JGrass, a multi-platform, multi-session GRASS - the framework and features

John Preston*, Andrea Antonello**, Riccardo Rigon**

* International Centre for Environmental and Nuclear Sciences (ICENS), University of the West Indies, Mona Campus, Kingston, Jamaica, tel: 8769271777, fax: 8769770768, email: john.preston@uwimona.edu.jm

** Department of Civil and Environmental Engineering / CUDAM, University of Trento, Italy, tel: 390461882610 fax: 390461882672, email: antonell@ing.unitn.it, riccardo.rigon@ing.unitn.it

Introduction

Geographic Information Systems (GIS) have been used since the 1970's as tools for visualising and analysing spatial data. In recent times with the advent of powerful personal computers, the Internet, and Open Source Free Software [1], GIS applications have migrated from the domain of large proprietary mainframes to the desktop, expanding their use to a wide range of disciplines.

The Geographic Resource Analysis Support Software (GRASS) [2] is the largest GIS open source software project, and from its birth in 1982 at the United States Army Corps of Engineers Laboratories (CERL) in Illinois, it has found its way into many Universities, and Research Laboratories. It has a wide array of raster, vector, point, image processing, graphics production capabilities, and is extensible through source code availability.

However, one of the main obstacles to its widespread adoption by GIS professionals in production environments is its lack of a modern graphical user interface (GUI) with all the attendant benefits, and major portability issues with the Windows and Macintosh computer systems. To address these issues a development effort has commenced that seeks to move GRASS into the main stream of GIS software used by researchers and professionals across all the major computer operating systems, freely available and extensible. JGrass as it is called is a Java™ based GIS that:

1. allows execution of GRASS on the Windows, Unix/Linux and Mac-OS platforms with a single code base;
2. provides an architecture that is extensible, with a framework for the incorporation of user developed modules, and providing specialised functions ranging from visualisation to interaction with the map data;
3. incorporates the wealth of GRASS code currently available.

It has 2D/3D display capability, a terrain engine for view-dependent visualization of large surfaces, and an interactive legend. It also provides network access to GRASS databases and relational database management systems, a print editor, and other tools commonly available in state of the art GIS applications.

While Java™ is not an Open Source operating environment, it does offer substantial benefits, and there are a number of Open Source implementations such as Kaffe [3] and Jalapeño [4] (now Jikes RVM) that are freely available, and Sun Microsystems is talking about open sourcing Java at sometime in the future.

Architecture

JGrass is not another GIS but a Java based implementation of GRASS, with some improvements, and the architecture was developed with this in mind. Java™ was chosen as the operating platform because of its modularity, portability, extensibility and reliability, that facilitates the development of a single code base for all operating platforms. It comes with a large offering of libraries (known as packages) that provide ready built services such as GUI components, distributed data objects, and image processing libraries, many of which are open source. While the core Java libraries offer almost all the services required, the graphics engine for JGrass is implemented using the Java OpenGL Bindings (jogl) [5] to OpenGL™ which is a vendor neutral window system independent 3D rendering application programming interface (API).

The JGrass architecture (Figure 1) provides:

- a framework within which modules run;
- plugin modules that provide interactive and non-interactive services to users or other modules;
- communication channels that provide a mechanism for sending messages between modules, and the framework, and;
- a suite of x.xxxx commands that perform the various GIS functions.

