Implementation of a General Web Application Program Interface for Geoinformatics

Jan Pytel
Department of Mapping and Cartography
Faculty of Civil Engineering, CTU in Prague
E-mail: pytel@fsv.cvut.cz

Keywords: Java language, C++language, servlets, CGI, web applications

Abstract

C++ language was used for creating web applications at the department of Mapping and Cartography for many years. Plenty of projects started to be very large-scale and complicated to maintain. Consequently, the traditional way of adding functionality to a Web Server which previously has been used (CGI programs) started being usefulness. I was looking for some solutions - particularly open source ones. I have tried many languages (solutions) and finally I chose the Java language and started writing servlets. Using the Java language (servlets) has significantly simplified the development of web applications. As a result, developing cycle was cut down. Because of Java JNI (Java Native Interface) it is still possible to use C++ libraries which we are using. The main goal of this article is to share my practical experiences with rewriting typical CGI web application and creating complex geoinformatic web application.

Introduction

The modern era brings new phenomenon: World Wide Web, a term frequently used (incorrectly) when referring to "The Internet". It stands for the universe of hypertext servers (HTTP servers), more commonly called "web servers", which are the servers that serve web pages to web browsers. A plain www page (HTML document) is static, which means that a text file doesn't change - for example: CV, research papers, etc. When someone would like create web pages that contain dynamic content a plain www pages are not sufficient: a solution is to create CGI programs with using languages like PHP, C++, Perl, etc.

PHP or C++ languages were used for creating web applications at the department of Mapping and Cartography for many years, some examples are:

- Internet access to the database of GPS observation via www. The project is written in C++ language:
  http://www.vugtk.cz/gpsdb
- online transformation between ETRF and S-JTSK
  http://gama.fsv.cvut.cz/~kost/vypocty .

Plenty of projects started to be very large-scale and complicated to maintain. Consequently, the traditional way of adding functionality to a Web Server which was used (CGI programs) has become unsustainable. I was looking for alternatives - particularly open source ones. I have tried many languages (solutions) and finally I chose the Java language and started

writing servlets. This paper briefly introduces the servlet concept and explains how to create general web application program interface for geoinformatics.

Common Gateway Interface - CGI

The Common Gateway Interface (CGI) programs are "normal" programs running on the server side — input data for CGI programs are requests from a client (data sent by web browser - HTTP header with other information). Output from CGI program is sent back to the client (web browser). This concept means that client side does not need to take care which type of page is requested. Side dynamic www pages and static www pages are transparent - client sends a request with data and obtains www page. The CGI programs have to be placed in the special directory (usually `/usr/lib/cgi-bin`), the directory where system expects CGI programs.

An example of dummy CGI program `date` (used script language bash) which returns current date in ISO 8601 format `'YEAR-MONTH-DAY'`:

```
#!/bin/bash

echo 'Content-type: text/html'
echo
echo '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">'
echo '<html>'
echo '<title>Current date</title>'
echo '</head>'
echo '<body>'
echo '<h1>Date:</h1>'
echo '<pre>'

/bin/date -I

echo '</pre>'
echo '</body>'
echo '</html>'
```

We simply copy the previous program/script into directory `cgi-bin` and set the file execution permission. No other steps are necessary. Thus I have demonstrated that developing CGI programs is pretty easy. We can test the develop CGI program in terminal now:

```
wget http://localhost/cgi-bin/date -O tmp-date && \
cat tmp-date && rm tmp-date
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">'
<html>
<title>Current date</title>
</head>
<body>
<h1>Date:</h1>
<pre>
2006-06-25
</pre>
</body>
```

```
</html>
```

Another example is CGI a program written in C++ language. The CGI program returns input (data sent by client - without HTTP header). From the following example it will be quite obvious that we have written "normal" C++ program which reads data from input and returns text page:

```
#include<iostream>
#include<string>

int main()
{
  using namespace std;

  string s;

  cout << "Content-type: text/html\n\n";
  cout << "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 3.2//EN]\"> \n";
  cout << "<html>\n";
  cout << "<head>\n";
  cout << "<title>Input data</title>\n";
  cout << "</head>\n";
  cout << "<body>\n";
  cout << "<h1>Input data:</h1>\n";
  cout << "<pre>\n";

  cin  >> s;

  cout << s;

  cout << "</pre>\n";
  cout << "</body>\n";
  cout << "</html>\n;
}
```

CGI programs can be written in many languages, for example PHP, C++, Python, etc. Creating CGI programs is pretty easy, but the CGI concept has some limitations:

- each request is answered in a separate process by a separate instance of CGI program (CGI program needs to be loaded and started for each CGI request)
- use database pooling or interaction between two CGI programs is problematic platform dependence
- lack of scalability

Servlets

A servlet is a Java application that runs within a Web server. Servlets recieve and respond the requests from Web clients. We have to use servlet container in order to run servlets. There are many of server containers available, we have chosen Apache Tomcat server container, discussed in the next section.

