeat	True

config | run | stop

az/deg

el/deg

dir/deg

eyes.left

CommonDwell ☑ ...

program	com.lafros.jmacs.pd.progs.antenna.steerable.CommonDwell
codebase	http://lafros.com/jmacs/pd/progs/antenna/steerable/1/
run id	1
state	running
desc	uses supplied az & el arrays, dwell-time

az/deg: -62.98

el/deg: 63.02

constants ☐ ...

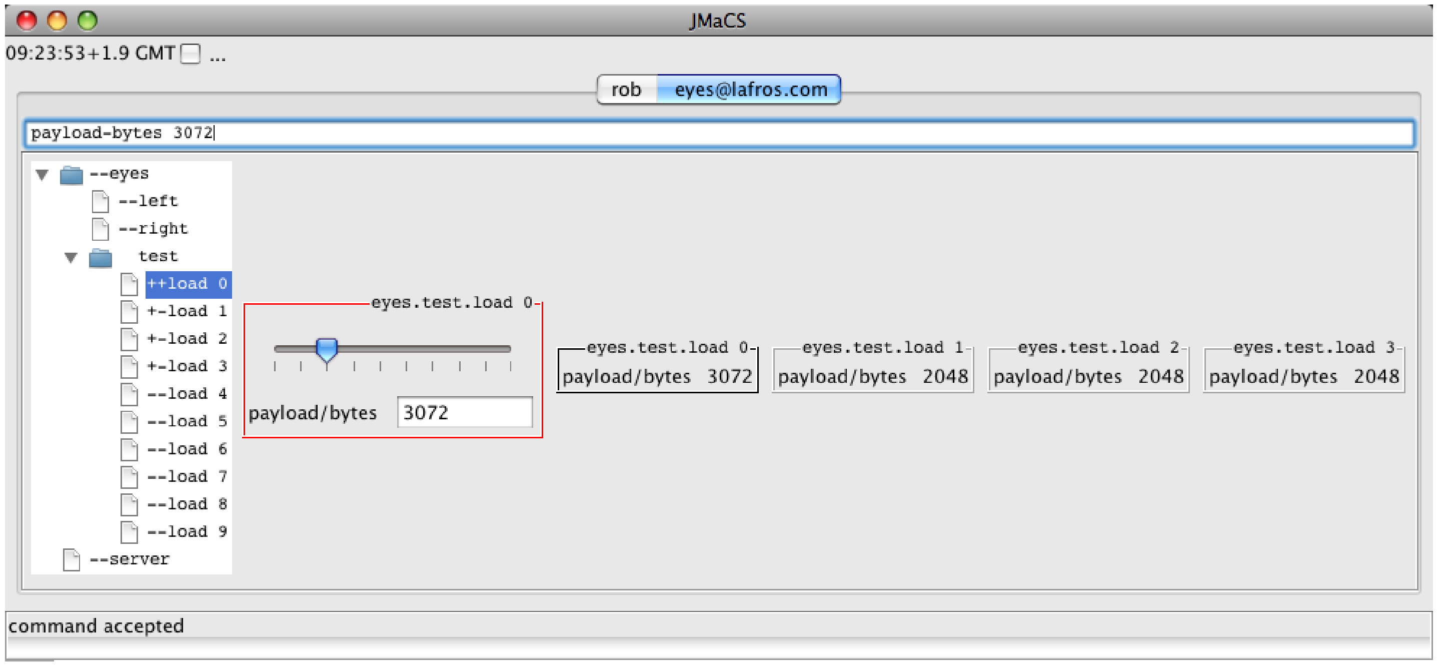
command accepted

Load testing

The demonstration target has **an additional ten DIs named 'eyes.test.load 0-9'**. These simply allow the size of their status samples to be adjusted. Various numbers of users, both local and remote, were logged in, various numbers of load-testing DI monitor windows were opened, and various sample sizes and sampling periods were configured.

The amount of data being sent to remote subscribers is still found to be the bottleneck, though less severe, and now thought to be limited by the external factor of Internet upload bandwidth: only about 32KiB/s, corresponding to e.g. four remote users, each having four monitor windows open, where each monitor is sent a 2KiB device status sample (on each sampling boundary).

This configuration was easily achieved using sampling periods down to 2s, even with all four remote UI clients running on the same host.



```
09:23:53+1.9 GMT ...
payload-bytes 3072
--eyes
  --left
  --right
  test
    ++load 0
    +-load 1
    +-load 2
    +-load 3
    --load 4
    --load 5
    --load 6
    --load 7
    --load 8
    --load 9
--server

eyes.test.load 0
payload/bytes 3072

eyes.test.load 0  payload/bytes 3072
eyes.test.load 1  payload/bytes 2048
eyes.test.load 2  payload/bytes 2048
eyes.test.load 3  payload/bytes 2048

command accepted
```

Conclusions

- The JMaCS monitoring and control software has been presented.
- Testing using the demonstration target has shown that it works well within the confines of the test.
- More extensive testing is therefore called for, where,
  - the server host has greater Internet upload bandwidth;
  - DIs are run on one or more separate hosts, preferably including remote ones.
- It is therefore tentatively claimed that,
  - monitoring and control using JMaCS is now viable;
  - the benefits of employing Java's remote polymorphism have been demonstrated.
- The experience of developing this software suggests that,
  - only very clearly defined Java projects should be attempted on a casual basis—the many facilities Java puts within reach still require a depth of understanding in order to be used effectively;
  - incorporating native code (when interfacing to hardware) can be challenging. However, this requirement is anticipated to have been reduced by newer versions of the Java platform, as well as by the advent of a realtime edition.

References

[1] Rob Dickens, Secure remote monitoring-and-control for the EISCAT Svalbard Radar: a case study in Java object-oriented design, 9th International EISCAT Workshop talk (Aug 1999).

[2] Rob Dickens, Monitoring and control of the 'radar.eiscat.esr' device, 10th International EISCAT Workshop poster (Jul 2001).

[3] Röttger, J., U.G. Wannberg and A.P. van Eyken, The EISCAT Scientific Association and the EISCAT Svalbard Radar Project, J. Geomag. Geoelectr., 47, 669-679 (1995).

[4] Bjørnå, N., B. Hultqvist, W. Kofman, J. Roettger, K. Schlegel, T. Turunen and D.M. Willis, The EISCAT Polar Cap Radar: Report on the design specification for an incoherent scatter radar facility based on the archipelago of Svalbard, prepared by the Polar Cap Radar Working Group established by the EISCAT Council, with financial support from the Rutherford Appleton Lab, UK (Nov 1990).

[5] Peter C. Dibble, Real-Time Java Platform Programming, Sun Microsystems Press, pages 7-9 (2002).

[6] James Gosling and Henry McGilton, The Java Language Environment, http://java.sun.com/docs/white/langenv (May, 1996).

[7] Daniel J. Berg and J. Steven Fritzinger, Advanced Techniques for Java Developers, Wiley, Chapter 7 Mastering Java Remote Method Invocation Techniques (1997).

[8] Rickard Öberg, Mastering RMI, Wiley (2001).

[9] Esmond Pitt and Kathleen McNiff, java.rmi, Addison Wesley (2001).

[10] For example, Steven John Metsker and William C. Wake, Design Patterns in Java, Addison Wesley, Chapter 3 (2006).

[11] For example, Peter Coad and Mark Mayfield, Java Design, Yourdon Press (1997).

[12] Douglas E. Comer and David L. Stevens, Internetworking with TCP/IP Volume III - Client-Server Programming and Applications, Prentice Hall, §2.3.5 (1994).