ATLAS
HERRAMIENTA DE CARTOGRAFÍA WEB Y GEOCODIFICACIÓN
V.2

MANUAL DEL USUARIO

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE SISTEMAS
SAN JUAN DE PASTO
2018

Tabla de contenido

Tabla de Figuras

1. ARQUITECTURA DE ATLAS

La herramienta ATLAS la conforman tres grandes componentes: el servidor, el panel de control y los componentes de desarrollo, como se muestra en la figura 1.

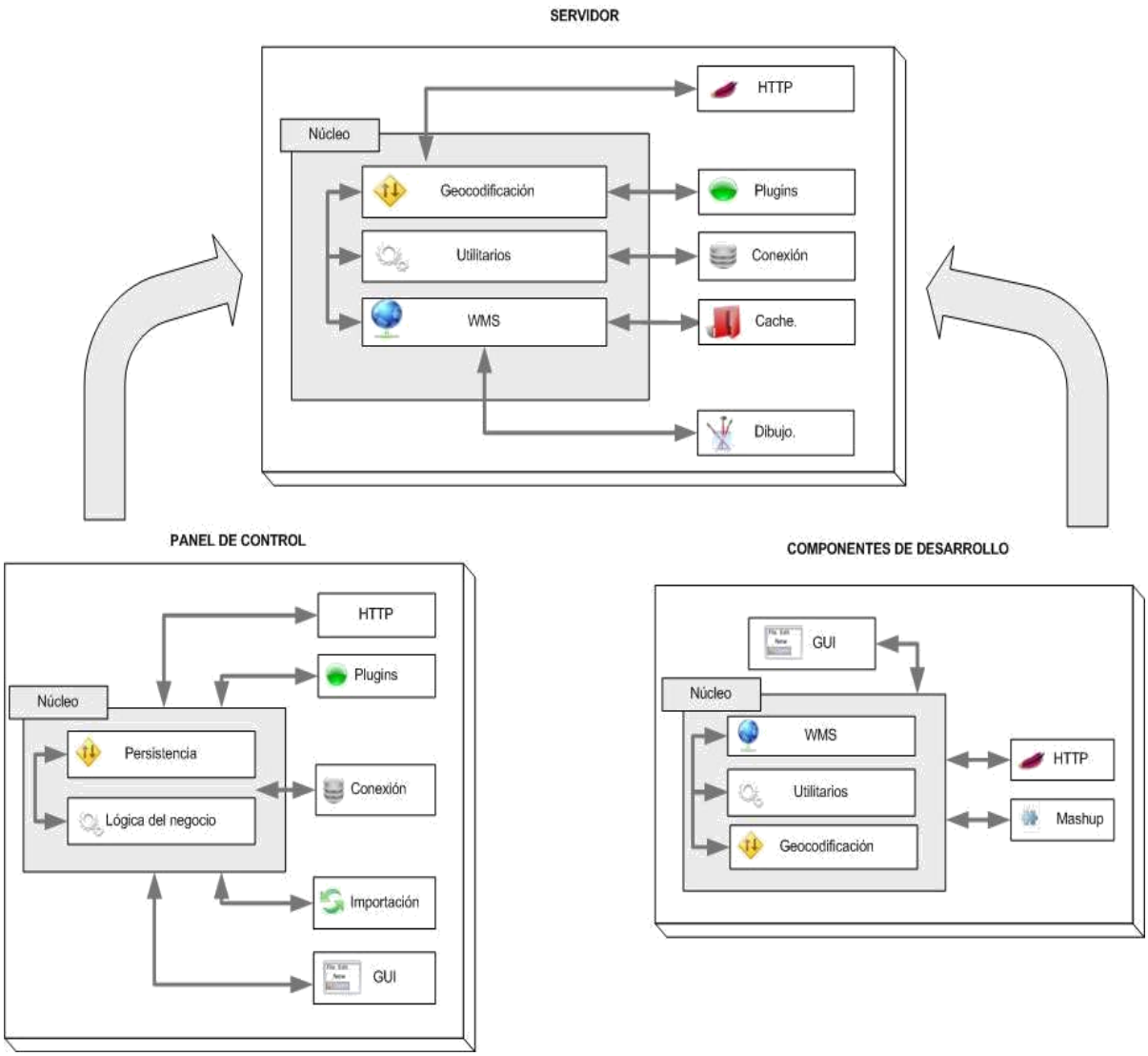

Figura 1. Arquitectura de la herramienta Atlas

1.1. SERVIDOR

Este componente tienen los siguientes módulos:

1.1.1. Módulo Núcleo

Compuesto por los submódulos:

Submódulo WMS. Este submódulo es el encargado de administrar el envío y recepción de las peticiones de conformidad con el estándar WMS.

Submódulo de Utilitarios. Este submódulo contiene clases e interfaces que resultan útiles a lo largo de toda la aplicación, por ello se colocan juntas en un módulo especial, compartido por las demás partes del sistema. Algunas de las clases forman la estructura lógica de la aplicación, mientras que otras contienen métodos estáticos que realizan funciones de utilidad específica como las operaciones de E/S con flujos.

Submódulo de Geocodificación. Este submódulo es el encargado de administrar la configuración, el envío y recepción de las peticiones al servicio de geocodificación.

1.1.2. Otros Módulos

Módulo HTTP. Este módulo es el encargado de administrar las comunicaciones de la herramienta mediante conexiones HTTP.

Módulo Plugins. Este módulo es el encargado de administrar y configurar el sistema de plugins para el servicio de geocodificación en la herramienta Atlas.

Módulo de Conexión. Este módulo es el encargado de mantener una comunicación constante entre la herramienta y la base de datos.

Módulo de Cache. Este módulo es el encargado de administrar la carga y el almacenamiento de las imágenes generadas por el servidor de cartografía en el sistema de archivos del servidor.

Módulo de Dibujo. Este módulo es el encargado de graficar cada una de las imágenes generadas solicitadas servidor, usando las geometrías vectoriales de los orígenes de datos, y de conformidad a todos los parámetros de la petición GetMap del estándar WMS.

1.2. PANEL DE CONTROL

Este componente tienen los siguientes módulos:

1.2.1. Módulo Núcleo

Compuesto por los submódulos:

Submódulo de Persistencia. Este submódulo permite realizar la carga y el almacenamiento de los objetos del sistema en la base de datos usando persistencia.

Submódulo de Lógica del Negocio. Este submódulo contiene todos las clases y programas qe permiten la geocodificación.

1.2.2. Otros Módulos

Ademas de los módulos de HTTP, Plugins y Conexión, elpanel decontrol tiene los siguientes módulos:

Módulo de importación. Este módulo es el encargado de importar los formatos de información geográfica soportados por Atlas a la base de datos, su objetivo se centra en verificar la consistencia de estas fuentes y realizar las transformaciones requeridas.

Módulo de interfaz gráfica - GUI. Este módulo da soporte visual a los demás módulos y se encarga de brindar al usuario una experiencia muy amigable durante la manipulación de la herramienta, de modo tal que resulte sencillo y fácil realizar todas las operaciones requeridas.

Módulo Mashup. Este módulo se encarga de administrar los componentes asociados a la producción de aplicaciones hibridas usando la biblioteca.

1.3. COMPONENTES DE DESARROLLO

El módulo Núcleo contiene los mismos componentes que el módulo Núcleo del servidor y dentro de otros módulos están los módulos de Módulo de interfaz gráfica – GUI, Módulo HTTP y Módulo Mashup.

1.4. PAQUETES DEL MÓDULO DE UTILIDAD

1.4.1. Paquete clientdata

Contiene las clases que representan la estructura de datos del sistema que se requiere en el componente de desarrollo del escritorio:

Clase BBox. Una representación ligera de un bounding box para ser usada en los componentes de desarrollo. Un bounding box encierra una región del espacio 2D usando dos coordenadas. Su interpretación depende del sistema de coordenadas de referencia, que define las unidades en que se expresan las coordenadas y la orientación de los ejes. Más detalles de esta clase se muestran en la tabla 1.

Tabla 1. Resumen de la clase BBox.

Constructores	
BBox(double minx, double miny, double maxx, double maxy) Construye dados valores máximos y mínimos.	
BBox(Element bBox) Construye desde un elemento XML compatible con WMS.	
Resumen de métodos	
Point2D	**getCenter**() Retorna el centro geométrico del bounding box.
double	**getHeight**() Retorna el alto del área delimitada.
double	**getMaxx**() Retorna el máximo valor del primer eje.
double	**getMaxy**() Retorna el máximo valor del segundo eje.
double	**getMinx**() Retorna el mínimo valor del primer eje.
double	**getMiny**() Retorna el mínimo valor del segundo eje.
String	**getUrlForm**() Retorna el bounding box en forma de KVP para ser usado en peticiones.
double	**getWidth**() Retorna el ancho del área delimitada.
void	**setMaxx**(double maxx) Establece el máximo valor del primer eje.
void	**setMaxy**(double maxy) Establece el máximo valor del segundo eje.
void	**setMinx**(double minx) Establece el mínimo valor del primer eje.
void	**setMiny**(double miny) Establece el mínimo valor del segundo eje.

Clase EpsgCRS. Representa un sistema de coordenadas de referencia del EPSG para uso en los componentes de desarrollo de escritorio. Más detalles de esta clase se muestran en la tabla 2.

Tabla 2. Resumen de la clase EpsgCRS.

Resumen de campos	
static int	**UNIT_ANGULAR** Indica que el sistema funciona con unidades angulares que se pueden expresar en términos de grados.
static int	**UNIT_LINEAR** Indica que el sistema funciona con unidades lineales que se pueden expresar en términos de metros.
Resumen de constructores	
EpsgCRS() Construye una nuevo objeto sin valores.	
EpsgCRS(Element crs) Construye desde el elemento XML que entrega el servidor como respuesta a una petición GetEPSGCodeInfo.	
Resumen de métodos	
String	**getAxis1**() Retorna el nombre del primer eje.
String	**getAxis2**() Retorna el nombre del segundo eje.
int	**getCode**() Retorna el código EPSG del sistema.
double	**getConversionFactor**() Retorna el factor por el que deben multiplicarse las medidas para convertirlas a la unidad base de sistema.
Element	**getEpsgCRS**(String code, URL serviceURL) Retorna un Element producto de una solicitud GetEPSGCodeInfo sobre el servidor indicado.
String	**getName**() Retorna el nombre EPSG del sistema.
int	**getType**() Retorna el tipo del sistema según las constantes de clase.
void	**setAxis1**(String axis1) Establece el nombre del primer eje.
void	**setAxis2**(String axis2) Establece el nombre del segundo eje.
void	**setCode**(int code) Establece el código EPSG del sistema.
void	**setConversionFactor**(double conversionFactor) Establece el factor por él que deben multiplicarse las medidas para convertirlas a la unidad base de sistema.
void	**setName**(String name) Establece el nombre EPSG del sistema.
void	**setType**(int type). Establece el tipo del sistema según las constantes de clase.

Clase Layer. Representa una capa para su uso en los componentes de desarrollo.Más detalles de esta clase se muestran en la tabla 3.

Tabla 3. Resumen de la clase Layer

Resumen de constructores	
Layer() Construye un objeto sin valores.	
Layer(Element layer, Project proyect) Construye un objeto desde un elemento XML y lo anexa a un proyecto.	
Resumen de métodos	
String	**getAtribution**() Retorna el responsable de la capa. De conformidad al estándar WMS.
BoundingBox	**getBoundigBox**() Retorna el bounding box en el sistema de coordenadas de referencia del proyecto. De conformidad al estándar WMS.
EpsgCRS	**getCRS**() Retorna el sistema de coordenadas de referencia para la capa.
Layer.Ex_BoundingBox	**getEx_BoundingBox**() Retorna el bounding box respecto a WGS 84. De conformidad al estándar WMS.
Java.util.List	**getKeywordList**() Retorna un listado de palabras clave. De conformidad al estándar WMS.
double	**getMaxscaledeno**() Retorna el denominador máximo de escala. De conformidad al estándar WMS.
double	**getMinscaledeno**() Retorna el denominador mínimo de escala. De conformidad al estándar WMS.
String	**getName**() Retorna el identificador de la capa. De conformidad al estándar WMS.
Project	**getProyect**() Retorna el proyecto al que pertenece la capa.
String	**getSummary**() Retorna un resumen de la capa. De conformidad al estándar WMS.
String	**getTitle**() Retorna una descripción breve para mostrar. De conformidad al estándar WMS.

boolean	**isQueryable**() Indica si la capa puede ser consultada en una operación GetFeatureInfo.
void	**setAtribution**(String atribution) Establece el responsable de la capa. De conformidad al estándar WMS.
void	**setBoundigBox**(BoundingBox boundigBox) Establece el bounding box en el sistema de coordenadas de referencia del proyecto. De conformidad al estándar WMS.
void	**setCRS**(EpsgCRS CRS) Establece el sistema de coordenadas de referencia para la capa.
void	**setEx_BoundingBox**(Layer.Ex_BoundingBox exBoundingBox) Establece el bounding box respecto a WGS 84. De conformidad al estándar WMS.
void	**setKeywordList**(List keywordList) Establece un listado de palabras clave. De conformidad al estándar WMS.
void	**setMaxscaledeno**(double maxscaledeno) Establece el denominador máximo de escala. De conformidad al estándar WMS.
void	**setMinscaledeno**(double minscaledeno) Establece el denominador mínimo de escala. De conformidad al estándar WMS.
void	**setName**(String name) Establece el identificador de la capa. De conformidad al estándar WMS.
void	**setProyect**(Project proyect) Establece el proyecto al que pertenece la capa.
void	**setQueryable**(boolean queryable) Establece si la capa puede ser consultada por una operación GetFeatureInfo.
void	**setSummary**(String summary) Establece el resumen de la capa. De conformidad al estándar WMS.
void	**setTitle**(String title) Establece descripción breve para mostrar. De conformidad al estándar WMS.

Clase Project. Representa un proyecto para su uso en los componentes de desarrollo, contiene un conjunto de capas. Más detalles de esta clase se muestran en la tabla 4.

Tabla 4. Resumen de la clase Project.

Resumen de constructores	
Project() Construye un proyecto con los campos vacíos.	
Project(Element project) Construye desde un elemento XML de un documento GetCapabilities.	
Resumen de métodos	
BBox	**getBbox**() Retorna el bounding box combinado de los bounding boxes de los orígenes de datos de las capas del proyecto. De conformidad al estándar WMS.
EpsgCRS	**getCrs**() Retorna el sistema de coordenadas de referencia.
List	**getLayers**() Retorna el listado de capas del proyecto.
String	**getName**() Retorna el identificador del proyecto. De conformidad al estándar WMS.
String	**getTitle**() Retorna una descripción breve para mostrar. De conformidad al estándar WMS.
void	**setBbox**(BBox bbox) Establece el bounding box en el sistema de coordenadas de referencia del proyecto. De conformidad al estándar WMS.
void	**setCrs**(EpsgCRS crs) Establece el sistema de coordenadas de referencia.
void	**setLayers**(Java.util.List layers) Establece el listado de capas del proyecto.
void	**setName**(String name) Establece el identificador del proyecto. De conformidad al estándar WMS.
void	**setTile**(String title) Establece una descripción breve para mostrar. De conformidad al estándar WMS.

1.5. PAQUETES DEL MÓDULO DE GEOCODER

Contiene las clases e interfaces que deben extender e implementar todos los geocoders.

Clase Geocoder. Clase base del sistema de geocoders. Más detalles de esta clase se muestran en la tabla 5.

Tabla 5. Resumen de la clase Geocoder.

Resumen de métodos	
abstract JDialog	**getAboutDialog**() Debe retornar un cuadro de diálogo con información sobre el geocoder.
static String	**getCleanAddress**(String address) Retorna una cadena en la que se han suprimido todos los signos de puntación y se ha removido el acento de todas la vocales, también se han eliminado palabras como CON, DE, EL, EN, LA, LAS, LO, LOS, MAS, MI, MIS, PARA, POR, QUE, SIN, SU, SUS, TU, TUS, YA y se ha convertido a mayúsculas.
abstract ConfigurationDialog	**getConfigurationDialog**(HibernateUtil hibernate, Project project, GeocoderConfig config) Debe retornar un diálogo de configuración listo para funcionar con los objetos que se entregan como parámetros.
abstract String	**getDescription**() Debe retornar una descripción del geocoder para ser mostrada en pantalla.
abstract String	**getName**() Debe retornar el nombre del geocoder para ser mostrado en pantalla.
abstract Project	**getProject**() Debe retornar el proyecto para el que funciona esta instancia del geocoder.
staticString	**getSingleSpaced**(String address) Retorna una cadena en la que todas las ocurrencias de varios espacios consecutivos se han reemplazado por un solo espacio.
abstract String	**getStandardizedAddress**(String address) Debe retornar la dirección indicada después de llevarla a una forma en que contiene elementos léxicos estandarizados para el proceso, por ejemplo, el reemplazo de "calle", "cll", "clle", "cl" por "CL".
protected static boolean	**iFind**(String pattern,String target) Indica si una cadena contiene otra, sin tener en cuenta mayúsculas o minúsculas.
abstract void	**init**(ConnectionProperties c) Es llamado por el gestor en servidor antes de empezar a hacer peticiones, aquí el geocoder debe realizar todas las acciones que sean necesarias antes de empezar a operar, por ejemplo, formar cachés y verificar el estado de las tablas requeridas.

abstract boolean	**isCapableFor**(String address) Debe retornar falso o verdadero en función de, si el geocoder puede o no procesar la dirección indicada, dicha dirección está exactamente como llega en la petición.
abstract Result[]	**locateAddress**(String[] parts) Debe determinar las posibles ubicaciones para una dirección dados sus componentes léxicos .
abstract String[]	**parseAdress**(String adress) Retorna los componentes léxicos de la dirección estandarizada que se entrega como parámetro.
abstract void	**setGeocoderConfig**(GeocoderConfig gc) Debe establecer el objeto de configuración del geocoder para el proyecto.
abstract void	**setProject**(Project project) Debe establecer el proyecto para el que funciona esta instancia del geocoder.

Clase ConfigurationDialog. Súper clase para los diálogos de configuración de geocoders. Es usada por el gestor de plugins en el panel de control. Los métodos set de esta clase normalmente deben llamarse desde la implementación del método getConfigurationDialog del la clase Geocoder. Más detalles de esta clase se muestran en la tabla 6.

Tabla 6. Resumen de la clase ConfigurationDialog.

Resumen de constructores	
ConfigurationDialog() Crea un diálogo en blanco, a este punto aún no está listo para ser visible .	
Resumen de métodos	
GeocoderConfig	**getConfig**() Retorna el objeto de configuración del geocoder para un proyecto en particular.
HibernateUtil	**getHibernate**() Retorna el objeto de conexión hibernate.
Project	**getProject**() Retorna el proyecto para el que se configura el geocoder.
void	**setConfig**(GeocoderConfig config) Establece el objeto de configuración del geocoder para un proyecto en particular.
void	**setHibernate**(HibernateUtil hibernate) Establece el objeto de conexión hibernate.
void	**setProject**(Project project)

	Establece el proyecto para el que se configura el geocoder.

Clase Result. Clase que representa un resultado de operación de geocodificación. En la tabla 7 se muestra más detalles de esta clase.

Tabla 7. Resumen de la clase Result.

Resumen de constructores	
Result(String address, Point2D location) Construye dados una dirección y una ubicación.	
Resumen de métodos	
String	**getAddress**() Retorna la descripción textual de la ubicación tal como el geocoder la interpretó.
Element	**getAsElement**() Retorna el resultado como un elemento XML para ser incluido en la respuesta.
String	**getAsJSON**() Retorna el resultado como un elemento JSON para ser incluido en la respuesta.
Point2D	**getLocation**() Retorna la ubicación en el sistema de coordenadas de referencia del proyecto al que corresponde.
void	**setAddress**(String address) Establece la descripción textual de la ubicación tal como el geocoder la interpretó.
void	**setLocation**(Point2D location) Establece la ubicación en el sistema de coordenadas de referencia del proyecto al que corresponde.

1.6. PAQUETES DEL MÓDULO DE PERSISTENCIA

1.6.1. Paquete hibernatedata.

Paquete que contiene las clases principales de la estructura de datos del sistema y que tienen la capacidad de ser llevadas al almacenamiento persistente.

Clase AtlasColor. Representa un color en formato argb. En la tabla 8 se presenta un resumen de esta clase

Tabla 8. Resumen de la clase AtlasColor.

Resumen de constructores	
AtlasColor() Construye un color vacío.	
Resumen de métodos	
void	**delete**(org.hibernate.classic.Session sess) Borra este objeto del almacenamiento persistente.
int	**getAlfa**() Retorna el componente alfa del color.
int	**getBlue**() Retorna el componente azul del color.
int	**getCodecolo**() Retorna el código del color, para persistencia.
Java.awt.Color	**getColor**() Retorna un objeto Color con el rgb de este objeto.
int	**getGreen**() Retorna el componente verde del color.
int	**getRed**() Retorna el componente rojo del color.
void	**save**(org.hibernate.classic.Session sess) Guarda este objeto en una sesión de hibernate.
void	**setAlfa**(int alfa) Establece El componente alfa del color.
void	**setBlue**(int blue) El componente azul del color.
void	**setCodecolo**(int codecolo) Código del color, para persistencia.
void	**setColor**(Java.awt.Color color) Establece los valores rgba de este color.
void	**setGreen**(int green) Establece el componente verde del color.
Void	**setRed**(int red) Establece el componente rojo del color.
void	**setRGBA**(int r, int g, int b, int a) Establece los valores rgba para este color.

Clase AtlasStroke. Representa un estilo de línea, compatible con los estilos de línea de AWT. En la tabla 9 se presenta un resumen de esta clase.

Tabla 9. Resumen de la clase AtlasStroke.

Resumen de constructores
AtlasStroke() Construye un estilo de línea sin valores.

Resumen de métodos	
void	**delete**(org.hibernate.classic.Session sess) Remueve este objeto del almacenamiento persistente.
BasicStroke	**getBasicStroke**() Retorna un BasicStroke por defecto, de 1 píxel de ancho.
BasicStroke	**getBasicStroke**(double factor) Retorna un BasicStroke con base en este estilo de línea después de multiplicar su ancho por el factor indicado.
int	**getCap**() Retorna el cap del estilo de línea , forma de las terminales.
int	**getCodestro**() Retorna el código del estilo de línea en el sistema de persistencia.
float[]	**getDash**() Retorna el patrón de punteado de la línea como un array obtenido desde la representación en cadena.
float	**getDashphase**() Retorna el desplazamiento en el patrón de punteado.
String	**getDashString**() Retorna el patrón de punteado de la línea como cadena, para ser almacenado en la persistencia.
int	**getJoin**() Retorna el join del estilo de línea , decoración de la intersecciones.
float	**getMiterlimit**() Retorna el Miter limit del estilo de línea.
float	**getWidth**() Retorna el ancho del estilo de línea.
void	**save**(org.hibernate.classic.Session sess) Envía este objeto al almacenamiento persistente.
void	**setBasicStroke**(Java.awt.BasicStroke basicStroke) Establece los atributos del objeto para que iguale al basicStroke indicado.
void	**setCap**(int cap) Establece el cap del estilo de línea , forma de las terminales.
void	**setCodestro**(int codestro) Establece el código del estilo de línea en el sistema de persistencia.
void	**setDash**(float[] dash) Establece el patrón de punteado de la línea como un array obtenido desde la representación en cadena.
void	**setDashphase**(float dashphase) Establece el desplazamiento en el patrón de punteado.
void	**setDashString**(String dashString)

	Establece el el patrón de punteado de la línea como cadena, para ser almacenado en la persistencia.
void	**setJoin**(int join) Establece el join del estilo de línea , decoración de la intersecciones.
	void **setMiterlimit**(float miterlimit) Establece el Miter limit del estilo de línea.
void	**setWidth**(float width) Establece el ancho del estilo de línea .

Clase GeocoderConfig. Representa la configuración de un geocoder para un proyecto en particular. El contenido de la cadena configuración depende el desarrollador del plugin. En la tabla 10 se presenta un resumen de esta clase.

Tabla 10. Resumen de la clase GeocoderConfig.

Resumen de constructores	
GeocoderConfig() Construye un objeto sin valores.	
Resumen de métodos	
void	**delete**(org.hibernate.Session sess) Remueve este objeto del almacenamiento persistente.
String	**getClassname**() Retorna el nombre de la clase principal de geocoder.
int	**getCodegeoc**() Retorna el código del objeto en el sistema de persistencia.
String	**getConfiguration**() Retorna la cadena de configuración del geocoder.
Project	**getProject**() Retorna el proyecto al que pertenece esta configuración.
void	**save**(org.hibernate.Session sess) Envía este objeto al almacenamiento persistente.
void	**setClassname**(String classname) Establece el nombre de la clase principal de geocoder.
void	**setCodegeoc**(int codegeoc) Establece el código del objeto en el sistema de persistencia.
void	**setConfiguration**(String configuration) Establece la cadena de configuración del geocoder.
void	**setProject**(Project project) Establece el proyecto al que pertenece esta configuración.

Clase Icon. Representa un icono para ser aplicado sobre una geometría tipo punto. Los iconos del sistema se almacenan como representaciones SVG. Un resumen de la clase Icon se muestra en la tabla 11.

Tabla 11. Resumen de la clase Icon.

Resumen de campos	
static int	**DEFAULT_SIZE** Ancho inicial por defecto para los iconos en el sistema.
Resumen de constructores	
Icon() Construye un icono con campos vacíos.	
Icon(byte[] binary, String mime, String description) Construye un icono con los parámetros dados. Usualmente el campo binary contiene la representación binaria de cadena del SVG del icono.	
Resumen de métodos	
void	**createThumb**() Crea la vista en miniatura del icono, debe llamarse antes de llamar a getThumb.
byte[]	**getBinaicon**() Retorna el contenido binario del icono, normalmente de la cadena del documento SVG que lo representa.
int	**getCodeicon**() Retorna el código del objeto en el sistema de persistencia.
String	**getDescicon**() Retorna una descripción breve del icono.
BufferedImage	**getIcon**(int maxWidth, int maxHeight) Retorna una imagen del icono.
BufferedImage	**getThumb**() Retorna una vista en miniatura del icono.
String	**getTypeicon**() Retorna el tipo mime del icono.
void	**setBinaicon**(byte[] binaicon) Establece el contenido binario del icono, normalmente de la cadena del documento SVG que lo representa.
void	**setCodeicon**(int codeicon) Establece el código del objeto en el sistema de persistencia.
void	**setDescicon**(String descicon) Establece una descripción breve del icono.
void	**setTypeicon**(String typeicon) Establece el tipo mime del icono.

Clase Layer. Una capa del sistema, es empleada por el panel de control y el servidor, soporta persistencia y los campos requeridos por el estándar WMS. Un resumen de la clase Layer se muestra en la tabla 12.

Tabla 12. Resumen de la clase Layer.

Resumen de constructores	
Layer() Construye un objeto sin valores.	
Resumen de métodos	
double	**getArea**() Retorna el área que ocupa la capa según su origen de datos. Los cálculos se realizan con base en el bounding box del origen de datos de la capa.
String	**getAtribution**() Retorna el responsable de la capa. De conformidad al estándar WMS.
String[]	**getClassValues**() Retorna un array que contiene un elemento por cada geometría en el origen de datos, se usa para determinar la regla indicada en simbologías de tipo clase.
int	**getCodelaye**() Retorna el código del objeto en el sistema de persistencia.
String	**getDesclaye**() Retorna una descripción breve para mostrar. De conformidad al estándar WMS.
StringBuilder	**getFormatFeatureInfo**(Connection conn, ArrayList features,String INFO_FORMAT, int crs) Retorna una respuesta featureInfo que contiene los features indicados en el formato solicitado.
String	**getKeywordList**() Retorna un listado de palabras clave. De conformidad al estándar WMS.
BufferedImage	**getLegend**() Retorna una imagen con la leyenda de la capa.
double	**getMaxscaledeno**() Retorna el denominador máximo de escala. De conformidad al estándar WMS.
double	**getMinscaledeno**() Retorna el denominador mínimo de escala. De conformidad al estándar WMS.
String	**getNameLaye**() Retorna el identificador de la capa. De conformidad al estándar WMS.
Project	**getProject**()

	Retorna el proyecto al que pertenece la capa.
double[]	**getRangeValues**() Retorna un array que contiene un elemento por cada geometría en el origen de datos, se usa para determinar la regla indicada en simbologías de tipo rango.
Source	**getSource**() Retorna el origen de datos asociado a la capa.
String	**getSummary**() Retorna el resumen de la capa. De conformidad al estándar WMS.
Symbology	**getSymbology**() Retorna la simbología de capa.
double	**getXmax**() Retorna el máximo valor del primer eje del bounding box del origen de datos de la capa.
double	**getXmin**() Retorna el mínimo valor del primer eje del bounding box del origen de datos de la capa.
Element	**getXMLCapability**(Java.io.InputStream schema, int version) Retorna un elemento XML que describe la capa en la versión WMS indicada.
double	**getYmax**() Retorna el máximo valor del segundo eje del bounding box del origen de datos de la capa.
double	**getYmin**() Retorna el mínimo valor del segundo eje del bounding box del origen de datos de la capa.
int	**getZorder**() Retorna el Indicador del orden de la capa dentro del proyecto.
boolean	**isQueryable**() Indica si la capa puede ser consultada por una operación GetFeatureInfo.
void	**loadFeatures**(Java.sql.Connection con) Llama al método de cargar geometrías del origen de datos y carga los valores para simbologías de clase y rango si es necesario.
void	**remove**(HibernateUtil hibernate) Remueve este objeto del almacenamiento persistente.
void	**setAtribution**(String atribution) Establece el responsable de la capa. De conformidad al estándar WMS.
void	**setCodelaye**(int codelaye) Establece el código del objeto en el sistema de persistencia.
void	**setDesclaye**(String desclaye) Establece una descripción breve para mostrar. De conformidad al estándar WMS.

void	**setKeywordList**(String keywordList) Establece una lista de palabras clave que describen la capa. Se trata de una cadena separada por comas. De conformidad al estándar WMS.
	void **setMaxscaledeno**(double maxscaledeno) Establece el denominador máximo de escala. De conformidad al estándar WMS.
	void **setMinscaledeno**(double minscaledeno) Establece De conformidad al estándar WMS, denominador mínimo de escala. De conformidad al estándar WMS.
void	**setProject**(Project project) Establece el proyecto al que pertenece la capa.
void	**setQueryable**(boolean queryable) Indica si la capa puede ser consultada por una operación GetFeatureInfo.
void	**setSource**(Source source) Establece el origen de datos asociado a la capa.
void	**setSummary**(String summary) Establece el resumen de la capa. De conformidad al estándar WMS.
void	**setSymbology**(Symbology symbology) Establece la simbología de capa.
void	**setZorder**(int zorder) Establece el indicador del orden de la capa dentro del proyecto.

Clase Project. Representa un proyecto para su uso en el servidor y en el panel de control, representa un conjunto de capas que comparten un sistema de coordenadas de referencia y un tema común. Un resumen de la clase Project se muestra en la tabla 13.

Tabla 13. Resumen de la clase Project.

Resumen de constructores
Project() Construye un objeto en blanco.

Resumen de métodos	
BoundingBox	**getBoundingBox**() Calcula el bounding box del proyecto con base en los bounding box de la capas que contiene.
int	**getCodeproj**() Retorna el código del objeto en el sistema de persistencia.
String	**getDescproj**() Retorna el nombre que mostrar el proyecto.
Java.util.Set	**getGeocoderConfigs**() Retorna el conjunto de las configuraciones de

	geocoders asociadas al proyecto.
Geocoder[]	**getGeocoders**() Retorna el listado de los geocoder asociados al proyecto, requiere que previa llamada a loadGeocoders().
Java.util.Set	**getLayers**() Retorna el conjunto de capas asociadas al proyecto.
String	**getNameProj**() Retorna el identificador del proyecto.
Service	**getService**() Retorna el servicio al que pertenece el proyecto.
ArrayList<Layer>	**getSortedLayers**() Retorna un listado de las capas del proyecto ordenadas según su Z-order.
Java.util.Set	**getSources**() Retorna el conjunto de los orígenes de datos asociados al proyecto.
Spatial_ref_sys	**getSpatial_ref_sys**() Retorna el sistema de referencia espacial del proyecto.
Element	**getXMLCapability**(Element root, int version) Obtiene un elemento XML que describe el proyecto y sus capas.
void	**loadGeocoders**() Carga los geocoders asignados al proyecto, solicita las operaciones requeridas al cargador de clases del sistema, requiere que los plugins de los geocoders estén presentes.
void	**setCodeproj**(int codeproj) Establece el código del objeto en el sistema de persistencia.
void	**setDescproj**(String descproj) Establece el nombre que mostrar el proyecto.
void	**setGeocoderConfigs**(Java.util.Set geocoderConfigs) Establece el conjunto de las configuraciones de geocoders asociadas al proyecto.
void	**setLayers**(Java.util.Set layers) Establece el conjunto de capas asociadas al sistema.
void	**setService**(Service service) Establece el servicio al que pertenece el proyecto.
void	**setSources**(Java.util.Set sources) Establece el conjunto de los orígenes de datos asociados al proyecto.
void	**setSpatial_ref_sys**(Spatial_ref_sys spatial_ref_sys) Establece el sistema de referencia espacial del

	proyecto.

Clase Rule. Define una regla para aplicar un estilo a gráfico a un registro en particular de una capa. La regla puede ser un intervalo o un valor exacto. Un resumen de la clase Rule se muestra en la tabla 14.

Tabla 14. Resumen de la clase Rule.

Resumen de constructores	
Rule() Construye un objeto en blanco.	
Resumen de métodos	
byte[]	**getBinarule**() Retorna el campo binario que contiene la imagen que debe usarse en caso de simbologías para capas de puntos.
int	**getCoderule**() Retorna el código del objeto en el sistema de persistencia.
String	**getDescrule**() Retorna una descripción para la regla.
double	**getMax**() Si la regla pertenece a un modelo de rangos, representa el valor máximo del intervalo.
double	**getMin**() Si la regla pertenece a una simbología de rangos, representa el valor mínimo del intervalo.
Style	**getStyle**() Retorna el estilo que debe aplicarse al registro en caso de ser afectado por la regla.
Symbology	**getSymbology**() Retorna la simbología a la que pertenece la regla.
String	**getValue**() Retorna el valor en caso de que la regla pertenezca a una simbología de clases.
void	**setBinarule**(byte[] binarule) Establece el campo binario que contiene la imagen que deben usarse en caso de simbologías para capas de puntos.
void	**setCoderule**(int coderule) Establece el código del objeto en el sistema de persistencia.
void	**setDescrule**(String descrule) Establece el una descripción para la regla.
void	**setMax**(double max) Si la regla pertenece a un modelo de rangos, representa el valor máximo del intervalo.
void	**setMin**(double min) Si la regla pertenece a una simbología de rangos, representa el valor mínimo del intervalo.

void	**setStyle**(Style style) Establece el estilo que debe aplicarse al registro en caso de ser afectado por la regla.
void	**setSymbology**(Symbology symbology) Establece la simbología a la que pertenece la regla.
void	**setValue**(String value) Establece el valor en caso de que la regla pertenezca a una simbología de clases.

Clase Service. Representa un servicio WMS. Es la raíz de la estructura de clases del sistema. Un resumen de la clase Service se muestra en la tabla 15.

Tabla 15. Resumen de la clase Service.

Resumen de campos	
String	**SIGNATURE** Firma del servidor.
static int	**VERSION_1_1_0** Versión 1.1.0 del estándar WMS .
static int	**VERSION_1_1_1** Versión 1.1.1 del estándar WMS .
static int	**VERSION_1_3_0** Versión 1.3.0 del estándar WMS .
static int	**VERSION_JSON** Versión JSON del documento de capabilities.
Resumen de constructores	
Service() Construye un objeto en blanco.	
Resumen de métodos	
int	**getCodeserv**() Retorna el identificador del objeto en el sistema de persistencia.
String	**getContactElectronicMailAddress**() Retorna el correo electrónico del encargado del servicio.
String	**getContactOrganization**() Retorna la organización encargada del servicio.
String	**getContactPerson**() Retorna la persona encargada del servicio.
Java.util.Set	**getIcons**() Retorna el conjunto de iconos del servicio.
String	**getKeywordList**() Retorna el conjunto de palabras clave.
String	**getLayerLimit**() Retorna el número máximo de capas por petición.
String	**getMaxHeight**() Retorna el alto máximo de la imagen por petición.
String	**getMaxWidth**()

	Retorna el ancho máximo de la imagen por petición.
Set	**getProjects**() Retorna el conjunto de proyectos del servicio.
String	**getProviderURL**() Retorna la URL con información del servicio.
static Service	**getService**(HibernateUtil hibernate) Retorna el objeto Service del sistema desde una conexión hibernate.
String	**getServiceURL**() Retorna la URL del servicio WMS.
String	**getSummary**() Retorna una descripción del servicio.
String	**getTitle**() Retorna una descripción del breve del servicio.
String	**getWMSCapabilities**(int version) Retorna el documento de capabilities del sistema según la versión .
String	**getWMSJSON**() Retorna el documento de capabilities en formato JSON.
boolean	**isPublicaccess**() Determina si el servicio tiene acceso al público.
void	**sendForceReload**(ServiceConnection sc) Envía al servidor la señal que indica que deben recargase todos los buffers desde la base de datos.
void	**setCodeserv**(int codeserv) Establece el identificador del objeto en el sistema de persistencia.
void	**setContactElectronicMailAddress**(String contactElectronicMailAddress) Establece el correo electrónico del encargado del servicio.
void	**setContactOrganization**(String contactOrganization) Establece la organización encargada del servicio.
void	**setContactPerson**(String contactPerson) Establece la persona encargada del servicio.
void	**setIcons**(Java.util.Set icons) Establece el conjunto de iconos del servicio.
void	**setKeywordList**(String keywordList) Establece el conjunto de palabras clave.
void	**setLayerLimit**(String layerLimit) Establece el número máximo de capas por petición.
void	**setMaxHeight**(String maxHeight) Establece el alto máximo de la imagen por petición.
void	**setMaxWidth**(String maxWidth) Establece el ancho máximo de la imagen por petición.
void	**setProjects**(Java.util.Set projects) Establece el conjunto de proyectos del servicio.
void	**setProviderURL**(String providerURL) Establece la URL con información del servicio.

void	**setPublicaccess**(boolean publicaccess) Establece si el servicio tiene acceso al público.
void	**setServiceURL**(String serviceURL) Establece la URL del servicio WMS.
void	**setSummary**(String summary) Establece la descripción del servicio.
void	**setTitle**(String title) Establece la descripción del breve del servicio.
void	**setWMSCapabilities**(String WMSCapabilities, int version) Establece el documento de capabilities del sistema según la versión .
void	**setWMSJSON**(String WMSJSON) Establece el documento de capabilities en formato JSON.
void	**updateCapabilities**(HibernateUtil hibernate, ServiceConnection sc) Actualiza los capabilities del servicio en todas sus versiones.

