ARQUITECTURA BASADA EN QoS PARA EL DESCUBRIMIENTO DE SERVICIOS WEB GEOGRÁFICOS

ANEXOS

Rafael Alberto Carrascal Reyes
Edwin Andrés Pachajoa Rodríguez

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática

Popayán
2008

TABLA DE CONTENIDO

A. ANEXO A. ANÁLISIS DE REQUISITOS

A.1 DESCRIPCIÓN DE LA APLICACIÓN.

Se necesita proponer y validar la arquitectura para un catálogo de recursos geocodificados que facilite el descubrimiento de servicios y datos hidrológicos. El sistema de catálogo debe mantener la información necesaria y suficiente de los recursos para garantizar búsquedas contextualizadas de acuerdo a la zona geográfica así como a su desempeño en términos de la calidad de servicio.

La arquitectura debe soportar interacciones con otros sistemas además de permitir búsquedas directas de personas interesadas en los servicios. Los usuarios se clasifican como proveedor de servicios y consumidor o usuario de servicios, el proveedor es responsable de crear y mantener los servicios y su descripción correspondiente, el consumidor es la persona o sistema que utiliza el servicio y es responsable por crear o mantener las interfaces programáticas (en caso de que se requieran) para acceder el servicio.

El sistema de catalogo debe proveer funciones de registro a los usuarios proveedores, que incluya los datos de la organización que presta el servicio, la persona responsable y las características del servicio como zona geográfica, aplicación, propósito, calidad, etc. Los tipos de recursos se describen mediante metadatos que relacionan datos geográficos, imágenes y servicios de manera generalizada

También debe proveer funciones de búsqueda a los usuarios consumidores, estas se realizan de acuerdo a un criterio de búsqueda que es la palabra clave con la cual se identifica el servicio, y un criterio de calidad deseable. La descripción de servicios Web hídricos puede ser discriminada en grupos de propiedades que entreguen información a niveles distintos de especificidad solicitada como reportes cortos, resúmenes o reportes completos. De la misma forma, las búsquedas de servicios pueden contener atributos opcionales y obligatorios. En general los servicios hídricos deben ser descriptivos y

simples para que un científico los utilice, esto quiere decir que la solicitud y respuesta deben ser entendibles, y finalmente los parámetros de respuesta deben servir de entrada para aplicaciones desktop como aplicaciones CAD, hojas de cálculo o paquetes matemáticos.

A.2 MODELO DE REQUERIMIENTOS.

A.2.1 Búsqueda y Registro de Servicios

Titulo	Búsqueda y registro de servicios
Descripción	Se requiere buscar servicios de manera simple. Los recursos se describen mediante metadatos, y deben incluirse en la operación de registro.
Tipo	Funcional
Prioridad	Alta
Propósito y Actividades	Método de búsquedas y registro definido.

A.2.2 Gestión de servicios para hidrología

Titulo	Gestión de servicios para hidrología
Descripción	Es deseable que los servicios hidrológicos cuenten con un proceso de gestión establecido y estándar
Tipo	No Funcional
Prioridad	Alta
Propósito y Actividades	Definir la arquitectura y el proceso de gestión

A.2.3 Gestión de Calidad de Servicio

Titulo	Gestión de Calidad de Servicio
Descripción	
Tipo	Funcional
Prioridad	Alta
Propósito y Actividades	Características de calidad de servicio establecidas. Método de manejo de calidad definido.

A.2.4 Gestión de contexto

Titulo	Gestión de contexto
Descripción	
Tipo	Funcional
Prioridad	Alta
Propósito y Actividades	Contexto de servicios definido para Colombia. Método de manejo de contexto definido.

A.2.5 Soporte de búsqueda y registro automático

Titulo	Soporte de búsqueda y registro automático
Descripción	
Tipo	Funcional
Prioridad	Baja
Propósito y Actividades	Método de búsqueda y registro automático establecido

B. ANEXO B. CATÁLOGO DE SERVICIOS WEB CON JUDDI

B.1 INTRODUCCIÓN.

jUDDI es una implementación Java del protocolo Universal de Descripción, Descubrimiento e Integración para Servicios Web (UDDI), proyecto realizado por el Apache Software Foundation (http://ws.apache.org/juddi/).

B.2 OBJETIVOS.

- Identificar las partes que componen un registro XML.
- Publicar Organizaciones en el registro mediante las interfaces provistas por jUDDI.
- Buscar los registros.
- Aprender a utilizar Categorizaciones.

B.2.1 Paquetes Necesarios.

- Motor de base de datos PostgreSQL (http://www.postgresql.org/download/)
- Tomcat Web Container de Apache (http://tomcat.apache.org/) .Preferible la versión 5+.
- Eclipse 3.x

B.3 DESARROLLO.

B.3.1 Instalación del motor de la base de datos.

- Descomprima el archivo de Postgres y siga las instrucciones para instalar la Base de datos.

B.3.2 Instalación del catálogo.

- Instale el servidor Web *Tomcat* de la Apache Software Foundation (http://tomcat.apache.org/).
- Descargue la distribución de jUDDI del sitio Web http://ws.apache.org/juddi/releases.html. El contenido del paquete jUDDI se muestra en la Figura B-1.

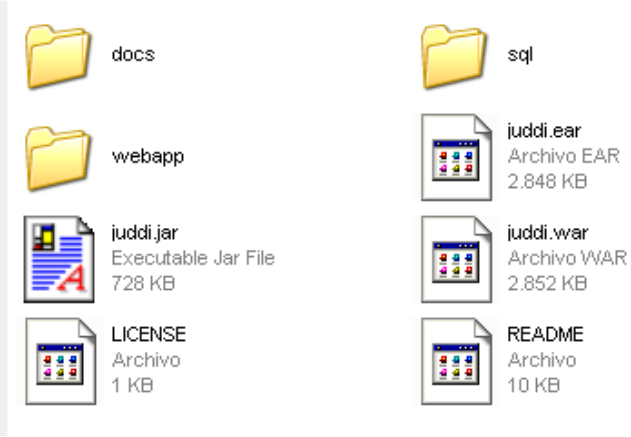


Figura B-1 Contenido de la Distribución de jUDDI

- Copie la carpeta webapp/*jUDDI* de la distribución en la carpeta *webapps* de *Tomcat.*
- Observe el resto de archivos que vienen con la distribución, en especial la carpeta *sql,* la cual trae los scripts de la base de datos del registro para diversos motores de bases de datos entre ellos: mysql, firebird, Oracle y postgres.
- Corra el motor de la base de datos (si la base de datos está instalada como servicio, omita este paso).
- Abra la aplicación PgAdmin que viene con la distribución de Postgres, y cree una nueva base de datos.
- En la base de datos, de clic en el botón SQL. (Forma de lápiz mostrada en la Figura B-2)

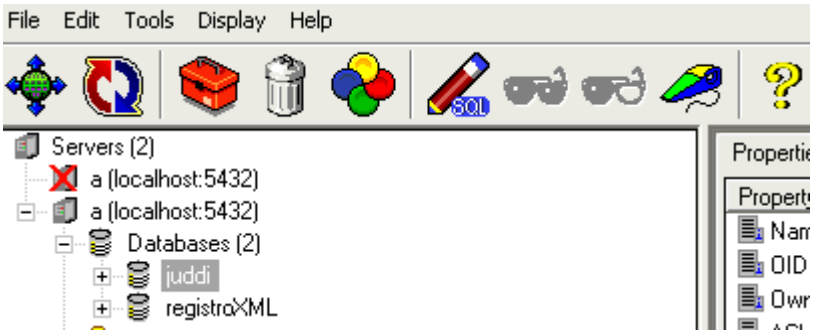


Figura B-2 Interfaz de PgAdmin

- En la interfaz que se despliega, abra el *script* de la base de datos que vimos anteriormente correspondiente a *postgres*, llamado *create_database.sql*. Figura B-3.

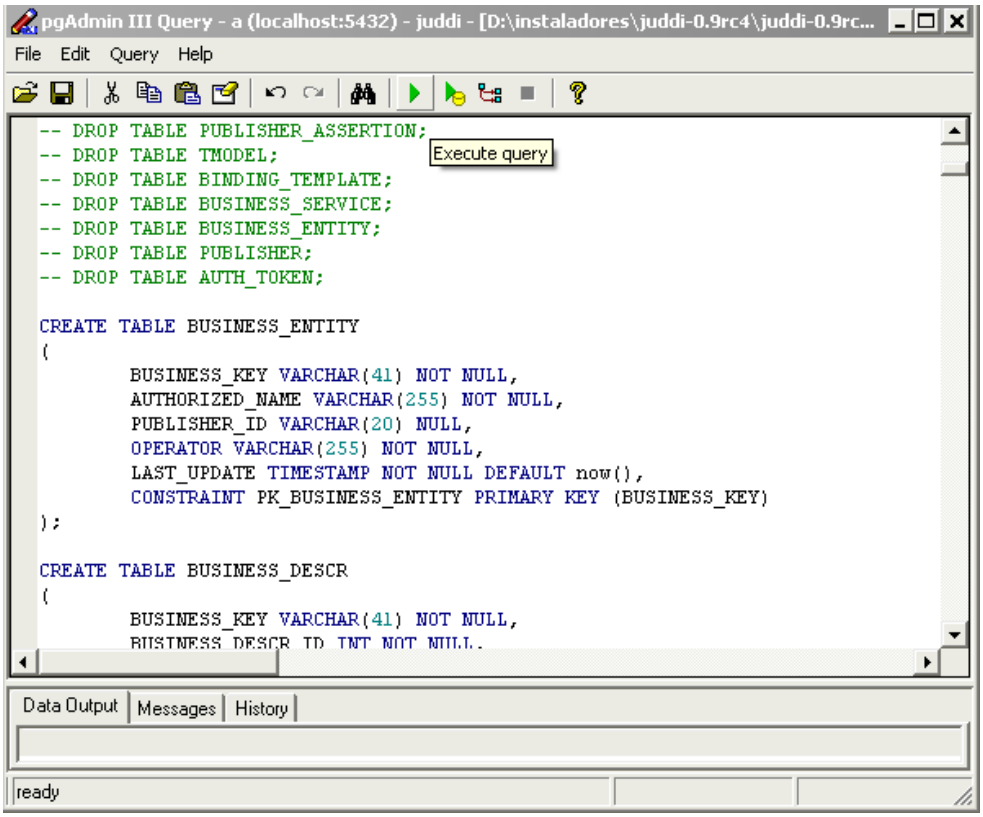


Figura B-3 Interfaz en pgAdmin para crear la base de datos.

