UDIG Platform Research

April 26, 2004

Submitted To: Program Manager
GeoConnections
Victoria, BC, Canada

Submitted By: Jody Garnett
David Zwiers
Richard Gould
Refractions Research Inc.
Suite 400 – 1207 Douglas Street
Victoria, BC V8W 2E7
E-mail: dzwiers@refractions.net
Phone: (250) 383-3022
Fax: (250) 383-2140

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

This document outlines our evaluation of application and plug-in frameworks in order to determine a platform with which to develop uDig. The choice of the application framework is critical to the success of the uDig project.

Application Framework depends on several factors:

- Professional Appearance
- Accessible Plug-In model
- Community Acceptance
- Compatibility with available GIS libraries
- Market Position

A successful framework must also balance the requirement of the project to finish in a timely manner, while providing a strong foundation for an active open-source development community.

2 uDig Overview

The User Friendly Desktop Internet GIS for OpenGIS Spatial Data Infrastructures project (*uDig*) will create an open source desktop GIS application, to make viewing, editing, and printing data from CGDI and local data sources simple for ordinary computer users.

Open source components are a critical part of the CGDI vision, because they allow organizations to deploy infrastructure widely, in a distributed fashion, without incurring multiple licensing fees. Open source components are also the most tractable for fast support of new OpenGIS interoperability standards.

There are already many different pieces of open source software that implement OpenGIS server standards: Mapserver implements WMS, GeoServer implements WMS and WFS-T, PostGIS implements SFSQL, DeeGree implements WMS and WFS, and so on. However, there is not a single piece of desktop software capable of binding information from all these servers together into a unified desktop view. *uDig* is the open source application which will bring CGDI data sources to the desktop, and integrate them with local data sources for standard business processes – data viewing, data editing, and data printing.

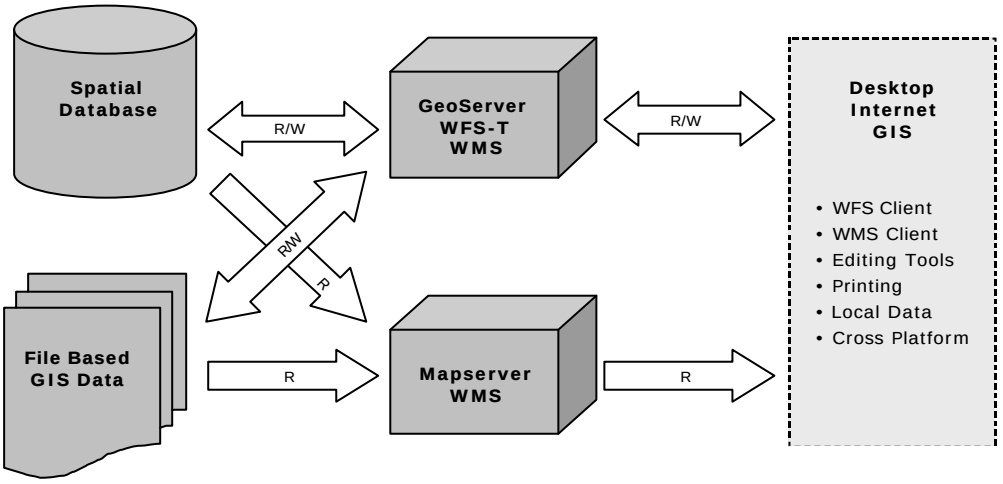

Figure -- 1 OpenGIS Spatial Infrastructures

3 APPLICATION FRAMEWORK BACKGROUND

This section provides an overview of the several factors used to evaluate Application Frameworks for uDig.

3.1 Existing Frameworks

The Java community plays host to several advanced application frameworks that are possible considerations for the uDig project.

3.1.1 NetBeans

NetBeans is the open source project associated with the Forte for Java offering from Sun. It was purchased by Sun in 1999 to fill out their development platform.

When Sun released NetBeans to the open source community in 2001, developers started utilizing the platform it was based on to develop their own modular applications.