Clase Source. Representa un origen de datos del sistema, una relación de objetos geométricos y datos alfanuméricos asociados. Un resumen de la clase Service se muestra en la tabla 16.

Tabla 16. Resumen de la clase Source.

Resumen de constructores	
Source () Construye un objeto sin valores	
Resumen de métodos	
Void	**calcularDimensiones**() Calcula el bounding box del origen de datos dada su geometría.
Envelope[]	**getBboxes**() Retorna un array con los bounding box de cada geometría en el origen.
int	**getCodesour**() Retorna el código del objeto en el sistema de persistencia.
Feature[]	**getFeatures**() Retorna un array con los objetos geométricos del origen.
List< String>	**getFieldNames**() Retorna un listado de los nombres de los datos alfanuméricos que acompañan a los objetos geométricos .
List< Integer>	**getFieldTypes**() Retorna un listado de los tipos de los datos alfanuméricos que acompañan a los objetos geométricos. Según tipos SQL de Java.
String	**getFielsour**() Retorna el nombre del campo en la relación que guarda la geometría del origen.

Java.lang.Class	**getGeomClass**() Retorna la clase geométrica del origen.
String	**getGeomType**() Retorna el nombre del tipo de la geometría de origen.
Java.util.Set	**getLayers**() Retorna las capas en las que se usa el origen.
Project	**getProject**() Retorna el proyecto al que pertenece el origen.
String	**getTablsour**() Retorna el nombre de la relación que guarda los datos del origen.
double	**getXmax**() Retorna el máximo valor en el primer eje.
double	**getXmin**() Retorna el mínimo valor en el primer eje.
double	**getYmax**() Retorna el máximo valor en el segundo eje.
double	**getYmin**() Retorna el mínimo valor en el segundo eje.
void	**loadFeatures**(Java.sql.Connection con) Carga en memoria los objetos geométricos del origen de datos.
void	**loadFieldMetadata**(Java.sql.Connection conn) Carga los metadatos del origen, consistente en los nombres y tipos de los campos alfanuméricos que acompañan a los objetos geométricos.
void	**setBboxes**(com.vividsolutions.jts.geom.Envelope[] bboxes) Establece el array con los bounding box de cada geometría en el origen.
void	**setCodesour**(int codesour) Establece el código del objeto en el sistema de persistencia.
void	**setFeatures**(Feature[] features) Establece el array con los objetos geométricos del origen.
void	**setFielsour**(String fielsour) Establece el nombre del campo en la relación que guarda la geometría del origen.
void	**setGeomType**(String geomType) Establece el nombre del tipo de la geometría de origen.
void	**setLayers**(Java.util.Set layers) Establece las capas en las que se usa el origen.
void	**setProject**(Project project) Establece el proyecto al que pertenece el origen.
void	**setTablsour**(String tablsour) Establece el nombre de la relación que guarda los datos del origen.
void	**setXmax**(double xmax)

	Establece el máximo valor en el primer eje.
void	**setXmin**(double xmin) Establece el mínimo valor en el primer eje.
void	**setYmax**(double ymax) Establece el máximo valor en el segundo eje.
void	**setYmin**(double ymin) Establece el mínimo valor en el segundo eje.

Clase Spatial_ref_sys. Sistema de referencia espacial según el EPSG. Un resumen de la clase Spatial_ref_sys se muestra en la tabla 17.

Tabla 17. Resumen de la clase Spatial_ref_sys.

Resumen de constructores	
Spatial_ref_sys() Construye un objeto sin valores.	
Resumen de métodos	
CoordinateReferenceSystem	**getCRS**() Retorna un CoordinateReferenceSystem con base en este objeto.
int	**getSrid**() Retorna el código del sistema espacial de referencia según el EPSG.
String	**getSrtext**() Retorna la descripción WKT del sistema espacial de referencia según el OGC.
void	**setSrid**(int srid) Establece el código del sistema espacial de referencia según el EPSG.
void	**setSrtext**(String srtext) Establece la descripción WKT del sistema espacial de referencia según el OGC.

Clase Style. Define un estilo que se aplica al momento de renderizar una geometría en caso de que cumpla una regla en particular. Un resumen de la clase Style se muestra en la tabla 18.

Tabla 18. Resumen de la clase Style.

Resumen de constructores	
Style() Construye un objeto sin valores.	
Resumen de métodos	
int	**getCodestyl**() Retorna el código del objeto en el sistema de persistencia.
static Style	**getDefaultRandomStyle**(Icon icon) Genera un estilo aleatorio que llevan las capas por defecto.
AtlasColor	**getFillcolo**() Retorna el color de relleno para la geometría.
Icon	**getIcon**() Retorna el icono del estilo.
BufferedImage	**getIconImage**() Renderiza la imagen del icono del estilo.
AtlasColor	**getLinecolo**() Retorna el color de línea.
AtlasStroke	**getLinestyl**() Retorna el estilo de línea.

void	**setCodestyl**(int codestyl) Establece el código del objeto en el sistema de persistencia.
void	**setFillcolo**(AtlasColor fillcolo) Establece el color de relleno para la geometría.
void	**setIcon**(Icon icon) Establece el icono del estilo.
void	**setLinecolo**(AtlasColor linecolo) Establece el color de línea.
void	**setLinestyl**(AtlasStroke linestyl) Establece el estilo de línea.

Clase Symbology. Representa una simbología que determina los estilos y reglas que se aplicarán al renderizar una capa. Un resumen de la clase Symbology se muestra en la tabla 19.

Tabla 19. Resumen de la clase Symbology.

Resumen de campos	
static int	**TYPE_CLASS_SYMB** Simbología por clases.
static int	**TYPE_FIXED_SYMB** Simbología fija.
static int	**TYPE_RANGE_SYMB** Simbología por rango.
Resumen de constructores	
Symbology() Construye un objeto con valores vacíos.	
Resumen de métodos	
void	**delete**(org.hibernate.classic.Session sess) Elimina este objeto del almacenamiento persistente.
int	**getCodesymb**() Retorna el identificador del objeto en el modelo de persistencia.
String	**getFieldName**() Retorna el nombre del campo en el origen de datos que se usará para determinar la aplicación de las reglas.
Layer	**getLayer**() Retorna la capa a la que pertenece la simbología.
Java.util.Set	**getRules**() Retorna el conjunto de reglas de la simbología.
int	**getTypeSymb**() Retorna el tipo de la simbología, según las constantes de la clase.

	void **save**(org.hibernate.classic.Session sess) Almacena este objeto del almacenamiento persistente.
void	**setCodesymb**(int codesymb) Establece el identificador del objeto en el modelo de persistencia.
void	**setFieldName**(String fieldName) Establece el nombre del campo en el origen de datos que se usará para determinar la aplicación de las reglas.
void	**setLayer**(Layer layer) Establece la capa a la que pertenece la simbología.
void	**setRules**(Java.util.Set rules) Establece el conjunto de reglas de la simbología.
void	**setTypeSymb**(int typeSymb) Establece el tipo de la simbología, según las constantes de la clase.

1.6.2. Paquete util

El paquete contiene clases de utilidades empleadas en diferentes lugares del sistema. En la tabla 20 se listan y describen las clases de paquete útil.

Tabla 20. Resumen del paquete util.

Resumen de clases	
ColorsList	Permite la generación de colores aleatorios.
Configuration	Permite la lectura y escritura del XML de la configuración del panel de control.
ConnectionProperties	Propiedades necesarias para establecer conexión entre el panel de control y un servidor.
CoordinateUtils	Utilidades para la conversión entre sistemas de coordenadas.
Cryptography	Utilidades para encriptar y des encriptar datos en AES.
DataBaseUtils	Utilidades para conexión directa con la base de datos.
EWKB	Conversión al formato EWKB.
FrmWait	Formulario modal de espera.
HibernateUtil	Utilidades para creación y mantenimiento de sesiones hibernate.
IOUtils	Utilidades para trabajo con flujos.
PostGis	Utilidades para detectar la instalación de postgis.
ServiceConnection	Objeto que representa una conexión entre el servidor y un panel de control.
ShapeFileUtils	Utilidades para importación de shapefiles.
WindowUtils	Utilidades para el manejo de ventanas y diálogos.

1.7. MODULO PANEL DE CONTROL

1.7.1. Paquete windows.configuration.

Contiene diálogos para la configuración de la herramienta.

Clase FrmConfiguration. Permite establecer opciones de la herramienta como el idioma de la interfaz, el look and feel e iniciar el administrador de plugins. La apariencia de esta clase se muestra en la figura 2.

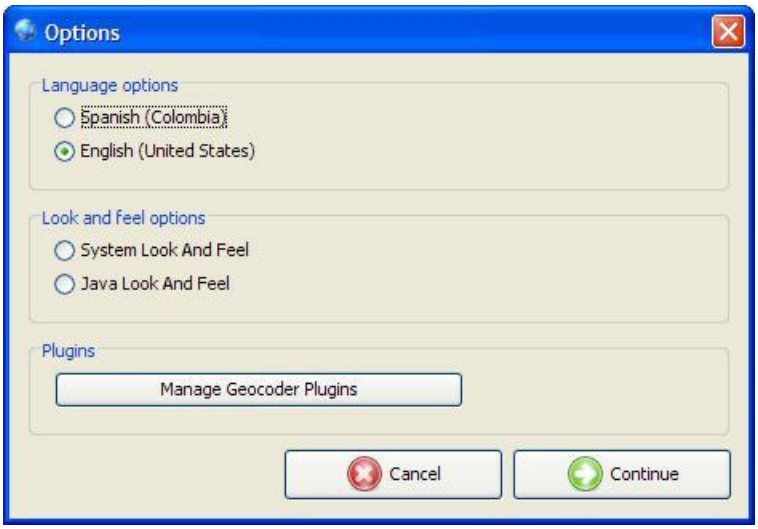

Figura 2. Apariencia de la clase FrmConfiguration

Clase FrmManageGeocoderPlugins. Permite agregar y eliminar plugins de geocodificación para su uso en el resto de la herramienta. La apariencia de esta clase se muestra en la figura 3.

.

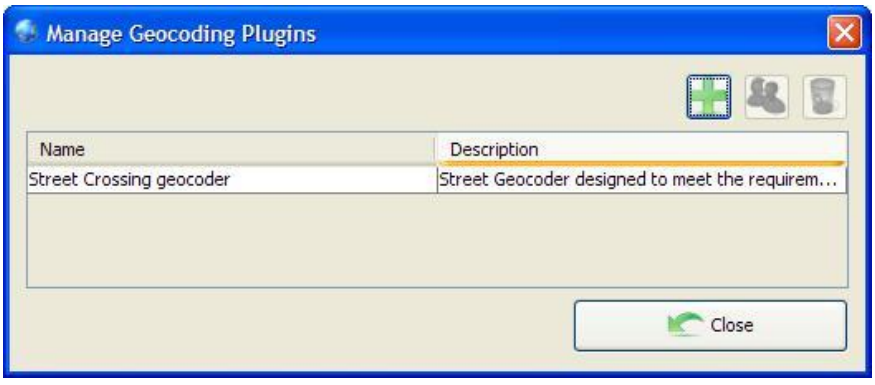

Figura 3. Apariencia de la clase FrmManageGeocoderPlugins

1.7.2. Paquete windows.connections

Contiene diálogos para la establecer conexión con un servidor.

Clase FrmConnectionProps. Permite establecer los datos de conexión con un servidor así como almacenar datos de conexiones previamente realizadas. La apariencia de esta clase se muestra en la figura 4.

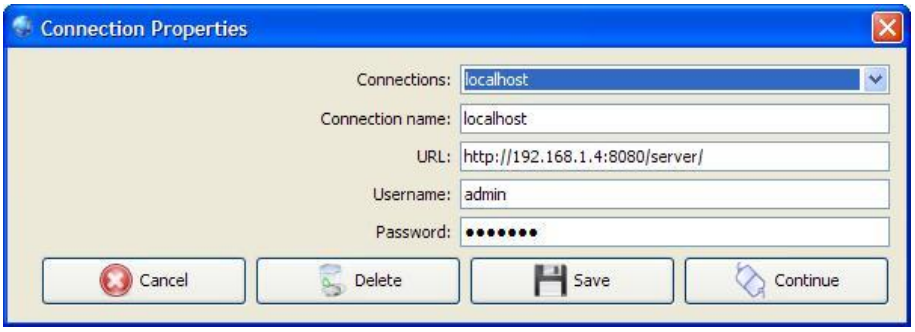

Figura 4. Apariencia de la clase FrmConnectionProps

1.7.3. Paquete windows.controls.

Contiene los componentes que conforman la ventana principal del la aplicación.

Clase PnlProjects. Lista los proyectos en el servidor seleccionado, permite agregar, eliminar y modificar proyectos, así como llamar a los diálogos de administrar geocoders y orígenes de datos para un proyecto en particular. La apariencia de esta clase se muestra en la figura 5.

Figura 5. Apariencia de la clase PnlProjects

Clase PnlLayers. Lista las capas para un proyecto particular, permite agregar, eliminar y modificar capas, así como modificar el orden y llamar a los diálogos de administrar simbologías, denominadores de escala y etiquetas, así como visualizar las leyendas de cada capa. La apariencia de esta clase se muestra en las figuras 6 y 7.

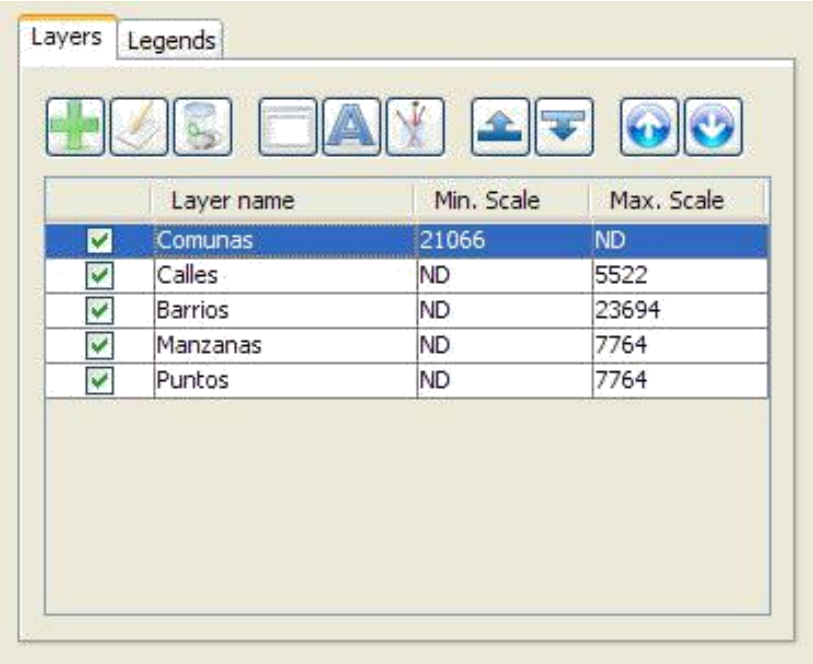

Figura 6. Apariencia de la clase PnlLayers en pestaña de capas

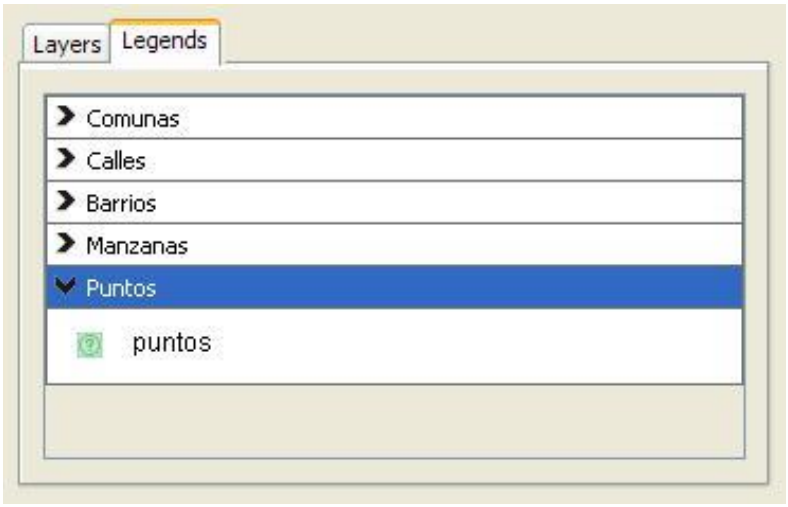

Figura 7. Apariencia de la clase PnlLayers en pestaña de leyendas

Clase PnlViewer. Permite navegar por el mapa del proyecto actual, visualizando los resultados de las modificaciones efectuadas, para ello utiliza la biblioteca de componentes de desarrollo. Realiza operaciones de zoom, desplazamiento, consultas a FeatureInfo y consultas al geocoder del proyecto actual. La apariencia de esta clase se muestra en la siguiente figura 8.

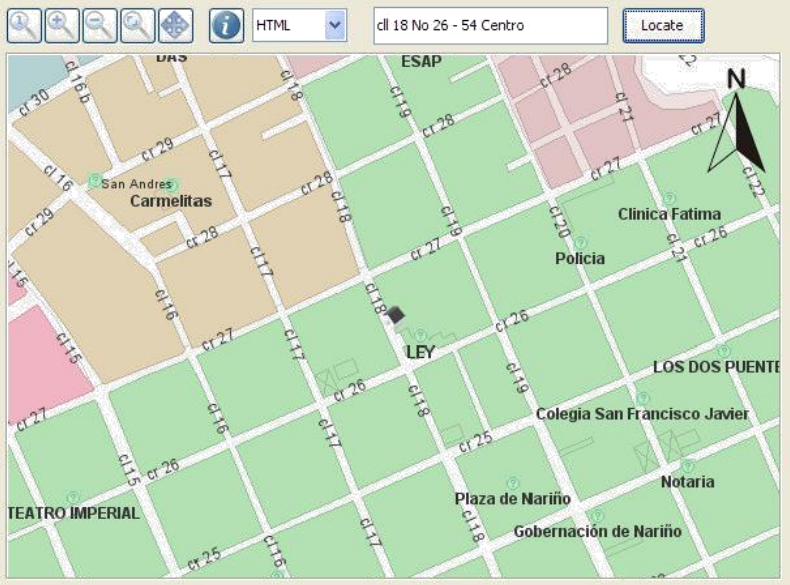

Figura 8. Apariencia de la clase PnlViewer

Clase AtlasColorChooser. Componente que permite la sección de un color con canal alfa. La apariencia de esta clase se muestra en la figura 9.

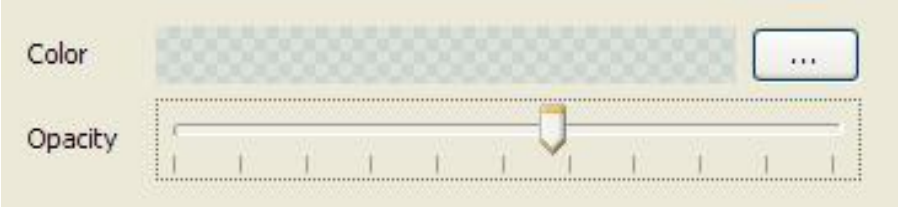

Figura 9. Apariencia de la clase AtlasColorChooser

1.7.4. Paquete windows.fonts.

Contiene los formularios para configurar las etiquetas aplicadas a las capas.

Clase FrmFonts. Formulario que permite ajustar las opciones de etiquetado para una capa en particular. La apariencia de esta clase se muestra en figura 10.

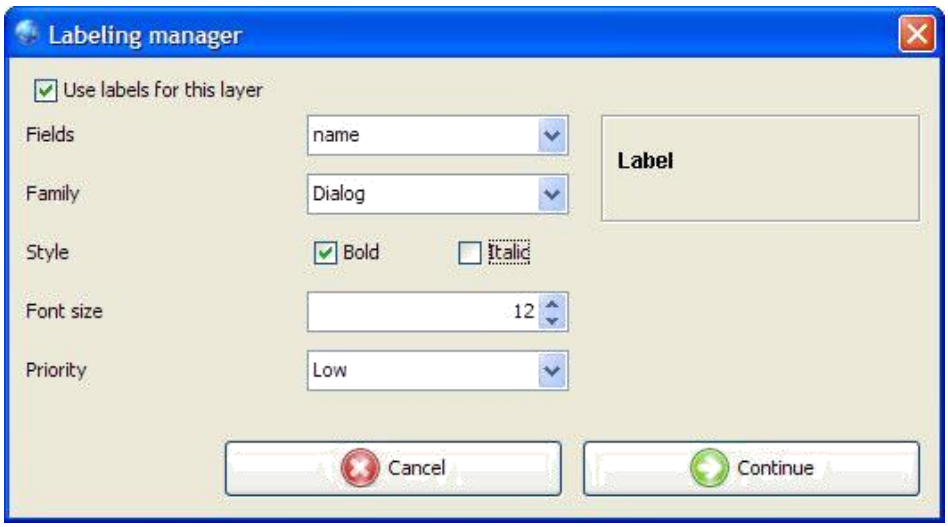

Figura 10. Apariencia de la clase FrmFonts

1.7.5. Paquete windows.geocoding.

Contiene los formularios para configurar los geocoders para un proyecto.

Clase FrmAdminGeocoders. Lista los geocoders disponibles, permite agregarlos y eliminarlos del proyecto así como configurar geocoders y consultar los cuadros de diálogo de ayuda. La apariencia de esta clase se muestra en la figura 11.

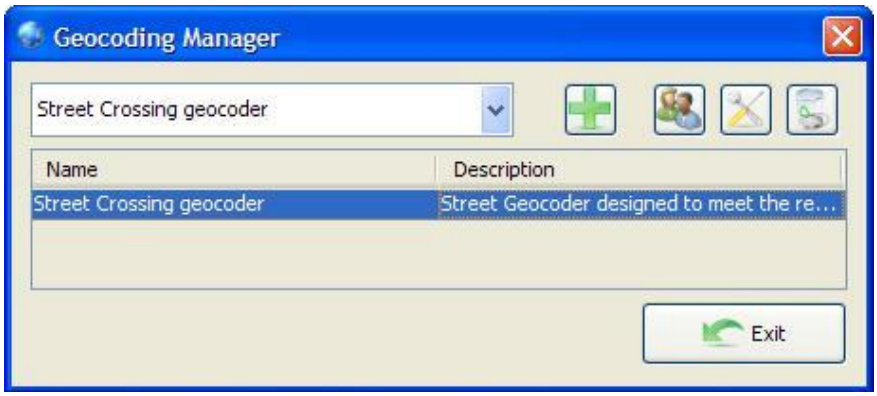

Figura 11. Apariencia de la clase FrmAdminGeocoders

1.7.6. Paquete windows.icons.

Contiene los diálogos y componentes para el trabajo con iconos.

Clase FrmAddIcons. Lista los archivos SVG contenidos en un directorio que se pueden usar como iconos, permite seleccionar aquellos que se van importar a la biblioteca. La apariencia de esta clase se muestra en la figura 12.

Figura 12. Apariencia de la clase FrmAddIcons

Clase FrmAdminIcons. Lista los iconos actuales de la biblioteca del sistema y permite las operaciones de agregar y eliminar. La apariencia de esta clase se muestra en la figura 13.

Figura 13. Apariencia de la clase FrmAdminIcons

Clase IconsPanel. Componente que permite mostrar un listado de vistas en miniatura de varios iconos, permite operaciones de selección, adición y eliminación. La apariencia de esta clase se muestra en la figura 14.

Figura 14. Apariencia de la clase IconsPanel

1.7.7. Paquete windows.layers.

Contiene los diálogos para trabajar con capas.

Clase FrmLayerProps. Permite establecer las propiedades de una capa de conformidad con el estándar WMS. El formulario permite crear capas nuevas así como editar la información de capas ya existentes. La apariencia de esta clase se muestra en la figura 15.

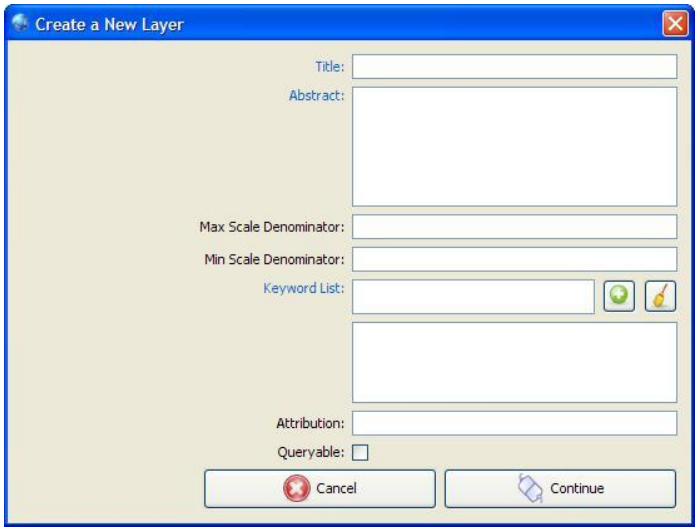

Figura 15. Apariencia de la clase FrmLayerProps

1.7.8. Paquete windows.projects.

Contiene los diálogos para trabajar con proyectos.

Clase FrmNewProyect. Permite establecer las propiedades de un proyecto de conformidad con el estándar WMS. El formulario permite crear proyectos nuevos así como editar la información de proyectos ya existentes. La apariencia de esta clase se muestra en la figura 16.

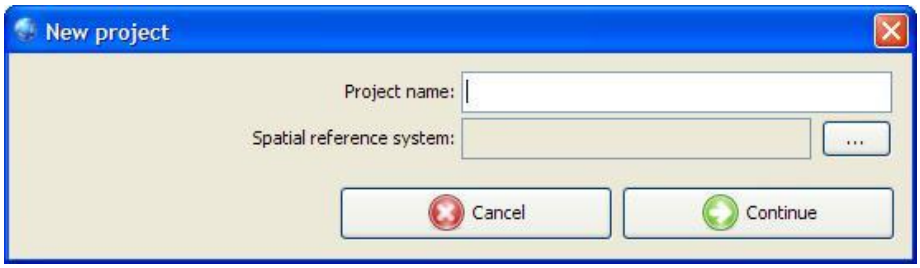

Figura 16. Apariencia de la clase FrmNewProyect

1.7.9. Paquete windows.service.

Contiene los diálogos para trabajar con el servicio.

Clase FrmServiceProps. Permite establecer las propiedades de un servicio de conformidad con el estándar WMS. La apariencia de esta clase se muestra en la figura 17.

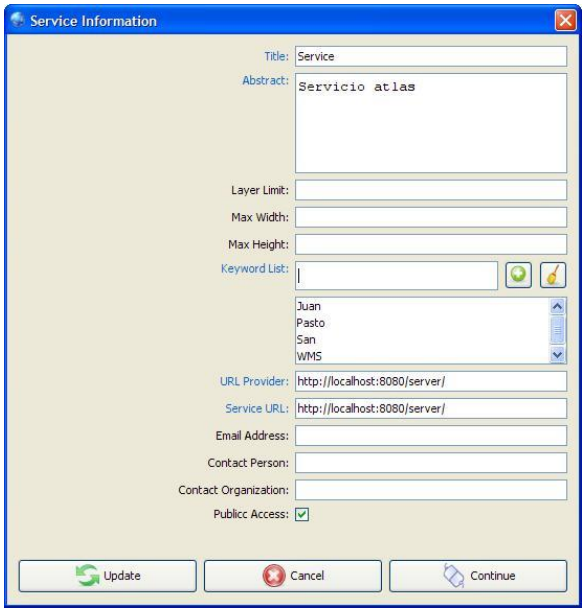

Figura 17. Apariencia de la clase FrmServiceProps

1.7.10. Paquete windows.servicetest.

Contiene los diálogos para realizar diferentes pruebas sobre el estado del servicio.

Clase FrmServerTest. Permite realizar diferentes pruebas sobre el estado de servidor, ofrece herramientas automáticas para corregir varias de estas situaciones así como documentación y sugerencias para corregir otras. La apariencia de esta clase se muestra en figura 18.

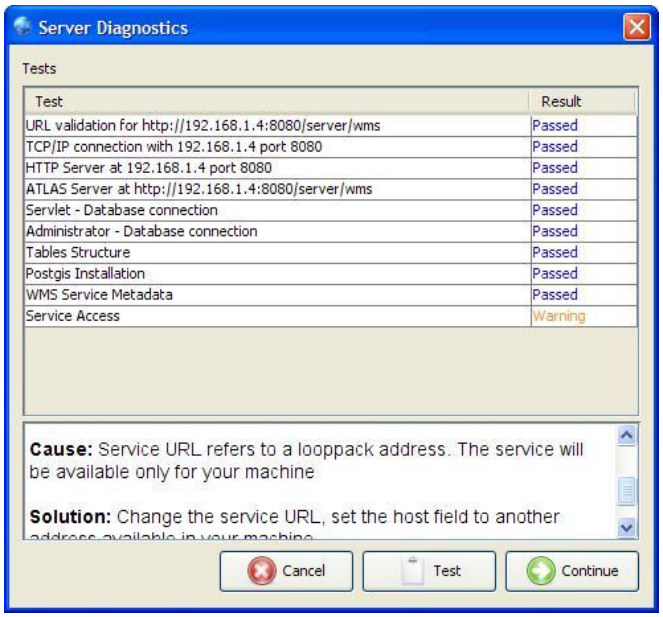

Figura 18. Apariencia de la clase FrmServerTest

1.7.11. Paquete windows.sources.

Contiene los diálogos para realizar tareas concernientes a orígenes de datos.

Clase FrmAddPostgis. Permite agregar un origen de datos desde postgis, para ello revisa las tablas estándar de tal extensión en busca de relaciones que contengan datos geométricos. La apariencia de esta clase se muestra en la figura 19.

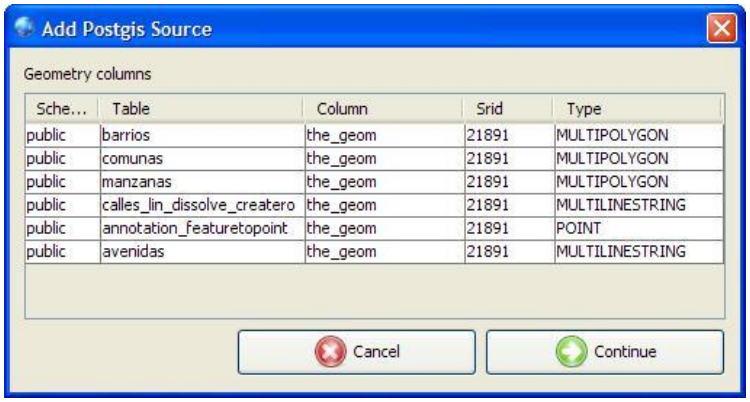

Figura 19. Apariencia de la clase FrmAddPostgis

Clase FrmAddShapeFile. Permite agregar un origen de datos desde un archivo shapefile de ESRI. Inicialmente presenta el diálogo de selección de archivo y luego presenta los campos

alfanuméricos hallados en el archivo para que el usuario seleccione cuáles de ellos desea importar. La apariencia de esta clase se muestra en la figura 20.

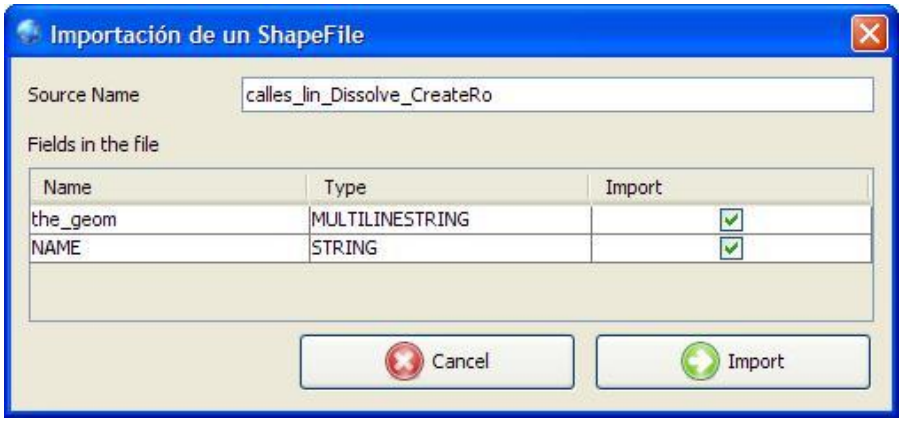

Figura 20. Apariencia de la clase FrmAddShapeFile

Clase FrmLayerSrc. Permite seleccionar el origen de datos que será usado por la capa. La apariencia de esta clase se muestra en la figura 21.

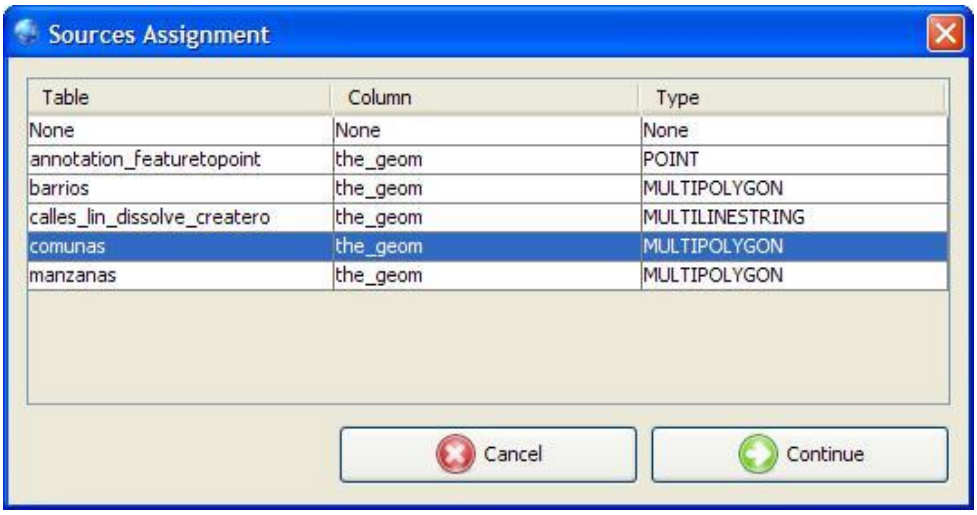

Figura 21. Apariencia de la clase FrmLayerSrc

Clase FrmSources. Permite administrar los orígenes de datos para un proyecto. Permite las operaciones de eliminar orígenes y agregarlos desde postgis o desde un archivo Shapefile de ESRI. La apariencia de esta clase se muestra en la figura 22.

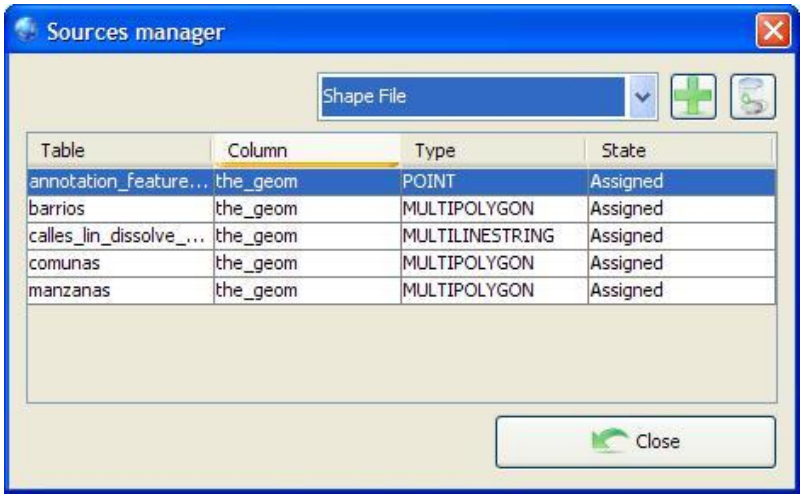

Figura 22. Apariencia de la clase FrmSources

1.7.12. Paquete windows.spatial_ref_sys.

Contiene los diálogos trabajar con sistemas de referencia espacial.

Clase FrmSelectRefSys. Permite seleccionar un sistema de coordenadas de referencia desde un archivo XML contenido en la aplicación. El usuario puede filtrar la lista presentada al escribir en una caja de texto. La apariencia de esta clase se muestra en la figura 23.

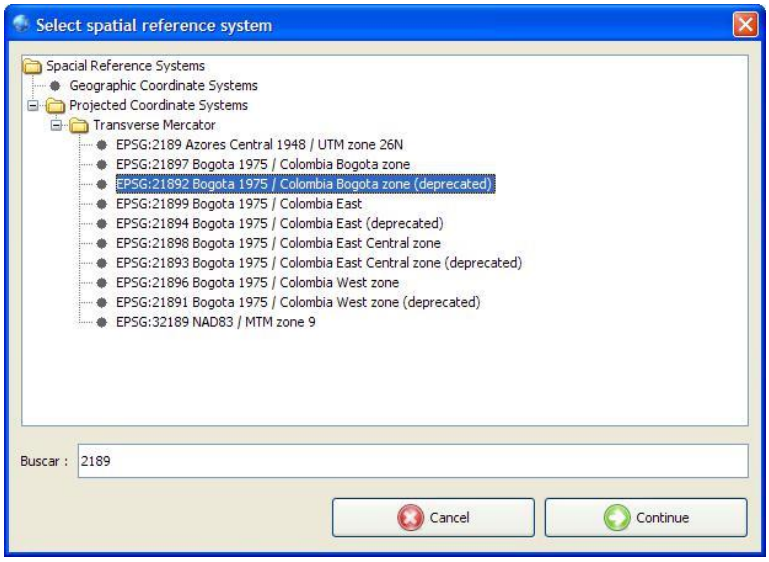

Figura 23. Apariencia de la clase FrmSelectRefSys

1.7.13. Paquete windows.style.

Contiene los componentes necesarios para trabajar con estilos, estos componentes se integran en formularios que permiten la definición de simbologías.

Clase PnlEditLineStyl. Permite la edición de estilos para líneas, notifica de los cambios en el estilo atreves de escuchadores. La apariencia de esta clase se muestra en la figura 24.

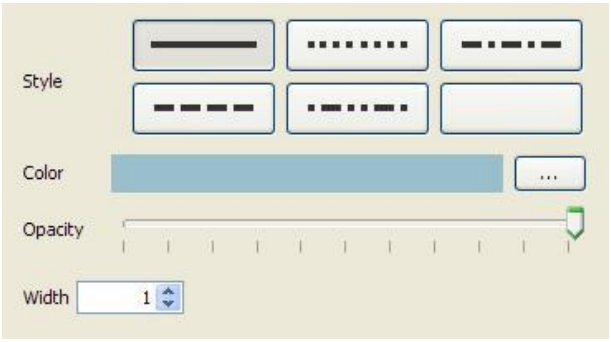

Figura 24. Apariencia de la clase PnlEditLineStyl

Clase PnlEditPointStyle. Permite la edición de estilos para puntos, notifica de los cambios en el estilo atreves de escuchadores. Permite seleccionar un icono desde la biblioteca de iconos del sistema, personalizar su tamaño, color de borde y de relleno. La apariencia de esta clase se muestra en la figura 25.