- Ejecute el script, note como se crean las tablas del registro entre ellas: Business Entity, Business Service, tModel y demás tablas de la implementación jUDDI.
- Repita el procedimiento anterior con el script *insert_publishers.sql* ingresando sus datos de usuario.

B.3.3 Configuración de Tomcat.

- Para referenciar la base de datos desde Tomcat 5, edite el archivo *server.xml* en la carpeta *conf* de Tomcat.
- Ingrese el siguiente tag antes de cerrar el tag <*Host*> del archivo *server.xml*

```
<Context path="/juddi" docBase="juddi" debug="5" reloadable="true"
crossContext="true">
<Resource name="jdbc/juddiDB" auth="Container"
type="javax.sql.DataSource"
username="*****" password="*****" driverClassName="org.postgresql.Driver"
url="jdbc:postgresql://localhost:3306/*****?autoReconnect=true"
validationQuery="select count(*) from PUBLISHER" />
</Context>
```

- Edite lo que está con asteriscos apropiadamente con el Usuario, password y nombre de la base de datos creada anteriormente.

B.3.4 Publicar una organización (entidad de negocio) en el catalogo.

- Corra el servidor Tomcat.
- Desde el navegador ingrese a http://localhost:8080/juddi/console/
- Si todo va bien aparecerá algo como en la Figura B-4:

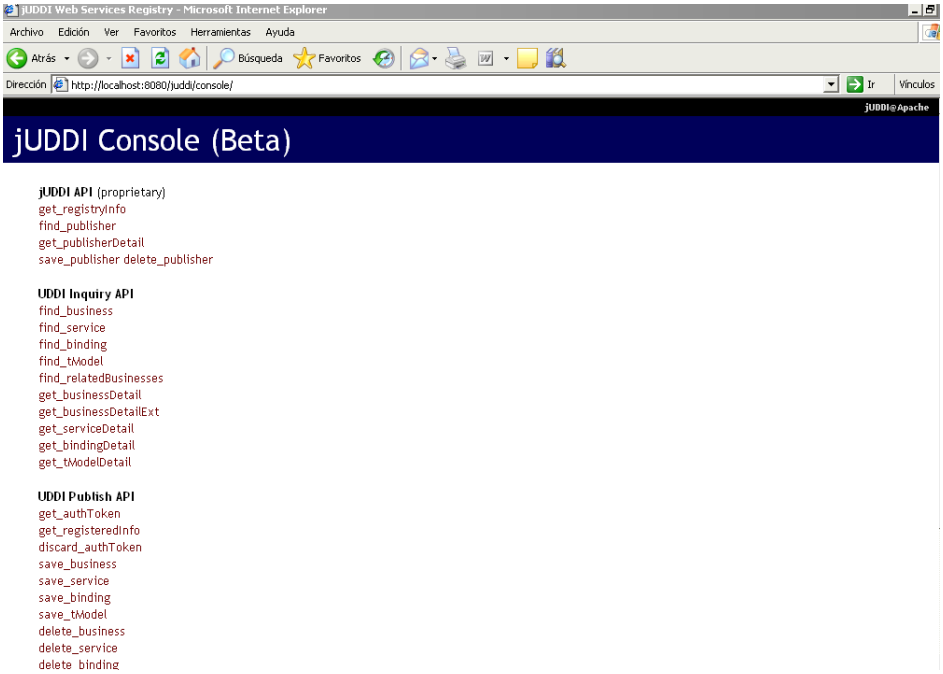


Figura B-4 Publicación en jUDDI

- Pruebe la conexión a la db del registro con las funciones del Publisher, por ejemplo la función *get_publisherDetail* (Figura B-5). Rellene la información de la interfaz con el nombre del usuario que ingresó al catálogo cuando este fue creado y luego oprima el botón *submit*. En las interfaces puede verse como se utilizan estructuras XML del protocolo SOAP para consultar el catálogo.

Figura B-5 Las interfaces jUDDI de gestión y publicación.

- Para publicar un negocio, un servicio y su tModel, requerimos un Authentication Token provisto por la implementación jUDDI. Esta autorización es temporal y solo puede ser solicitada por un usuario (Publisher) registrado en el catalogo. Ir a la interfaz *get_authToken* (Figura B-6), y rellenar con la información solicitada: userID y cred (credenciales).

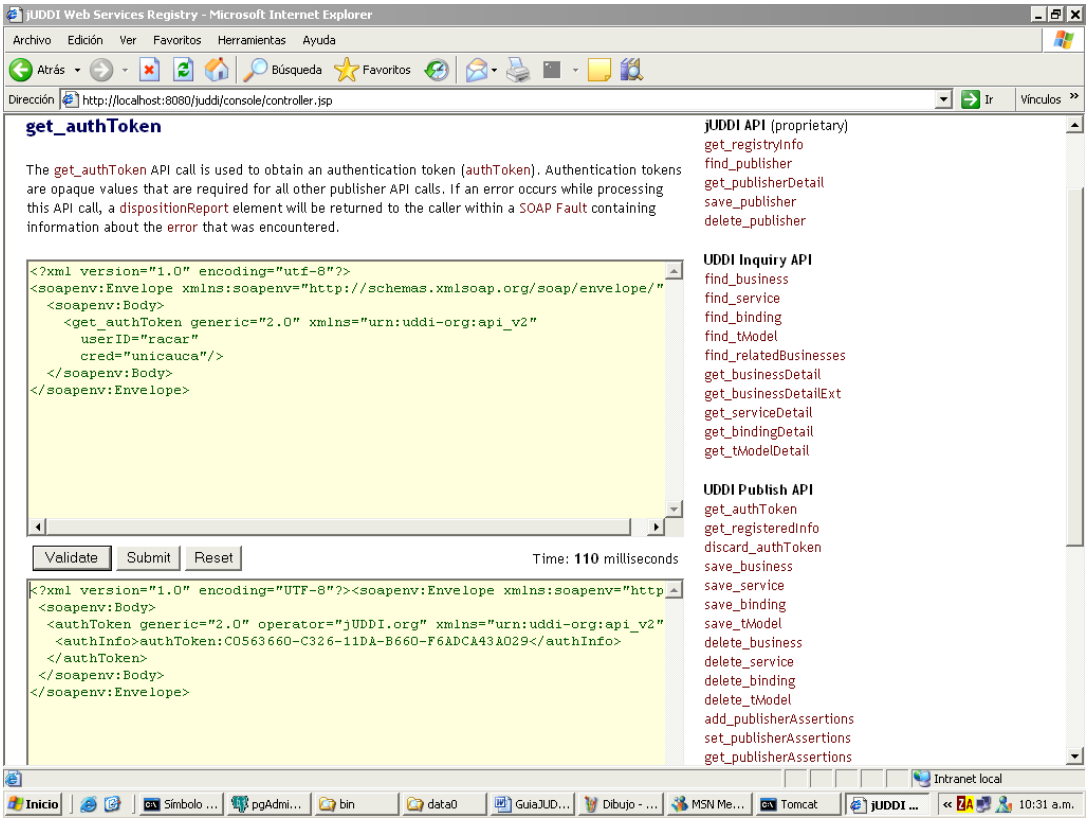


Figura B-6 Interfaz *get_authToken*

- Algo como esto es la autorización para usar el catálogo:

authToken:C0563660-C326-11DA-B660-F6ADCA43A029

Cópielo de la interfaz.

- Ahora vamos a publicar un nuevo negocio, con la interfaz *save_business,* Ingresar, por ejemplo:

authInfo*: authToken:C0563660-C326-11DA-B660-F6ADCA43A029*

Name: Universidad del Cauca

Description: Institución de educación superior.

PesonName: Armando

Phone: 8209800

Email: jaordonez@unicauca.edu.co

- El token de autorización es requerido, de lo contrario, no se podrá registrar.
- Si todo está en orden, podrá verificarse el ingreso de un nuevo Bussines por la interfaz del PgAdmin en la tabla *business_name.*

B.3.5 Buscar en el catalogo desde una aplicación externa.

