Milestone 2 Report

uDig

October 12, 2004

Submitted To: Program Manager
GeoConnections
Victoria, BC, Canada

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

TABLE OF CONTENTS

1 SUMMARY OF WORK ACCOMPLISHED 3

2 ENCOUNTERED PROBLEMS AND SOLUTIONS 4

2.1 WEB REGISTRY SERVICE DOCUMENTATION 4
2.2 CATALOG SERVICES 4
2.3 STREAMING FEATURES 4
2.4 ECLIPSE/MACINTOSH 5
2.5 WMS PRINTING 5

3 WORK PLAN 7

1 SUMMARY OF WORK ACCOMPLISHED

The goal of this project is to build a GIS Desktop Client that integrates with Open GIS Consortium (OGC) Open Web Services (OWS).

The Application Framework we are using is called Eclipse Rich Client Platform. Eclipse Rich Client Platform is an offshoot of a popular Java Integrated Development Environment and offers a unique approach to Java User Interface development.

All required documents have been completed for this second milestone. These documents are:

Phase 2.4

- Data Access Developers Guide

Phase 2.10

- uDig Framework & User Interface Recommendations
- Draft Users Guide (online)
- Milestone Report (this document)

For Milestone 2 we have completed the following software release:

Phase 2.10

- Alpha release of uDig application (version 0.4)

We have been very pleased with the assistance provided by our partners and the open source community in supporting the uDig project.

This work is publicly available via the uDig project website, community wiki and Subversion repository.

2 ENCOUNTERED PROBLEMS AND SOLUTIONS

2.1 Web Registry Service Documentation

The availability of OGC Web Registry Service documentation has been resolved. We have discovered that Web Registry Service has been renamed "OGC Web Services Stateless Catalog Profile". It is still difficult to acquire current information about this specification as it remains in draft form. This profile operates as an HTTP binding with the recent release of Catalog Services version 2.0. We will be speaking with Ionic to locate a test server.

2.2 Catalog Services

The Catalog 2.0 specification mentioned above incorporates names for core concepts: as an example the entity Catalog has been renamed Discovery. The specification is more complete, including a BNF grammar for queries. These changes and additions will result in a slightly more involved implementation then expected.

Our involvement with GeoAPI continues to pay off with respect to implementing these services. GeoAPI has become an official OGC Working Group, and is supported by The Deegree Project (reference implementation of Catalog Services version 2.0). These contacts will ensure our questions are answered in a timely manner, and may provide us with APIs to work against.

2.3 Streaming Features

One of the goals of the WFS portion of this project was to “stream” features over the Internet. Aside from the usual Internet client issues, we have encountered two main problems: Schema Parsing and Content Streaming.

Schema parsing and interpretation was recognized early on in the design stage as a particularly difficult portion of a WFS client. Our initial solution to this problem was outlined in our WFS Design Document (delivered for the previous milestone). At that time a proof of concept had not been completed. We have since ironed out the remaining glitches, producing a strong schema parsing algorithm.

The second problem was particularly interesting because of the options available. Although streaming content, or parsing content on-the-fly, is not a new concept, we had to perform this task while running within an existing parser. This meant that we had three options: create our own XML parser, link directly into an existing parser, or use the generic XML parser API. The first option was well beyond the scope of this project, and was not reasonable as there are many good XML parser implementations freely available. The second option has been used in the past, but unfortunately the result is tightly tied to one implementation,

which greatly reduces the flexibility of the resulting code. Therefore the third option was preferable.

Using the existing generic API was superb when parsing documents at once, but it did not provide an option for pausing the parser. Therefore, we decided to place the parser in its own thread of execution, allowing it to communicate with the main thread of execution through a buffer. This is not a new design, but had been untested as our plan included pausing the Internet connection and the parser at will. Our WFS client successfully streams features over the Internet, pausing both the parser and the Internet connection as required (including a configurable timeout).

2.4 Eclipse/Macintosh

There were problems encountered with making Eclipse work correctly with Macintosh systems. These occurred because uDig uses Java Advanced Imaging (JAI) to improve rendering performance. Java Advanced Imaging only works with Java images, however Eclipse uses its own type of images. In order to use JAI with Eclipse we decided to use a SWT_AWT bridge, which is included with Eclipse. The SWT_AWT bridge allows Java AWT GUI constructs to be embedded into Eclipses interfaces. Unfortunately, the SWT_AWT bridge requires Java 5.0 on Macintosh. Unlike Linux, which also requires Java 5.0, Java is part of Macintosh's operating system, so it cannot simply be installed. In order to make uDig operate on Macintosh systems, we had to create a class that converted Java images into Eclipse images. Unfortunately this was only part of the solution.

Java AWT and Eclipse SWT (Eclipse's user interface components) use different classes for input and output. Along with creating a class to convert Java images to Eclipse images, new mouse listeners had to be created. Additionally, the methods that provide instant user feedback, for example a zoom box, use different classes. It may seem like using the SWT_AWT bridge is a poor design decision, however, the decision was made because in the near future (perhaps 1-2 years from now), the SWT_AWT bridge will be present on all platforms. The SWT_AWT bridge also permits a faster implementation.

2.5 WMS Printing

When printing WMS maps, the quality of the output required is often too high for the WMS to handle. Typically, requests higher than 2000 pixels by 2000 pixels are not accepted. Common printing resolution requirements easily exceed this. A system was required that split the desired image into multiple, smaller WMS requests and tiled them back together.

The Tiling WMS Renderer was created to split the printed image into manageable chunks, or tiles. It communicates with the server in an attempt to determine the maximum size of request allowed, but defaults to 2000 by 2000 if it is unable to do so. It then splits the image evenly into the biggest tiles available and requests each of them individually, reconstructing the image once all the tiles are complete.

A foreseeable problem with this system is that some WMS servers include a decorator within every request, so a tiled WMS image would include a decorator inside each individual tile. One possible solution is to overlap the tiles a bit, covering up the area where the decorator resides.

3 WORK PLAN

Our work plan for the next milestone, Milestone 3, is as follows:

Completion Date	Event	Specific Tasks
October 24, 2004	Release 0.5	1) User Interface Prototype complete 2) Style Editor 3) Implement recommendations to Printing Framework
December 17, 2004	Release 0.6	1) Reprojection Support 2) Implement recommendations to WFS DataStore 3) Template Definition 4) Extend Printing Framework 5) Testing support for ArcSDE, OracleSpatial, MySQL, PostGIS 6) WMS Post and SLD
January 4, 2005	Release 0.7	1) Feature and Raster Renderers 2) Layer Manager and Selection 3) Catalog Services and Stateless Catalog Profile
January 11, 2005	Milestone 3	Milestone 3 complete: **Beta release** of uDig application