NetBeans features:

- A significant amount of built-in management modules
- Is based on Swing widget toolkit
- A large community

NetBeans limitations:

- Lack of quality documentation
- The use of Swing provides an interface with poor platform integration
- Difficulty developing plug-ins could hinder community involvement

Resources:

- http://www.netbeans.org/
- http://wwws.sun.com/software/sundev/jde/

3.1.2 Eclipse

Eclipse presents itself as an “Integrated Development Environment for anything, and nothing in particular”. Eclipse was started as a research project by IBM, and has recently been handed over to an Eclipse Board of Stewards. IBM uses Eclipse as the starting point for most of their Development offerings, including the WebSphere and Rational software portfolios.

As part of the Eclipse 3.0 development, this successful IDE is spinning off an Application Framework.

Eclipse is based on the use of the Standard Windows Toolkit (SWT).

Eclipse Features:

- Considerable marketing momentum and community excitement
- Use of SWT provides an Professional Appearance on Windows
- Mature, well documented Plug-In Model

Eclipse Limitations:

- Application Framework is a recent addition to Eclipse 3.0 scheduled for summer release
- Use of SWT on GTK based Linux systems does not perform adequately
- Use of SWT may alienate members of the community from Linux, or those seeking to reuse components in Swing based applications

Eclipse represents a clear win on many fronts. Any outstanding questions center around the use of the SWT windowing toolkit.

Resources:

- http://www.eclipse.org/

3.1.3 Jump Unified Mapping Platform (JUMP)

JUMP is the creation of the Victoria BC based company Vivid Solutions. Vivid has a strong reputation in the GIS community through the development of the Java Topology Suite (JTS). JUMP is an offshoot of this project with many of the same goals as the uDig project.

JUMP Features:

- An active plug-in development community
- JTS based Geometry
- JUMP Feature Model

Limitations of JUMP:

- Feature Model is limited by memory
- Feature Model does not support nested Attributes, or multiple Geometries
- Plug-Model is well documented, but not well defined

Refractions Research has a strong history of collaboration with both Vivid Solutions and the JUMP project. While CVS access is public, project involvement (and CVS commit status) is relatively closed.

To be useful for the uDig project, JUMP would need to be converted to use the Geotools2 definition of Feature, and the GeoTools2 DataStore API.

Resources:

- http://www.vividsolutions.com/jump/

3.2 User Interface Toolkits

There are several User Interface toolkits available to the Java developer.

3.2.1 Abstract Window Toolkit (AWT)

AWT provides a mapping to the native OS supplied widget set against a common Java widget set. This mapping is provided by an Abstract Factory Pattern centred around the AWTToolkit class.

AWT has fallen into disfavour over the years; the task of isolating the AWT widget model from the Platform Specific Widgets used for user interaction has proved difficult. This limitation is often presented as the paraphrase "Write once, test everywhere".

AWT has several limitations:

- The performance of AWT is limited by the creation and mapping events between AWT and native widgets. This overhead limits the porting of Java applications to small hand held devices.
- Impedance mismatch between Native Widgets and AWT model
  For example, Mac's AWT implementation forces users to hold down key combinations to mimic the AWT model of multiple button mice.

3.2.2 Swing

Swing represents an attempt to eliminate the two major limitations of the AWT toolkit. Swing dispenses with the native widgets altogether, preferring instead to provide a "pure Java" solution with associated cross platform compatibility.

To ease the pain of having Java applications appear different, a very strong "Look and Feel" separation has been imposed on the Swing API. The intention has been to allow the development of Look and Feel implementations that mimic the appearance of native widgets. In practice this has resulted in Swing applications that look a year or two out of date, and never quite work exactly as users expect.

3.2.3 Standard Window Toolkit (SWT)

SWT represents a rethinking of the Java widget set problem by IBM. It represents a return to the roots of the problem and takes an approach similar to AWT. Native widgets are mapped to a set of Java classes.

SWT provides its mapping as a platform specific jar with associated native code. The advantage of this approach is that the SWT API is well defined and easy to implement as native code. This contrasts with the AWT approach in which considerable effort was spent focusing on a cross platform common model.

As part of Eclipse 3.0, SWT has gained the ability to integrate with AWT and Swing, including the embedding of normal AWT/Swing widgets in SWT applications.