- En Eclipse crear un proyecto *Cliente_juddi.*
- Cargue las librerías disponibles en *\juddi\WEB-INF\lib*
- Cree la clase *FindBusinesses.java* y copie el siguiente código.

```
import org.apache.juddi.datatype.response.*;
import org.apache.juddi.datatype.Name;
import org.apache.juddi.error.RegistryException;
import org.apache.juddi.proxy.RegistryProxy;
import org.apache.juddi.IRegistry;
import java.util.Vector;
import java.util.Properties;


public class FindBusinesses
{
   private IRegistry registry = null;
   private Vector businesses = null;

   private IRegistry getRegistry()
   {
      if (registry == null)
      {
         Properties props = new Properties();
         props.setProperty("juddi.proxy.adminURL",
                           "http://127.0.0.1:8080/juddi/admin");
         props.setProperty("juddi.proxy.inquiryURL",
                           "http://127.0.0.1:8080/juddi/inquiry");
         props.setProperty("juddi.proxy.publishURL",
                           "http://127.0.0.1:8080/juddi/publish");
         props.setProperty("juddi.proxy.securityProvider",
                           "com.sun.net.ssl.internal.ssl.Provider");
         props.setProperty("juddi.proxy.protocolHandler",
                           "com.sun.net.ssl.internal.www.protocol");
         registry = new RegistryProxy(props);
      }
      return registry;
   }

   public Vector find(String Bname)
   {
      IRegistry registry = getRegistry();

      try
      {
         Vector inNames = new Vector();
         inNames.add(new Name(Bname));
         BusinessList list =
```

```
            registry.findBusiness(inNames,
                                  null,
                                  null,
                                  null,
                                  null,
                                  null,
                                  100);
          // registry.
           BusinessInfos infos = list.getBusinessInfos();


           businesses = infos.getBusinessInfoVector();


        }
        catch (RegistryException regex)
        {
           DispositionReport dispReport =
              regex.getDispositionReport();
           if (dispReport != null)
           {
              Vector resultVector = dispReport.getResultVector();
              if (resultVector != null)
              {
                 Result result = (Result)resultVector.elementAt(0);
                 int errNo = result.getErrno();
                 System.err.println("Errno: " + errNo);

                 ErrInfo errInfo = result.getErrInfo();
                 if (errInfo != null)
                 {
                    System.err.println("ErrCode: " +
                                       errInfo.getErrCode());
                    System.err.println("ErrMsg: " +
                                       errInfo.getErrMsg());
                 }
              }
           }
           else
           {
              System.err.println("Exception: " + regex);
           }
        }

     return businesses;
    }
}
```

- Cree una clase con el nombre *GetAuthToken.java* y copie el siguiente código.

```
import org.apache.juddi.datatype.response.ErrInfo;
import org.apache.juddi.datatype.response.Result;
import org.apache.juddi.datatype.response.DispositionReport;
import org.apache.juddi.datatype.response.AuthToken;
```

```
import org.apache.juddi.error.RegistryException;
import org.apache.juddi.proxy.RegistryProxy;
import org.apache.juddi.IRegistry;

import java.util.Vector;
import java.util.Properties;


public class GetAuthToken
{
   private IRegistry registry = null;

   private IRegistry getRegistry()
   {
      if (registry == null)
      {
         Properties props = new Properties();
         props.setProperty("juddi.proxy.adminURL",
            "http://127.0.0.1:8080/juddi/admin");
         props.setProperty("juddi.proxy.inquiryURL",
            "http://127.0.0.1:8080/juddi/inquiry");
         props.setProperty("juddi.proxy.publishURL",
            "http://127.0.0.1:8080/juddi/publish");
         props.setProperty(
             "juddi.proxy.securityProvider",
             "com.sun.net.ssl.internal.ssl.Provider");
         props.setProperty("juddi.proxy.protocolHandler",
             "com.sun.net.ssl.internal.www.protocol");
         registry = new RegistryProxy(props);
      }
      return registry;
   }

   public AuthToken getToken()
   {
      IRegistry registry = getRegistry();

      String userID = "******";
      String password = "******";

      try
      {
         AuthToken authToken =
            registry.getAuthToken(userID, password);
         return authToken;
      }
      catch (RegistryException regex)
      {
         DispositionReport dispReport =
            regex.getDispositionReport();
         if (dispReport != null)
         {
            Vector resultVector =
               dispReport.getResultVector();
            if (resultVector != null)
            {
```

```
            Result result =
               (Result)resultVector.elementAt(0);
            int errNo = result.getErrno();
            System.err.println("Errno: " + errNo);

            ErrInfo errInfo = result.getErrInfo();
            if (errInfo != null)
            {
               System.err.println("ErrCode: " +
                                  errInfo.getErrCode());
               System.err.println("ErrMsg: " +
                                  errInfo.getErrMsg());
            }
         }
      }
      else
      {
         System.err.println("Exception: " + regex);
      }
   }

   return null;
}

public void discardToken(AuthToken token)
{
   IRegistry registry = getRegistry();

   try
   {
      registry.discardAuthToken(token.getAuthInfo());
   }
   catch (RegistryException regex)
   {
      DispositionReport dispReport =
         regex.getDispositionReport();
      Vector resultVector =
         dispReport.getResultVector();
      if (resultVector != null)
      {
         Result result =
            (Result)resultVector.elementAt(0);
         int errNo = result.getErrno();
         System.err.println("Errno: " + errNo);

         ErrInfo errInfo = result.getErrInfo();
         if (errInfo != null)
         {
            System.err.println("ErrCode: " +
               errInfo.getErrCode());
            System.err.println("ErrMsg: " +
               errInfo.getErrMsg());
         }
      }
   }
}
```

```
}
```

- Cree una clase con el nombre *Test.java* y copie el siguiente código.

```
import org.apache.juddi.datatype.Description;
import org.apache.juddi.datatype.response.AuthToken;
import org.apache.juddi.datatype.response.BusinessInfo;
import org.apache.juddi.datatype.Name;

import java.util.Enumeration;
import java.util.Vector;



public class Test
{
   public Vector bus;
   public Enumeration enum;
   public Enumeration enum1;
   public String Bname;
   public static void main(String[] args)
   {
      Test app = new Test();
   }

   public Test()
   {
     Bname="U"; /*Nombre de la empresa o institución a buscar en el
registro*/
      GetAuthToken getAuthToken = new GetAuthToken();
      AuthToken token = getAuthToken.getToken();
      if (token == null)
      {
         System.err.println("Unable to obtain authToken");
         return;
      }

      FindBusinesses findBusinesses = new FindBusinesses();
      bus = findBusinesses.find(Bname);
      BusinessInfo info=null;
      if (bus != null){
      for (int i = 0; i < bus.size(); i++)
      {
          info =
            (BusinessInfo)bus.elementAt(i);
         Vector outNames = info.getNameVector();
         Vector outDesc = info.getDescriptionVector();
         enum =outNames.elements();
         enum1=outDesc.elements();
      }


      while (enum.hasMoreElements())
      {
         Name businessName = (Name) enum.nextElement();
```

```
            Description des = (Description)enum1.nextElement();

            System.out.println("Encontrado en el registro: " +
businessName.getValue());
            System.out.println("Descripción: " + des.getValue());

        }
        }else{
             System.out.println("La empresa (bussines entity) no se
encuentra registrada");
        }

        getAuthToken.discardToken(token);
    }
}
```

- Corra la clase Test.java

B.4 CATEGORIZACIONES EN jUDDI.

Una de las características más poderosas de UDDI es el uso de categorizaciones, que permite describir en forma precisa los registros de servicios. Es la forma en que UDDI permite agregar metadatos a los servicios.

La aplicación trae por defecto dos grupos de categorizaciones aceptadas internacionalmente: Los códigos de productos y servicios de Naciones Unidas (United Nations Standard Products and Services Code® , UNSPSC®) y el sistema de clasificación de la industria norteamericana (North American Industry Classification System, NAICS). Es Posible agregar categorías de clasificación propias, sin embargo es recomendable no hacerlo a menos que sea un consenso aceptado por los usuarios del catalogo.

Para nuestro ejemplo tomemos una clasificación NAICS para agregar a la entidad de negocio creada, en nuestro caso es una institución universitaria (http://www.census.gov/epcd/naics02/def/ND611310.HTM#N611310):

📖 611310: Colleges, Universities, and Professional Schools

Vamos a ingresar un Nuevo registro, esta vez asignándole esta categoría, ingrese a la consola de juddi (http://localhost:8080/juddi/console/)

Nuevamente escogemos la opción, save_business y agregamos un mensaje asi:
(recuerde solicitar un token de autorización antes)

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <save_business generic="2.0" xmlns="urn:uddi-org:api_v2">
      <authInfo>authToken:A988FC10-E792-11DB-AACA-D060FA5F262A</authInfo>
      <businessEntity businessKey="">
        <name>Universidad del Cauca</name>
        <categoryBag>
        <keyedReference
            tModelKey="uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2"
            keyName="Colleges, Universities, and Professional Schools"
            keyValue="611310"/>
         </categoryBag>
      </businessEntity>
    </save_business>
  </soapenv:Body>
</soapenv:Envelope>
```

Ya hemos asignado la categoría a nuestra institución. Verificamos; con el siguiente mensaje debe aparecer el/los registro/s asociados con la categoría especificada en una lista.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <find_business maxRows="100" generic="2.0" xmlns="urn:uddi-org:api_v2">

      <categoryBag>
        <keyedReference    tModelKey="uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2"
keyValue="611310" />
      </categoryBag>

    </find_business>
  </soapenv:Body>
</soapenv:Envelope>
```