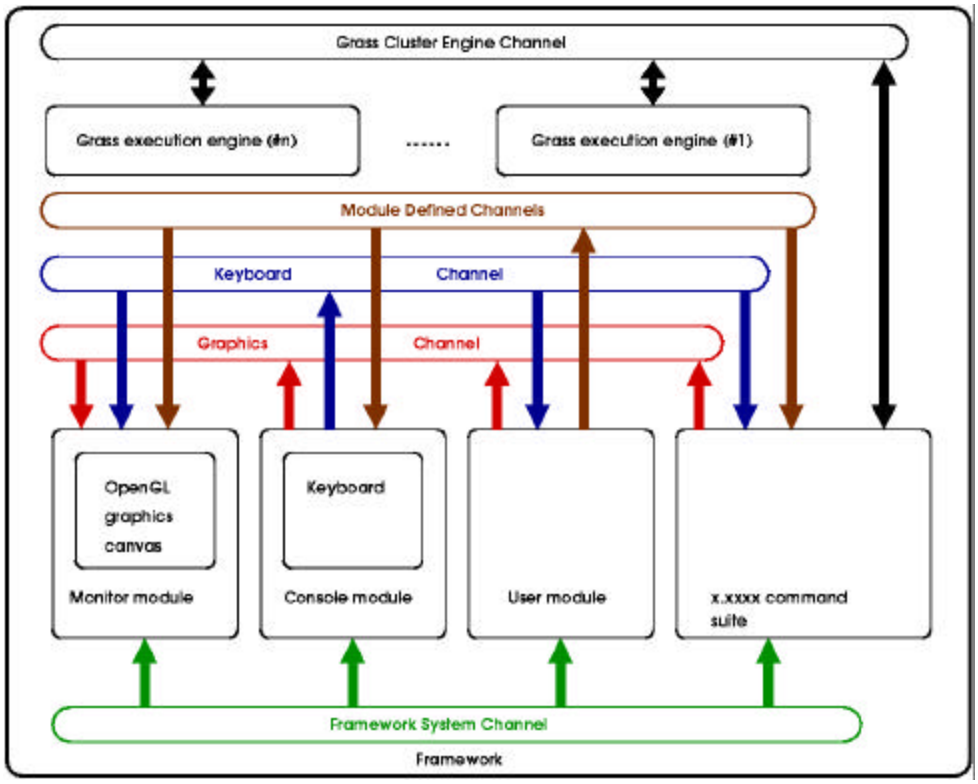

Figure 1, JGrass architecture

System configuration is handled with text files using a XML format style to allow easy re-configuration. Configurable items include:

- framework details such as window, menus, and toolbar properties;
- plugin module properties;
- language files for user prompts, messages, and help text;
- icon theme files;
- application data sources, and menu details.

Framework

The framework is a Java™ Swing based GUI with menus and toolbars, and functions for housing modules. At start up it reads a configuration file (Figure 2) and loads the required

```
<framework>
 <resourcebundle locale="" baseprefix="jgrass" extprefix="jgrassext" theme="" />
 <authloginconfig value="login.config"/>
 <templates value="etc/templates"/>
 <workspace value="etc/workspace" />
 <grassengine type="local" config="commands.xml" nativepath="/usr/local/grass53/bin" />
    ....
</framework>
<menubar fontname="georgia.ttf" fontsize="12">
   <menuroot name="frame.menu.file.label">
     <menuitem name="frame.menu.loadapplication.label" image="jgrass.logo"
       tip="frame.menu.loadapplication.tip" action="MainFrame|LoadGrassEnvironment|"/>
     <menuitem name="separator"/>
     ....
     ....
     ....
     <menuitem name="frame.menu.exit.label" image="frame.menu.icon" tip="frame.menu.exit.tip"
       action="MainFrame|Exit|"/>
   </menuroot>
 </menubar>
 <toolbar>
   <button name="Load" icon="application.load.icon" tip="frame.menu.loadapplication.tip"
    action="MainFrame|LoadGrassEnvironment|"/>
   <button name="separator"/>
   <button name="SaveAs" icon="module.saveas.icon" tip="frame.menu.saveas.tip" action="|SaveAs|"/>
   <button name="Print" icon="module.print.icon" tip="frame.menu.print.tip" action="|Print|"/>
   <button name="separator"/>
   <button name="Help" icon="help.icon" tip="frame.menu.generalhelp.tip"
     action="MainFrame|HtmlViewer|"/>
 </toolbar>
<plugins>
 <module name="Monitor" label="Monitor" load="startup" class="jgrass.module.monitor.Monitor">
   <param name="fontname" value="LucidaSansRegular.ttf"/>
   <param name="fontsize" value="12"/>
     ....
     ....
   <legend_window name="bgcolor" value="#cccccc"/>
   <legend_window name="bgimage" value="monitor.legendbackgroundimage.icon"/>
   <legend_window name="fontname" value="LucidaSansRegular.ttf"/>
   <toolbaritem name="viewmode" action="Monitor|ActionHandler|ToggleView" icon="monitor.viewmode.icon"
icondown="*monitor.viewmode.icon" tip="3D viewing parameters"/>
   <toolbaritem name="legend" action="Monitor|ActionHandler|ToggleLegend" icon="monitor.legend.icon"
icondown="*monitor.legend.icon" tip="Toggle the map legend window."/>
 </module>
 <module name="Console" label="Console" load="startup" class="jgrass.module.console.Console">
   <param name="font" fontname="LucidaSansRegular.ttf" fontsize="12"/>
   <param name="icon" value="console.png"/>
   <param name="bgcolor" value="#EDD093"/>
   <param name="fgcolor" value="#000000"/>
   <param name="prompt" value="GRASS: "/>
     ....
     ....
 </module>
```

Figure 2, Framework configuration file.

menus, toolbars and modules, which in turn take their instructions from configuration details. For example, menus and toolbars when selected parse their action tag, which is a '|' delimited string with the first item specifying the name of the module to receive the message (blank signifying the active module), the second specifying the name of the function to be called, and all remaining parameters passed to the specified function. This allows reconfiguration of the system without having to recompile code.