Figura 25. Apariencia de la clase PnlEditPointStyle

Clase PnlEditPolygonStyle. Permite la edición de estilos para polígonos, notifica de los cambios en el estilo atreves de escuchadores. Permite seleccionar un estilo y color de borde, así como un color de relleno. La apariencia de esta clase se muestra en la figura 26.

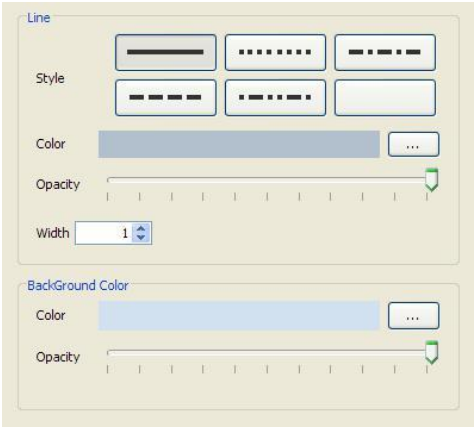

Figura 26. Apariencia de la clase PnlEditPolygonStyle

Paquete windows.symbology.

Contiene los formularios y componentes para el trabajo con simbologías de puntos, líneas y polígonos. Utiliza los componentes del paquete windows.style

Clase ClassRuleGenerator. Permite la generación de reglas de simbología, formando clases dado un campo del origen de datos de la capa. Luego, de ser necesario, entregará el estilo de cada regla generada al componente de la clase Windows.style que corresponda según el tipo de geometría. La apariencia de esta clase se muestra en la figura 27.

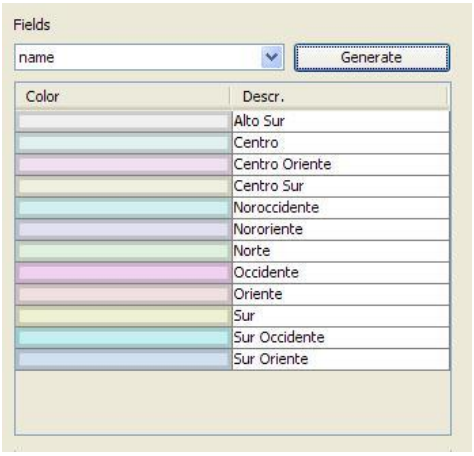

Figura 27. Apariencia de la clase ClassRuleGenerator

Clase RangeRuleGenerator. Permite la generación de reglas de simbología formando rangos dado un campo del origen de datos de la capa y el número deseado de clases. Luego, de ser necesario, entregará el estilo de cada regla generada al componente de la clase Windows.style que corresponda. La apariencia de esta clase se muestra en la figura 28.

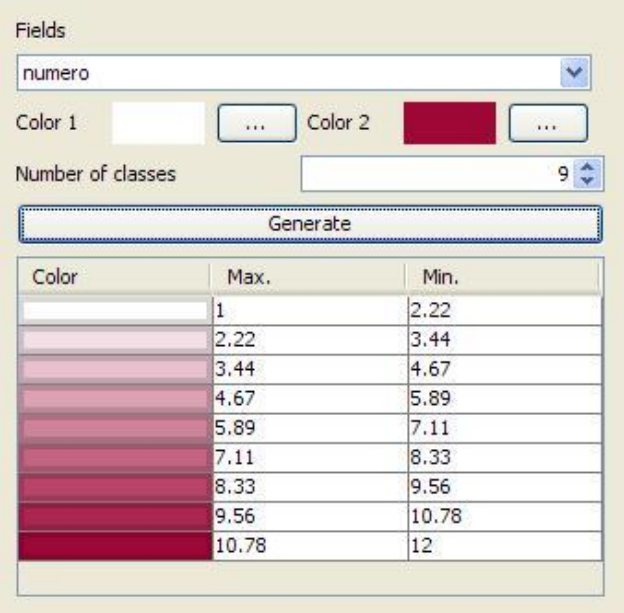

Figura 28. Apariencia de la clase RangeRuleGenerator

Clase FixedRuleGenerator. Permite la generación una regla para simbología que se aplicará a todos los elementos de la capa. La apariencia de esta clase se muestra en la figura 29.

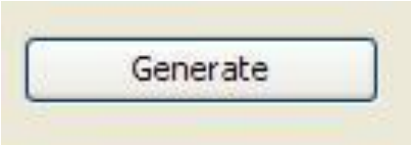

Figura 29. Apariencia de la clase FixedRuleGenerator

Clase FrmSymbLine. Permite la administración de simbología para capas con orígenes de datos tipo línea. La apariencia de esta clase se muestra en las figuras 30, 31 y 32.

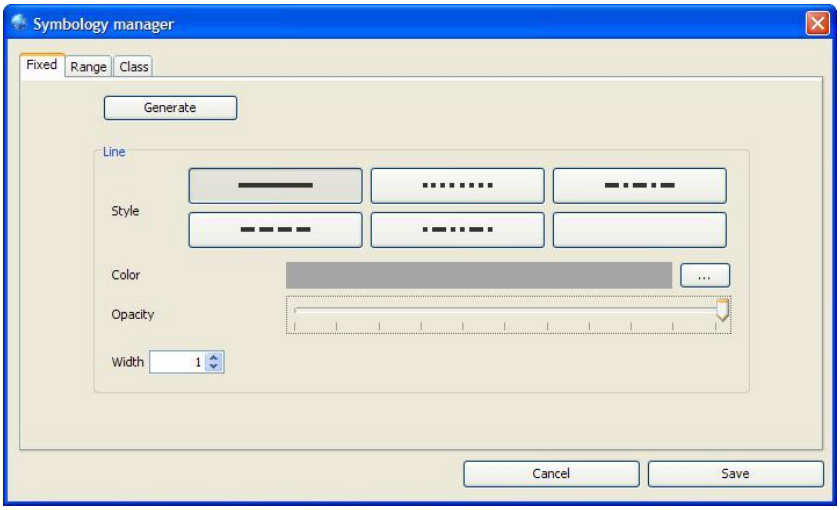

Figura 30. Apariencia de la clase FrmSymbLine con simbología fija

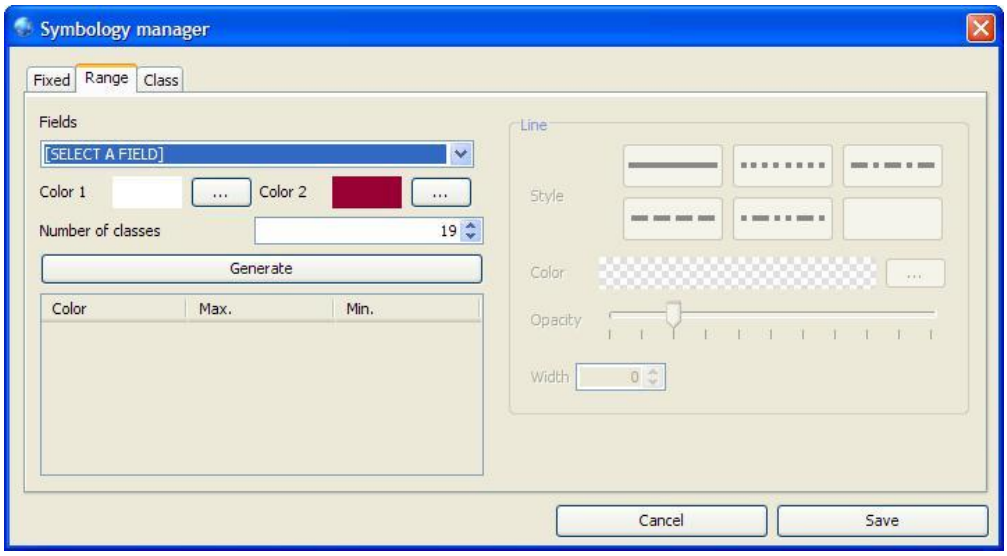

Figura 31. Apariencia de la clase FrmSymbLine con simbología por rangos

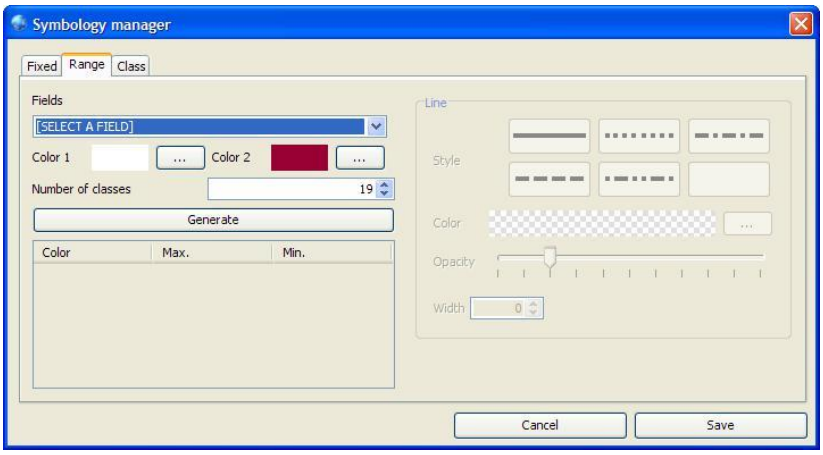

Figura 32. Apariencia de la clase FrmSymbLine con simbología por clases

Clase FrmSymbPoint. Permite la administración de simbología para capas con orígenes de datos tipo línea. La apariencia de esta clase se muestra en las figuras 33, 34 y 35.

.

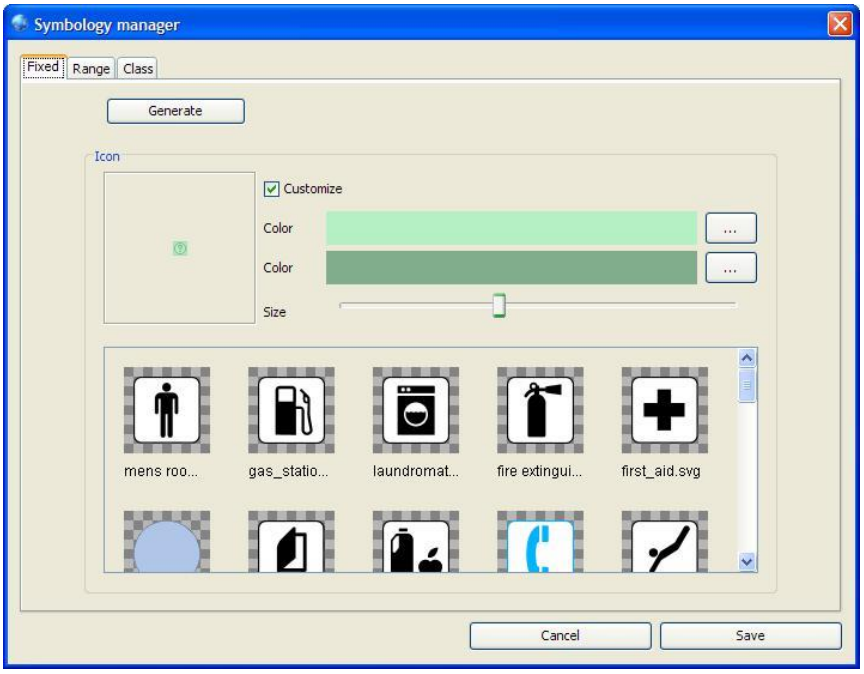

Figura 33. Apariencia de la clase FrmSymbPoint con simbología fija

.

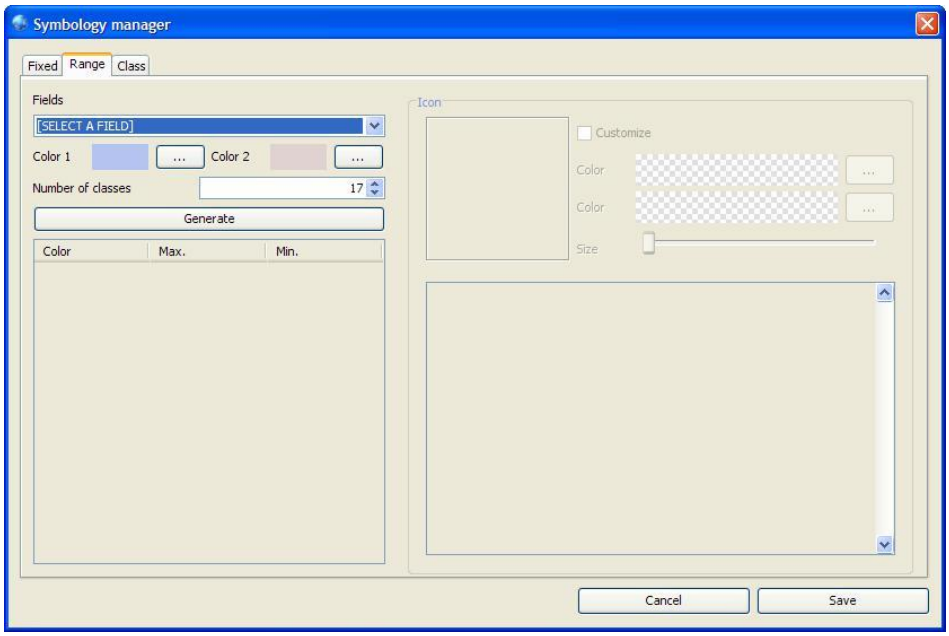

Figura 34. Apariencia de la clase FrmSymbPoint con simbología por rangos

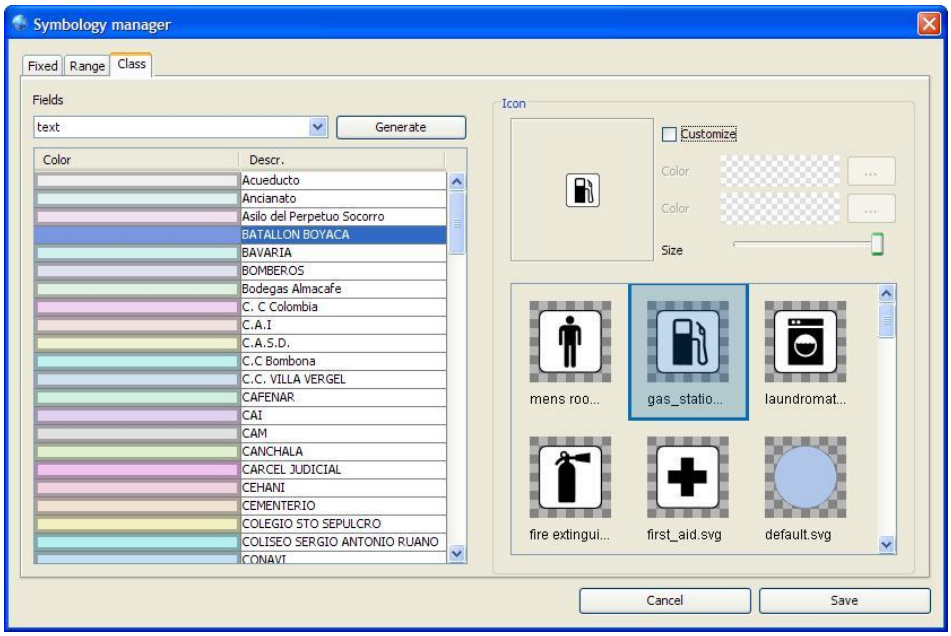

Figura 35. Apariencia de la clase FrmSymbPoint con simbología por clases

Clase FrmSymbPoly. Permite la administración de simbología para capas con orígenes de datos tipo polígono. La apariencia de esta clase se muestra en las figuras 36, 37 y 38.

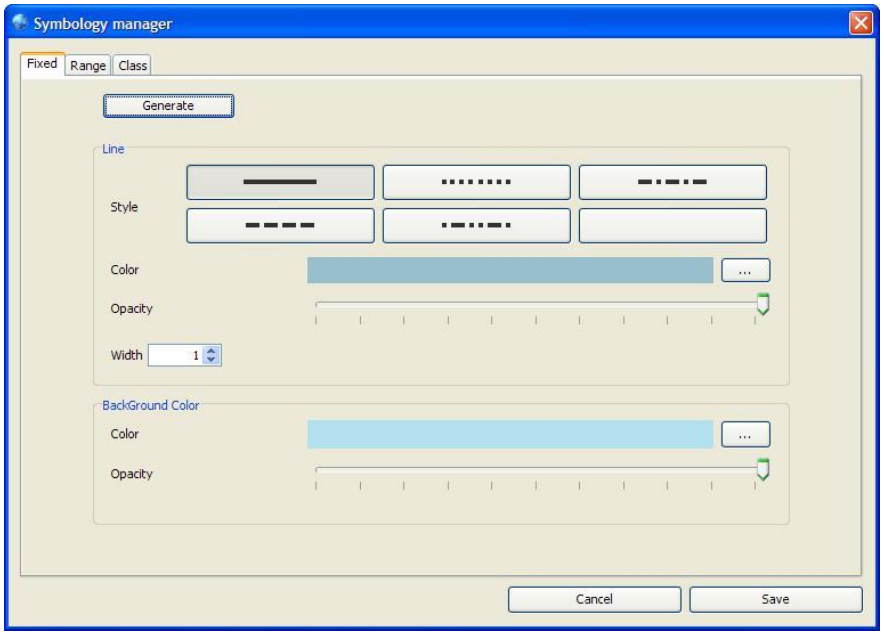

Figura 36. Apariencia de la clase FrmSymbPoly con simbología fija

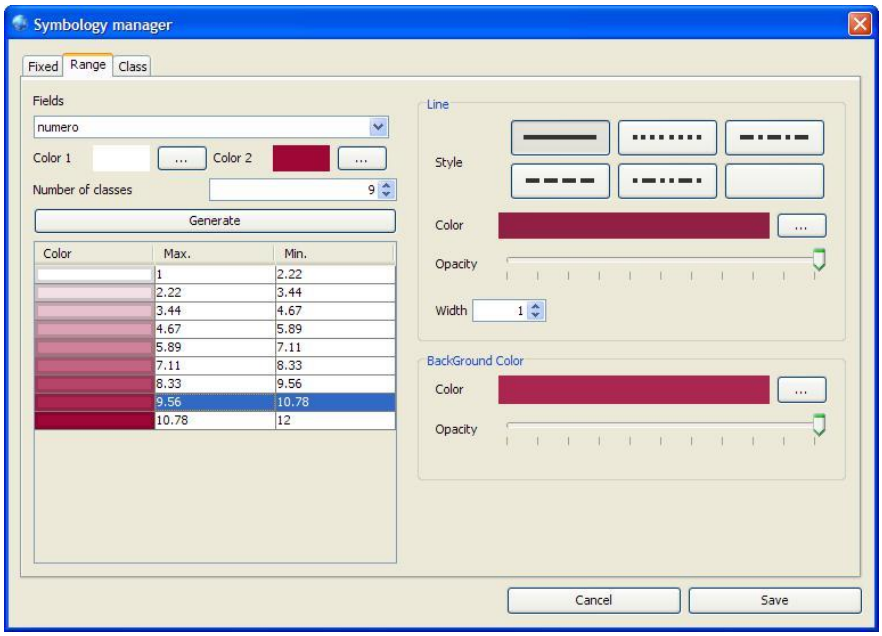

Figura 37. Apariencia de la clase FrmSymbPoly con simbología por rangos

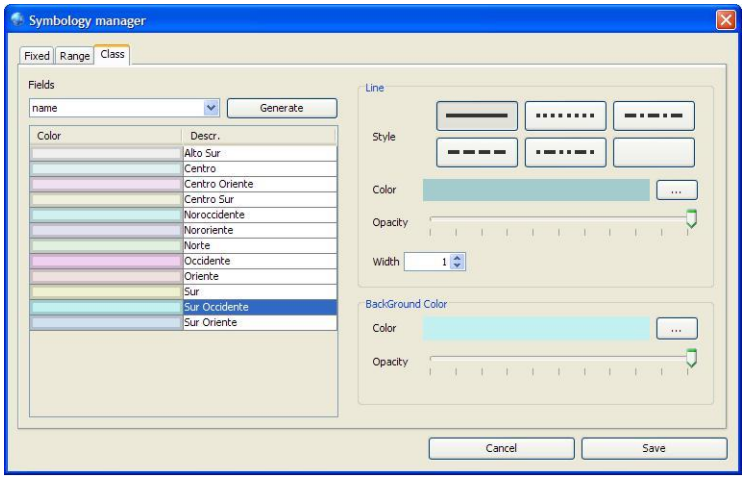

Figura 38. Apariencia de la clase FrmSymbPoly con simbología por clases

1.8. MODULO DE SERVIDOR

Es una aplicación J2EE basada en servlets que se encarga de contestar las peticiones HTTP realizadas por los clientes.

1.8.1. Paquete exceptions

Contiene la clase encargada del manejo de excepciones a nivel WMS.

Clase WMSExceptionUtil. Clase de utilidad, permite generar el documento XML de una excepción WMS. Más detalles de esta clase se muestran en la tabla 21.

Tabla 21. Resumen de la clase WMSExceptionUtil.

Resumen de campos.	
String	**CurrentUpdateSequence**
String	**InvalidCRS**
String	**InvalidDimensionValue**
String	**InvalidFormat**
String	**InvalidPoint**
String	**InvalidUpdateSequence**
String	**LayerNotDefined**
String	**LayerNotQueryable**
String	**MissingDimensionValue**
String	**OperationNotSupported**
String	**StyleNotDefined**
Resumen de métodos	
String	**getXMLException**(String code) Retorna un documento XML que contiene información sobre una excepción en el servicio WMS según lo especifica el estándar.

1.8.2. Paquete geocoder.

Contiene la clase encargada del manejo de peticiones de geocodificación.

Clase service. Servlet encargado de responder a peticiones de geocodificación.
Más detalles de esta clase se muestran en la tabla 22.

Tabla 22. Resumen de la clase geocoder.service.

Resumen de constructores	
service() Constructor en blanco por defecto, requerido por el contenedor de servlets.	
Resumen de métodos	
protected void	**doGet**(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Responde las peticiones realizadas al geocoder mediante el método HTTP GET, el contenido de la URL de la petición normalmente es redactado por los componentes de la biblioteca de desarrollo.
protected void	**doPost**(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Responde a las peticiones realizadas al servlet vía HTTP POST, estas peticiones normalmente son señales administrativas que indican instrucciones, como reiniciar el servlet para adoptar cambios en la configuración o en el material de referencia.
String	**getServletInfo**() Retorna una cadena descriptiva sobre el servlet.
Void	**init**(javax.servlet.ServletConfig config) Método de inicialización llamado por el contenedor de servlets.

1.8.3 Paquete wmsserver.

Contiene la clase encargada del manejo de peticiones WMS.

*Clase service***.** Servlet encargado de responder a peticiones de WMS. Más detalles de esta clase se muestran en la tabla 23.

Tabla 23. Resumen de la clase wmsserver.service.

Resumen de constructores	
service() Constructor en blanco por defecto, requerido por el contenedor de servlets.	
Resumen de métodos	
protected void	**doGet**(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Encargado de responder las peticiones realizadas vía HTTP GET, que comprenden las tareas asociadas al servicio WMS, así como recuperación de leyendas por capa e información sobre sistema s de referencia.
protected void	**doPost**(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Encargado de responder todas las peticiones realizadas mediante HTTP POST, reservado para funciones administrativas como: • Reportar conectividad entre el servlet y la base de datos. • Entregar los datos de conexión a la base de datos al panel de control. • Re iniciar el servlet a fin de adoptar cambios en las configuración o en el material de georeferencial.
void	**init**(javax.servlet.ServletConfig config) Método de inicialización llama do por el contenedor de servlets.

1.9. MODULO WEB ARCHIVE

Web Archive es una aplicación de escritorio creada para facilitar la preparación del archivo WAR del servidor.

1.9.1. Paquete tool.

Contiene los diálogos de la herramienta.

Clase DialogDB. Permite establecer opciones sobre la forma en que los servlets de la aplicación se conectan con la base de datos. La apariencia de esta clase se muestra en la figura 39.

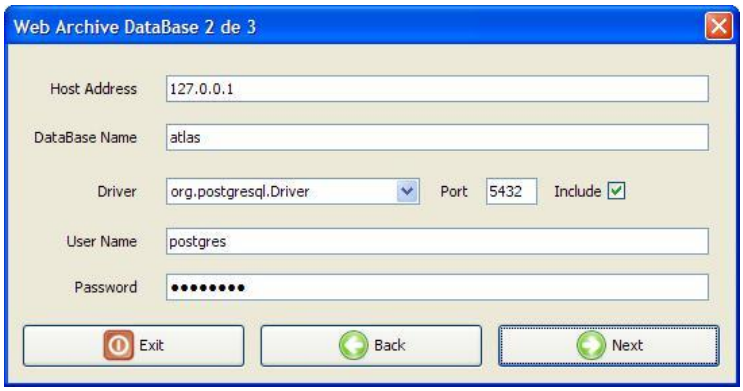

Figura 39. Apariencia de la clase DialogDB

Clase DialogGeocoder. Permite establecer los plugins de geocodificación para el servidor. La apariencia de esta clase se muestra en la figura 40.

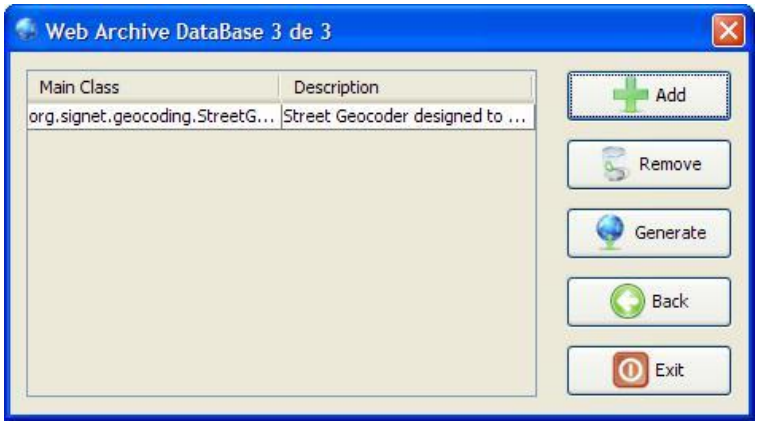

Figura 40. Apariencia de la clase DialogGeocoder

Clase DialogWMS. Permite establecer las opciones generales de la aplicación, como configuraciones de seguridad y ubicación de la aplicación en el contenedor. La apariencia de esta clase se muestra en la figura 41.

Figura 41. Apariencia de la clase DialogWMS

1.10. BIBLIOTECA PARA DESARROLLO DE APLICACIONES JAVA STANDARD EDITION

1.10.1. Paquete gui.

Contiene las clases principales de la biblioteca.

Clase MapPanel. Clase principal de la biblioteca, extiende un JPanel por lo que puede ubicarse a cualquier nivel de una jerarquía swing, en ella se dibujan los mapas y captura las acciones requeridas para la interacción con el usuario. En la tabla 24 semuestra el resumen de esta clase.

Tabla 24. Resumen de la clase MapPanel.

Resumen de campos	
static int	**MODEFEATUREINFO** Modo de operación en que el usuario puede obtener información sobre el contenido del mapa arrastrando el mouse para dibujar un círculo que representa el área de interés o simplemente haciendo clic en el mapa.
static int	**MODEGETCOORD** Modo de operación donde al arrastrar el mouse se logra el mismo efecto que en MODEPAN, pero produce eventos al hacer clic en el mapa y al hacer clic en un marcador.
static int	**MODEPAN** Modo de operación en que el usuario puede desplazarse por el mapa arrastrando el mouse.
static int	**MODEZOOMBOX** Modo de operación en que el usuario puede acercase al mapa arrastrando

	el mouse para dibujar una caja que encierre el área que desea visualizar.
static int	**MODEZOOMMINUS** Modo de operación en que el usuario puede alejarse al mapa haciendo clic en el.
static int	**MODEZOOMPLUS** Modo de operación en que el usuario puede acercarse al mapa haciendo clic en el.
static int	**NOMODESELECTED** Modo de operación en que las acciones del usuario son ignoradas.

Resumen de constructores

MapPanel() Construye un MapPanel por defecto, debe llamarse a los métodos de configuración antes de que el control sea útil.

Resumen de métodos

void	**addMarker**(Marker marker) Agrega un marcador y actualiza la visualización.
void	**clearCache**() Vacía el caché de imágenes del control para que este solicite las versiones más recientes al servidor.
Point2D	**getCenter**() Retorna el centro actual de la visualización en coordenadas de mapa.
String	**getCurrentFeatureInfoFormat**() Retorna el formato que se seleccionó para que el servidor entregue información GetFeatureInfo.
String	**getCurrentImageFormat**() Retorna el formato que se seleccionó para que el servidor entregue imágenes.
Marker	**getCurrentMaker**() Retorna el marcador actual.
Project	**getCurrentProject**() Retorna el proyecto actual.
EventManager	**getEventManager**() Retorna el administrador de eventos asociado a este control.
String[]	**getFeatureInfoFormats**() Retorna los formatos de GetFeatureInfo soportados por el servidor.
Layer[]	**getFeatureInfoLayers**() Retorna el listado de capas a usarse en peticiones GetFeatureInfo.
String[]	**getImageFormats**() Retorna los formatos de imagen soportados por el servidor.
Layer[]	**getLayers**() Retorna el array de capas seleccionadas para solicitar imágenes, la modificación del contenido de este listado puede tener consecuencias inesperadas.

BufferedImage	**getLeyend**(clientdata.Layer layer) Retorna la imagen con la leyenda de esta capa.
List<Marker>	**getMarkers**() Retorna una lista de los marcadores del mapa, la modificación del contenido de esta lista puede tener consecuencias inesperadas.
int	**getMode**() Retorna el modo de operación actual.
Project[]	**getProjects**() Retorna el listado de proyectos disponibles en el servidor.
URL	**getURL**() Retorna la URL del servidor Atlas actual.
BBox	**getVisibleBbox**() Retorna el bounding box que representa la visualización actual.
Layer[]	**getVisibleLayers**() Retorna el listado de las capas que son visibles al nivel de acercamiento actual.
double	**getWMScale**() Retorna la escala de la visualización actual según el estándar . WMS.
int	**getZoomLevel**() Retorna el nivel de acercamiento actual como un número entero, positivo para alejase del mapa y negativo para acercarse.
boolean	**isMouseWheelEnabled**() Indica si está o no habilitado el comportamiento que permite cambiar el zoom con la rueda el mouse.
void	**moveCenter**(Point2D distpx) Desplaza el centro del mapa en forma horizontal y vertical, tantas unidades de pantalla como indiquen las componentes de este punto.
void	**removeAllMarkers**() Remueve todos los marcadores y actualiza la visualización.
void	**removeMarker**(Marker marker) Remueve un marcador y actualiza la visualización.
void	**sendGeocoderRequest**(String address) Envía una petición al geocoder del proyecto actual, las respuestas deben recibiese programando un escuchador de eventos.
void	**setCenter**(Java.awt.geom.Point2D center) Establece el centro de la visualización en unidades de mapa.
void	**setCurrentFeatureInfoFormat**(String format) Establece el formato en que el servidor debe retornar la información GetFeatureInfo, debe ser uno de los formatos declarados por el servidor.
void	**setCurrentImageFormat**(String format) Establece el formato en que el servidor debe retornar las imágenes debe ser uno de los formatos declarados por el servidor.
void	**setCurrentMaker**(Marker marker)

	Establece el marcador actual.
void	**setCurrentProject**(clientdata.Project currentProject) Establece el proyecto actual, debe pertenecer al array de proyectos disponibles en el servidor.
void	**setFeatureInfoLayers**(clientdata.Layer[] layersFeatureInfo) Establece el listado de capas a usarse en peticiones GetFeatureInfo.
void	**setLayers**(clientdata.Layer[] layers) Establece las capas que se usarán para visualización , deben pertenecer al listado de capas del proyecto actual.
void	**setMode**(int mode) Establece el modo de operación actual de acuerdo a las constantes de clase.
void	**setMouseWheelEnabled**(boolean enabled) Establece si está o no habilitado el comportamiento que permite hacer zoom con la rueda el mouse.
void	**setURL**(Java.net.URL appUrl) Establece la URL del servidor Atlas con el que se desea trabajar, y recupera la información correspondiente al contenido disponible en este.
void	**setZoomLevel**(int lod) Establece el nivel de acercamiento actual como un número entero, positivo para alejase del mapa y negativo para acercarse.
void	**updateView**() Actualiza la visualización para reflejar cambios en los datos.
void	**ZoomIn**() Incrementa el acercamiento y actualiza la visualización.
void	**ZoomOut**() Reduce el acercamiento y actualiza la visualización.
void	**zoomToProject**() Ajusta el nivel de acercamiento y centro de tal modo que se visualice todo el proyecto.

Clase EventManager. Clase que funciona en conjunto con MapPanel y permite administrar los escuchadores a diferentes eventos así como dispararlos. Más detalles de esta clase se muestran en la tabla 25.

Tabla 25. Resumen de la clase EventManager.

Resumen de constructores
EventManager(MapPanel map) Construye un manejador de eventos para el MapPanel dado.

Resumen de métodos	
Void	**addDisplayListener**(DisplayListener l) Agrega un escuchador de movimiento de mouse en el mapa.
Void	**addFeatureInfoListener**(FeatureInfoListener fl) Agrega un escuchador de peticiones FeatureInfo.
Void	**addGeocodingListener**(GeocodingListener gl) Agrega un escuchador de respuestas del geocoder.
Void	**addMarkerClickedListener**(MarkerClickedListener mcl) Agrega un escuchador de clic de mouse en un marcador.
Void	**addMouseClickListener**(MouseClickListener l) Agrega un escuchador de clic de mouse en el mapa.
Void	**addMouseMoveListener**(MouseMoveListener ml) Agrega un escuchador de movimiento de mouse en el mapa.
Void	**removeGeocodingListener**(GeocodingListener gl) Remueve un escuchador de respuestas del geocoder.
Void	**removeDisplayMoveListener**(DisplayListener l) Remueve un escuchador de movimiento de mouse en el mapa.
Void	**removeFeatureInfoListener**(FeatureInfoListener fl) Remueve un escuchador de peticiones FeatureInfo.
Void	**removeMarkerClickedListener**(MarkerClickedListener mcl) Remueve un escuchador de clic de mouse en un marcador.
Void	**removeMouseClickListener**(MouseClickListener l) Remueve un escuchador de clic de mouse en el mapa.
Void	**removeMouseMoveListener**(MouseMoveListener al) Remueve un escuchador de movimiento de mouse en el mapa.

1.10.2. Paquete events.

Contiene las clases e interfaces asociadas a diferentes eventos manipulados por el EventManager. En la tablas 26 y 27 se listan y describen las clases e interfaces del paquete events.

Tabla 26. Resumen de interfaces del paquete Event.

Resumen de interfaces	
DisplayListener	Escuchador para eventos DisplayEvent.
FeatureInfoListener	Escuchador para eventos FeatureInfoEvent.
GeocodingListener	Escuchador para eventos GeocodingEvent.
MarkerClickedListener	Escuchador para eventos MarkerClickedEvent.
MouseClickListener	Escuchador para eventos MouseClickListener.
MouseMoveListener	Escuchador para eventos MouseMoveListener.

Tabla 27. Resumen de clases del paquete Event.

Resumen de clases

DisplayEvent	Evento disparado cuando cambia la escala o el bounding box que describe lo que se está presentando.
FeatureInfoEvent	Evento disparado cuando el servidor responde a una petición GetFeatureInfo realizada desde el MapPanel.
GeocodingEvent	Evento disparado cuando el servidor responde a una petición de geocodificación realizada desde el MapPanel.
MarkerClickedEvent	Evento disparado cuando el usuario hace clic sobre un marcador y el modo está fijado en MODEGETCOORD.
MouseClickEvent	Evento disparado cuando el usuario hace clic sobre un área del mapa y el modo está fijado en MODEGETCOORD.
MouseMoveEvent	Evento disparado cuando el usuario mueve el mouse sobre el mapa.

1.10.3. Paquete mashup.

Asociado a la producción de aplicaciones hibridas usando la biblioteca.

Clase Marker. Representa un punto que puede ubicarse sobre el mapa, usualmente simboliza información propia del domino de la aplicación final, logrando de este modo el mashup con la cartografía del servidor. Más detalles de esta clase se muestran en la tabla 28.

Tabla 28. Resumen de la clase Marker.

Resumen de constructores	
Marker(MapPanel map, Point2D coordinate, BufferedImage normal) Construye un marcador para un MapPanel, ubicándolo en la coordenada indicada, y representándolo con la imagen especificada.	
Resumen de métodos	
void	**findPosition**() Calcula las coordenadas en píxeles que corresponden a este marcador dado el estado del MapPanel.
Point2D	**getCoordinate**() Retorna las coordenadas del marcador respecto al mapa.
BufferedImage	**getCurrent**() Retorna la imagen que representa al marcador cuando es el actual.
MapPanel	**getMap**() Retorna el objeto MapPanel en el que se encuentra el marcador.
BufferedImage	**getNormal**() Retorna la imagen que representa al marcador en su estado normal.
BufferedImage	**getOver**() Retorna la imagen que representa al marcador cuando el mouse pasa sobre él.
Point2D	**getScreen**() Retorna las coordenadas en pantalla del marcador, en el estado

	actual del MapPanel.
Boolean	**isPointInside**(int x, int y) Indica si un píxel con coordenadas x, y toca al marcador teniendo en cuenta su posición actual en la pantalla.
void	**setCoordinate**(Java.awt.geom.Point2D coordinate) Establece las coordenadas del marcador respecto al mapa.
void	**setCurrent**(Java.awt.image.BufferedImage current) Establece la imagen que representa al marcador cuando es el actual.
void	**setNormal**(Java.awt.image.BufferedImage normal) Establece la imagen que representa al marcador en su estado normal.
void	**setOver**(Java.awt.image.BufferedImage over) Establece la imagen que representa al marcador cuando el mouse pasa sobre él.

1.11 BIBLIOTECA PARA DESARROLLO DE APLICACIONES WEB 2.0 CON JAVASCRIPT

1.11.1. Paquete Atlas.

Contiene las clases principales de la biblioteca.

Clase Layer. Una capa del sistema, pertenece a un proyecto y puede ser incluida en las peticiones dirigidas al servidor. Más detalles de esta clase se muestran en la tabla 29.

Tabla 29. Resumen de la clase Layer.

Resumen de métodos	
String	**getName** () Retorna el identificador único de la capa. De conformidad al estándar WMS.
String	**getTitle** () Retorna una descripción breve para mostrar. De conformidad al estándar WMS.
Number	**getMaxscaledeno** () Retorna el denominador mínimo de escala. De conformidad al estándar WMS.
Number	**getMinscaledeno** () Retorna el denominador mínimo de escala. De conformidad al estándar WMS.

Clase CRS. Representa un sistema de referencia espacial que proporciona al control información sobre la forma en que debe interpretar la información espacial entregada por el servidor. Más detalles de esta clase se muestran en la tabla 30.