Para realizar búsquedas desde una aplicación externa, utilizando categorizaciones remplazamos la clase *FindBusinesses.java* por:

```
import org.apache.juddi.datatype.response.*;
import org.apache.juddi.datatype.CategoryBag;
import org.apache.juddi.datatype.KeyedReference;
import org.apache.juddi.datatype.Name;
import org.apache.juddi.error.RegistryException;
import org.apache.juddi.proxy.RegistryProxy;
import org.apache.juddi.IRegistry;
import java.util.Vector;
import java.util.Properties;


public class FindBusinesses
{
   private IRegistry registry = null;
   private Vector businesses = null;
   private CategoryBag catBag = null;
   private KeyedReference keyRef = null;

   private IRegistry getRegistry()
   {
   if (registry == null)
      {
         Properties props = new Properties();
         props.setProperty("juddi.proxy.adminURL",
                           "http://127.0.0.1:8080/juddi/admin");
         props.setProperty("juddi.proxy.inquiryURL",
                           "http://127.0.0.1:8080/juddi/inquiry");
         props.setProperty("juddi.proxy.publishURL",
                           "http://127.0.0.1:8080/juddi/publish");
         props.setProperty("juddi.proxy.securityProvider",
                           "com.sun.net.ssl.internal.ssl.Provider");
         props.setProperty("juddi.proxy.protocolHandler",
                           "com.sun.net.ssl.internal.www.protocol");
         registry = new RegistryProxy(props);
      }
      return registry;
   }

   public Vector find(String Bname)
   {
      IRegistry registry = getRegistry();
      catBag=new CategoryBag();
      keyRef=new KeyedReference();
      keyRef.setTModelKey("uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2");
      keyRef.setKeyValue("611310");
      catBag.addKeyedReference(keyRef);

      try
      {
```

```
         Vector inNames = new Vector();
         inNames.add(new Name(Bname));

         BusinessList list =
            registry.findBusiness(null,
                                  null,
                                  null,
                                  catBag,//category Bag
                                  null,
                                  null,
                                  100);
        //registry.
         BusinessInfos infos = list.getBusinessInfos();


         businesses = infos.getBusinessInfoVector();


      }
      catch (RegistryException regex)
      {
         DispositionReport dispReport =
            regex.getDispositionReport();
         if (dispReport != null)
         {
            Vector resultVector = dispReport.getResultVector();
            if (resultVector != null)
            {
               Result result = (Result)resultVector.elementAt(0);
               int errNo = result.getErrno();
               System.err.println("Errno: " + errNo);

               ErrInfo errInfo = result.getErrInfo();
               if (errInfo != null)
               {
                  System.err.println("ErrCode: " +
                                     errInfo.getErrCode());
                  System.err.println("ErrMsg: " +
                                     errInfo.getErrMsg());
               }
            }
         }
         else
         {
            System.err.println("Exception: " + regex);
         }
      }

    return businesses;
   }
}
```

En este caso ya no buscamos por el nombre de la organización sino una categoría los que nos entrega una lista de entidades de negocio que cumplen con el criterio de búsqueda (en nuestro caso es una sola organización).

Corra nuevamente la clase *Test.java.* Ingrese más registros dentro de la categoría o cree nuevas categorías en el registro si lo desea.

De forma similar es posible registrar, buscar o categorizar servicios, modelos T, bindings, y demás estructuras UDDI de acuerdo a lo que necesitemos hacer.

C. ANEXO C. PRÁCTICA MAPSERVER

C.1 OBJETIVOS.

C.1.1 Objetivo General.

Tener conocimiento sobre la utilización del servidor de mapas MapServer en su versión 2.2.6 y su aplicación en los Sistemas de Información Geográfica.

C.1.2 Objetivos Específicos.

- Conocer la estructura del servidor y su configuración.
- Conocimiento del archivo de configuración MapFile.
- Introducción a los entornos con mapas georreferenciados.
- Conocer las capacidades de búsqueda que brinda el servidor.

C.2 INTRODUCCIÓN.

Para la presente práctica se utiliza la versión de MapServer 2.2.6 que se puede descargar desde la url: http://mapserver.gis.umn.edu/download, éste es un paquete comprimido recomendado para el entorno Windows el cual incluye: Apache, PHP, MapServer CGI y MapScript. A continuación se empieza con la instalación del paquete MS4W.

C.3 INSTALACIÓN MS4W.

Una vez descargado el archivo ms4w_2.2.6.zip, se descomprime en un directorio por ejemplo en C:\ms4w.

Dentro de la carpeta C:\ms4w encontramos el archivo apache-install.bat, lo ejecutamos con doble click para poner en funcionamiento el Servidor Web Apache, aparece una ventana de comandos como en la Figura C-1:

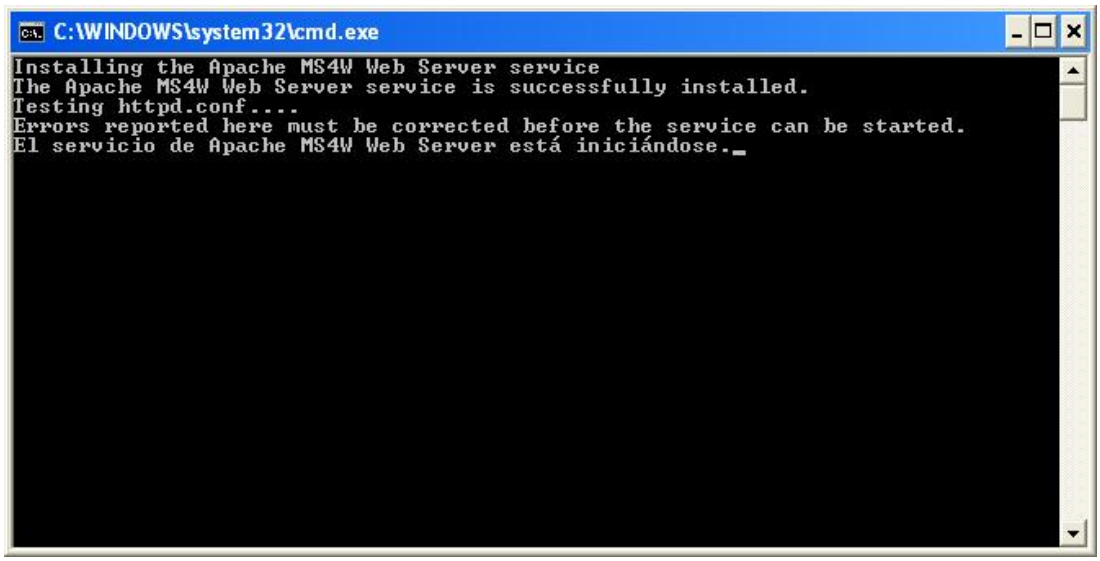


Figura C-1 Ventana de Inicio del Servicio de Apache MS4W

Para probar si el servidor web está funcionando apropiadamente, abrimos una ventana de navegador cualquiera (Internet Explorer, Mozilla, etc.) y escribimos la siguiente URL: http://localhost/ o http://127.0.0.1/ y debe aparecer la página principal en el navegador como en la Figura C-2:

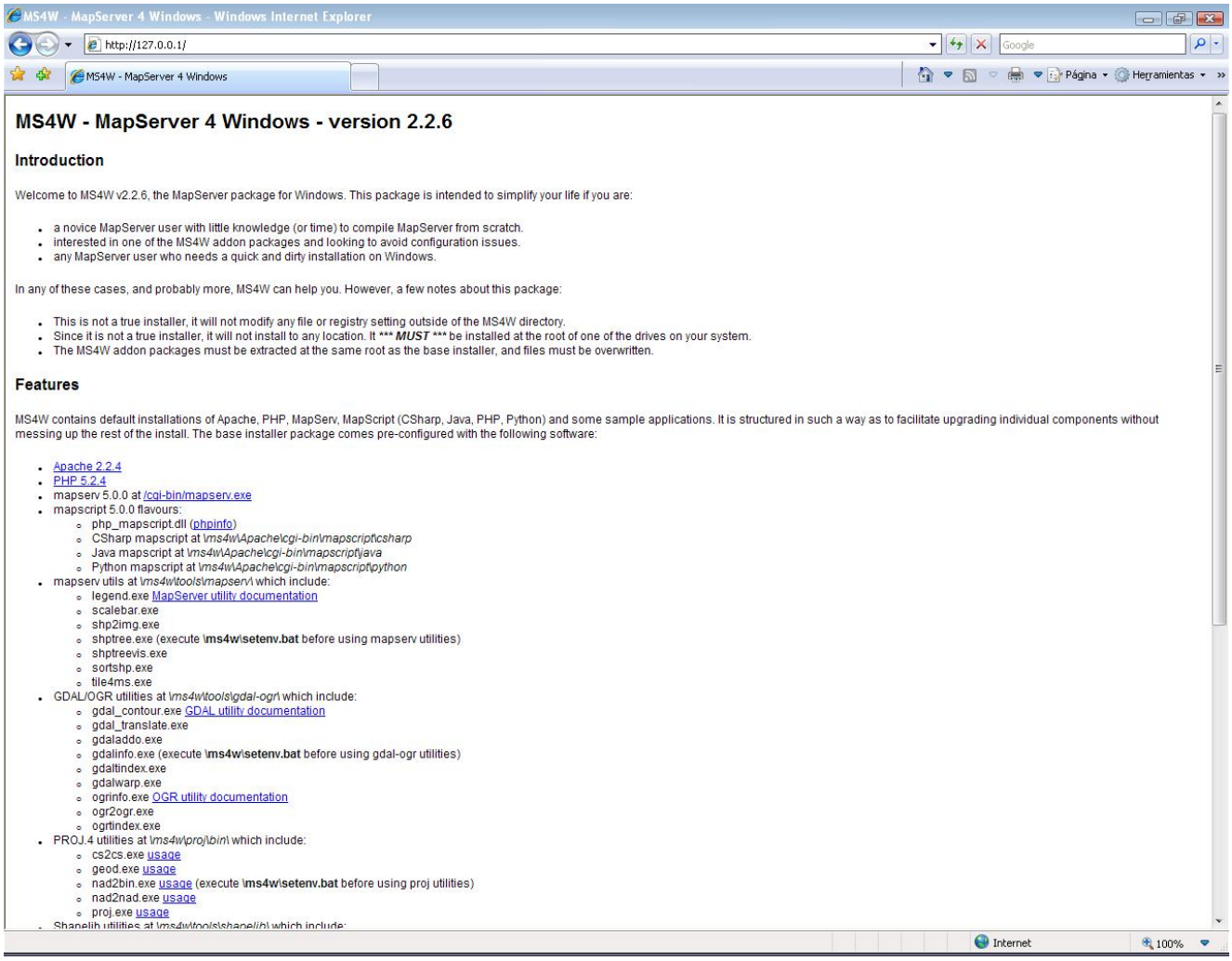


Figura C-2 Página principal de MapServer en el navegador

C.4 MAPSERVER

Es una aplicación de carácter libre distribuida bajo la licencia GPL8 construida por la Universidad de Minnesota, corre dentro de un servidor http bajo plataformas Linux/Apache, Windows XP/9x y funciona bajo la norma CGI (Common Gateway Interface) la cual establece los parámetros de comunicación entre un servidor Web y un programa, para que éste último interactúe con Internet. Pero es distinto si el usuario utiliza MapScript y el acceso a la API de MapServer cuando desarrolla aplicaciones más avanzadas.

La Figura C-3 muestra una arquitectura de servicios web geográficos con MapServer, pero el servidor de mapas es ArcIMS y el servidor de datos ArcSDE, que son herramientas software propietarias de ESRI[1].

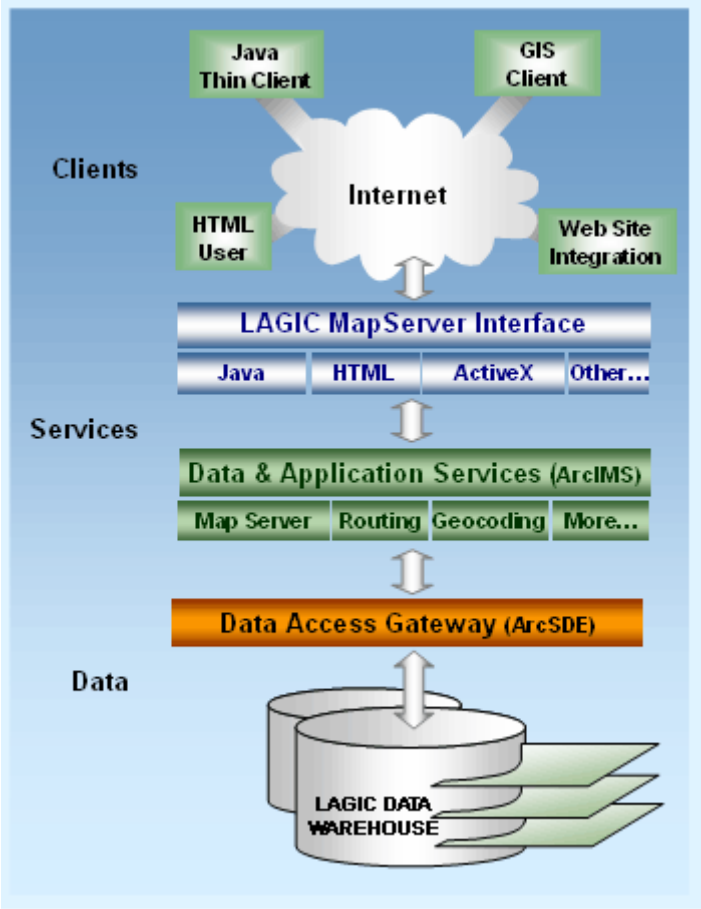


Figura C-3 Ejemplo de aplicación con MapServer en LAGIC[2]

El CGI utiliza los siguientes recursos:

- Servidor http como Apache o Microsoft IIS
- Software MapServer
- Archivo MapFile que controla lo que MapServer hace con los datos.
- Archivo TemplateFile que controla la aplicación MapServer en la ventana del navegador.

[1] ESRI. The GIS Software Leader. http://www.esri.com/
[2] LAGIC. Louisiana Geographic Information Center. http://lagic.lsu.edu/

- Fuente de datos SIG.

Las características de MapServer son:

- Dibujo y etiquetado en múltiples escalas.
- Valores de escalas.
- Símbolos y colores adaptables.
- Acceso a los atributos de los datos por medio de funciones.
- Generación automática de leyendas.
- Utilización de datos en forma de mosaico.

La aplicación MapServer viene compilada en el archivo mapserv.exe en el directorio C:\ms4w\Apache\cgi-bin y la información o fuente de datos se almacena en el directorio de documentos del servidor http, por ejemplo: C:\ms4w\Apache\htdocs.

C.4.1 ARCHIVO MapFile

Define los datos que serán usados en la aplicación, además de mostrar y consultar parámetros. Contiene información acerca de cómo se dibujará el mapa, la leyenda y los resultados de las búsquedas. Tiene normalmente una extensión .map.

C.4.2 ARCHIVO TemplateFile

Le permite al autor controlar la posición de presentación del mapa y la leyenda, igualmente determina las vías que tiene el usuario para interactuar con la aplicación por ejemplo: browse, query, zoom, etc.

C.4.3 Conjunto de datos SIG.

Utiliza archivos ESRI-Shapefile con formato vector por defecto, pero también puede utilizar archivos en formato ráster. Soporta tipos de archivo como geoTiff, Tiff y otros tantos. El conjunto de datos se almacena en un directorio y son referenciados en el MapFile.

Ahora se prueba si MapServer está funcionando, al igual que se realizó con Apache, se abre un navegador y se escribe lo siguiente: http://localhost/cgi-bin/mapserv.exe, debe aparecer la Figura C-4:

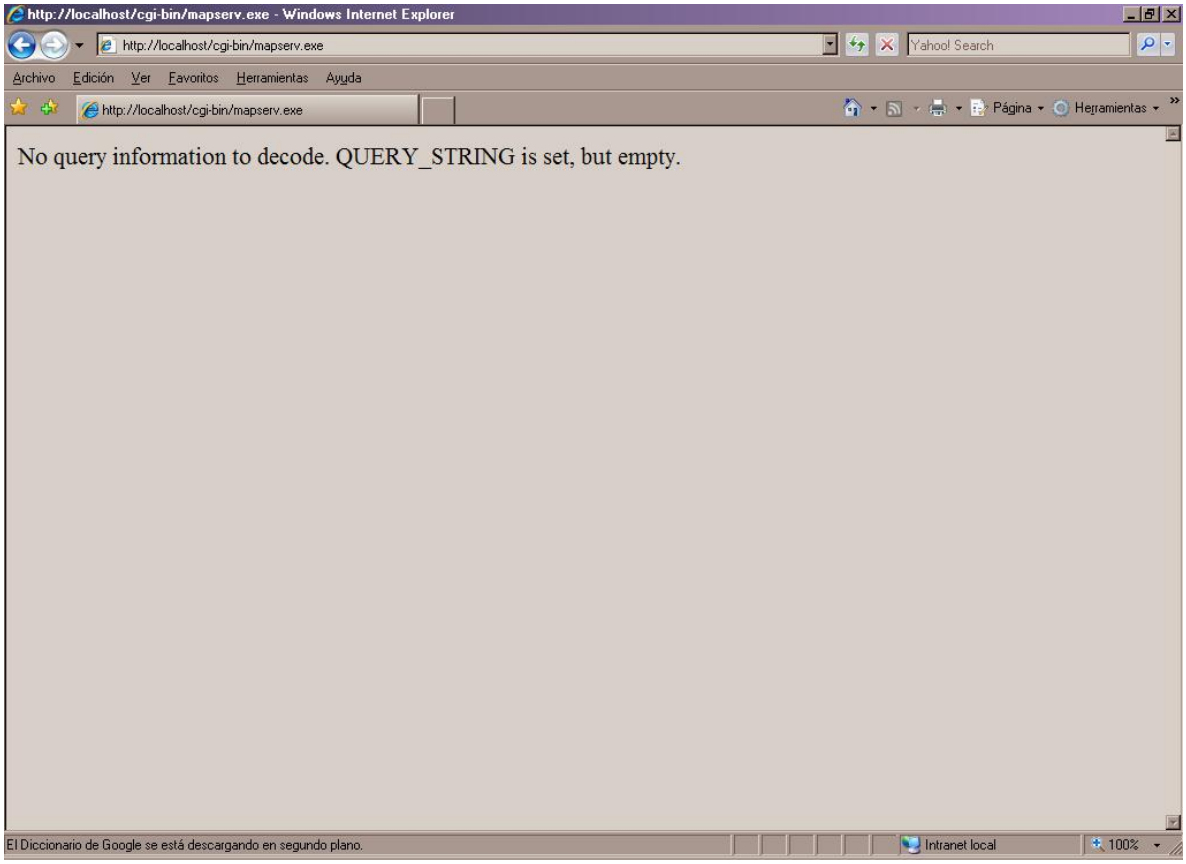


Figura C-4 Interfaz de MapServer iniciado.

El mensaje que muestra la ventana significa que MapServer está trabajando pero aún no tiene ningún tipo de datos para procesar.

A continuación se presenta la forma de desplegar un mapa con una sola capa.