The framework also provides a fully internationalised system able to display multiple local languages and different icon themes using language and icon theme files, which are selected at runtime via the configuration. It also provides a temporary storage area that programs can use to feed the output of one command to the input of another.

Modules

Plug in modules are components that provide interactive and non-interactive services, and can be loaded at start-up time or upon user request. They can run in a separate thread from the framework and interact through communication channels. JGrass provides a number of system modules such as:

- htmlviewer which provides a HTML browser for viewing data and help text;
- monitor, which provides a 2D/3D graphics monitor for viewing the map data;
- console, which provides a user console for the entry of user commands;
- chartviewer which provides a 2D/3D chart engine;
- mapprinter which provides a WYSIWYG printer editor
- SQL database manager/editor that provides an interactive database SQL statement editor.

Data I/O

JGrass provides access to a number of GIS data formats natively by abstracting data sources and implementing readers and writers to a common standard. GRASS, ESRI Shape and SQL databases are among the first sets of readers and writers that have been implemented. In the case of SQL databases, the Java Database Connectivity (JDBC)[6] API provides cross-DBMS connectivity to a wide range of SQL databases and access to other tabular data sources, such as spreadsheets or flat files. Because all JGrass programs use readers and writers to access data, the addition of new native formats, accessible to all programs, is relatively simple.

JGrass features support for the Common Internet File Systems (CIFS) [7] which provides access to shared files and directories on SMB file servers (i.e. a Microsoft Windows "share"), thus allowing GRASS locations to be either local or remote (Figure 3).

```
<application name="JGRASS Test Database" desc="GRASS Spearfish dataset" group="general">
  <gui class="jgrass.module.grassgui.grass" icon="Cut16.gif" />
  <data dbase="file:/home/grassdata" location="spearfish" desc="South Dakota">
    <mapset name="PERMANENT" desc="permanent mapset description"/>
  </data>
  …..
  …..
</application>
<application name="Geochemistry of Jamaica" desc="Jamaican soil geochemistry" group="general">
  <gui class="jgrass.module.grassgui.grass" icon="Cut16.gif" />
  <data dbase="cifs://jgrass;user:password@localhost/grassdata" location="jamaica" desc="Jamaica" >
    <mapset name="PERMANENT" desc="permanent mapset description"/>
  </data>
  …..
  …..
</application>
```

Figure 3, GRASS location configuration.

Core modules

Communication System
The communication system provides a thread safe synchronous one way communication channel using broadcaster objects which maintain a list of listener objects that are traversed whenever a message is to be broadcast.

Console System
The console system is a plug in module that translates user input into Java statements for execution by a Java engine (Beanshell) [8]. The engine, which currently runs in a separate thread, will in the future be integrated into a cluster of execution engines, the design of which has not yet been settled.

To harness the wealth of native GRASS code already available, the console module allows the execution of non-interactive native GRASS commands that do not require access to the display monitor, via the Java Native Interface (JNI) [9]. Once the command line parameters are gathered and the required GRASS environment variables set, it is passed to the Java engine for execution.