3.3 Open Source GIS Libraries

The open source Java community is very active and has produced a large body of high quality code.

While several of these libraries overlap each other the development community is in constant communication and seeks to avoid duplication. An example of this is Geotools' plans to switch over to GeoAPI interfaces, as they are made available.

3.3.1 GeoTools (GT2)

Geotools 2 is an open source, Java GIS toolkit for developing OpenGIS compliant solutions. It has a modular architecture, which allows extra functionality to be added or removed easily. Geotools 2 aims to support Open GIS and other relevant standards as they are developed.

The aim of the project is to develop a core set of Java objects in a framework, which makes it easy for others to implement OGC-compliant, server-side services or provide OGC compatibility in standalone applications or applets. The GeoTools 2 project comprises a core API of interfaces and default implementations of those interfaces.

Capabilities:

- Feature Model
- Coordinate Reference System
- Reprojection
- Feature Loading
- Grid Loading
- Style Layer Descriptor
- Widget Independent Rendering
- Validation
- Graph and Network support

3.3.2 Java Topology Suite (JTS)

The Java Topology Suite (JTS) is an API providing a spatial object model and fundamental geometric functions. It implements the geometry model defined in the OpenGIS Consortium Simple Features Specification for SQL.

Capabilities:

- Geometry classes
- Stable Spatial Operations
- Spatial predicates (based on the DE-9IM model)
- Overlay functions (intersection, difference, union, symmetric difference)
- Buffer
- Convex Hull
- Area and distance functions
- Topological validity checking

3.3.3 GeoAPI

There is an enormous amount of effort being expended out in the Open Source community towards building GIS solutions. GeoAPI aims to reduce duplication by providing a neutral, interface only, API which projects can use to interoperate.

Existing mappings:

- CRS
- Grid Coverage

Future Mappings:

- Geometry Interface
- Feature
- Metadata

3.3.4 OpenMap

OpenMap is a JavaBeans based toolkit for building applications and applets needing geographic information. OpenMap is not OGC based and is therefore not suitable for the uDig project.

4 EVALUATION METHOD

We will evaluate the Application Frameworks based on several areas of interest as outlined in the following sections.

4.1 Market Positioning

Each framework provides its own advantages and disadvantages from a marketing perspective. Each framework will be considered in terms of its broader acceptance as a measure of its life expectancy. Any marketing opportunities will also be considered.

Specifics:

- Framework history and recent developments
- Framework short term goals (over the course of uDig development)
- Framework plans over two years

4.2 Professional Appearance

The evaluation of application appearance is subjective in nature. Our evaluation of acceptance is based on the end-user's experience. The resulting application must appear to be a native application, and operate as expected.

At one level this evaluation comes down to an appraisal of the User Interface Toolkit used by each Framework.

- AWT: No framework is based on the AWT toolkit
- Swing: Provides poor integration with the Windows Desktop, user interface "look" is not considered sufficient.
- SWT: Provides excellent integration on Windows Desktop, but poor performance on Linux.

Specifics:

- Visual Integration with OS
- User Interface Responsiveness
- Use of Drag and Drop
- Use of native file selectors

4.3 Plug-In Model

The accessibility of the plug-in model is important to the long-term success of the uDig project. In order for an open source project to succeed it is very helpful to allow developers to contribute code based on modules or plug-ins.

This separation allows developers to work independently, and often at arms length from the uDig project while still providing value to end-users.

Our ability to solicit open source contributions will often be reduced to how long it takes an interested party to contribute a feature to the application that they desire.

Specifics:

- Impression of Plug-In Design
- Documentation accessibility and quality
- Quickstart or tutorial showing a respect for new developers
- User Interface guidelines
- Version controlled plug-in management
- Time required to create a simple Plug-In
- Average Plug-In Size

4.4 Application Framework

A successful project is one in which the desired product is completed within the allocated budget. To this end, it is important for us to understand how cost effective our time is for each platform.

There are multiple factors, which can affect the tool-adoption of a product, some of which include user interface appearance, usability, and installation time. Product size indirectly affects tool adoption, and should be considered. Overly large products can hurt tool adoption in two ways: length of download or length of installation times.

