Customizing ArcIMS Using the Java Connector and Python

Randal Goss

The ArcIMS Java connector provides the most complete and powerful object model for creating customized ArcIMS Web sites. Java, however, with its code/compile/run development model, lacks the ease of use of scripting languages for rapid development and prototyping of Web applications. To leverage the power of the Java connector with the rapid development cycle of scripting languages, we are utilizing the Python scripting language. Python is a relatively simple but powerful cross platform programming language that, in addition to running in Windows, was ported to Java. Python in Java, frequently referred to as Jython, is fully integrated with the Java environment allowing the programmer to use and even write Java classes and applets. This paper explores the process of incorporating Jython into JavaServer Pages or Servlets to quickly deploy highly customized ArcIMS Web sites.

Providing data to the users, the people who need it to make key business or policy decisions, has historically been one the the fundamental challenges of GIS professionals. As a result, GIS systems have evolved from monolithic proprietary systems to client server environments supported on standardized data models. One component of this evolution is the incorporation of GIS information into web based clients such as web browsers, or Java based applets. The dissemination of GIS information inside of an interactive client that can be deployed on a web browser is very effective. It can allow users, with little or no training, to view, query, and perform various analysis upon the GIS data.

ArcIMS, ESRI's internet Map Server, fulfills the server role discussed above. It communicates with clients using a standardized XML schema (ArcXML), thus allowing for tremendous variability in the type and function of clients. ESRI has provided ArcIMS users with several standard clients including the HTML Client, and the Java Client. Both of these clients offer a broad base of functionality to the user and are quite effective for the purpose that they were developed. Although these clients can be customized, the amount of time required to make significant changes begins to increase disproportionately with an increasing level of customization due to the size and complexity of the original application. Consequently, ESRI has provided several connectors to the ArcIMS server that can be used to build custom clients meeting specific needs. These include the ActiveX connector, the Cold Fusion connector, and the Java Connector. Of the three, the Java connector is arguably the best supported connector for ArcIMS and provides and object model that wraps much of the functionality of ArcXML.

As users become accustomed to GIS, especially with respect to web clients, they will begin to request very specific customizations which solve niche needs. These solutions generally involve the integration of GIS visualization with data entry, storage, and retrieval. Although these applications might only impact a small number of individuals, they can result in highly productive tools. Unfortunately, the development of custom clients is generally time consuming and can be cost prohibitive when they only address the needs of a small number of people. To make development under these circumstances cost effective, what is necessary is a platform meeting the following criteria:

- It should facilitate rapid application development and the use of reusable components.
- It must support dynamic data driven applications, those that can seamlessly integrate the GIS component with the relational database component.
- Ideally, it should be cross platform, running on all of the ArcIMS platforms.
- It must be easily learned.

There are many possible scripting solutions, each with their own specific strengths and weaknesses. However, there was one solution which seemed ideally suited for ArcIMS development. This solution was the utilization of the ArcIMS Java Connector with a combination of Java and a version of the Python scripting language which runs inside of the Java Virtual Machine(JVM). This version of python is commonly referred to as Jpython or Jython. The benefits realized by deploying this combination are:

- The developer can rapidly prototype an application, and later optimize performance by converting performance limiting code to pure Java.
- Jython is simple to use and quick to learn, yet it is very powerful and well suited for larger projects
- The Java Connector is an excellent connector to the ArcIMS server. It supports cross platform applications and encapsulates most of the functionality of ArcXML.
- There is a seamless integration between Java and Jython. Jython can use and extend Java Classes, while Java can incorporate classes written in Jython.
- The java environment provides the Java Database Connectivity (JDBC) for standardized cross platform access to most database management systems. Jython further simplifies database access with the zxJDBC module.
- The python scripting language is used by the ArcGIS 9 Geoprocessor. This can reduce developer training costs, as a single language can be used for both ArcGIS automation and ArcIMS web site development.
- The jython interpreter can be used interactively to explore the Java Connector data model. This can be a useful tool for learning and experimenting with the object model, and also for administration of the server.

It is important to note that jython scripts execute slower that native Java classes. This loss in performance, however, does not tend to impact the performance of an ArcIMS application as the expense of rendering a map is significantly more than executing a script. In many cases, using a scripting language will have little to no impact on the performance of an ArcIMS web site. If a script component did, however, become a limiting factor in the performance of a site, it would be a simple matter to re-write the script directly in Java.