Display System

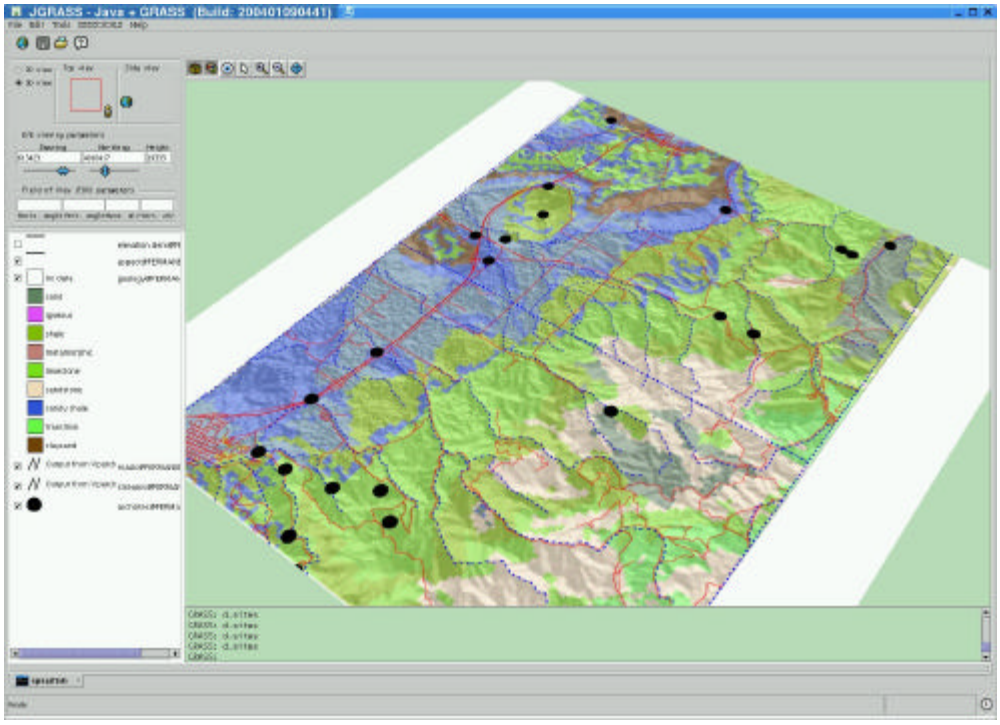

Figure 4, 2D/3D Display subsystem

The display system is a plug in module that provides an OpenGL canvas for the rendering of map images. It maintains a graphics communication channel that programs can use to send graphics events, such as display objects to be added to the display list for rendering, and display rendering details. Display objects contain raster, vector or point map data, along with attribute, drawing style and colour display information, and JGrass provides colour table information for all classes of display objects using the colour rules previously only available for GRASS raster data. For vector and point data display, a text

based symbol display language is available that allows the drawing of images, polygons or circles, and specification of filling patterns and line width.

All rendering is done in 3D, with the default 2D view locating the eye of the user directly above the map, looking down. The rendered map is comprised of a static terrain layer and a dynamic surface layer. The static terrain layer contains data that will not change during the JGrass session, while the dynamic layer contains data that may change, such as a GPS coordinate stream. During map rendering all display list layers tagged as static are merged using alpha blending into a single image on top of which all the remaining layers are drawn. In addition a list of height objects holding information on the height of the display grid nodes is maintained, and each raster, vector or point display object can use any one of these objects to specify the height value to use for display in the 3D space.

To enhance visualization the Real-time Optimally Adapting Meshes (ROAM) [10] algorithm is being ported to JGrass. It constructs the map terrain using a dynamically computed multiresolution triangular mesh for different views, taking into account the level of detail required for each view, minimizing geometric distortions, and maintaining a fixed triangle count. The execution time is proportional to the number of triangle changes per frame, which is typically a few percent of the output mesh size, hence performance is insensitive to the resolution and extent of the input terrain.

JGrass provides a cartographic quality map printing facility (Figure 5) to output maps in a wide range of hardcopy and electronic formats. When the user requests printing, a WYSIWYG print editor is activated that has its own paper-space, supporting a full range of formats, and toolbox allowing the drawing and modification of the properties (stroke, fill, dimension, etc) of simple shapes, importation of images and the like. This preview can be saved as an XML file, printed to a pdf file, or output to a hardcopy device.