Specifics:

- Time required to create a simple Stand-Alone Application
- Application Size

4.5 Compatibility with GIS libraries

The most mature renderers available are currently part of the Geotools2 library. The available rendering API offers widget independence and Style Layer Descriptor (SLD) support. This represents significant intellectual property that uDig should leverage in order to be successful (and standards compliant).

On a practical level, geotools2 provides two implementations of their rendering API, both based on the Swing API. If this proves insufficient based on performance we may need to explore other rendering technologies. A separate rendering technologies report is being prepared.

Specifics:

- Use of geotools2 “LiteRenderer”
- Availability of OpenGL support or Java3D

4.6 Community Acceptance

We are concerned with project acceptance with several target communities.

Specifics:

- Target uDig user base (Organizations with OGC based workflow)
- Geotools2 development community
- JUMP development community

5 NET BEANS (FORTE FOR JAVA)

5.1 Quick Summary

Evaluation	Status
Integration	None
Responsiveness	Windows good, Linux great
Drag and Drop	Yes
File Selector	Swing
History	Sun
Short term	3.5 release with redesigned user interface
Long term	Supported by Sun, declining market share
Plug-In Design	Complicated, hard to learn
Plug-In Size	10.1 KB
Plug-In Dev. Time	> 4 hrs
Internal Communication	Excellent
Application Size	11.4 MB
Application Dev. Time	>1.5 hrs
Documentation	Poor
Quickstart	No
UI Guidelines	Swing User Interface Guidelines
Versioned	Yes
LiteRenderer	Yes
OpenGL	Yes
OGC	--
Geotools2	--
JUMP	--

5.2 Application

5.2.1 Development Time

When completing this evaluation for the NetBeans platform, we stopped after 1.5 hours as we had not yet found a working example. At this time we were not close to creating a simple stand-alone application. Therefore we terminated this experiment.

5.2.2 Documentation Accessibility and Quality

One of the substantial factors when learning a new API is documentation, in particular the documentation provided for the novice developer.

The NetBeans platform has:

- Very few examples of creating standalone applications using their framework
- Not provided tutorials including working code
- Out-of-date for the most recent release.

NetBeans failed to provide useful or relevant documentation.

5.2.3 Professional Appearance

The opening appearance of an application can have a huge impact on the adoptability of a tool, especially a client desktop application. We investigated two versions of NetBeans: Windows and Linux.

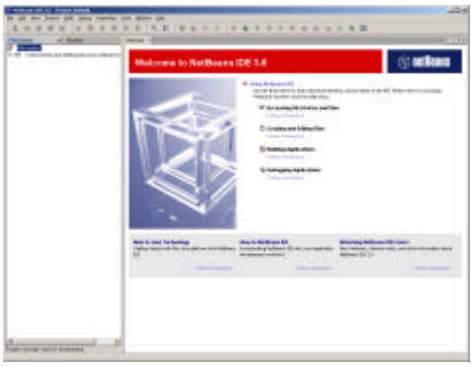

Figure -- 2: NetBeans in Windows

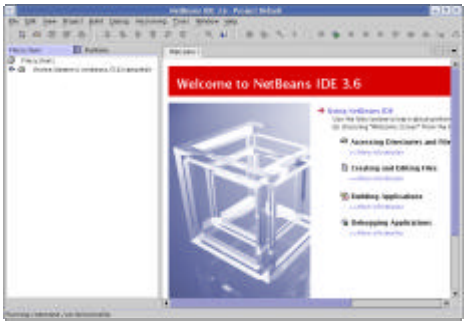

Figure -- 3: NetBeans in Linux

Both of the NetBeans platforms (Figure 2 and Figure 3) appear pleasing to the eye, and do not represent any immediate shortcomings with respect to the appearance of the application. As we see in these Figures, the Linux and Windows versions of NetBeans are almost identical.

5.2.4 Performance

We measured performance through user experiences. We tested each of the versions of NetBeans, with high levels of success.

Both the Windows and Linux versions of NetBeans performed well, giving timely user responses. For both versions we do not foresee any performance issues related to platform choice.