Tabla 30. Resumen de la clase CRS.

Resumen de métodos	
Number	**getCode** () Retorna el código EPSG del sistema.
String	**getName** () Retorna una descripción breve del sistema.
String	**getAxis1** () Retorna el primer eje del sistema, puede ser “north”, “sout”, “east” o “west”.
String	**getAxis2** () Retorna el segundo eje del sistema, puede ser, “north”, “sout”, “east” o “west”.
String	**getUnitType** () Retorna el tipo de la unidad puede ser "angular" o "lineal".
Number	**getConvertionFactor** () Retorna el número por el que debe multiplicarse cualquier medida expresada en este sistema para llevarla a la unidad fundamental, metros o grados decimales según el caso.

Clase Project. Un proyecto del sistema, define un conjunto de capas y el sistema de coordenadas de referencia en que se encuentran. Más detalles de esta clase se muestran en la tabla 31.

Tabla 31. Resumen de la clase Project.

Resumen de métodos	
String	**getName** () Identificador único de la capa. De conformidad al estándar WMS.
String	**getTitle** () Retorna una descripción breve para mostrar. De conformidad al estándar WMS.
BoundingBox	**getBoundingBox** () Retorna el Bonding box que describe los límites del proyecto en el sistema de coordenadas de este.
CRS	**getEpsgCrs** () Retorna el sistema de coordenadas de referencia del proyecto.
Array	**getLayers** ()

	Retorna un array de Layer, capas que contiene el proyecto.

Clase coordinateutils. Métodos útiles para convertir coordenadas de pantalla a mapa y viceversa. Más detalles de esta clase se muestran en la tabla 32.

Tabla 32. Resumen de la clase coordinateutils.

COORDINATEUTILS	
Resumen de métodos	
Point	**translateXYtoIJ** (bbox,width,heigth,x,y,crs) Convierte medidas en unidades de mapa a unidades de pantalla.
Point	**translateIJtoXY** (bbox,dimI,dimJ,coordI,coordJ,crs) Convierte medidas en píxeles a unidades de mapa.

Clase BoundigBox. Una representación ligera de un boundig box, encierra una región del espacio 2D usando dos coordenadas. Su interpretación depende del sistema de coordenadas de referencia que define las unidades en que se expresan las coordenadas y la orientación de los ejes. Más detalles de esta clase se muestran en la tabla 33.

Tabla 33. Resumen de la clase BoundigBox.

Resumen de constructores	
BoundingBox (crs,minx,miny,maxx,maxy) Construye dados valores máximos y mínimos.	
Resumen de métodos	
String	**getCRS** () Retorna el sistema de coordenadas de referencia del bounding box.
Number	**getMinX** () Retorna el mínimo valor del primer eje.
Number	**getMinY** () Retorna el mínimo valor del segundo eje.
Number	**getMaxX** () Retorna el máximo valor del primer eje.
Number	**getMaxY** () Retorna el máximo valor del segundo eje.
Number	**getWidth** () Retorna el ancho del área delimitada.
Number	**getHeight** () Retorna el alto del área delimitada.

Clase Point. Representa una ubicación en el espacio bidimensional. Más detalles de esta clase se muestran en la tabla 34.

Tabla 34. Resumen de la clase Point.

Resumen de constructores	
Point (Number x, Number y) Construye dados los parámetros.	
Resumen de	
métodos Number	**getX** () Retorna el valor del componente X de la ubicación.
Number	**getY** () Retorna el valor del componente Y de la ubicación.
Ninguno	**setX** (Number x) Establece el valor del componente X de la ubicación.
Ninguno	**setY** (Number y) Establece el valor del componente Y de la ubicación.
Ninguno	**setLocation** (Number x, Number y) Establece los valores de la ubicación.
Boolean	**equals** (Point point) Indica si point es igual a este punto.
Point	**clone** () Retorna un nuevo punto en la ubicación de este punto.

Clase Marker. Representa un punto que puede ubicarse sobre el mapa, usualmente simboliza información propia del domino de la aplicación final, logrando de este modo el mashup con la cartografía del servidor. Más detalles de esta clase se muestran en la tabla 35.

Tabla 35. Resumen de la clase Marker.

Resumen de constructores	
Marker (coordinate, data, imgNormal, imgOver, imgSelect, width, height) Point Coordinate, representa la ubicación del marcador en el mapa. Array Data, cualquier información adicional que acompañe al marcador. String imgNormal, url de la imagen del marcador en su estado normal. String imgOver, url de la imagen del marcador cuando el mouse pasa sobre él. String imgSelect, url de la imagen del marcador cuando está seleccionado. Number width, ancho del marcador. Number height, alto del marcador.	
Resumen de métodos	
Point	**getCoordinate** () Retorna la ubicación del marcador en el mapa.

Array	**getData** () Retorna cualquier información adicional que acompañe al marcador.
String	**getNormalImage** () Retorna la URL de la imagen del marcador en su estado normal.
String	**getOverImage** () Retorna la URL de la imagen del marcador cuando el mouse pasa sobre él.
String	**getSelectImage** () Retorna la URL de la imagen del marcador cuando está seleccionado.
Number	**getWidth** () Retorna el ancho del marcador.
Number	**getHeight** () Retorna el alto del marcador.

Clase Result. Representa un punto que puede ubicarse sobre el mapa, usualmente simboliza información propia del domino de la aplicación final, logrando de este modo el mashup con la cartografía del servidor. Más detalles de esta clase se muestran en la tabla 36.

Tabla 36. Resumen de la clase Result.

Resumen de métodos	
Number	**getX** () Retorna el primer componente de la ubicación.
Number	**getY** () Retorna el segundo componente de la ubicación.
String	**getAddress** () Retorna la dirección tal como la interpretó el servidor.
Ninguno	**setX**(Number x) Establece el primer componente de la ubicación.
Ninguno	**setY**(Number y) Establece el segundo componente de la ubicación.
Ninguno	**setAddress** (String address) Establece la dirección tal como la interpretó el servidor.

Clase MapPanel. Clase principal de la biblioteca encargada de la construcción y administración de los elementos necesarios para mostrar imágenes e interactuar con el usuario. Más detalles de esta clase se muestran en la tabla 37.

Tabla 37. Resumen de la clase MapPanel.

Resumen de constructores	
MapPanel (String div, String mainpath) Construye un nuevo MapPanel, div indica el elemento en el que debe crearse y mainpath es la ruta a biblioteca Atlas.	
Resumen de métodos	
Ninguno	**addListener** (String type, Function fun) Agrega la función fun para que sea llamada cada vez que se genere un evento de tipo type en el control.
Ninguno	**addMarker** (Marker marker) Agrega un marcador y actualiza la visualización.
Point	**getCenter** () Retorna el centro actual de la visualización en coordenadas de mapa.
Point	**getCoordinate** () Retorna la coordenada de mapa asociada la posición actual del mouse en el control.
String	**getCurrentFeatureInfoFormat** () Retorna el formato que se seleccionó para que el servidor entregue información GetFeatureInfo.
String	**getCurrentImageFormat** () Retorna el formato que se seleccionó para que el servidor entregue imágenes.
Marker	**getCurrentMarker** () Retorna el marcador actual.
Point	**getCurrentMousePosition** () Retorna las coordenadas en píxeles de la posición actual de mouse respecto al control.
Project	**getCurrentProject** () Retorna el proyecto actual.
Array	**getFeatureInfoFormats** () Retorna un Array de String que contiene los formatos de GetFeatureInfo soportados por el servidor
Array	**getFeatureInfoLayers** () Retorna un Array de String que contiene un listado de capas a usarse en peticiones GetFeatureInfo.
Number	**getHeight** () Retorna el alto del control en píxeles.
Array	**getImageFormats** () Retorna un Array de String con los formatos de imagen soportados por el servidor.
Array	**getLayers** () Retorna el Array de Layer de las capas seleccionadas para solicitar imágenes, la modificación del contenido de este listado puede tener consecuencias inesperadas.
String	**getLegend** (Layer layer)

	Retorna la URL de la imagen con la leyenda de la capa layer.
Array	**getMarkers** () Retorna una lista de los marcadores del mapa, la modificación del contenido de esta lista puede tener consecuencias inesperadas.
Number	**getMode** () Retorna el modo de operación actual.
Array	**getProjects** () Retorna un Array de Project que contiene el listado de proyectos disponibles en el servidor.
Array	**getResponseGeocoder** () Retorna un Array de Result con la última respuesta a una petición del geocodificación.
String	**getURL** () Retorna la URL del servidor Atlas actual.
BoundingBox	**getVisibleBbox** () Retorna un BoundingBox que representa la visualización actual.
Array	**getVisibleLayers** () Retorna el listado de las capas que son visibles al nivel de acercamiento actual.
Number	**getWidth** () Retorna el ancho de control en píxeles.
Number	**getWMScale** () Retorna la escala de la visualización actual según el estándar WMS.
Number	**getZoomLevel** () Retorna el nivel de acercamiento actual como un número entero, positivo para alejase del mapa y negativo para acercarse.
Boolean	**isMouseWheelEnabled** () Indica si está o no habilitado el comportamiento que permite hacer zoom con la rueda el mouse.
Ninguno	**moveCenter** (Point disp) Desplaza el centro del mapa en forma horizontal y vertical, tantas unidades de pantalla como indiquen las componente
Ninguno	**removeAllMarkers** () Remueve todos los marcadores y actualiza la visualización.
Ninguno	**removeListener** (String type, Object fun) Remueve la función fun para que deje de ser llamada cada vez que se genere un evento de tipo type en el control.
Ninguno	**removeMarker** (Marker marker) Remueve un marcador y actualiza la visualización.
Ninguno	**sendGeocoderRequest** (String address) Envía una petición al geocoder el proyecto actual, las respuestas deben recibiese programando un escuchador de eventos.
Ninguno	**setCenter** (Point center) Establece el centro de la visualización en unidades de mapa.

Ninguno	**setCurrentFeatureInfoFormat** (String format) Establece el formato en que el servidor debe retornar la información GetFeatureInfo, debe ser uno de los formatos declarados por el servidor.
Ninguno	**setCurrentImageFormat** (String format) Establece el formato en que el servidor debe retornar las imágenes debe ser uno de los formatos declarados por el servidor.
Ninguno	**setCurrentMarker** (Marker marker) Establece el marcador actual.
Ninguno	**setCurrentProject** (Project project) Establece el proyecto actual, debe pertenecer al array de proyectos disponibles en el servidor.
Ninguno	**setFeatureInfoLayers** (Array infolayers) Establece el Array de Layer, con el listado de capas a usarse en peticiones GetFeatureInfo.
Ninguno	**setFeatureInfoTargetWindow** (name,windowfeatures) Establece que los resultados de peticiones GetFeatureInfo deben presentarse en una ventana nueva. Los parámetros son los mismos que en el método window.open de javascript. Es el comportamiento por defecto.
Ninguno	**setFeatureInfoTargetFrame** (String id) Establece que los resultados de peticiones GetFeatureInfo deben presentarse en un iFrame cuyo id sea igual al parámetro.
Ninguno	**setLayers** (Array layers) Establece el Array de Layer, con el listado de las capas que se usarán para visualización, deben pertenecer al listado de capas del proyecto actual.
Ninguno	**setMode** (mode) Establece el modo de operación actual de acuerdo a las constantes de clase.
Ninguno	**setMouseWheelEnabled** (Boolean state) Establece si está o no habilitado el comportamiento que permite hacer zoom con la rueda el mouse.
Ninguno	**setURL** (String urlApp) Establece la URL del servidor Atlas con el que se desea trabajar, y recupera la información correspondiente al contenido disponible en este.
Ninguno	**setZoomLevel** (Number lod) Establece el nivel de acercamiento actual como un número entero, positivo para alejase del mapa y negativo para acercarse.
Ninguno	**updateView** () Actualiza la visualización para reflejar cambios en los datos.
Ninguno	**zoomIn** ()

	Incrementa el acercamiento y actualiza la visualización.
Ninguno	**zoomOut** () Reduce el acercamiento y actualiza la visualización.
Ninguno	**zoomToProject** () Ajusta el nivel de acercamiento y centro de tal modo que se visualice todo el proyecto.

1.12 BIBLIOTECA PARA DESARROLLO DE APLICACIONES JAVA MOBILE EDITION

1.12.1 Paquete data.

Contiene la estructura de datos que representa contenido del servidor.

Clase BBox. Una representación ligera de un bounding box para ser usada en los componentes de desarrollo. Un bounding box encierra una región del espacio 2D usando dos coordenadas. Su interpretación depende del sistema de coordenadas de referencia, que define las unidades en que se expresan las coordenadas y la orientación de los ejes. Más detalles de esta clase se muestran en la tabla 38.

Tabla 38. Resumen de la clase BBox.

Resumen de constructores	
BBox(double minx, double miny, double maxx, double maxy) Construye dados valores máximos y mínimos.	
BBox(org.kxml.kdom.Element bBox) Construye desde un elemento XML compatible con WMS.	
Resumen de métodos	
boolean	**equals**(BBox bbox) Indica si bbox es igual a este objeto.
Point	**getCenter**() Retorna el centro geométrico del bounding box.
String	**getCRS**() Retorna el CRS de este objeto.
double	**getHeight**() Retorna el alto del área delimitada.
double	**getMaxx**() Retorna el máximo valor del primer eje.
double	**getMaxy**() Retorna el máximo valor del segundo eje.
double	**getMinx**() Retorna el mínimo valor del primer eje.
double	**getMiny**() Retorna el mínimo valor del segundo eje.

Java.lang.String	**getUrlForm**() Retorna el bounding box en forma de KVP para ser usado en peticiones.
double	**getWidth**() Retorna el ancho del área delimitada.
boolean	**intersects**(BBox bbox) Indica si bbox se intersecta con este objeto.
void	**setCRS**(String crs) Establece el CRS de este objeto.
void	**setMaxx**(double maxx) Establece el máximo valor del primer eje.
void	**setMaxy**(double maxy) Establece el máximo valor del primer eje.
void	**setMinx**(double minx) Establece el mínimo valor del primer eje.
void	**setMiny**(double miny) Establece el mínimo valor del segundo eje.

Clase CRS. Representa un sistema de coordenadas de referencia del EPSG para uso en los componentes de desarrollo de aplicaciones móviles. En la tabla 39 se muestran mas detalles de esta clase.

Tabla 39. Resumen de la clase CRS.

Resumen de campos	
static int	**UNIT_ANGULAR** Indica que el sistema funciona con unidades angulares que se pueden expresar en términos de grados.
static int	**UNIT_LINEAR** Indica que el sistema funciona con unidades lineales que se pueden expresar en términos de metros.
Resumen de constructores	
CRS(String code, String urlService) Construye recuperando los datos del sistema code desde el servidor Atlas ubicado en urlService.	
Resumen de métodos	
String	**getAxis1**() Retorna el nombre del primer eje.
String	**getAxis2**() Retorna el nombre del segundo eje.
String	**getCode**() Retorna el código EPSG del sistema.
double	**getConversionFactor**() Retorna el factor por el que deben multiplicarse las medidas para

	convertirlas a la unidad base de sistema.
String	**getName**() Retorna el nombre EPSG del sistema.
int	**getType**() Retorna el tipo del sistema según las constantes de clase.
String	**getUnitName**() Retorna el nombre de la unidad de medida de este sistema.
void	**setAxis1**(String axis1) Establece el nombre del primer eje.
void	**setAxis2**(String axis2) Establece el nombre del segundo eje.
void	**setCode**(String code) Establece el código EPSG del sistema.
void	**setConversionFactor**(double conversionFactor) Establece el factor por él que deben multiplicarse las medidas para convertirlas a la unidad base de sistema.
void	**setName**(String name) Establece el nombre EPSG del sistema.
void	**setType**(int type) Establece el tipo del sistema según las constantes de clase.
void	**setUnitName**(String unitName) Establece el nombre de la unidad de medida de este sistema.

Clase Layer. Una capa del sistema, pertenece a un proyecto y puede ser incluida en las peticiones dirigidas al servidor. Más detalles de esta clase se muestran en la tabla 40.

Tabla 40. Resumen de la clase Layer.

Resumen de constructores	
Layer() Construye un Layer en blanco.	
Layer(Element layer, Project project) Construye desde un elemento XML valido y anexa al proyecto project.	
Resumen de métodos	
String	**getAtribution**() Retorna el responsable de la capa.
BBox	**getBoundigBox**() Retorna el bounding box en el sistema de coordenadas de referencia del proyecto.
CRS	**getCRS**() Retorna el sistema de coordenadas de referencia para la capa.
Layer.Ex_BoundingBox	**getEx_BoundingBox**()

	Retorna el bounding box respecto a WGS 84.
Java.util.Vector	**getKeywordList**() Retorna un listado de palabras clave.
double	**getMaxscaledeno**() Retorna el denominador máximo de escala.
double	**getMinscaledeno**() Retorna el denominador mínimo de escala.
String	**getName**() Retorna el identificador de la capa.
Project	**getProject**() Retorna el proyecto al que pertenece la capa.
String	**getSummary**() Retorna un resumen de la capa.
String	**getTitle**() Retorna una descripción breve para mostrar.
boolean	**isQueryable**() Indica si la capa puede ser consultada en una operación GetFeatureInfo.
void	**setAtribution**(String atribution) Establece el responsable de la capa.
void	**setBoundigBox**(BBox boundigBox) Establece el bounding box en el sistema de coordenadas de referencia del proyecto.
void	**setCRS**(CRS CRS) Establece el sistema de coordenadas de referencia para la capa.
void	**setEx_BoundingBox**(Layer.Ex_BoundingBox bbox) Establece el bounding box respecto a WGS 84.
void	**setKeywordList**(Java.util.Vector keywordList) Establece un listado de palabras clave.
void	**setMaxscaledeno**(double maxscaledeno) Establece el denominador máximo de escala.
void	**setMinscaledeno**(double minscaledeno) Establece el denominador mínimo de escala.
void	**setName**(String name) Establece el identificador de la capa.
void	**setProject**(Project project) Establece el proyecto al que pertenece la capa.
void	**setQueryable**(boolean queryable) Establece si la capa puede ser consultada por una operación GetFeatureInfo.
void	**setSummary**(String summary) Establece el resumen de la capa.
void	**setTitle**(String title)

	Establece descripción breve para mostrar.

Clase Project. Representa un proyecto para su uso en los componentes de desarrollo, contiene un conjunto de capas. Más detalles de esta clase se muestran en la tabla 41.

Tabla 41. Resumen de la clase Project.

Resumen de constructores	
Project() Construye un proyecto con los campos vacíos.	
Project(Element project, String urlService) Construye un proyecto desde un elemento XML conectándose al servidor Atlas en urlService.	
Resumen de métodos	
BBox	**getBbox**() Retorna el bounding box combinado de los bounding boxes de los orígenes de datos de las capas del proyecto.
CRS	**getCrs**() Retorna el sistema de coordenadas de referencia.
Java.util.Vector	**getLayers**() Retorna el listado de capas del proyecto.
String	**getName**() Retorna el identificador del proyecto.
String	**getTitle**() Retorna una descripción breve para mostrar.
void	**setBbox**(BBox bbox) Establece el bounding box en el sistema de coordenadas de referencia del proyecto.
void	**setCrs**(CRS crs) Establece el sistema de coordenadas de referencia.
void	**setLayers**(Java.util.Vector layers) Establece el listado de capas del proyecto.
void	**setName**(String name) Establece el identificador del proyecto.
void	**setTile**(String title) Establece una descripción breve para mostrar.

1.12.2. Paquete events.

Contiene las clases e interfaces necesarias para trabajar con los eventos generados por el control.

Clase FeatureInfoEvent. Evento disparado cuando el control recibe la respuesta a una petición GetFeatureInfo. Más detalles de esta clase se muestran en la tabla 42.

Tabla 42. Resumen de la clase FeatureInfoEvent.

Resumen de constructores	
FeatureInfoEvent(MapPanel source, String featureInfo, String url) Construye un evento desde el MapPanel source, entrega la respuesta del servidor en featureInfo e indica la dirección de la petición en url.	
Resumen de métodos	
String	**getFeatureInfo**() Retorna la respuesta del servidor.
MapPanel	**getSource**() Retorna el control que origina el evento.
String	**getUrl**() Retorna la URL de la petición GetFeatureInfo.

Clase GeocodingEvent. Evento disparado cuando el control recibe la respuesta a una petición de geocoder. Más detalles de esta clase se muestran en la tabla 43.

Tabla 43. Resumen de la clase GeocodingEvent.

Resumen de constructores	
GeocodingEvent(MapPanel source, String project, String request, Result[] addresses) Construye un evento originado desde source, como respuesta una petición request para el geocoder de project que arroja como respuesta address.	
Resumen de métodos	
Result[]	**getAddresses**() Retorna la lista de direcciones a la petición.
MapPanel	**getMapSource**() Retorna el MapPanel de origen.
String	**getProject**() Retorna el proyecto sobre el que se realizó la petición.
String	**getRequest**() Retorna la petición realizada.

Clase MarkerTapEvent. Evento lanzado cuando el usuario toca la pantalla con el lápiz del dispositivo y hay un marcador es esta ubicación. Más detalles de esta clase se muestran en la tabla 44.

Tabla 44. Resumen de la clase MarkerTapEvent.

Resumen de constructores	
MarkerTapEvent(MapPanel mapPanel, Point screen, Point map, Marker marker) Construye un evento indicando a mapPanel como fuente, screen como la coordenada en pantalla y map como la coordenada en el sistema de referencia del proyecto actual.	
Resumen de métodos	
MapPanel	**getMapPanel**() Retorna el MapPanel que originó el evento.
Marker	**getMarker**() Retorna el marcador del evento.
Point	**getPointMap**() Retorna la coordenada en el sistema de referencia del proyecto actual.
Point	**getPointScreen**() Retorna

Clase PointerTapEvent. Evento lanzado cuando el usuario toca la pantalla con el lápiz del dispositivo. Más detalles de esta clase se muestran en las tablas 45 y 46.

Tabla 45. Resumen de la clase PointerTapEvent.

Resumen de constructores	
PointerTapEvent(MapPanel mapPanel, Point screen, Point map) Construye un evento indicando a mapPanel como fuente, screen como la coordenada en pantalla y map como la coordenada en el sistema de referencia del proyecto actual.	
Resumen de métodos	
MapPanel	**getMapPanel**() Retorna el MapPanel que originó el evento.
Point	**getPointMap**() Retorna la coordenada en el sistema de referencia del proyecto actual.
Point	**getPointScreen**() Retorna la coordenada en pantalla del evento.

Tabla 46. Interfaces del paquete events.

Interface Summary	
FeatureInfoListener	Escuchador de eventos de FeatureInfoEvent.
GeocodingListener	Escuchador de eventos de tipo GeocodingEvent.
MarkerTapListener	Escuchador de eventos de tipo MarkerTapEvent.
PointerTapListener	Escuchador de eventos de tipo MarkerTapEvent.

1.12.3. Paquete geocoding.

Contiene las clases usadas para interpretar los resultados de las operaciones de geocodificación.

*Clase Result***.** Clase que representa un posible resultado de una operación de geocodificación. Más detalles de esta clase se muestran en la tabla 47.

Tabla 47. Resumen de la clase Result.

Resumen de constructores	
Result(String address, Point location) Construye dados una dirección y una ubicación.	
Resumen de métodos	
String	**getAddress**() Retorna la descripción textual de la ubicación , tal como el geocoder la interpretó.
Point	**getLocation**() Retorna la ubicación en el sistema de coordenadas de referencia del proyecto al que corresponde.

1.12.4. Paquete mobile.

Contiene las clases principales de la biblioteca.

Clase MapPanel. Clase principal de la biblioteca, extiende de CustomItem por lo que puede ubicarse en un formulario, en ella se dibujan los mapas y captura las acciones requeridas para la interacción con el usuario. Más detalles de esta clase se muestran en la tabla 48.

Tabla 48. Resumen de la clase MapPanel.

Resumen de campos	
static int	**MODEFEATUREINFO** Modo de operación en que el usuario puede obtener información sobre el contenido del mapa arrastrando el mouse para dibujar un círculo que representa el área de interés o simplemente haciendo clic en el mapa.
static int	**MODEGETCOORD** Modo de operación donde al arrastrar el lápiz se logra el mismo efecto que en MODEPAN, pero produce eventos al hacer tocar el mapa y al tocar un marcador.
static int	**MODEPAN** Modo de operación en que el usuario puede desplazarse por

	el mapa arrastrando el mouse.
static int	**MODEZOOMBOX** Modo de operación en que el usuario puede acercase al mapa arrastrando el lápiz para dibujar una caja que encierre el área que desea visualizar.
static int	**MODEZOOMMINUS** Modo de operación en que el usuario puede alejarse al mapa tocando la pantalla con el lápiz.
static int	**MODEZOOMPLUS** Modo de operación en que el usuario puede acercase al mapa tocando la pantalla con el lápiz.
static int	**NOMODESELECTED** Modo de operación en que las acciones del usuario son ignoradas.

Resumen de constructores

MapPanel(int width, int height)
Construye un MapPanel por defecto, con el ancho y alto indicados, debe llamarse a los métodos de configuración antes de que el control sea útil.

Resumen de métodos

void	**addMarker**(Marker marker) Agrega un marcador y actualiza la visualización.
void	**clearCache**() Vacía el caché de imágenes del control para que este solicite las versiones más recientes al servidor.
int	**getBloqSize**() Retorna el tamaño de las imágenes que conforman la presentación.
Point	**getCenter**() Retorna el centro actual de la visualización en coordenadas de mapa.
String	**getCurrentFeatureInfoFormat**() Retorna el formato que se seleccionó para que el servidor entregue información GetFeatureInfo.
String	**getCurrentImageFormat**() Retorna el formato que se seleccionó para que el servidor entregue imágenes.
Marker	**getCurrentMarker**() Retorna el marcador actual.
Project	**getCurrentProject**() Retorna el proyecto actual.
EventManager	**getEventManager**() Retorna el administrador de eventos asociado a este control.
Java.util.Vector	**getFeatureInfoFormats**()

	Retorna los formatos de GetFeatureInfo soportados por el servidor.
Java.util.Vector	**getFeatureInfoLayers**() Retorna el listado de capas a usarse en peticiones GetFeatureInfo.
int	**getHeight**() Retorna el alto del control.
Java.util.Vector	**getImageFormats**() Retorna los formatos de imagen soportados por el servidor.
Java.util.Vector	**getLayers**() Retorna el array de capas seleccionadas para solicitar imágenes, la modificación del contenido de este listado puede tener consecuencias inesperadas.
Image	**getLegend**(Layer layer) Retorna la imagen con la leyenda de esta capa.
Java.util.Vector	**getMarkers**() Retorna una lista de los marcadores del mapa, la modificación del contenido de esta lista puede tener consecuencias inesperadas.
int	**getMode**() Retorna el modo de operación actual.
Java.util.Vector	**getProjects**() Retorna el listado de proyectos disponibles en el servidor.
String	**getURL**() Retorna la URL del servidor Atlas actual.
BBox	**getVisibleBbox**() Retorna el bounding box que representa la visualización actual.
Java.util.Vector	**getVisibleLayers**() Retorna el listado de las capas que son visibles al nivel de acercamiento actual.
int	**getWidth**() Retorna el ancho del control.
double	**getWMScale**() Retorna la escala de la visualización actual según el estándar.
int	**getZoomLevel**() Retorna el nivel de acercamiento actual como un número entero, positivo para alejase del mapa y negativo para acercarse.
void	**moveCenter**(Point point) Desplaza el centro del mapa en forma horizontal y vertical, tantas unidades de pantalla como indiquen las componentes de este punto.
void	**removeAllMarkers**() Remueve todos los marcadores y actualiza la visualización.

void	**removeMarker**(Marker marker) Remueve un marcador y actualiza la visualización.
void	**sendGeocoderRequest**(String address) Envía una petición al geocoder del proyecto actual, las respuestas deben recibiese programando un escuchador de eventos.
void	**setBloqSize**(int bloqSize) Establece el tamaño de las imágenes que conforman la presentación.
void	**setCache**(int sizeK) Establece el tamaño del caché de imágenes.
void	**setCenter**(Point center) Establece el centro de la visualización en unidades de mapa.
Void	**setCurrentFeatureInfoFormat**(String currentInfoFormat) Establece el formato en que el servidor debe retornar la información GetFeatureInfo, debe ser uno de los formatos declarados por el servidor.
void	**setCurrentImageFormat**(String currentImageFormat) Establece el formato en que el servidor debe retornar las imágenes debe ser uno de los formatos declarados por el servidor.
void	**setCurrentMarker**(Marker currentMarker) Establece el marcador actual.
void	**setCurrentProject**(Project project) Establece el proyecto actual, debe pertenecer al array de proyectos disponibles en el servidor.
void	**setFeatureInfoLayers**(Java.util.Vector layersFeatureInfo) Establece el listado de capas a usarse en peticiones GetFeatureInfo.
void	**setHeight**(int height) Establece el alto del control.
void	**setLayers**(Java.util.Vector layers) Establece las capas que se usarán para visualización, deben pertenecer al listado de capas del proyecto actual.
void	**setMode**(int mode) Establece el modo de operación actual de acuerdo a las constantes de clase.
void	**setURL**(String url) Establece la URL del servidor Atlas con el que se desea trabajar, y recupera la información correspondiente al contenido disponible en este.
void	**setWidth**(int width) Establece en ancho del control.
void	**setZoomLevel**(int level)

	Establece el nivel de acercamiento actual como un número entero, positivo para alejase del mapa y negativo para acercarse.
void	**updateView**() Actualiza la visualización para reflejar cambios en los datos.
void	**zoomIn**() Incrementa el acercamiento y actualiza la visualización.
void	**zoomOut**() Reduce el acercamiento y actualiza la visualización.
void	**zoomToProject**() Ajusta el nivel de acercamiento y centro de tal modo que se visualice todo el proyecto.

Clase Marker. Representa un punto que puede ubicarse sobre el mapa, usualmente simboliza información propia del domino de la aplicación final, logrando de este modo el mashup con la cartografía del servidor. Más detalles de esta clase se muestran en la tabla 49.

Tabla 49. Resumen de la clase Marker.

Resumen de constructores	
Marker(Point coordinate, javax.microedition.lcdui.Image defaultImage) Construye un marcador para un MapPanel, ubicándolo en la coordenada indicada, y representándolo con la imagen especificada.	
Marker(Point coordinate, javax.microedition.lcdui.Image normalImage, javax.microedition.lcdui.Image currentImage) Construye un marcador para un MapPanel, ubicándolo en la coordenada indicada, y representándolo con las imagenes especificadas.	
Resumen de métodos	
Point	**getCoordinate**() Retorna las coordenadas del marcador respecto al mapa.
Image	**getCurrentImage**() Retorna la imagen que representa al marcador cuando es el actual.
MapPanel	**getMapPanel**() Retorna el objeto MapPanel en que se encuentra el marcador.
Image	**getNormalImage**() Retorna la imagen que representa al marcador en su estado normal.
Point	**getScreen**() Retorna un punto en píxeles ubicado en la esquina superior izquierda tieniendo en cuenta en ancho y alto del marcador.
boolean	**isPointInside**(Point point) Indica si un píxel con coordenadas x, y toca al marcador teniendo en cuenta su posición actual en la pantalla.
void	**setCoordinate**(Point coordinate) Establece las coordenadas del marcador respecto al mapa.
void	**setCurrentImage**(javax.microedition.lcdui.Image selectedImage) Establece la imagen que representa al marcador cuando es el actual.
void	**setMapPanel**(MapPanel mapPanel) Establece el MapPanel al que pertenece el marcador.
void	**setNormalImage**(javax.microedition.lcdui.Image defaultImage) Establece la imagen que representa al marcador en su estado normal.

2. PRUEBAS Y RESULTADOS

Las pruebas simulan distintas situaciones para determinar el comportamiento del servidor Atlas, tanto en su estabilidad, capacidad de respuesta y rendimiento. Para este tipo de pruebas se construyen aplicaciones con Java Standard Edition que prueban aspectos específicos de este servidor.

Las pruebas, se realizaron en dos computadores personales, con las especificaciones que se muestran en la tabla 50. En uno de los equipos se instaló el servidor Atlas, equipo sobre el cual recaen las pruebas, mientras que en el otro, se instalaron las distintas aplicaciones construidas para realizar las pruebas, esto se debe al hecho de que cada aplicación de prueba simula varios clientes, consumiendo gran cantidad de recursos del equipo.

Tabla 50. Características de los computadores usados en las pruebas.

Procesador	Intel Core I7 2.30 GHz
Memoria RAM	8 Gb DDR2
Disco Duro	SATA de 1 Tb
Sis. Operativo	Windows 10.
Tarjeta de Red	PCI velocidad 10/100T

2.1. DATOS USADOS EN LAS PRUEBAS.

Los datos utilizados fueron obtenidos a partir del archivo de Autocad ciudad_pasto.dwg que contiene un mapa de la ciudad de Pasto, que representa mediante líneas las comunas, barrios, y manzanas. Además, contiene etiquetas sobre las calles y algunos lugares de referencia. Este archivo pertenece al plan de organización territorial del 2003, para la ciudad de San Juan de Pasto.

Los archivos de Autocad deben ser transformados en archivos apropiados para aplicaciones geográficas, para ello, se usa la herramienta CAD2Shape que entrega ShapeFiles de ESRI, una vez en este formato se realizaron operaciones de translación, además se agregaron atributos que no lograron interpretarse correctamente en el proceso de transformación.

Luego se corrigieron diferentes defectos en la geometría y se procedió a remover información irrelevante. Sin embargo en ciertos casos fue necesario construir capas totalmente nuevas. Como resultado de estos procesos se obtuvieron los archivos puntos.shp, comunas.shp, barrios.shp, manzanas.shp y malla.shp, que se detallan a continuación.

Puntos.shp

La capa puntos.shp (ver figura 42) representa algunos lugares de la ciudad, contiene el nombre de cada lugar, cuenta con un total de 126 geometrías y un tamaño 49.26 KB. Para la construcción de esta capa se tomó algunas de las etiquetas de archivo ciudad_pasto.dwg y se exportaron al formato shp, se eliminaron etiquetas duplicadas e irrelevantes.

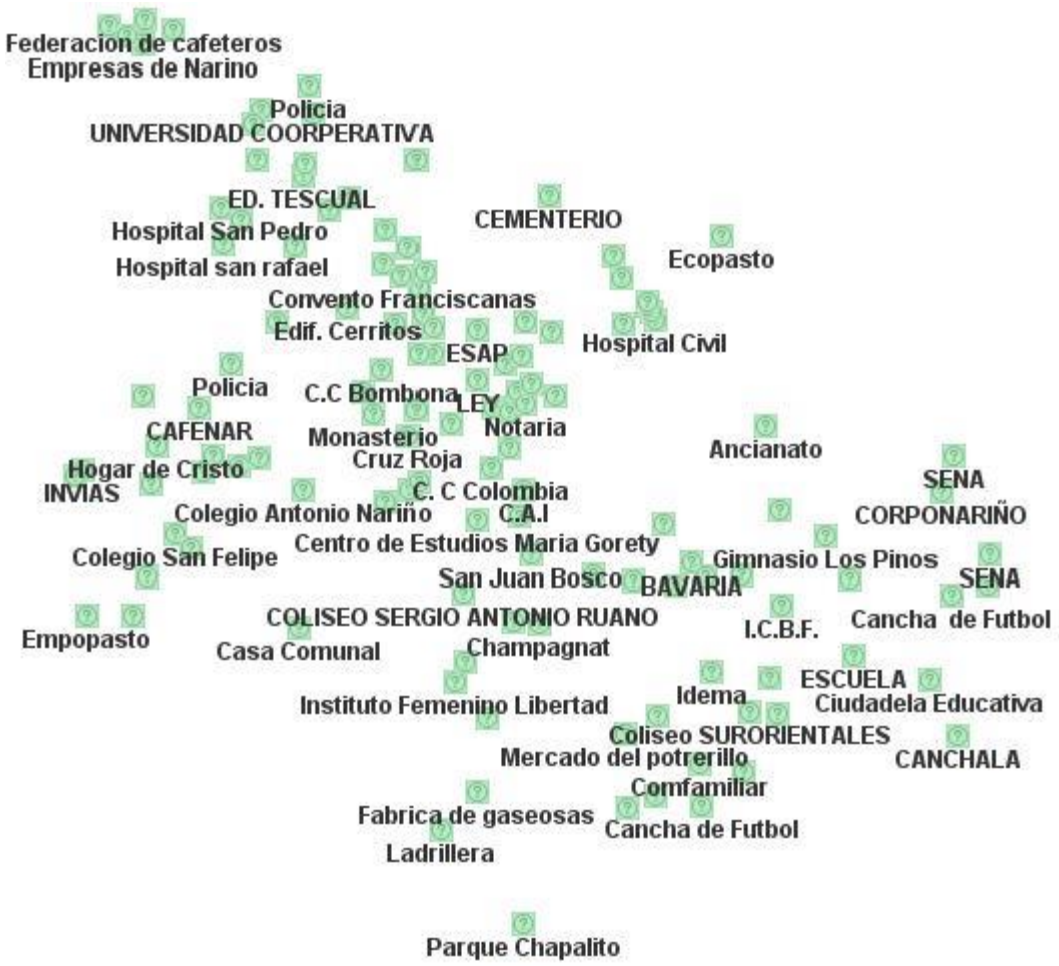

Figura 42. Apariencia de la capa Puntos.shp

Comunas.shp

La capa comunas.shp (ver figura 43) es una geometría tipo polígono, representa las comunas de la ciudad de Pasto, contiene la información de número y nombre de las comunas, cuenta con un total de 12 geometrías y un tamaño 24KB. Para la construcción de esta capa se tomó los polígonos de archivo ciudad_pasto.dwg, se exportaron al formato shp, y se agregaron los atributos manualmente.

Figura 43. Apariencia de la capa Comunas.shp

Barrios.shp

Barrios.shp (ver figura 44) es una geometría tipo polígono, representa los barrios de la ciudad de Pasto, contiene la información de nombre del barrio, y el número de comuna a la cual pertenece, cuenta con un total de 284 geometrías y un tamaño 173KB. Para la construcción de esta capa se tomó los polígonos de los barrios del archivo ciudad_pasto.dwg, se realizó operaciones geométricas entre las etiquetas y los polígonos para agregar los atributos a la capa.

Figura 44. Apariencia de la capa Barrios.shp

Manzanas.shp

La capa manzanas.shp (ver figura 45) es una geometría tipo polígono, representa las manzanas de la ciudad de Pasto, contiene por cada manzana el número del barrio al cual pertenece, cuenta con un total de 4027 geometrías y un tamaño 1.53MB. Para la construcción de esta capa se tomó las líneas del archivo ciudad_pasto.dwg transfomado, se realizó una etapa de edición para completar geometrías y se transformó la capa de líneas a polígonos, además se realizaron operaciones geográficas para agregar los atributos.