The remainder of this paper will focus on the procedure for installing jython, configuring it for use within a java servlet container, and then preparing a simple example to demonstrate the simplicity and strength of the platform. This article assumes that the user has successfully installed a Java Runtime Environment (JRE), ArcIMS, and a servlet container. The examples presented were run with JRE 1.4.2, ArcIMS 4.01, and Jakarta-Tomcat 3.3.1a. They reference an ArcIMS service called `TestService`. Also, the examples are presented for users running Microsoft Windows 2000 or higher operating system.

Obtaining and Installing Jython

Jython is an OpenSource project, therefore can be downloaded from www.jython.org

and installed without charge. The current release implements the Python 2.1 specification and requires a JVM greater than version 1.x. Jython will not run on the JVM version 1.5. There is, however, a new version under development which will be compatible with JVM 1.5. Installation can be a bit tricky, although there is ample information on the web site to troubleshoot most problems. In most cases, to install jython, download the jython21.class file and run the following command:

java -cp . jython-21

This should start a graphical installer which will then prompt the user for additional information. Once jython is installed, the interpreter can be started by executing the "jython.bat" file located in the installation directory. This will start an interactive session of the jython interpreter. Commands can be entered at the command line (prefaced by `>>>`) and are executed when the enter key is pressed. A python script can be executed simply by providing the name of the script as an argument to the jython.bat command. For example, to run a script file titled "sample.py" the command would be "jython.bat sample.py".

Connecting Jython to ArcIMS

To use jython in conjunction with ArcIMS, the ArcIMS Java Connector library must be available. These libraries are located on the ArcIMS CD or in the installation directory of ArcIMS. Within the installation directory they can be found in the "<ArcIMS HOME>\ArcIMS\Middleware\Java_Connector" subdirectory. The libraries can be transferred by copying all of the .jar files in this subdirectory into the "lib\ext" subdirectory of the JRE. For example, if the Java Software Development Kit is installed to c:\j2sdk1.4.2_03, then they should be copied to "c:\j2sdk1.4.2_03\jre\lib\ext\".

Once the libraries have been copied, the jython connection to ArcIMS can be tested. The first step is to import packages from the ESRI Java Connector. Refer to the ESRI Java Documents for additional documentation on the packages in the Java Connector. This example utilizes the `com.esri.aims.mtier.io` package which contains the ConnectionProxy class and the `com.esri.aims.mtier.model.map` package which contains the Map, Legend, and Layers objects. The Map, Legend, and Layers objects define most of the components of an ArcIMS service.

```
>>> from com.esri.aims.mtier.io import *
>>> from com.esri.aims.mtier.model.map import *
```

Now that the packages are imported, the next step is to establish a connection with the ArcIMS Server. For this example, the server IP address is "10.10.8.43", the ArcIMS service is "TestService". The server is listening on the default port of 5300.

```
>>> con=ConnectionProxy()
>>> con.setConnectionType("tcp")
>>> con.setHost("10.10.8.43")
>>> con.setPort(5300)
>>> con.setService("TestService")
```

Once the connection is established, the Map class can be used to request an image from the server. The following sequence of commands creates a new map, sets the width and height to 400 and 600 pixels respectively, initializes the map service, and generates a new image. The image URL is printed after the getURL method.

```
>>> map=Map()
>>> map.setHeight(400)
>>> map.setWidth(600)
>>> map.initMap(con,750,0,0,0,0)
>>> map.refresh()
>>> map.getMapOutput().getURL()
'http://10.10.8.43/output/iwhrs_localhost996014746657.jpg'
```

Classes in Jython
Taking this a step further, a class can be written in jython that subclasses the Java Connector's Map class and connects to a given service when the class is instantiated.

```
from com.esri.aims.mtier.io import *
from com.esri.aims.mtier.model.map import *
class ImsMap(Map):
      def __init__(self,service):
            con=ConnectionProxy()
            con.setConnectionType("tcp")
            con.setHost("10.10.8.43")
            con.setPort(5300)
            con.setService(service)
            self.con=con
      def startMap(self):
            Map.setHeight(self,400)
            Map.setWidth(self,600)
            Map.initMap(self,con,750,0,0,0,0)
            Map.refresh(self)
            return Map.getMapOutput(self).getURL()


>>> map=ImsMap("TestService")
>>> map.startMap()
'http://10.10.8.43/output/iwhrs_localhost996014746663.jpg'
>>> map.getMapUnits()
'feet'
```

Notice that the getMapUnits method was inherited from the Java Connector's Map class. This seamless integration between jython and Java is perhaps its greatest strength. As noted before, Java classes can be directly subclasses by jython, while classes written in Jython can be subclassed by Java. Another advantage is the fact that casting of objects is managed implicitly by the jython interpreter. This helps to keep the jython scripts simple and legible.