On SUN pages (SUN Company is creator of servlets) we can read more precise explanation: "A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes."

Java Servlet API [1] is a class library for servlets. Java Servlet API contains class HttpServlet, which provides methods, such as `doGet` and `doPost` methods for handling HTTP services. In other words: when we would like create new servlet we have to create new class extends `HttpServlet` class and override methods `doGet` and `doPost` - refer to next example.

Servlets have several advantages over CGI:

- servlet does not run in a separate process, stays in memory between requests
- there is only one single instance which answers all requests concurrently - this saves memory and allows a servlet to easily manage persistent data
- platform independence
- Java language has very rich libraries for working with HTTP request, HTTP responses, etc.

An example of first servlet:

```
package cz.examples;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet
{
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException
  {
    res.setContentType("text/html");

    PrintWriter out = res.getWriter();

    out.println("<html>");
    out.println("<head><title>Hello world.</title></head>");
    out.println("<body>Hello world</body>");
    out.println("</html>");

    out.close();
  }

  protected void doPost(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException
  {
    doGet(req,res);
  }
```

```
}
```

When servlet `HelloWorldServlet` is requested a URL inside web browser, sentence "Hello world" will appear. Because there is only a single instance which answers all requests concurrently - it means that we can easily manage persistent data. For example, we would like to know how many times servlet `HelloWorldServlet` was requested:

```
package cz.examples;

import java.io.*;
import javax.servlet.*;

import javax.servlet.http.*;

public class HelloWorldRequestedServlet extends HttpServlet
{
  private int times = 0;

  protected void doGet(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException
  {
    res.setContentType("text/html");

    PrintWriter out = res.getWriter();

    out.println("<html>");
    out.println("<head><title>Hello world.</title></head>");
    out.println("<body>Hello world!<br /> This page was requested " +
                      ++times + " times.</body>");
    out.println("</html>");

    out.close();
  }

  protected void doPost(HttpServletRequest req,
                        HttpServletResponse res)
  throws ServletException, IOException
  {
    doGet(req,res);
  }
}
```

To achieve this behaviour using CGI programs, it would be quite complicated. Java Servlet API contains rich set of useful classes: one of those classes is HttpServletRequest which provides request information for HTTP servlets and contains many useful methods. Next example is servlet `NumericalServlet`, the servlet excepted two parameters `argument1` and `argument2` and returns the following results:

```
package cz.examples;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class NumericalServlet extends HttpServlet
{
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    res.setContentType("text/html");

    String argument1 = req.getParameter("argument1");
    String argument2 = req.getParameter("argument2");

    PrintWriter out = res.getWriter();

   if ( ( (argument1 == null) || (argument2 == null) )
   {
     out.println("<html><body>Error - wrong arguments</body></html>");
     out.close();

     return;
   }

   double arg1 =  Double.parseDouble(argument1);
   double arg2 =  Double.parseDouble(argument2);

   out.println("<html>");
   out.println("<head><title>NumericalServlet</title></head>");
   out.println("<body>");
   out.println("Results:<br />");

   out.println(arg1 + " + " + arg2 + " = " + (arg1 + arg2) + "<br />");
   out.println(arg1 + " - " + arg2 + " = " + (arg1 - arg2) + "<br />");
   out.println(arg1 + " * " + arg2 + " = " + (arg1 * arg2) + "<br />");

   out.println("</html>");
   out.close();
  }

  protected void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    doGet(req,res);
  }
}
```

Java Servlet Container — Apache Tomcat

When working with servlets, we have to use servlet container in order to run servlets. There are many server containers available, we have chosen Apache Tomcat Servlet/JSP container. This container is free of software and is released under Apache Software Licence (http://www.apache.org/licences). First of all, we have to download binary distribution (`jakarta-tomcat-5.0.28.tar.gz`) from the site http://tomcat.apache.org/. It is sup-

posed that we have installed Java Development Kit (JDK) 1.2 or later platform.

Now we have to decide in which directory Tomcat will be located (directory represents the root of Tomcat installation) - common directory is `/opt` where we will extract gzipped tarball of binary distribution:

```
$mv jakarta-tomcat-5.0.28.tar.gz /opt/
$tar xvzf jakarta-tomcat-5.0.28.tar.gz
```

Now it is necessary to modify two files

- `/opt/jakarta-tomcat-5.0.28/conf/server.xml` - we modify an attribute `port` of element `<Server>`. The attribute `port` describes on which port Tomcat will be running
- `/opt/jakarta-tomcat-5.0.28/bin/catalina.sh` - we have to set the `JAVA_HOME` environment variable to tell Tomcat where to find Java:
  `set JAVA_HOME=/opt/jdk1.5.0`

From now Tomcat is prepared for running. Before starting Tomcat (by `/opt/jakarta-tomcat-5.0.28/bin/startup.sh`) we should deploy our applications.

