UDIG Printing Technologies Report

June 14, 2004

Submitted To: Program Manager
GeoConnections
Victoria, BC, Canada

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

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

This document outlines the evaluation of available printing technologies and their capabilities in order to determine which technology to use in the development of uDig.

The choice of an appropriate printing technology is a critical aspect of the uDig project.

Printing technology depends on several factors:

- Close to Cartographic Quality Output
- Cross-platform compatibility
- Ability to print to PDF

We found a limited number of printing solutions for the Java platform and have focused on the native Java printing support and SWT. The requirement to print PDF files seems best met by a combination of Java Platform printing and the use of an open-source PDF library.

2 OVERVIEW OF UDIG

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.

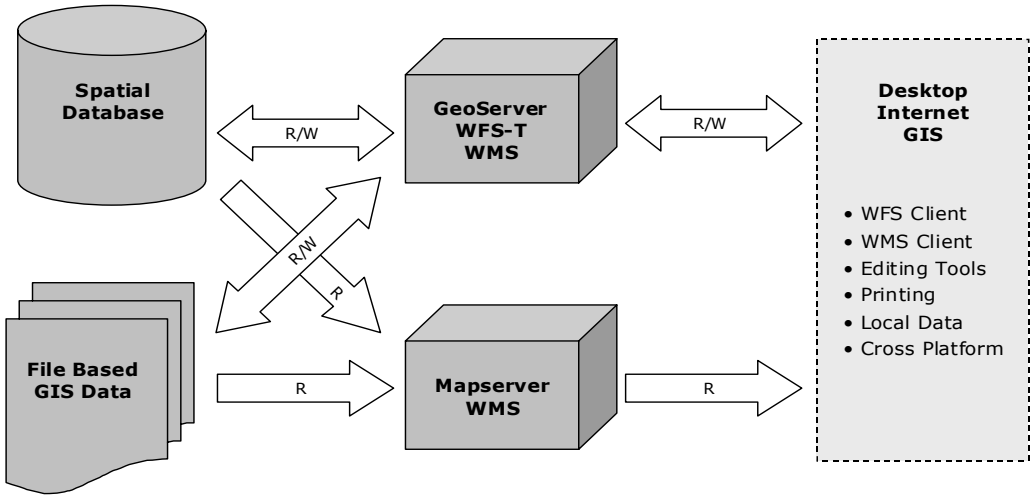

Figure 1 – OpenGIS Spatial Infrastructures

3 OVERVIEW OF PRINTING REQUIREMENTS

It is critical for the success of uDig that an appropriate printing technology is implemented. Users of uDig should be able to create standard and large format cartographic output.

The printing technologies will be evaluated on the following criteria:

- Accuracy, with regards to pixel orientation and spacing
- Cross-platform compatibility
- Ability to print to large-scale printers (e.g., Plotters)
- WYSIWYG – What is displayed on screen is exactly what is printed
- Ability to handle a large volume of vector information
- Integration with Geotools2 Rendering Pipeline

To address some of these issues it would be extremely helpful to output Adobe PDF files so we can rely on local ports of the Acrobat Reader program as an alternative way to provide physical output should any issues arise. It also provides an alternative method to digitally transfer outputs between individuals.

4 JAVA PRINTING

4.1 History of Java Printing

Taken from the Java Print Service API User Guide:

> Basic printing support for the Java platform was first introduced in the Java Development Kit, version 1.1 in 1997. The JDK 1.1 printing API provided developers with a basic framework for printing the user-interface content from client applications. JDK 1.1 printing, also called the AWT Printing API, was designed around the java.awt.PrintJob class, which encapsulates a printing request. The PrintJob class creates a subclass of Graphics, which implements the rendering calls to image the page.
>
> In 1998, the SDK 1.2 advanced printing on the Java platform with the java.awt.print package by allowing applications to print all Java 2D graphics, which includes 2D graphics, text, and images.
>
> For the SDK version 1.3, the JobAttributes and PageAttributes classes were introduced to AWT printing so that client applications could specify the properties of a print job and the attributes of a page.

The Java Print Service was introduced in SDK version 1.4, elaborating on the Java Print API. Here is a list of features that the new service adds to Java Printing, taken from the Java Print Service API User Guide:

- Both client and server applications can discover and select printers based on their capabilities and specify the properties of a print job. Thus, the JPS provides the missing component in a printing subsystem: programmatic printer discovery.
- Implementations of standard IPP attributes are included in the JPS API as first-class objects.
- Applications can extend the attributes included with the JPS API.
- Third parties can plug in their own print services with the Service Provider Interface.