Jython and Tomcat
To develop an ArcIMS web application using jython, jython and the ArcIMS connector must be deployed on a servlet container. The following example uses the Apache Jakarta-Tomcat servlet container, which is the reference implementation of the Java Servlet and Java Server Page specifications. There are essentially four steps for deploying jython and the ArcIMS java connector on Jakarta-Tomcat, they are:

1. Create a new context for the web site. This example creates the context `sample` including the following directory structure:

- <path to jakarta>\webapps\sample
- <path to jakarta>\webapps\sample\WEB-INF
- <path to jakarta>\webapps\sample\WEB-INF\lib

2. Copy the jython.jar library module, and the ArcIMS Java Connector jar files into the sample contexts lib subdirectory (<path to jakarta>\webapps\sample\WEB-INF\lib).

3. Create a `web.xml` deployment descriptor file that defines a PyServlet descriptor mapping. At a minimum, the deployment descriptor file should contain:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

      <web-app>
            <servlet>
                  <servlet-name>
                        pyservlet
                  </servlet-name>
                  <servlet-class>
                        org.python.util.PyServlet
                  </servlet-class>
            </servlet>
            <servlet-mapping>
                  <servlet-name>
                        pyservlet
                  </servlet-name>
                  <url-pattern>*.py</url-pattern>
            </servlet-mapping>
      </web-app>
```

4. Write the python servlets. The following sample servlet, GetImsMap.py, should be deployed in the context's root directory (\webapps\sample):

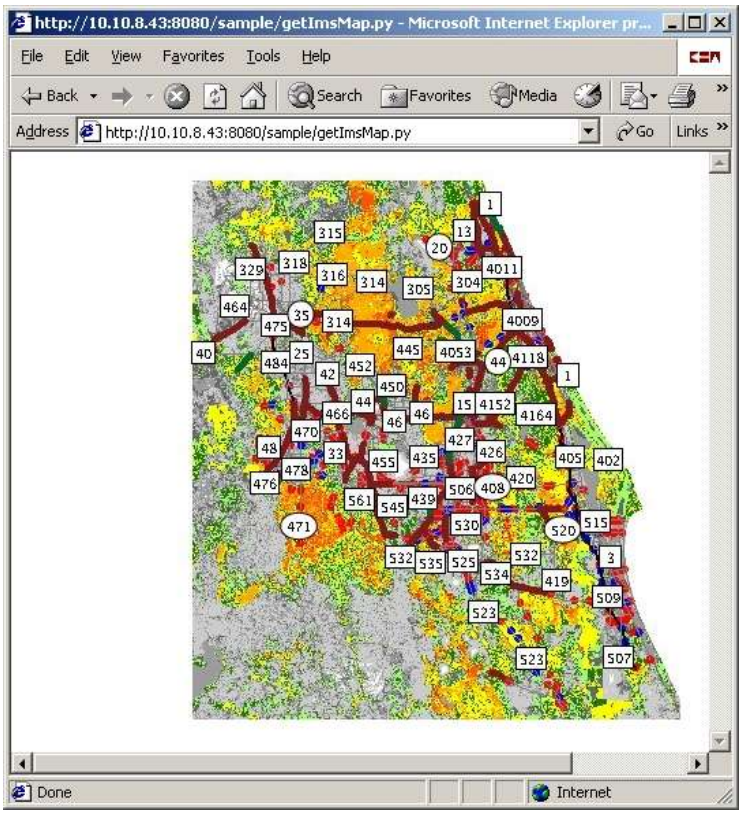
```
from javax.servlet.http import HttpServlet
from com.esri.aims.mtier.io import *
from com.esri.aims.mtier.model.map import *

class GetImsMap(HttpServlet):
      def doGet(self,req,res):
            out=res.writer
            res.contentType="text/html"
            map=ImsMap("sampleService")
            print>>out, "<html><body>"
            print>>out, "<img src="+map.startMap()
            print>>out, "></body></html>"

class ImsMap(Map):
      def __init__(self,service):
            con=ConnectionProxy()
            con.setConnectionType("tcp")
```