Deploying application

In order to be executed, a web application must be deployed on a servlet container. A web application contains servlets and it is defined as a hierarchy of directories and files in a standard layout. The top-level directory of the web application hierarchy is also the document root of web application. There are plain HTML files at this place. A web application has defined this hierarchy of directories:

- `*.html`, `*.htm` - The HTML pages, along with the other files, that must be visible to the client browser (stylesheet files, and images)
- `/WEB-INF/web.xml` - The Web Application Deployment Descriptor for the application. This is a XML file describing the servlets which make up an application, along with any initialization parameters.
- `/WEB-INF/classes/` - This directory contains any Java class files (and associated resources) required for the application, including both servlet and non-servlet classes.
- `/WEB-INF/lib/` - This directory contains JAR files that consist of Java class files required for an application.

In order to continue with our examples (servlets `HelloWorldServlet`, `HelloWorldRequestedServlet`, `NumericalServlet`) we have to create a web application (named `exampleservlets`):

- create directory `/opt/jakarta-tomcat-5.0.28/webapps/exampleservlets`, in this directory create the above mentioned hierarchy
- compile (with program `javac`) all three servlets
- copy `.class` files into `/WEB-INF/classes/` directory (because the examples are in package `cz.examples`, copy `.class` files into `/WEB-INF/classes/cz/examples/` directory)

- modify `/WEB-INF/web.xml`. As mentioned above, the `/WEB-INF/web.xml` file contains the Web Application Deployment Descriptor for a application. As the filename extension implies, this file is an XML document, and defines everything about the application that a server needs to know. The complete syntax and semantics for the deployment descriptor is defined in [2]. In our case `web.xml` looks like:

```
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
   "http://java.sun.com/dtd/web-app_2_3.dtd">

  <web-app>

   <display-name>Example webapplication</display-name>
   <description>
    This is a simple webapplication for demonstrating.
   </description>

   <servlet>
    <servlet-name>HelloWorldServlet</servlet-name>
    <servlet-class>cz.examples.HelloWorldServlet</servlet-class>
   </servlet>

   <servlet-mapping>
    <servlet-name>HelloWorldServlet</servlet-name>
    <url-pattern>/HelloWorldServlet</url-pattern>
   </servlet-mapping>

   <servlet>
    <servlet-name>HelloWorldRequestedServlet</servlet-name>
    <servlet-class>cz.examples.HelloWorldRequestedServlet</servlet-class>
   </servlet>

   <servlet-mapping>
    <servlet-name>HelloWorldRequestedServlet</servlet-name>
    <url-pattern>/HelloWorldRequestedServlet</url-pattern>
   </servlet-mapping>

   <servlet>
    <servlet-name>NumericalServlet</servlet-name>
    <servlet-class>cz.examples.NumericalServlet</servlet-class>
   </servlet>

   <servlet-mapping>
    <servlet-name>NumericalServlet</servlet-name>
    <url-pattern>/NumericalServlet</url-pattern>
   </servlet-mapping>

  </web-app>
```

Now everything needed for running the web application is done. After starting up Tomcat, we can use web browser and test our examples. We can find the servlets on the following addresses (port means port where tomcat is running):

http://localhost:port/exampleservlets/NumericalServlet,
http://localhost:port/exampleservlets/HelloWorldServlet,
http://localhost:port/exampleservlets/HelloWorldRequestedServlet.

General Web Application Program Interface for Geoinformatics

Requirements for used technology

During the development general web application program interface for geoinformatics were required following technology features:

- the development under OS GNU/Linux with using free software [3]
- platform independency
- ability to use effectively existing source codes (most of them in C++ language) already finished projects
- used MVC paradigm
- database access, database connection pooling
- using of sessions and session management

Java language and technology of servlets were chosen for development of the project. Servlet technology satisfies all our requirements.

This chapter contains all used technologies and share experiences with creating general web application program interface for geoinformatics (application with using servlet technology and MVC paradigm).

Used design patterns

Interface is fully object-oriented and using several design patterns (a design pattern is a general repeatable solution to a commonly-occurring problem in software design). For example we used patterns *Singleton*, *AbstractFactory*, *Observer* and *Facade*.

One of the main goals was selection of object oriented model, which would separate computing core from presentation part (the way how the results will be displayed for the user on the screen). The programmer of computing core usually does not care for the layout of input and output data. The programmer usually only describe the expected data on the input side together with description of the results returned by program. Ideal solution turned to be MVC software design pattern, i.e. design pattern on which this system is based on.