4.2 SDK version 1.2 – Java Print API

The Java Print API introduced in SDK 1.2 uses a callback system and manages the printing process, similar to how it manages the AWT widgets. The application notifies the printing system that it wishes to print. When the printer is ready to receive data, it requests the pages from the application.

From the Java2D Tutorial:

> This *callback printing model* enables printing to be supported on a wide range of printers and systems. It even allows users to print to a bitmap printer from a computer that doesn't have enough memory or disk space to hold the bitmap of an entire page. In this situation the printing system will ask your application to render the page repeatedly so that it can be printed as a series of smaller images. (These smaller images are typically referred to as *bands*, and this process is commonly called *banded printing*.)
>
> To support printing, an application needs to perform two tasks:
>
> - Job control--managing the print job
> - Imaging--rendering the pages to be printed

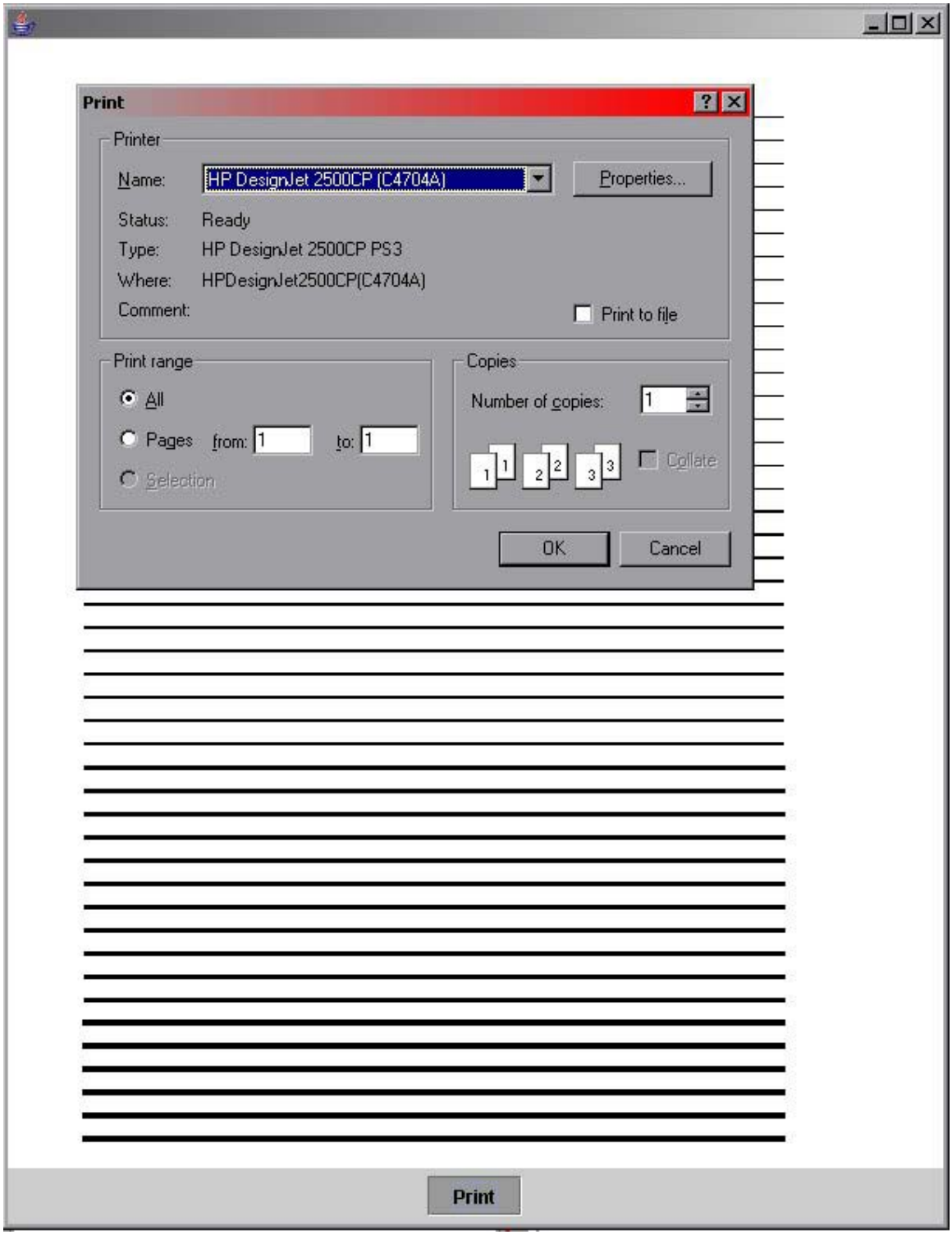

Figure 2 – Java Print API Native Dialog Screenshot

Pros:

- Provides complex functionality
- Supports printing multiple page layouts in one print job
- Supports printing jobs larger than the current memory can hold
- Mature
- Abundance of quality documentation
- What you see is what you get

Cons:

- Does not support discovery of printers based on their capabilities
- Print job size depends on driver support

Abilities:

- Printing to PDF (through external library)
- Printing to Paper
- Is Cross Platform
- Uses native dialogs (maybe, haven't tested linux yet)
- Prints Vectors
- Prints Rasters
- Prints Vectors on Rasters
- Accuracy of Line widths seems to be very good.
- Prints Text
- Rendering of greater than 1/72 inch

The Java Printing API seems to be the most suitable API for our needs. It supports printing to paper, as well as to PDFs (through use of an external library). It works on every platform that Java runs on, using the native print dialog, if available.

The API supports drawing text, vectors, rasters, and vectors on rasters. The accuracy of the data being printed is often more accurate than what is rendered on the screen, especially regarding line width. It is capable of rendering the printing graphics at greater than 1/72 of an inch.

It easily supports printing to pages of varying sizes, allowing output to pages as large as each printer driver supports.

4.2.1 Example

Here is an example application that draws various shapes on the screen and will print when a button is pressed, taken from the Java2D Tutorial:

```
public class ShapesPrint extends JPanel
                         implements Printable, ActionListener {
...
public int print(Graphics g, PageFormat pf, int pi)
                          throws PrinterException {
    if (pi >= 1) {
        return Printable.NO_SUCH_PAGE;
    }
    drawShapes((Graphics2D) g);
    return Printable.PAGE_EXISTS;
}
...
public void drawShapes(Graphics2D g2) {
    Dimension d = getSize();
    int gridWidth = 400/6;
    int gridHeight = 300/2;
    int rowspacing = 5;
    int columnspacing = 7;
    int rectWidth = gridWidth - columnspacing;
    int rectHeight = gridHeight - rowspacing;
    ...

    int x = 85;
    int y = 87;
    ...
    g2.draw(new Rectangle2D.Double(x,y,rectWidth,rectHeight));
    ...
```

The job control code is in the `ShapesPrint` `actionPerformed` method.

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() instanceof JButton) {
        PrinterJob printJob = PrinterJob.getPrinterJob();
        printJob.setPrintable(this);
        if (printJob.printDialog()) {
            try {
                printJob.print();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

4.3 Printing to PDF Format

There are several APIs that can be used to print from a graphics object to a PDF file. The one that looked the most promising is Retep PDF II (available at http://retep.org/retep/home.do). There is very little modification needed for a program to use this library as it implements itself on top of the Java Print API.

Other PDF libraries available:

Library / package	License	Description
Adobe Acrobat Reader for Java	Freeware	Read and display PDF documents. A viewer application and a JavaBean are available. Works with Java 1.1.8+.
Big Faceless PDF library	Commercial	Write PDF documents, with support for various advanced features.
Etymon PJ	GPL / Commercial	Read and write PDF documents. There are two versions, classic and professional. Classic comes under the GPL and requires Java 1.1. Professional is commercial and requires Java 1.4.
FOP	Apache License	Write PDF documents, render them from XML/XSL sources.
gnujpdf	GPL	Write PDF documents. An extension of the retepPDF project.
iText	LGPL	Write PDF documents. Requires Java 1.2.
jPDFPrinter	Commercial (trial version available)	Pure Java library to write PDF files. Library can be used like a Java printer job. Thus, existing Java code for printing can be reused to create PDF files.
JPedal	LGPL	Read PDF documents. This library can both extract content from PDFs and rasterize them.
jPDF	Commercial (trial version available on request)	Manipulation of PDF files, especially suited for the server side. Features include PDF generation from templates, splitting, merging, parsing and encryption. Written in pure Java (requires Java 1.3 or higher).
PDFBox	LGPL	Library to access PDF files. A utility to convert to text is included.
PDFlib	Commercial	Read and write PDF documents. This is a C library which has Java JNI bindings.
retepPDF	GPL	Write PDF documents.
Saffron Document Server	Commercial	Reads PostScript (.ps) documents and generates PDF, HTML, RTF, TIFF, and other formats. Configured as a server for concurrent document rendering.
SmartJPrint	GPL-like	Pure Java library to write PDF files. Generates PDF files from Swing GUI components, provides preview functionality.
XMLMill	Commercial, trial version available	Create PDF documents from XML/XSL.