```
        con.setHost("10.10.8.43")
        con.setPort(5300)
        con.setService(service)
        self.con=con
def startMap(self):
        Map.setHeight(self,400)
        Map.setWidth(self,600)
        Map.initMap(self,self.con,750,0,0,0,0)
        Map.refresh(self)
        return Map.getMapOutput(self).getURL()
```

To run the servlet that was just created, make sure that the servlet container and ArcIMS server is running, start a browser and enter the url:

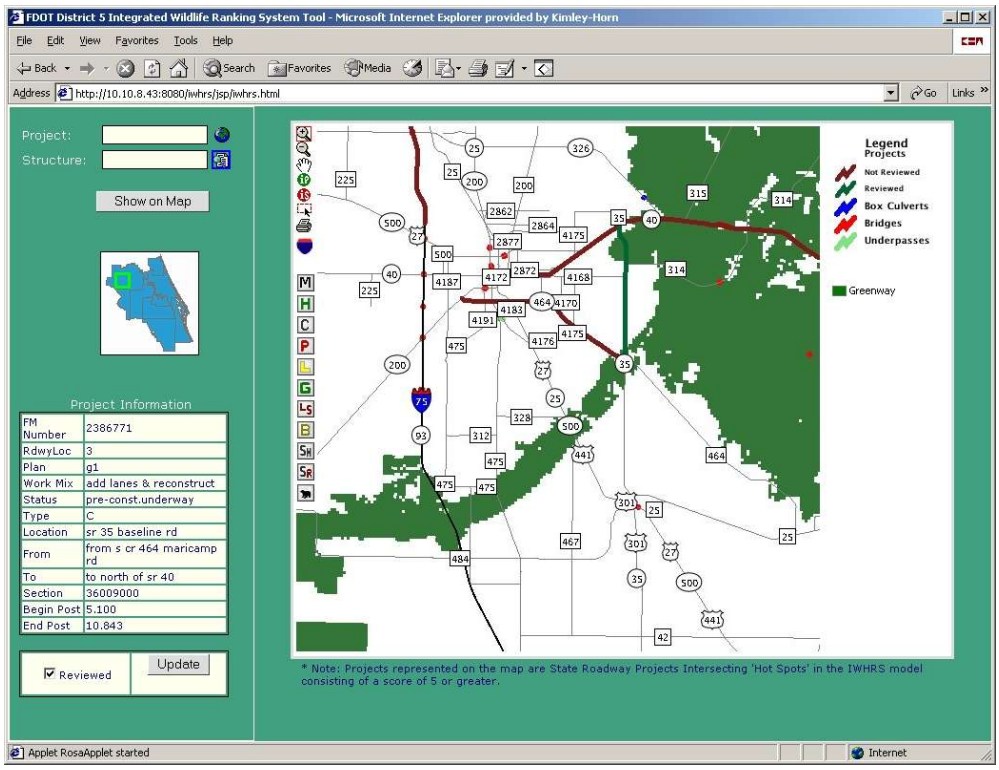
http://<server address>:8080/sample/GetImsMap.py

If everything is working correctly, an image of the ArcIMS service should be visible in the web browser. My sample is depicted below:



We have, with the help of a small java applet (ROSA http://www.maptools.org/rosa/index.phtml), been able to create an extended Java Connector library and basic template that encapsulates most of the required functionality of a GIS client. Deploying a customized ArcIMS client using jython and these tools requires little effort, and is much more amenable to customization than the heavier clients provided by ESRI. An example of this tool, presented below, lets the

user modify the table depicted on the left, and will modify the map symbology as the underlying data is changed. By incorporating our extended classes, the map frame and its functions were developed with less than two hundred lines of python code.



Conclusion

The combined use of jython and java provides developers with an excellent choice for development of ArcIMS web sites. Its simplicity and flexibility make it ideal for prototyping and rapidly deploying customized ArcIMS web sites.

There are many other possibilities regarding the configuration or deployment of jython servlets, such as the creation and deployment of WAR archives, or the incorporation of the standard python libraries in the web site, but these topics are beyond the scope of this paper. Interested readers are encouraged to visit the jython web site. Several technical books have also been published on the use and configuration of jython. Links to these books are also presented on the jython web site.

Randal Goss
Kimley-Horn and Associates, Inc.
8711 Perimeter Park Blvd.
Suite 4
Jacksonville, FL 32216
Phone (904) 998-2084
Fax (904) 998-2197