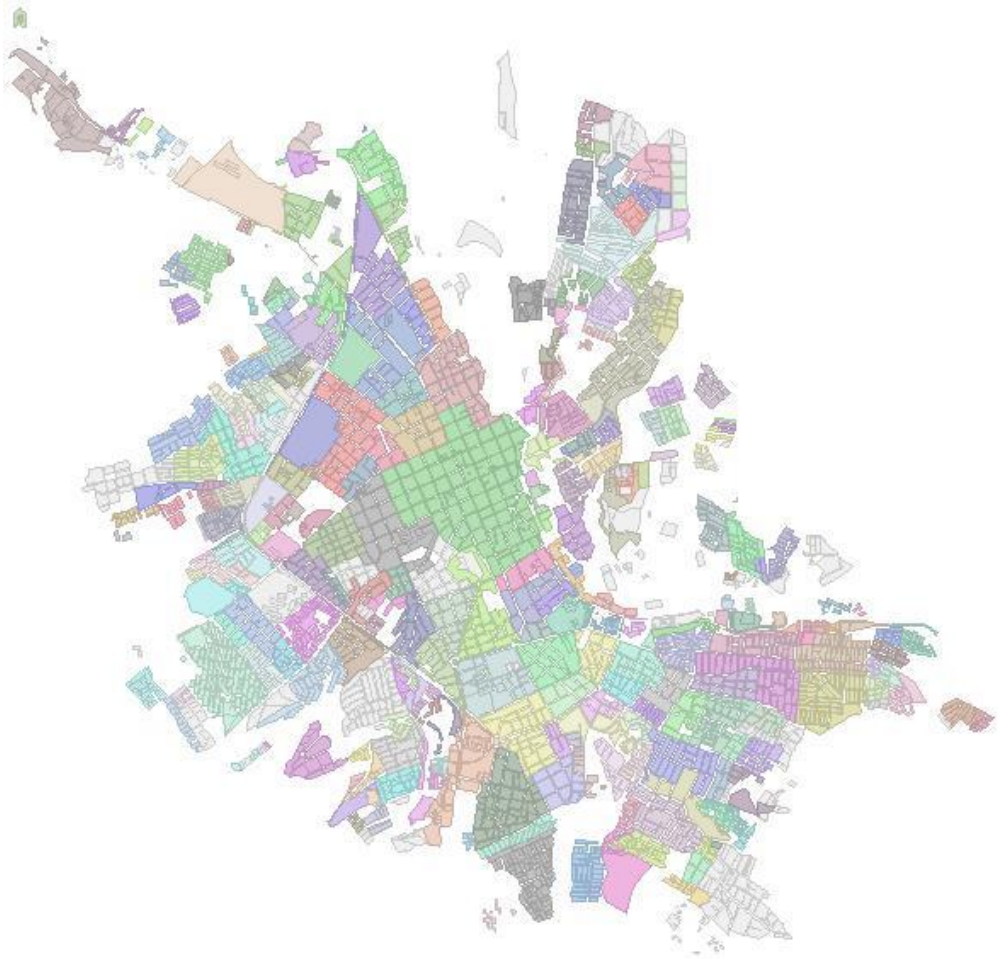

Figura 45. Apariencia de la capa Manzana.shp

Malla.shp

La capa malla.shp (ver figura 46) es una geometría tipo poli línea, representa la malla vial de la ciudad de Pasto, contiene información sobre los nombres de las calles y carreras, cuenta con un total de 214 geometrías y un tamaño 109KB. Para la construcción de esta capa se tomó como referencia los datos de manzanas y etiquetas de las calles de los archivos dwg, con una tableta digitalizadora se dibujó la mayor cantidad de calles y carreras de la ciudad de Pasto que estaban presentes en el material.

Figura 46. Apariencia de la capa Malla.shp

2.2. PRUEBAS SOBRE EL SERVIDOR DE CARTOGRAFÍA.

Las pruebas consisten en medir el desempeño del servidor Atlas mientras este responde peticiones de imágenes, teniendo en cuenta los posibles estados del caché del servidor, así como también la cantidad de clientes simultáneos que realizan las peticiones.

La función primordial de un servidor de cartografía Web es entregar imágenes sobre porciones de mapas, cuando un usuario solicita un mapa a través de un cliente Atlas, este divide la región a visualizar en varias regiones más pequeñas y solicita al servidor una imagen de cada una de ellas, a fin de optimizar el consumo de recursos.

En adelante la imagen de cada región pequeña se denominará “imagen individual” y el conjunto de imágenes que representan la totalidad de la región solicita por el usuario se denominará “imagen compuesta”. Como se muestra en la figura 47.

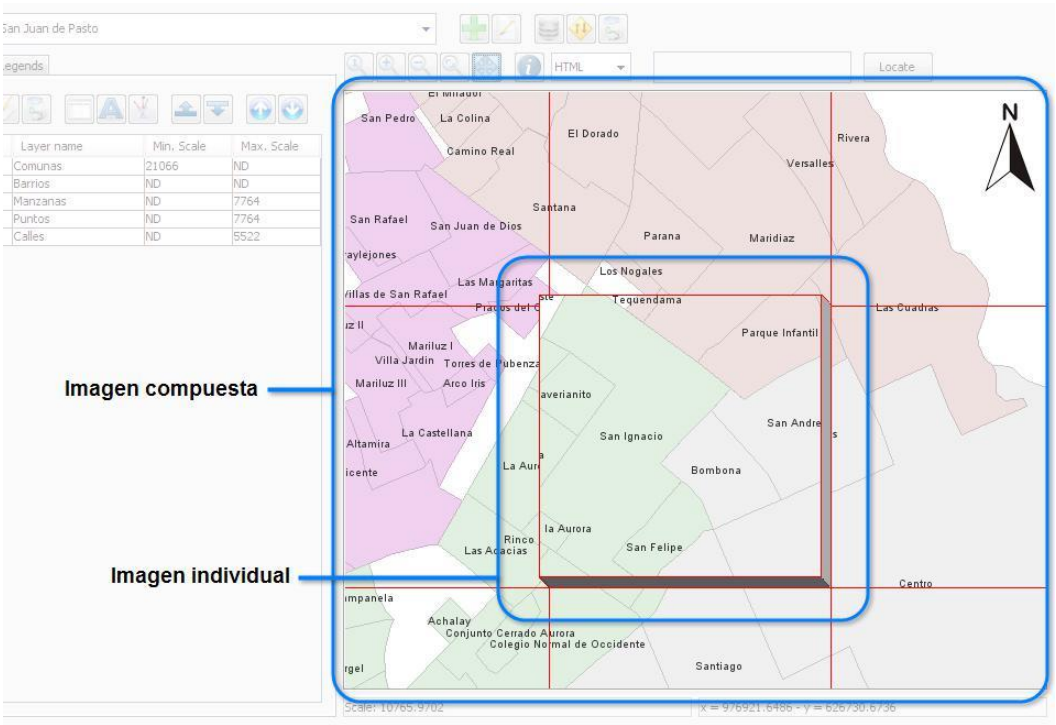

Figura 47. Imagen compuesta e imagen individual

2.2.1. Pruebas de imágenes individuales.

El objetivo de estas pruebas es comprobar el tiempo de generación de imágenes individuales del servidor Atlas, cada imagen contiene datos de las capas puntos, comunas, barrios, manzanas y malla descritas anteriormente, y un tamaño de 256x256 píxeles. Para esta prueba se desarrolló una aplicación que envía peticiones simultáneas al servidor, como se muestra en la figura 48.

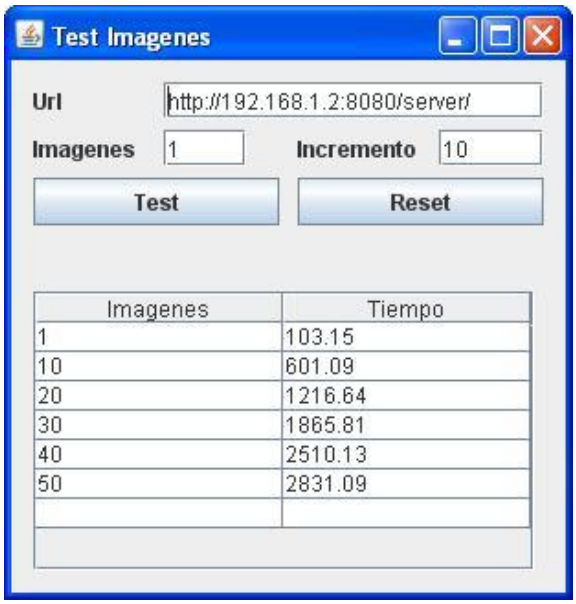

Figura 48. Apariencia de la aplicación para pruebas individuales

Prueba No 1. Esta prueba consiste en enviar un número de peticiones de “imágenes individuales” aleatorias, simutaneamente al servidor Atlas y por cada una, medir el tiempo de respuesta del servidor con la opción de caché deshabilitada. En la tabla 51 se muestra los resultados de esta prueba comparados con un servidor B de comportamiento constante, es decir, un servidor que responde las peticiones de igual manera, sin importar el número de peticiones simultáneas, La figura 49 muestra los datos en una gráfica de dispersió n.

Tabla 51. Tiempos en pruebas de imágenes individuales sin caché.

Imágenes	Tiempos Atlas (ml)	Tiempos B (ml)
1	103,15	103,15
10	601,09	103,15
20	1216,64	103,15
30	1865,81	103,15
40	2510,14	103,15
50	2831,09	103,15
60	3292,68	103,15
70	3731,48	103,15
80	4146,08	103,15
90	4339,85	103,15
100	4986,94	103,15

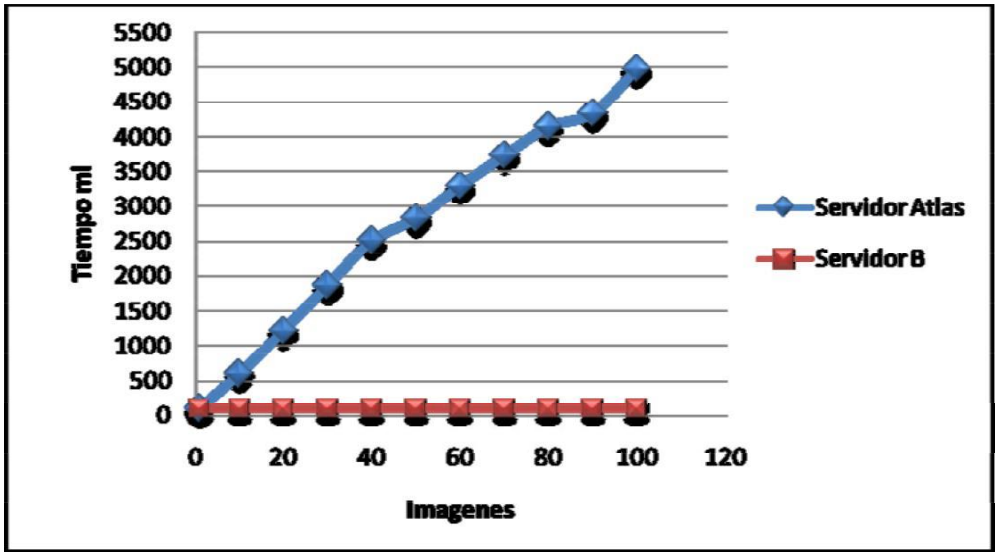

Figura 49. Tiempos en pruebas de imágenes individuales sin caché

Al deshabilitar la opción de caché del servidor Atlas, se observa de forma más drástica el efecto de la concurrencia en los tiempos de respuesta del servidor, esto se debe a que cada imagen individual debe ser dibujada, incrementando el consumo de recursos. Sin embargo, este tiempo adicional se ve más que compensado con el incremento en la capacidad del servidor al poder contestar peticiones simultáneamente.

Prueba No 2. Esta prueba consiste en enviar un número de peticiones de "imágenes individuales" aleatorias, simultáneamente al servidor Atlas y por cada una, medir el tiempo de respuesta del servidor con la opción de caché habilitada, para esta prueba en particular el caché se encuentra en formación. La tabla 52 muestra los resultados de esta prueba comparados con un servidor B de comportamiento constante, es decir, un servidor que responde las peticiones de imágenes de igual manera, sin importar el número de peticiones simultáneas, La figura 50 muestra los datos en una gráfica de dispersión.

Tabla 52. Tiempos en pruebas de imágenes individuales con caché en formación.

Clientes	Tiempos Atlas (ml)	Tiempos B (ml)
1	8,60	8,6
10	11,49	8,6
20	6,64	8,6

30	6,72	8,6
40	6,00	8,6
50	7,26	8,6
60	5,77	8,6
70	9,56	8,6
80	6,34	8,6
90	6,17	8,6
100	6,12	8,6

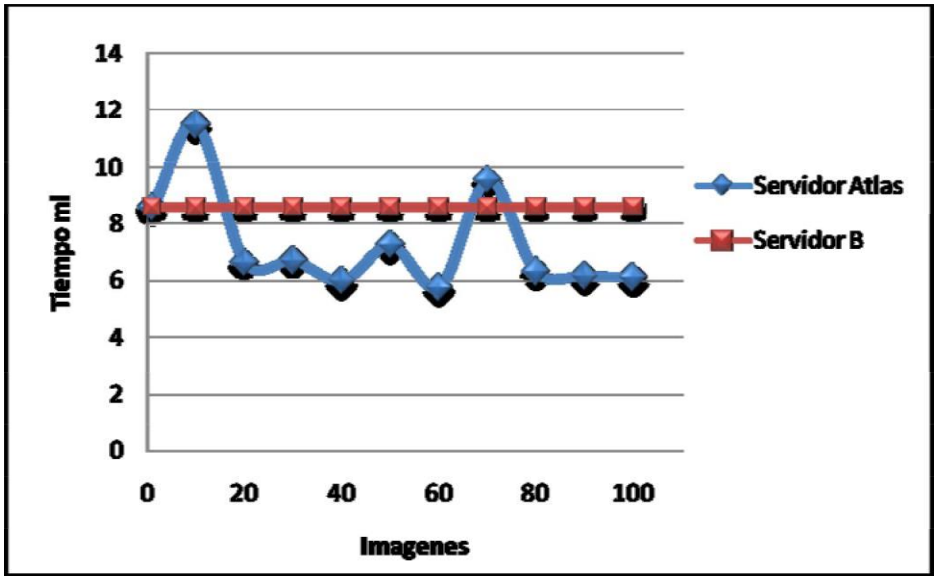

Figura 50. Tiempos en pruebas de imágenes individuales con caché en formación

Al habilitar la opción de caché, que se encuentra en formación se observa como el tiempo de respuesta se mantiene prácticamente independiente del número de imágenes individuales simultáneas que se soliciten, De esta prueba se puede concluir que para mejorar el rendimiento del servidor Atlas es necesario habilitar el uso de caché.

Prueba No 3. Esta prueba consiste en enviar un número de peticiones de “imágenes individuales” aleatorias, simultáneamente al servidor Atlas y por cada una, medir el tiempo de respuesta del servidor con la opción de caché habilitada, para esta prueba el caché ya se encuentra formado. La tabla 53 muestra los resultados de esta prueba comparados con un servidor B que no soporta peticiones simultáneas. La figura 51 muestra los datos en una gráfica de dispersión.

Tabla 53. Tiempos en pruebas de imágenes individuales con caché formado.

Imágenes	Tiempos Atlas (ml)	Tiempos B (ml)
1	9,35	8,55
10	29,7	85,5
20	39,1	171
30	53,1	256,5
40	72,65	342
50	85,95	427,5
60	102,35	513
70	116,45	598,5
80	132,8	684
90	153,1	769,5
100	178,9	855

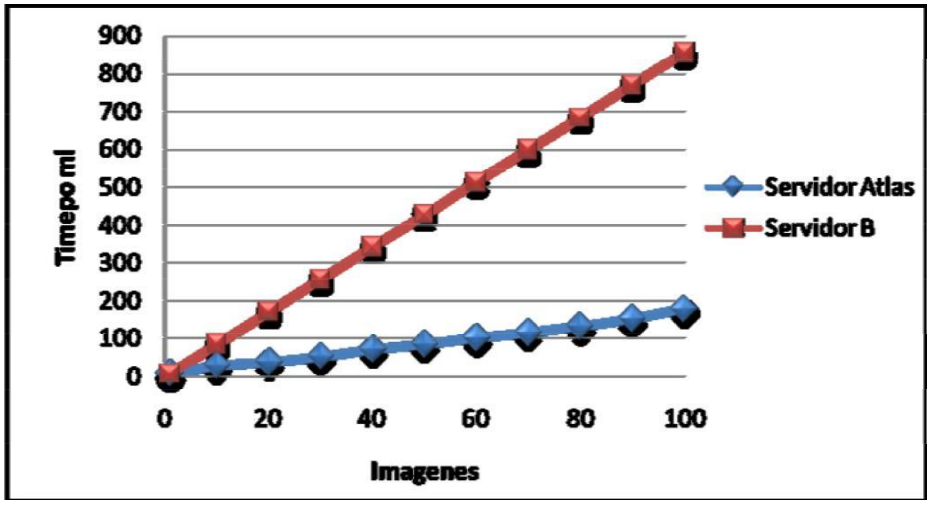

Figura 51. Tiempos en pruebas de imágenes individuales con caché formado

Cuando el caché de servidor Atlas se encuentra formado se observa como la capacidad de responder peticiones simultáneas del servidor Atlas permite obtener mejores tiempos de respuesta. El hecho de contar con un caché formado permite al servidor responder satisfactoriamente frente al aumento de concurrencia.

2.2.2. Pruebas de simulación de clientes.

El objetivo de estas pruebas es comprobar el comportamiento del servidor Atlas al interactuar con clientes simultáneos, los cuales envían un conjunto de peticiones de imágenes individuales para formar una “imagen compuesta” (ver figura 52). Para esta prueba se desarrolló una aplicación que simula el comportamiento de varios clientes, con base en la biblioteca de desarrollo para aplicaciones Java SE, como se muestra en la figura 53.

Cada cliente simulado realiza peticiones de “imágenes compuestas” aleatorias, en diferentes posiciones y niveles de zoom, en total cada “imagen compuesta” tiene un tamaño de 640 x 480 píxeles.

Cada una de las “imágenes compuestas” solicitadas al servidor Atlas, estan conformadas por varias de “imágenes individuales”, entre 8 y 15, que gracias a diferentes operaciones realizadas por elementos del componente de desarrollo se repiten, inclusive entre diferentes clientes.

Esto, con el objetivo de analizar al servidor y la experiencia que puede entregar a los usuarios finales que lo consultan mediante clientes Atlas.

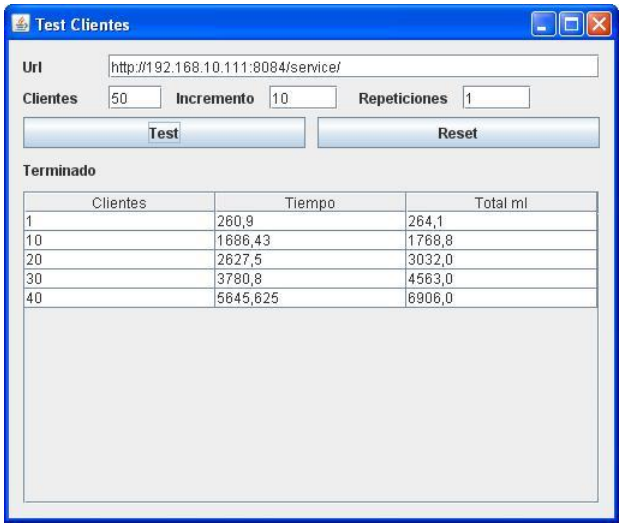

Figura 52. Apariencia de la aplicación para pruebas de clientes

Prueba No 1. Esta prueba consiste en enviar peticiones de “imágenes compuestas” aleatorias, simutaneamente al servidor Atlas, incrementado el número de clientes. Por cada imagen compuesta se mide el tiempo de respuesta del servidor con la opción de caché deshabilitada, En la tabla 54 se muestran los resultados de esta prueba, comparados con aquellos obtenidos al usar el servidor Atlas con el caché habilitado y formado. La figura 53, muestra los datos en una gráfica de dispersión.

Tabla 54. Tiempos en pruebas con clientes simulados A.

Clientes	Tiempos Sin Cache (ml)	Tiempos con Cache formado (ml)
1	148,5	28,1
10	1150	101,5
20	1800	229,7
30	2618,8	360,9
40	3185,9	517,2
50	3879,7	632,8
60	4456,2	739,1
70	5242,2	885,9
80	6057,8	971,9
90	6762,5	1136
100	7654,7	1225

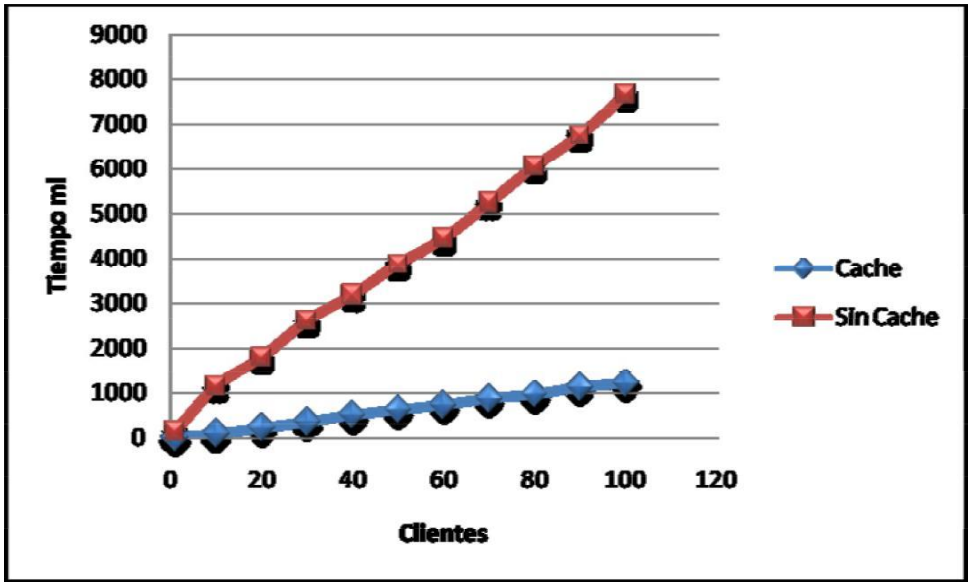

Figura 53. Tiempos en pruebas con clientes simulados A

Con la opción de caché del servidor Atlas deshabilitada se observa como el servidor emplea más tiempo en responder una “imagen compuesta” a los clientes, sin embargo, al habilitar el caché de servidor, el tiempo de respuesta mejora sustancialmente. De esta prueba se puede concluir que para manejar clientes el servidor Atlas necesita habilitar la opción de caché.

Prueba No 2. Esta prueba consiste en enviar peticiones de “imágenes compuestas” aleatorias, simutaneamente al servidor Atlas, incrementado el número de clientes. Por cada imagen compuesta se mide el tiempo de respuesta del servidor con la opción el caché habilitado y en formación. En la tabla 55 se muestran los resultados de esta prueba, comparados con aquellos obtenidos al usar el servidor Atlas con el caché habilitado y formado. La figura 54, muestra los datos en una gráfica de dispersión.

Tabla 55. Tiempos en pruebas con clientes simulados B.

Clientes	Tiempos con Cache en formación (ml)	Tiempo con Cache formado (ml)
1	289	28,1
10	567,1	101,5
20	473,4	229,7
30	493,7	360,9
40	551,6	517,2
50	651,5	632,8
60	754,6	739,1
70	900	885,9
80	996,8	971,9
90	1096,9	1136
100	1262,5	1225

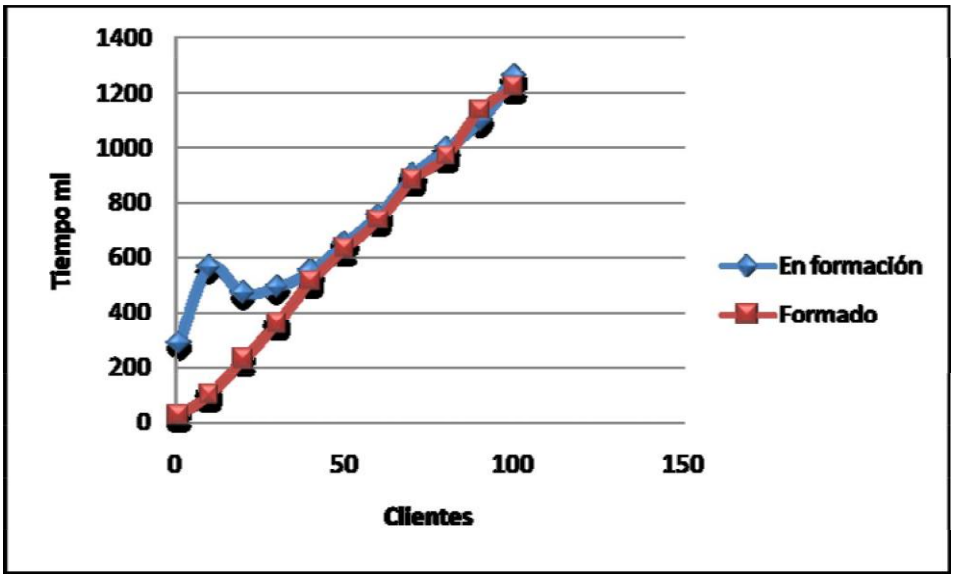

Figura 54. Tiempos en pruebas con clientes simulados B

Con la opción de caché del servidor Atlas habilitada y en formación se observa como la técnica empleada por los componentes de desarrollo, en la que una “imagen compuesta” está divida en varias “imágenes individuales” más pequeñas, beneficia notablemente el rendimiento general de sistema, ya que las peticiones recaen sobre las mismas imágenes y el caché entra en acción economizando gran cantidad de recursos.

2.3. PRUEBAS SOBRE EL SERVIDOR DE GEOCODIFICACIÓN

El objetivo de estas pruebas es comprobar el tiempo de respuesta y efectividad del servidor de geocodificación, así como también, analizar la cobertura de la fuente de datos.

La fuente de datos utilizada es malla.shp, la cual se tomó como material de referencia para el plugin de geocodificación Atlas Street Geocoder. Dado que la operación de geocodificación es una tarea compleja, la efectividad del geocoder se mide respecto al número de casos en que este entrega alguna respuesta.

Las direcciones alfanuméricas para estas pruebas se obtuvieron de la base de datos de empresas afiliadas a la cámara de comercio de San Juan de Pasto del 2004.

Prueba No 1. Esta prueba consite en enviar varias peticiones al servidor de **g**eocodificación y medir por cada una de ellas en tiempo de respuesta. Todas las peticiones enviadas corresponden al formato requerido por el geocoder, de modo tal, que sobre ellas se realice el proceso completo de geocodificación. Para esta prueba se desarrolló una aplicación que envía peticiones simultáneas al servidor de geocodificación (ver figura 55). Los tiempos obtenidos se muestran en la tabla 56 y en la figura 56 se muestra los datos en una grafica de dispersión.

Figura 55. Apariencia de la aplicación para pruebas de geocodificación

Tabla 56. Tiempos de respuesta a peticiones simultáneas de geocodificación.

Peticiones	Tiempo (ml)
1	1,60
32	48,10
64	87,31
96	140,50
128	228,88
160	250,58
192	307,53
224	423,62
256	407,51
288	535,64
320	585,09

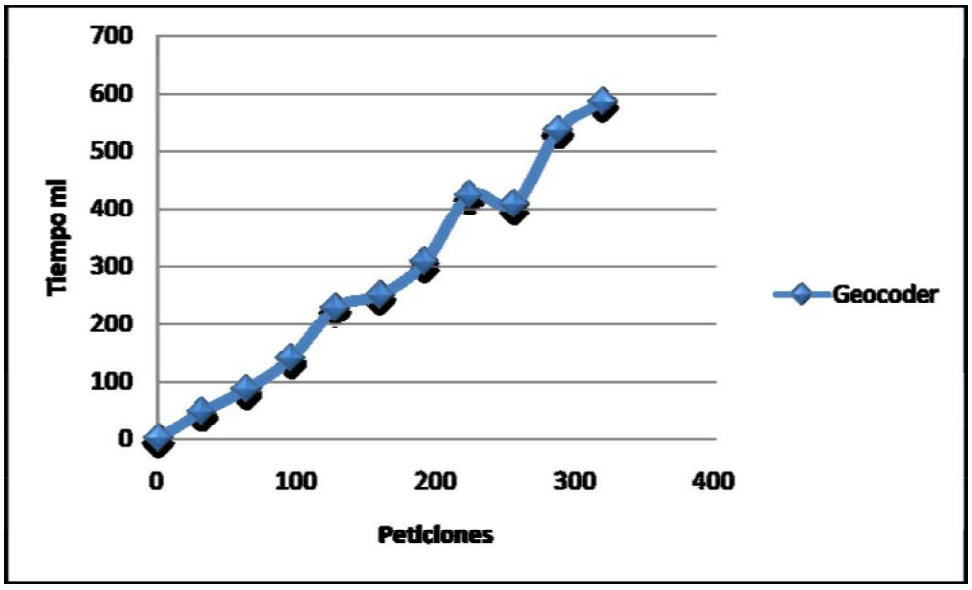

Figura 56. Tiempos de respuesta a peticiones simultáneas de geocodificación

Gracias a la tecnología Java EE, se observa como la administración de peticiones simultáneas del servidor Atlas, en conjunto con la eficiencia del Atlas Street Crossing Geocoder, permiten al servidor Atlas entregar resultados de geocodificación en tiempos menores a un segundo para más de 300 peticiones simultáneas.

Prueba No 2. Esta prueba consiste en enviar un total de 8489 peticiones al servidor de geodificación con direcciones de empresas ubicadas en diferentes sitios de la ciudad de Pasto tomadas de la base de datos de afiliados a la cámara de comercio del 2004. El objetivo de esta prueba de análisis de datos es determinar el porcentaje de direcciones que cumplen el formato de calle carrera, requerido por el gecoder Atlas Street Crossing Geocoder. Los resultados se presentan en la tabla 57. La figura 57 muestra los datos en una gráfica circular.

Tabla 57. Porcentajes de direcciones de la ciudad.

	Direcciones	Porcentaje
Total	8489	100 %
Formato requerido	7519	89 %
Otros formatos	970	11 %

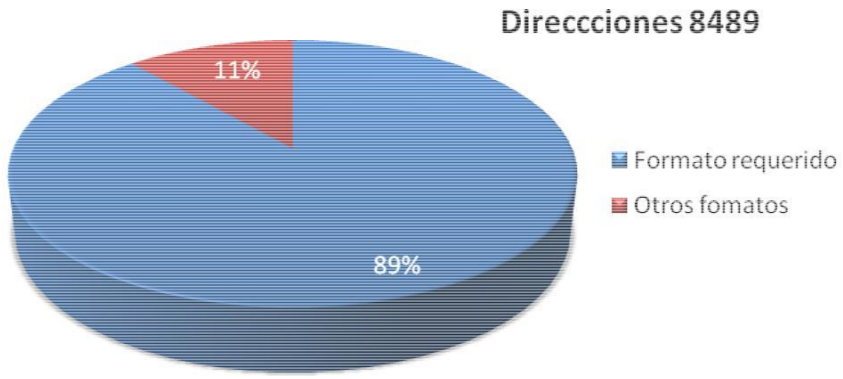

Figura 57. Porcentaje de Direcciones de la ciudad

Dado que el algoritmo Atlas Street Crossing Geocoder se diseño e implemento con base en la nomenclatura oficial de la ciudad de Pasto se ve que la cobertura del servidor de geocodificación para el sector empresarial de la ciudad es satisfactoria.

Prueba No 3. Esta prueba consiste en enviar un total de 5792 peticiones al servidor de geodificación con direcciones de empresas ubicadas en diferentes sitios de las comunas centro y centro sur tomadas de la base de datos de afiliados a la cámara de comercio del 2004. El objetivo de esta prueba de análisis de datos es determinar el porcentaje de direcciones que cumplen el formato de calle carrera, requerido por el gecoder Atlas Street Crossing Geocoder. Los resultados se presentan en la tabla 58. La figura 58 muestra los datos en una gráfica circular.

Tabla 58. Porcentajes de direcciones comuna centro.

	Direcciones	Porcentaje
Total	5792	100 %
Reclama	5258	91 %
No reclama	534	9 %

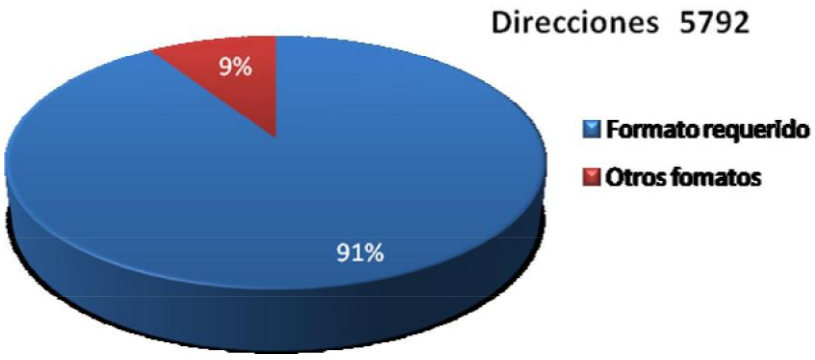

Figura 58. Porcentajes de direcciones comuna centro

Dado que el algoritmo Atlas Street Crossing Geocoder se diseño e implemento con base en la nomenclatura oficial de la ciudad de Pasto se ve que la cobertura del servidor de geocodificación es mayor en la comuna centro y centro sur de la ciudad de Pasto, ya que en ella predomina la nomenclatura oficial de la ciudad, es decir, de calles y carreras.

Prueba No 4. Esta prueba consiste en enviar un total de 7519 peticiones al servidor de geocodificación con direcciones de empresas ubicadas en diferentes sitios de la ciudad de Pasto, de las que se sabe cumplen con el formato de calles y carreras requerido por el geocoder. Estas direcciones fueron tomadas de la base de datos de afiliados a la cámara

de comercio del 2004. El objetivo de esta prueba es determinar el porcentaje de direcciones que el servidor es capaz de responder satisfactoriamente. Los resultados se presentan en la tabla 59. La figura 59 muestra los datos en una gráfica circular.

Tabla 59. Direcciones geocodificadas en la ciudad.

	Direcciones	Pocentajes
Total	7519	100 %
Respuesta	5670	75 %
Sin respuesta	1849	25 %

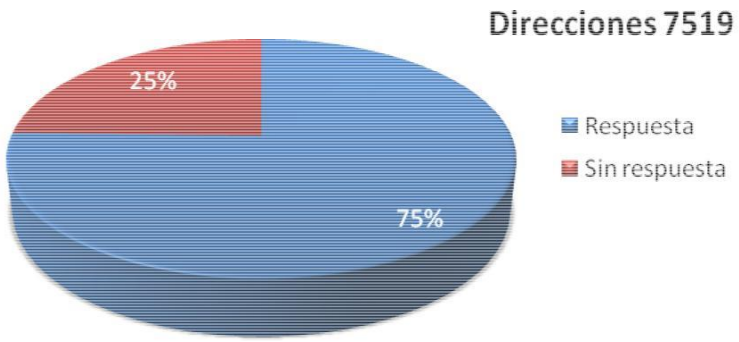

Figura 59. Direcciones geocodificadas en la ciudad

Dado que las peticiones enviadas al servidor con direcciones ubicadas en diferentes lugares de la ciudad de Pasto cumplen el formato requerido por el geocoder, se puede concluir que la eficacia de este, es adecuada para la ciudad de Pasto en general. Cabe señalar que la eficacia del geocoder depende en gran medida de la calidad, diversidad y cobertura del material de referencia.

*Prueba No 5***.** Esta prueba consiste en enviar un total de 5258 peticiones al servidor de geocodificación con direcciones de empresas ubicadas en diferentes sitios de las comunas centro y centro sur, de las que se sabe, cumplen con el formato de calles y carreras requerido por el geocoder. Estas direcciones fueron tomadas de la base de datos de afiliados a la cámara de comercio del 2004. El objetivo de esta prueba es determinar el porcentaje de direcciones que el servidor es capaz de responder satisfactoriamente. Los resultados se presentan en la tabla 60. La figura 60 muestra los datos en una gráfica circular.

Tabla 60. Direcciones geocodificadas en la comuna centro

	Direcciones	Porcentajes
Total	5258	100 %
Respuesta	4278	81 %
Sin respuesta	980	19 %

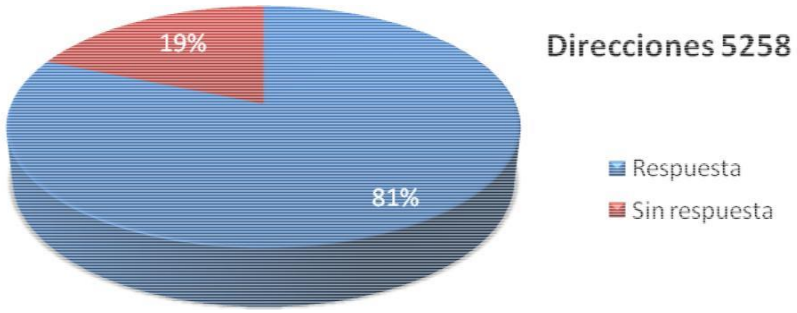

Figura 60. Direcciones geocodificadas en la comuna centro

Dado que las peticiones enviadas al servidor con direcciones ubicadas en las comunas, centro y centro sur de la ciudad de Pasto, cumplen el formato requerido por el geocoder, se puede concluir que la eficacia de este, es mayor en estas comunas. Esto se debe a la mayor cantidad de datos con que se cuenta en el material de referencia sobre estas zonas.

3. INSTALACIÓN DE LAS HERRAMIENTAS ATLAS

Ejecutar el instalador atlas_setup.exe.

Primero debe seleccionarse el idioma de la interfaz del instalador, como se muestra en la figura 61.

Figura 61. Selección de idioma en el instalador de Atlas

A continuación el instalador presenta una pantalla de bienvenida como en se muestra en la figura 62.

Figura 62. Pantalla de bienvenida en el instalador de Atlas

Luego, debe aceptarse los términos de la licencia GNU, como se muestra en la figura 63.