1. **Mapa con una sola capa.**

Para comenzar se copia dentro del directorio C:\ms4w\apps\ la fuente de datos, es decir, los mapas o archivos .shp (shapefile) y junto a ella el archivo .map (mapfile). Como fuente de datos se utiliza un mapa llamado unicauca.shp que estará guardado

en el directorio C:\ms4w\apps\sig\data\ y el archivo ucauca.map estará en el directorio C:\ms4w\apps\sig\, se edita el archivo .map con el siguiente código:

```
MAP
  IMAGETYPE      PNG
  EXTENT                 1052584.238976  761500.619778  1053300.122902
762850.685620
  SIZE           800 600
  SHAPEPATH      "data"
  IMAGECOLOR     245 255 255

  # Start of LAYER DEFINITIONS ------------------------------------

  LAYER # edificio polygon layer begins here
    NAME        edificio
    DATA        unicauca
    STATUS    DEFAULT
    TYPE        POLYGON

CLASSITEM  "AREA"

    CLASS
      NAME 'CDU'
      EXPRESSION '44409.83'
      COLOR        207 150 66
      OUTLINECOLOR 32 32 32
    END

    CLASS
      NAME 'Servicios Generales'
      EXPRESSION '9428.16'
      COLOR        20 10 145
      OUTLINECOLOR 32 32 32
    END

   CLASS
      NAME 'FACENED'
```

```
      EXPRESSION '23612.12'
      COLOR        20 158 52
      OUTLINECOLOR 32 32 32
    END

CLASS
      NAME 'Ingenierías'
      EXPRESSION '29178.44'
      COLOR        50 205 145
      OUTLINECOLOR 32 32 32
    END

CLASS
      NAME 'Residencias'
      EXPRESSION '1729.22'
      COLOR        240 1 14
      OUTLINECOLOR 32 32 32
    END

  END # States polygon layer ends here

LAYER # cancha line layer begins here
    NAME         cancha
    DATA         canchas
    STATUS       OFF
    TYPE         LINE

    CLASS
      NAME       "cancha"
      STYLE
        COLOR    241 2 15
      END
    END
  END # States line layer ends here

LAYER # plano line layer begins here
    NAME         plano
```

```
    DATA         manzanas
    STATUS       OFF
    TYPE         LINE

    CLASS
      NAME       "plano"
      STYLE
        COLOR    32 32 32
      END
    END
  END # States line layer ends here

LAYER # rio line layer begins here
    NAME         rio
    DATA         drenajes
    STATUS       OFF
    TYPE         LINE

    CLASS
      NAME       "rio"
      STYLE
        COLOR    75 115 247
      END
    END
  END # States line layer ends here

  # End of LAYER DEFINITIONS ------------------------------

END # end of map file/object
```

Ahora explicamos las partes que componen el archivo .map:

La estructura que sigue el .map es jerárquica, se compone de múltiples objetos formando un árbol dependiendo de la complejidad de la aplicación. Al inicio del archivo siempre va el objeto MAP, cada objeto se define con un nombre y se cierra con la palabra END. El símbolo # se utiliza para añadir comentarios.

MAP: todo archivo mapfile comienza con éste objeto.

IMAGETYPE: define el formato de salida de los mapas mostrados en el navegador. En este caso se utiliza el formato PNG, pero puede ser GIF, WBMP, JPEG. Para especificar otros formatos de salida como PDF, SWF o GeoTIFF se utiliza el objeto OUTPUTFORMAT que es más complejo.

EXTENT: define la extensión o borde limitante de salida del mapa. El formato se escribe así: <Valor más bajo en X> <Valor más bajo en Y> <Valor más alto en X> <Valor más alto en Y> separados con un espacio. Debe escribirse en las mismas unidades de la fuente de datos.

SIZE: define el tamaño de salida del mapa en pixeles, por ejemplo: 800x600 los valores van separados por un espacio.

SHAPEPATH: es la ruta donde se almacenan los datos.

IMAGECOLOR: representa el color de fondo del mapa. Los valores se representan por el código RGB, por ejemplo: 255 Red, 255 Green y 255 Blue da como resultado Blanco.

Ahora se explica los objetos que se definen para representar las capas de los mapas.

LAYER: determina el inicio de una capa dentro del objeto MAP. Por defecto se puede establecer hasta un límite de 100 capas, pero se puede modificar editando el archive map.h.

NAME: es el identificador de la capa, MapServer lo usa para activar o desactivar cada capa.

DATA: es el nombre del directorio o carpeta donde se encuentran los datos, por ejemplo los archivos shapefile.

TYPE: identifica los tipos de datos que se va a utilizar, por ejemplo: POLYGON, LINE o POINT. También se puede especificar datos RASTER o ANNOTATION.

STATUS: muestra el estado de cada capa, es decir, si está activa o no. Por defecto la palabra DEFAULT quiere decir que está activa.

A continuación se definen los objetos que representan una clase.

CLASS: representa el inicio de una clase dentro del objeto LAYER. Se pueden definir hasta un máximo de 50 clases.

NAME: es el nombre de la clase, un objeto LAYER puede tener múltiples clases, es necesario dar un nombre descriptivo apropiado porque MapServer lo utiliza para el despliegue de leyendas.

Finalmente están los objetos que definen el estilo del mapa.

STYLE: marca el comienzo del objeto STYLE. Se utiliza en la superposición de capas en un mapa complejo.

COLOR: es el color con el que se pinta un polígono o una línea. Usa el formato RGB.

OUTLINECOLOR: define color en los bordes de un polígono, usa los valores del formato RGB, por defecto no viene activada.

Si se abre una ventana de navegador y escribimos la siguiente URL: http://localhost/cgibin/mapserv.exe?map=/ms4w/apps/sig/ucauca.map&layer=edificio&layer=cancha&layer=plano&mode=map tenemos la Figura C-5:

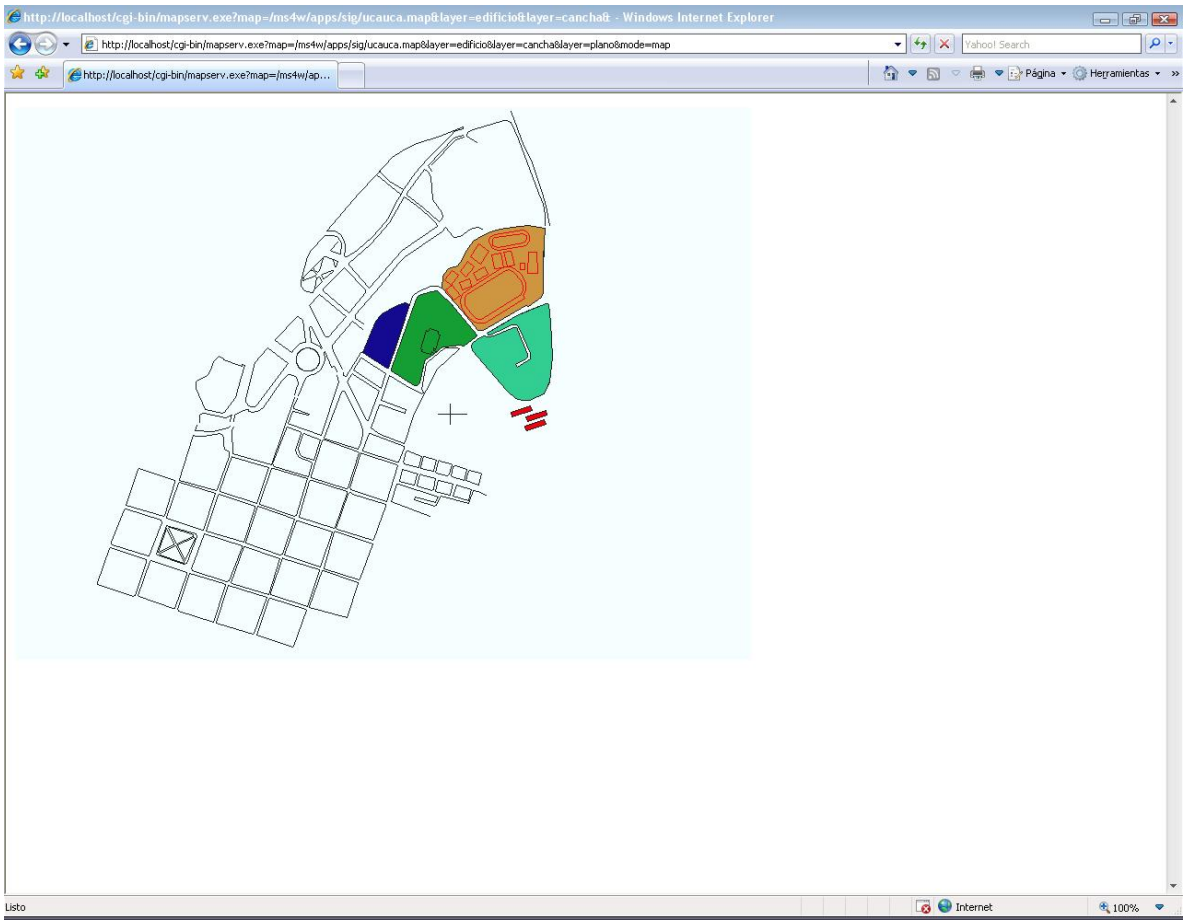


Figura C-5 Despliegue de un mapa con una capa.