MVC paradigm is a way of breaking application into three parts [4]:

Model - represents of the information on which application operates (e.g. describe of database system, computing model of geodetic tasks, ...)

View - renders model into a form suitable for users, typically a user interface element. MVC is often seen in web application where the view is the HTML page and the code which gathers dynamic data for the page

Controller – responds to events, typically user actions, and invokes changes on the model and perhaps the view.

Template engine Velocity

The selection of template engine for component View was essential from the point of view of the creation application. The complete separation of the Model, View respectively from the presentation part was the key requirement on the template engine. The developer only describes required input and output data (specifies names of variables representing input and results data – e.g. collections, strings, etc.).

Developers have chosen template engine Velocity:
http://jakarta.apache.org/velocity .

Template engine Velocity is one of the parts of the Jakarta's project developed by Apache software foundation and is released under the licence Apache Licence. The web pages of the project contain the following:

The Velocity User Guide is intended to help page designers and content providers get acquainted with Velocity and the syntax of its simple yet powerful scripting language, the Velocity Template Language (VTL). Many of the examples in this guide deal with using Velocity to embed dynamic content in web sites, but all VTL examples are equally applicable to other pages and templates.

Velocity is a Java-based template engine. It permits web page designers to reference methods defined in Java code. Web designers can work in parallel with Java programmers to develop web sites according to the Model-View-Controller (MVC) model, meaning that web page designers can focus solely on creating a well-designed site, and programmers can focus solely on writing top-notch code.

It is necessary to stick to the following pattern when using Velocity:

- initialization
- creation of object context
- fulfillment of the Velocity context by data
- selection of template
- merging context and template into output file

Velocity Context represents variable part of the page. From the programmer's standpoint, Velocity context is a map of Objects. Velocity Template Language (VLT) is very simple; for the description of the template language refer to:
http://jakarta.apache.org/velocity/docs/user-guide.html

.

The following text contains the extracts from the template language:

```
<html>
<body>
#set( $foo = "Velocity" )
Hello $foo World!
</body>
<html>

#if( $value < 0 )
 <strong>value $value is negative</strong>
#elseif( $value  == 0 )
  <strong>value $value is equal 0</strong>
#else
 <strong>value $value is positive</strong>
#end

<p>
#foreach( $student in $university )
$student.nickname $student.surname - $student.birthday<br />
 #end
 </p>
```

Computing task — from developer's standpoint

General web application program interface for geoinformatics allows adding arbitrary computing tasks. Those tasks are accessible for the users using well-arranged menu. Currently, it is possible to add the following three types of tasks into the system:

- tasks written in Java language, implementing interface `ComputingTask`
- tasks written in arbitrary programming language, distributed as standalone executable programs – the system allows execution of those programs
- tasks written in C++ language.

The task written in Java language is only class implementing interface `ComputingTask`:

```
public interface ComputingTask {
void setParameters(Map<String, String> input);
Map  getResults();
            OutputStream getResultStream();
             void compute();
             boolean wasComputed();
}
```

Input data for the computing task are stored in `Map` collection. The key representing name of the variable input has `String` type. As a result, it is necessary explicitly retype data to appropriate data types. The results may be returned in collection `Map`, or in the class `OutputStream`.

The similar approach used in Java language is applied also for the tasks written in C++ this case, JNI code is used. Final application is called Manala and can be found on the following

web page http://gama.fsv.cvut.cz/manala.

Conclusion

The Java language (servlets) has significantly simplified the development of web applications. As a result, developing cycle was cut down. Because of Java JNI (Java Native Interface) it is still possible to use C++ libraries which are done. We have started using the Java language particularly for web applications in 2005. We have rewritten many of our applications. Switch development from "typical CGI programming" to "Java servlet programming" is surprisingly easy with amazing benefit.

Development of General web application program interface for geoinformatics has brought completely new requirements on the web application program development. The current technology for development www applications used by the department of Mapping and Cartography turned to be insufficient. The development has been significantly improved by using servlet technology and framework MVC. As a result we developed application program interface for geoinformatics Manala.

References

1. Java Servlet API, http://java.sun.com/products/servlet/index.html
2. http://tomcat.apache.org/tomcat-5.5-doc/appdev/deployment.html
3. http://www.gnu.org
4. http://en.wikipedia.org/wiki/Model_view_controller
5. Common Gateway Interface, http://hoohoo.ncsa.uiuc.edu/cgi/intro.html
6. Apache Tomcat Tutorial, http://www.coreservlets.com/Apache-Tomcat-Tutorial
7. Servlet Essentias, http://www.novocode.com/doc/servlet-essentials/
8. Apache Tomcat Servlet Container, http://jakarta.apache.org