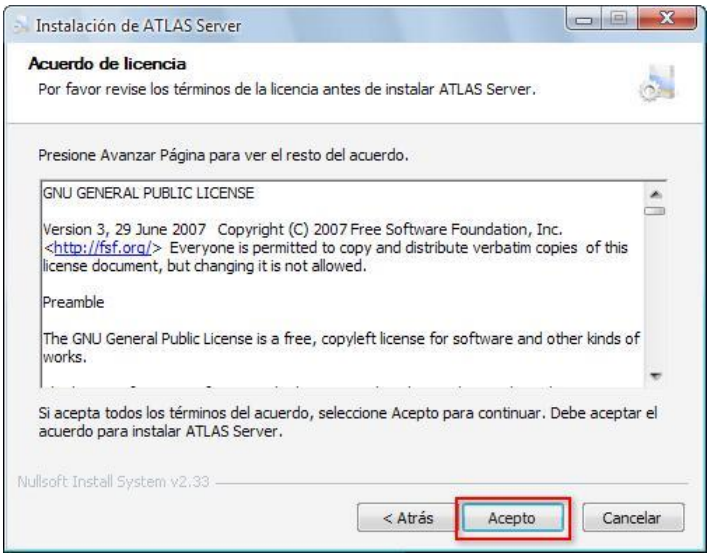

Figura 63. Aceptación de licencia en el instalador de Atlas

Luego debe seleccionarse los componentes a instalar, para este caso se instalarán todos, como se muestra en la figura 64.

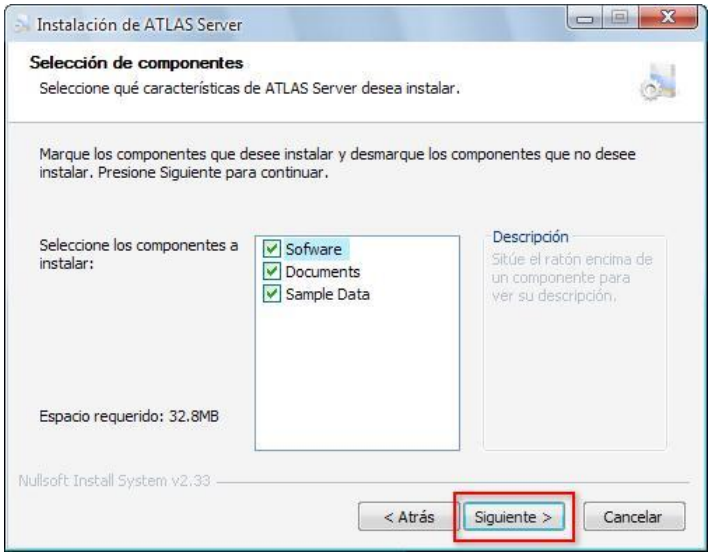

Figura 64. Selección de componentes en el instalador de Atlas

A continuación se selecciona la ruta para la instalación de los archivos, esta puede ser modificada, sin embargo, en este caso se tomará la opción por defecto, como se muestra en la figura 65.

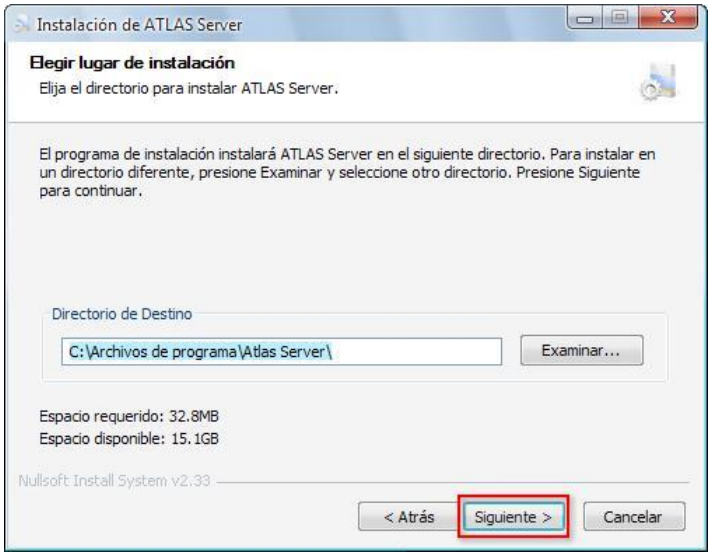

Figura 65. Selección de ruta instalación en el instalador de Atlas

A continuación se debe seleccionar el grupo de menú inicio donde se crearán los accesos directos, como se muestra en la figura 66.

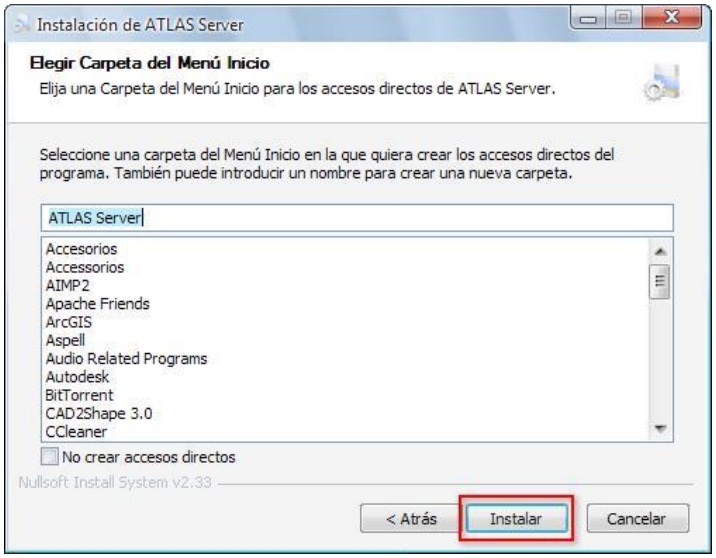

Figura 66. Selección de grupo en el menú inicio en el instalador de Atlas

Luego inicia la descompresión de los archivos y una vez finalizada, ha concluido el proceso de instalación. En el menú inicio se encuentra un grupo como el que se muestra en la figura 67.

Figura 67. Carpeta Atlas en el menú inicio de Windows

3.1. INSTALACIÓN DEL SERVIDOR ATLAS.

Debe iniciarse la aplicación desde el menú inicio como se muestra en la figura 68.

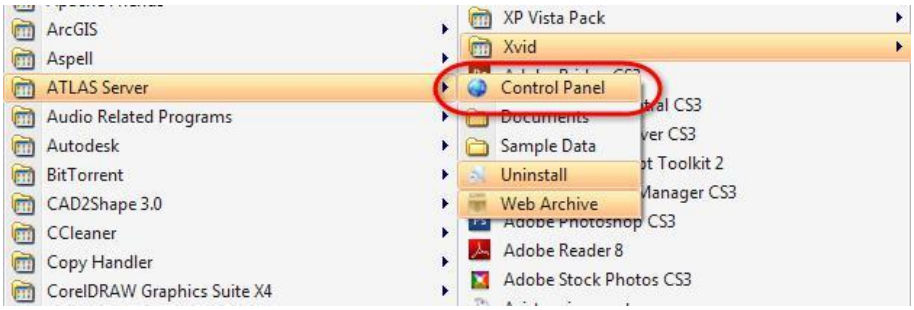

Figura 68. Iniciar el panel de control Atlas

Una vez inciada la aplicación, en la parte superior se encuentra en botón “WEB Archive”, (ver figura 69) que lanzará una herramienta que permite la preparación del archivo WAR que se llevará posteriormete al servidor.

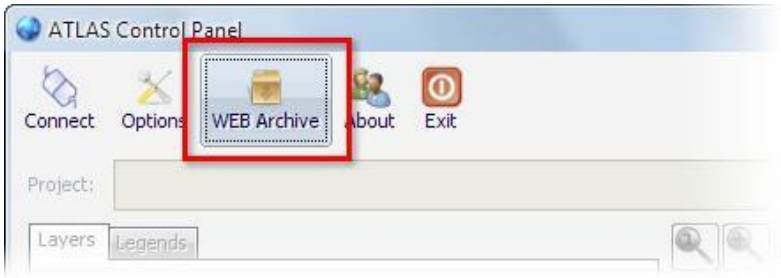

Figura 69. Iniciar el WEB Archive

La ventana que se presenta (ver figura 70) permite establecer opciones generales sobre la forma en que se intalará la aplicación en el contenedor de servlets y las opciones de seguridad y caché.

El campo **“Deploy Path”**, corresponde a ruta dentro del contenedor que tendrá la aplicación.

Los campos **“WMS Servlet URL”** y **“Geocoder Servlet URL”**, son informativos, y muestran las rutas de los servidores de catografia y geocodificación respectivamente.

Los campos **“User Name”** y **“Password”** corresponden a los datos que serán solicitados al momento en que alguien desee modificar el contenido del servidor mediante el panel de control.

El campo **“Cache Size”**, permite establecer el tamaño del caché en mega bytes, el caché resulta de gran ayuda para el rendimiento del servidor, para deshabilitarlo, basta colocar este campo en “0” , pero se recomienda mantenerlo habilitado y tan alto como sea posible.

Todos los campos de este formulario deben ser diligenciados, una vez hecho esto de presiona **“Next”**, o **“Exit”** en caso de que se desee cancelar el proceso.

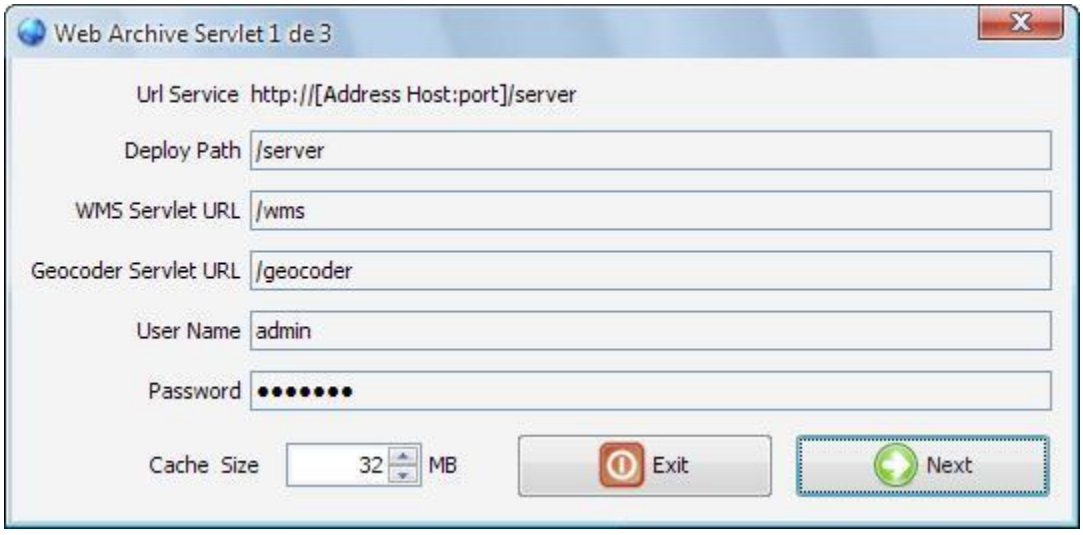

Figura 70. Primer paso del asistente para configuración de archivos WAR

Una vez completado el paso anterior debe mostrarse el formulario de paso dos, que se muestra en la figura 71.

Este formulario permite establecer las opciones de conexión entre el contenedor de servlet y la base de datos PostgreSQL.

El campo **“IP Adress”**, corresponde a la dirección IP del servidor PosgtreSQL y debe ser diligenciado.

El campo **“DataBase Name”**, corresponde al nombre de la base de datos donde se instalarán las tablas del servidor Atlas, este campo debe ser diligenciado.

El campo **“Port”** corresponde al puerto donde funciona el servidor PostreSQL, por defecto 5432, el valor puede ser modificado, y al final debe contener algún valor.

Los campos **“User Name”** y **“Password”** corresponden a los datos de seguridad de un usuario PostgreSQL que tenga autorización para modificar la estructura de la base de datos indicada en el campo **“DataBaseName”**.

Una vez diligenciados los datos requeridos debe hacerse clic en **“Next”** para continuar al paso final, **“Back”**, para regresar a editar las opciones de generales de configuración o **“Exit”** para abandonar el proceso.

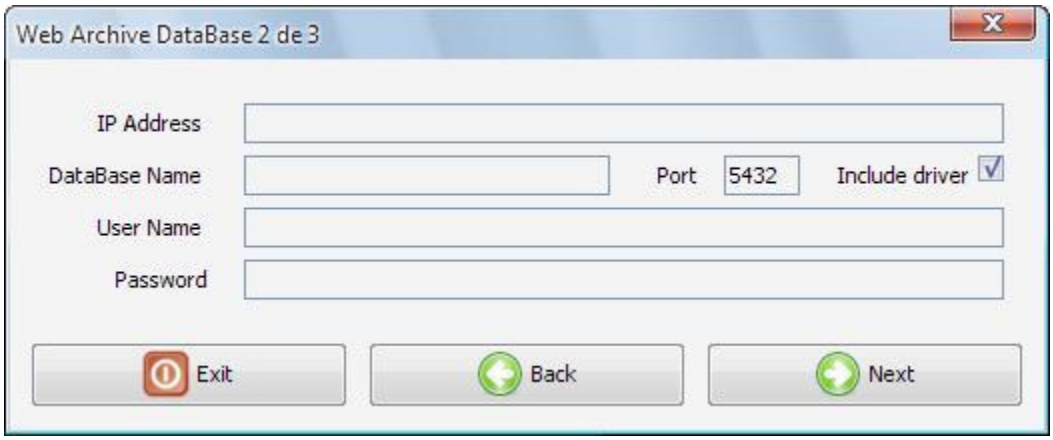

Figura 71. Segundo paso del asistente para configuración de archivos WAR

El último paso en el proceso consiste en establecer el listado de plugins que se instalarán en el servidor Atlas (ver figura 72).

Si se desea agregar un plugin, debe hacerse clic en el botón **“Add”**, este despliega el diálogo de abrir archivos, mostrando solo archivos .JAR, una vez ahí se debe indicar la ruta al plugin y este aparecerá en la lista, si el JAR seleccionado no es un plugin de geocodificación Atlas, se notificará la situación.

Si se desea remover algún plugin que se encuentre en las lista debe seleccionarse, y entonces, hacer clic sobre el botón **"Remove"**.
No es obligatorio agregar plugins, si se desea, se puede proceder directamente a la generación del archivo WAR, para ello, debe hacerse clic sobre el botón **"Generate"**.
A continuación se despliega el diálogo de guardar archivos, y una vez seleccionada la ruta el sistema coloca en ella el archivo resultante.

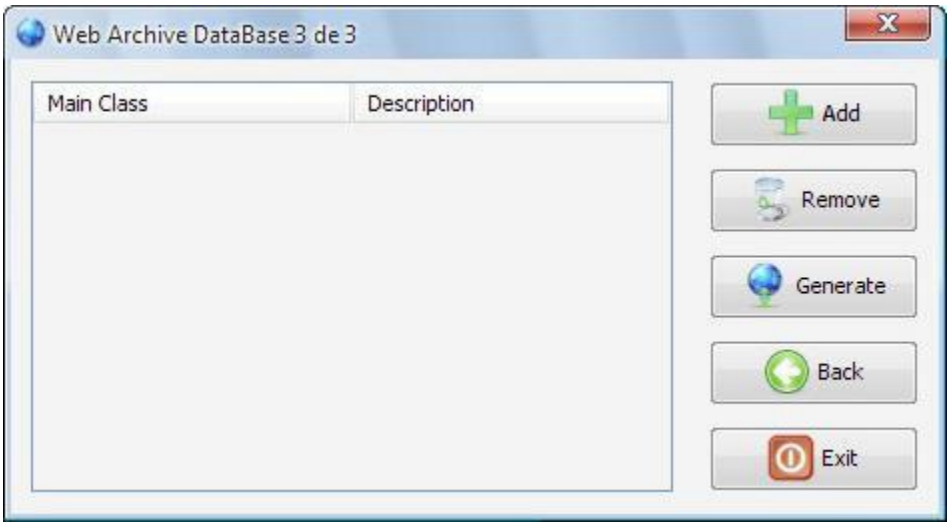

Figura 72. Tercer paso del asistente para configuración de archivos WAR

Ahora el archivo generado puede ser instalado en el contenedor de servlets. Para este caso se cubre el proceso de instalación bajo Apache Tomcat. Tomcat debe estar configurado y corriendo.
Al ingresar en la dirección del servidor Tomcat, para este caso, http://localhost:8080/, debe obtenerse una pantalla de bienvenida como se muestra en la figura 73. Una vez ahí debe hacerse clic sobre la opción "Tomcat Manager".

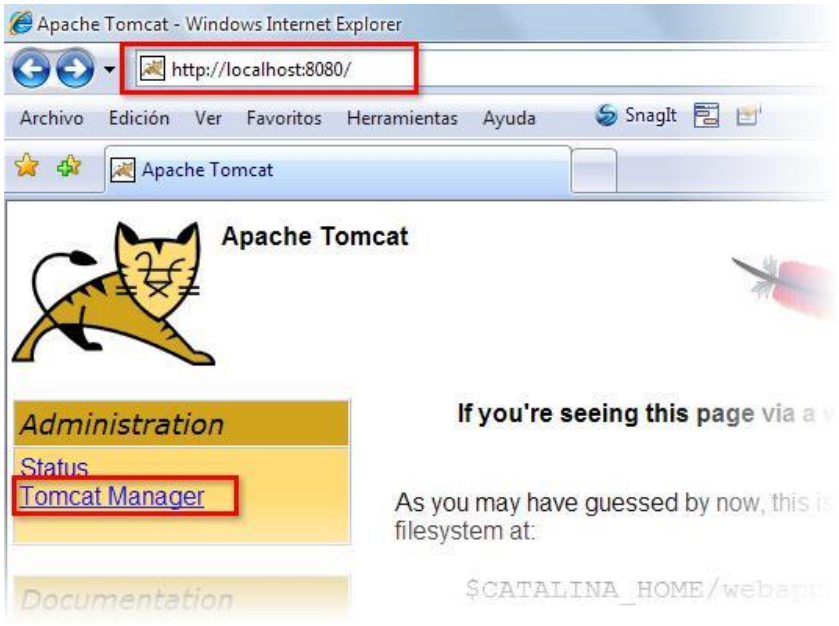

Figura 73. Página principal de Tomcat

El servidor procede a solicitar la contraseña del “manager” del sistema que se estableció en el archivo users.xml, durante el proceso de instalación.

Una vez ingresados estos datos, se carga el administrador de aplicaciones Tomcat, que se muestra en la figura 74.

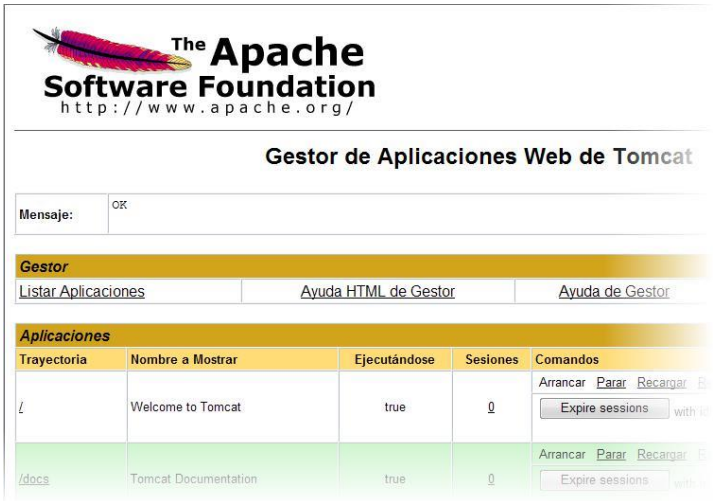

Figura 74. Gestor de aplicaciones Web de Tomcat

Una vez ahí, hay que dirigirse a selección de “Archivo WAR a desplegar” y hacer clic en el botón “examinar” en el cuadro de diálogo que aparece, se de seleccionar el archivo creado en el paso anterior y luego “desplegar”, como se muestra en la figura 75.

Figura 75. Sección, archivo WAR a desplegar

Si todo ha salido bien, la página debe recargarse y la aplicación debe aparecer en la lista, como se muestra en la figura 76.

Figura 76. Aplicación correctamente instalada en el servidor Tomcat

3.2 ADMINISTRACIÓN DEL SERVIDOR ATLAS.

Esta guía muestra como configurar servidor Atlas, usando la herramienta panel de control. Una vez instalado el servlet en el contenedor, tal como se describió en la guía de instalación del servidor Atlas, se puede administrar el contenido y la forma en opera el servidor Atlas.

Bajo Windows, debe iniciarse la aplicación desde el menú inicio como se muestra en la figura 77.

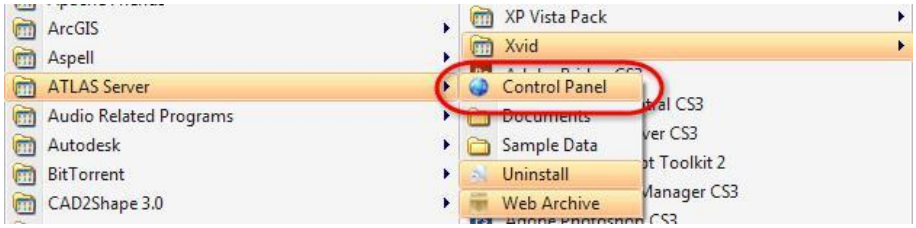

Figura 77. Iniciar el panel de control Atlas

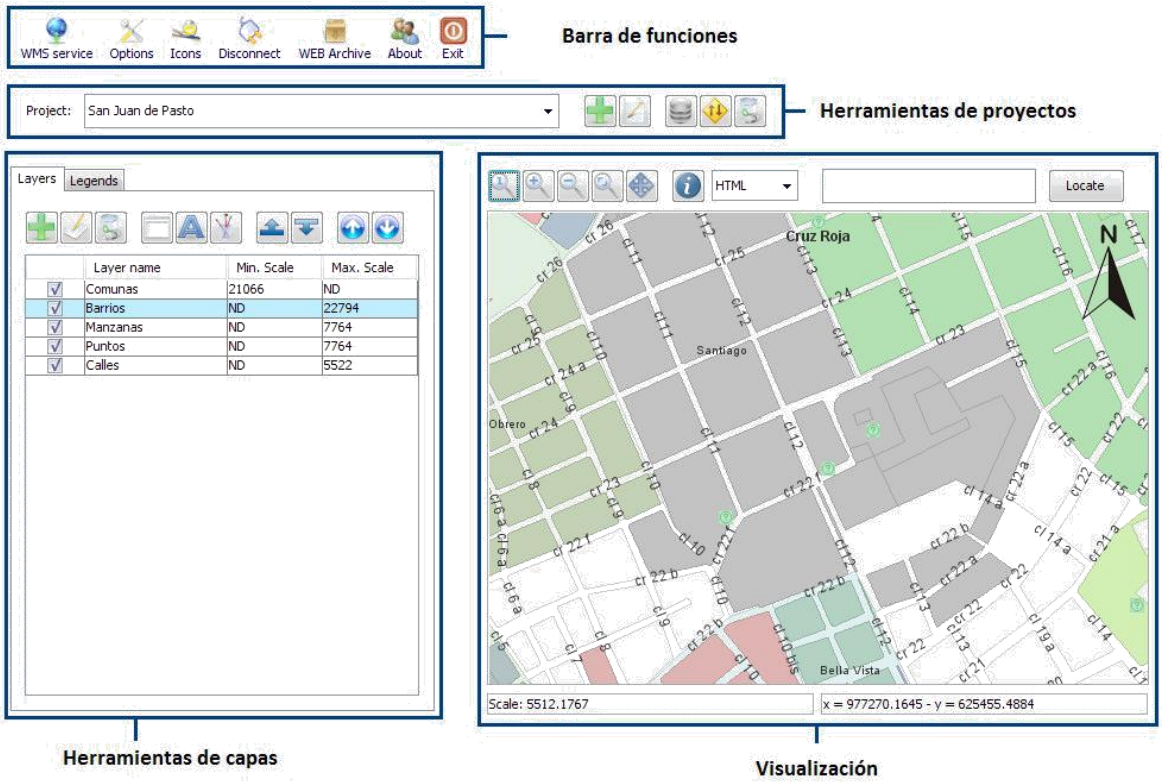

Figura 78. Distribución de componentes del panel de control

Una vez iniciada la herramienta se puede observar la ventana principal de esta, que está dividida en varias secciones, como se muestra en la figura 78.

La barra de funciones, contiene botones para ejecutar procedimientos que afectan a todo el servidor y al panel de control, estas funciones se detallan más adelante y la apariencia de la barra de muestra en la figura 79.

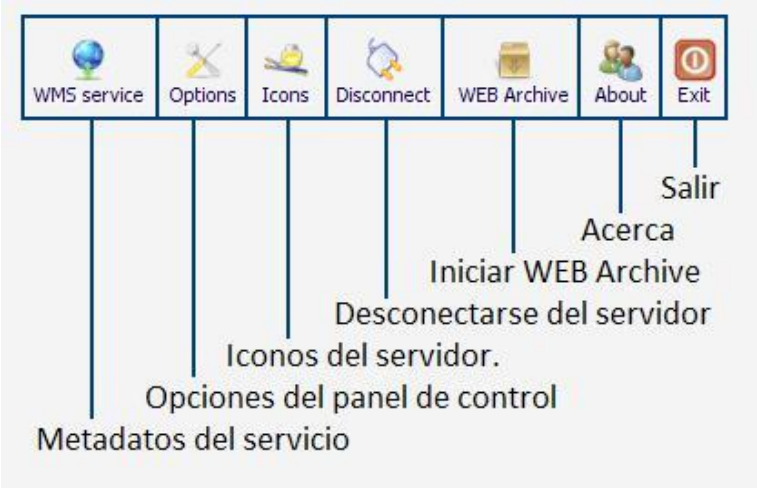

Figura 79. Barra de funciones del panel de control

La barra de proyectos, contiene funciones para seleccionar el proyecto actual y aplicar diferentes operaciones sobre él, como se muestra en la figura 80.

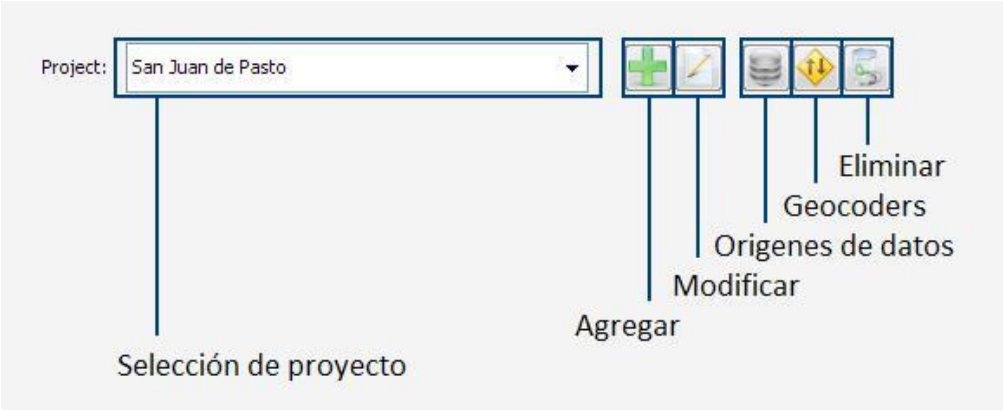

Figura 80. Barra de proyectos

La sección de **herramientas de capas** lista las capas y leyendas del proyecto actual y permite realizar diferentes operaciones sobre ellas, como se muestra en la figura 81.

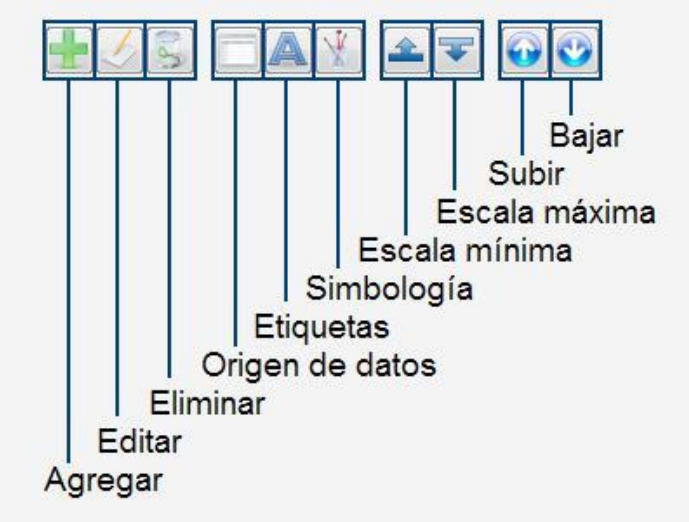

Figura 81. Herramientas de capas

La sección de **visualización**, asume el lugar de un cliente que se contecta al servidor Atlas y permte observar los efectos que tienen las diferentes operaciones realizadas desde el panel de control, sus opciones se muestran en la figura 82.

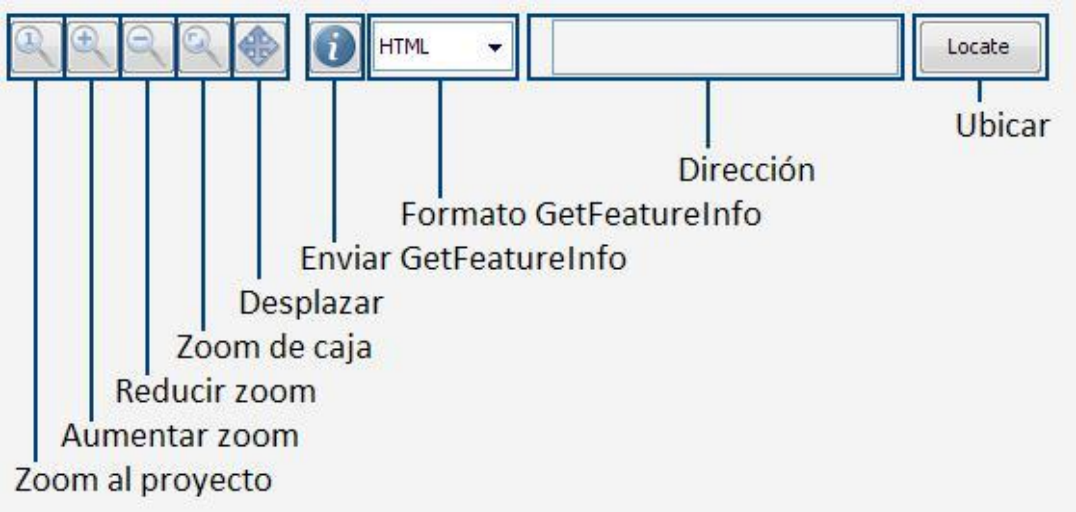

Figura 82. Sección de visualización

3.2.1 Configurar las opciones de la herramienta.

Para cambiar las opciones generales de la herramienta se debe hacer clic sobre el botón **“Options”** de la barra de funciones, entonces, se presenta un formulario como el que se muestra en la figura 83.

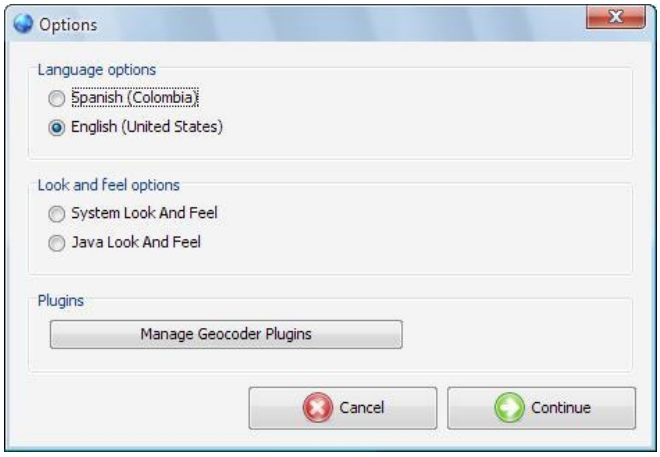

Figura 83. Formulario, opciones de la herramienta

3.2.2 Administrar plugins de geocoders.

Para administrar los plugins de geocodificación se debe hacer clic sobre el botón **“Manage Geocoder Plugins”**, entonces se despliega un formulario como se muestra en la figura 84. Al hacer clic sobre el botón agregar, se muestra el cuadro de diálogo abrir archivo, mostrando solo archivos .JAR, al seleccionar un archivo, este se agrega a la lista de plugins, para remover un plugin, hay que seleccionarlo de la lista y presionar el botón “remover”. El modificar la lista de plugins requiere reiniciar la herramienta.

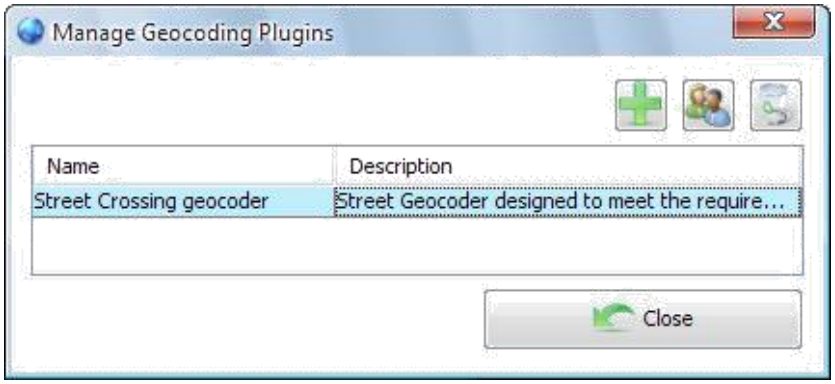

Figura 84. Formulario, administración de plugins

3.2.3 Establecer conexión con un servidor.

El primer paso para utilizar el panel de control es conectarse a un servidor ya establecido, para esto, debe hacerse clic en el botón **"Connect"**, ubicado en la esquina superior izquierda de la barra de funciones, que se muestra en la figura 85.

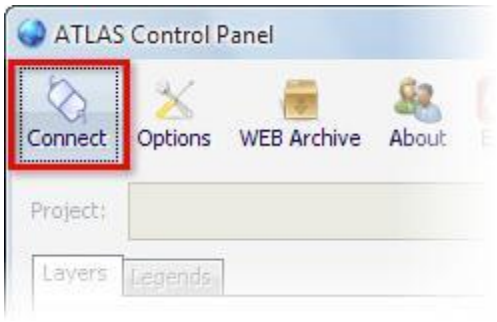

Figura 85. Botón connect en el panel de control

Luego de hacer clic sobre el botón, se muestra el formulario de conexión. Este formulario, permite almacenar los datos de conexión de varios servidores, de modo que solo hace falta escribirlos la primera vez que se desea establecer dicha conexión, apariencia de este formulario se muestra en la figura 86.

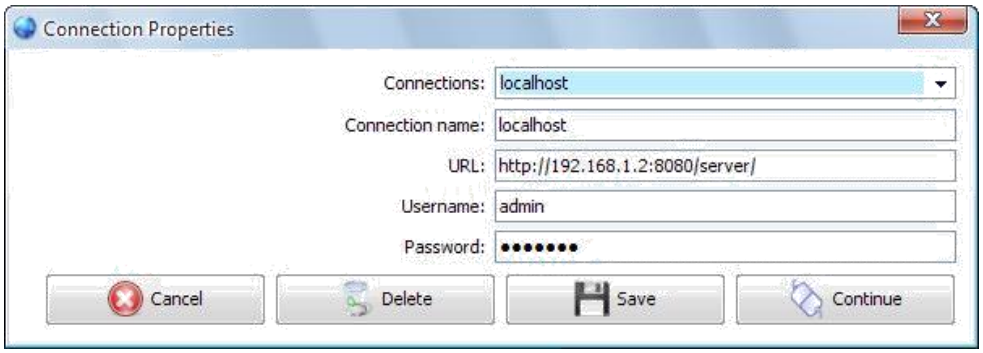

Figura 86. Propiedades de la conexión en el panel de control

En este formulario, el cuadro de selección **“Connections”**, muestra datos de conexión previamente almacenados, para eliminar estos datos, se debe hacer clic en **“Delete”**, para intentar conexión con los datos tal como están, se debe hacer clic en **“Continue”**, la opción **“[New Connection]”** que se encuentra en este cuadro permite crear una nueva conexión.

El campo **“URL”**, corresponde a la dirección del servidor Atlas con el que se desea establecer conexión.

Los campos **“Username”** y **“Password”**, corresponden los datos de autenticación del servidor Atlas, que fueron establecidos durante el proceso de instalación como indica la guía de instalación del servidor Atlas.

Todos los campos en este formulario son obligatorios, cuando están completos, debe hacerse clic en el botón **“Continue”**.

Esto conduce al formulario que valida los datos suministrados (ver figura 87), y en caso de encontrarlos correctos, adelanta diferentes pruebas sobre el servidor que permiten corregir automáticamente diferentes problemas, y ofrecen ayuda sobre como corregir otros.

Entre otras tareas, este formulario verifica la intergridad de la base de datos del sistema, y en caso de encontrar dificultades preguntará si se desea instalarlas.

El botón **“test”** permite repetir las pruebas, y el botón **“Cancel”,** regresa al formulario de datos de la conexión, si todas las pruebas concluyen de forma satisfactoria, el boton **“continue”**, estará habilitado, el permite cerrar el diálogo para continuar a la herramienta.

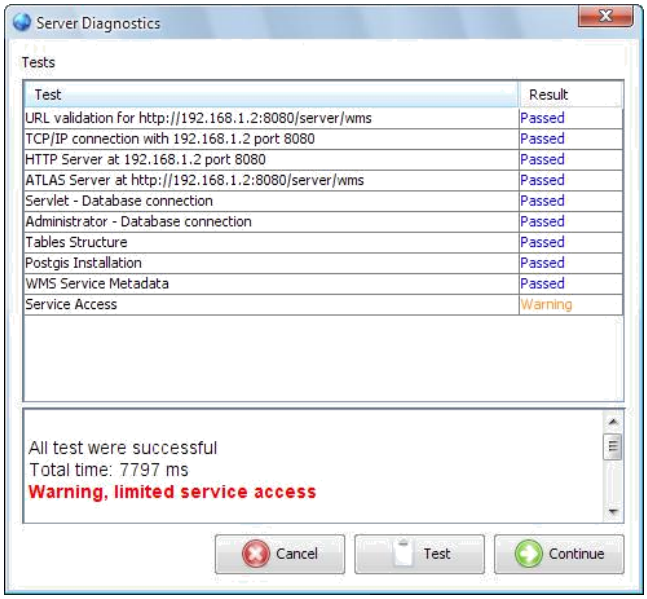

Figura 87. Diagnósticos del servidor

El formulario también verifica los datos del servicio WMS, en caso de encontrarlos incorrectos, muestra el formulario para ingresarlos, como se ve en la figura 88.