5.2.5 Product Size

The base NetBeans platform, without any plug-ins, or the ability to complete any tasks is 11.4 MB on Windows. This includes the NetBeans framework, and the required graphics libraries.

5.3 Plug-In Development

5.3.1 Development Time

A total of four hours was spent attempting to develop a plug-in for the NetBeans IDE to no avail. It is difficult to say whether we were close or not, as we could find no working example to compare against.

5.3.2 Documentation Accessibility and Quality

A large factor in designing and implementing a plug-in for a program is dependant on the amount of quality documentation or tutorials available to get one started.

A majority of the time spend attempting to develop the plug-in was spent hunting down documentation, mostly without success. A lot of the work done was guesswork, along with some probably out of date documentation that was stumbled upon at one point.

In addition, no working examples or tutorials could be located.

5.3.3 Product Size

The end size of our non-working plug-in totals 11KB, about five times that of a minimal working Eclipse plug-in. Still, for plug-ins this small, size comparisons are really irrelevant.

5.3.4 Internal Communication

The NetBeans plug-in communication model is acclaimed as one of its best features. It make use of a series of well defined abstractions and a mature event model to allow plug-ins to communicate in an effient manner.

6 ECLIPSE

6.1 Quick Summary

Evaluation	Status
Integration	Good windows/ Linux GTK, Poor Linux Motif
Responsiveness	Windows/ Linux Motif good, Linux GTK bad
Drag and Drop	Yes
File Selector	Native
History	IBM research project
Short term	3.0 target for mid summer, with application framework and AWT/Swing integration
Long term	Recently handed over to an independent body, IBM and others porting development tools
Plug-In Design	Quick to learn, difficult cross plug-in communication
Plug-In Size	2.12 KB
Plug-In Dev. Time	0.5 hrs
Internal Communication	Sufficient
Application Size	5.3 MB
Application Dev. Time	1.5 hrs
Documentation	Excellent
Quickstart	Excellent
UI Guidelines	Yes
Versioned	Excellent
LiteRenderer	Unknown, claimed for 3.0 branch
OpenGL	Beta
OGC	--
Geotools2	Enthusiastic non-Linux community
JUMP	--

6.2 Application

6.2.1 Development Time

When completing this evaluation for the Eclipse platform, we spent 1.5 hours before a simple application had been completed. This included working through a tutorial and deploying an empty application.

6.2.2 Documentation Accessibility and Quality

One of the substantial factors when learning a new API is documentation, in particular the documentation provided for the novice developer.

The Eclipse platform has:

- Multiple concrete examples of creating standalone applications using their framework
- Tutorials including working code
- Up-to-date for the most recent release.

Creating a sample standalone application using the Eclipse platform was well documented.

6.2.3 Professional Appearance

The opening appearance of an application can have a huge impact on the adoptability of a tool, especially a client desktop application.

We investigated three versions of Eclipse: Windows, GTK for Linux and Motif for Linux.