4.4 SDK Version 1.4 – Java Print Service

Taken from the Java Print Service Guide:

> The Java Print Service API is an extension of the Java Print API that allows printing on all platforms. It includes an extensible print attribute set based on the standard attributes specified in the Internet Printing Protocol (IPP) 1.1 from the IETF. With the attributes, client and server applications can discover and select printers that have the capabilities specified by the attributes. In addition to the included `StreamPrintService`, which allows applications to transcode data to different formats, a third party can dynamically install their own print services through the Service Provider Interface.

The Java Print Service is a new API. As a result, there is not much quality documentation nor many examples available, and it does not explain itself very clearly. It does not seem relevant to the development or implementation of uDig.

4.4.1 Example

Taken from Java Print Service Javadocs:

> This code demonstrates a typical use of the Java Print Service API: locating printers that can print five double-sided copies of a Postscript document on size A4 paper, creating a print job from one of the returned print services, and calling print.

```
FileInputStream psStream;
try {
   psStream = new FileInputStream("file.ps");
} catch (FileNotFoundException ffne) {
}
if (psStream == null) {
    return;
}

DocFlavor psInFormat = DocFlavor.INPUT_STREAM.POSTSCRIPT;
Doc myDoc = new SimpleDoc(psStream, psInFormat, null);
PrintRequestAttributeSet aset =
     new HashPrintRequestAttributeSet();
aset.add(new Copies(5));
aset.add(MediaSize.A4);
aset.add(Sides.DUPLEX);
PrintService[] services =
  PrintServiceLookup.lookupPrintServices(psInFormat, aset);
if (services.length > 0) {
   DocPrintJob job = services[0].createPrintJob();
   try {
     job.print(myDoc, aset);
   } catch (PrintException pe) {}
}
```

Packages persist in javax.print.

5 JFREEREPORT

JFreeReport is a free Java report library. It has the following features:

- full on-screen print preview;
- data obtained via Swing's `TableModel` interface (making it easy to print data directly from your application);
- XML-based report definitions;
- output to the screen, printer or various export formats (PDF, HTML, CSV, Excel, plain text);
- support for servlets (uses the JFreeReport extensions)
- complete source code included (subject to the GNU Lesser General Public Licence);
- extensive source code documentation;