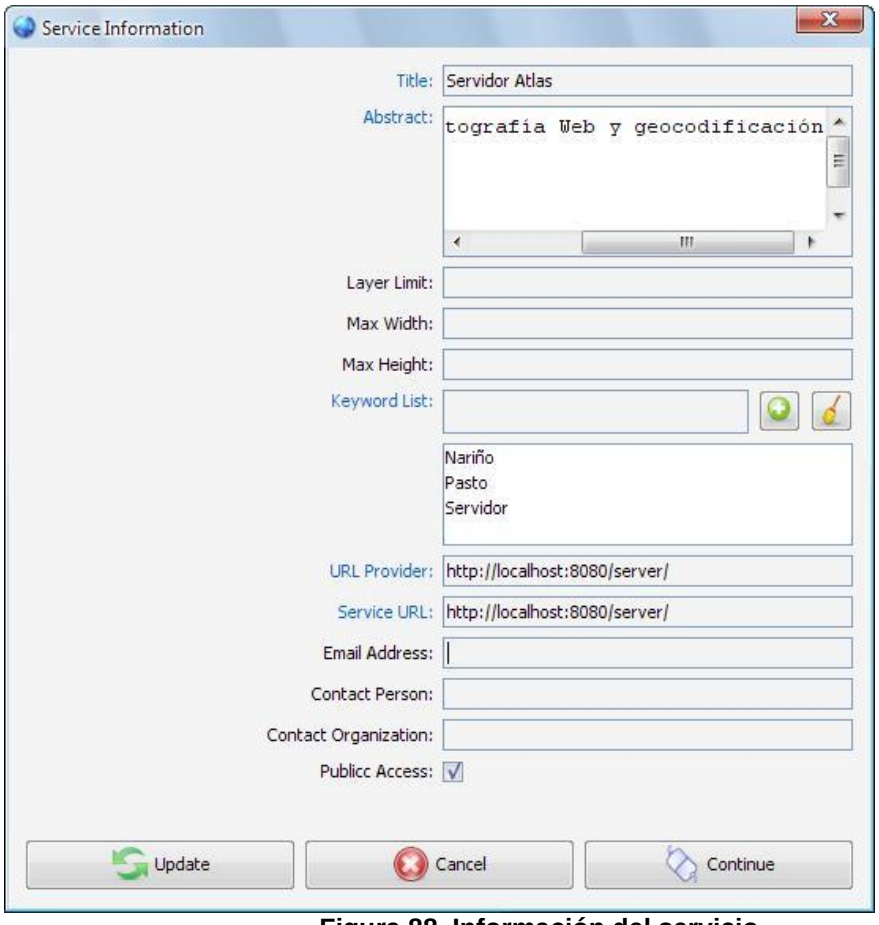

Figura 88. Información del servicio

En este formulario, el campo **“Title”** corresponde al título del servicio, es decir, una descripción breve para mostrar.

El campo **“Abstract”** corresponde al texto breve que sirve como resumen del servicio.

El campo **“Layer Limit”** corresponde al número máximo de capas que se puede solicitar en una petición GetMap, dejar este campo en blanco indica que no hay límite.

El campo **“Max Width”** corresponde al ancho máximo de un mapa en una petición GetMap, dejar este campo en blanco, indica que no hay límite.

El campo **“Max Height”** corresponde al alto máximo de un mapa en una petición GetMap, dejar este campo en blanco, indica que no hay límite.

El campo **“Keyword List”** corresponde a un listado de palabras clave para describir el servicio WMS.

El campo **“URL Provider”** corresponde a una URL con información adicional sobre el servicio.

El campo **“Service URL”** corresponde a la dirección URL del servicio WMS.

El campo **“Email Address”** corresponde a la dirección de correo electrónico del administrador del servicio.

El campo **“Contact Person”** corresponde al nombre del administrador del servicio.

El campo **“Contact Organization”** corresponde al nombre de la organización que presta el servicio.

El campo **“Public Access”** indica si el público general puede o no realizar peticiones al servidor, está opción es útil en caso de que se esté realizando alguna labor de mantenimiento sobre el servidor. Este diálogo también se puede acceder desde el botón WMS Service, como se muestra en la figura 89.

Solo los campos cuya etiqueta se muestra de color azul, son obligatorios, los demás son opcionales.

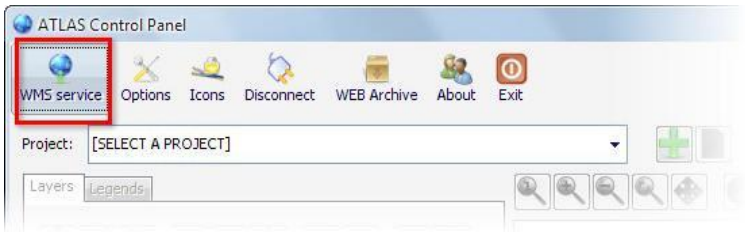

Figura 89. Botón WMS Service

3.2.4 Administrar proyectos.

Para trabajar con proyectos deben usarse los botones ubicados en el panel de proyectos.

Al presionar el botón **"Agregar"**, se presenta el formulario de datos del proyecto, como se muestra en el formulario de la figura 90.

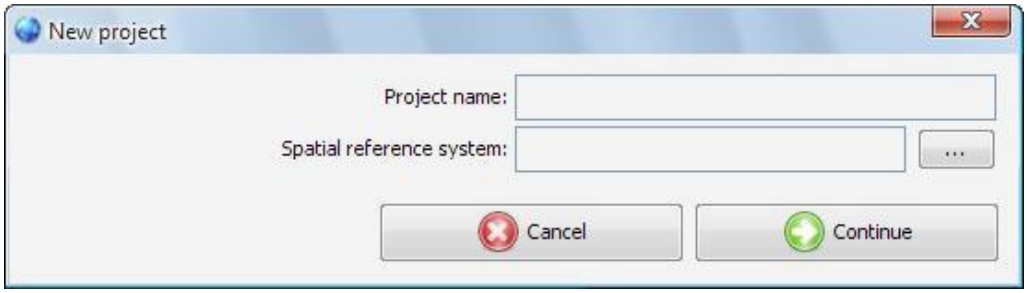

Figura 90. Formulario de nuevo proyecto

En este formulario, **"Project Name"** corresponde al nombre del proyecto y **"Spatial Reference System"** al sistema espacial de referencia, el botón de **"Cancel"**, abandona el proceso sin realizan cambios, y el botón **"Continue"**, verifica la información proporcionada y guarda el proyecto. Los dos campos son obligatorios, para seleccionar un sistema espacial de referencia debe hacerse clic sobre el botón **"…".**

La información que se proporciona en este formulario puede ser modificada luego, para ello, debe usarse el cuadro de selección **"selecció n de proyecto"** y luego presionar el botón **"editar"**, de igual modo, para remover el proyecto debe usarse el botón **"eliminar"**.

Para seleccionar el sistema espacial de referencia, se despliega un formulario como se muestra en la figura 91.

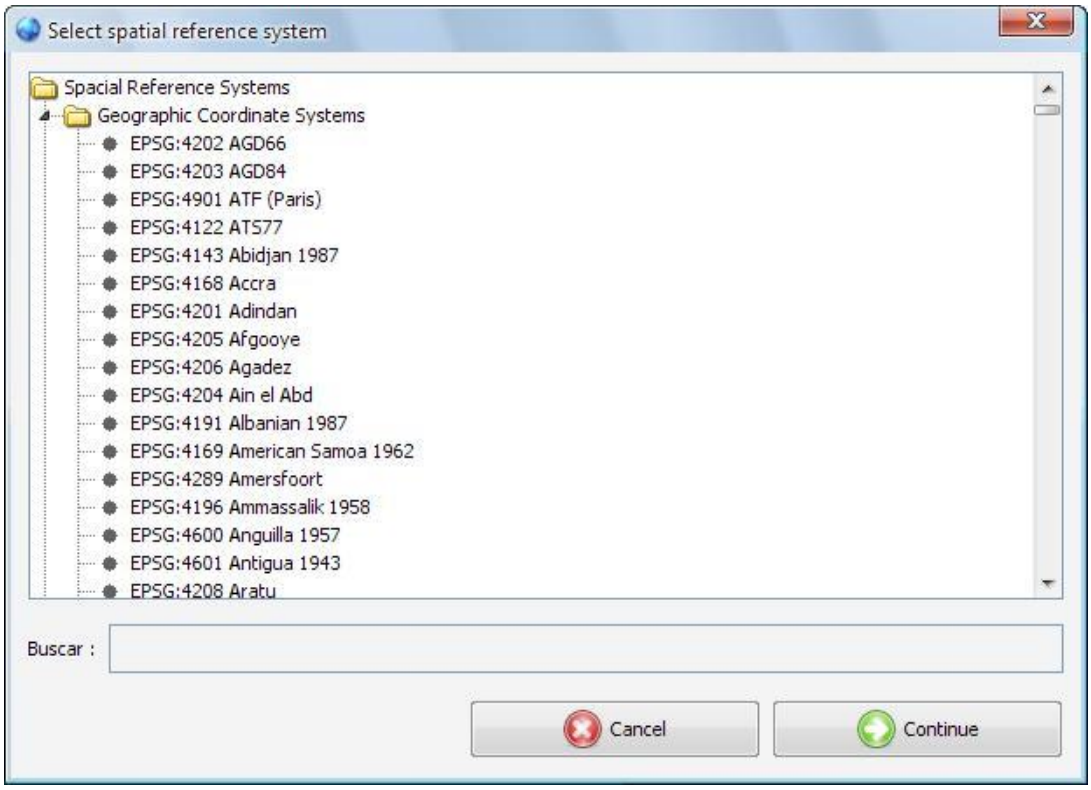

Figura 91. Formulario selección de sistema de referencia espacial

En el formulario de selección de sistema de referencia espacial, el cuadro **"Buscar"**, permite filtrar los sistemas de referencia por su nombre a medida que se escribe.

También es posible navegar por el árbol hasta encontrar el sistema deseado, una vez el sistema está seleccionado, debe hacerse clic en **"continue"**¸ el hacer clic en**"Cancel",** hace que se cierre el formulario sin realizar ningún cambio.

3.2.5 Administrar orígenes de datos al proyecto.

Los orígenes de datos de un proyecto se utilizan como fuente de información para las capas y sirven también como material de referencia para geocodificación, para administrar los orígenes de datos de un proyecto se debe seleccionar un proyecto y hacer clic en el botón que muestra la figura 92.

Figura 92. Botón administrar orígenes de datos

Al presionar este botón, debe aparecer el formulario, administrador de orígenes (ver figura 93), que presenta una lista de los orígenes presentes en el proyecto, indicando por cada uno, la tabla donde se encuentra, el nombre del campo geométrico de la tabla, el tipo de la geometría y el estado del origen, en función de si ha sido o no usado por alguna capa del proyecto.

Para remover una capa del proyecto, se debe seleccionar la capa de la lista y hacer clic en el botón eliminar.

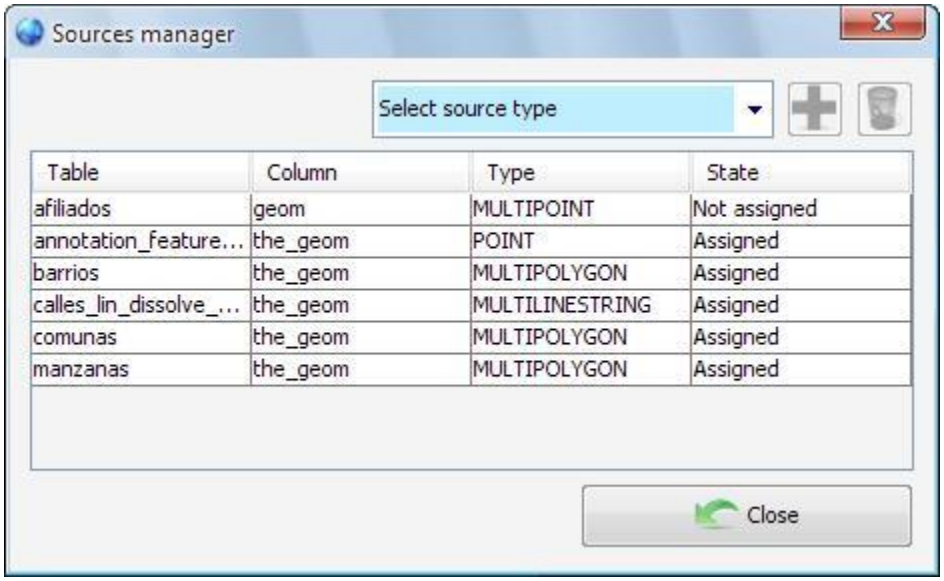

Figura 93. Formulario, administrador de orígenes

El cuadro de selección **"Select source type"** permite elegir el tipo del origen de datos que se desea agregar, una vez realizada la selección se debe hacer clic en el botón agregar.

Para agregar un origen de datos desde un ShapeFile de ESRI, la selección debe estár como indica la figura 94.

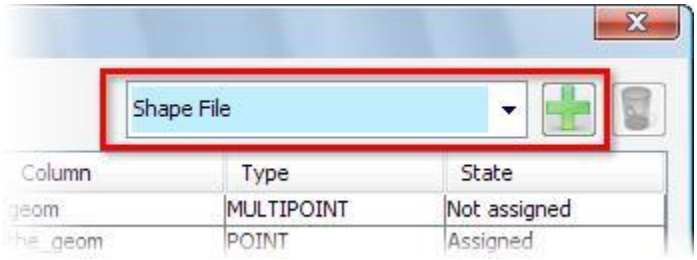

Figura 94. Agregar un origen de datos ShapeFile

Al hacer clic sobre el botón **"agregar"** se presenta el cuadro de diálogo de abrir archivo, mostrando únicamente archivos ShapeFile (.shp), una vez seleccionado el archivo, se mostrará un formulario como el de la figura 95.

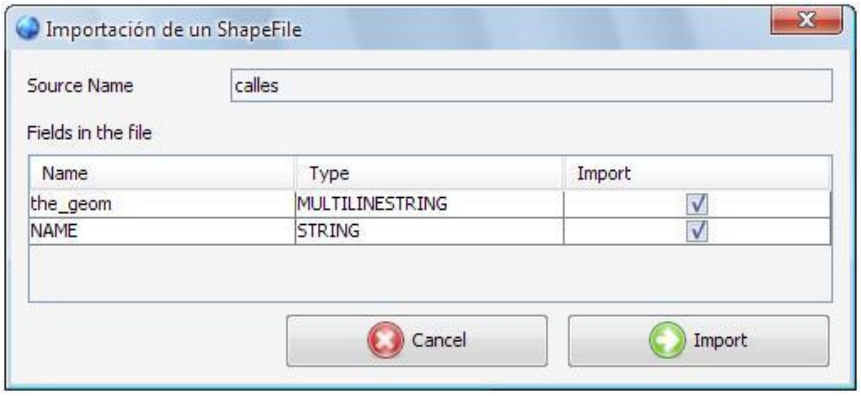

Figura 95. Formulario, importación de un ShapeFile

En este formulario, **"Source Name"**, indica el nombre que tendrá el origen en el proyecto, la tabla **"Fields in the file"**, muestra los campos hallados en el archivo, se puede seleccionar cuáles de ellos se importarán modificando las casillas de selección de la columna **"import"**.

Luego de establecer los campos que se van a importar y el nombre del origen, se debe hacer clic sobre el botón **"Import"**, al hacer clic sobre el botón **"Cancel"** se cierra el formulario sin hacer cambios.

Para agregar un origen de datos desde Postgis, la selección debe estár como indica la figura 96, además, postgis debe estár instalado en la base de datos del sistema.

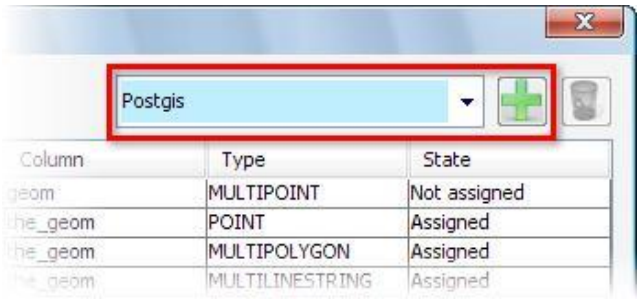

Figura 96. Agregar un origen de datos Postgis

Al hacer clic sobre el botón **"agregar"** se mostrará un formulario como el de la figura 97.

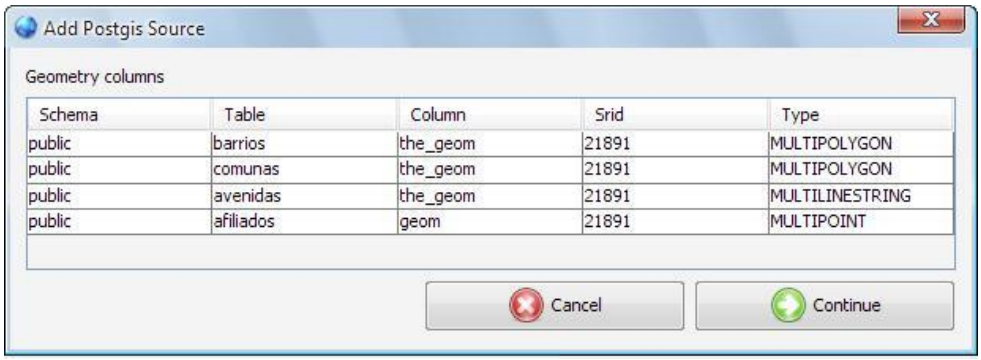

Figura 97. Formulario, importación Postgis

En este formulario, se presenta un listado de las tablas del sistema que son compatibles con Postgis, se debe hacer clic sobre el botón **"Continue"** para agregar la fuente, al hacer clic sobre el botón **"Cancel"** se cierra el formulario sin hacer cambios.

3.2.6 Administrar geocoders del proyecto.

Un geocoder permite que el proyecto tenga la capacidad de procesar direcciones, para ello, se requiere agregar y configurar dichos geocoders.

Para iniciar el administrador de geocoders, se debe hacer clic en el botón geocoders de la barra de proyectos.

Luego de hacer clic sobre el botón, debe aparecer un formulario como el que se muestra en la figura 98.

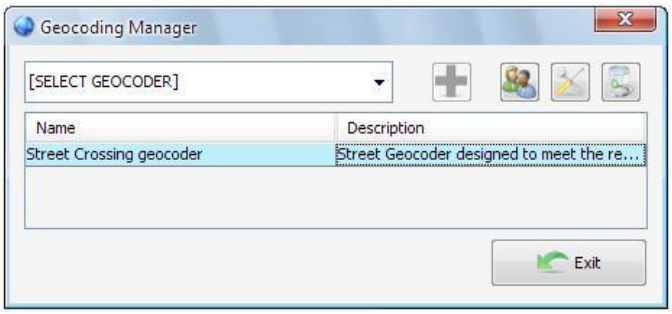

Figura 98. Administrador de geocoders

En este formulario, el cuadro de selección **"[SELECT GEOCODER]"**, permite seleccionar el geocoder que se va agregar, luego se debe hacer clic sobre el botón **"agregar"**, ahora, el plugin ya está agregado y forma parte del proyecto, sin embargo, aun no está configurado, para esto, se debe seleccionar el geocoder de la lista, y hacer clic sobre el botón **"Configurar"**, como se muestra en la figura 99. Las opciones a configurar de cada geocoder dependerán del proveedor del plugin.

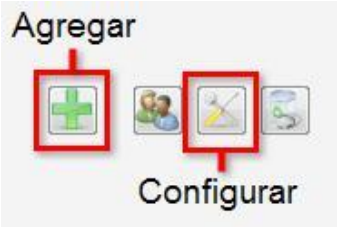

Figura 99. Botones en la administración de geocoders

3.2.7 Administrar capas.

Para trabar con capas, primero se debe seleccionar un proyecto, eligiéndolo del cuadro de selección que se muestra en la figura 100.

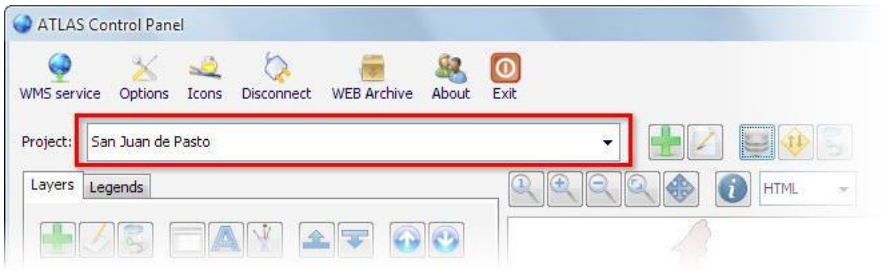

Figura 100. Cuadro de selección de proyectos

Una vez seleccionado el proyecto, en la sección de herramientas para capas se habilitarán varios botones

Al hacer clic sobre el botón **"agregar"**, se muestra un formulario (ver figura 101) que permite consignar los datos para la nueva capa, esta información se puede modificar luego al presionar el botón **"Editar"**, la capa se remueve presionando el botón **"Eliminar".**

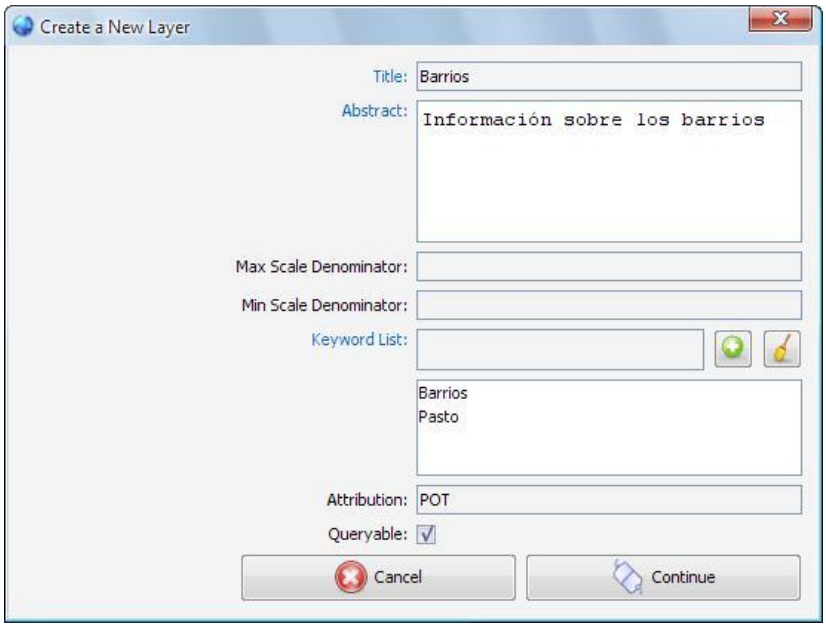

Figura 101. Formulario, nueva capa

En el formulario, nueva capa, el campo **“Title”** corresponde al título de la capa, es decir, una descripción breve para mostrar.

El campo **“Abstract”** corresponde al texto breve que sirve como resumen del servicio.

El campo **“Max Scale Denominator”** corresponde máximo denominador de escala al que es visible esta capa, dejar este campo en blanco indica que no hay límite.

El campo **“Min Scale Denominator”** corresponde mínimo denominador de escala al que es visible esta capa, dejar este campo en blanco indica que no hay límite.

El campo **“Keyword List”** corresponde a un listado de palabras clave para describir la capa.

El campo **“Attribution”** corresponde a nombre de la fuente de los datos que se presentan en la capa.

El campo **“Queryable”** indica si la capa puede o no incluirse en peticiones GetFeatureInfo.

Solo los campos cuya etiqueta se muestra de color azul, son obligatorios, los demás son opcionales.

3.2.8 Capas y orígenes de datos

Una vez se ha agregado una capa al un proyecto, aun es necesario indicar el origen de datos del que se tomará la geometría, para ello, debe seleccionarse la capa desde la lista que se muestra en la figura 102 y presionar el botón que se muestra en la figura 103.

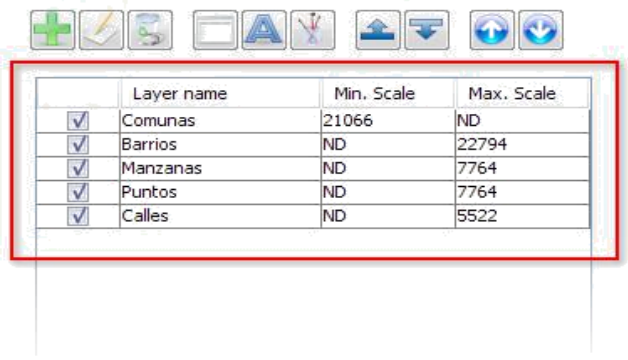

Figura 102. Lista de capas

Figura 103. Botón, configurar orígenes de datos

Al presionar este botón, se presenta un formulario como se ve en la figura 104.

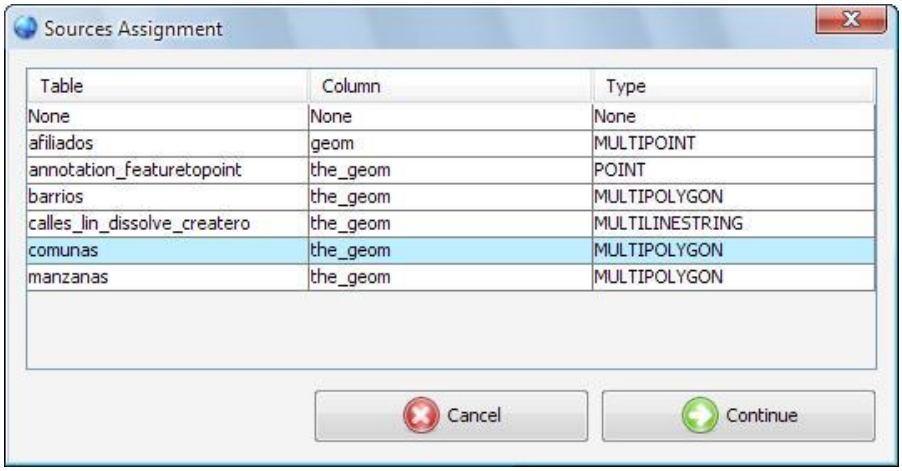

Figura 104. Formulario, asignación de orígenes

El formulario, asignación de orígenes, muestra un listado de los orígenes de datos que contiene el proyecto, para asignar uno de estos orígenes a la capa, basta con seleccionarlo de la lista y hacer clic en **"Continue"**, presionar **"Cancel"**, ocaciona que se cierre el formulario sin hacer cambios.

Al seleccionar la primera fuente de la lista, **"none"**, indica que la capa no tiene origen de datos, y por lo tanto, no será dibujada.

3.2.9 Administración de etiquetado.

El etiquetado, permite que junto con la geometría de una capa se dibujen letreros que describen lo que se está presentando. Para modificar las opciones de etiquetado de una capa debe hacerse clic sobre el botón que se muestra en la figura 105.

Figura 105. Botón, administrar etiquetado

Al presionar este botón, debe aparecer un formulario como el que se muestra en la figura 106.

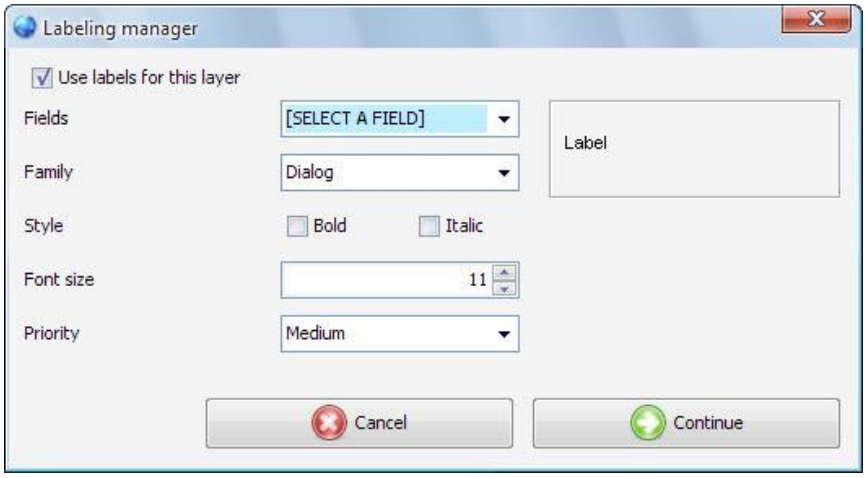

Figura 106. Formulario, administrar etiquetas

En este formulario, la opción **"use labels for this layer"**, indica si esta capa usará o no etiquetas, el cuadro de selección **"Fields"**, muestra los campos que contiene el origen de datos de la capa, debe seleccionarse el campo cuyos valores se mostrarán, el cuadro de selección **"Family"** presenta las diferentes fuentes con las que se puede dibujar las etiquetas, **"Style"**, presenta las opciones **"Bold"**, para dibujar las etiquetas en negrita, e **"Italic"**, que permite dibujarlas cursivas. **"Priority"**, indica la acción que debe tomar el servidor al encontrar dos etiquetas que colisionan, las etiquetas de prioridad más baja,son removidas, mientras que entre dos etiquetas de la misma prioridad, sobrevive la ultima en dibujarse.

3.2.10 Simbología

La simbología determina la forma en que presentarán las geometrías, para editar la simbología de una capa, hay que seleccionar una de la lista y hacer clic en el botón que muestra la figura 107.

Figura 107. Botón editar simbología

El formulario que aparece, presenta tres alternativas para la edición de simbologías de capas, **"Fixed"**, **"Range"** y **"Class"**, que se muestran en la figura 108.

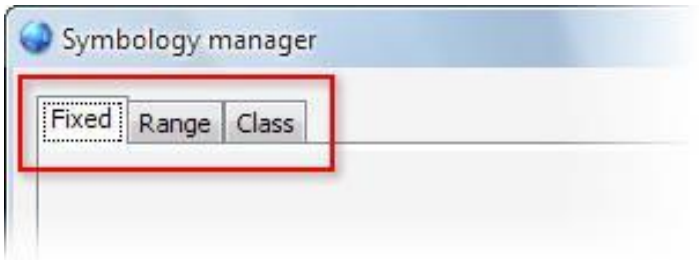

Figura 108. Alternativas de simbología

La simbología **"Fixed"**, presenta todos los objetos de la capa de la misma forma, basta con hacer clic en el botón **"Generate"** y editar el estilo según el tipo de la geometría de capa, los estilos se tratan más adelante.

La simbología **"Range"** (ver figura 109), requiere que el origen de datos de la capa tenga un campo numérico, entonces, forma un cierto número de intervalos, y cada objeto se dibuja de acuerdo al intervalo que le corresponda. La apariencia del generador de rangos se muestra en la figura.

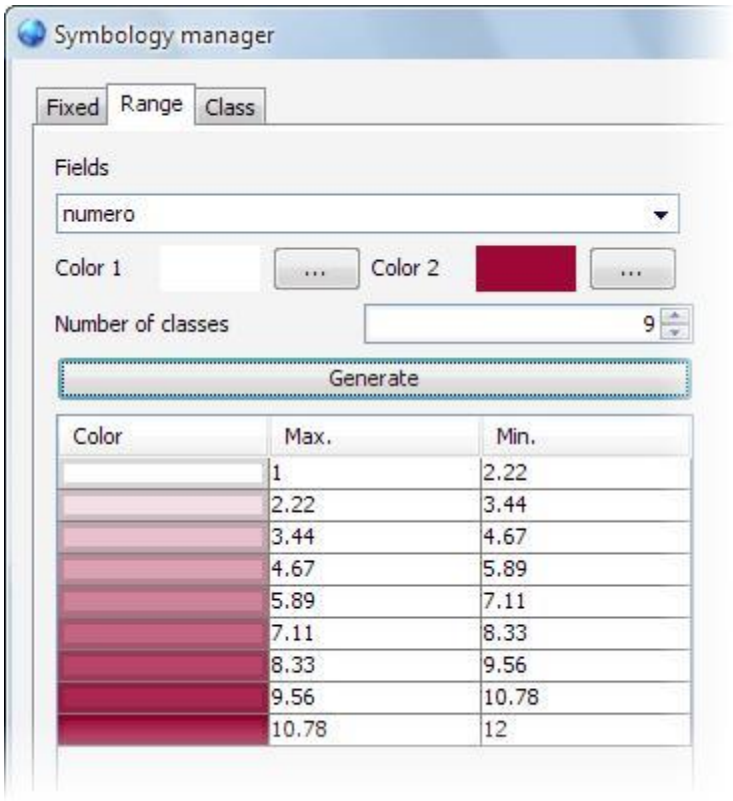

Figura 109. Simbología por rangos

En la pestaña **"Range"**, el cuadro de selección **"Fields"**, indica los campos numéricos que tiene el origen de datos de la capa, al formar los rangos, los colores de los rangos varían desde **"Color 1"** hasta **"Color 2"**, el campo **"Number of classes"**, indica cuantos intervalos se van a crear, el sistema sugiere un número con base en la cantidad de datos.

Una vez establecidos estos datos, basta con hacer clic sobre el botón **“Generate”**. Los intervalos generados aparecerán en la tabla, al seleccionar alguno de los intervalos, se puede editar su estilo, el tema de estilos, se cubre más adelante.

La simbología **“Class”** (ver figura 209), funciona con cualquier campo del origen de datos, basta con seleccionar el campo deseado y presionar el botón **“Generate”**, entonces, en la tabla aparecerán los diferentes valores hallados para este campo. Al seleccionar alguno de los valores, se puede editar su estilo, el tema de estilos, se cubre más adelante. La apariencia del generador de clases se muestra en la figura 110.

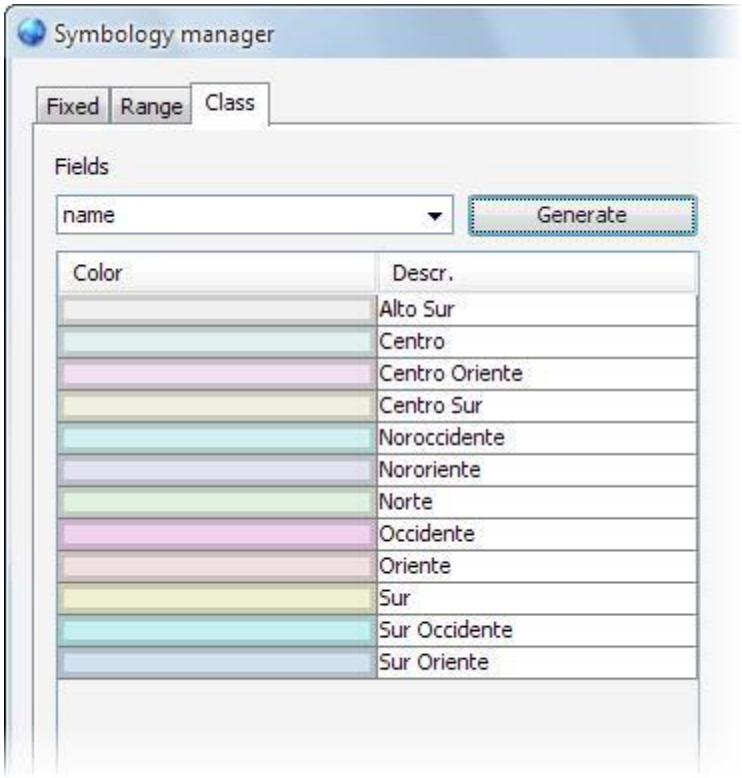

Figura 110. Simbología por clases

3.2.11 Estilos

El estilo depende del tipo de de la geometría de la capa, así, hay diferentes estilos para puntos, líneas y polígonos.

Para una geometría tipo punto (ver figura 111), es posible seleccionar un icono que se colocará en la ubicación del punto, el campo **“Customize”**, indica si los colores del icono deben o no ser reemplazados, en caso de haber elegido personalizar los colores, se puede elegir uno de fondo y uno de relleno, la barra **“Size”**, permite modificar el tamaño del icono.

Figura 111. Estilos para geometrías tipo punto

Para una geometría tipo línea, es posible cambiar su estilo, su color y su grosor, como se muestra en la figura 112.

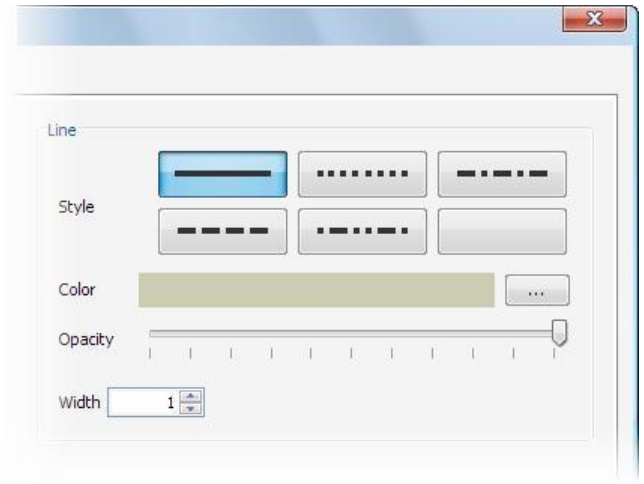

Figura 112. Estilos para geometrías tipo línea

Para una geometría tipo polígono, es posible editar el estilo de la línea de contorno, de la misma manera en que se edita el estilo para una geometría tipo línea, pero además se puede modificar el color de relleno, como se muestra en la figura 113.

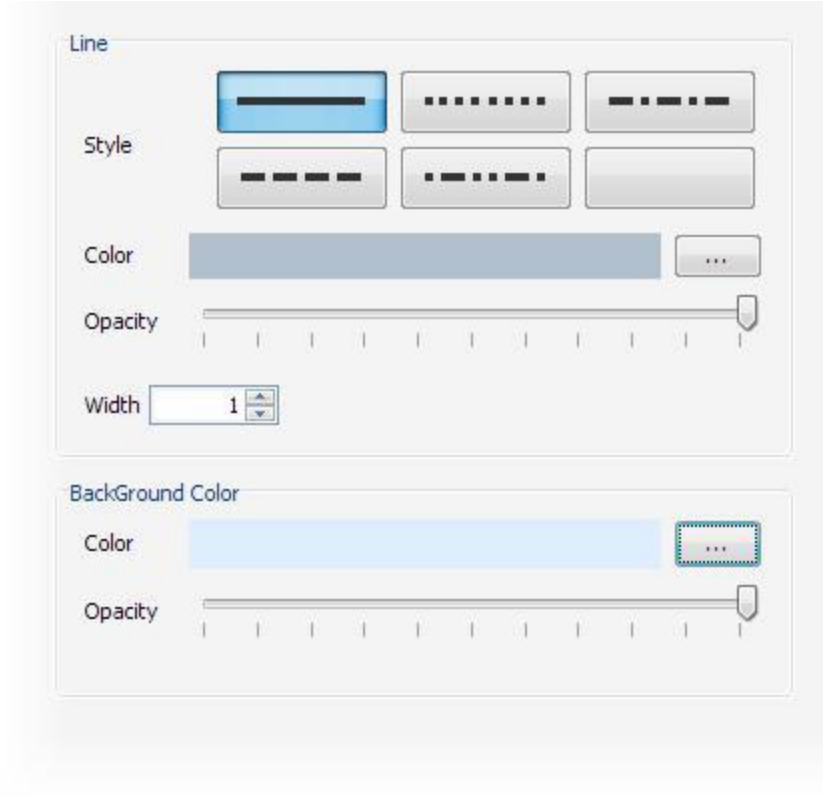

Figura 113. Estilos para geometrías tipo polígono

3.2.12 Cambiar los denominadores de escala.

Para modificar los denominadores de escala de una capa (ver figura 114), se requiere usar la sección de visualización para determinar la escala deseada, y presionar el botón de escala máxima o de escala mínima.