Ahora si escribimos la siguiente URL: http://localhost/cgi-bin/mapserv.exe?map=/ms4w/apps/sigfinal/ucauca.map&layer=edificio&layer=cancha&layer=plano&layer=rio&mode=map tenemos una capa adicional, la cual dibuja un río como en la Figura C-6:

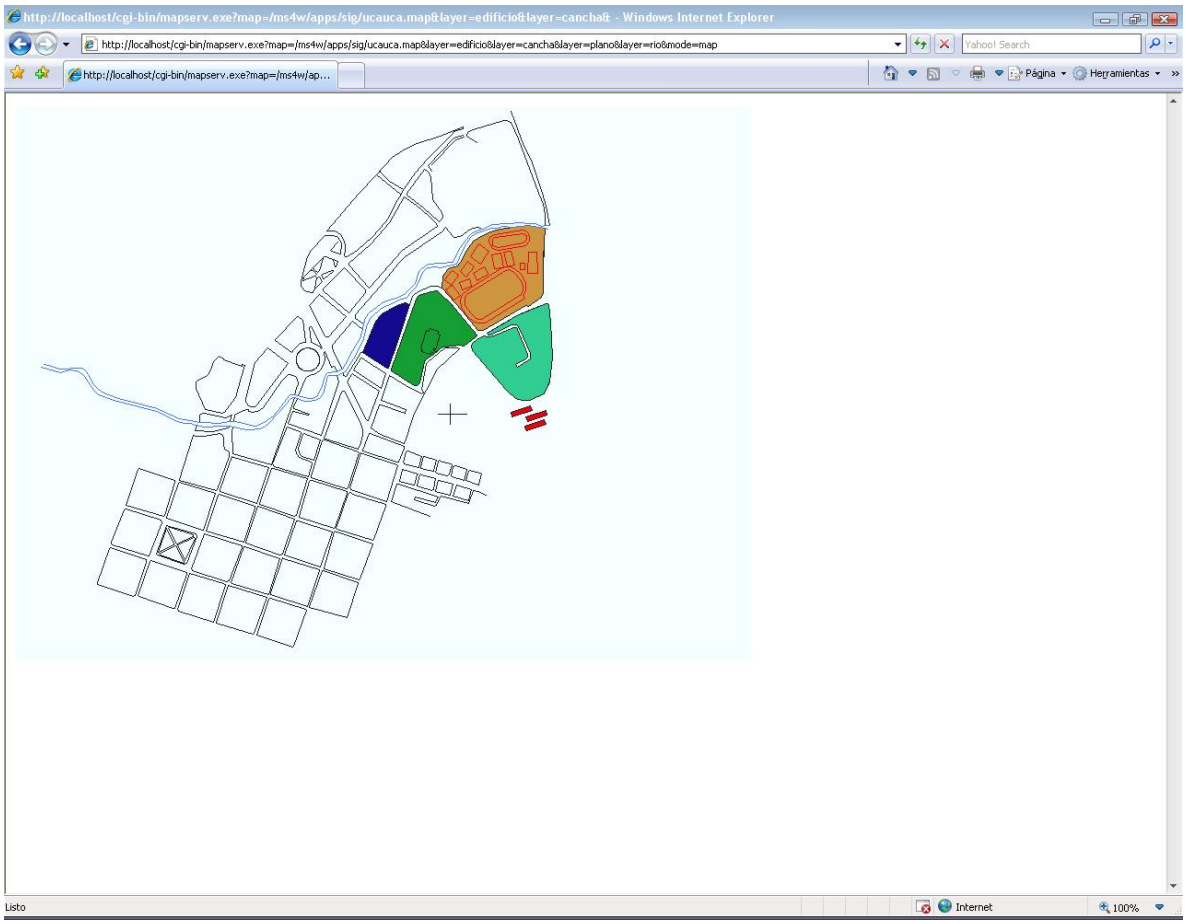


Figura C-6 Despliegue de un mapa con dos capas.

C.5 CONEXIÓN DESDE MAPSERVER

Requisitos:

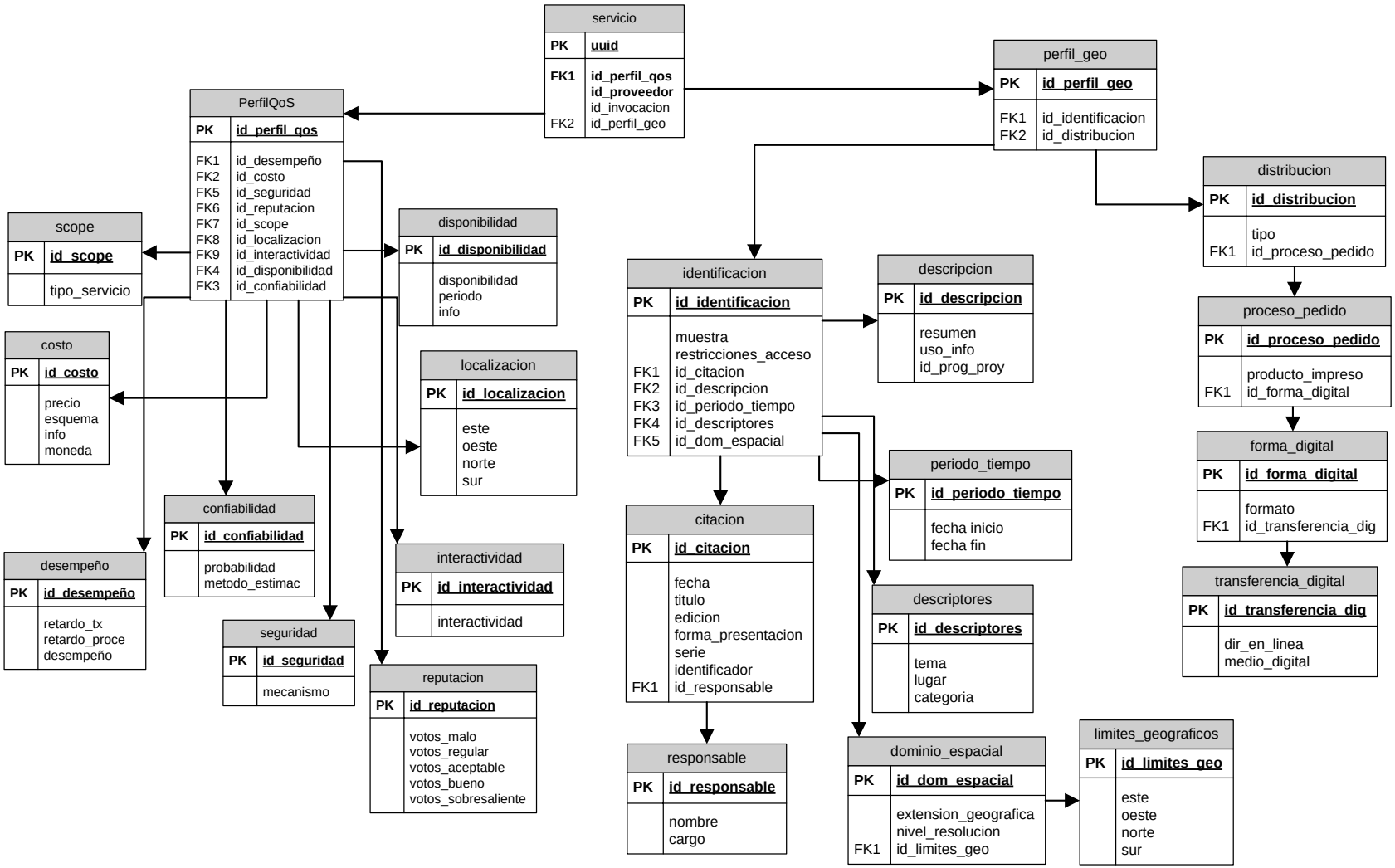
- MapServer correctamente instalado y funcionando.
- Conexión a Internet por medio de DSL, RDSI, T1 o cable en el servidor que contiene y corre MapServer. En lo posible para obtener mejores resultados buscar disponibilidad de una conexión de alta velocidad.

Los pasos son los siguientes:

1. Editar el archivo de configuración .map
2. Seleccionar el servidor WMS que se desea agregar. Se puede obtener una lista completa de las capas WMS disponibles desde la funcionalidad GetCapabilities.
3. http://icvmapas.icv.gva.es/wms?VERSION=1.1.1&REQUEST=getCapabilities&SERVICE=WMS (Archivo XML editable con cualquier procesador de texto).
4. Agregar el siguiente código para una capa WMS adicional:

```
LAYER
NAME "layer_wms_icv"
TYPE raster
STATUS on
CONNECTION http://icvmapas.icv.gva.es/wms
CONNECTIONTYPE wms
METADATA
“wms_srs” “EPSG:23030”
“wms_name” “layer1,layer2”
“wms_server_version” “1.1.1”
“wms_formatlist” “image/png, image/jpeg”
“wms_format” “image/png”
END
END
```

D. ANEXO D. MODELO ENTIDAD – RELACIÓN



E. ANEXO E. MANUAL TÉCNICO Y DE USUARIO

E.1 MANUAL TÉCNICO

E.1.1 Instalación del registro de servicios

- Instalar la Base de datos, para este caso se utilizó postgreSQL.
- Instale el contenedor Web *Tomcat* de la Apache Software Foundation (http://tomcat.apache.org/).
- Descargue la distribución de juddi del sitio Web http://ws.apache.org/juddi/releases.html
- Copie la carpeta *webapp/juddi* de la distribución en la carpeta *webapps* de *Tomcat.*
- Ejecute el script de la base de datos que viene con la distribución de juddi.
- Para referenciar la base de datos desde Tomcat 5, edite el archivo *server.xml* en la carpeta *conf* de Tomcat.
- Ingrese el siguiente tag antes de cerrar el tag <*Host*> del archivo *server.xml*

```
<Context path="/juddi" docBase="juddi" debug="5" reloadable="true"
crossContext="true">
<Resource name="jdbc/juddiDB" auth="Container"
type="javax.sql.DataSource"
username="*****" password="*****" driverClassName="org.postgresql.Driver"
url="jdbc:postgresql://localhost:3306/*****?autoReconnect=true"
validationQuery="select count(*) from PUBLISHER" />
</Context>
```

- Edite lo que está con asteriscos apropiadamente con el usuario, password y nombre de la base de datos creada anteriormente.