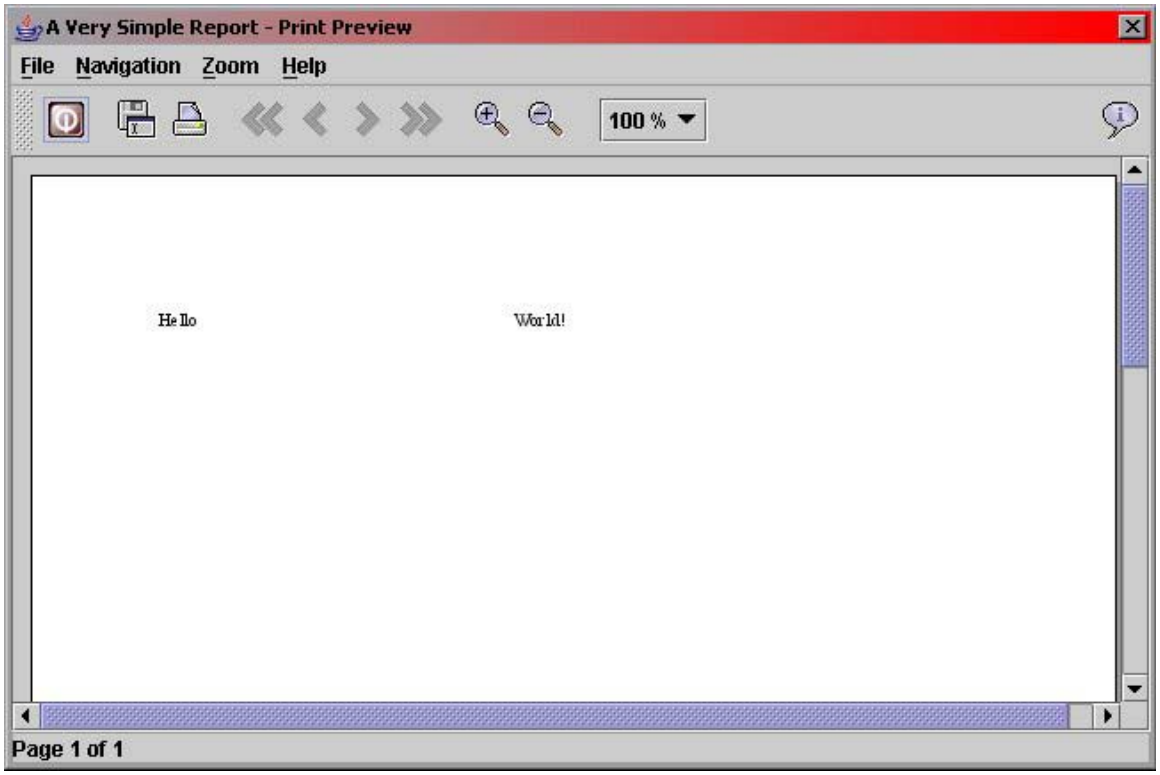

Figure 3 – JFreeReport Demo Screenshot

JFreeReport uses the Java Print API to do its printing and may integrate nicely with uDig, if it suits our needs.

6 SWT PRINTING API

The SWT API offers its own printing aspect. It is relatively new, consisting only of three classes. It is a very bare-bones system, allowing one to expand as needed.

Pros:

- Simple to use
- Works well with the SWT graphics package

Cons:

- Lacking documentation, especially with regards to support of page layouts and formats.
- The GTK implementation of SWT is missing printing.
- A lot more overhead is required.
- Does not work well with Java2D.

SWT printing support in GTK notes:
https://bugs.eclipse.org/bugs/show_bug.cgi?id=24796

6.1.1 Example

Example code taken from the Eclipse site:

```
/*
 * Copyright (c) 2000, 2003 IBM Corp.  All rights reserved.
 * This file is made available under the terms of the Common Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/cpl-v10.html
 */

/*
 * Printing example snippet: print "Hello World!" in black, outlined in red, to default
printer
 *
 * For a list of all SWT example snippets see
 * http://dev.eclipse.org/viewcvs/index.cgi/%7Echeckout%7E/platform-swt-
home/dev.html#snippets
 */
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.printing.*;

public class Main {

public static void main (String [] args) {
      Display display = new Display();
      Shell shell = new Shell(display);
      shell.open ();
      PrinterData data = Printer.getDefaultPrinterData();
      if (data == null) {
             System.out.println("Warning: No default printer.");
             return;
      }
      Printer printer = new Printer(data);
      if (printer.startJob("SWT Printing Snippet")) {
             Color black = printer.getSystemColor(SWT.COLOR_BLACK);
             Color white = printer.getSystemColor(SWT.COLOR_WHITE);
             Color red = printer.getSystemColor(SWT.COLOR_RED);
             Rectangle trim = printer.computeTrim(0, 0, 0, 0);
             Point dpi = printer.getDPI();
             int leftMargin = dpi.x + trim.x; // one inch from left side of paper
             int topMargin = dpi.y / 2 + trim.y; // one-half inch from top edge of
paper
             GC gc = new GC(printer);
             Font font = gc.getFont(); // example just uses printer's default font
             if (printer.startPage()) {
                    gc.setBackground(white);
                    gc.setForeground(black);
                    String testString = "Hello World!";
                    Point extent = gc.stringExtent(testString);
                    gc.drawString(testString, leftMargin, topMargin +
font.getFontData()[0].getHeight());
                    gc.setForeground(red);
                    gc.drawRectangle(leftMargin, topMargin, extent.x, extent.y);
                    printer.endPage();
             }
             gc.dispose();
             printer.endJob();
             }
      printer.dispose();
      while (!shell.isDisposed ()) {
             if (!display.readAndDispatch ()) display.sleep ();
      }
      display.dispose();
      }
}
```