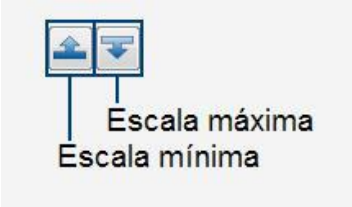

Figura 114. Botones denominadores de escala

Para remover un denominador de escala hay que hacer clic derecho sobre la capa en la lista, y entonces se desplegará el menú que permite ejecutar estas acciones como se muestra en la figura 115.

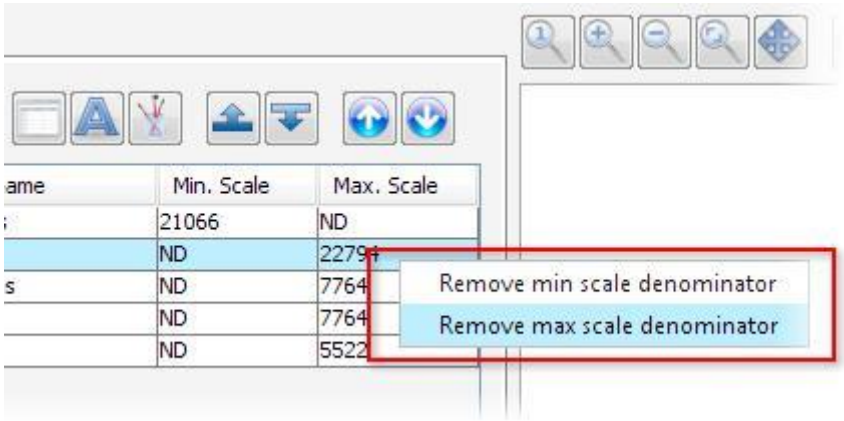

Figura 115. Remover denominadores de escala

3.2.13 Cambiar el orden de las capas.

Las capas de dibujan en el mismo orden en que se presentan en la lista, de modo que la primera capa en la lista es la menos visible, para modificar el orden de las capas, se debe seleccionar la capa que se desea mover y hacer clic en los botones que muestra la figura 116.

Figura 116. Botones de subir y bajar capa

4. UTILITARIOS DEL SERVIDOR ATLAS

4.1 DESARROLLO PLUGINS DE GEOCODIFICACIÓN

Los plugins de geocodificación permiten agregar a Atlas funcionalidad adicional en este campo, sin la necesidad de recompilar toda la aplicación, los plugins se colocan en el servidor Atlas como se describe en la sección de instalación de servidor, y se usan en el panel de control, como se describe en la sección de administración.

Para crear un plugin indistintamente del IDE que se use, deben incluirse las bibliotecas que se muestran en la tabla 61.

Tabla 61. Bibliotecas en la construcción de geocoders.

Biblioteca	**Archivo**
Java Topology Suit	jts-1.8.jar
Hibernate	hibernate.jar
GeoTools	gt2-main-2.3.0.jar gt2-api-2.3.0.jar gt2-referencing-2.3.0.jar
GeoAPI	geoapi-nongenerics-2.1-M2.jar
Javax.units	jsr108-0.01.jar
Vecmath	vecmath-1.3.1.jar
Atlas Common	atlas_common.jar

4.1.1 La clase atlas.geocoding.Geocoder.

Esta clase es la base de todo el sistema de geocoders y plugins en Atlas. Se trata de una clase abstracta, por lo tanto, todos sus métodos deben ser implementados para construir un geocoder. Algunos métodos son usados en el panel de control y otros, en el servidor.

4.1.2 Panel de control.

Cada geocoder es configurado en función de un proyecto, y puede tomar cualquier origen de datos de este como material de referencia.

4.1.3 Métodos.

Cada proyecto cuenta con el método **"getSources"**, que retorna el conjunto de sus orígenes de datos, cada origen de datos, cuenta con los métodos **"getFieldNames"**, que retorna los nombres de los campos el origen, y **"getFieldTypes"**, que retorna los tipos de campos, como constantes enteras de la clase **"java.sql.Types"**, además, cada origen tiene los métodos **"getTablsour"** y **"getFielsour"**, que retornan el nombre de la tabla y el nombre del campo con información geométrica, esta tabla se encuentra en la base de datos del sistema. Los métodos **"getGeomClass"** y **"getGeomType"** retornan información sobre el tipo de geometría del origen.

Un objeto HibernateUtil, provee distintas alternativas para acceder a la base de datos del sistema, mediante el método **"begin"**, entrega una sesión hibernate, que debe ser cerrada con **"close"** o **"commit"**. El objeto cuenta también con el método **"getConnection"**, que retorna directamente un objeto de conexión **"java.sql.Connection"**.

Un objeto GeocoderConfig, guarda información sobre la configuración de un geocoder para un proyecto, al implementar un geocoder los métodos **"getConfiguration"** y **"setConfiguration"** se usan para guardar y recuperar una cadena, que contiene dicha configuración en el formato que el desarrollador del plugin considere conveniente, por ejemplo CSV o XML.

La clase org.atlas.geocoding.ConfigurationDialog, extiende de JDialog y debe ser extendida nuevamente por el desarrollador del plugins, está pensada para permitir a un administrador, configurar un geocoder desde el panel de control, tiene un objeto Project, un objeto GeocoderConfig, y un objeto HibernateData. Un posible flujo de trabajo de este formulario incluye.

- Leer la configuración actual del geocoder (Opcional).

- Presentar las opciones de configuración, tomando como referencia los orígenes de datos del proyecto.

- Modificar los orígenes de datos o la base de datos a fin de construir datos pre-calculados (Opcional).

- Modificar el campo de configuración del objeto GeocoderConfig.

- Usar el método setCancelled, para indicar que el usuario no canceló la operación.

El geocoder debe entregar un cuadro de diálogo de este tipo cuando se llama al método **"getConfigurationDialog"**, este método tiene como parámetros un objeto Project, un objeto GeocoderConfig, y un objeto HibernateData.

El método **"getName"**, debe retornar el nombre del geocoder, una descripción breve.

El método **"getDescription"**, debe retornar una descripción más extensa del geocoder.

El método **"getAboutDialog"** debe retornar un cuadro de diálogo con información sobre el geocoder, este diálogo, debe ser una subclase de JDialog.

Servidor. Otros métodos de la clase son usados desde el servidor, a fin de preparar al geocoder para resolver direcciones y luego durante el proceso de resolución en sí mismo.

El método **"isCapableFor"**, recibe por parámetro una cadena que contiene la dirección tal como llega en la petición. El método debe retornar un booleano, indicando si el geocoder está o no en capacidad de procesar la dirección, se trata de una revisión rápida, por ejemplo, un geocoder que trabaje con nomenclatura de calles y carreras no podrá trabajar con direcciones que no contengan alguna de estas palabras o sus sinónimos. Sin embargo, es posible, que a pesar de que este método arroje positivo para una dirección, más adelante no entregue resultados, ya que solo es una revisión inicial y rápida.

El método **"getStandardizedAddress"** recibe por parámetro una cadena que contiene la dirección tal como llega en la petición. Este método debe realizar todas las transformaciones sobre la dirección que se consideren necesarias antes de dividir la cadena en componentes. Por ejemplo, en el caso del geocoder para nomenclatura de calles y carreras, en este paso deben reemplazarse todas las ocurrencias de "calle" , "cl" , "cle", etc por "CL".

El método **"parseAdress"** recibe por parámetro la cadena de una dirección que ha sido entregada previamente al método **"getStandardizedAddress"**, y debe retornar las partes que integran la dirección. Por ejemplo, en el caso del geocoder para nomenclatura de calles y carreras, retorna el tipo de la dirección que puede ser "CL" o "CR", el nombre de la vía principal, el nombre de la intersección y el número de la puerta.

El método **“locateAddress”** recibe por parámetro el array de cadenas que representa las partes de la dirección, y debe entregar las posibles ubicaciones de esta, representando cada una con un objeto **“Result”**.

El método “**getConfigurationDialog”** debe retornar un diálogo de configuración listo para funcionar con los objetos que se entregan como parámetros, que son, el proyecto, el objeto de configuración del geocoder y un objeto de conexión hibernate.

El método “**init”** es llamado por el servidor antes de empezar a hacer peticiones, aquí el geocoder debe realizar todas las acciones que sean necesarias antes de empezar a operar, por ejemplo, formar caché y verificar el estado de las tablas requeridas. El objeto tiene por parámetro un objeto tipo ConnectionProperties, que puede ser transformado directamente en una conexión a la base de datos usando el método **“createConnection”** de la clase DataBaseUtils.

Finalmente la clase que extiende a Geocoder, debe aparecer como clase principal en el manifiesto de archivo JAR que contenga las demás clases.

4.2 DESARROLLO DE APLICACIONES JAVA ME.

En esta sección se cubre la realización de una sencilla aplicación para dispositivos móviles que demuestra los conceptos básicos sobre la creación de aplicaciones con la biblioteca móvil de Atlas, esta guía usará el IDE NetBeans 6.0.

4.2.1 Configuración de entorno.

Primero, debe crearse un nuevo proyecto en NetBeans, para ello debe hacerse clic sobre el botón mostrado en la figura 117.

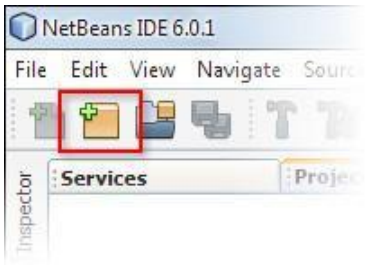

Figura 117. Botón nuevo proyecto en NetBeans

Luego, en la ventana que aparece deben seleccionase la categoría Mobility y a continuación seleccionar de la sección de proyectos “MIDP Aplication” como se muestra en la figura 118, finalmente debe hacer clic en “next”.

Nota. Si la categoría Mobility no se lista, vaya al menú Tools, Pluings y actualice la versión de NetBeans, agregando el paquete Mobility.

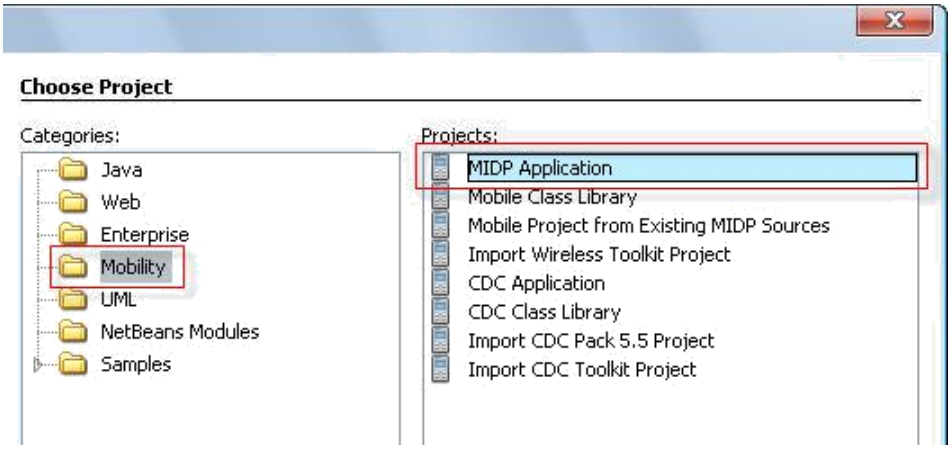

Figura 118. Nuevo proyecto java ME en NetBeans

En el diálogo de la figura 119 se indica el nombre del proyecto y se indica a NetBeans que no debe crear una clase principal, luego debe hacerse click en next.

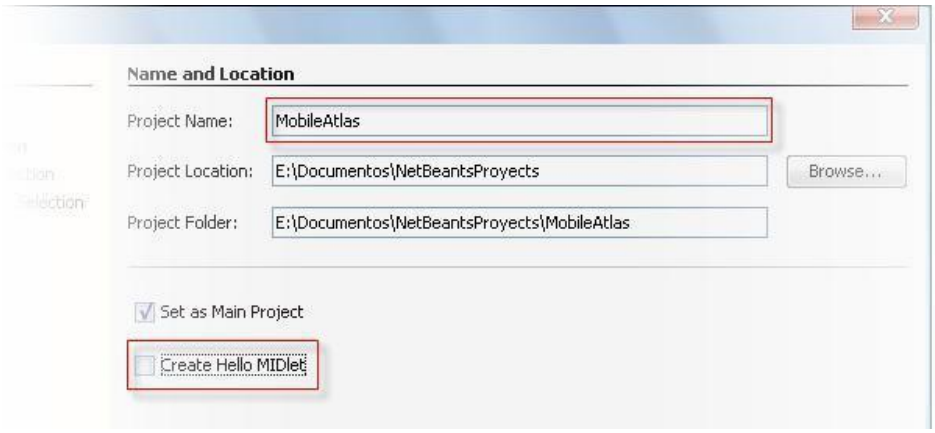

Figura 119. Opciones para una nueva aplicación java en NetBeans

El diálogo de la figura 120 indica el tipo de emulador, dipositvo, configuracción del dipositivo y perfil del dispositivo, que se deben seleccionar como se muestra en la figura 120, luego debe hacerce clic en next.

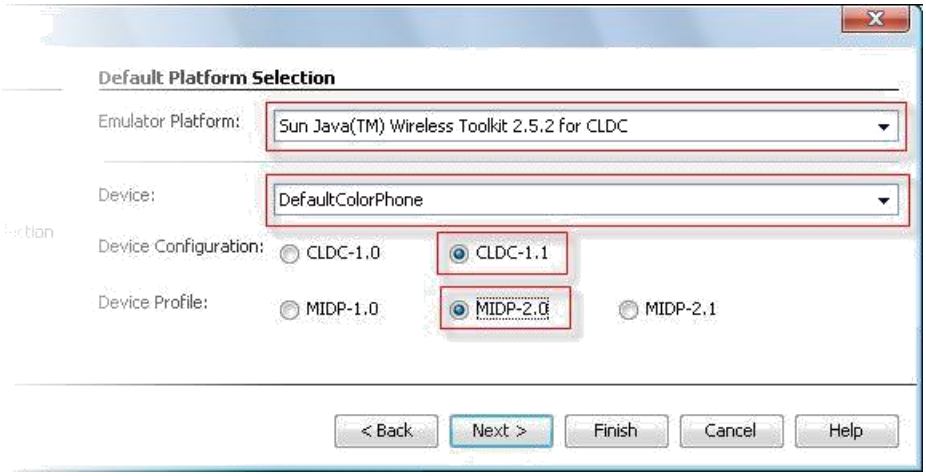

Figura 120. Selección de plataforma por defecto

El diálogo siguiente agrega más configuraciones al proyecto, para este ejemplo se usan las opciones por defecto, como se muestra en la figura 121, finalmente debe hacer clic en el botón finalizar.

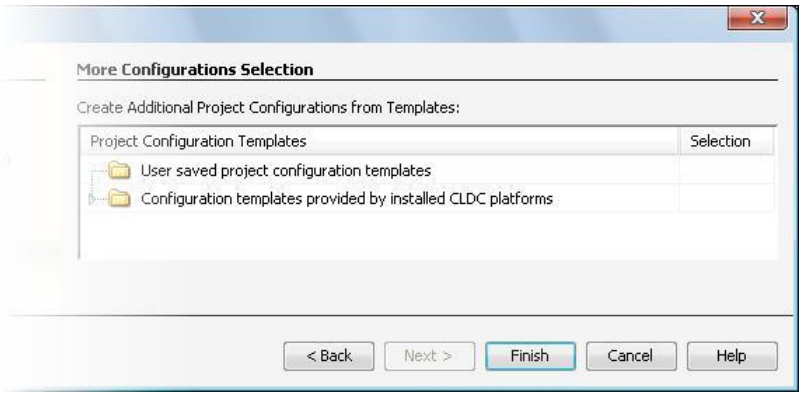

Figura 121. Configuraciones adicionales

El proyecto aparece ahora en el árbol de proyectos como se muestra en la figura 122.

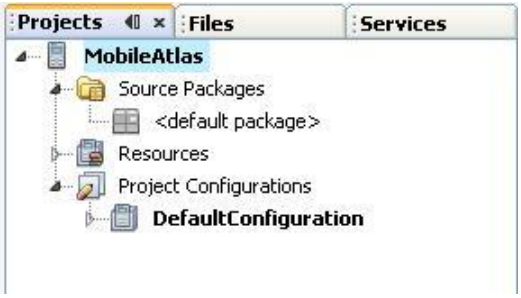

Figura 122. Árbol de proyecto en NetBeans

Luego se deben agregar los recursos requeridos. Para ello debe desplegarse el menú que se muestra en la figura 123.

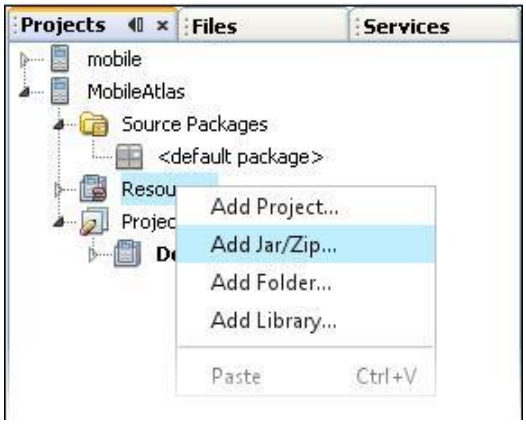

Figura 123. Menu agregar recurso en NetBeans

Luego, debe agregarse el archivo *atlas_mobile.jar*. El árbol de proyecto debe verse como se muestra en la figura 124.

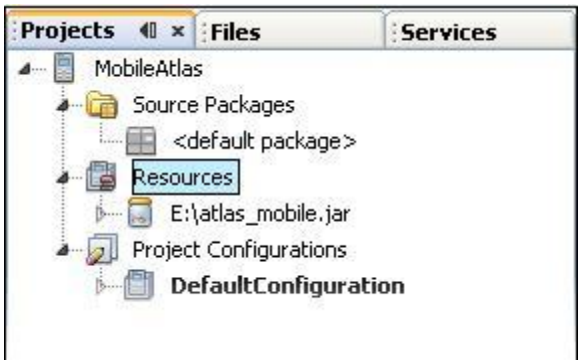

Figura 124. Resource en un proyecto NetBeans

Ahora, se debe crear un “Visual MIDlet” en la aplicación, usando el menú que se muestra en la figura 125.

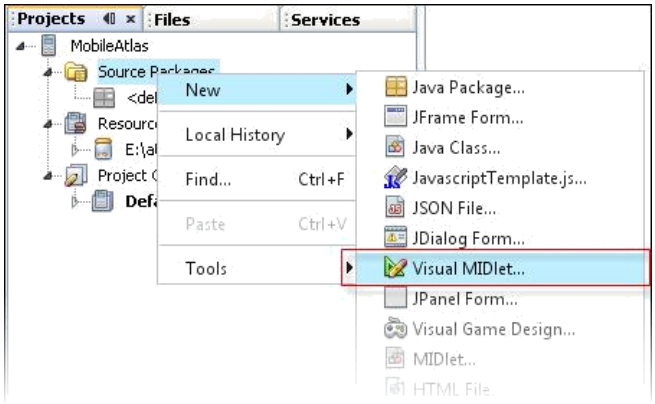

Figura 125. Creación de un Visual MIDlet en NetBeans

En la ventana que aparece, se debe establecer el nombre del MIDlet para este ejemplo denominado “Atlas”, como se muestra en la figura 126 y luego se debe hacer clic en finish.

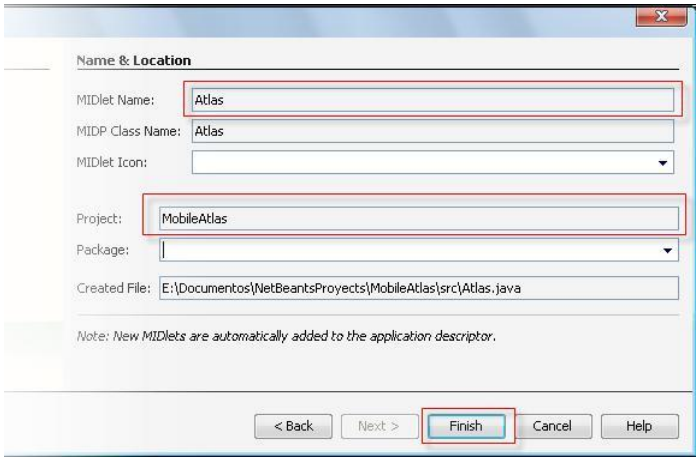

Figura 126. Propiedades del visual MIDlet

El árbol del proyecto debe verse ahora como indica la figura 127.

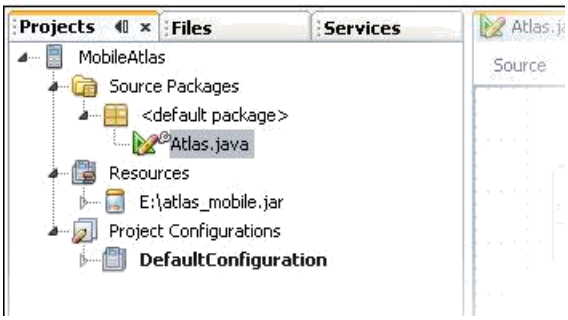

Figura 127. Árbol de proyecto en NetBeans con un MIDlet nuevo

Al hacer doble clic sobre el nodo Atlas.java, este se mostrará en el recuadro central, debe hacerse clic sobre el botón “Source” en caso de que este no esté marcado (ver figura 128).

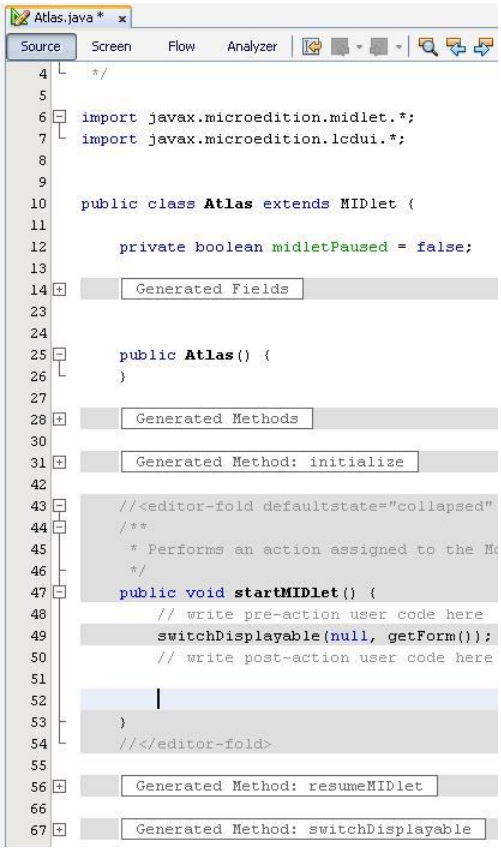

Figura 128. Vista Source en NetBeans

Para estén ejemplo el segmento de *imports* debe ser igual al que se muestra en la figura 129.

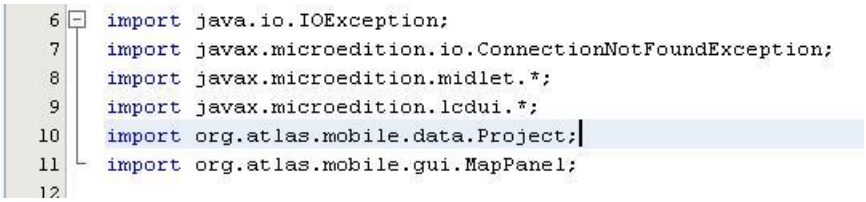

Figura 129. Imports biblioteca Atlas para apliciones Java ME

Para el manejo de comandos la clase Atlas debe implementar CommandListener como se muestra en la figura 130.

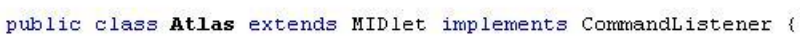

Figura 130. Implementación de CommandListener

Ahora se debe agregar los siguientes atributos a la clase, como se muestra en la figura 131.

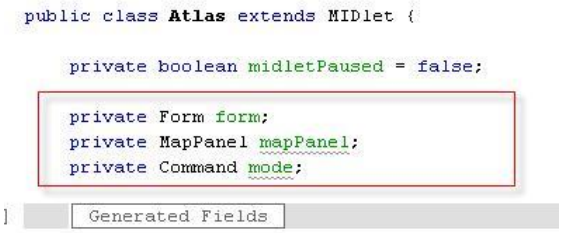

Figura 131. Atributos de la clase MIDlet

El constructor de MIDlet Atlas debe modificase como se muestra en la figura 132.

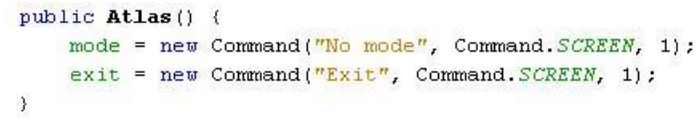

Figura 132. Constructor de la clase Atlas

Al realizar correctamente los anteriores pasos, se ha establecido un proyecto base para continuar con el manejo del MapPanel.

4.2.2 Conectarse con un servidor.

Para esta y las siguientes secciones se asume que existe un servidor Atlas instalado y funcionando en la dirección http://192.168.10.111:8084/service

Ahora se deben construir los siguientes métodos para crear la estructura del MIDlet, con el componente MapPanel (ver figuras 133 y 134).

```
public MapPanel getMapPanel() {
    if (mapPanel == null) {
        try {
            mapPanel = new MapPanel(230, 250);
            mapPanel.setMode(MapPanel.NOMODESELECTED);
            mapPanel.setURL("http://192.168.10.111:8080/service/");
            Project project = (Project) getMapPanel().getProjects().elementAt(0);
            mapPanel.setCurrentProject(project);
            mapPanel.setLayers(project.getLayers());
            mapPanel.setCache(2000);
            mapPanel.updateView();
        } catch (ConnectionNotFoundException ex) {
            alert = new Alert("Information", "Without connection", null, AlertType.INFO);
            switchDisplayable(alert, getForm());
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
    return mapPanel;
}
```

Figura 133. Código de acceso a una instancia de MapPanel

```
public Form getForm() {
    if (form == null) {
        form = new Form("Atlas", new Item[]{getMapPanel()});
        form.addCommand(mode);
        form.addCommand(exit);
        form.setCommandListener(this);
    }
    return form;
}
```

Figura 134. Código de acceso a una instancia de Formulario

Para lanzar la aplicación se debe modificar el método startMIDlet como se muestra en la figura 135.

```
    public void startMIDlet() {
        // write pre-action user code here

        // write post-action user code here
         switchDisplayable(null, getForm());
        getDisplay().setCurrentItem(getMapPanel());
    }
```

Figura 135. Código de lanzamiento del MIDlet

Para ejecutar la aplicación se debe desplegar el menú del proyecto MobileAtlas y seleccionar la opción “Run”, como se muestra en la figura 136.

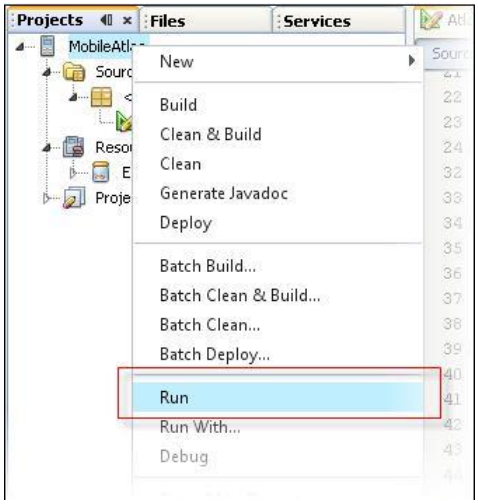

Figura 136. Submenu para ejecutar la aplicación

Si todo esta correcto, la aplicación debe verse como en la figura 137, y al ejecutarse no se debe presentar ninguna excepción. La pantalla inicial pregunta si la aplicación debe conectase al servidor, a lo que el usuario debe responder de forma afirmativa.

Figura 137. Apariencia de la aplicación MobileAtlas

4.2.3 Interactuar con un mapa.

Para permitir la interacción con el mapa es necesario el uso de modos en el ejemplo anterior, el mapa es estático, con el siguiente código (ver figura 138) el componente tiene la capacidad de acercarse, alejarse y desplazarse por el mapa.

```
public void commandAction(Command cmd, Displayable arg1) {
     if (cmd == exit) {
        exitMIDlet();
    }

    getForm().removeCommand(cmd);
    switch (getMapPanel().getMode()) {
        case MapPanel.NOMODESELECTED:
            mode = new Command("Zoom In", Command.SCREEN, 0);
            getMapPanel().setMode(MapPanel.MODEZOOMPLUS);
            break;
        case MapPanel.MODEZOOMPLUS:
            mode = new Command("Zoom Out", Command.SCREEN, 0);
            getMapPanel().setMode(MapPanel.MODEZOOMMINUS);
            break;
        case MapPanel.MODEZOOMMINUS:
            mode = new Command("Drag", Command.SCREEN, 0);
            getMapPanel().setMode(MapPanel.MODEPAN);
            break;
        case MapPanel.MODEPAN:
            mode = new Command("No mode", Command.SCREEN, 0);
            getMapPanel().setMode(MapPanel.NOMODESELECTED);
            break;
    }

    getForm().addCommand(mode);
    getDisplay().setCurrentItem(getMapPanel());

}
```

Figura 138. Código para administrar el manejo de modos

El botón de pantalla mode servirán para seleccionar el modo deseado, cada vez que el usuario presione el botón se cambiará a dicho modo. Los modos de operación disponibles para el MapPanel son:

MODEFEATUREINFO. Modo de operación en que el usuario puede obtener información sobre el contenido del mapa arrastrando el mouse para dibujar un círculo que representa el área de interés o simplemente haciendo clic en el mapa.

*MODEGETCOORD.*Modo de operación donde al arrastrar el mouse se logra el mismo efecto que en MODEPAN, pero produce eventos al hacer clic en el mapa y al hacer clic en un marcador.

MODEPAN. Modo de operación en que el usuario puede desplazarse por el mapa arrastrando el mouse.

MODEZOOMBOX. Modo de operación en que el usuario puede acercase al mapa arrastrando el mouse para dibujar una caja que encierre el área que desea visualizar.

MODEZOOMMINUS. Modo de operación en que el usuario puede alejarse al mapa haciendo clic en el.

MODEZOOMPLUS. Modo de operación en que el usuario puede acercarse al mapa haciendo clic en el.

Ahora, es cuestión de ejecutar la aplicación y probar el efecto de cada modo haciendo uso del puntero, las teclas de desplazamiento y la tecla central del dispositivo.

4.2.4 Procesar información GetFeatureInfo.

El servidor Atlas entrega información GetFeatureInfo en formato HTML, GML y texto plano, para este ejemplo se utiliza un nuevo formulario para recibir respuestas en texto plano. Para esto se agrega un escuchador al MapPanel como se muestra en la figura 139.

```
getMapPanel().getEventManager().addFeatureInfoListener(new FeatureInfoListener() {

    public void actionPerformed(FeatureInfoEvent ev) {

        Form formFI = new Form("Atlas Feature Info");
        Command back = new Command("Back", Command.BACK, 0);
        StringItem si = new StringItem("", ev.getFeatureInfo());
        formFI.addCommand(back);
        formFI.append(si);

        switchDisplayable(null, formFI);

        formFI.setCommandListener(new CommandListener() {

            public void commandAction(Command comm, Displayable disp) {
                if (comm.getCommandType() == Command.BACK) {
                    switchDisplayable(null, getForm());
                    getDisplay().setCurrentItem(getMapPanel());
                }
            }
        });

    }
});
```

Figura 139. Modificación al método startMIDlet para GetFeatureInfo

Para lanzar la petición GetFeatureInfo debe modificar el administrador de modos como se muestra en la figura 140.

```
public void commandAction(Command cmd, Displayable arg1) {
     if (cmd == exit) {
        exitMIDlet();
    }

    getForm().removeCommand(cmd);
    switch (getMapPanel().getMode()) {
        case MapPanel.NOMODESELECTED:
            mode = new Command("Zoom In", Command.SCREEN, 0);
            getMapPanel().setMode(MapPanel.MODEZOOMPLUS);
            break;
        case MapPanel.MODEZOOMPLUS:
            mode = new Command("Zoom Out", Command.SCREEN, 0);
            getMapPanel().setMode(MapPanel.MODEZOOMMINUS);
            break;
        case MapPanel.MODEZOOMMINUS:
            mode = new Command("Drag", Command.SCREEN, 0);
            getMapPanel().setMode(MapPanel.MODEPAN);
            break;
        case MapPanel.MODEPAN:
            mode = new Command("Info", Command.SCREEN, 0);
            getMapPanel().setMode(MapPanel.MODEFEATUREINFO);
            break;
         case MapPanel.MODEFEATUREINFO:
            mode = new Command("No mode", Command.SCREEN, 0);
            getMapPanel().setMode(MapPanel.NOMODESELECTED);
            break;
    }

    getForm().addCommand(mode);
    getDisplay().setCurrentItem(getMapPanel());

}
```

Figura 140. Modificación código administr ador de modos

El constructor también debe modificarse como se muestra en la figura 150.

```
public MapPanel getMapPanel() {
    if (mapPanel == null) {
        try {
            mapPanel = new MapPanel(230, 250);
            mapPanel.setMode(MapPanel.NOMODESELECTED);
            mapPanel.setURL("http://127.0.0.1:8080/service/");
            Project project = (Project) getMapPanel().getProjects().elementAt(0);
            mapPanel.setCurrentProject(project);
            mapPanel.setLayers(project.getLayers());

            mapPanel.setCurrentInfoFormat("text/plain");

            mapPanel.setCache(2000);
            mapPanel.updateView();
        } catch (ConnectionNotFoundException ex) {
            alert = new Alert("Information", "Without connection", null, AlertType.INFO);
            switchDisplayable(alert, getForm());
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
    return mapPanel;
}
```

Figura 141. Modificación al metodo getMapPanel para FeatureInfo

Con las modificaciones anteriores al establecer el modo de GetFeatureInfo y arrastrar el apuntador en el mapa debe obtenerse un resultado como el mostrado en la imagen 151.

Figura 142. Área de interés de una consulta GetFeatureInfo

Al levantar el apuntador se debe mostrar un nuevo formulario con el resultado de la consulta en formato texto plano como se muestra en la figura 143.

Figura 143. Respuesta getFeatureInfo en formato texto plano

4.2.5 Administrar marcadores.

Un marcador es un punto que se agrega al mapa para resaltar una ubicación, en este ejemplo se va a permitir al usuario agregar marcadores al mapa tocando la pantalla con el apuntador, para ello se usa el modo MODEGETCOORD del MapPanel y un escuchador al tocar la pantalla del dispositivo.

Para esto se agrega un escuchador al MapPanel como se muestra en la figura 144.

```
getMapPanel().getEventManager().addPointerTapListener(new PointerTapListener() {

    public void actionPerformed(PointerTapEvent ev) {
        try {

            byte[] bytesH = IO.readFromClassPath("/demo/home.png");
            Image homeImage = Image.createImage(bytesH, 0, bytesH.length);

            Marker marker = new Marker(ev.getPointMap(), homeImage, homeImage);
            getMapPanel().addMarker(marker);
            getMapPanel().setCurrentMarker(marker);
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
});
```

Figura 144. Modificación al método startMIDlet para detectar toques en la pantalla

Para cambiar al modo MODEGETCOORD es necesario modificar el administrador de modos como se muestra en la figura 145.

```
case MapPanel.MODEFEATUREINFO:
    mode = new Command("Marker", Command.SCREEN, 0);
    getMapPanel().setMode(MapPanel.MODEGETCOORD);
    break;
case MapPanel.MODEGETCOORD:
    mode = new Command("No mode", Command.SCREEN, 0);
    getMapPanel().setMode(MapPanel.NOMODESELECTED);
    break;
```

Figura 145. Código para agremas el modo MODEGETCOORD

Una vez aplicados estos cambios, al seleccionar el modo agregar marcador y hacer un toque sobre el mapa, se agrega un nuevo marcador.

4.2.6 Comunicación con el geocoder.

Para este ejercicio se enviará una petición al geocoder del proyecto seleccionado, en el punto indicado, se centrará el mapa y se agregará un marcador. Para esto hace falta que el servidor Atlas de prueba tenga un geocoder configurado para el proyecto.

Una vez más, se debe agregar un escuchador para permitir al componente obtener la respuesta de geocodificación del servidor el código que muestra la figura 146.

```
getMapPanel().getEventManager().addGeocodingListener(new GeocodingListener() {

    public void actionPerformed(GeocodingEvent ev) {
        try {
            Result[] res = ev.getAddresses();
            if (res.length > 0) {

                MapPanel map = ev.getMapSource();
                switchDisplayable(null, getForm());

                map.setZoomLevel(-15);
                map.setCenter(res[0].getLocation());
                map.updateView();

                map.removeMarker(geocoderMarker);
                geocoderMarker.setLabel(res[0].getAddress());
                geocoderMarker.setCoordinate(res[0].getLocation());
                map.addMarker(geocoderMarker);
            } else {
                alert = new Alert("Information", "No address found", null, AlertType.INFO);
                switchDisplayable(alert, getForm());
            }

            getDisplay().setCurrentItem(getMapPanel());

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
});
```

Figura 146. Modificaciones al método startMIDlet para manejo de geocoding

Para acceder a la opción de geocoder, se debe agregar un botón al formulario con el código que se muestra en la figura 147.

```
geocoder = new Command("Geocoding", Command.SCREEN, 1);
form.addCommand(geocoder);
```

Figura 147. Código para agregar botón del geocoder

Para enviar la petición se recomienda la creación de un nuevo formulario en el cual se agrega una caja de texto para escribir la dirección y un botón para enviar la petición al servidor. Para lograr esto en el administrador de modos se debe agregar el siguiente código (ver figura 148).

```
if (cmd == geocoder) {

    Form formGeo = new Form("Atlas Geocoder");
    Command locate = new Command("Locate", Command.OK, 0);
    Command back = new Command("Back", Command.BACK, 0);
    address.setString("");

    formGeo.append(address);
    formGeo.addCommand(back);
    formGeo.addCommand(locate);
    switchDisplayable(null, formGeo);

    formGeo.setCommandListener(new CommandListener() {

        public void commandAction(Command comm, Displayable disp) {
            if (comm.getCommandType() == Command.OK) {

                if (!address.getString().equals("")) {
                    try {
                        getMapPanel().sendGeocoderRequest(address.getString());
                    } catch (Exception ex) {
                        ex.printStackTrace();
                    }
                }
            }

            if (comm.getCommandType() == Command.BACK) {
                switchDisplayable(null, getForm());
                getDisplay().setCurrentItem(getMapPanel());
            }

        }
    });

    return;
}
```

Figura 148. Código para enviar la petición al servidor de geocodificación

Al aplicar estos cambios, al escribir una dirección en la caja de texto y hacer un toque sobre el botón “Locate”, la dirección geocodificada debe verse en el mapa representada por un marcador, como se muestra en la figura 149.

Figura 149. Apariencia de la aplicación respuesta geocoder