E.1.2 Instalación de la aplicación del catalogo

- Copiar las librerías (contenidas en Prototipo/lib) provistas en el CD entregado con esta monografía, en el directorio *Config/lib* de *Tomcat*.
- Copie la carpeta *CatalogoHidroServicios* (contenida en Prototipo) en la carpeta *webapps* de *Tomcat.*
- En la carpeta config del proyecto (*CatalogoHidroServicios/Web/xml*) edite el archivo configuración ingresando los datos de la ubicación del registro de servicios (URL), y el usuario.

E.2 MANUAL DE USUARIO

E.2.1 Proceso de Registro de Servicios

- En la página principal de click en la opción *Registro* esto abre la interfaz para el proveedor de servicios. Figura E-1.

Figura E-1 Página Principal del Catálogo

- Si ya está registrado como proveedor, ingrese su login y password y a continuación de click en el botón ingresar.
- Si no está registrado proceda al link registro de proveedores como se ve en la Figura E-2.

Figura E-2 Interfaz de Ingreso a Registro de Servicios

- Una vez ingresado como proveedor, ingrese la información básica del servicio que desea registrar, y de click en siguiente, como se ve en la Figura E-3.

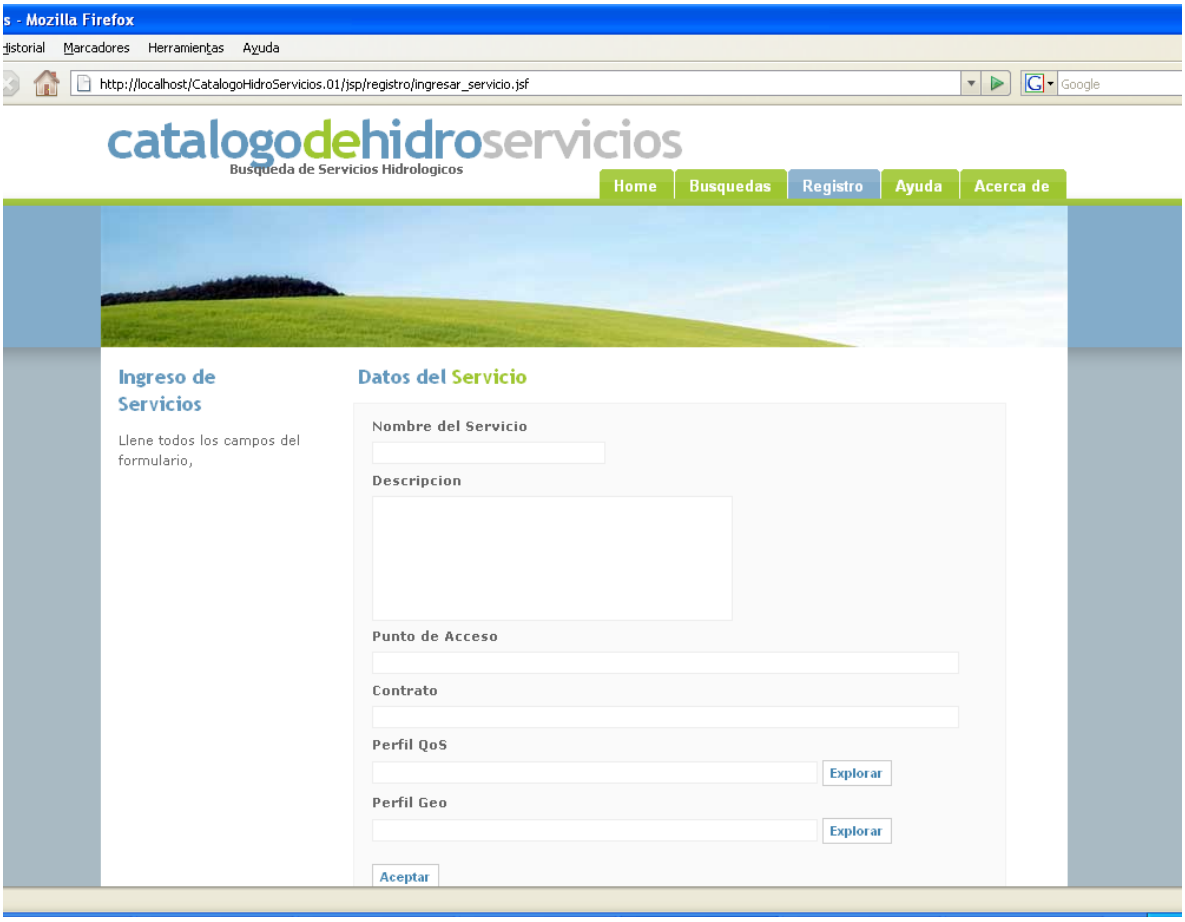


Figura E-3 Interfaz de Registro de Servicios

- A continuación ingrese la ruta del perfil geográfico asociado al servicio. Ingrese la información de calidad del servicio. Ingrese los demás datos.
- De Aceptar y espere confirmación. Con esto se da por concluido el registro.

E.2.2 Proceso de Búsqueda

- En la página principal de click en la opción *Búsquedas* esto abre la interfaz para búsqueda de servicios. Figura E-4.

Figura E-4 Pagina Principal

- Ingrese la pabla clave para hacer la búsqueda, seguido de los atributos de calidad relevantes y el peso asignado a cada atributo. Figura E-5.

Figura E-5 Interfaz de Búsqueda de Servicios

- Navegue por los diferentes servicios encontrados, ordenados de acuerdo a la selección realizada.
- Consulte la información detallada de un servicio dando click en la opción monitor de metadatos. Figura E-6.

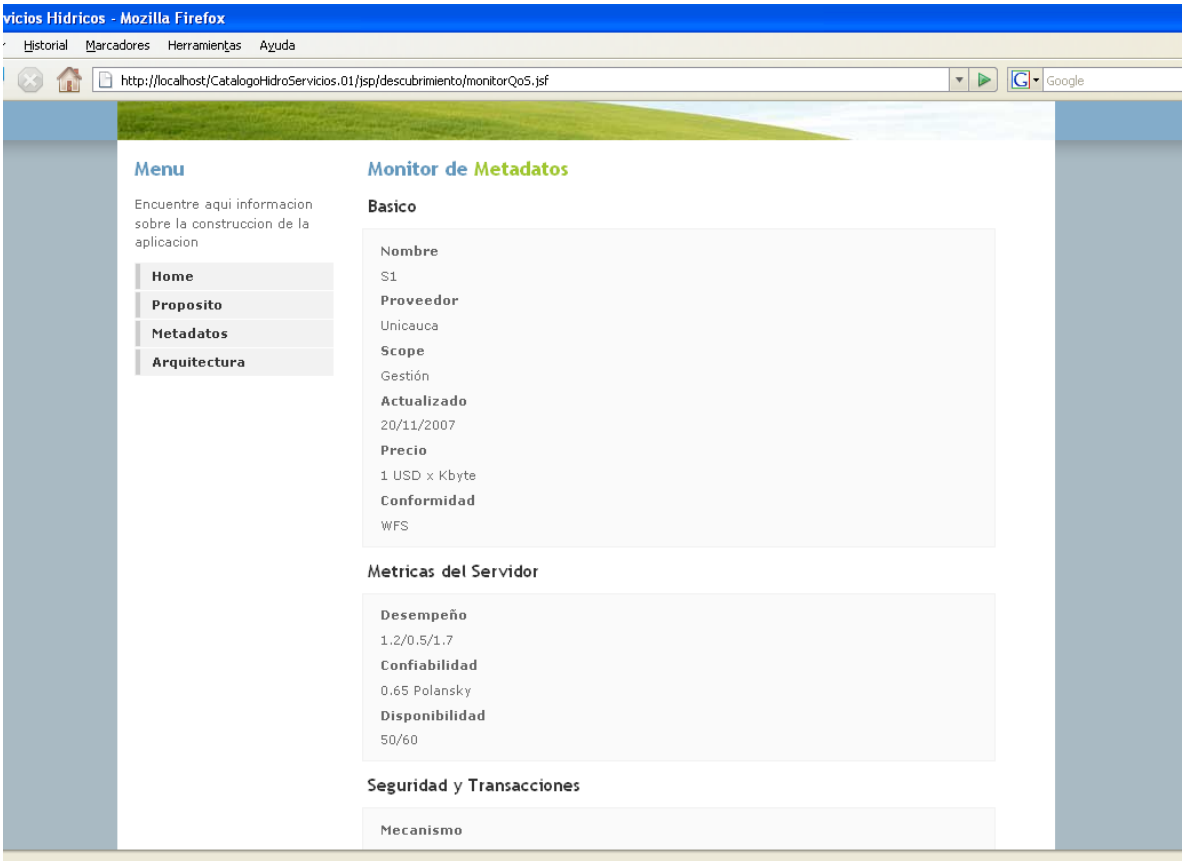


Figura E-6 Monitor de Metadatos