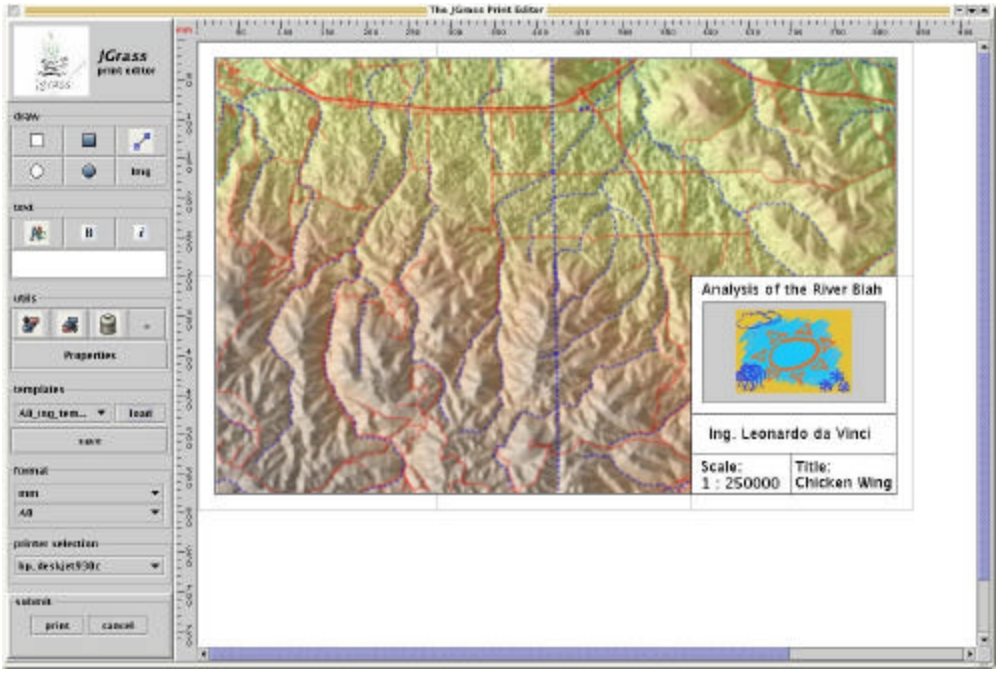

Figure 5, JGrass map printing editor.

GIS Environment
JGrass expands on a number of GRASS features such as:

1. Data mask, that can be either a classical GRASS raster mask, or a rectangular window or a polygon. The data mask controls the visibility of loaded raster, vector and point data. The outer boundary of the masked area, also known as the "Data Mask Window", can be visualised;
2. Elevation layers that are used as the height field in a 3 dimensional view. JGrass allows the loading of multiple elevation layers from raster maps and during the rendering process each raster, vector and point display layer can use any one of these as its height field;
3. Sessions. JGrass allows concurrent session in different windows.

Conclusion

JGrass was started to develop a multi-platform, single code base GRASS GIS environment, and the results to date are encouraging. A number of issues remain, such as memory usage and speed, however these are not insurmountable.

The work to date has been greatly assisted by the choice of Java™ as the operating platform, the large number of open source packages that are available, and individuals who are willing to contribute. An initial alpha release is projected for the end of this year, and the expected increased usage and critique will no doubt improve the software.

It is our hope and expectation that JGrass will provide an extensible base of tools for professionals and researchers that match and surpass those available on any other platform, both in quality and capability.

References

[1] Mitasova H., Neteler M, Freedom in geoinformation science and software development: A GRASS GIS contribution, Proceedings of the Open Source Free Software GIS-GRASS users conference 2002 – Trento, Italy, 11th-13 September 2002.

[2] Neteler M, Mitasova H, Open Source GIS: A Grass GIS Approach 1st Ed. 2002, Kluwer Academic Publishers, Boston, Dordrecht ISBN 1-4020-7088-8.

[3] Kaffe, a free implementation of Java Virtual Machine, http://www.kaffe.org/

[4] B. Alpern, C. R. Attanasio, J. J. Barton, M. G. Burke, P.Cheng, J.-D. Choi, A. Cocchi, S. J. Fink, D. Grove, M. Hind, S. F. Hummel, D. Lieber, V. Litvinov, M. F. Mergen, T. Ngo, J. R. Russell, V. Sarkar, M. J. Serrano, J. C. Shepherd, S. E. Smith, V. C. Sreedhar, H. Srinivasan, and J. Whaley, The Jalapeño Virtual Machine, IBM System Journal, Vol 39, No 1, February 2000. (Source code available as of version 2.0.0 of Jikes RVM.)

[5] The JOGL Project, reference implementation of the Java bindings for OpenGL API, http://jogl.dev.java.net.

[6] M. Fisher, J. Ellis, J. Bruce, JDBC™ API Tutorial and Reference, Third Edition, Addison-Wesley Publishers.

[7] Common Internet File System Technical Reference Revision 1.0, Storage Networking Industry Association Networked Attached Storage working group, http://www.snia.org.

[8] Beanshell, Java source interpreter with object scripting language features, http://www.beanshell.org/

[9] S. Liang, The Java™ Native Interface: Programmer's Guide and Specification, Addison Wesley Longman, Inc. June 1999.

[10] Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein, ROAMing Terrain: Realtime Optimally Adapting Meshes, IEEE Visualization '97, pages 81-88, IEEE Computer Society Press, October 1997, http://www.cognigraph.com/ROAM_homepage/index.html