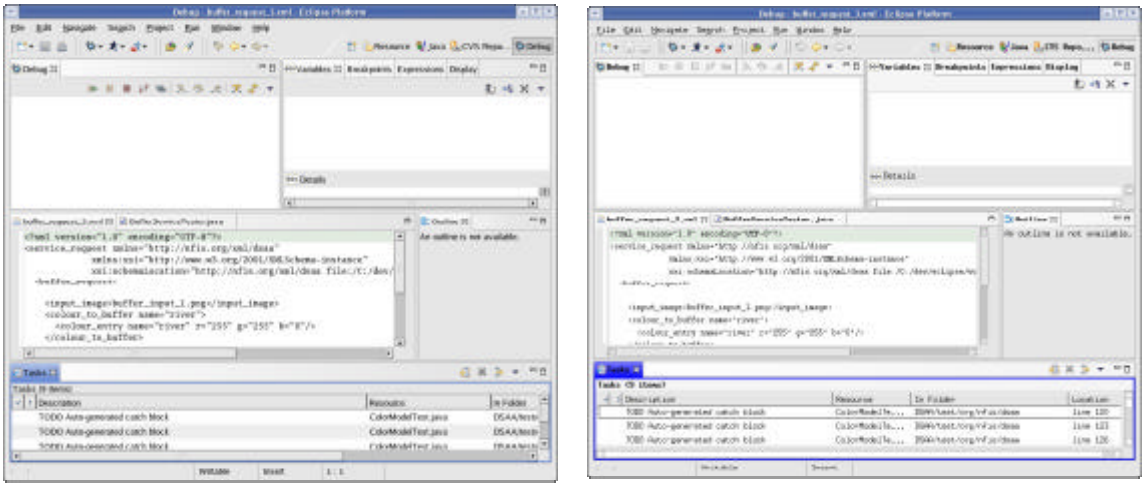

Figure -- 4: Eclipse Linux (GTK/Motif)

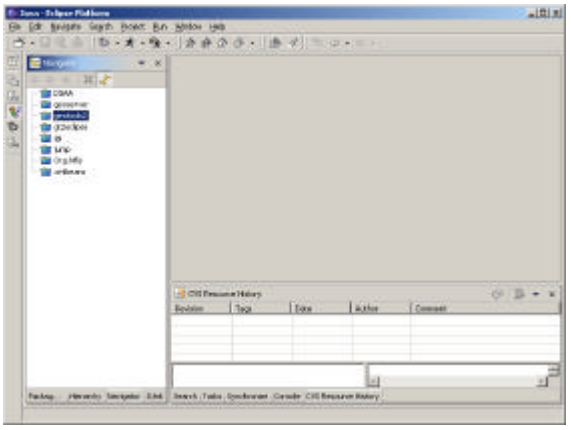

Figure -- 5: Eclipse Windows

In Figures 4 and 5 we see some of the differences in appearance between the versions. Of the three versions, the Motif version looks the worst. The GTK and Windows versions are both aesthetically pleasing.

Although it is a performance issue it should be noted that the GTK release of eclipse was unusable due to performance problems

6.2.4 Performance

We measured performance through user experiences. We tested each of the three versions of Eclipse, with varying success.

Both the Windows version and the Linux Motif version performed reasonably well, without substantial delays. For both of these versions we do not foresee any performance issues related to platform choice.

Unfortunately, the Linux GTK version did not perform well, and is well below standard.

The commercial version of Eclipse (WebSphere) has moved to at Linux/qt SWT implementation to resolve performance problems on Linux platforms. A linux/fox binding is also underdevelopment.

Refractions could look into licensing the linux/qt version of SWT for uDig, although restrictions prevent this from being an option for open-source development. The QT toolkit is being presented as a competitor with Java for enterprise development.

6.2.5 Product Size

The base Eclipse platform, without any plug-ins, or the ability to complete any tasks is 5.3 MB on Windows. This includes the Eclipse framework, and the required SWT libraries.

6.3 Plug-In Development

6.3.1 Development Time

After working at this for fifty minutes, we had a minimal working Eclipse plug-in that was ready to re-deploy on other systems easily. The process was very smooth and easy and problem-free.

6.3.2 Documentation Accessibility and Quality

The documentation for Eclipse is extraordinary. It is well organized, readable, and relevant. Tutorials and working examples are abundant.

The tutorial used for implementing the plug-in explained each step nicely as it guided us through the process.

6.3.3 Product Size

The final deployable plug-in size totals 2.12 KB. The plug-in itself was very minimal, consisting of one class that generated a new view pane within Eclipse itself and displayed a string.

6.3.4 Internal Communication

Internal communication within the Eclipse platform is not well documented, and may be too specific for our needs. There are sections dedicated within the Resources package that allow communcation between plug-ins, but they might be specifically geared for the IDE. Other aspects of the workbench have areas that other plug-ins can hook into which may be sufficient. For example, an editor can have a plug-in listen for a save request.

7 JUMP UNIFIED MAPPING PROJECT (JUMP)

7.1 Quick Summary

Evaluation	Status
Integration	Swing Loose 'n' Free based
Responsiveness	Painful, slow for large datasets, memory bound
Drag and Drop	No
File Selector	Swing
History	Vivid Solutions Inc. Open Source project
Short term	Active development, moving to public plug-in CVS
Long term	Refractor into separate projects
Plug-In Design	Simple
Plug-In Size	10 kB
Plug-In Dev. Time	1 hr for simple plug-in, 2 weeks for full functionality
Internal Communication	Good, similar to a servlet context
Application Size	8.3 mB
Application Dev. Time	Already set-up
Documentation	Incomplete but of good quality
Quickstart	Yes
UI Guidelines	None
Versioned	1.1.1
LiteRenderer	Requires extensive redesign
OpenGL	Requires extensive redesign
OGC	Receptive
Geotools2	License incompatibility
JUMP	Positive

7.2 Application

7.2.1 Development Time

Jump is already set-up as a standalone application. The application is GPL so any changes made to the core application must be made available to the community. Unfortunately though, this takes time as this is not an 'open-developer' project.

7.2.2 Documentation Accessibility and Quality

The documentation is sparse and incomplete. Where documentation exists, the documents are high quality.

7.2.3 Professional Appearance

JUMP is Swing based and does not provide the desired professional appearance.

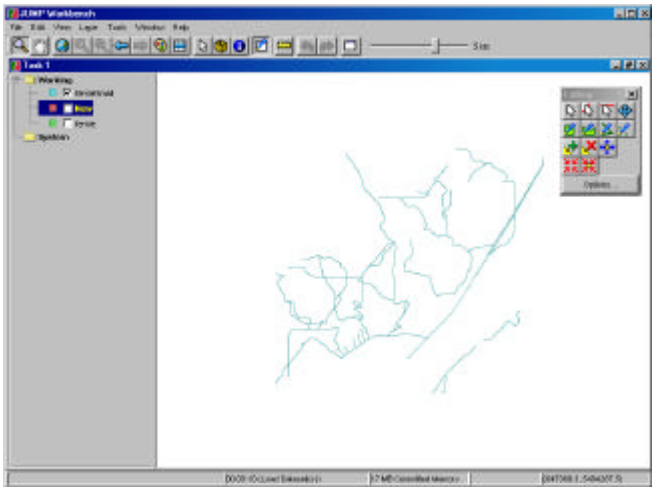

Figure -- 6: JUMP on Windows

Specifically:

- Use of MDI interface
- Non standard tree widget
- Toolbars are based on Windows 95 widget set resulting in an application that stands out against a Windows 2000k or Windows XP installation

To improve this aspect of jump we would follow the Java User Interface guideline, and upgrade to the JDK 1.5 beta.

Changes of this nature would be considered high risk, as JUMP, while open source, is not an open-developer project.

7.2.4 Performance

The JUMP Feature model is memory based, resulting in a application that is slow and memory bound. Initial load from database or shapefile data source must first be loaded into memory before being used on screen.

The extensive use of progress bars provides immediate feedback for most operations.

7.2.5 Product Size

The 8.3 MB download size of Jump 1.1.1 is very attractive.

7.3 Plug-In Development

7.3.1 Development Time

Initial plug-in development time can be completed in less than one hour. In this case the tutorials aided greatly in plug-in development.

Resources:

- http://www.vividsolutions.com/jump/bin/JUMP Developer Guide.pdf

7.3.2 Documentation Accessibility and Quality

The plug-in documentation, where available, is of high quality. At times we found the documentation was incomplete.

We also found keyboard limitations involving GUI shortcuts and menu selections.

7.3.3 Product Size

Requires manual install, and modification of application plugin.xml file. JUMP plug-ins average 10 kB in size.

7.3.4 Internal Communication

JUMP uses a model similar to that of a servlet context to allow plug-ins to communicate with each other. The use of black board pattern at both the task and application level is provided for informal ad hock communication between consitlations of plug-ins.

This enabled plug-ins to dynamically discover and communicate with each other with out direct a dependency. This is a strong well understood model similar to the J2EE use of Application, Servlet and Request context.

For communication with the existing JUMP Framework plug-ins make use of direct dependency and the Framework event model. This presents a clear separation between plug-ins and the JUMP Framework in terms of communication style.

As a bridge several JUMP derived applications post a large grained object model onto the clipboard where other plug-ins can register for specific events. This does introduce direct dependency into inter plug-in communication with associated configuration complexity.